BTS · SIO

BTS SIO · MATHÉMATIQUES POUR L'INFORMATIQUE

Algorithmique appliquée

Lire, compléter, écrire et tester des algorithmes, en pseudo-code et en Python.

Version élève

1. Rappels — Ce qu'il faut savoir avant 2
2. Référentiel — Ce que l'examen demande 6
3. Cours — Comprendre, pas à pas 25
4. Exercices — S'entraîner 98
5. Annales — Les vrais sujets du BTS 176
6. Corrigés — Exercices et annales corrigés 183
7. Fiche — Tout le cours en résumé 414
8. QCM — Réviser vite, en boucle 420
9. Annexes — Les sujets officiels 449

RAPPELS

Ce qu'il faut savoir avant

Ce thème utilise quatre outils de calcul. Vous les avez sûrement vus au collège ou au lycée. Relisez-les, puis faites le petit exercice de chacun.

Faire tourner un programme Python

Dans ce thème, chaque algorithme s'écrit aussi en Python, pour le tester. Il y a deux façons de lancer un programme.

1. **Dans la version écran de ce document** : sous chaque code Python, cliquez sur « Exécuter ». Le résultat s'affiche dessous. « Modifier » vous laisse changer le code. Il faut une connexion internet la première fois.
2. **Sur votre ordinateur, avec Thonny** : c'est un logiciel gratuit, fait pour apprendre Python. Ouvrez Thonny et tapez le code dans la grande zone du haut. Enregistrez le fichier avec un nom qui finit par `.py`. Cliquez ensuite sur le bouton vert « Exécuter ». Le résultat s'affiche dans la zone du bas, appelée la **console**.

Le jour de l'examen, vous écrivez vos algorithmes sur papier, sans ordinateur. Python sert à vous entraîner : vous écrivez d'abord sur papier, puis vous vérifiez sur machine.

Un **commentaire** est une ligne que l'ordinateur ignore. Il sert à expliquer le code aux humains.

En Python, un commentaire commence par `#`. En pseudo-code, il commence par `//`.

> **Checkpoint**
>
> 1. Dans Thonny, où s'affiche le résultat d'un programme ?
> 2. Que fait l'ordinateur d'une ligne qui commence par `#` en Python ?

L'ordre des opérations

Dans un calcul sans parenthèses, on fait d'abord les puissances, puis les multiplications et les divisions, puis les additions et les soustractions. Les parenthèses passent avant tout.

Calcul	Ce qu'on fait d'abord	Résultat
$2 + 3 \times 4$	$3 \times 4 = 12$	14
$(2 + 3) \times 4$	$2 + 3 = 5$	20
2×3^2	$3^2 = 9$	18

Checkpoint

1. Combien vaut $10 - 2 \times 3$?
2. Combien vaut $(10 - 2) \times 3$?

La division euclidienne

Diviser 17 par 5 « avec un reste », c'est chercher combien de fois 5 tient dans 17, et ce qui reste. 5 tient 3 fois dans 17, car $3 \times 5 = 15$. Il reste $17 - 15 = 2$.

On écrit : $17 = 5 \times 3 + 2$. Le nombre 3 est le **quotient**. Le nombre 2 est le **reste**.

Le reste est toujours plus petit que le diviseur : ici, 2 est plus petit que 5.

Division	On écrit	Quotient	Reste
17 par 5	$17 = 5 \times 3 + 2$	3	2
20 par 4	$20 = 4 \times 5 + 0$	5	0
7 par 10	$7 = 10 \times 0 + 7$	0	7

Quand le reste vaut 0, la division « tombe juste » : 20 est **divisible** par 4. Un nombre est **pair** quand le reste de sa division par 2 vaut 0.

Checkpoint

1. Quel est le quotient et le reste de 29 divisé par 6 ?
2. 45 est-il divisible par 9 ?

Les puissances de 2

2^n se lit « 2 puissance n ». C'est 2 multiplié par lui-même n fois. Par convention, $2^0 = 1$.

n	0	1	2	3	4	5	6	7	8	10
2^n	1	2	4	8	16	32	64	128	256	1 024

Chaque puissance est le double de la précédente. On s'en sert pour savoir combien de valeurs tiennent sur un nombre de bits.

Checkpoint

1. Combien vaut 2^9 ?
2. Combien vaut $2^8 - 1$?

Écrire un petit entier en binaire

En binaire, on n'utilise que les chiffres 0 et 1. Chaque position vaut une puissance de 2, en partant de la droite : 1, 2, 4, 8, 16...

Pour lire un nombre binaire, on additionne les puissances de 2 là où il y a un 1 :

Position	3	2	1	0
Valeur	8	4	2	1
Chiffres de 1101	1	1	0	1

1101 en binaire vaut $8 + 4 + 1 = 13$.

Pour écrire 13 en binaire, on fait l'inverse. On prend la plus grande puissance de 2 qui tient dans 13 : c'est 8. Il reste 5.

Dans 5, on prend 4. Il reste 1. Dans 1, on prend 1.

On met un 1 aux positions 8, 4 et 1, un 0 à la position 2 : 1101.

Checkpoint

1. Combien vaut 101 en binaire ?
2. Écrivez 6 en binaire.

Les réponses des rappels

Cherchez d'abord seul. Vérifiez ensuite ici.

Faire tourner un programme Python

Checkpoint

1. Dans Thonny, où s'affiche le résultat d'un programme ? → **Dans la console, la zone du bas.**

2. Que fait l'ordinateur d'une ligne qui commence par `#` en Python ? → **Rien : c'est un commentaire, il l'ignore.**

L'ordre des opérations

Checkpoint

1. Combien vaut $10 - 2 \times 3$? → **4. On fait $2 \times 3 = 6$ d'abord.**

2. Combien vaut $(10 - 2) \times 3$? → **24.**

La division euclidienne

Checkpoint

1. Quel est le quotient et le reste de 29 divisé par 6 ? → **Quotient 4, reste 5, car $29 = 6 \times 4 + 5$.**

2. 45 est-il divisible par 9 ? → **Oui. $45 = 9 \times 5 + 0$, le reste vaut 0.**

Les puissances de 2

Checkpoint

1. Combien vaut 2^9 ? → **512, le double de 256.**

2. Combien vaut $2^8 - 1$? → **255.**

Écrire un petit entier en binaire

Checkpoint

1. Combien vaut 101 en binaire ? → **5, car $4 + 1 = 5$.**

2. Écrivez 6 en binaire. → 110**, car $6 = 4 + 2$.**

RÉFÉRENTIEL

Ce que l'examen demande

Cet onglet donne les **textes officiels, recopiés mot pour mot**. Chaque texte a son annexe : l'image exacte de la page, et le PDF à voir ou à télécharger.

Trois textes définissent ce que vous devez savoir :

1. Le **référentiel du BTS SIO** : il dit quels modules de maths sont au programme.
2. La **définition de l'épreuve E3** : elle dit comment vous êtes évalué.
3. Le **programme du module « Algorithmique appliquée »** : il dit, capacité par capacité, ce que vous devez savoir faire.

Votre épreuve : la forme ponctuelle

Votre établissement est un établissement privé : vous ne passez pas le contrôle en cours de formation. Vous passez la **forme ponctuelle** de l'épreuve E3, partie B du texte officiel : **un écrit de 2 heures**, avec deux ou trois exercices. L'un d'eux « évalue la capacité « programmer » » : c'est exactement le format des annales. Il n'y a pas d'oral de maths.

La réforme de 2024

L'arrêté du 8 juillet 2024 modifie le référentiel du BTS SIO, à partir de la session 2025. Les épreuves sont renumérotées : les maths, qui étaient l'épreuve E2, deviennent l'**épreuve E3**. Le contenu des maths ne change pas : mêmes modules, mêmes façons d'évaluer. Dans le corps du référentiel, l'unité garde son ancien nom, « U2 ».

La dernière partie de cet onglet relie chaque capacité du programme à la notion du cours où vous la travaillez.

1. Le référentiel du BTS SIO : les modules de maths

Source : référentiel du BTS « Services informatiques aux organisations », arrêté du 29 avril 2019 modifié par l'arrêté du 8 juillet 2024, annexe « U2 – Mathématiques pour l'informatique ».

Annexe A1 Le référentiel officiel, page 83 du PDF, à la fin de ce document.

U2 – Mathématiques pour l'informatique

Le programme est conçu avec l'intention de permettre l'acquisition des bases mathématiques nécessaires à la compréhension et la maîtrise des finalités spécifiques du BTS Services informatiques aux organisations et de démarches mathématiques et algorithmiques permettant d'en appréhender la pertinence et l'efficacité pour évoluer dans un environnement numérique. On distingue trois objectifs principaux :

- comprendre et résoudre les problèmes mathématiques élémentaires auxquels une personne informaticienne est couramment confrontée (calcul binaire, masque de réseau, opérateurs logiques...) ;
- comprendre et manipuler les objets mathématiques fréquemment utilisés en programmation, de manière à pouvoir exploiter informatiquement une solution mathématique préalablement construite ;
- résoudre des problèmes numériques nécessitant la mise en œuvre d'algorithmes qu'il s'agit de construire, de mettre en forme et dont on comparera éventuellement les performances.

D'une manière générale, la recherche et la mise en œuvre d'algorithmes en utilisant les moyens informatiques propres à la section sont au centre de cette formation.

De par la spécificité des compétences visées, mais aussi afin d'atténuer les difficultés inhérentes au passage à l'abstraction mathématique, les nouveaux concepts gagnent à être introduits après une observation de phénomènes rendue possible par la pratique de la programmation. Toute démarche mise en œuvre à l'aide d'un programme informatique ou d'une calculatrice permettant de faciliter la compréhension d'un concept ou d'une méthode en l'illustrant graphiquement, numériquement ou dans un contexte lié à la spécialité doit être privilégiée. L'utilisation large des moyens informatiques (calculatrice et ordinateur), repose sur la connaissance d'éléments de syntaxe et de fonctions spécifiques à l'outil utilisé mais sans exigence de virtuosité.

Pour la programmation, un langage de haut niveau doit être effectivement et régulièrement utilisé par le professeur comme par les élèves. Le langage Python, langage de programmation simple d'utilisation, libre d'installation et largement enseigné est recommandé sans être imposé. Dans le cas où un autre langage est choisi, il convient de s'assurer qu'il est accepté par le centre d'examen. Dans certaines situations où leur utilisation semble plus pertinente, on peut notamment utiliser, un tableur, une calculatrice, un logiciel de calcul formel.

Le programme est constitué des modules suivants décrits par le programme de mathématiques des brevets de technicien supérieur (arrêté du 4 juin 2013) :

- arithmétique ;
- calcul matriciel ;
- algèbres de Boole ;
- savoirs de la théorie des ensembles ;
- graphes et ordonnancement ;

- algorithmique appliquée, à l'exception de l'item « récursivité » et de l'item « analyse d'algorithme ».

2. L'épreuve E3 — Mathématiques pour l'informatique

Source : arrêté du 8 juillet 2024, annexe IV, « Épreuve E3 – Mathématiques pour l'informatique », Journal officiel du 10 juillet 2024.

Annexe A2 La définition officielle de l'épreuve, pages 11 à 12 du PDF, à la fin de ce document.

EPREUVE E3 — Mathématiques pour l'informatique — Coefficient 3

A. – Contrôle en cours de formation

L'épreuve comporte deux situations d'évaluation, une pratique comptant pour un coefficient 1 et une écrite comptant pour un coefficient 2.

A.1. – Première situation d'évaluation constituée d'une épreuve pratique orale ; horaire indicatif : de 20 minutes (+ 1 heure de préparation) – Coefficient 1

A partir d'une situation problème, de consignes écrites et la fourniture de composants logiciels utilisables pour la résolution demandée, le candidat doit fournir une production manuscrite comportant un ou plusieurs algorithmes qu'il met en œuvre sur machine puis commente oralement.

1. Finalités et objectifs

L'objectif est d'évaluer la capacité de la personne candidate à analyser un énoncé, formaliser une démarche de résolution de problème, écrire, interpréter et éventuellement modifier ou compléter un ou plusieurs algorithmes relatifs au module d'algorithmique appliquée.

2. Contenu

A partir d'une situation problème, de consignes écrites et éventuellement la fourniture de composants logiciels utilisables pour la résolution demandée, le candidat doit fournir une production manuscrite comportant un ou plusieurs algorithmes qu'il met en œuvre sur machine puis commente oralement.

Aucun langage ni formalisme particulier n'est imposé pour l'écriture des algorithmes.

Il convient d'éviter toute difficulté théorique et toute technicité excessive.

Les compétences liées aux autres modules de cette unité ne sont pas évaluées dans cette épreuve. Si la réalisation demandée s'appuie sur les techniques abordées dans un autre module, les méthodes requises sont fournies dans le sujet.

3. Critères d'évaluation

Les compétences attendues sont évaluées sur la base des critères suivants :

- maîtrise des connaissances liées au module d'algorithmique appliquée ;
- efficacité et pertinence de la solution proposée ;
- correction et cohérence de l'utilisation du formalisme retenu ;
- qualité de la mise œuvre, notamment la lisibilité (indentation, commentaires, etc.) ;
- efficacité de l'implémentation ;
- pertinence de l'utilisation des composants logiciels disponibles ;
- adéquation des tests de validation effectués ;
- aptitude à proposer des éléments de correction pertinents.

4. Modalités d'évaluation : contrôle en cours de formation

Cette évaluation est constituée d'un oral de 20 minutes précédé d'une heure de préparation. Le candidat présente sa solution algorithmique et son implémentation (durée 10 minutes maximum), puis participe à un entretien d'explicitation conduit par la commission (durée 10 minutes maximum).

La préparation se déroule en deux parties :

- une première partie de 30 mn, sur table, qui fait l'objet d'une trace écrite susceptible d'être examinée par la commission ;
- une seconde partie de 30 mn sur un équipement dédié mis à disposition par le centre d'examen. Durant cette phase, le candidat peut librement accéder à l'aide syntaxique éventuellement disponible dans l'environnement de mise en œuvre du langage utilisé.

La commission d'interrogation est constituée du professeur chargé de l'enseignement du module d'algorithmique appliquée en STS services informatiques aux organisations.

A l'issue des évaluations, l'équipe pédagogique de l'établissement de formation adresse au jury, pour chaque candidat, la proposition de note sur 20 points accompagnée de la grille d'aide à l'évaluation renseignée (dont le modèle est fourni par la circulaire d'organisation de l'examen). Conformément à la réglementation, le jury peut demander à avoir communication des dossiers d'évaluation des candidats. Ces documents sont tenus à la disposition du jury et de l'autorité rectorale pour la session considérée jusqu'à la session suivante.

A.2. – Deuxième situation d'évaluation constituée d'une évaluation écrite ; horaire indicatif : 55 minutes – Coefficient 2

1. Finalités et objectifs

Cette épreuve vise à évaluer les compétences acquises par la personne candidate dans le cadre de l'unité de Mathématiques pour l'informatique. Elle a pour objectifs d'apprécier les connaissances et leur mobilisation dans des situations variées, de vérifier l'aptitude au raisonnement et les capacités à analyser un problème, à justifier l'emploi des méthodes utilisées et à interpréter les résultats et d'apprécier la rigueur et la précision de son expression écrite et orale.

2. Contenu

L'épreuve s'appuie sur un sujet comportant un ou deux exercices couvrant une partie significative des modules autres que l'algorithmique appliquée de l'unité de mathématiques pour l'informatique. Chaque exercice présente une situation concrète, en relation avec les activités professionnelles visées par le diplôme.

3. Critères d'évaluation

Les compétences attendues sont évaluées sur la base des critères suivants :

- maîtrise des connaissances figurant au programme ;
- pertinence des sources d'information mobilisées ;
- adaptation de la stratégie choisie au problème à résoudre ;
- efficacité dans la mise en œuvre de cette stratégie ;
- rigueur et pertinence dans l'utilisation des savoir-faire figurant au programme de mathématiques ;
- cohérence de l'argumentation employée ;
- aptitude à analyser un résultat avec pertinence ;
- qualité d'expression écrite ou orale.

4. Modalités d'évaluation : contrôle en cours de formation

Le contrôle en cours de formation comporte une situation d'évaluation qui se déroule au cours du deuxième semestre de la deuxième année. L'épreuve dure cinquante-cinq minutes.

Elle est constituée d'un ou deux exercices avec des questions de difficulté progressive.

L'emploi de la calculatrice est autorisé selon la législation en vigueur. En fonction des besoins et afin de ne pas introduire de discriminations liées aux performances de la calculatrice employée, certaines formules de base peuvent être rappelées en tête du sujet (relations fonctionnelles, suites arithmétiques et géométriques, etc.).

La correction est assurée par le professeur de mathématiques enseignant en STS services informatiques aux organisations.

A l'issue des évaluations, l'équipe pédagogique de l'établissement de formation adresse au jury, pour chaque candidat, la proposition de note sur 20 points accompagnée de la grille d'aide à l'évaluation renseignée (dont le modèle est fourni par la circulaire d'organisation de l'examen). Conformément à la réglementation, le jury peut demander à avoir communication des dossiers d'évaluation des candidats. Ces documents sont tenus à la disposition du jury et de l'autorité rectorale pour la session considérée jusqu'à la session suivante.

B. – Forme ponctuelle : épreuve écrite de 2 heures

Pour les candidats non éligibles au CCF, l'épreuve ponctuelle prend la forme d'une épreuve écrite de 2 heures sur un sujet comportant deux ou trois exercices couvrant une partie significative du programme de l'unité, pour un total de 20 points. L'un des exercices évalue la capacité « programmer » en lien avec le module d'algorithmique appliquée.

3. Le programme du module « Algorithmique appliquée »

Source : programme de mathématiques des brevets de technicien supérieur, arrêté du 4 juin 2013, module « Algorithmique appliquée », pages 55 à 59.

Annexe A3 Le programme officiel, pages 60 à 64 du PDF, à la fin de ce document.

ALGORITHMIQUE APPLIQUÉE

L'objectif de ce module est de construire des algorithmes, et de les programmer dans un langage informatique, dans le but de résoudre des problèmes de niveau raisonnable.

Les thématiques abordées lors de l'étude de ce module sont très ouvertes, mais l'objectif visé, à l'intérieur du processus de conception, est ciblé. On veille à ce que les situations proposées soient mathématiquement achevées. Alors qu'une solution, peut être décrite de manière très libre, textuelle ou graphique, par formules ou symboles, par l'exemple ou de manière inachevée, on s'attache ici à les exprimer en utilisant les outils algorithmiques usuels, pour les rendre directement convertibles et exécutables sur machine.

Afin de faciliter la compréhension des mécanismes et la détection d'éventuelles erreurs, il est impératif de concrétiser les algorithmes par l'emploi d'un langage de programmation et de conduire l'étudiant à réaliser des tests. La tâche inverse, consistant à comprendre un algorithme, présente également un grand intérêt pour l'assimilation des mécanismes et lors d'opérations de maintenance.

Les sujets empruntés à la vie courante peuvent être utilisés à chaque fois qu'ils permettent d'illustrer un mécanisme simple avec pertinence. Sinon, on préfère utiliser des sujets dérivés directement d'autres modules mathématiques et, avec un certain équilibre, des sujets associés à des thématiques informatiques (par exemple : codage, cryptage et décryptage, redondance de

sécurité, tri itératif et tri récursif, parcours de graphes). Ces sujets peuvent être abordés afin d'illustrer des concepts fondamentaux d'algorithmique sans qu'aucune connaissance spécifique ne soit exigible de l'étudiant dans ces derniers domaines.

Ce module vise à développer les compétences spécifiques suivantes :

- comprendre un algorithme et expliquer ce qu'il fait ;
- modifier un algorithme existant pour obtenir un résultat différent ;
- concevoir une procédure, un algorithme simple ;
- transcrire un algorithme dans un langage informatique ;
- s'interroger sur l'efficacité d'un algorithme.

Les contrôles d'exécution constituent le cœur des mécanismes algorithmiques de base. À ce titre, on attache un soin tout particulier à leur étude progressive mais détaillée. Leur maîtrise pratique est essentielle et les évaluations doivent être centrées sur eux.

Pour l'écriture des algorithmes, une représentation textuelle convenablement indentée avec des indicateurs de début et de fin explicites facilite la lecture. Pour aider à la compréhension, il est utile également d'indiquer un en-tête composé d'un nom, d'un rôle, de l'indication des données d'entrée et de sortie. Des commentaires sont ajoutés, notamment pour préciser le rôle des variables ou fournir des indications méthodologiques.

Un algorithme est indépendant de tout langage de programmation ; aussi aucun langage n'est imposé, mais il convient de s'assurer qu'il est accepté par le centre d'examen. On privilégie un langage de programmation simple d'utilisation et libre d'installation. L'existence d'une communauté d'utilisateurs et de bibliothèques facilite le développement.

La présentation linéaire du programme, avec une entrée par les contenus n'indique pas d'ordre dans sa mise en œuvre. Aussi les concepts fondamentaux (algorithme, finitude, modularité, identifiant, type, constante, variable, fonction, procédure, expression numérique, expression conditionnelle et plus généralement booléenne…) sont acquis par l'usage, sans faire appel à des définitions formelles préalables.

CONTENUS	CAPACITÉS ATTENDUES	COMMENTAIRES
Types de données		
Types simples : entier naturel, entier relatif, réel, booléen.	• Connaître les modes de codage et gérer les différences entre données mathématiques et informatiques : – domaine de valeurs ; – représentation exacte ou approchée, précision.	Il n'est pas nécessaire de connaître la représentation exacte des données en machine, notamment en ce qui concerne les flottants.
Chaîne de caractères.		On évite de considérer une chaîne comme un tableau de caractères.
Tableaux de données : – de type homogène à une ou deux dimensions ; – à deux dimensions dans lequel, soit les lignes soit les colonnes, peuvent être de types différents.	• Construire un tableau. • Traiter les données d'un tableau : – accéder à ses différents éléments en lecture et en écriture ; – traiter les éléments d'une ligne ou d'une colonne ; – copier un tableau.	On adapte la construction et l'exploitation de ces tableaux aux possibilités de l'outil informatique utilisé. Les structures de données et les objets ne sont pas au programme de mathématiques : ils figurent dans ceux des enseignements professionnels.
Procédure et fonction : – paramètres d'entrée ; – valeur(s) retournée(s) par une fonction ; – variables globales ou locales.	• Gérer les transferts en entrée seule, en sortie seule, en entrée et sortie. • Utiliser les variables globales et locales.	Sans aborder la programmation objet, les concepts de modularité et d'interface doivent être connus.
Instructions élémentaires Lecture, écriture. Affectation, affectation récursive.	• Saisir une donnée depuis le clavier, manipuler les variables et afficher sur l'écran. • Gérer la chronologie des valeurs contenues dans les variables et produire la trace d'un algorithme.	Les fichiers ne sont pas au programme de mathématiques. En standard, la mise en forme des affichages est limitée à l'utilisation de séparateurs et de changements de ligne. Sinon, les outils nécessaires sont fournis et décrits. La gestion des pointeurs n'est pas au programme.

CONTENUS	CAPACITÉS ATTENDUES	COMMENTAIRES
Opérateurs Opérateurs numériques : addition, soustraction, multiplication, division, exponentiation, quotient et reste de la division entière, signe. Fonctions mathématiques usuelles.	• Transformer une expression mathématique en expression numérique en ligne et réciproquement.	« Fonctions mathématiques usuelles » doit être entendu au sens informatique et inclure les fonctions d'arrondi, ainsi qu'un générateur de nombres pseudo-aléatoires uniforme dans un intervalle.
Opérateurs de comparaison : =, <> ou !=, <, <=, >, >=.		On compare soit des valeurs numériques, soit des chaînes de caractères.
Opérateurs booléens : non, et, ou, oux.	• Traduire la condition (éventuellement composée) d'une itération (tant que / répéter jusqu'à ce que) ou d'une alternative (si) sous forme d'expression booléenne.	
Opérateurs booléens bit à bit.	• Appliquer des opérations booléennes sur les bits.	On interprète notamment en termes de masque, de mise à un, de mise à zéro, de changement d'état.
Opérateur de chaînes : concaténation. Fonctions permettant l'extraction en début, milieu ou fin, la recherche d'un motif.	• Construire et manipuler des chaînes et construire les messages affichés à l'écran.	L'usage d'expressions régulières simples est possible, mais l'étude des expressions régulières est hors programme.
Transtypage	• Gérer les différents types de données et effectuer des conversions d'un type vers un autre.	D'autres instructions, fonctions ou procédures peuvent être introduites dans l'écriture d'algorithmes. Les descriptions sémantiques et syntaxiques précises sont alors mises à disposition de l'étudiant.
Structures de contrôle et d'exécution. Exécution séquentielle. Exécution à structure	• Mettre en œuvre ces structures.	Afin d'en maîtriser le fonctionnement, les structures d'exécution sont elles-mêmes

CONTENUS	CAPACITÉS ATTENDUES	COMMENTAIRES
conditionnelle (si-alors-sinon). Exécution à structure itérative (pour) et (tant que / répéter jusqu'à ce que).		présentées sous forme d'algorithmes.
Construction des structures itératives : raisonnement par récurrence, initialisation, mise à jour itérative, calcul itératif, mise en forme finale.	• Gérer une itération en distinguant la préparation, l'itération elle-même et la mise en forme finale.	Le raisonnement par récurrence n'a pas à être évalué pour des démonstrations. Il est introduit pour servir de base à une construction des itérations. Le calcul itératif est souvent récursif
Symboles $\sum_{i=}$ et $\prod_{i=}$, traduction algorithmique.	• Gérer une somme ou un produit d'un nombre variable d'opérandes dépendant d'un paramètre.	Généralement la variable affectée récursivement est initialisée à l'élément neutre de l'opération utilisée, avant d'entrer dans l'itération.
Structures imbriquées	• Gérer des structures imbriquées.	On traite également des exemples où les éléments de contrôle des structures internes dépendent de ceux des structures externes. Le nombre d'imbrications n'est pas limité, sauf pour les itérations en dépendance, où on se limite à deux. On évite les excès de complexité, ainsi que les constructions ne correspondant pas à un besoin concret.
	• Traiter un produit matriciel.	La formule de calcul des coefficients d'un produit est donnée.

Hors programme du BTS SIO

Les lignes suivantes font partie du programme national de 2013. Le référentiel du BTS SIO les retire : « algorithmique appliquée, à l'exception de l'item « récursivité » et de l'item « analyse d'algorithme » ». Elles sont recopiées ici pour que le texte officiel soit complet. Elles ne sont ni enseignées ni évaluées.

CONTENUS	CAPACITÉS ATTENDUES	COMMENTAIRES
Récursivité. Nécessité d'un test. Nécessité de cas particuliers résolus sans appel à la récursivité. Finitude.	• Mettre en œuvre ou exploiter une fonction ou une procédure récursive simple.	On peut traiter des exemples de récursivité terminale, de conversion en algorithme non récursif, de récursivité non terminale. On ne présente pas de récursivité mutuelle entre plusieurs procédures.
Analyse d'algorithmes		Ce paragraphe se veut être une simple sensibilisation aux notions de complexité, de correction, de recherche d'erreur et de finalité d'un algorithme. En cas d'évaluation portant sur l'un de ces thèmes, on apporte suffisamment d'indications pour limiter les prérequis.
Notions de complexité temporelle et spatiale.	• Calculer un nombre minimal ou maximal d'opérations significatives ou la taille globale de données choisies.	On présente des variantes produisant les mêmes résultats avec des complexités très différentes. Aucune connaissance théorique n'est exigible.
Validation et débogage.	• Procéder à un suivi de variables par la production d'une trace ou l'utilisation de jeux d'essai pour le test d'un algorithme et la recherche d'erreur.	On propose des algorithmes délibérément erronés dont les défauts sont repérés puis débogués. Selon le langage choisi, on peut montrer les fonctions permettant le suivi de variables, le débogage pas à pas. Afin de mieux sensibiliser à certains risques, on peut présenter des cas d'effets indésirables (effets de bord, évaluation partielle lors de calcul d'expressions booléennes, débordements ou

CONTENUS	CAPACITÉS ATTENDUES	COMMENTAIRES
		approximations numériques, transtypage, utilisation d'indices hors domaine,...) et leurs conséquences spectaculaires. Aucune théorie n'est au programme.
Interprétation d'algorithmes.	• Savoir reconstituer le rôle d'un algorithme.	L'ajout d'une ou plusieurs procédures ou fonctions à un ensemble interdépendant peut constituer une excellente base. On se limite à des cas simples.

4. Où chaque capacité est travaillée

Les deux tableaux suivants sont écrits par nous. Le premier reprend chaque capacité attendue, mot pour mot. Le second reprend les commentaires du programme qui disent comment enseigner. La colonne « Notion » renvoie au cours : chaque notion se termine par ses 9 exercices, sur trois niveaux. Cochez dans votre tête : « je sais le faire seul ».

Les capacités attendues

Capacité attendue (texte officiel)	Notion	Annales
Connaître les modes de codage et gérer les différences entre données mathématiques et informatiques : domaine de valeurs ; représentation exacte ou approchée, précision.	2.1, 2.2	
Construire un tableau.	5.1, 5.2	
Traiter les données d'un tableau : accéder à ses différents éléments en lecture et en écriture ; traiter les éléments d'une ligne ou d'une colonne ; copier un tableau.	5.1, 5.2	2023
Gérer les transferts en entrée seule, en sortie seule, en entrée et sortie.	5.3, 5.4	2022, 2025 Q2, 2025 Q3, 2026 Q3 (entrée seule)
Utiliser les variables globales et locales.	5.4	
Saisir une donnée depuis le clavier, manipuler les variables et afficher sur l'écran.	1.2, 1.3	
Gérer la chronologie des valeurs contenues dans les variables et produire la trace d'un algorithme.	1.4	2022, 2025 Q3, 2026 Q3
Transformer une expression mathématique en expression numérique en ligne et réciproquement.	3.1, 3.2	2026 Q3
Traduire la condition (éventuellement composée) d'une itération (tant que / répéter jusqu'à ce que) ou	3.3, 4.1, 4.3	2023, 2025 Q2, 2025 Q3, 2026 Q2, 2026 Q3

Capacité attendue (texte officiel)	Notion	Annales
d'une alternative (si) sous forme d'expression booléenne.		
Appliquer des opérations booléennes sur les bits.	3.4	
Construire et manipuler des chaînes et construire les messages affichés à l'écran.	2.3, 3.5	2024, 2025 Q3
Gérer les différents types de données et effectuer des conversions d'un type vers un autre.	2.4	
Mettre en œuvre ces structures.	1.1, 4.1, 4.2, 4.3	2022, 2023, 2024, 2025 Q2, 2025 Q3, 2026 Q2, 2026 Q3
Gérer une itération en distinguant la préparation, l'itération elle-même et la mise en forme finale.	4.4	2022, 2023, 2024, 2026 Q2
Gérer une somme ou un produit d'un nombre variable d'opérandes dépendant d'un paramètre.	4.4	2022
Gérer des structures imbriquées.	4.5	2022, 2023, 2024, 2026 Q2
Traiter un produit matriciel.	4.5	

Les commentaires du programme et les compétences du module

Texte officiel	Notion
Comprendre un algorithme et expliquer ce qu'il fait ; modifier un algorithme existant pour obtenir un résultat différent ; concevoir une procédure, un algorithme simple ; transcrire un algorithme dans un langage informatique.	1.4, 4.3, 5.3
S'interroger sur l'efficacité d'un algorithme.	4.5, 5.1
Une représentation textuelle convenablement indentée avec des indicateurs de début et de fin explicites ; un en-tête composé d'un nom, d'un rôle, de l'indication des données d'entrée et de sortie ; des commentaires.	1.1
Conduire l'étudiant à réaliser des tests.	4.1
On évite de considérer une chaîne comme un tableau de caractères.	2.3
En standard, la mise en forme des affichages est limitée à l'utilisation de séparateurs et de changements de ligne.	1.3
« Fonctions mathématiques usuelles » doit être entendu au sens informatique et inclure les fonctions d'arrondi, ainsi qu'un générateur de nombres pseudo-aléatoires uniforme dans un intervalle.	3.2
On compare soit des valeurs numériques, soit des chaînes de caractères.	3.3
On interprète notamment en termes de masque, de mise à un, de mise à zéro, de changement d'état.	3.4
Afin d'en maîtriser le fonctionnement, les structures d'exécution sont elles-mêmes présentées sous forme d'algorithmes.	4.3
Le raisonnement par récurrence [...] est introduit pour servir de base à une construction des itérations. Le calcul itératif est souvent récursif	4.4
	4.4

Texte officiel	Notion
Généralement la variable affectée récursivement est initialisée à l'élément neutre de l'opération utilisée, avant d'entrer dans l'itération.	
Le nombre d'imbrications n'est pas limité, sauf pour les itérations en dépendance, où on se limite à deux.	4.5
La formule de calcul des coefficients d'un produit est donnée.	4.5
Sans aborder la programmation objet, les concepts de modularité et d'interface doivent être connus.	5.3, 5.4
Tableaux de données à deux dimensions dans lequel, soit les lignes soit les colonnes, peuvent être de types différents.	5.2
Des sujets associés à des thématiques informatiques (par exemple : codage, cryptage et décryptage, redondance de sécurité, tri itératif et tri récursif, parcours de graphes).	3.1, 4.2, 5.1, 5.2

Ce qui n'est pas au programme n'est pas enseigné : les fichiers, les pointeurs, les structures de données et les objets, l'étude des expressions régulières, la récursivité et l'analyse d'algorithmes.

Pour vous entraîner à l'épreuve, faites les annales, puis les deux sujets types à la fin de l'onglet Exercices, en conditions réelles : sur papier, sans ordinateur.

COURS

Comprendre, pas à pas

Le plan du cours

1. Chapitre 1 — Écrire un algorithme
    - 1.1 Du problème au programme
    - 1.2 Les variables et l'affectation
    - 1.3 Lire et écrire
    - 1.4 La trace d'un algorithme
    - Ce qu'on retient du chapitre 1
2. Chapitre 2 — Les types de données
    - 2.1 Les types simples
    - 2.2 Les limites de la machine
    - 2.3 Les chaînes de caractères
    - 2.4 Changer de type : le transtypage
    - Ce qu'on retient du chapitre 2
3. Chapitre 3 — Les opérateurs
    - 3.1 Les opérateurs numériques
    - 3.2 Les fonctions mathématiques usuelles
    - 3.3 Comparer et combiner des conditions
    - 3.4 Les opérateurs bit à bit
    - 3.5 Les opérateurs de chaînes
    - Ce qu'on retient du chapitre 3
4. Chapitre 4 — Les structures de contrôle
    - 4.1 Faire un choix : Si... Alors... Sinon
    - 4.2 Répéter un nombre connu de fois : la boucle Pour
    - 4.3 Répéter tant que : les boucles Tant que et Répéter
    - 4.4 Construire une boucle de calcul
    - 4.5 Les structures imbriquées
    - Ce qu'on retient du chapitre 4
5. Chapitre 5 — Les tableaux et les fonctions
    - 5.1 Les tableaux à une dimension
    - 5.2 Les tableaux à deux dimensions
    - 5.3 Les fonctions et les procédures
    - 5.4 Paramètres et variables locales
    - Ce qu'on retient du chapitre 5

Un algorithme, c'est une suite d'instructions précises qui résout un problème. Dans ce cours, vous apprenez à les lire, à les compléter, à les écrire, puis à les faire tourner en Python.

Chapitre 1 — Écrire un algorithme

Inès travaille au service informatique d'un lycée. Chaque matin, elle compte les postes en panne et calcule le pourcentage de postes qui marchent. Elle le fait à la main, sur une feuille.

Elle se trompe parfois. Son responsable lui demande d'écrire un petit programme qui fait le calcul à sa place.

À la fin de ce chapitre, vous saurez écrire ce programme en pseudo-code et en Python, et vérifier qu'il est juste.

> **La question de départ**
>
> Une recette de cuisine dit : « Ajoutez le sel. Faites cuire 10 minutes. Goûtez. » Qu'est-ce qui se passe si on lit ces lignes dans le désordre ?

1.1 Du problème au programme

Une recette marche parce que chaque étape est claire et vient au bon moment. Comment écrire des étapes aussi claires pour un ordinateur ?

Un **ALGORITHME** est une suite d'instructions qui résout un problème, en un nombre fini d'étapes. L'ordinateur les exécute **dans l'ordre**, de haut en bas : c'est l'**EXÉCUTION SÉQUENTIELLE**.

Un ordinateur ne devine rien. Il fait exactement ce qu'on écrit, ni plus, ni moins. Si une étape manque, il ne la rajoute pas.

On écrit un algorithme dans deux langages :

1. Le **PSEUDO-CODE** : des mots français, simples et précis. C'est la forme des sujets du BTS. Il ne tourne sur aucune machine.
2. Le **Python** : un vrai langage de programmation. Il tourne sur l'ordinateur : on peut tester.

Voici le calcul d'Inès. Le lycée a 120 postes, 9 sont en panne.

PSEUDO-CODE

```
total ← 120
pannes ← 9
marche ← total - pannes
pourcentage ← marche / total * 100
Écrire "Postes qui marchent :",
pourcentage, "%"
```

PYTHON

```
total = 120
pannes = 9
marche = total - pannes
pourcentage = marche / total * 100
print("Postes qui marchent :",
pourcentage, "%")
```

CE QUI S'AFFICHE

```
Postes qui marchent : 92.5 %
```

Les deux colonnes disent la même chose, ligne pour ligne. Dans la version écran de ce cours, le bouton « Exécuter » lance le Python.

Un algorithme complet commence par un **en-tête** : son nom, son rôle, ce qu'il reçoit, ce qu'il produit. On l'écrit en commentaire, après `//` en pseudo-code et après `#` en Python.

PSEUDO-CODE

```
// Nom : taux_de_marche
// Rôle : calculer le pourcentage de postes qui marchent
// Entrées : total et pannes, deux entiers
// Sortie : le pourcentage affiché à l'écran
```

Les règles d'écriture du pseudo-code

- Un bloc à l'intérieur d'une structure est décalé de 4 espaces.
- Chaque structure se ferme par son « Fin de » : `Fin de Si`, `Fin de Pour`, `Fin de Tant que`, `Fin de Fonction`.
- L'algorithme commence par son en-tête : nom, rôle, entrées, sortie.
- Un commentaire `//` dit le rôle d'une variable qui n'est pas évident : `n ← 0 // nombre de postes en panne`.

On le fait ensemble — le stock de câbles

1. Le magasin a 40 câbles. On en vend 15. Quelle instruction donne le stock restant, en pseudo-code ?

2. Quelle instruction affiche le stock ?

3. Qu'affiche le programme ?

4. Que se passe-t-il si on écrit l'affichage avant le calcul ?

	Exemple	Pourquoi
✔	Calculer `marche`, puis `pourcentage`, puis afficher	Chaque ligne utilise ce qui a été calculé au-dessus.
✘	Afficher `pourcentage` à la première ligne	La variable n'existe pas encore : l'ordinateur s'arrête sur une erreur.
✘	Écrire « calculer le pourcentage » sans dire comment	Ce n'est pas une instruction : l'ordinateur ne devine pas la formule.

Checkpoint

1. Dans quel ordre l'ordinateur exécute-t-il les instructions ?
2. Quelle est la différence entre le pseudo-code et Python ?

À vous ! Les exercices de la notion 1.1

★☆☆ **Appliquer**
1.1.1 à 1.1.3
page 98

★★☆ **S'entraîner**
1.1.4 à 1.1.6
page 99

★★★ **Aller plus loin**
1.1.7 à 1.1.9
page 100

Commencez par le niveau 1. Passez au niveau suivant quand les trois exercices sont réussis.

1.2 Les variables et l'affectation

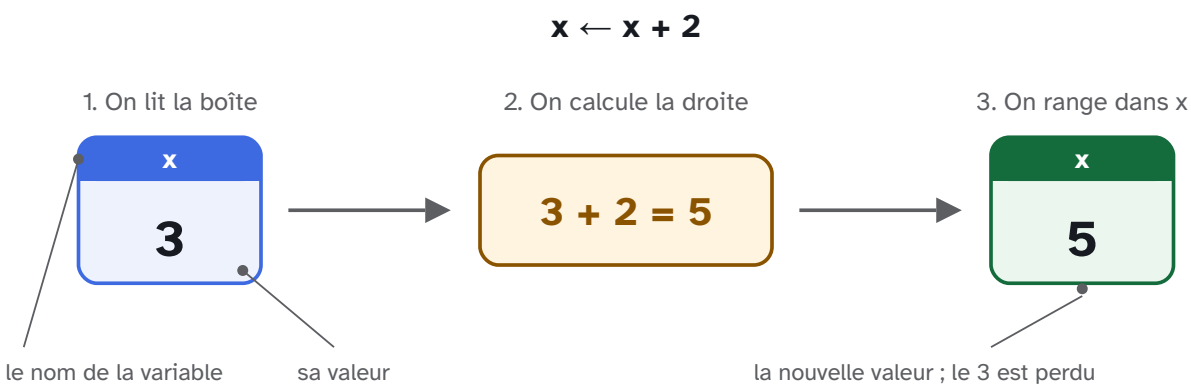
Inès a rangé des nombres dans `total`, `pannes`, `marche`. Mais que sont ces noms, et que fait exactement la flèche `←` ?

Une **VARIABLE** est une case de la mémoire, avec un nom, qui contient une valeur.
L'**AFFECTATION** range une valeur dans une variable.

En pseudo-code, on l'écrit avec la flèche `←`. En Python, avec le signe `=`.

Pensez à une boîte avec une étiquette. L'étiquette, c'est le nom.

Le contenu, c'est la valeur. On peut vider la boîte et mettre autre chose dedans : l'ancienne valeur disparaît.



x ← x + 2 : on lit l'ancienne valeur, on calcule, on range le résultat dans la même boîte

Comment ça marche :

1. L'ordinateur **calcule** d'abord tout ce qui est à droite de la flèche.
2. Il **range** le résultat dans la variable de gauche.
3. L'**ancienne valeur** de la variable est **perdue**.

Parfois, la nouvelle valeur dépend de l'ancienne : `compteur ← compteur + 1`. On prend la valeur actuelle, on ajoute 1, on range le résultat au même endroit.

C'est une **MISE À JOUR**. Le programme officiel du BTS l'appelle « affectation récursive » : retenez simplement « mise à jour ».

PSEUDO-CODE

```
compteur ← 0
compteur ← compteur + 1
compteur ← compteur + 1
Écrire compteur
```

PYTHON

```
compteur = 0
compteur = compteur + 1
compteur = compteur + 1
print(compteur)
```

CE QUI S'AFFICHE

```
2
```

En maths, `x = x + 1` est impossible. En informatique, c'est une instruction : « prends x, ajoute 1, range dans x ». En Python, on peut aussi écrire `compteur += 1`, et `compteur -= 1` pour enlever 1.

On le fait ensemble — échanger deux valeurs

1. On a `a ← 3` et `b ← 8`. On veut que `a` vaille 8 et `b` vaille 3. Que vaut `a` après `a ← b` ?
2. Que vaut `b` après `b ← a` ?
3. Comment garder le 3 ?
4. Quelles sont les trois lignes de l'échange ?

	Exemple	Pourquoi
✔	`prix ← prix * 2`	On lit l'ancien prix, on le double, on range le résultat dans `prix`.
✘	`5 ← x`	La flèche va toujours vers une variable. On ne peut pas ranger quelque chose dans le nombre 5.
✘	Croire que `y ← x` lie `y` à `x` pour toujours	On copie la valeur une fois. Si `x` change ensuite, `y` ne bouge pas.

Checkpoint

1. On exécute `n ← 10`, puis `n ← n - 3`, puis `n ← n * 2`. Que vaut `n` ?
2. Comment écrit-on `n ← n * 2` en Python ?

À vous ! Les exercices de la notion 1.2

★☆☆ **Appliquer**
1.2.1 à 1.2.3
page 101

★★☆ **S'entraîner**
1.2.4 à 1.2.6
page 102

★★★ **Aller plus loin**
1.2.7 à 1.2.9
page 103

Commencez par le niveau 1. Passez au niveau suivant quand les trois exercices sont réussis.

1.3 Lire et écrire

Le programme d'Inès marche, mais les nombres 120 et 9 sont écrits dans le code. Demain, il y aura peut-être 12 pannes. Comment laisser Inès taper le nombre elle-même ?

LIRE, c'est demander une valeur à l'utilisateur, au clavier, et la ranger dans une variable. **ÉCRIRE**, c'est afficher un message ou une valeur à l'écran.

Un programme qui ne lit rien fait toujours le même calcul. Un programme qui lit devient un outil : on le lance, on tape les valeurs du jour, il répond.

Action	Pseudo-code	Python
Lire un texte	`Lire nom`	`nom = input("Nom ? ")`
Lire un entier	`Lire pannes`	`pannes = int(input("Pannes ? "))`
Lire un nombre à virgule	`Lire prix`	`prix = float(input("Prix ? "))`
Écrire	`Écrire "Bonjour", nom`	`print("Bonjour", nom)`

Attention

En Python, `input` renvoie **toujours du texte**. Même si on tape 9, on reçoit le texte `"9"`.

Pour calculer, on le transforme en nombre avec `int(...)` ou `float(...)`. Le chapitre 2 explique pourquoi.

PSEUDO-CODE

```
Écrire "Nombre de pannes ?"
Lire pannes
marche ← 120 - pannes
Écrire "Postes qui marchent :", marche
```

PYTHON

```
pannes = int(input("Nombre de pannes ? "))
marche = 120 - pannes
print("Postes qui marchent :", marche)
```

CE QUI S'AFFICHE

```
Nombre de pannes ? 12
Postes qui marchent : 108
```

Dans `print`, la virgule sépare les morceaux et ajoute un espace entre eux. Pour aller à la ligne, on fait un autre `print`. Le programme officiel du BTS limite la mise en forme à ça : des séparateurs et des retours à la ligne.

On le fait ensemble — dire bonjour

1. On veut demander le prénom. Quelle ligne Python ?
2. Faut-il `int(...)` autour ?
3. Quelle ligne affiche « Bonjour Inès » ?
4. Et en pseudo-code ?

	Exemple	Pourquoi
✔	`age = int(input("Âge ? "))`	On lit du texte et on le change aussitôt en entier.
✘	`age = input("Âge ? ")` puis `age + 1`	`age` est du texte. Python refuse d'ajouter un nombre à du texte.
✘	`print("Bonjour" prenom)`	Il manque la virgule entre les deux morceaux : erreur de syntaxe.

Checkpoint

1. Que fait l'instruction `Lire x` ?
2. Pourquoi écrit-on `float(input(...))` pour un prix ?

À vous ! Les exercices de la notion 1.3

★☆☆ **Appliquer**
1.3.1 à 1.3.3
page 104

★★☆ **S'entraîner**
1.3.4 à 1.3.6
page 105

★★★ **Aller plus loin**
1.3.7 à 1.3.9
page 106

Commencez par le niveau 1. Passez au niveau suivant quand les trois exercices sont réussis.

1.4 La trace d'un algorithme

Inès veut vérifier un algorithme écrit par un collègue, sans ordinateur. Comment savoir ce qu'il fait, ligne par ligne ?

La **TRACE** est un tableau qui montre la valeur de chaque variable après chaque instruction. On le remplit à la main, ligne par ligne, comme si on était l'ordinateur.

C'est la question la plus fréquente à l'examen : « Que renvoie l'algorithme pour… ? Justifier. »

La trace est la justification. Elle prouve que vous avez suivi le chemin, pas deviné le résultat.

Comment ça marche :

1. On fait **une colonne par variable**, plus une colonne pour l'étape.
2. On lit l'algorithme **ligne par ligne**, dans l'ordre.
3. À chaque affectation, on écrit la **nouvelle valeur** dans la colonne de la variable.
4. Une variable qui n'existe pas encore vaut `–`.

PSEUDO-CODE

```
x ← 4
y ← x + 3
x ← x * y
y ← y - x
```

Étape	Instruction	x	y
1	`x ← 4`	4	–
2	`y ← x + 3`	4	7
3	`x ← x * y`	28	7
4	`y ← y - x`	28	-21

En Python, on vérifie une trace en ajoutant des `print` :

PYTHON

```python
x = 4
y = x + 3
x = x * y
y = y - x
print("x =", x, "y =", y)
```

CE QUI S'AFFICHE

```
x = 28 y = -21
```

On le fait ensemble — le panier

1. L'algorithme est : `prix ← 20`, `qte ← 3`, `total ← prix * qte`, `total ← total - 5`. Combien de colonnes de variables ?

2. Que vaut `total` à l'étape 3 ?

3. Que vaut `total` à l'étape 4 ?

4. Que vaut `prix` à la fin ?

	Exemple	Pourquoi
✔	Recopier la valeur de `y` à chaque ligne, même si elle ne change pas	On voit l'état complet de la mémoire à chaque étape.
✘	Calculer `x ← x * y` avec l'ancienne valeur de `y`	On prend toujours la valeur de la ligne juste au-dessus.
✘	Donner seulement le résultat final	À l'examen, sans trace, la réponse n'est pas justifiée.

Checkpoint

1. Faites la trace de `a ← 5`, `b ← a * 2`, `a ← b - a`. Que valent `a` et `b` ?

2. À quoi sert une trace ?

À vous ! Les exercices de la notion 1.4

★☆☆ **Appliquer**
1.4.1 à 1.4.3
page 107

★★☆ **S'entraîner**
1.4.4 à 1.4.6
page 108

★★★ **Aller plus loin**
1.4.7 à 1.4.9
page 110

Commencez par le niveau 1. Passez au niveau suivant quand les trois exercices sont réussis.

Ce qu'on retient du chapitre 1

> **Si vous ne devez retenir que trois choses**
>
> 1. Les instructions s'exécutent dans l'ordre, de haut en bas.
>
> 2. `x ← expression` calcule d'abord la droite, puis range le résultat dans `x`. L'ancienne valeur est perdue.
>
> 3. Pour savoir ce que fait un algorithme, faites sa trace : une colonne par variable, une ligne par étape.

Inès a maintenant son programme. Elle tape le nombre de pannes, il affiche le résultat. Et quand un collègue lui envoie un algorithme, elle sait le vérifier sur papier, avec une trace.

Chapitre 2 — Les types de données

Yanis développe la fiche d'inscription d'une salle de sport. Il range l'âge du client, son poids, son nom, et s'il a payé ou non. Un jour, la fiche affiche un poids de 72,300000000000001 kg.

Un autre jour, l'âge « 25 » plus 1 donne « 251 ». Yanis ne comprend pas.

À la fin de ce chapitre, vous saurez choisir le bon type pour chaque donnée, et vous saurez expliquer ces deux bugs.

> **La question de départ**
>
> Le numéro de téléphone 06 12 34 56 78 : faut-il le ranger comme un nombre ou comme un texte ?

2.1 Les types simples

Une variable contient une valeur. Mais une valeur peut être un âge, un prix, une réponse oui ou non. L'ordinateur les range-t-il de la même façon ?

Le **TYPE** d'une variable dit quelle sorte de valeur elle contient, et donc ce qu'on peut faire avec. Les quatre types simples du programme sont l'entier naturel, l'entier relatif, le réel et le booléen.

Le type, c'est comme le rayon d'un magasin. On ne range pas le lait avec les tournevis. Chaque rayon a ses règles : le lait va au frais, les tournevis non.

Type	Ce qu'il contient	Exemples	En Python
Entier naturel	un nombre entier positif ou nul	0, 57 utilisateurs, 443 pour un port	`int`
Entier relatif	un nombre entier, positif ou négatif	-7 °C, un solde de -120 €	`int`
Réel	un nombre à virgule	73,4 % d'occupation du processeur, 12,67 de moyenne	`float`
Booléen	seulement deux valeurs : Vrai ou Faux	serveur en ligne ? a payé ?	`bool` : `True`, `False`

En Python, un réel s'écrit avec un **point** : `12.67`, jamais `12,67`. La division `/` donne toujours un réel, même quand elle tombe juste : `8 / 2` vaut `4.0`. La fonction `type` donne le type d'une valeur.

PYTHON

```python
print(type(57), type(-7), type(73.4), type(True))
print(8 / 2, type(8 / 2))
```

CE QUI S'AFFICHE

```
<class 'int'> <class 'int'> <class 'float'> <class 'bool'>
4.0 <class 'float'>
```

Python n'a pas de type « entier naturel » à part : un entier positif est un `int` comme les autres. C'est à vous de vérifier qu'il reste positif.

On le fait ensemble — la fiche de la salle de sport

1. Quel type pour l'âge du client ?
2. Quel type pour son poids, 72,3 kg ?
3. Quel type pour « a payé l'abonnement » ?
4. Quel type pour l'écart de poids depuis l'inscription, par exemple -4 kg ?

	Exemple	Pourquoi
✔	`connecte = True`	Une question à deux réponses se range dans un booléen.
✘	`moyenne = 12` pour une moyenne de 12,67	Un entier perd la partie après la virgule.
✘	`paye = "oui"`	Un texte peut valoir « Oui », « OUI », « o »... Le booléen n'a que deux valeurs, sans erreur de frappe.

Checkpoint

1. Quel type pour le nombre de fichiers dans un dossier ?
2. Quel est le type de `3 > 5` ?

À vous ! Les exercices de la notion 2.1

★☆☆ **Appliquer**
2.1.1 à 2.1.3
page 111

★★☆ **S'entraîner**
2.1.4 à 2.1.6
page 112

★★★ **Aller plus loin**
2.1.7 à 2.1.9
page 113

Commencez par le niveau 1. Passez au niveau suivant quand les trois exercices sont réussis.

2.2 Les limites de la machine

En maths, les nombres n'ont pas de fin. On peut toujours ajouter 1. Est-ce vrai dans un ordinateur ?

Le **DOMAINE DE VALEURS** d'un type est l'ensemble des valeurs qu'il peut contenir. En maths, il est infini.

Dans une machine, il est **limité**, parce que chaque valeur est rangée sur un nombre fixe de bits. Un **BIT** est un chiffre binaire : 0 ou 1.

Pensez au compteur kilométrique d'une vieille voiture, à 6 chiffres. Après 999 999 km, il repasse à 000 000. Le compteur n'a pas de place pour un 7e chiffre.

Comment ça marche :

1. Avec **n bits**, on peut écrire 2^n valeurs différentes. On lit « 2 puissance n ».
2. Un entier naturel sur n bits va donc de 0 à $2^n - 1$. Sur 8 bits : de 0 à 255.
3. Un entier relatif sur n bits partage ces valeurs entre négatifs et positifs : il va de -2^{n-1} à $2^{n-1} - 1$. Sur 16 bits : de -32 768 à 32 767.
4. Si un calcul sort du domaine, c'est un **DÉPASSEMENT DE CAPACITÉ**. Le résultat devient faux.

Bits	Entier naturel	Entier relatif	Exemple
8	0 à 255	-128 à 127	un nombre d'une adresse IPv4
16	0 à 65 535	-32 768 à 32 767	un numéro de port
32	0 à 4 294 967 295	-2 147 483 648 à 2 147 483 647	un entier en langage C

Les entiers sont **exacts** tant qu'ils restent dans leur domaine : 42 est rangé exactement. Python agrandit seul la place d'un `int` : il ne déborde jamais. Mais d'autres langages, comme le C, débordent.

Les réels, eux, sont presque toujours **approchés**. La machine les écrit en binaire. Or beaucoup de nombres simples, comme 0,1, n'ont pas d'écriture binaire finie.

C'est comme 1/3 en décimal : 0,3333... ne s'arrête jamais.

La machine coupe quelque part. Il reste une petite erreur.

PYTHON

```python
print(0.1 + 0.2)
print(0.1 + 0.2 == 0.3)
print(round(0.1 + 0.2, 2))
```

CE QUI S'AFFICHE

```
0.30000000000000004
False
0.3
```

C'est le bug de Yanis : son poids de 72,3 est rangé à peu près. Pour afficher un réel, on l'arrondit. Pour comparer deux réels, on ne teste jamais l'égalité exacte.

Instant histoire — 1996 : la fusée détruite par un nombre trop grand

Le 4 juin 1996, la fusée Ariane 5 décolle de Kourou pour son premier vol.

Moins de 40 secondes plus tard, elle se détruit en vol.

Un programme copiait une valeur liée à la vitesse dans un entier relatif sur 16 bits. Ariane 5 allait plus vite qu'Ariane 4 : la valeur était trop grande.

L'ordinateur de bord a reçu des données fausses et a perdu le contrôle.

Qu'est-ce qui aurait évité la catastrophe ?

On le fait ensemble — un compteur sur 8 bits

1. Combien de valeurs différentes sur 8 bits ?

2. Quelle est la plus grande valeur d'un entier naturel sur 8 bits ?

3. Un compteur sur 8 bits vaut 255. On ajoute 1. Que se passe-t-il ?

4. Faut-il 8 ou 16 bits pour compter 1 000 visiteurs ?

	Exemple	Pourquoi
✔	`round(prix, 2)` pour afficher un prix	On cache la petite erreur des réels à l'affichage.
✘	Comparer `0.1 + 0.2 == 0.3` et attendre Vrai	La comparaison vaut Faux : 0,1 + 0,2 n'est pas exactement 0,3 dans la machine.
✘	Ranger 300 dans un entier naturel sur 8 bits	300 dépasse 255 : dépassement de capacité.

Checkpoint

1. Quelle est la plus grande valeur d'un entier naturel sur 10 bits ?

2. Pourquoi `0.1 + 0.2` n'affiche-t-il pas 0.3 ?

À vous ! Les exercices de la notion 2.2

★☆☆ **Appliquer**
2.2.1 à 2.2.3
page 114

★★☆ **S'entraîner**
2.2.4 à 2.2.6
page 115

★★★ **Aller plus loin**
2.2.7 à 2.2.9
page 115

Commencez par le niveau 1. Passez au niveau suivant quand les trois exercices sont réussis.

2.3 Les chaînes de caractères

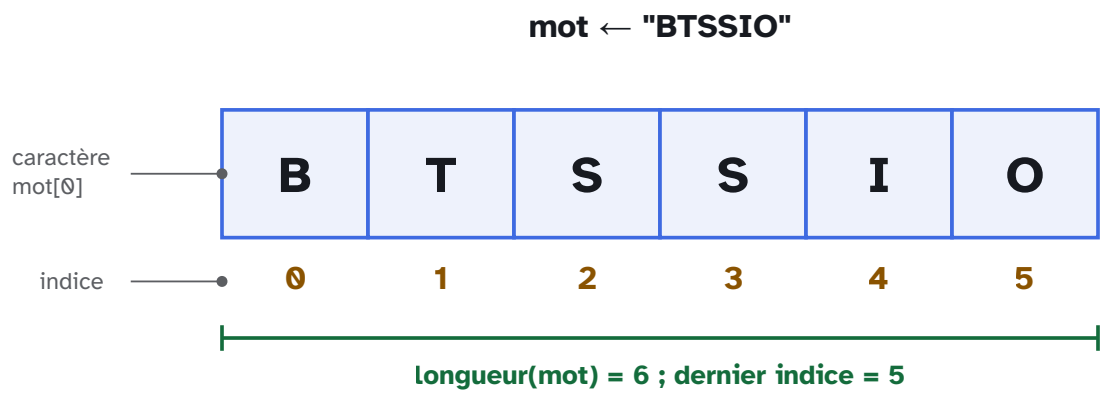
Le nom du client, son adresse mail, son mot de passe ne sont pas des nombres. Comment l'ordinateur range-t-il du texte ?

Une **CHAÎNE DE CARACTÈRES** est une suite de caractères : lettres, chiffres, espaces, ponctuation. On l'écrit entre guillemets : `"Yanis"`, `"192.168.1.10"`. En Python, son type est `str`, pour *string* en anglais.

Une chaîne est **un texte, d'un seul bloc**. On ne la modifie pas case par case. On lit ses caractères et on lui applique des fonctions, qui en donnent une information ou en extraient un morceau. C'est comme une phrase écrite au stylo : on peut la lire, la recopier, en citer un mot, mais pas gommer une lettre.

Comment ça marche :

1. `longueur(chaine)` donne le nombre de caractères. En Python : `len(chaine)`.
2. Chaque caractère a une **position**, appelée **INDICE**. Le premier caractère a l'indice **0**.
3. `chaine[i]` **extrait** le caractère d'indice `i` : c'est un petit morceau de la chaîne, de longueur 1. Les sujets du BTS l'écrivent ainsi : `code[i]`, `chaine[k]`.
4. Le dernier caractère a donc l'indice `longueur(chaine) - 1`.



La chaîne "BTSSIO" : six caractères, de l'indice 0 à l'indice 5

PSEUDO-CODE

```
mot ← "BTSSIO"
Écrire longueur(mot)
Écrire mot[0]
Écrire mot[5]
```

PYTHON

```
mot = "BTSSIO"
print(len(mot))
print(mot[0])
print(mot[5])
```

CE QUI S'AFFICHE

```
6
B
O
```

On lit un caractère, mais on ne le **modifie** pas. Écrire `mot[0] ← "X"` est impossible : Python s'arrête sur une erreur `TypeError`.

Pour changer un texte, on range une nouvelle chaîne dans la variable. Une chaîne n'est pas un tableau.

PYTHON

```
mot = "BTSSIO"
mot[0] = "X"
```

CE QUI S'AFFICHE

```
TypeError: 'str' object does not support item assignment
```

Attention : `"123"` et `123` ne sont pas la même chose. Le premier est une chaîne de trois caractères.

Le second est un nombre. On ne calcule pas sur une chaîne.

Dans la machine, chaque caractère est rangé sous forme d'un nombre. Le code ASCII, pour *American Standard Code for Information Interchange*, donne par exemple 65 pour `A` et 97 pour `a`. Vous n'avez pas à connaître ces codes : retenez seulement qu'une chaîne est rangée caractère par caractère.

On le fait ensemble — une adresse mail

1. On a `mail ← "ines@lycee.fr"`. Combien de caractères ?
2. Quel est l'indice du premier caractère ?
3. Que vaut `mail[4]` ?
4. Quel est l'indice du dernier caractère ?

	Exemple	Pourquoi
✔	`mot[len(mot) - 1]` pour le dernier caractère	La longueur moins 1, car on compte à partir de `0`.
✘	`mot[len(mot)]`	Cet indice n'existe pas : Python s'arrête sur une erreur `IndexError`.
✘	`mot[1]` pour le premier caractère	Le premier caractère est à l'indice `0`. `mot[1]` est le deuxième.
✘	`mot[0] ← "X"` pour changer la première lettre	On ne modifie pas un caractère d'une chaîne : c'est une erreur.

Checkpoint

1. Que vaut `"Python"[2]` ?
2. Quelle est la longueur de `"BTS SIO"` ?

À vous ! Les exercices de la notion 2.3

★☆☆ **Appliquer**
2.3.1 à 2.3.3
page 116

★★☆ **S'entraîner**
2.3.4 à 2.3.6
page 117

★★★ **Aller plus loin**
2.3.7 à 2.3.9
page 118

Commencez par le niveau 1. Passez au niveau suivant quand les trois exercices sont réussis.

2.4 Changer de type : le transtypage

Revenons au second bug de Yanis : « 25 » plus 1 donne « 251 ». D'où vient ce résultat ?

Le **TRANSTYPAGE** consiste à changer le type d'une valeur : d'un texte vers un nombre, d'un nombre vers un texte. On l'appelle aussi conversion.

C'est indispensable, car les données arrivent souvent en texte : ce qu'on tape au clavier, une ligne d'un fichier, un formulaire web. Pour calculer, il faut d'abord les changer en nombres.

Conversion	Pseudo-code	Python	Résultat
texte → entier	`entier("42")`	`int("42")`	42
texte → réel	`réel("3.5")`	`float("3.5")`	3.5
nombre → texte	`chaîne(42)`	`str(42)`	"42"
réel → entier	`entier(3.9)`	`int(3.9)`	3 : la virgule est coupée, pas arrondie
booléen → entier	`entier(Vrai)`	`int(True)`	1

Voici le bug de Yanis, puis sa correction :

PYTHON

```
age = "25"
print(age + "1")
print(int(age) + 1)
print("Âge : " + str(int(age) + 1))
```

CE QUI S'AFFICHE

```
251
26
Âge : 26
```

Avec deux chaînes, le `+` les **colle** : c'est la concaténation, au chapitre 3. Avec deux nombres, il les **additionne**. Le même signe fait deux choses selon le type.

On le fait ensemble — un prix tapé au clavier

1. L'utilisateur tape `12.5`. Quel est le type de ce que renvoie `input` ?
2. Comment la changer en nombre ?
3. Que donne `int("12.5")` ?
4. Comment afficher « Total : 25.0 » en collant du texte et un nombre ?

	Exemple	Pourquoi
✔	`int(input("Quantité ? "))`	On convertit tout de suite le texte lu en entier.
✘	`float("3,5")`	Python attend un point. Avec une virgule, c'est une erreur `ValueError`.
✘	`"Total : " + 25`	On ne colle pas un texte et un nombre. Il faut `str(25)`.

Checkpoint

1. Que vaut `int(7.8)` ?
2. Que vaut `"3" + "4"` en Python ?

À vous ! Les exercices de la notion 2.4

★☆☆ **Appliquer**
2.4.1 à 2.4.3
page 119

★★☆ **S'entraîner**
2.4.4 à 2.4.6
page 120

★★★ **Aller plus loin**
2.4.7 à 2.4.9
page 121

Commencez par le niveau 1. Passez au niveau suivant quand les trois exercices sont réussis.

Ce qu'on retient du chapitre 2

Si vous ne devez retenir que trois choses

1. Choisissez le type selon la donnée : entier, réel, booléen ou chaîne. Un numéro qu'on ne calcule pas est une chaîne.

2. Les entiers sont exacts mais limités. Les réels sont approchés : arrondissez pour afficher, ne testez jamais leur égalité exacte.

3. Ce qu'on lit au clavier est du texte : convertissez-le avec `int` ou `float` avant de calculer.

Yanis a corrigé sa fiche. Il convertit l'âge en entier avant d'ajouter 1.

Il arrondit le poids à une décimale avant de l'afficher. Et le numéro de téléphone reste une chaîne.

Chapitre 3 — Les opérateurs

Léa est stagiaire chez un hébergeur de sites web. On lui confie trois petits outils à écrire. Le premier convertit une durée en secondes en heures, minutes et secondes.

Le deuxième dit si un mot de passe est accepté. Le troisième lit les droits d'accès d'un fichier, codés sur quelques bits.

À la fin de ce chapitre, vous saurez écrire les calculs, les conditions et les manipulations de texte dont Léa a besoin.

> **La question de départ**
>
> Une vidéo dure 200 minutes. Combien d'heures et de minutes ? Comment avez-vous fait le calcul ?

3.1 Les opérateurs numériques

Pour calculer, un algorithme utilise des **opérateurs** : des signes qui combinent des valeurs. Lesquels a-t-on, et comment les écrit-on ?

Un **OPÉRATEUR NUMÉRIQUE** prend des nombres et donne un nombre. En plus des quatre opérations, l'informatique utilise beaucoup deux opérateurs : le **quotient entier** et le **reste** de la division euclidienne.

La division euclidienne, c'est le partage avec un reste. On partage 17 bonbons entre 5 enfants : chacun en a 3, et il en reste 2. Le quotient est 3, le reste est 2.

Opération	Pseudo-code	Python	Exemple	Résultat
Addition, soustraction	`+`, `-`	`+`, `-`	`7 - 2`	5
Multiplication	`*`	`*`	`7 * 2`	14
Division	`/`	`/`	`7 / 2`	3.5
Quotient entier	`div`	`//`	`17 // 5`	3
Reste	`mod`	`%`	`17 % 5`	2
Puissance	`^`	`**`	`2 ** 10`	1024
Signe	`-x`	`-x`	`-(3 - 8)`	5

Le reste sert partout. Un nombre `n` est **pair** si `n mod 2 = 0`. Il est **divisible** par 7 si `n mod 7 = 0`.

Voici l'outil de Léa, qui convertit une durée :

PSEUDO-CODE

```
duree ← 7384
heures ← duree div 3600
reste ← duree mod 3600
minutes ← reste div 60
secondes ← reste mod 60
Écrire heures, "h", minutes, "min",
secondes, "s"
```

PYTHON

```
duree = 7384
heures = duree // 3600
reste = duree % 3600
minutes = reste // 60
secondes = reste % 60
print(heures, "h", minutes, "min",
secondes, "s")
```

CE QUI S'AFFICHE

```
2 h 3 min 4 s
```

Écrire une formule en ligne. Une formule de maths s'écrit sur une ligne dans un programme. Les fractions et les puissances deviennent des opérateurs, et les **parenthèses** gardent l'ordre des calculs.

Comme en maths, `*` et `/` passent avant `+` et `-`. La puissance passe avant tout.

En maths	En ligne
$\frac{a+b}{2}$	`(a + b) / 2`
$3x^2 - 5$	`3 * x ** 2 - 5`
$\frac{1}{1+t}$	`1 / (1 + t)`

On le fait ensemble — 100 000 secondes

1. Combien d'heures entières dans 100 000 secondes ?
2. Combien de secondes restent après ces heures ?
3. Combien de minutes dans ce reste ?
4. Et combien de secondes ?

	Exemple	Pourquoi
✔	`(a + b) / 2` pour une moyenne	Les parenthèses font l'addition avant la division.
✘	`a + b / 2` pour une moyenne	La division passe d'abord : on calcule a + (b / 2).
✘	`17 / 5` pour avoir 3	`/` donne 3.4. Le quotient entier s'écrit `div`, ou `//` en Python.

Checkpoint

1. Que valent `23 div 4` et `23 mod 4` ?
2. Comment écrire $\frac{x-1}{x+1}$ en ligne ?

À vous ! Les exercices de la notion 3.1

★☆☆ **Appliquer**
3.1.1 à 3.1.3
page 122

★★☆ **S'entraîner**
3.1.4 à 3.1.6
page 122

★★★ **Aller plus loin**
3.1.7 à 3.1.9
page 123

Commencez par le niveau 1. Passez au niveau suivant quand les trois exercices sont réussis.

3.2 Les fonctions mathématiques usuelles

Certains calculs reviennent tout le temps : arrondir un prix, prendre une racine, tirer un nombre au hasard. Faut-il les écrire à chaque fois ?

Une **FONCTION USUELLE** est un calcul tout prêt, fourni par le langage. On l'appelle par son nom, avec la valeur entre parenthèses : `abs(-7)` donne 7.

C'est comme la touche « racine » de la calculatrice. Vous ne refaites pas le calcul à la main : vous appuyez sur la touche.

Fonction	Pseudo-code	Python	Exemple	Résultat
Valeur absolue	`abs(x)`	`abs(x)`	`abs(-7)`	7
Arrondi au plus près	`arrondi(x, 2)`	`round(x, 2)`	`round(3.678, 2)`	3.68
Arrondi par défaut	`partie_entière(x)`	`math.floor(x)`	`math.floor(3.7)`	3
Arrondi par excès	`plafond(x)`	`math.ceil(x)`	`math.ceil(3.2)`	4
Racine carrée	`racine(x)`	`math.sqrt(x)`	`math.sqrt(49)`	7.0
Entier au hasard entre a et b	`alea(a, b)`	`random.randint(a, b)`	`random.randint(1, 6)`	1, 2, 3, 4, 5 ou 6
Réel au hasard entre a et b	`alea_reel(a, b)`	`random.uniform(a, b)`	`random.uniform(0, 1)`	par exemple 0.4172…

En Python, `floor`, `ceil` et `sqrt` sont dans le module `math`, et `randint` et `uniform` dans le module `random`. On écrit `import math` ou `import random` en haut du programme.

PYTHON

```python
import math
print(round(3.678, 2), math.floor(3.7), math.ceil(3.2), math.sqrt(49))
print(math.ceil(130 / 50))
```

CE QUI S'AFFICHE

```
3.68 3 4 7.0
3
```

La dernière ligne répond à une vraie question : il faut 3 cartons de 50 pour ranger 130 câbles. On arrondit **par excès**, sinon des câbles restent dehors.

Le hasard d'un ordinateur est **pseudo-aléatoire** : il vient d'un calcul. Les nombres semblent tirés au hasard, mais une formule les produit. Pour un jeu ou une simulation, ça suffit.

Ce tirage est **uniforme** : chaque valeur de l'intervalle a la même chance de sortir. Avec `randint(1, 6)`, le 6 sort aussi souvent que le 1, comme avec un vrai dé.

On le fait ensemble — le dé

1. Quelle instruction Python simule un dé à 6 faces ?
2. Les bornes 1 et 6 peuvent-elles sortir ?
3. Quelle ligne faut-il avant ?
4. Comment tirer une note entière entre `0` et `20` ?

	Exemple	Pourquoi
✔	`math.ceil(130 / 50)` cartons	On arrondit par excès : un carton incomplet compte quand même.
✘	`round(130 / 50)` cartons	`round(2.6)` donne 3 ici, mais `round(2.2)` donnerait 2 et laisserait des câbles dehors.
✘	`sqrt(49)` sans `import math`	Python ne connaît pas `sqrt` seul : il faut `math.sqrt` et l'import.

Checkpoint

1. Que donne `math.floor(-2.5)` ?
2. Que donne `round(12.3456, 1)` ?

À vous ! Les exercices de la notion 3.2

★☆☆ **Appliquer**
3.2.1 à 3.2.3
page 125

★★☆ **S'entraîner**
3.2.4 à 3.2.6
page 125

★★★ **Aller plus loin**
3.2.7 à 3.2.9
page 126

Commencez par le niveau 1. Passez au niveau suivant quand les trois exercices sont réussis.

3.3 Comparer et combiner des conditions

Pour savoir si un mot de passe est accepté, l'outil de Léa doit tester plusieurs règles à la fois. Comment écrit-on une condition, puis plusieurs ?

Un **OPÉRATEUR DE COMPARAISON** compare deux valeurs et donne un **booléen** : Vrai ou Faux. Un **OPÉRATEUR BOOLÉEN** combine des booléens : NON, ET, OU, OUX.

Comparer, c'est poser une question fermée à l'ordinateur. « Le mot de passe fait-il au moins 8 caractères ? »

La réponse est oui ou non. Pour plusieurs questions, on les relie avec ET ou OU, comme en français.

Comparaison	Pseudo-code	Python
égal	`=`	`==`
différent	`≠`, ou `<>` dans certains textes	`!=`
plus petit, plus petit ou égal	`<`, `≤`	`<`, `<=`
plus grand, plus grand ou égal	`>`, `≥`	`>`, `>=`

On compare des nombres, mais aussi des chaînes : `"Alice" < "Bob"` est Vrai. Les chaînes se comparent caractère par caractère, comme dans un dictionnaire.

Attention : les caractères se comparent par leur code. Les chiffres viennent avant les majuscules, et les majuscules avant les minuscules : `"a" < "B"` est Faux.

a	b	NON a	a ET b	a OU b	a OUX b
Vrai	Vrai	Faux	Vrai	Vrai	Faux
Vrai	Faux	Faux	Faux	Vrai	Vrai
Faux	Vrai	Vrai	Faux	Vrai	Vrai
Faux	Faux	Vrai	Faux	Faux	Faux

En une phrase chacun :

- **NON a** inverse : Vrai devient Faux.
- **a ET b** est Vrai seulement si les deux sont Vrai.
- **a OU b** est Vrai si au moins un des deux est Vrai.
- **a OUX b**, le « ou exclusif », est Vrai si exactement un des deux est Vrai. En Python, on l'écrit `a != b` entre deux booléens.

Pseudo-code	NON	ET	OU	OUX
Python	`not`	`and`	`or`	`!=` entre booléens

Python lit un ET de gauche à droite et **s'arrête dès qu'un test est faux** : la suite n'est pas lue. Avec OU, il s'arrête dès qu'un test est vrai. On s'en sert pour protéger un test : `k < len(mot) and mot[k] == "S"` ne lit jamais une case qui n'existe pas.

Traduire une condition. Une règle en français devient une expression booléenne. On repère chaque petite condition, puis les mots « et », « ou », « ne… pas ».

Quand ET et OU se mélangent, ET passe avant OU, comme la multiplication avant l'addition. Pour ne pas se tromper, mettez des parenthèses autour de chaque morceau : `(a ET b) OU c`.

PSEUDO-CODE

```
mdp ← "Sio2026!"
long_ok ← longueur(mdp) ≥ 8
interdit ← mdp = "motdepasse"
accepte ← long_ok ET NON interdit
Écrire accepte
```

PYTHON

```python
mdp = "Sio2026!"
long_ok = len(mdp) >= 8
interdit = mdp == "motdepasse"
accepte = long_ok and not interdit
print(accepte)
```

CE QUI S'AFFICHE

```
True
```

Attention

`=` et `==` ne font pas la même chose en Python. `x = 5` **range** 5 dans x, et `x == 5` **teste** si x vaut 5.

En pseudo-code, la flèche `←` range, et le `=` teste.

On le fait ensemble — la majorité

1. Règle : « l'utilisateur a entre 18 et 25 ans inclus ». Quelles sont les deux petites conditions ?
2. Quel mot les relie ?
3. Quelle expression en Python ?
4. Que vaut-elle pour `age = 30` ?

	Exemple	Pourquoi
✔	`jour == "samedi" or jour == "dimanche"`	Chaque comparaison est écrite en entier des deux côtés du `or`.
✘	`jour == "samedi" or "dimanche"`	`"dimanche"` seul n'est pas une comparaison. Python ne compare pas `jour` à dimanche, et la condition devient vraie même un lundi.
✘	`age >= 18 or age <= 25` pour « entre 18 et 25 »	Avec OU, tout âge passe : 40 est bien supérieur à 18. Il faut ET.

Checkpoint

1. Que vaut `(3 > 2) and (5 < 1)` ?
2. Traduisez « le stock est vide ou le produit n'est plus vendu » avec `stock` et `vendu`.

À vous ! Les exercices de la notion 3.3

★☆☆ **Appliquer**
3.3.1 à 3.3.3
page 128

★★☆ **S'entraîner**
3.3.4 à 3.3.6
page 128

★★★ **Aller plus loin**
3.3.7 à 3.3.9
page 129

Commencez par le niveau 1. Passez au niveau suivant quand les trois exercices sont réussis.

3.4 Les opérateurs bit à bit

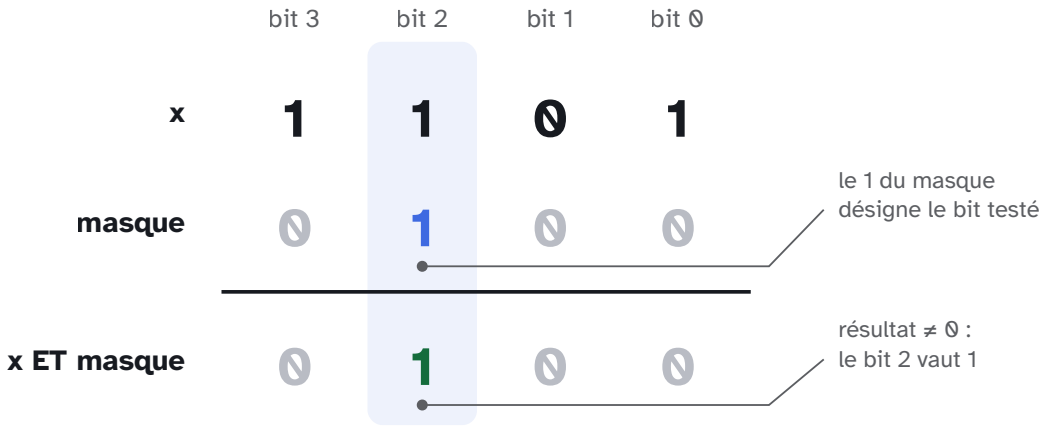
Les droits d'accès d'un fichier tiennent sur trois bits : lire, écrire, exécuter. Comment lire ou changer un seul de ces bits ?

Un **OPÉRATEUR BIT À BIT** applique ET, OU, OUX ou NON à chaque bit d'un nombre, rang par rang. On l'utilise avec un **MASQUE** : un nombre choisi exprès, dont les 1 désignent les bits qu'on veut toucher.

Un masque, c'est comme le pochoir d'un peintre. Les trous laissent passer la peinture sur certaines zones seulement. Le reste ne bouge pas.

Les bits se numérotent **de droite à gauche**, en commençant à 0. Dans `0b101`, le bit 0 vaut 1, le bit 1 vaut 0, le bit 2 vaut 1. En Python, `0b` signale un nombre écrit en binaire.

Ce qu'on veut	Opérateur	Python	Le masque contient
Tester un bit	ET bit à bit	`x & masque`	un 1 au rang testé
Mettre à 1 un bit	OU bit à bit	`x | masque`	un 1 au rang à allumer
Mettre à 0 un bit	ET avec l'inverse	`x & ~masque`	un 1 au rang à éteindre
Changer l'état d'un bit	OUX bit à bit	`x ^ masque`	un 1 au rang à inverser



Les bits 3, 1 et 0 sont mis à 0 par le masque : seul le bit 2 passe.

Masque : on teste le bit 2 de 1101 avec le masque 0100 ; seul le bit 2 passe

Les droits d'un fichier : le bit 2 pour lire, le bit 1 pour écrire, le bit 0 pour exécuter. `0b101` veut dire : lire oui, écrire non, exécuter oui.

PYTHON

```
droits = 0b101
print(bin(droits & 0b100))
print(bin(droits | 0b010))
print(bin(droits & ~0b001))
print(bin(droits ^ 0b001))
```

CE QUI S'AFFICHE

```
0b100
0b111
0b100
0b100
```

Le signe `~` inverse tous les bits : `~0b001` met des 1 partout, sauf au bit 0. La fonction `bin` affiche un nombre en binaire.

La 1re ligne donne un résultat différent de 0 : le droit de lire est présent. La 2e ajoute le droit d'écrire.

La 3e retire le droit d'exécuter. La 4e l'inverse : il était là, il disparaît.

On le fait ensemble — tester le bit 1 de 1011

1. Quel masque pour le bit 1 ?
2. Que vaut `1011 ET 0010`, rang par rang ?
3. Le bit 1 vaut-il 1 ?
4. Quel masque et quel opérateur pour mettre ce bit à 0 ?

	Exemple	Pourquoi
✔	`x | 0b0100` pour allumer le bit 2	OU avec 1 donne toujours 1, OU avec 0 ne change rien.
✘	Compter les bits à partir de la gauche	Le bit 0 est le plus à droite, celui des unités.
✘	`x & 0b0100` pour mettre le bit 2 à 0	Ça éteint tous les autres bits. Pour éteindre seulement le bit 2, c'est `x & ~0b0100`.

Checkpoint

1. Que vaut `0b1000 | 0b0001` ?
2. Avec quel masque change-t-on l'état du bit 3 ?

À vous ! Les exercices de la notion 3.4

★☆☆ **Appliquer**
3.4.1 à 3.4.3
page 130

★★☆ **S'entraîner**
3.4.4 à 3.4.6
page 131

★★★ **Aller plus loin**
3.4.7 à 3.4.9
page 132

Commencez par le niveau 1. Passez au niveau suivant quand les trois exercices sont réussis.

3.5 Les opérateurs de chaînes

Un journal de serveur contient des lignes comme `ERREUR 2026-10-18 disque plein`. Comment en tirer la date, ou savoir si le mot « ERREUR » y est ?

Les **OPÉRATEURS DE CHAÎNES** permettent de coller des chaînes, d'en extraire un morceau et d'y chercher un motif. Coller deux chaînes s'appelle la **CONCATÉNATION**. Un **MOTIF** est un petit texte qu'on cherche dans un plus grand.

On manipule du texte partout en informatique : des adresses mail, des adresses de sites web, des journaux d'erreurs, les messages affichés à l'utilisateur.

Ce qu'on veut	Pseudo-code	Python	Exemple	Résultat
Coller	`a + b`	`a + b`	`"BTS" + " SIO"`	`"BTS SIO"`
Le début, k caractères	`sous_chaine(s, 0, k)`	`s[:k]`	`"ERREUR 42"[:6]`	`"ERREUR"`
Le milieu, de i à j exclu	`sous_chaine(s, i, j)`	`s[i:j]`	`"BTS SIO"[4:7]`	`"SIO"`
La fin, k caractères	`sous_chaine(s, n - k, n)`	`s[-k:]`	`"rapport.pdf"[-3:]`	`"pdf"`
Chercher un motif	`position(s, motif)`	`s.find(motif)`	`"a@b.fr".find("@")`	1, ou -1 si absent

Dans ce tableau, `n` est la longueur de la chaîne. En Python, un indice négatif compte depuis la fin : `s[-1]` est le dernier caractère, et `s[-3:]` les trois derniers. `s[i:]` va de l'indice `i` jusqu'à la fin.

En Python, certaines fonctions s'écrivent **après** la variable, avec un point : `s.find(motif)` cherche dans `s`. On les appelle des méthodes. C'est seulement une autre façon d'écrire `position(s, motif)`.

Dans `s[i:j]`, le caractère d'indice `i` est **pris**, celui d'indice `j` est **exclu**. On prend donc `j - i` caractères.

PYTHON

```
ligne = "ERREUR 2026-10-18 disque plein"
print(ligne[:6])
print(ligne[7:17])
print(ligne.find("disque"))
print(ligne.find("réseau"))
message = "Date : " + ligne[7:17]
print(message)
```

CE QUI S'AFFICHE

```
ERREUR
2026-10-18
18
-1
Date : 2026-10-18
```

Le programme officiel permet aussi les expressions régulières simples, sans les étudier. Ce cours n'en a pas besoin.

On le fait ensemble — l'extension d'un fichier

1. On a `nom ← "photo_vacances.jpg"`. Quelle est la longueur ?
2. L'extension fait 3 caractères. Quels indices prend-elle ?
3. Quelle instruction Python la donne ?
4. Comment savoir si le nom contient « vacances » ?

	Exemple	Pourquoi
✔	`"Bonjour " + prenom`	On colle deux chaînes, avec l'espace dans la première.
✘	`s[4:6]` pour avoir 3 caractères	La borne 6 est exclue : on n'a que les indices 4 et 5. Il faut `s[4:7]`.
✘	`trouve = s.find("@") > 0` pour tester la présence	`find` renvoie 0 si le motif est au début : la présence serait ratée. On teste `s.find("@") != -1`.

Checkpoint

1. Que vaut `"192.168.1.10"[:3]` ?
2. Que vaut `"bonjour".find("j")` ?

À vous ! Les exercices de la notion 3.5

★☆☆ **Appliquer**
3.5.1 à 3.5.3
page 133

★★☆ **S'entraîner**
3.5.4 à 3.5.6
page 134

★★★ **Aller plus loin**
3.5.7 à 3.5.9
page 135

Commencez par le niveau 1. Passez au niveau suivant quand les trois exercices sont réussis.

Ce qu'on retient du chapitre 3

Si vous ne devez retenir que trois choses

1. `div` et `mod` donnent le quotient et le reste : `n mod 2 = 0` teste si `n` est pair.
2. Une condition composée s'écrit avec ET, OU, NON, et chaque comparaison s'écrit en entier.
3. Pour un bit, on utilise un masque : ET pour tester, OU pour mettre à 1, OUX pour inverser.

Les trois outils de Léa sont prêts. Le convertisseur de durée utilise `div` et `mod`.

Le contrôle du mot de passe combine ses règles avec ET. Et les droits du fichier se lisent avec un masque.

Chapitre 4 — Les structures de contrôle

Hugo gère le parc informatique d'une médiathèque. Il veut un programme qui vérifie chaque poste : s'il est trop vieux, il faut le remplacer. Il veut aussi compter les postes à remplacer, et redemander une valeur tant qu'elle n'est pas correcte.

Jusqu'ici, nos programmes lisent les lignes une fois, de haut en bas. Ça ne suffit plus.

À la fin de ce chapitre, vous saurez faire des choix et des répétitions dans un algorithme. C'est le cœur du programme, et le cœur de l'examen.

La question de départ

Vous devez écrire « Bonjour » 1 000 fois. Allez-vous écrire 1 000 lignes ?

4.1 Faire un choix : Si… Alors… Sinon

Hugo veut un message différent selon l'âge du poste. Comment faire exécuter une instruction seulement dans certains cas ?

Une **STRUCTURE CONDITIONNELLE** exécute un bloc d'instructions seulement si une condition est vraie. On l'appelle aussi une **ALTERNATIVE**. Elle s'écrit `Si … Alors … Sinon … Fin de Si`.

C'est un carrefour. Selon le panneau, on prend à gauche ou à droite. On ne prend jamais les deux routes.

Comment ça marche :

1. L'ordinateur **évalue la condition** : Vrai ou Faux.
2. Si elle est **Vrai**, il exécute le bloc après `Alors`.
3. Sinon, il exécute le bloc après `Sinon`, s'il y en a un.
4. Puis il continue **après** `Fin de Si`.

Pour plus de deux cas, on enchaîne avec `Sinon si`. Les conditions sont testées dans l'ordre. Le premier cas vrai gagne, les suivants sont ignorés.

PSEUDO-CODE

```
age ← 7
Si age ≥ 6 Alors
    Écrire "À remplacer"
Sinon si age ≥ 4 Alors
    Écrire "À surveiller"
Sinon
    Écrire "En bon état"
Fin de Si
```

PYTHON

```python
age = 7
if age >= 6:
    print("À remplacer")
elif age >= 4:
    print("À surveiller")
else:
    print("En bon état")
```

CE QUI S'AFFICHE

```
À remplacer
```

En Python, le bloc est marqué par **l'indentation** : les lignes décalées de 4 espaces après les deux-points `:`. Il n'y a pas de `Fin de Si`. C'est le retour à gauche qui ferme le bloc.

On le fait ensemble — avec un poste de 5 ans

1. On remplace `age ← 7` par `age ← 5`. La condition `age ≥ 6` est-elle vraie ?
2. Que teste-t-on ensuite ?
3. Qu'affiche le programme ?
4. Le bloc `Sinon` est-il exécuté ?

	Exemple	Pourquoi
✔	Tester `age ≥ 6` avant `age ≥ 4`	Le cas le plus fort d'abord. Un poste de 7 ans affiche « À remplacer ».
✘	Tester `age ≥ 4` avant `age ≥ 6`	Un poste de 7 ans s'arrête au premier test : « À surveiller ». Le cas « À remplacer » n'arrive jamais.
✘	Oublier les deux-points ou l'indentation en Python	Python s'arrête sur une erreur de syntaxe.

Choisir un jeu d'essai

Pour tester un algorithme qui contient des Si, on ne tape pas une valeur au hasard. On prépare un **JEU D'ESSAI** : une liste de valeurs à tester, chacune avec son résultat attendu.

1. **Une valeur par branche** : ici 7, 5 et 2 ans.
2. **Les valeurs limites**, là où le test change de résultat : 6 et 4.
3. **Un cas d'erreur**, s'il existe : un âge négatif, par exemple.

Âge testé	Résultat attendu	Résultat obtenu
7	À remplacer	
6	À remplacer	
5	À surveiller	
4	À surveiller	
2	En bon état	
-1	valeur refusée	

On remplit la dernière colonne en exécutant le programme. Ici, -1 donne « En bon état » : le test révèle qu'il manque un contrôle de l'âge. À l'examen, un sujet peut vous demander le résultat pour une valeur précise : votre jeu d'essai vous y prépare.

Checkpoint

1. On a `x ← -3`. Qu'affiche `Si x > 0 Alors Écrire "positif" Sinon Écrire "négatif ou nul" Fin de Si` ?

2. Combien de blocs sont exécutés dans un `Si … Sinon si … Sinon` ?

À vous ! Les exercices de la notion 4.1

★☆☆ **Appliquer**
4.1.1 à 4.1.3
page 136

★★☆ **S'entraîner**
4.1.4 à 4.1.6
page 137

★★★ **Aller plus loin**
4.1.7 à 4.1.9
page 139

Commencez par le niveau 1. Passez au niveau suivant quand les trois exercices sont réussis.

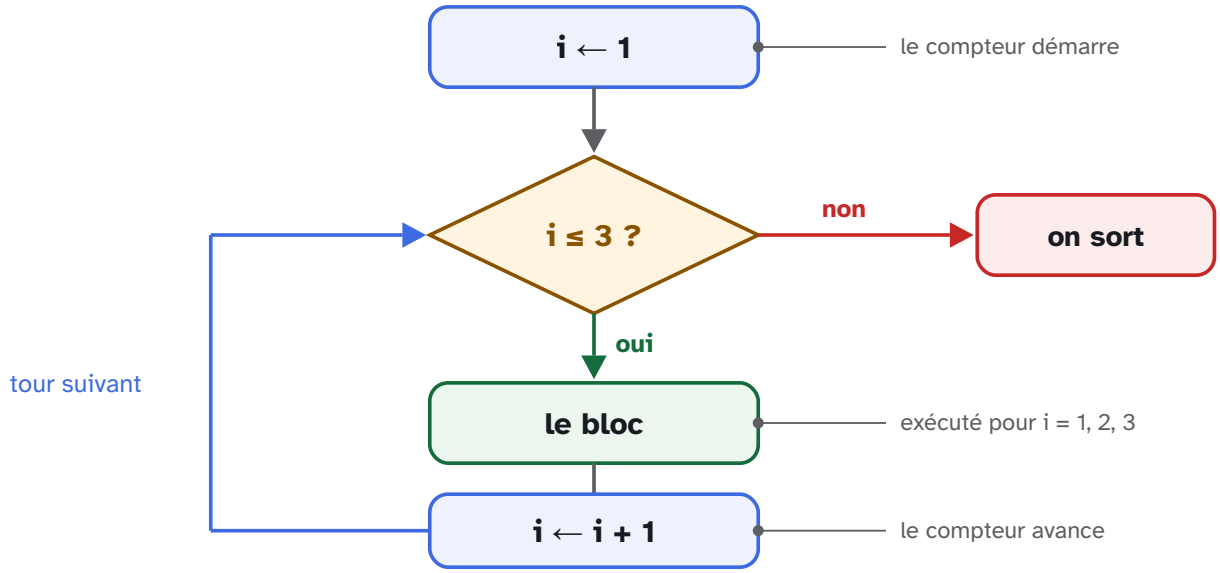
4.2 Répéter un nombre connu de fois : la boucle Pour

Hugo a 30 postes à vérifier. Il sait combien il y en a. Comment répéter le même traitement pour chacun, sans écrire 30 fois le code ?

Une **BOUCLE** répète un bloc d'instructions. Chaque passage dans la boucle s'appelle une **ITÉRATION**, ou un tour.

La boucle **POUR** sert quand on connaît le nombre de tours à l'avance. Elle utilise un **COMPTEUR**, une variable qui prend une nouvelle valeur à chaque tour.

C'est l'entraîneur qui dit : « 10 pompes. » Vous comptez 1, 2, 3... jusqu'à 10, puis vous vous arrêtez.



La boucle Pour : i prend les valeurs 1, 2, 3, et le bloc s'exécute à chaque tour

Comment ça marche :

1. Le compteur prend la **première valeur**.
2. On exécute **le bloc**.
3. Le compteur passe à la **valeur suivante**. On recommence le bloc.
4. Après la **dernière valeur**, on sort de la boucle.

PSEUDO-CODE

```
somme ← 0
Pour i allant de 1 à 4 Faire
    somme ← somme + i
Fin de Pour
Écrire somme
```

PYTHON

```
somme = 0
for i in range(1, 5):
    somme = somme + i
print(somme)
```

```
CE QUI S'AFFICHE
10
```

En Python, `range(1, 5)` donne 1, 2, 3, 4 : **la borne de droite est exclue**. Pour aller de 1 à n, on écrit `range(1, n + 1)`. `range(n)` seul va de 0 à n - 1 : c'est pratique pour les indices d'une chaîne.

Si la fin est plus petite que le début, la boucle Pour fait **0 tour** : `Pour i allant de 5 à 3 Faire` ne fait rien. En Python, `range(5, 4)` est vide.

Étape	i	somme
départ	–	0
tour 1	1	1
tour 2	2	3
tour 3	3	6
tour 4	4	10

On le fait ensemble — compter les « 1 » d'un code

1. On a `code ← "1011001"`. Combien de tours de boucle pour lire tous les caractères ?
2. De quel indice à quel indice va `i` ?
3. Que fait-on à chaque tour ?
4. Quelle valeur faut-il donner à `compt` avant la boucle ?
5. Que vaut `compt` à la fin ?

	Exemple	Pourquoi
✔	`for i in range(len(code)):`	`i` prend tous les indices de 0 à `len(code) - 1`.
✘	`for i in range(1, 10):` pour aller de 1 à 10	La borne 10 est exclue : on s'arrête à 9. Il faut `range(1, 11)`.
✘	`compt ← 0` écrit **dans** la boucle	Le compteur repart à 0 à chaque tour : à la fin, il vaut 0 ou 1.

Checkpoint

1. Combien de fois s'exécute le bloc de `Pour i allant de 3 à 7 Faire` ?
2. Quelles valeurs donne `range(4)` ?

À vous ! Les exercices de la notion 4.2

★☆☆ **Appliquer**
4.2.1 à 4.2.3
page 140

★★☆ **S'entraîner**
4.2.4 à 4.2.6
page 141

★★★ **Aller plus loin**
4.2.7 à 4.2.9
page 143

Commencez par le niveau 1. Passez au niveau suivant quand les trois exercices sont réussis.

4.3 Répéter tant que : les boucles Tant que et Répéter

Hugo veut que l'utilisateur tape un âge entre 0 et 15. Tant que la valeur est fausse, on redemande. Combien de fois ?

On ne sait pas à l'avance. La boucle Pour ne convient pas.

La boucle **TANT QUE** répète un bloc **tant qu'une condition est vraie**. Elle teste la condition **avant** chaque tour.

La boucle **RÉPÉTER ... JUSQU'À** exécute le bloc, **puis** teste : elle fait donc toujours au moins un tour, et elle s'arrête quand la condition devient vraie.

C'est comme remplir un verre au robinet. Tant que le verre n'est pas plein, on continue de verser. On ne sait pas d'avance combien de secondes il faudra.

Comment ça marche, pour Tant que :

1. On **teste** la condition.
2. Si elle est **Vrai**, on exécute le bloc, puis on revient à l'étape 1.
3. Si elle est **Faux**, on sort de la boucle.
4. Le bloc doit **changer** quelque chose dans la condition. Sinon, elle reste vraie pour toujours : c'est une **BOUCLE INFINIE**.

Voici le calcul d'un reste par soustractions. C'est l'idée du sujet du BTS 2025 :

PSEUDO-CODE

```
n ← 80
Tant que n ≥ 26 Faire
    n ← n - 26
Fin de Tant que
Écrire n
```

PYTHON

```
n = 80
while n >= 26:
    n = n - 26
print(n)
```

CE QUI S'AFFICHE

```
2
```

Étape	n	n ≥ 26 ?
départ	80	Vrai
tour 1	54	Vrai
tour 2	28	Vrai
tour 3	2	Faux : on sort

Lire une fonction dans un sujet

Les sujets du BTS écrivent presque toujours l'algorithme dans une fonction, comme ceci :

PSEUDO-CODE

```
Fonction reste_division_par_26(N)
    Tant que N ≥ 26 Faire
        N ← N - 26
    Fin de Tant que
    Renvoyer N
Fin de Fonction
```

Lisez-la ainsi : la fonction **reçoit** une valeur, rangée dans `N`. Elle fait son calcul. `Renvoyer N` donne le **résultat**.

`reste_division_par_26(80)` veut donc dire : « lance le calcul avec N = 80 ». Le résultat est 2.

La capsule 5.3 explique les fonctions en détail. Pour les exercices de ce chapitre, cette lecture suffit.

Python n'a pas de boucle Répéter. On l'imite avec `while True`, une boucle sans fin, et `break`, qui force la sortie :

PSEUDO-CODE

```
Répéter
    Écrire "Âge du poste (0 à 15) ?"
    Lire age
Jusqu'à age ≥ 0 ET age ≤ 15
Écrire "Âge enregistré :", age
```

PYTHON

```
while True:
    age = int(input("Âge du poste (0 à 15) ? "))
    if age >= 0 and age <= 15:
        break
print("Âge enregistré :", age)
```

CE QUI S'AFFICHE

```
Âge du poste (0 à 15) ? 40
Âge du poste (0 à 15) ? 12
Âge enregistré : 12
```

Je connais le nombre de tours	Je ne le connais pas, je peux faire 0 tour	Je ne le connais pas, je fais au moins 1 tour
Pour	**Tant que**	**Répéter … Jusqu'à**

Les boucles Pour et Répéter, écrites avec Tant que

Chaque boucle est elle-même un petit algorithme. Un simple Tant que suffit pour écrire les deux autres.

La boucle `Pour i allant de 1 à 4 Faire … Fin de Pour` fait exactement ceci :

PSEUDO-CODE

```
i ← 1
Tant que i ≤ 4 Faire
    // le bloc de la boucle
    i ← i + 1
Fin de Tant que
```

La boucle `Répéter … Jusqu'à condition` exécute le bloc une fois, puis le refait tant que la condition est fausse :

PSEUDO-CODE

```
// le bloc de la boucle, une première fois
Tant que NON condition Faire
    // le bloc de la boucle
Fin de Tant que
```

Dans la version Tant que du Pour, on voit tout le travail du compteur. Il part de 1. Il est testé avant chaque tour, et il avance de 1 à la fin du tour.

On le fait ensemble — chercher une lettre

1. On cherche le premier `"S"` dans `"BTSSIO"`. On part de `k ← 0`. Quelle est la condition pour continuer ?

2. Que met-on dans le bloc ?

3. Que vaut `k` à la sortie ?

4. Que se passe-t-il si on cherche `"A"`, absent du mot ?

	Exemple	Pourquoi
✔	`Tant que k < longueur(mot) ET mot[k] ≠ lettre Faire`	On s'arrête si on trouve la lettre, ou si on arrive au bout.
✘	`Tant que n ≥ 26 Faire` avec un bloc qui ne change pas `n`	La condition reste vraie pour toujours : boucle infinie.
✘	`Jusqu'à age < 0` pour « jusqu'à ce que l'âge soit correct »	Répéter s'arrête quand la condition est **vraie**. Il faut écrire la condition d'un âge correct.

Checkpoint

1. Qu'affiche `n ← 100`, `Tant que n > 10 Faire n ← n div 2 Fin de Tant que`, `Écrire n` ?

2. Quelle boucle fait toujours au moins un tour ?

À vous ! Les exercices de la notion 4.3

★☆☆ **Appliquer**
4.3.1 à 4.3.3
page 145

★★☆ **S'entraîner**
4.3.4 à 4.3.6
page 146

★★★ **Aller plus loin**
4.3.7 à 4.3.9
page 147

Commencez par le niveau 1. Passez au niveau suivant quand les trois exercices sont réussis.

4.4 Construire une boucle de calcul

Hugo veut la valeur totale de son parc, ou le nombre de postes à remplacer. Ce sont des calculs qui s'accumulent tour après tour. Comment construire ce genre de boucle sans se tromper ?

Une boucle de calcul se construit toujours en **trois temps** : la **PRÉPARATION**, l'**ITÉRATION** et la **MISE EN FORME FINALE**. On l'appelle un **calcul itératif**. Pendant l'itération, une variable **accumule** le résultat : c'est un **ACCUMULATEUR**.

C'est comme une tirelire. On la vide au départ.

À chaque pièce, on ajoute sa valeur. À la fin, on compte et on annonce le total.

Comment ça marche :

1. **Préparation** : on donne à l'accumulateur sa valeur de départ. Pour une somme, c'est **0**. Pour un produit, c'est **1**. Ce sont les nombres qui ne changent rien : 0 pour l'addition, 1 pour la multiplication. On les appelle des **éléments neutres**.
2. **Itération** : à chaque tour, on met à jour l'accumulateur avec la valeur du tour.
3. **Mise en forme finale** : après la boucle, on calcule et on présente le résultat : division pour une moyenne, arrondi, message.

En maths, une somme qui dépend de n s'écrit avec le symbole **Σ**, la lettre grecque sigma :

$$\sum_{i=1}^{n} i^2 = 1^2 + 2^2 + \cdots + n^2$$

On lit « somme des i au carré, pour i allant de 1 à n ». Le **Π**, la lettre grecque pi majuscule, fait la même chose avec un produit : $\prod_{i=1}^{n} i = 1 \times 2 \times \cdots \times n$.

Chaque Σ devient une boucle Pour, avec un accumulateur qui part de 0. Chaque Π devient une boucle Pour, avec un accumulateur qui part de 1.

PSEUDO-CODE

```
n ← 4
s ← 0
p ← 1
Pour i allant de 1 à n Faire
    s ← s + i ^ 2
    p ← p * i
Fin de Pour
Écrire "Somme :", s, "Produit :", p
```

PYTHON

```
n = 4
s = 0
p = 1
for i in range(1, n + 1):
    s = s + i ** 2
    p = p * i
print("Somme :", s, "Produit :", p)
```

```
CE QUI S'AFFICHE
Somme : 30 Produit : 24
```

En Python, il existe `sum(...)`. On ne s'en sert pas ici : à l'examen, c'est la boucle qu'on vous demande d'écrire.

Construire la boucle avec une propriété

Pour écrire une boucle de calcul sans erreur, on cherche ce qui reste vrai à la fin de chaque tour. Pour la somme des entiers : « après le tour i, `s` contient 1 + 2 + … + i ».

1. C'est vrai au départ : avant le premier tour, `s` vaut `0`, la somme de zéro nombre.
2. Si c'est vrai après le tour i - 1, l'instruction `s ← s + i` le rend vrai après le tour i.
3. C'est donc vrai après le dernier tour : `s` contient la somme cherchée.

C'est le **RAISONNEMENT PAR RÉCURRENCE**. On ne vous demande pas de le rédiger : il sert à construire la boucle.

Un calcul qui part de la valeur précédente définit une **SUITE** : $u_{n+1} = f(u_n)$. Par exemple, une taille qui double chaque jour : $t_{n+1} = 2 \times t_n$. La boucle calcule les termes un par un, avec `t ← 2 * t`.

On le fait ensemble — la moyenne de 5 notes

1. On lit 5 notes au clavier. Quelle boucle ?
2. Quelle préparation ?
3. Que fait-on à chaque tour ?
4. Quelle mise en forme finale ?

	Exemple	Pourquoi
✔	`p ← 1` avant un produit	1 ne change pas le produit. Avec `0`, le résultat serait toujours `0`.
✘	`moyenne ← total / 5` écrit dans la boucle	On divise à chaque tour un total pas encore complet. La division se fait après.
✘	`s ← 0` oublié	La variable n'existe pas au premier tour : erreur, ou valeur au hasard dans d'autres langages.

Checkpoint

1. Quelle valeur de départ pour calculer $\prod_{i=1}^{5} i$?
2. Traduisez $\sum_{k=1}^{10} 2k$ en boucle Python.

À vous ! Les exercices de la notion 4.4

★☆☆ Appliquer
4.4.1 à 4.4.3
page 149

★★☆ S'entraîner
4.4.4 à 4.4.6
page 150

★★★ Aller plus loin
4.4.7 à 4.4.9
page 151

Commencez par le niveau 1. Passez au niveau suivant quand les trois exercices sont réussis.

4.5 Les structures imbriquées

Hugo a plusieurs salles, et plusieurs postes par salle. Il veut un message seulement pour les postes trop vieux. Comment mettre une boucle dans une boucle, ou une condition dans une boucle ?

Des **STRUCTURES IMBRIQUÉES** sont des structures placées l'une dans l'autre : une condition dans une boucle, une boucle dans une boucle. La structure intérieure s'exécute **entièrement** à chaque tour de la structure extérieure.

C'est l'horloge. L'aiguille des minutes fait un tour complet pendant que celle des heures avance d'un cran.

Comment ça marche, pour deux boucles :

1. La boucle **extérieure** prend sa première valeur.
2. La boucle **intérieure** fait **tous** ses tours.
3. La boucle extérieure passe à la valeur suivante, et la boucle intérieure **recommence** depuis le début.
4. Le bloc intérieur s'exécute donc (tours extérieurs) × (tours intérieurs) fois.

PSEUDO-CODE

```
Pour i allant de 1 à 3 Faire
    ligne ← ""
    Pour j allant de 1 à i Faire
        ligne ← ligne + "*"
    Fin de Pour
    Écrire ligne
Fin de Pour
```

PYTHON

```
for i in range(1, 4):
    ligne = ""
    for j in range(1, i + 1):
        ligne = ligne + "*"
    print(ligne)
```

CE QUI S'AFFICHE

```
*
**
***
```

Ici, la boucle intérieure **dépend** de la boucle extérieure : elle va jusqu'à `i`. Le programme officiel limite ces boucles qui dépendent l'une de l'autre à deux niveaux.

Le produit de deux matrices. C'est l'exemple type de boucles imbriquées. Une **MATRICE** est un tableau de nombres, en lignes et en colonnes. La formule du produit $C = A \times B$ est donnée à l'examen :

$$c_{ij} = \sum_{k=1}^{n} a_{ik} \times b_{kj}$$

On lit : pour la case en ligne i, colonne j de C, on multiplie la ligne i de A par la colonne j de B. On multiplie terme à terme, puis on additionne.

Il faut trois boucles imbriquées : i pour les lignes, j pour les colonnes, k pour la somme.

En Python, on range une matrice comme une liste de lignes. `A[i][k]` est le nombre de la ligne `i`, colonne `k`. On compte à partir de `0`.

La capsule 5.2 détaille ces tableaux à deux dimensions.

PYTHON

```
A = [[1, 2], [3, 4]]
B = [[5, 6], [7, 8]]
C = [[0, 0], [0, 0]]
for i in range(2):
    for j in range(2):
        s = 0
        for k in range(2):
            s = s + A[i][k] * B[k][j]
        C[i][j] = s
print(C)
```

CE QUI S'AFFICHE

```
[[19, 22], [43, 50]]
```

On le fait ensemble — combien de tours ?

1. `Pour i allant de 1 à 4` contient `Pour j allant de 1 à 3`. Combien de fois s'exécute le bloc intérieur ?
2. Que vaut `j` au tout premier passage ?
3. Quand `i` passe de 1 à 2, que vaut `j` ?
4. Où remettre `s ← 0` dans le produit de matrices ?

	Exemple	Pourquoi
✔	`s ← 0` entre la boucle j et la boucle k	Chaque case du résultat a sa propre somme.
✘	`s ← 0` une seule fois, avant toutes les boucles	Les sommes de toutes les cases s'ajoutent : seule la première case est juste.
✘	Mal indenter la boucle intérieure	En Python, l'indentation dit ce qui est dedans. Une ligne mal décalée sort de la boucle.

Combien de tours ?

Deux algorithmes peuvent donner le même résultat avec un travail très différent. Pour les comparer, on compte les tests faits par la boucle.

Pour savoir si `n` est premier, la version A compte tous les diviseurs de 1 à `n`. La version B cherche un diviseur entre 2 et `n - 1`, et s'arrête au premier trouvé :

```
PSEUDO-CODE
i ← 2
Tant que i < n ET n mod i ≠ 0 Faire
    i ← i + 1
Fin de Tant que
premier ← (i = n)
```

n	Version A : tests de reste	Version B : tests de reste
91	91	6
97	97	95

Pour 91, qui vaut 7 × 13, la version B s'arrête dès 7. Pour 97, qui est premier, elle fait presque autant de tests que A.

À l'examen, on peut vous demander combien de fois une boucle tourne : comptez les tours sur un petit exemple.

Checkpoint

1. Combien de lignes affiche le programme des étoiles si on remplace 3 par 5 ?

2. Dans une boucle Pour de 1 à 10, on écrit `Si i mod 2 = 0 Alors compt ← compt + 1`. Que vaut `compt` à la fin, s'il partait de 0 ?

À vous ! Les exercices de la notion 4.5

★☆☆ **Appliquer**
4.5.1 à 4.5.3
page 152

★★☆ **S'entraîner**
4.5.4 à 4.5.6
page 154

★★★ **Aller plus loin**
4.5.7 à 4.5.9
page 155

Commencez par le niveau 1. Passez au niveau suivant quand les trois exercices sont réussis.

Ce qu'on retient du chapitre 4

> **Si vous ne devez retenir que trois choses**
>
> 1. Si pour choisir, Pour quand on connaît le nombre de tours, Tant que quand on ne le connaît pas.
>
> 2. Une boucle de calcul : préparation avant, mise à jour dedans, mise en forme après. Une somme part de 0, un produit de 1.
>
> 3. Dans une boucle Tant que, le bloc doit faire avancer la condition, sinon la boucle ne s'arrête jamais.

Le programme d'Hugo parcourt maintenant chaque salle, chaque poste. Il affiche les postes à remplacer, les compte, et redemande une valeur tant qu'elle est fausse.

Chapitre 5 — Les tableaux et les fonctions

Sarah est développeuse dans une petite entreprise. Elle doit traiter les notes de tous les stagiaires : la moyenne de chacun, la moyenne de chaque module, la meilleure note.

Elle a déjà écrit trois fois le même calcul de moyenne, à trois endroits du programme. Et ses 24 stagiaires ont chacun 4 notes : 96 variables, c'est impossible à gérer.

À la fin de ce chapitre, vous saurez ranger beaucoup de valeurs dans un tableau, et écrire un calcul une seule fois dans une fonction.

> **La question de départ**
>
> Dans un tableur, comment retrouvez-vous la note du 3e stagiaire au 2e module ?

5.1 Les tableaux à une dimension

Sarah a 4 notes pour un stagiaire. Plutôt que 4 variables, peut-on les ranger ensemble, sous un seul nom ?

Un **TABLEAU** range plusieurs valeurs sous un seul nom. Chaque valeur est un **ÉLÉMENT**, repéré par son **indice**, à partir de 0.

Un tableau à une dimension est une simple liste de valeurs. En Python, on l'appelle une **liste** et on l'écrit entre crochets : `[12, 15, 9, 18]`.

Un tableau, c'est comme une rangée de casiers numérotés à la piscine. Un seul nom pour la rangée, un numéro par casier.

Comment ça marche :

1. **Construire** : `notes ← [12, 15, 9, 18]`. On peut aussi partir d'un tableau **vide**, `T ← []`, puis le remplir avec `ajouter`.
2. **Lire** un élément : `notes[2]` vaut 9.
3. **Écrire** un élément : `notes[2] ← 11` remplace 9 par 11.
4. **Parcourir** : une boucle sur les indices, de `0` à `longueur(notes) - 1`. Si l'indice ne sert pas, on parcourt directement les valeurs : `Pour chaque x dans notes Faire`. En Python : `for x in notes:`.
5. **Ajouter** à la fin : `ajouter(notes, 14)`. En Python : `notes.append(14)`.

Le plus souvent, un tableau est **homogène** : tous ses éléments ont le même type. La capsule 5.2 montre un cas où les colonnes ont des types différents.

PSEUDO-CODE

```
notes ← [12, 15, 9, 18]
notes[2] ← 11
meilleure ← notes[0]
Pour i allant de 1 à longueur(notes) -
1 Faire
    Si notes[i] > meilleure Alors
        meilleure ← notes[i]
    Fin de Si
Fin de Pour
Écrire notes, meilleure
```

PYTHON

```python
notes = [12, 15, 9, 18]
notes[2] = 11
meilleure = notes[0]
for i in range(1, len(notes)):
    if notes[i] > meilleure:
        meilleure = notes[i]
print(notes, meilleure)
```

CE QUI S'AFFICHE

```
[12, 15, 11, 18] 18
```

C'est la méthode du **maximum** : on suppose que le premier est le meilleur, puis on compare avec chacun des suivants.

Copier un tableau. C'est le piège classique de Python. `b = a` ne copie pas le tableau : `b` devient un **deuxième nom** pour le **même** tableau. Pour une vraie copie, on écrit `b = a.copy()`.

PYTHON

```python
a = [1, 2, 3]
b = a
c = a.copy()
b[0] = 99
print(a, c)
```

CE QUI S'AFFICHE

```
[99, 2, 3] [1, 2, 3]
```

Modifier `b` a modifié `a`. La copie `c`, elle, n'a pas bougé. C'est comme un raccourci sur le bureau : il ouvre le même fichier, pas une copie.

En pseudo-code, on copie un tableau **élément par élément**. On part d'un tableau vide, et on y ajoute chaque valeur :

PSEUDO-CODE

```
copie ← []
Pour chaque x dans a Faire
    ajouter(copie, x)
Fin de Pour
copie[0] ← 99
Écrire a, copie
```

PYTHON

```python
a = [1, 2, 3]
copie = []
for x in a:
    copie.append(x)
copie[0] = 99
print(a, copie)
```

CE QUI S'AFFICHE

```
[1, 2, 3] [99, 2, 3]
```

On le fait ensemble — la note la plus basse

1. On a `notes ← [14, 8, 17, 11]`. Quelle valeur de départ pour `pire` ?
2. De quel indice part la boucle ?
3. Quelle condition dans la boucle ?
4. Que vaut `pire` à la fin ?

	Exemple	Pourquoi
✔	`meilleure ← notes[0]` au départ	On part d'une vraie note du tableau.
✘	`meilleure ← 0` au départ, pour des températures	Si toutes les valeurs sont négatives, le maximum trouvé est 0, qui n'est pas dans le tableau.
✘	`for i in range(len(notes) + 1):`	Le dernier indice vaut `len(notes)`, qui n'existe pas : erreur `IndexError`.

Checkpoint

1. On a `T ← [5, 3, 8]`. Que vaut `T[longueur(T) - 1]` ?
2. Après `a = [1, 2]`, `b = a`, `b.append(3)`, que vaut `a` ?

À vous ! Les exercices de la notion 5.1

★☆☆ **Appliquer**
5.1.1 à 5.1.3
page 157

★★☆ **S'entraîner**
5.1.4 à 5.1.6
page 158

★★★ **Aller plus loin**
5.1.7 à 5.1.9
page 159

Commencez par le niveau 1. Passez au niveau suivant quand les trois exercices sont réussis.

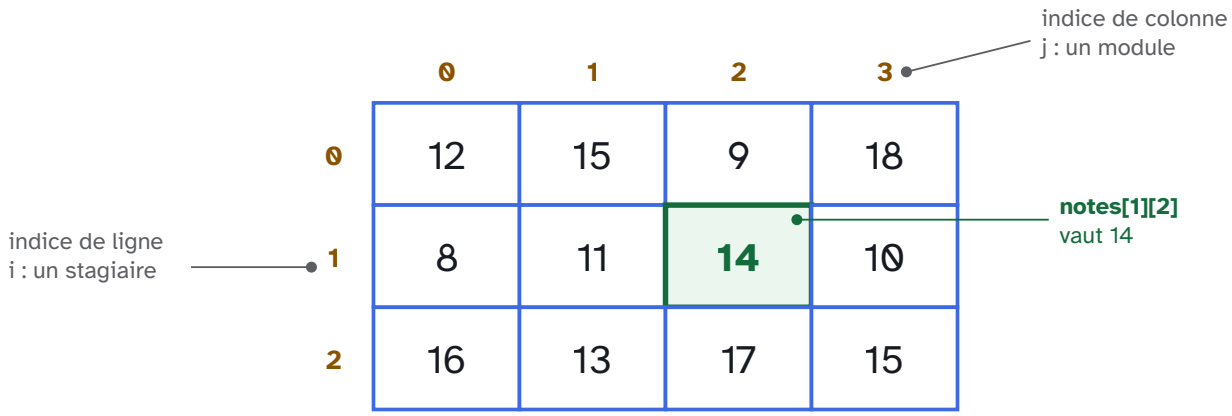
5.2 Les tableaux à deux dimensions

Sarah a 24 stagiaires et 4 modules. Un tableau par stagiaire, c'est encore 24 noms. Comment ranger toute la grille sous un seul nom ?

Un **TABLEAU À DEUX DIMENSIONS** range des valeurs en **lignes** et en **colonnes**, comme un tableur. On repère un élément par deux indices : la ligne, puis la colonne.

`M[i][j]` est l'élément de la ligne `i`, colonne `j`. En Python, c'est une liste de listes : chaque ligne est une liste.

C'est la grille d'une bataille navale. « B3 » désigne une case : une ligne, une colonne.



On donne toujours la ligne d'abord, puis la colonne.

Le tableau notes à 3 lignes et 4 colonnes : notes[1][2] est à la ligne 1, colonne 2

PYTHON

```python
notes = [[12, 15, 9, 18],
         [8, 11, 14, 10],
         [16, 13, 17, 15]]
print(notes[1][2])
print(len(notes), "lignes,", len(notes[0]), "colonnes")
```

CE QUI S'AFFICHE

```
14
3 lignes, 4 colonnes
```

Comment ça marche, pour traiter une ligne ou une colonne :

1. **Une ligne** `i` : on fixe `i`, et on fait varier `j` de 0 au nombre de colonnes moins 1.
2. **Une colonne** `j` : on fixe `j`, et on fait varier `i` de 0 au nombre de lignes moins 1.
3. **Tout le tableau** : deux boucles imbriquées, `i` dehors, `j` dedans.

PSEUDO-CODE

```
// moyenne de la colonne 2 (le module 3)
s ← 0
Pour i allant de 0 à 2 Faire
    s ← s + notes[i][2]
Fin de Pour
Écrire s / 3
```

PYTHON

```python
notes = [[12, 15, 9, 18],
         [8, 11, 14, 10],
         [16, 13, 17, 15]]
s = 0
for i in range(3):
    s = s + notes[i][2]
print(s / 3)
```

CE QUI S'AFFICHE

```
13.333333333333334
```

Pour **construire** un tableau à deux dimensions, on utilise aussi deux boucles. Pour une grille de 3 lignes et 4 colonnes remplie de 0, on fabrique chaque ligne, puis on l'ajoute à la grille :

PSEUDO-CODE

```
grille ← []
Pour i allant de 0 à 2 Faire
    ligne ← []
    Pour j allant de 0 à 3 Faire
        ajouter(ligne, 0)
    Fin de Pour
    ajouter(grille, ligne)
Fin de Pour
grille[1][2] ← 7
Écrire grille
```

PYTHON

```python
grille = []
for i in range(3):
    ligne = []
    for j in range(4):
        ligne.append(0)
    grille.append(ligne)
grille[1][2] = 7
print(grille)
```

CE QUI S'AFFICHE

```
[[0, 0, 0, 0], [0, 0, 7, 0], [0, 0, 0, 0]]
```

La ligne repart vide à chaque tour de `i` : chaque ligne est un tableau à elle.

Dans un tableau à deux dimensions, les **lignes** ou les **colonnes** peuvent avoir des types différents. Ici, ce sont les colonnes : chaque ligne décrit un stagiaire, avec son nom, son âge, s'il est présent.

PYTHON

```python
stagiaires = [["Sarah", 22, True],
              ["Tom", 19, False]]
for ligne in stagiaires:
    print(ligne[0], "a", ligne[1], "ans")
```

CE QUI S'AFFICHE

```
Sarah a 22 ans
Tom a 19 ans
```

`for ligne in stagiaires` parcourt directement les lignes, sans indice. C'est pratique quand on n'a pas besoin du numéro.

On peut aussi ranger ces données dans l'autre sens : une ligne de noms, une ligne d'âges. Ce sont alors les lignes qui ont des types différents.

On le fait ensemble — la moyenne du stagiaire 2

1. Quelle ligne du tableau `notes` ?
2. Quel indice varie dans la boucle ?
3. Quelle instruction dans la boucle ?
4. Quel résultat ?

	Exemple	Pourquoi
✔	`notes[i][j]` : la ligne d'abord	L'ordre est toujours ligne, puis colonne.
✘	`notes[2, 1]` en Python	Python attend `notes[2][1]`, avec deux paires de crochets.
✘	Une boucle sur `i` de 0 à 3 pour parcourir une colonne	Il n'y a que 3 lignes, d'indices 0 à 2. L'indice 3 n'existe pas.

Checkpoint

1. Que vaut `notes[0][3]` ?
2. Pour additionner toute la ligne 1, quel indice reste fixe ?

À vous ! Les exercices de la notion 5.2

★☆☆ **Appliquer**
5.2.1 à 5.2.3
page 161

★★☆ **S'entraîner**
5.2.4 à 5.2.6
page 162

★★★ **Aller plus loin**
5.2.7 à 5.2.9
page 163

Commencez par le niveau 1. Passez au niveau suivant quand les trois exercices sont réussis.

5.3 Les fonctions et les procédures

Sarah a écrit trois fois le calcul de moyenne. Si elle trouve une erreur dedans, elle devra la corriger trois fois. Comment écrire un calcul une seule fois, et s'en servir partout ?

Une **FONCTION** est un morceau d'algorithme qui porte un nom, reçoit des valeurs et **renvoie** un résultat. Une **PROCÉDURE** est pareille, mais ne renvoie rien : elle fait une action, comme afficher. Les valeurs reçues s'appellent les **PARAMÈTRES**.

Une fonction, c'est une machine à café. On met une capsule, l'eau. Elle rend un café.

On n'a pas besoin de savoir ce qui se passe dedans pour s'en servir. C'est l'idée de **modularité** : on découpe un gros programme en petites pièces qu'on teste une par une.

Comment ça marche :

1. On **définit** la fonction une fois : son nom, ses paramètres, son bloc.
2. `Renvoyer` donne le résultat et **termine** la fonction.
3. On **appelle** la fonction par son nom, avec des valeurs : `moyenne([12, 15, 9])`.
4. Le résultat peut être rangé dans une variable, affiché, ou utilisé dans un calcul.

PSEUDO-CODE

```
Fonction moyenne(T)
    s ← 0
    Pour i allant de 0 à longueur(T) -
1 Faire
        s ← s + T[i]
    Fin de Pour
    Renvoyer s / longueur(T)
Fin de Fonction

Procédure afficher_bilan(nom, T)
    Écrire nom, ":", moyenne(T)
Fin de Procédure

afficher_bilan("Sarah", [12, 15, 9, 18])
```

PYTHON

```python
def moyenne(T):
    s = 0
    for i in range(len(T)):
        s = s + T[i]
    return s / len(T)

def afficher_bilan(nom, T):
    print(nom, ":", moyenne(T))

afficher_bilan("Sarah", [12, 15, 9, 18])
```

CE QUI S'AFFICHE

```
Sarah : 13.5
```

La procédure appelle la fonction : on assemble les pièces. Une fonction peut aussi renvoyer **plusieurs valeurs** : `Renvoyer (s, p)`. En Python, on écrit `return s, p`, et on récupère les deux résultats avec deux variables, dans le même ordre :

PYTHON

```python
def somme_et_produit(T):
    s = 0
    p = 1
    for x in T:
        s = s + x
        p = p * x
    return s, p

total, produit = somme_et_produit([2, 3, 4])
print(total, produit)
```

CE QUI S'AFFICHE

```
9 24
```

En pseudo-code, on écrit les deux variables entre parenthèses, à gauche de la flèche : `(total, produit) ← somme_et_produit([2, 3, 4])`.

`Renvoyer` peut aussi se trouver **dans une boucle**. Il arrête alors la fonction dès qu'on a trouvé ce qu'on cherche :

PSEUDO-CODE

```
Fonction contient(T, v)
    Pour i allant de 0 à longueur(T) -
1 Faire
        Si T[i] = v Alors
            Renvoyer Vrai
        Fin de Si
    Fin de Pour
    Renvoyer Faux
Fin de Fonction
```

PYTHON

```python
def contient(T, v):
    for i in range(len(T)):
        if T[i] == v:
            return True
    return False

print(contient([4, 8, 15], 8),
contient([4, 8, 15], 3))
```

CE QUI S'AFFICHE

```
True False
```

Avec 8, la fonction s'arrête au 2e tour : la boucle ne va pas plus loin. Avec 3, la boucle va jusqu'au bout sans trouver, et on arrive à `Renvoyer Faux`.

On le fait ensemble — la fonction est_pair

1. Quel paramètre reçoit la fonction ?
2. Que renvoie-t-elle ?
3. Quel est son bloc ?
4. Que renvoie `est_pair(7)` ?

	Exemple	Pourquoi
✔	`m ← moyenne(notes)` puis `Écrire m`	La fonction renvoie, le programme principal décide quoi en faire.
✘	`Écrire s / longueur(T)` à la place de `Renvoyer` dans une fonction	La fonction affiche mais ne rend rien : on ne peut pas réutiliser le résultat.
✘	Écrire des lignes après `Renvoyer` dans le même bloc	`Renvoyer` termine la fonction : ces lignes ne s'exécutent jamais.

Checkpoint

1. Quelle est la différence entre une fonction et une procédure ?
2. Que renvoie `moyenne([10, 20])` ?

À vous ! Les exercices de la notion 5.3

★☆☆ **Appliquer**
5.3.1 à 5.3.3
page 165

★★☆ **S'entraîner**
5.3.4 à 5.3.6
page 166

★★★ **Aller plus loin**
5.3.7 à 5.3.9
page 167

Commencez par le niveau 1. Passez au niveau suivant quand les trois exercices sont réussis.

5.4 Paramètres et variables locales

Dans la fonction `moyenne`, Sarah utilise une variable `s`. Son programme principal a aussi une variable `s`.

Vont-elles se mélanger ? Et une fonction peut-elle modifier le tableau qu'on lui donne ?

Une **VARIABLE LOCALE** est créée dans une fonction : elle n'existe que pendant l'appel, et seulement dans la fonction. Une **VARIABLE GLOBALE** est créée dans le programme principal : elle est visible partout. Ce qui passe entre le programme et la fonction s'appelle l'**INTERFACE** : les paramètres et le résultat.

Une variable locale, c'est le brouillon d'un exercice. Vous l'utilisez pour calculer, puis vous le jetez. Seul le résultat, recopié au propre, compte.

Comment passent les données :

Sens du transfert	Ce qui se passe	Exemple
Entrée seule	la fonction reçoit une valeur et ne la modifie pas	`moyenne(notes)` lit les notes
Sortie seule	le paramètre reçoit un résultat, sans que la procédure lise ce qu'il contenait	`remplir(T, n)` met n zéros dans un tableau vide `T`
Entrée et sortie	la fonction reçoit un tableau et le modifie	une procédure qui ajoute 1 point à chaque note

En Python, une valeur simple, comme un nombre ou un texte, ne sort jamais par un paramètre : elle sort par `return`. Seul un tableau peut sortir par un paramètre.

PYTHON

```python
def remplir(T, n):
    for i in range(n):
        T.append(0)

def bonus(T):
    for i in range(len(T)):
        T[i] = T[i] + 1

def total(T):
    s = 0
    for x in T:
        s = s + x
    return s

zeros = []
remplir(zeros, 3)
notes = [12, 15, 9]
s = 100
bonus(notes)
print(zeros)
print(notes, total(notes), s)
```

CE QUI S'AFFICHE

```
[0, 0, 0]
[13, 16, 10] 39 100
```

La procédure `remplir` n'a jamais lu `zeros` : elle l'a seulement rempli. C'est un transfert en sortie seule.

La procédure `bonus` a modifié le tableau `notes` : c'est un transfert en entrée et sortie. La variable `s` de `total` est locale : le `s` global vaut toujours 100.

Attention

En Python, une fonction peut modifier une variable globale, avec le mot `global`. C'est à éviter :

PYTHON

```python
compteur = 0

def ajouter_un():
    global compteur
    compteur = compteur + 1

ajouter_un()
ajouter_un()
print(compteur)
```

CE QUI S'AFFICHE

```
2
```

Rien dans l'appel `ajouter_un()` ne montre que `compteur` change. Ce genre de bug est difficile à trouver. Bonne habitude : tout ce dont la fonction a besoin entre par ses paramètres, et tout ce qu'elle produit sort par `Renvoyer`.

On le fait ensemble — qui voit quoi ?

1. Dans `def f(): y = 5`, puis `print(y)` hors de la fonction : que se passe-t-il ?
2. On a `x = 10` dans le programme principal, puis `def g(): return x + 1`. Que renvoie `g()` ?
3. `moyenne(notes)` modifie-t-elle `notes` ?
4. Pourquoi vaut-il mieux passer `x` en paramètre plutôt que le lire en global ?

	Exemple	Pourquoi
✔	`def aire(l, L): return l * L`	Tout entre par les paramètres, tout sort par `return`.
✘	Utiliser hors de la fonction une variable créée dedans	Elle est locale : elle disparaît à la fin de l'appel.
✘	Croire que `bonus(notes)` laisse `notes` intact	Un tableau passé en paramètre peut être modifié par la fonction.

Checkpoint

1. Où existe une variable locale ?
2. Une procédure qui trie le tableau qu'on lui donne : quel transfert ?

À vous ! Les exercices de la notion 5.4

★☆☆ **Appliquer**
5.4.1 à 5.4.3
page 169

★★☆ **S'entraîner**
5.4.4 à 5.4.6
page 170

★★★ **Aller plus loin**
5.4.7 à 5.4.9
page 172

Commencez par le niveau 1. Passez au niveau suivant quand les trois exercices sont réussis.

Ce qu'on retient du chapitre 5

Si vous ne devez retenir que trois choses

1. Un tableau se parcourt avec une boucle sur les indices, de 0 à longueur - 1. En deux dimensions : `M[ligne][colonne]`.
2. Une fonction renvoie un résultat avec `Renvoyer`. Une procédure agit sans rien renvoyer.
3. Faites entrer les données par les paramètres et sortir le résultat par `Renvoyer`. Pour copier un tableau en Python, utilisez `.copy()`.

Sarah a maintenant un seul tableau pour toutes les notes, et une seule fonction `moyenne`. Elle l'appelle pour chaque stagiaire et pour chaque module. Quand elle corrige la fonction, tout le programme est corrigé d'un coup.

Les réponses du cours

Cherchez d'abord seul. Vérifiez ensuite ici.

Chapitre 1 — Écrire un algorithme

La question de départ

Une recette de cuisine dit : « Ajoutez le sel. Faites cuire 10 minutes. Goûtez. » Qu'est-ce qui se passe si on lit ces lignes dans le désordre ? → **Le plat est raté. On goûte avant de cuire, ou on sale après. L'ordre des instructions compte. Un algorithme, c'est pareil.**

1.1 Du problème au programme

On le fait ensemble — le stock de câbles

1. Le magasin a 40 câbles. On en vend 15. Quelle instruction donne le stock restant, en pseudo-code ? → `reste ← 40 - 15`

2. Quelle instruction affiche le stock ? → `Écrire "Stock :", reste`

3. Qu'affiche le programme ? → `Stock : 25`

4. Que se passe-t-il si on écrit l'affichage avant le calcul ? → **Le programme veut afficher `reste` avant qu'il existe : c'est une erreur.**

Checkpoint

1. Dans quel ordre l'ordinateur exécute-t-il les instructions ? → **De haut en bas, une par une. C'est l'exécution séquentielle.**

2. Quelle est la différence entre le pseudo-code et Python ? → **Le pseudo-code sert à écrire et lire l'algorithme. Python tourne sur la machine : on peut tester.**

1.2 Les variables et l'affectation

On le fait ensemble — échanger deux valeurs

1. On a `a ← 3` et `b ← 8`. On veut que `a` vaille 8 et `b` vaille 3. Que vaut `a` après `a ← b` ? → **8. L'ancienne valeur de `a`, 3, est perdue.**

2. Que vaut `b` après `b ← a` ? → **8 aussi. On a perdu le 3 : l'échange est raté.**

3. Comment garder le 3 ? → **Dans une troisième variable, avant de l'écraser : `temp ← a`.**

4. Quelles sont les trois lignes de l'échange ? → **`temp ← a`, puis `a ← b`, puis `b ← temp`.**

Checkpoint

1. On exécute `n ← 10`, puis `n ← n - 3`, puis `n ← n * 2`. Que vaut `n` ? → **14. On calcule 10 - 3 = 7, puis 7 × 2 = 14.**

2. Comment écrit-on `n ← n * 2` en Python ? → **`n = n * 2`, ou `n *= 2`.**

1.3 Lire et écrire

On le fait ensemble — dire bonjour

1. On veut demander le prénom. Quelle ligne Python ? → **`prenom = input("Prénom ? ")`**

2. Faut-il `int(...)` autour ? → **Non. Un prénom est du texte, on le garde tel quel.**

3. Quelle ligne affiche « Bonjour Inès » ? → **`print("Bonjour", prenom)`**

4. Et en pseudo-code ? → **`Lire prenom`, puis `Écrire "Bonjour", prenom`**

Checkpoint

1. Que fait l'instruction `Lire x` ? → **Elle attend une valeur tapée au clavier et la range dans `x`.**

2. Pourquoi écrit-on `float(input(...))` pour un prix ? → **Parce qu'`input` renvoie du texte, et qu'un prix peut avoir une virgule. `float` le change en nombre réel.**

1.4 La trace d'un algorithme

On le fait ensemble — le panier

1. L'algorithme est : `prix ← 20`, `qte ← 3`, `total ← prix * qte`, `total ← total - 5`. Combien de colonnes de variables ? → **Trois : `prix`, `qte`, `total`.**

2. Que vaut `total` à l'étape 3 ? → **60.**

3. Que vaut `total` à l'étape 4 ? → **55. On lit 60, on enlève 5.**

4. Que vaut `prix` à la fin ? → **20. Aucune ligne ne l'a changé.**

Checkpoint

1. Faites la trace de `a ← 5`, `b ← a * 2`, `a ← b - a`. Que valent `a` et `b` ? → **`a` vaut 5 et `b` vaut 10. On a 10 - 5 = 5.**

2. À quoi sert une trace ? → **À suivre la valeur des variables étape par étape, pour comprendre l'algorithme ou trouver une erreur.**

Chapitre 2 — Les types de données

La question de départ

Le numéro de téléphone 06 12 34 56 78 : faut-il le ranger comme un nombre ou comme un texte ? → **Comme un texte. On ne fait jamais de calcul dessus, et un nombre perdrait le 0 du début.**

2.1 Les types simples

On le fait ensemble — la fiche de la salle de sport

1. Quel type pour l'âge du client ? → **Entier naturel. Un âge n'est jamais négatif et n'a pas de virgule.**

2. Quel type pour son poids, 72,3 kg ? → **Réel.**

3. Quel type pour « a payé l'abonnement » ? → **Booléen. La réponse est oui ou non.**

4. Quel type pour l'écart de poids depuis l'inscription, par exemple -4 kg ? → **Entier relatif. Il peut être négatif.**

Checkpoint

1. Quel type pour le nombre de fichiers dans un dossier ? → **Entier naturel.**

2. Quel est le type de `3 > 5` ? → **Booléen. Une comparaison vaut Vrai ou Faux, ici Faux.**

2.2 Les limites de la machine

Instant histoire — 1996 : la fusée détruite par un nombre trop grand

Qu'est-ce qui aurait évité la catastrophe ? → **Vérifier que la valeur tenait dans le domaine du type avant de la copier, ou choisir un type plus grand.**

On le fait ensemble — un compteur sur 8 bits

1. Combien de valeurs différentes sur 8 bits ? → $2^8 = 256$ **valeurs.**

2. Quelle est la plus grande valeur d'un entier naturel sur 8 bits ? → **255, car on commence à 0.**

3. Un compteur sur 8 bits vaut 255. On ajoute 1. Que se passe-t-il ? → **Dépassement de capacité : il repasse à 0, comme le compteur kilométrique.**

4. Faut-il 8 ou 16 bits pour compter 1 000 visiteurs ? → **16 bits. 1 000 dépasse 255.**

Checkpoint

1. Quelle est la plus grande valeur d'un entier naturel sur 10 bits ? → $2^{10} - 1 = 1023$.

2. Pourquoi `0.1 + 0.2` n'affiche-t-il pas 0.3 ? → **Parce que 0,1 et 0,2 sont rangés en binaire de façon approchée. Les deux petites erreurs s'ajoutent.**

2.3 Les chaînes de caractères

On le fait ensemble — une adresse mail

1. On a `mail ← "ines@lycee.fr"`. Combien de caractères ? → **13.**

2. Quel est l'indice du premier caractère ? → **0.**

3. Que vaut `mail[4]` ? → `@` **. On compte i=0, n=1, e=2, s=3, @=4.**

4. Quel est l'indice du dernier caractère ? → **12, soit 13 - 1.**

Checkpoint

1. Que vaut `"Python"[2]` ? → `t` **. P=0, y=1, t=2.**

2. Quelle est la longueur de `"BTS SIO"` ? → **7. L'espace est un caractère.**

2.4 Changer de type : le transtypage

On le fait ensemble — un prix tapé au clavier

1. L'utilisateur tape `12.5`. Quel est le type de ce que renvoie `input` ? → **Une chaîne :** `"12.5"` **.**

2. Comment la changer en nombre ? → `float("12.5")` **, qui donne le réel 12.5.**

3. Que donne `int("12.5")` ? → **Une erreur.** `int` **n'accepte que l'écriture d'un entier.**

4. Comment afficher « Total : 25.0 » en collant du texte et un nombre ? → `"Total : " + str(25.0)` **. On change d'abord le nombre en texte.**

Checkpoint

1. Que vaut `int(7.8)` ? → **7. La partie après la virgule est coupée.**

2. Que vaut `"3" + "4"` en Python ? → `"34"` **. Ce sont deux chaînes : elles sont collées.**

Chapitre 3 — Les opérateurs

La question de départ

Une vidéo dure 200 minutes. Combien d'heures et de minutes ? Comment avez-vous fait le calcul ? → **3 h 20 min. On cherche combien de fois 60 tient dans 200 : 3 fois. Il reste 200 - 180 = 20 minutes. C'est une division euclidienne.**

3.1 Les opérateurs numériques

On le fait ensemble — 100 000 secondes

1. Combien d'heures entières dans 100 000 secondes ? → `100000 div 3600` **= 27.**

2. Combien de secondes restent après ces heures ? → `100000 mod 3600` **= 2 800.**

3. Combien de minutes dans ce reste ? → `2800 div 60` **= 46.**

4. Et combien de secondes ? → `2800 mod 60` **= 40. Résultat : 27 h 46 min 40 s.**

Checkpoint

1. Que valent `23 div 4` et `23 mod 4` ? → **5 et 3, car 23 = 4 × 5 + 3.**

2. Comment écrire $\frac{x-1}{x+1}$ en ligne ? → `(x - 1) / (x + 1)`

3.2 Les fonctions mathématiques usuelles

On le fait ensemble — le dé

1. Quelle instruction Python simule un dé à 6 faces ? → `random.randint(1, 6)`

2. Les bornes 1 et 6 peuvent-elles sortir ? → **Oui.** `randint` **inclut les deux bornes.**

3. Quelle ligne faut-il avant ? → `import random`

4. Comment tirer une note entière entre 0 et 20 ? → `random.randint(0, 20)`

Checkpoint

1. Que donne `math.floor(-2.5)` ? → **-3. L'arrondi par défaut va vers le plus petit entier.**

2. Que donne `round(12.3456, 1)` ? → **12.3**

3.3 Comparer et combiner des conditions

On le fait ensemble — la majorité

1. Règle : « l'utilisateur a entre 18 et 25 ans inclus ». Quelles sont les deux petites conditions ? → **`age ≥ 18` et `age ≤ 25`.**

2. Quel mot les relie ? → **ET. Les deux doivent être vraies.**

3. Quelle expression en Python ? → **`age >= 18 and age <= 25`**

4. Que vaut-elle pour `age = 30` ? → **`False`. 30 ≥ 18 est Vrai, mais 30 ≤ 25 est Faux.**

Checkpoint

1. Que vaut `(3 > 2) and (5 < 1)` ? → **`False`. La seconde condition est fausse.**

2. Traduisez « le stock est vide ou le produit n'est plus vendu » avec `stock` et `vendu`. → **`stock == 0 or not vendu`**

3.4 Les opérateurs bit à bit

On le fait ensemble — tester le bit 1 de 1011

1. Quel masque pour le bit 1 ? → **`0010`. Un seul 1, au rang 1.**

2. Que vaut `1011 ET 0010`, rang par rang ? → **`0010`**

3. Le bit 1 vaut-il 1 ? → **Oui, car le résultat n'est pas nul.**

4. Quel masque et quel opérateur pour mettre ce bit à 0 ? → **ET avec l'inverse du masque, `1101` : on obtient `1001`.**

Checkpoint

1. Que vaut `0b1000 | 0b0001` ? → **`0b1001`**

2. Avec quel masque change-t-on l'état du bit 3 ? → **`0b1000`, avec l'opérateur OUX `^`.**

3.5 Les opérateurs de chaînes

On le fait ensemble — l'extension d'un fichier

1. On a `nom ← "photo_vacances.jpg"`. Quelle est la longueur ? → **18.**

2. L'extension fait 3 caractères. Quels indices prend-elle ? → **15, 16 et 17.**

3. Quelle instruction Python la donne ? → **`nom[-3:]`, ou `nom[15:18]`.**

4. Comment savoir si le nom contient « vacances » ? → **`nom.find("vacances")` est différent de -1. Ici, il vaut 6.**

Checkpoint

1. Que vaut `"192.168.1.10"[:3]` ? → **`"192"`**

2. Que vaut `"bonjour".find("j")` ? → **3. b=0, o=1, n=2, j=3.**

Chapitre 4 — Les structures de contrôle

La question de départ

Vous devez écrire « Bonjour » 1 000 fois. Allez-vous écrire 1 000 lignes ? → **Non. On dit à l'ordinateur de répéter la même ligne 1 000 fois. C'est une boucle.**

4.1 Faire un choix : Si... Alors... Sinon

On le fait ensemble — avec un poste de 5 ans

1. On remplace `age ← 7` par `age ← 5`. La condition `age ≥ 6` est-elle vraie ? → **Non. 5 n'est pas supérieur ou égal à 6.**

2. Que teste-t-on ensuite ? → **`age ≥ 4`. C'est Vrai.**

3. Qu'affiche le programme ? → **`À surveiller`**

4. Le bloc `Sinon` est-il exécuté ? → **Non. Un seul bloc est exécuté : le premier dont la condition est vraie.**

Checkpoint

1. On a `x ← -3`. Qu'affiche `Si x > 0 Alors Écrire "positif" Sinon Écrire "négatif ou nul" Fin de Si` ? → **`négatif ou nul`**

2. Combien de blocs sont exécutés dans un `Si … Sinon si … Sinon` ? → **Un seul.**

4.2 Répéter un nombre connu de fois : la boucle Pour

On le fait ensemble — compter les « 1 » d'un code

1. On a `code ← "1011001"`. Combien de tours de boucle pour lire tous les caractères ? → **7, un par caractère.**

2. De quel indice à quel indice va `i` ? → **De 0 à 6.**

3. Que fait-on à chaque tour ? → **`Si code[i] = "1" Alors compt ← compt + 1 Fin de Si`**

4. Quelle valeur faut-il donner à `compt` avant la boucle ? → **0. On n'a encore rien compté.**

5. Que vaut `compt` à la fin ? → **4.**

Checkpoint

1. Combien de fois s'exécute le bloc de `Pour i allant de 3 à 7 Faire` ? → **5 fois : 3, 4, 5, 6, 7.**

2. Quelles valeurs donne `range(4)` ? → **0, 1, 2, 3.**

4.3 Répéter tant que : les boucles Tant que et Répéter

On le fait ensemble — chercher une lettre

1. On cherche le premier `"S"` dans `"BTSSIO"`. On part de `k ← 0`. Quelle est la condition pour continuer ? → **`mot[k] ≠ "S"` : on n'a pas encore trouvé.**

2. Que met-on dans le bloc ? → **`k ← k + 1`, pour passer au caractère suivant.**

3. Que vaut `k` à la sortie ? → **2.**

4. Que se passe-t-il si on cherche `"A"`, absent du mot ? → **`k` dépasse 5 et `mot[k]` n'existe plus : erreur. Il faut aussi tester `k < longueur(mot)`.**

Checkpoint

1. Qu'affiche `n ← 100`, `Tant que n > 10 Faire n ← n div 2 Fin de Tant que`, `Écrire n` ? → **6. On passe par 50, 25, 12, puis 6.**

2. Quelle boucle fait toujours au moins un tour ? → **Répéter … Jusqu'à.**

4.4 Construire une boucle de calcul

On le fait ensemble — la moyenne de 5 notes

1. On lit 5 notes au clavier. Quelle boucle ? → **Pour, de 1 à 5 : on connaît le nombre de tours.**

2. Quelle préparation ? → **`total ← 0`**

3. Que fait-on à chaque tour ? → **`Lire note`, puis `total ← total + note`**

4. Quelle mise en forme finale ? → **`moyenne ← total / 5`, puis `Écrire arrondi(moyenne, 2)`. Elle se fait après la boucle, une seule fois.**

Checkpoint

1. Quelle valeur de départ pour calculer $\prod_{i=1}^{5} i$? → **1. Le résultat est 120.**

2. Traduisez $\sum_{k=1}^{10} 2k$ en boucle Python. → **`s = 0`, puis `for k in range(1, 11):` et `s = s + 2 * k`. Le résultat est 110.**

4.5 Les structures imbriquées

On le fait ensemble — combien de tours ?

1. `Pour i allant de 1 à 4` contient `Pour j allant de 1 à 3`. Combien de fois s'exécute le bloc intérieur ? → **4 × 3 = 12 fois.**

2. Que vaut `j` au tout premier passage ? → **1.**

3. Quand `i` passe de 1 à 2, que vaut `j` ? → `j` **repart à 1. La boucle intérieure recommence.**

4. Où remettre `s ← 0` dans le produit de matrices ? → **Juste avant la boucle sur k, pour chaque case (i, j).**

Checkpoint

1. Combien de lignes affiche le programme des étoiles si on remplace 3 par 5 ? → **5 lignes, de 1 à 5 étoiles.**

2. Dans une boucle Pour de 1 à 10, on écrit `Si i mod 2 = 0 Alors compt ← compt + 1`. Que vaut `compt` à la fin, s'il partait de 0 ? → **5. On compte les nombres pairs : 2, 4, 6, 8, 10.**

Chapitre 5 — Les tableaux et les fonctions

La question de départ

Dans un tableur, comment retrouvez-vous la note du 3e stagiaire au 2e module ? → **Par sa ligne et sa colonne : ligne du stagiaire 3, colonne du module 2. Un tableau en programmation marche pareil.**

5.1 Les tableaux à une dimension

On le fait ensemble — la note la plus basse

1. On a `notes ← [14, 8, 17, 11]`. Quelle valeur de départ pour `pire` ? → `notes[0]`**, soit 14.**

2. De quel indice part la boucle ? → **De 1. L'élément 0 est déjà dans** `pire`**.**

3. Quelle condition dans la boucle ? → `Si notes[i] < pire Alors pire ← notes[i]`

4. Que vaut `pire` à la fin ? → **8.**

Checkpoint

1. On a `T ← [5, 3, 8]`. Que vaut `T[longueur(T) - 1]` ? → **8, le dernier élément.**

2. Après `a = [1, 2]`, `b = a`, `b.append(3)`, que vaut `a` ? → **`[1, 2, 3]`. `a` et `b` sont le même tableau.**

5.2 Les tableaux à deux dimensions

On le fait ensemble — la moyenne du stagiaire 2

1. Quelle ligne du tableau `notes` ? → **La ligne d'indice 2 : `[16, 13, 17, 15]`.**

2. Quel indice varie dans la boucle ? → **`j`, la colonne, de `0` à 3.**

3. Quelle instruction dans la boucle ? → **`s ← s + notes[2][j]`**

4. Quel résultat ? → **61 / 4 = 15,25.**

Checkpoint

1. Que vaut `notes[0][3]` ? → **18.**

2. Pour additionner toute la ligne 1, quel indice reste fixe ? → **L'indice de ligne, `i = 1`. On fait varier `j`.**

5.3 Les fonctions et les procédures

On le fait ensemble — la fonction est_pair

1. Quel paramètre reçoit la fonction ? → **Un entier `n`.**

2. Que renvoie-t-elle ? → **Un booléen : Vrai si `n` est pair.**

3. Quel est son bloc ? → **`Renvoyer n mod 2 = 0`**

4. Que renvoie `est_pair(7)` ? → **Faux.**

Checkpoint

1. Quelle est la différence entre une fonction et une procédure ? → **La fonction renvoie un résultat. La procédure fait une action et ne renvoie rien.**

2. Que renvoie `moyenne([10, 20])` ? → **15.0**

5.4 Paramètres et variables locales

On le fait ensemble — qui voit quoi ?

1. Dans `def f(): y = 5`, puis `print(y)` hors de la fonction : que se passe-t-il ? → **Erreur. `y` est locale à `f`, elle n'existe pas dehors.**

2. On a `x = 10` dans le programme principal, puis `def g(): return x + 1`. Que renvoie `g()` ? → **11. La fonction peut lire une variable globale.**

3. `moyenne(notes)` modifie-t-elle `notes` ? → **Non. Elle lit seulement : transfert en entrée seule.**

4. Pourquoi vaut-il mieux passer `x` en paramètre plutôt que le lire en global ? → **La fonction devient indépendante : on la teste seule et on la réutilise ailleurs.**

Checkpoint

1. Où existe une variable locale ? → **Seulement dans la fonction, pendant l'appel.**

2. Une procédure qui trie le tableau qu'on lui donne : quel transfert ? → **Entrée et sortie. Elle reçoit le tableau et le modifie.**

EXERCICES

S'entraîner

1.1 — Du problème au programme

‹ Revoir le cours 1.1

★☆☆ **Niveau 1 — Appliquer**

Exercice 1.1.1 — Remettre dans l'ordre ★☆☆

Un magasin vend un casque 50 € hors taxes. On veut afficher son prix toutes taxes comprises. La TVA est de 20 % du prix hors taxes.

Les instructions ont été mélangées :

PSEUDO-CODE

```
TTC ← HT + TVA
Écrire "Prix TTC :", TTC
HT ← 50
TVA ← HT * 0.2
```

1. Remettez les quatre instructions dans le bon ordre.
2. Qu'affiche l'algorithme ?

Exercice 1.1.2 — Pseudo-code ou Python ? ★☆☆

Voici six lignes.

```
a. x ← 3
b. print(x)
c. Écrire x
d. x = 3
e. total = prix * qte
f. total ← prix * qte
```

1. Pour chaque ligne, dites si elle est écrite en pseudo-code ou en Python.
2. Associez chaque ligne de pseudo-code à la ligne Python qui dit la même chose.

Exercice 1.1.3 — Une vraie instruction ? ★☆☆

Un élève a écrit ces lignes pour calculer la surface d'une salle de 8 m sur 6 m.

```
PSEUDO-CODE
longueur ← 8
largeur ← 6
calculer la surface
Écrire surface
```

1. Quelle ligne n'est pas une instruction que l'ordinateur peut exécuter ? Pourquoi ?
2. Remplacez-la par une vraie instruction.

★★☆ Niveau 2 — S'entraîner

Exercice 1.1.4 — Traduire en Python ★★☆ Sur machine

Voici un algorithme en pseudo-code.

```
PSEUDO-CODE
longueur ← 8
largeur ← 6
surface ← longueur * largeur
Écrire "Surface :", surface, "m²"
```

1. Écrivez le même algorithme en Python, ligne pour ligne.
2. Exécutez-le. Qu'affiche-t-il ?
3. Changez la largeur en 7,5. En Python, on écrit `7.5`, avec un point. Que devient l'affichage ?

```
PYTHON
longueur = 8
# à compléter
```

Exercice 1.1.5 — Écrire l'en-tête ★★☆

Voici un algorithme sans en-tête.

```
PSEUDO-CODE
HT ← 50
TVA ← HT * 0.2
TTC ← HT + TVA
Écrire "Prix TTC :", TTC
```

Écrivez son en-tête en commentaires : le nom, le rôle, les entrées et la sortie.

Exercice 1.1.6 — Le taux de pannes ★★☆ Sur machine

Inès compte 80 postes dans le lycée. Ce matin, 76 marchent. Elle veut connaître le **pourcentage de postes en panne**.

1. Écrivez l'algorithme en pseudo-code. Utilisez les variables `total`, `marche`, `pannes` et `pourcentage`.
2. Traduisez-le en Python et exécutez-le.

Pour le pourcentage, multipliez d'abord par 100, puis divisez par le total.

★★★ Niveau 3 — Aller plus loin

Exercice 1.1.7 — Le programme qui plante ★★★

Ce programme doit afficher le prix d'un abonnement de 3 mois à 15 € par mois. Il s'arrête sur une erreur.

```python
print("Total :", total)
mois = 3
prix = 15
total = mois * prix
```

1. Quelle ligne provoque l'erreur ? Pourquoi ?
2. Corrigez le programme.
3. Ajoutez deux lignes : l'une calcule le prix pour 12 mois, avec une remise de 20 € sur le total ; l'autre l'affiche.
4. Le prix passe à 18 € par mois. Quelle seule ligne changez-vous ? Qu'affiche alors le programme ? Pourquoi les deux totaux suivent-ils seuls ?

Exercice 1.1.8 — L'espace libre du disque ★★★ Sur machine

Un serveur a un disque de 500 Go. 365 Go sont occupés.

Écrivez un algorithme, en pseudo-code puis en Python, qui affiche :

- l'espace libre, en Go ;
- le pourcentage d'espace libre.

Ajoutez son en-tête en commentaires.

Exercice 1.1.9 — Quelles lignes peut-on échanger ? ★★★

```
PSEUDO-CODE
a ← 4
b ← 5
c ← a * b
d ← a + b
Écrire c, d
```

1. Qu'affiche l'algorithme ?
2. Peut-on échanger `a ← 4` et `b ← 5` sans changer le résultat ? Et `c ← a * b` et `d ← a + b` ?
3. Peut-on échanger `b ← 5` et `c ← a * b` ? Pourquoi ?
4. Donnez une règle : quand deux lignes peuvent-elles changer de place ?

1.2 — Les variables et l'affectation

‹ Revoir le cours 1.2

★☆☆ Niveau 1 — Appliquer

Exercice 1.2.1 — La valeur finale ★☆☆

```
PSEUDO-CODE
p ← 4
p ← p + 6
p ← p * 3
```

Que vaut `p` après chaque ligne ?

Exercice 1.2.2 — Affectation juste ou fausse ? ★☆☆

Pour chaque ligne, dites si l'affectation est correcte. Si elle est fausse, dites pourquoi.

```
PSEUDO-CODE
x ← 5
5 ← x
x ← x + 1
x + 1 ← x
total ← prix * qte
```

Exercice 1.2.3 — Le stock de câbles ★☆☆ Sur machine

Un magasin a 40 câbles. Il en vend 15, puis il en reçoit 30.

PSEUDO-CODE

```
stock ← 40
stock ← stock - 15
stock ← stock + 30
Écrire "Stock :", stock
```

1. Que vaut `stock` à la fin ?
2. Écrivez le programme en Python et vérifiez.

★★☆ Niveau 2 — S'entraîner

Exercice 1.2.4 — Échanger deux valeurs ★★☆ Sur machine

On a `a ← 3` et `b ← 8`. On veut que `a` vaille 8 et `b` vaille 3.

PSEUDO-CODE

```
a ← 3
b ← 8
temp ← . . .
a ← . . .
b ← . . .
```

1. Complétez les trois lignes.
2. Vérifiez en Python.

Exercice 1.2.5 — Une copie, pas un lien ★★☆

PSEUDO-CODE

```
x ← 7
y ← x
x ← x + 5
```

1. Que vaut `x` à la fin ?
2. Que vaut `y` à la fin ?
3. Un camarade dit : « `y` vaut 12, car `y` est égal à `x` ». A-t-il raison ?

Exercice 1.2.6 — Le compteur de visiteurs ★★☆ Sur machine

Une médiathèque compte ses visiteurs. Le compteur part de 0. Il entre 12 personnes, puis 7, puis 3 personnes sortent.

1. Écrivez l'algorithme avec une seule variable `visiteurs` et des mises à jour.
2. Traduisez-le en Python en utilisant `+=` et `-=`.

★★★ Niveau 3 — Aller plus loin

Exercice 1.2.7 — Faire tourner trois valeurs ★★★ Sur machine

On a `a ← 1`, `b ← 2` et `c ← 3`. On veut obtenir `a` = 2, `b` = 3 et `c` = 1 : chaque valeur avance d'une place.

1. Écrivez les affectations, avec une variable `temp`.
2. Vérifiez en Python.

Exercice 1.2.8 — Le solde du compte ★★★ Sur machine

Le compte d'une petite association contient 200 €. Elle paie une facture de 45,50 €, puis reçoit un don de 120 €. En Python, 45,50 s'écrit `45.50`, avec un point.

1. Écrivez l'algorithme avec une seule variable `solde`.
2. Traduisez-le en Python avec `-=` et `+=`.
3. Ajoutez une ligne qui retire 15 € de frais. Quel solde s'affiche ?

Exercice 1.2.9 — Doubler encore et encore ★★★

PSEUDO-CODE

```
x ← 1
x ← x * 2
x ← x * 2
x ← x * 2
x ← x * 2
x ← x * 2
```

1. Que vaut `x` à la fin ?
2. Combien de lignes `x ← x * 2` faut-il, après `x ← 1`, pour que `x` vaille 1 024 ?
3. Quel lien voyez-vous avec les puissances de 2 ?
4. On part de `x ← 2` et on répète `x ← x * x`. Après combien de lignes `x` dépasse-t-il 1 000 ? Combien de lignes `x ← x * 2` faudrait-il, en partant aussi de 2 ? Quelle méthode est la plus rapide ?

1.3 — Lire et écrire

‹ Revoir le cours 1.3

★★★ **Niveau 1 — Appliquer**

Exercice 1.3.1 — Le bon type à la lecture ★★★

On lit au clavier les valeurs suivantes. Pour chacune, dites s'il faut écrire `int(input(...))`, `float(input(...))` ou seulement `input(...)`.

1. L'âge d'un client.
2. Le prix d'un article, par exemple 12,99 €.
3. Le prénom d'un client.
4. Le nombre d'élèves d'une classe.
5. La température d'une salle, par exemple 21,5 °C.

Exercice 1.3.2 — Dire bonjour ★★★ Sur machine

Écrivez un programme qui demande le prénom de l'utilisateur, puis affiche « Bonjour » suivi du prénom.

```python
prenom = input("Prénom ? ")
# à compléter
```

Exercice 1.3.3 — L'âge de l'an prochain ★★★

Ce programme doit afficher l'âge de l'utilisateur l'an prochain. Il s'arrête sur une erreur.

```python
age = input("Âge ? ")
print("L'an prochain :", age + 1)
```

1. Pourquoi ce programme plante-t-il ?
2. Corrigez-le.

★★☆ **Niveau 2 — S'entraîner**

Exercice 1.3.4 — Celsius vers Fahrenheit ★★☆ Sur machine

Un technicien lit la température d'une salle serveur en degrés Celsius. Le fabricant de la climatisation donne ses seuils en degrés Fahrenheit. La formule est :

$$F = C \times \frac{9}{5} + 32$$

On lit : F égale C fois 9 cinquièmes, plus 32.

1. Écrivez la formule sur une ligne, avec les variables `c` et `f`. Une fraction devient une division : $\frac{9}{5}$ s'écrit `9 / 5`.
2. Complétez le programme : il lit une température en Celsius et affiche la température en Fahrenheit.
3. Testez-le avec 25 °C. Vous devez obtenir 77.0 °F : Python écrit `77.0`, car `c` est un réel.

PYTHON

```
c = float(input("Température en °C ? "))
f = 0      # à compléter
print(c, "°C =", f, "°F")
```

Exercice 1.3.5 — Le prix avec remise ★★☆ Sur machine

Une boutique en ligne calcule le prix d'une commande. Le client tape le prix d'un article, la quantité et le taux de remise en pourcentage. Le programme doit :

1. calculer le montant brut : prix × quantité ;
2. calculer la remise : montant brut × taux / 100 ;
3. calculer le montant net : montant brut – remise ;
4. afficher le montant net.

Écrivez l'algorithme en pseudo-code, puis en Python. Testez-le avec un prix de 12,5 €, 4 articles et 10 % de remise.

Exercice 1.3.6 — Le périmètre d'une salle ★★☆

On veut le périmètre d'une salle rectangulaire. L'utilisateur tape la longueur et la largeur, en mètres. Le périmètre vaut 2 × (longueur + largeur).

1. Écrivez l'algorithme en pseudo-code : deux lectures, un calcul, un affichage.
2. Écrivez-le en Python. Testez avec 8 m et 6 m.

★★★ **Niveau 3 — Aller plus loin**

Exercice 1.3.7 — Passer de 3 notes à 4 notes ★★★

Cet algorithme calcule la moyenne de 3 notes.

```
PSEUDO-CODE
Lire n1
Lire n2
Lire n3
somme ← n1 + n2 + n3
moyenne ← somme / 3
Écrire "Moyenne :", moyenne
```

1. Qu'affiche-t-il pour les notes 12, 15 et 9 ?
2. Modifiez-le pour qu'il calcule la moyenne de 4 notes.
3. Qu'affiche votre version pour 12, 15, 9 et 14 ?

Exercice 1.3.8 — Heures et minutes en minutes ★★★ Sur machine

Une vidéo dure 2 h 35 min. On veut un programme qui lit les heures et les minutes, puis affiche :

- la durée totale en minutes ;
- la durée totale en secondes.

1. Écrivez l'algorithme en pseudo-code.
2. Écrivez-le en Python et testez avec 2 h 35 min.

Exercice 1.3.9 — La facture du magasin ★★★ Sur machine

Un magasin vend trois articles : une clé USB à 8 €, un câble à 5 €, une souris à 12 €. Le client tape la quantité de chaque article.

Le programme affiche le total hors taxes, la TVA à 20 % et le total TTC. Calculez la TVA avec `HT * 20 / 100`.

1. Écrivez l'algorithme en pseudo-code, **avec son en-tête** : nom, rôle, entrées, sortie. Ajoutez un commentaire `//` pour le rôle de chaque variable calculée.
2. Écrivez-le en Python, avec le même en-tête en commentaires `#`. Testez avec 2 clés, 3 câbles et 1 souris.

1.4 — La trace d'un algorithme

‹ Revoir le cours 1.4

★☆☆ **Niveau 1 — Appliquer**

Exercice 1.4.1 — La trace d'un calcul ★☆☆

PSEUDO-CODE

```
x ← 7
y ← x - 2
x ← x + y
y ← x * 2
```

1. Recopiez le tableau et faites la trace.
2. Donnez les valeurs de `x` et de `y` à la fin.

Étape	Instruction	x	y
1	`x ← 7`		
2	`y ← x - 2`		
3	`x ← x + y`		
4	`y ← x * 2`		

Exercice 1.4.2 — Trois variables ★☆☆

PSEUDO-CODE

```
a ← 2
b ← a + 4
c ← a * b
a ← c - b
```

Faites la trace dans ce tableau.

Étape	Instruction	a	b	c
1	`a ← 2`			
2	`b ← a + 4`			
3	`c ← a * b`			
4	`a ← c - b`			

Exercice 1.4.3 — Une seule variable ★☆☆

```
PSEUDO-CODE
s ← 0
s ← s + 5
s ← s + 8
s ← s - 3
```

Faites la trace de `s`. Que vaut `s` à la fin ?

Étape	Instruction	s
1	`s ← 0`	
2	`s ← s + 5`	
3	`s ← s + 8`	
4	`s ← s - 3`	

★★☆ Niveau 2 — S'entraîner

Exercice 1.4.4 — Une trace avec une lecture ★★☆

```
PSEUDO-CODE
Lire x
y ← x * x
x ← y - x
Écrire x, y
```

1. L'utilisateur tape 6. Faites la trace.
2. Qu'affiche l'algorithme ?
3. Qu'afficherait-il si l'utilisateur tapait 3 ?

Étape	Instruction	x	y
1	`Lire x`		
2	`y ← x * x`		
3	`x ← y - x`		

Exercice 1.4.5 — La moyenne fausse ★★☆

Cet algorithme doit calculer la moyenne de deux notes, 12 et 15.

PSEUDO-CODE

```
a ← 12
b ← 15
m ← a + b / 2
Écrire m
```

1. Faites la trace. Que vaut `m` ?
2. La moyenne attendue est 13,5. D'où vient l'erreur ?
3. Corrigez l'instruction qui calcule `m`.

Exercice 1.4.6 — Trouver l'erreur dans une trace ★★☆ Sur machine

PSEUDO-CODE

```
p ← 3
q ← p * 4
p ← q - p
q ← q + p
```

Un élève a rempli cette trace :

Étape	p	q
1	3	–
2	3	12
3	9	12
4	9	15

1. À quelle étape l'élève s'est-il trompé ? Quelle erreur a-t-il faite ?
2. Écrivez le programme en Python avec un `print` à la fin pour vérifier.

★★★ **Niveau 3 — Aller plus loin**

Exercice 1.4.7 — Que fait cet algorithme ? ★★★

PSEUDO-CODE

```
a ← 2
b ← 3
a ← a + b
b ← a - b
a ← a - b
```

1. Faites la trace de l'algorithme.
2. Comparez les valeurs de départ et d'arrivée de `a` et de `b`. Que fait l'algorithme, en une phrase ?
3. Refaites la trace avec `a ← 10` et `b ← 4` pour confirmer.

Exercice 1.4.8 — Le prix change après coup ★★★

PSEUDO-CODE

```
prix ← 20
qte ← 3
total ← prix * qte
prix ← 25
Écrire total
```

1. Faites la trace.
2. Qu'affiche l'algorithme : 60 ou 75 ? Justifiez avec la trace.
3. On veut afficher le total avec le nouveau prix. Quelle ligne faut-il ajouter, et où ?

Exercice 1.4.9 — Retrouver la valeur de départ ★★★

PSEUDO-CODE

```
Lire x
x ← x + 4
x ← x * 3
Écrire x
```

1. L'utilisateur a tapé un nombre, et l'algorithme a affiché 27. Quel nombre a-t-il tapé ? Expliquez votre raisonnement. Pour démarrer : juste après `x ← x * 3`, `x` vaut 27. Que valait `x` juste avant cette instruction ?
2. Vérifiez en faisant la trace avec ce nombre.

2.1 — Les types simples

‹ Revoir le cours 2.1

★☆☆ **Niveau 1 — Appliquer**

Exercice 2.1.1 — Choisir le type ★☆☆

Pour chaque donnée, choisissez le type le plus adapté : entier naturel, entier relatif, réel, booléen ou chaîne de caractères.

1. Le nombre d'utilisateurs connectés à un serveur.
2. La température relevée par un capteur dans une chambre froide, en degrés entiers.
3. L'utilisation du processeur, affichée en pourcentage, par exemple 73,4.
4. « Le serveur web est-il en ligne ? »
5. L'âge d'un client.

Exercice 2.1.2 — Vrai ou faux ? ★☆☆

Dites si chaque phrase est vraie ou fausse. Justifiez en une phrase.

1. Le mot de passe `admin123` peut être rangé dans un entier.
2. En Python, `3 > 5` est un entier.
3. Une adresse IP, l'adresse d'un ordinateur sur un réseau, comme 192.168.0.1, est faite de 4 entiers naturels.
4. Le numéro 443, qui désigne le service web sur un serveur, est un entier naturel.
5. En Python, `type(7.0)` et `type(7)` donnent le même type.

Exercice 2.1.3 — Le type en Python ★☆☆

Donnez le type Python de chaque variable : `int`, `float` ou `bool`.

1. `users = 57`
2. `temp = -7`
3. `cpu = 73.4`
4. `connecte = True`
5. `prix = 12.0`

★★☆ **Niveau 2 — S'entraîner**

Exercice 2.1.4 — La fiche d'un produit ★★☆

Un magasin d'informatique décrit chaque produit avec cinq données :

- son nom ;
- son prix en euros ;
- le nombre en stock ;
- s'il est en promotion ;
- l'écart de stock depuis l'inventaire, qui peut être négatif.

1. Donnez le type de chaque donnée.
2. Écrivez en Python les cinq affectations pour une souris à 19,90 €, 12 en stock, en promotion, avec un écart de -3.

Exercice 2.1.5 — Prévoir le type ★★☆ Sur machine

1. Sans ordinateur, écrivez ce qu'affiche chaque ligne.
2. Exécutez le programme pour vérifier.

```python
print(type(7))
print(type("7"))
print(type(7.0))
print(type(7 > 3))
print(type("True"))
```

Exercice 2.1.6 — Le mauvais type ★★☆

Expliquez en une ou deux phrases le problème posé par chaque choix.

1. On range une moyenne de 12,67 dans un entier.
2. On range « le client a payé » dans une chaîne, `"oui"` ou `"non"`.
3. On range le code postal 01000 dans un entier.

★★★ **Niveau 3 — Aller plus loin**

Exercice 2.1.7 — La fiche d'inscription ★★★ Sur machine

Yanis écrit la fiche d'inscription d'une salle de sport. Elle demande le prénom, l'âge et le poids du client, puis les affiche.

1. Donnez le type de chaque donnée.
2. Complétez le programme. Le poids peut avoir une virgule.
3. Testez avec Léa, 19 ans, 58,5 kg. En Python, on tape `58.5`, avec un point.
4. La fiche doit aussi garder « le client a payé ». Ajoutez une variable `paye` qui vaut Vrai, sans la lire au clavier.
5. Ajoutez une ligne qui affiche le type des quatre variables. Comparez avec vos réponses à la question 1.

```python
prenom = input("Prénom ? ")
age = 0      # à compléter
poids = 0    # à compléter
print(prenom, age, poids)
```

Exercice 2.1.8 — Type et valeur d'une expression ★★★

Pour chaque expression Python, donnez sa valeur et son type.

1. `7 * 2`
2. `7 / 2`
3. `8 / 2`
4. `3 > 2`
5. `2.0 * 3`

Exercice 2.1.9 — Une application de livraison ★★★

Une application suit des colis. Un développeur a écrit ces affectations :

```python
nb_colis = 4
poids = 2.5
livre = "non"
code_suivi = 12
temp_camion = -18
```

Le code de suivi réel est `FR0012AB`.

1. Donnez le bon type de chaque donnée.
2. Trouvez les deux affectations mal choisies, et corrigez-les.

2.2 — Les limites de la machine

‹ Revoir le cours 2.2

★☆☆ **Niveau 1 — Appliquer**

Exercice 2.2.1 — Combien de valeurs ? ★☆☆

Complétez le tableau. Utilisez le tableau des puissances de 2 des rappels.

Nombre de bits	Nombre de valeurs	Plus grand entier naturel
4		
8		
10		

Exercice 2.2.2 — Les grandes tailles ★☆☆ Sur machine

1. Quelle est la plus grande valeur d'un entier naturel sur 16 bits ?
2. Et sur 32 bits ?
3. Exécutez ce programme pour vérifier vos deux réponses. En Python, la puissance s'écrit `**` : `2 ** 16` veut dire 2^{16}.

```python
print(2 ** 16 - 1)
print(2 ** 32 - 1)
```

Exercice 2.2.3 — Exact ou approché ? ★☆☆

Dans la machine, chaque valeur est-elle rangée exactement, ou à peu près ?

1. L'entier 42.
2. Le réel 0,1.
3. L'entier 1024.
4. Le réel 0,3.
5. Le booléen Vrai.

★★☆ **Niveau 2 — S'entraîner**

Exercice 2.2.4 — Vérifier des bornes ★★☆

1. Un entier relatif sur 16 bits va de -32 768 à 32 767. Vérifiez ces deux bornes avec la règle du cours.
2. La distance de la Terre au Soleil fait environ 150 000 000 km. Tient-elle dans un entier naturel sur 32 bits ?
3. Un compteur est un entier naturel sur 8 bits. Après 255, il repart à 0 : 255 + 1 donne 0, 255 + 2 donne 1. Il vaut 250. Que donne 250 + 10 ?

Exercice 2.2.5 — L'addition qui tombe faux ★★☆ Sur machine

1. Sans ordinateur, que devrait afficher chaque ligne de ce programme ?
2. Exécutez-le. Notez ce qui s'affiche vraiment.
3. Expliquez la différence.
4. Modifiez la dernière ligne pour que le test affiche `True`.

PYTHON

```python
print(0.2 + 0.4)
s = 0.1 + 0.1 + 0.1
print(s)
print(s == 0.3)
```

Exercice 2.2.6 — Combien de bits ? ★★☆

Pour chaque donnée, choisissez le plus petit nombre de bits qui suffit : 8, 16 ou 32. Dites aussi si c'est un entier naturel ou relatif.

1. L'âge d'une personne, de 0 à 130 ans.
2. Le nombre de visiteurs d'un site web en un jour, jusqu'à 50 000.
3. La population de la France, environ 68 millions.
4. Une température extérieure, de -40 à 60 °C.

★★★ **Niveau 3 — Aller plus loin**

Exercice 2.2.7 — Compter en centimes ★★★ Sur machine

Une caisse additionne trois articles : 4,35 €, 2,10 € et 0,55 €.

1. Calculez le total à la main.
2. Exécutez le programme. Que remarquez-vous ?

3. Beaucoup de logiciels de caisse comptent en centimes, dans des entiers. Pourquoi ?

```python
total = 4.35 + 2.10 + 0.55
print(total)
total_centimes = 435 + 210 + 55
print(total_centimes)
```

Exercice 2.2.8 — Une valeur trop grande ★★★

Un programme range une vitesse dans un entier relatif sur 16 bits. Sur un nouveau modèle de drone, la vitesse mesurée peut atteindre 40 000.

1. Quelle est la plus grande valeur d'un entier relatif sur 16 bits ?
2. La valeur 40 000 tient-elle ? Que se passe-t-il sinon ?
3. Pour un entier relatif, quel nombre de bits faut-il choisir : 16 ou 32 ?
4. Quelle vérification aurait évité le problème ?

Exercice 2.2.9 — Comparer deux prix ★★★ Sur machine

Un site vérifie que le prix de trois câbles à 0,10 € fait bien 0,30 €.

1. Exécutez le programme. Le test est-il réussi ?
2. Expliquez le résultat.
3. Corrigez la ligne du test pour qu'il réussisse.

```python
prix = 0.1 * 3
print(prix)
print(prix == 0.3)
```

2.3 — Les chaînes de caractères

‹ Revoir le cours 2.3

★☆☆ Niveau 1 — Appliquer

Exercice 2.3.1 — Le code d'un badge ★☆☆

Dans un lycée, chaque badge porte un code comme `badge ← "B3-2041"`. Le premier caractère est le bâtiment. Le deuxième est l'étage.

1. Quelle est la longueur du code ?

2. Quelle instruction extrait le bâtiment ? Que vaut-elle ?
3. Quelle instruction extrait l'étage ? Que vaut-elle ?
4. Écrivez l'instruction pseudo-code qui affiche `Bâtiment B étage 3`, avec `Écrire` et des virgules.

Exercice 2.3.2 — Compter les caractères ★☆☆

Donnez la longueur de chaque chaîne.

1. `"BTS SIO"`
2. `"192.168.1.10"`
3. `"Bonjour !"`
4. `""`

Exercice 2.3.3 — Texte ou nombre ? ★☆☆

Pour chaque valeur, dites si c'est une chaîne ou un nombre, et si on peut calculer avec.

1. `"123"`
2. `123`
3. `"12.5"`
4. `12.5`

★★☆ Niveau 2 — S'entraîner

Exercice 2.3.4 — Une adresse mail ★★☆ Sur machine

On a `mail = "lea.martin@lycee.fr"`.

1. Quelle est la longueur de `mail` ?
2. À quel indice est le caractère `@` ? Comptez à la main.
3. Écrivez les instructions Python qui affichent la longueur, le premier caractère, le dernier et le caractère d'indice 10.
4. Exécutez pour vérifier.

Exercice 2.3.5 — La trace d'une chaîne ★★☆

Faites la trace de cet algorithme. Donnez la valeur de chaque variable à la fin.

```pseudo-code
mot ← "RESEAU"
n ← longueur(mot)
premier ← mot[0]
dernier ← mot[n - 1]
quatrieme ← mot[3]
```

Exercice 2.3.6 — Deux erreurs d'indice ★★☆

Un élève veut afficher la première et la dernière lettre de `"Python"`. Il écrit :

```python
mot = "Python"
print(mot[1])
print(mot[len(mot)])
```

1. Que va afficher le premier `print` ? Est-ce la première lettre ?
2. Que se passe-t-il avec le deuxième `print` ?
3. Corrigez les deux lignes.

★★★ Niveau 3 — Aller plus loin

Exercice 2.3.7 — Corriger un prénom ★★★

Une fiche contient `prenom ← "lea"`. On veut une majuscule au début. Un élève écrit :

```pseudo-code
prenom ← "lea"
prenom[0] ← "L"
```

1. Que se passe-t-il à la deuxième ligne ? Pourquoi ?
2. Comment obtenir `"Lea"` dans la variable `prenom`, avec une seule affectation ?
3. Expliquez la différence entre « modifier un caractère » et « ranger une nouvelle chaîne dans la variable ».

Exercice 2.3.8 — Première et dernière lettre ★★★ Sur machine

1. Écrivez un programme qui lit un mot au clavier, puis affiche sa longueur, sa première lettre et sa dernière lettre. Écrivez d'abord le pseudo-code, puis le Python. Testez avec `Python`.
2. Testez avec un mot d'une seule lettre, `A`. Que valent la première et la dernière lettre ? Pourquoi ?

3. L'utilisateur appuie sur Entrée sans rien taper. Que vaut la longueur ? Que se passe-t-il à l'affichage ? Pourquoi ?

Exercice 2.3.9 — L'avant-dernier caractère ★★★ Sur machine

Un code d'accès a toujours au moins deux caractères, par exemple `4F7A2C`. Le service informatique veut afficher son avant-dernier caractère.

1. Pour `4F7A2C`, quel est l'avant-dernier caractère ? À quel indice est-il ?
2. Donnez l'indice de l'avant-dernier caractère pour une chaîne quelconque.
3. Écrivez le programme Python qui lit un code et affiche son avant-dernier caractère. Testez avec `4F7A2C`, puis avec `XY`.

2.4 — Changer de type : le transtypage

‹ Revoir le cours 2.4

★☆☆ **Niveau 1 — Appliquer**

Exercice 2.4.1 — Les conversions de base ★☆☆

Donnez le résultat de chaque conversion.

1. `int("42")`
2. `float("3.5")`
3. `str(42)`
4. `int(3.9)`
5. `int(True)`

Exercice 2.4.2 — Quelle conversion après input ? ★☆☆

Pour chaque donnée lue avec `input`, dites quelle conversion il faut : aucune, `int` ou `float`.

1. L'âge.
2. Le prix d'un article, par exemple 12.99.
3. Le prénom.
4. Le code postal.
5. Le nombre d'articles commandés.

Exercice 2.4.3 — Additionner ou coller ? ★☆☆

Que donne chaque expression en Python ?

1. `"10" + "5"`
2. `10 + 5`
3. `int("10") + int("5")`

★★☆ Niveau 2 — S'entraîner

Exercice 2.4.4 — Prévoir un transtypage ★★☆ Sur machine

1. Pour chaque ligne, écrivez ce que Python va afficher, sans l'exécuter.
2. Exécutez le programme pour vérifier.
3. Que donnerait `int("3.5")` ? Pourquoi ?

PYTHON

```python
print(int("42") + 8)
print("42" + "8")
print(float("3.5") * 2)
print(int(9.99))
print(str(7) + "7")
print(int(True) + 1)
```

Exercice 2.4.5 — Le total d'une commande ★★☆ Sur machine

Complétez le programme. Il lit le prix d'un article et la quantité. Il affiche ensuite un seul message, collé avec `+`, par exemple `Total : 37.5 €`.

Testez avec 12.5 et 3.

PYTHON

```python
prix = input("Prix ? ")       # à corriger
qte = input("Quantité ? ")    # à corriger
total = 0                     # à remplacer par le calcul
print("Total :", total)       # à remplacer par un message collé avec +
```

Exercice 2.4.6 — Corriger les conversions ★★☆

Chaque ligne s'arrête sur une erreur. Expliquez pourquoi, puis corrigez-la.

PYTHON

```python
print("Total : " + 25)
print(float("3,5"))
print(int("12.5"))
```

★★★ Niveau 3 — Aller plus loin

Exercice 2.4.7 — Minutes depuis minuit ★★★ Sur machine

Un badge de pointage lit l'heure et les minutes d'arrivée sous forme de texte, par exemple `"14"` et `"35"`. On veut le nombre de minutes écoulées depuis minuit.

1. Calculez-le à la main pour 14 h 35.
2. Écrivez l'algorithme en pseudo-code, avec deux `Lire`.
3. Écrivez le programme Python, et testez avec 14 et 35.

Exercice 2.4.8 — La note coupée ★★★ Sur machine

Une note est tapée au clavier, par exemple `15.75`. Le logiciel doit afficher sa partie entière, ici 15.

1. Pourquoi `int(input("Note ? "))` s'arrête-t-il sur une erreur avec `15.75` ?
2. Complétez le programme en trois lignes : on lit le texte, on le change en réel, puis le réel en entier.
3. `int` arrondit-il, ou coupe-t-il ? Testez avec `15.75`.

PYTHON

```python
texte = input("Note ? ")
reel = 0     # à compléter
note = 0     # à compléter
print(note)
```

Exercice 2.4.9 — Le code postal qui perd son zéro ★★★

Un développeur range les codes postaux dans des entiers. Pour Bourg-en-Bresse, il écrit `cp = int("01000")`.

1. Que vaut `cp` ?
2. Que donne ensuite `str(cp)` ? Est-ce le bon code postal ?
3. Quel type faut-il choisir ? Pourquoi ?
4. Le même développeur range l'âge en texte : `age = "25"`. Que donne `age + "1"` ? Et `int(age) + 1` ?

5. Écrivez une règle qui dit quand choisir une chaîne et quand choisir un nombre.

3.1 — Les opérateurs numériques

‹ Revoir le cours 3.1

★☆☆ **Niveau 1 — Appliquer**

Exercice 3.1.1 — Quotient et reste ★☆☆

1. Calculez `47 div 5` et `47 mod 5`.
2. Calculez `100 mod 7`.
3. Le nombre 2026 est-il pair ? Écrivez le test en pseudo-code.

Exercice 3.1.2 — Divisible ou pas ★☆☆

Donnez la valeur, Vrai ou Faux, de chaque test.

1. `27 mod 3 = 0`
2. `28 mod 3 = 0`
3. `2025 mod 2 = 0`
4. `343 mod 7 = 0`

Exercice 3.1.3 — Écrire une formule en ligne ★☆☆

1. Écrivez en ligne, avec les opérateurs du cours :
    1. $\frac{a+b}{a-b}$
    2. $3x^2+1$
2. Écrivez en maths l'expression en ligne `a / b + c`.

★★☆ **Niveau 2 — S'entraîner**

Exercice 3.1.4 — Convertir une durée ★★☆ Sur machine

Un serveur affiche depuis combien de secondes il tourne. On veut l'afficher en heures, minutes et secondes.

1. Complétez le programme. Il lit un nombre de secondes et affiche les heures, les minutes et les secondes.
2. Testez-le avec `4000` secondes.

```python
duree = int(input("Durée en secondes ? "))
heures = 0      # à compléter
reste = 0       # à compléter
minutes = 0     # à compléter
secondes = 0    # à compléter
print(heures, "h", minutes, "min", secondes, "s")
```

Exercice 3.1.5 — La moyenne pondérée ★★☆

À un contrôle, la note `a` compte 3 fois, et la note `b` compte 2 fois. On dit que `a` a le **coefficient** 3, et `b` le coefficient 2. Pour la moyenne, on multiplie chaque note par son coefficient, on additionne, puis on divise par 3 + 2 = 5.

1. Écrivez la moyenne en ligne.
2. Calculez-la pour `a = 6` et `b = 2`, puis pour `a = 12` et `b = 15`.
3. Un élève écrit `3 * a + 2 * b / 5`. Que calcule-t-il vraiment pour `a = 12` et `b = 15` ?

Exercice 3.1.6 — Quel jour dans 100 jours ? ★★☆

On numérote les jours : 0 pour lundi, 1 pour mardi, 2 pour mercredi, 3 pour jeudi, 4 pour vendredi, 5 pour samedi, 6 pour dimanche. Aujourd'hui, `jour ← 0` : c'est lundi.

1. Pourquoi le reste de la division par 7 donne-t-il le jour de la semaine ?
2. Écrivez l'expression qui donne le numéro du jour dans `n` jours.
3. Quel jour serons-nous dans 100 jours ? Dans 31 jours ?

★★★ Niveau 3 — Aller plus loin

Exercice 3.1.7 — Les chiffres d'un nombre ★★★ Sur machine

On lit un entier de 3 chiffres, par exemple 472. On veut ses chiffres, sans le transformer en chaîne.

1. Quelle expression donne le chiffre des unités ?
2. Quelle expression donne le chiffre des dizaines ?
3. Quelle expression donne le chiffre des centaines ?
4. Écrivez le programme Python qui affiche les trois chiffres, puis leur somme. Testez-le avec 472.

Exercice 3.1.8 — La trace d'un calcul avec div et mod ★★★

PSEUDO-CODE

```
a ← 47
b ← 5
q ← a div b
r ← a mod b
a ← b
b ← r
q ← a div b
r ← a mod b
```

1. Faites la trace de cet algorithme : une colonne pour `a`, `b`, `q`, `r`.
2. Que valent les quatre variables à la fin ?
3. Vérifiez la première division : `5 × q + r` redonne-t-il 47 ?

Exercice 3.1.9 — Coder une lettre ★★★ Sur machine

Cet exercice reprend le début d'un sujet de BTS. Chaque lettre a un **rang** : son numéro dans l'alphabet, en partant de 0.

A	B	C	D	E	F	G	H	I	J	K	L	M
0	1	2	3	4	5	6	7	8	9	10	11	12

N	O	P	Q	R	S	T	U	V	W	X	Y	Z
13	14	15	16	17	18	19	20	21	22	23	24	25

Pour coder la lettre de rang `n`, on calcule le reste de la division de $7n + 5$ par 26. Ce reste est le rang de la lettre codée.

1. Écrivez l'expression en ligne du rang codé, avec `mod`.
2. Codez la lettre C, de rang 2.
3. Codez la lettre T, de rang 19.
4. Vérifiez en Python.

3.2 — Les fonctions mathématiques usuelles

‹ Revoir le cours 3.2

★☆☆ **Niveau 1 — Appliquer**

Exercice 3.2.1 — Trois arrondis ★☆☆ Sur machine

On a `x ← 7.456`.

1. Que donne l'arrondi au plus près, à 2 décimales ?
2. Que donne l'arrondi par défaut, à l'entier ?
3. Que donne l'arrondi par excès, à l'entier ?
4. Exécutez ce programme pour vérifier vos trois réponses.

```python
import math
x = 7.456
print(round(x, 2), math.floor(x), math.ceil(x))
```

Exercice 3.2.2 — L'écart de température ★☆☆

Un capteur mesure -3 °C la nuit et 5 °C le jour.

1. Quel calcul donne l'écart entre les deux, sans signe ?
2. Que donne `abs(12 - 20)` ? Et `abs(20 - 12)` ?

Exercice 3.2.3 — Des racines carrées ★☆☆

1. Une salle carrée a une aire de 49 m². Quelle instruction Python donne la longueur de son côté ?
2. Que donne `math.sqrt(144)` ?
3. Que donne `round(math.sqrt(2), 2)` ?

★★☆ **Niveau 2 — S'entraîner**

Exercice 3.2.4 — Arrondir au bon sens ★★☆ Sur machine

Un technicien range 130 câbles dans des cartons de 50. Un article coûte 19,987 € avant arrondi.

1. Combien de cartons faut-il ? Quelle fonction utiliser ?
2. Quel prix affiche-t-on, arrondi au centime le plus proche ?
3. Que donne l'arrondi par défaut de 19,987, à l'entier ?

Exécutez ce programme et comparez avec vos réponses :

```python
import math
print(math.ceil(130 / 50))
print(round(19.987, 2))
print(math.floor(19.987))
```

Exercice 3.2.5 — Les pages d'un rapport ★★☆ Sur machine

Un rapport affiche les résultats d'une recherche, 20 lignes par page. Dans cet exercice, le nombre de lignes n'est jamais un multiple de 20 : la dernière page n'est jamais pleine.

1. Pour 53 lignes, combien faut-il de pages ?
2. Combien de lignes y a-t-il sur la dernière page ?
3. Écrivez un programme qui lit le nombre de lignes et affiche ces deux nombres. Testez avec 53.

Exercice 3.2.6 — Le prix TTC ★★☆ Sur machine

Un article coûte 19,99 € hors taxes. Le prix TTC, toutes taxes comprises, est le prix hors taxes multiplié par 1,2.

1. Calculez à la main 19,99 × 1,2. Puis exécutez `print(19.99 * 1.2)` en Python et notez ce qui s'affiche. Pourquoi est-ce différent ?
2. Quel prix TTC affiche-t-on, au centime le plus proche ?
3. Un élève coupe après 2 décimales avec `math.floor(prix * 100) / 100`. Quel prix obtient-il ? Est-ce un arrondi au plus près ?

```python
import math
prix = 19.99 * 1.2
print(prix)
```

★★★ Niveau 3 — Aller plus loin

Exercice 3.2.7 — Simuler le hasard ★★★ Sur machine

Un **port** est le numéro qui désigne un service sur un ordinateur, entre 0 et 65 535 : 443 pour le web sécurisé, par exemple.

1. Quelle instruction tire 0 ou 1, comme pile ou face ?
2. Un serveur choisit un port au hasard entre 1024 et 65535. Quelle instruction ?

3. On simule le temps de réponse d'un serveur : un réel au hasard entre 0,5 et 2 secondes. Quelle instruction ?
4. Exécutez le programme ci-dessous trois fois. Les résultats sont-ils les mêmes ?
5. Un élève simule un dé avec `round(random.uniform(1, 6))`. Le réel tiré va de 1 à 6. Quels réels donnent 1 une fois arrondis ? Quels réels donnent 3 ? Le tirage du dé est-il encore uniforme ?

```python
import random
print(random.randint(0, 1), random.randint(1024, 65535))
print(random.uniform(0.5, 2))
```

Exercice 3.2.8 — La distance entre deux points ★★★

Sur un plan, la distance entre le point A de coordonnées (x_1, y_1) et le point B de coordonnées (x_2, y_2) est :

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

1. Écrivez cette formule en ligne, en Python.
2. Calculez la distance entre A(1 ; 2) et B(4 ; 6).
3. Un élève écrit `math.sqrt(x2 - x1 ** 2 + y2 - y1 ** 2)`. Quelle erreur fait-il ?

Exercice 3.2.9 — Arrondir au quart d'heure ★★★ Sur machine

Un technicien facture ses interventions par quarts d'heure entamés. 37 minutes de travail sont facturées 45 minutes.

1. Combien de quarts d'heure entamés dans 37 minutes ?
2. Écrivez l'expression qui donne les minutes facturées pour `m` minutes.
3. Écrivez le programme qui lit `m` et affiche les minutes facturées. Testez avec 37, puis avec 60.

3.3 — Comparer et combiner des conditions

‹ Revoir le cours 3.3

★☆☆ **Niveau 1 — Appliquer**

Exercice 3.3.1 — Vrai ou faux ? ★☆☆

Donnez la valeur de chaque comparaison, en Python.

1. `7 > 3`
2. `10 != 10`
3. `"abc" < "abd"`
4. `4 <= 4`

Exercice 3.3.2 — Les quatre opérateurs ★☆☆

On a `a ← Vrai` et `b ← Faux`. Donnez la valeur de :

1. `NON a`
2. `a ET b`
3. `a OU b`
4. `a OUX b`

Exercice 3.3.3 — Majeur ? ★☆☆

On a `majeur ← age ≥ 18`.

1. Que vaut `majeur` pour `age = 17` ? Pour `age = 18` ?
2. Écrivez cette ligne en Python.

★★☆ **Niveau 2 — S'entraîner**

Exercice 3.3.4 — Traduire une condition ★★☆

Traduisez chaque règle en expression booléenne, en pseudo-code puis en Python. Puis donnez sa valeur pour les données indiquées.

1. « L'âge est entre 18 et 25 ans inclus. » Avec `age = 30`.
2. « Le jour est samedi ou dimanche. » Avec `jour = "dimanche"`.
3. « Le stock n'est pas vide et le produit est encore vendu. » Avec `stock = 0` et `vendu = True`.

Exercice 3.3.5 — Le mot de passe accepté ★★☆ Sur machine

Un site accepte un mot de passe si ces trois règles sont vraies :

- il fait au moins 8 caractères ;
- il est différent de l'identifiant ;
- il est différent du mot de passe interdit `motdepasse`.

1. Écrivez la condition `accepte` en pseudo-code, avec `mdp`, `login` et `longueur`.
2. Complétez le programme Python et testez-le avec l'identifiant `ines` et le mot de passe `Sio2026!`.
3. Sans ordinateur, que vaut `accepte` pour les mots de passe `ines` et `motdepasse` ?

```python
login = "ines"
mdp = input("Mot de passe ? ")
accepte = False     # à compléter
print(accepte)
```

Exercice 3.3.6 — L'année bissextile ★★☆

Une année est bissextile, avec un 29 février, si elle est divisible par 4 mais pas par 100, ou si elle est divisible par 400.

1. Traduisez cette règle en expression booléenne, en pseudo-code, avec la variable `an`. Mettez des parenthèses autour de chaque morceau.
2. Donnez sa valeur pour 2024, 1900, 2000 et 2026.

★★★ Niveau 3 — Aller plus loin

Exercice 3.3.7 — L'ordre des tests compte ★★★

On a `mot ← "BTSSIO"` et `k ← 6`.

1. Que vaut `k < longueur(mot)` ?
2. Que fait Python avec `k < len(mot) and mot[k] == "S"` ? Avec quel résultat ?
3. Que fait Python avec `mot[k] == "S" and k < len(mot)` ?
4. Quelle règle en tirez-vous ?

Exercice 3.3.8 — Le contraire d'une condition ★★★

Un club donne accès à la salle si `age ≥ 18 ET inscrit`.

Une règle vous aide : le contraire de `A ET B` s'écrit `NON A OU NON B`. L'accès est refusé dès qu'une des deux conditions manque.

1. Écrivez la condition « accès refusé » avec NON devant toute la condition.
2. Écrivez-la avec la règle, sans NON devant la parenthèse. Le contraire de `age ≥ 18` est `age < 18`.
3. Vérifiez vos deux écritures pour `age = 16` et `inscrit = Vrai`, puis pour `age = 20` et `inscrit = Faux`.

Exercice 3.3.9 — Le magasin est-il ouvert ? ★★★ Sur machine

Un magasin est ouvert de 9 h à 12 h et de 14 h à 18 h, sauf le dimanche.

À 9 h, il est ouvert. À 12 h, il est fermé. De même, à 14 h il est ouvert, et à 18 h il est fermé.

1. Écrivez la condition `ouvert` avec les variables `h`, une heure entière, et `jour`. Mettez des parenthèses autour de chaque morceau.
2. Complétez le programme et testez-le avec 15 h un mardi.
3. Sans ordinateur : que vaut `ouvert` à 12 h un lundi ? À 10 h un dimanche ?

PYTHON

```python
h = int(input("Heure ? "))
jour = input("Jour ? ")
ouvert = False      # à compléter
print(ouvert)
```

3.4 — Les opérateurs bit à bit

‹ Revoir le cours 3.4

★☆☆ **Niveau 1 — Appliquer**

Exercice 3.4.1 — Rang par rang ★☆☆

Calculez à la main, rang par rang :

1. `1100 ET 1010`
2. `1100 OU 1010`
3. `1100 OUX 1010`

Exercice 3.4.2 — Repérer les bits ★☆☆

On a le nombre binaire `10110`.

1. Que vaut le bit 0 ? Le bit 1 ? Le bit 4 ?
2. Quel masque désigne le bit 3, sur 5 bits ?

Exercice 3.4.3 — Tester un bit ★☆☆

On a `x = 0b1011`.

1. Quel masque teste le bit 2 ?
2. Que vaut `x & masque` ? Le bit 2 vaut-il 1 ?

★★☆ Niveau 2 — S'entraîner

Exercice 3.4.4 — Les droits d'un fichier ★★☆

Les droits d'un fichier tiennent sur 3 bits : le bit 2 pour lire, le bit 1 pour écrire, le bit 0 pour exécuter. Un fichier a les droits `droits = 0b110`.

1. Quels droits a-t-il ?
2. Avec quel masque et quel opérateur teste-t-on le droit d'exécuter ? Quel est le résultat ?
3. Comment ajouter le droit d'exécuter ? Quel résultat ?
4. Comment retirer le droit d'écrire, en partant de `0b110` ? Quel résultat ?
5. Comment inverser le droit de lire, en partant de `0b110` ? Quel résultat ?

Exercice 3.4.5 — Éteindre un bit avec l'inverse ★★☆

On a `x = 0b1111`, et on veut mettre le bit 2 à 0.

1. Quel est le masque du bit 2, sur 4 bits ?
2. Écrivez l'inverse de ce masque, bit par bit.
3. Calculez `x ET inverse`.
4. Pourquoi `x ET masque` ne convient pas ?

Exercice 3.4.6 — Le clignotant ★★☆ Sur machine

Quatre voyants sont rangés dans les 4 bits de `voyants`. Un 1 veut dire allumé. Au départ, `voyants = 0b0101`.

1. Quel masque et quel opérateur inversent les 4 voyants à la fois ?
2. Que vaut `voyants` après une inversion ? Après une deuxième ?

3. Exécutez le programme pour vérifier.
4. Ajoutez à la fin une ligne qui éteint seulement le voyant du bit 3, et une ligne qui l'affiche. Que s'affiche-t-il ?

```python
voyants = 0b0101
voyants = voyants ^ 0b1111
print(bin(voyants))
voyants = voyants ^ 0b1111
print(bin(voyants))
```

★★★ Niveau 3 — Aller plus loin

Exercice 3.4.7 — Le réseau d'une adresse ★★★ Sur machine

Une **adresse IP** est l'adresse d'un ordinateur sur un réseau. Elle s'écrit avec 4 nombres de 0 à 255, comme `192.168.1.198`.

Un réseau découpe ces nombres avec un masque : les bits à 1 du masque désignent le réseau, les bits à 0 désignent l'ordinateur. Deux ordinateurs sont dans le même réseau si leur nombre ET le masque donne le même résultat.

On regarde seulement le dernier nombre, avec le masque `11000000`, soit 192.

1. L'ordinateur A a le nombre 198, soit `11000110`. Calculez `11000110 ET 11000000`, rang par rang, puis en décimal.
2. L'ordinateur B a le nombre 230, soit `11100110`. Calculez `230 ET 192`.
3. L'ordinateur C a le nombre 70, soit `01000110`. Calculez `70 ET 192`.
4. Quels ordinateurs sont dans le même réseau que A ? Vérifiez vos calculs en Python.

```python
print(198 & 192, 230 & 192, 70 & 192)
```

Exercice 3.4.8 — Tester deux bits à la fois ★★★

Un registre d'alarmes tient sur 8 bits. Le bit 1 signale « porte ouverte », le bit 2 signale « mouvement ».

1. Quel masque désigne les bits 1 et 2 ensemble ?
2. Pour `x = 0b10110110`, que vaut `x ET masque` ? Et pour `x = 0b10110010` ?
3. Écrivez l'expression booléenne « les deux alarmes sont allumées », avec `x` et le masque.
4. Écrivez l'expression « au moins une des deux alarmes est allumée ».
5. Donnez la valeur des deux expressions pour les deux valeurs de `x`.

Exercice 3.4.9 — Les options d'un objet connecté ★★★ Sur machine

Une montre range ses options sur 4 bits : bit 0 = wifi, bit 1 = bluetooth, bit 2 = GPS, bit 3 = NFC. NFC veut dire *Near Field Communication* : c'est le paiement sans contact. Au départ, `options = 0`.

Répondez d'abord sur papier.

1. On allume le wifi et le GPS d'un seul coup. Quel masque, quel opérateur ?
2. On teste le GPS. Quel calcul ?
3. On éteint le wifi. Quel calcul ?
4. On inverse le NFC. Quel calcul ?
5. Faites la trace de `options` en binaire.
6. Complétez le programme : remplacez chaque `0` marqué « à compléter » par le bon masque. Exécutez-le et comparez avec votre trace.

```python
options = 0
options = options | 0      # masque à compléter
print(bin(options))
print(options & 0)         # masque à compléter
options = options & ~0     # masque à compléter
print(bin(options))
options = options ^ 0      # masque à compléter
print(bin(options))
```

3.5 — Les opérateurs de chaînes

‹ Revoir le cours 3.5

★☆☆ **Niveau 1 — Appliquer**

Exercice 3.5.1 — Coller des chaînes ★☆☆

Que vaut chaque expression ?

1. `"Ines" + " " + "Martin"`
2. `"BTS" + "SIO"`
3. `"3" + "4"`

Exercice 3.5.2 — Extraire un morceau ★☆☆

On a `s = "BTS SIO 2026"`. Que vaut :

1. `s[:3]`
2. `s[4:7]`
3. `s[-4:]`

Exercice 3.5.3 — Chercher un motif ★☆☆

Que vaut :

1. `"bonjour".find("j")`
2. `"bonjour".find("z")`
3. `"a@b.fr".find("@")`

★★☆ Niveau 2 — S'entraîner

Exercice 3.5.4 — Découper une adresse mail ★★☆ Sur machine

On a `mail = "ines.martin@lycee-sio.fr"`.

1. Quelle instruction donne la position de `@` ? Quelle est sa valeur ?
2. Avec cette position `p`, extrayez l'identifiant, avant `@`.
3. Extrayez le domaine, après `@`.
4. Extrayez les 2 dernières lettres, l'extension.
5. Construisez le message `Domaine : lycee-sio.fr` par concaténation.

PYTHON

```python
mail = "ines.martin@lycee-sio.fr"
p = 0       # à compléter : la position de @
print(p)
# à compléter
```

Exercice 3.5.5 — Le nom et l'extension d'un fichier ★★☆ Sur machine

On lit un nom de fichier, comme `rapport.pdf`. Il contient un seul point.

1. Comment trouver la position `p` du point ?
2. Comment extraire le nom, avant le point ? L'extension, après ?
3. Complétez le programme et testez-le avec `rapport.pdf`.

```python
fichier = input("Nom du fichier ? ")
p = 0               # à compléter : la position du point
nom = ""            # à compléter
extension = ""      # à compléter
print(nom, extension)
```

Exercice 3.5.6 — Construire un message ★★☆

On a `prenom = "Tom"` et `total = 42.456`.

1. Construisez par concaténation le message `Tom doit 42.46 €`.
2. Pourquoi faut-il `str` ?
3. Pourquoi faut-il `round` ?

★★★ Niveau 3 — Aller plus loin

Exercice 3.5.7 — Retourner une date ★★★ Sur machine

Une date arrive au format `18/10/2026`, toujours avec 2 chiffres pour le jour et le mois. Un logiciel attend le format `2026-10-18`.

1. Quels indices donnent le jour ? Le mois ? L'année ?
2. Construisez la date au nouveau format par concaténation.
3. Le mois doit aussi servir à un calcul. Comment obtenir le mois sous forme d'entier ?
4. Complétez le programme et testez-le avec `18/10/2026`.

```python
date = input("Date (jj/mm/aaaa) ? ")
nouvelle = ""       # à compléter
mois = 0            # à compléter
print(nouvelle)
print("Mois suivant :", mois + 1)
```

Exercice 3.5.8 — Une adresse du réseau local ? ★★★

Une adresse IP commence par `192.168.` quand elle appartient à un réseau local. On a `ip = "192.168.1.25"`.

1. Écrivez un test avec une tranche de `ip`.
2. Un élève écrit `ip.find("192.168.") != -1`. Que vaut ce test pour `"10.192.168.5"` ? Est-ce juste ?
3. Écrivez un test avec `find` qui vérifie que l'adresse **commence** par `192.168.`.

Exercice 3.5.9 — Fabriquer un identifiant ★★★ Sur machine

Un lycée fabrique l'identifiant d'un élève ainsi : la première lettre du prénom, puis les 7 premières lettres du nom, au plus. Il ajoute `@lycee.fr` pour l'adresse mail.

1. Quelle instruction donne la première lettre du prénom ?
2. Que donne `nom[:7]` pour `martin` ? Pour `fontaine-dupont` ?
3. Écrivez le programme qui lit le prénom et le nom, puis affiche l'adresse mail. Testez avec `ines` et `fontaine-dupont`.
4. Ajoutez la variable `court`, qui vaut Vrai si l'identifiant fait au plus 6 caractères.

PYTHON

```python
prenom = input("Prénom ? ")
nom = input("Nom ? ")
identifiant = ""    # à compléter
print(identifiant + "@lycee.fr")
```

4.1 — Faire un choix : Si… Alors… Sinon

‹ Revoir le cours 4.1

★☆☆ **Niveau 1 — Appliquer**

Exercice 4.1.1 — Pair ou impair ★☆☆ Sur machine

Un programme lit un entier et dit s'il est pair ou impair.

1. Complétez l'algorithme.

PSEUDO-CODE

```
Lire n
Si n mod 2 = . . . Alors
    Écrire "pair"
Sinon
    Écrire . . .
Fin de Si
```

1. Qu'affiche-t-il pour 14 ? Pour 7 ?
2. Complétez le programme Python, puis testez-le avec 7.

```python
n = int(input("Un entier ? "))
if n % 2 == 0:
    print("pair")
```

Exercice 4.1.2 — Suivre un Si ★☆☆

On considère cet algorithme :

```
Si x > 10 Alors
    y ← x - 10
Sinon
    y ← x + 10
Fin de Si
Écrire y
```

1. Que vaut `y` si `x` vaut 12 ?
2. Que vaut `y` si `x` vaut 4 ?
3. Que vaut `y` si `x` vaut 10 ? Attention au signe `>`.

Exercice 4.1.3 — Le digicode ★☆☆

La porte de la salle serveur s'ouvre avec le code `"4721"`. Le programme lit un code tapé par l'utilisateur.

1. Écrivez l'algorithme en pseudo-code : s'il tape le bon code, on affiche « Porte ouverte », sinon « Code refusé ».
2. Pourquoi le code est-il rangé dans une chaîne, et pas dans un entier ?

★★☆ Niveau 2 — S'entraîner

Exercice 4.1.4 — La mention ★★☆

On veut afficher la mention d'un étudiant selon sa moyenne :

- 16 ou plus : « Très bien » ;
- 14 ou plus : « Bien » ;
- 12 ou plus : « Assez bien » ;
- 10 ou plus : « Passable » ;
- sinon : « Non admis ».

1. Écrivez l'algorithme en pseudo-code avec `Si … Sinon si … Sinon`.

2. Qu'affiche-t-il pour une moyenne de 13,5 ? Et de 16 ?
3. Un collègue a écrit les tests dans l'ordre inverse, en commençant par `moyenne ≥ 10`. Qu'affiche sa version pour 16 ? Pourquoi ?

Exercice 4.1.5 — Le prix du billet ★★☆ Sur machine

Un cinéma fixe le prix du billet selon l'âge :

- moins de 12 ans : 6 € ;
- 65 ans ou plus : 7 € ;
- les autres : 10 €.

1. Écrivez l'algorithme en pseudo-code. Il lit l'âge et affiche le prix.
2. Écrivez le programme en Python.
3. Préparez un jeu d'essai : une valeur par branche et les valeurs limites 11, 12, 64, 65, avec le prix attendu. Puis testez.

```python
age = int(input("Âge ? "))
# à compléter
```

Exercice 4.1.6 — Le plus grand de trois ★★☆

On a trois nombres `a`, `b` et `c`. On veut ranger le plus grand dans `grand`.

1. Complétez l'algorithme.

```
Si a ≥ b ET a ≥ c Alors
    grand ← a
Sinon si . . . Alors
    grand ← b
Sinon
    grand ← . . .
Fin de Si
```

1. Que vaut `grand` pour `a = 4`, `b = 9`, `c = 2` ? Quel bloc est exécuté ?
2. Et pour `a = 5`, `b = 5`, `c = 1` ?

★★★ **Niveau 3 — Aller plus loin**

Exercice 4.1.7 — L'année bissextile ★★★

Une année est bissextile si elle est divisible par 4 mais pas par 100, ou si elle est divisible par 400.

1. Traduisez cette règle en une condition, avec `mod`, `ET`, `OU`.
2. Écrivez l'algorithme qui lit une année et affiche « bissextile » ou « non bissextile ».
3. Donnez le résultat pour 2024, 1900 et 2000, dans un tableau.

Exercice 4.1.8 — Les frais de livraison ★★★ Sur machine

Un site de vente calcule les frais de livraison :

- commande de moins de 50 € : 4,90 € de frais ;
- de 50 € à moins de 100 € : 2,50 € ;
- 100 € ou plus : livraison gratuite ;
- un client « premium » a toujours la livraison gratuite.

Le programme lit le montant, puis la réponse à « Client premium ? (o/n) ».

1. Écrivez l'algorithme en pseudo-code. Quel cas faut-il tester en premier ?
2. Écrivez le programme Python. Testez-le avec 60 € pour un client premium, puis 60 € pour un client normal.

```python
montant = float(input("Montant ? "))
premium = input("Client premium ? (o/n) ")
# à compléter
```

Exercice 4.1.9 — Le type de triangle ★★★

Un programme lit les trois côtés `a`, `b`, `c` d'un triangle. Il affiche :

- « équilatéral » si les trois côtés sont égaux ;
- « isocèle » si exactement deux côtés sont égaux ;
- « quelconque » sinon.

Voici la version d'un stagiaire :

```
PSEUDO-CODE
Si a = b OU b = c OU a = c Alors
    Écrire "isocèle"
Sinon si a = b ET b = c Alors
    Écrire "équilatéral"
Sinon
    Écrire "quelconque"
Fin de Si
```

1. Qu'affiche ce programme pour `a = 5`, `b = 5`, `c = 5` ? Est-ce juste ?
2. Expliquez l'erreur.
3. Corrigez l'algorithme.

4.2 — Répéter un nombre connu de fois : la boucle Pour

‹ Revoir le cours 4.2

★☆☆ **Niveau 1 — Appliquer**

Exercice 4.2.1 — La table de multiplication ★☆☆ Sur machine

On veut afficher la table de 7, de 7 × 1 à 7 × 10, une ligne par calcul : `7 x 1 = 7`.

1. Complétez l'algorithme.

```
PSEUDO-CODE
Pour i allant de . . . à . . . Faire
    Écrire 7, "x", i, "=", . . .
Fin de Pour
```

1. Corrigez le programme Python, puis exécutez-le.

```python
PYTHON
for i in range(1, 2):    # à corriger
    print(7, "x", i, "=", 0)    # à compléter
```

Exercice 4.2.2 — Trace d'une boucle Pour ★☆☆

```pseudo
s ← 0
Pour i allant de 1 à 4 Faire
    s ← s + 2 * i
Fin de Pour
Écrire s
```

1. Faites la trace : une ligne par tour, avec `i` et `s`.
2. Qu'affiche l'algorithme ?

Exercice 4.2.3 — Combien de tours ? ★☆☆

Pour chaque boucle, donnez le nombre de tours et les valeurs prises par le compteur.

1. `Pour i allant de 3 à 8 Faire`
2. `for i in range(2, 10):`
3. `for i in range(5):`
4. `Pour k allant de 0 à longueur("BTSSIO") - 1 Faire`

★★☆ Niveau 2 — S'entraîner

Exercice 4.2.4 — Compter une lettre ★★☆

On veut compter combien de fois la lettre `e` apparaît dans le mot `"developpement"`.

1. Complétez l'algorithme.

```pseudo
mot ← "developpement"
compt ← . . .
Pour i allant de 0 à . . . Faire
    Si mot[i] = "e" Alors
        . . .
    Fin de Si
Fin de Pour
Écrire compt
```

1. Que vaut `compt` à la fin ? Donnez les indices des `e`.

Exercice 4.2.5 — Compter les voyelles ★★☆ Sur machine

On veut compter les voyelles d'un mot lu au clavier. Les voyelles sont a, e, i, o, u, y, en minuscules ou en majuscules.

L'idée : on range toutes les voyelles dans une chaîne `voyelles`. Pour chaque caractère du mot, on le cherche dans cette chaîne. `position(voyelles, mot[i])` renvoie -1 si le caractère n'est pas dans `voyelles` : ce n'est pas une voyelle.

1. Complétez les pointillés.

```
PSEUDO-CODE
Lire mot
voyelles ← "aeiouyAEIOUY"
cpt ← . . .
Pour i allant de 0 à . . . Faire
    Si position(voyelles, mot[i]) ≠ -1 Alors
        . . .
    Fin de Si
Fin de Pour
Écrire cpt
```

1. Que donne l'algorithme pour `Algorithmique` ? Listez les voyelles trouvées.
2. Écrivez le programme en Python. Testez-le avec `Algorithmique`, `Python`, `SIO` et `Crypt`.

Exercice 4.2.6 — Le mot à l'envers ★★☆

On veut écrire un mot à l'envers : `"SIO"` devient `"OIS"`.

Idée : on part d'une chaîne vide. À chaque tour, on colle la lettre lue **devant** ce qu'on a déjà.

1. Complétez l'algorithme.

```
PSEUDO-CODE
mot ← "SIO"
envers ← ""
Pour i allant de 0 à longueur(mot) - 1 Faire
    envers ← . . .
Fin de Pour
Écrire envers
```

1. Faites la trace pour `"SIO"`.
2. Que donne l'algorithme pour `"radar"` ? Qu'a ce mot de particulier ?

★★★ **Niveau 3 — Aller plus loin**

Exercice 4.2.7 — Le bit de parité ★★★ Sur machine

Pour détecter une erreur de transmission, on ajoute un bit à la fin d'un code binaire :

- si le code contient un nombre pair de « 1 », on ajoute `"0"` ;
- sinon, on ajoute `"1"`.

1. Écrivez l'algorithme qui part de `code ← "1011001"` et affiche le code complété.
2. Quel est le code complété pour `"1011001"` ? Pour `"1100000"` ?
3. Écrivez-le en Python et testez-le.
4. À l'arrivée, le récepteur vérifie le code de 8 bits : il doit contenir un nombre pair de « 1 ». Il reçoit `"10110011"`. Le code est-il arrivé intact ? Avec quel calcul de votre algorithme le savez-vous ?

Exercice 4.2.8 — Le mot de passe sécurisé ★★★ Sur machine

Un mot de passe est valide s'il a au moins 8 caractères, au moins une majuscule et au moins un chiffre.

Pour « au moins une majuscule », on utilise un **drapeau** : un booléen qui vaut Faux au départ. Dès qu'on trouve une majuscule, il passe à Vrai, et il le reste.

Les caractères se comparent par leur code : les chiffres d'abord, puis les majuscules, puis les minuscules. Les majuscules de `"A"` à `"Z"` se suivent.

Donc un caractère `ch` est une majuscule si `ch ≥ "A" ET ch ≤ "Z"`. C'est un chiffre si `ch ≥ "0" ET ch ≤ "9"`.

1. Complétez l'algorithme.

```
PSEUDO-CODE
Lire m
maj ← Faux
chiffre ← Faux
Pour i allant de 0 à longueur(m) - 1 Faire
    ch ← m[i]
    Si ch ≥ "A" ET ch ≤ "Z" Alors
        . . .
    Fin de Si
    Si . . . Alors
        chiffre ← Vrai
    Fin de Si
Fin de Pour
valide ← longueur(m) ≥ 8 ET . . .
Écrire valide
```

1. Que donne l'algorithme pour `Sio2026!`, `motdepasse1` et `ABCD12` ? Présentez les résultats dans un tableau.
2. Écrivez le programme en Python et testez-le.

Exercice 4.2.9 — Compter les diviseurs ★★★

Un **diviseur** de `n` est un entier `i` tel que `n mod i = 0`. Un nombre premier est un entier qui a exactement deux diviseurs : 1 et lui-même.

1. Écrivez un algorithme qui lit `n`, compte ses diviseurs entre 1 et `n`, puis affiche ce nombre.
2. Faites la trace pour `n = 6`. Que vaut le compteur à la fin ?
3. Ajoutez à la fin un test qui affiche « premier » ou « pas premier ».
4. Que donne l'algorithme pour 13 ?

4.3 — Répéter tant que : les boucles Tant que et Répéter

‹ Revoir le cours 4.3

★☆☆ **Niveau 1 — Appliquer**

Exercice 4.3.1 — Trace d'une boucle Tant que ★☆☆

```
PSEUDO-CODE
n ← 100
tours ← 0
Tant que n > 1 Faire
    n ← n div 3
    tours ← tours + 1
Fin de Tant que
Écrire n, tours
```

1. Faites la trace : une ligne par test de la condition.
2. Qu'affiche l'algorithme ?

Exercice 4.3.2 — Un Pour écrit avec Tant que ★☆☆

```
PSEUDO-CODE
Pour i allant de 1 à 3 Faire
    Écrire i * 10
Fin de Pour
```

1. Réécrivez cette boucle avec un Tant que. N'oubliez ni la valeur de départ de `i`, ni la ligne qui le fait avancer.
2. Faites la trace des deux versions. Affichent-elles la même chose ?

Exercice 4.3.3 — Une note contrôlée ★☆☆ Sur machine

On veut lire une note. Tant qu'elle n'est pas entre 0 et 20, on la redemande.

1. Complétez l'algorithme.

```
PSEUDO-CODE
Répéter
    Écrire "Note sur 20 ?"
    Lire note
Jusqu'à note ≥ 0 ET . . .
Écrire "Note enregistrée :", note
```

1. Complétez la condition du programme Python, puis testez-le en tapant 25, puis -3, puis 14.

PYTHON

```python
while True:
    note = float(input("Note sur 20 ? "))
    if note >= 0:    # à compléter
        break
print("Note enregistrée :", note)
```

★★☆ **Niveau 2 — S'entraîner**

Exercice 4.3.4 — Chercher une lettre ★★☆

Cette fonction cherche la première position d'une lettre dans une chaîne. Si la lettre est absente, elle renvoie la longueur de la chaîne.

PSEUDO-CODE

```
Fonction chercher(lettre, chaine)
    k ← 0
    Tant que k < longueur(chaine) ET chaine[k] ≠ lettre Faire
        k ← k + 1
    Fin de Tant que
    Renvoyer k
Fin de Fonction
```

1. Faites la trace de `chercher("I", "BTSSIO")`. Que renvoie-t-elle ?
2. Que renvoie `chercher("A", "BTSSIO")` ? Pourquoi la boucle s'arrête-t-elle ?
3. Que se passerait-il sans la condition `k < longueur(chaine)` ?

Exercice 4.3.5 — Quotient et reste par soustractions ★★☆ Sur machine

On veut le quotient et le reste de la division de `a` par `b` sans utiliser `div` ni `mod`. Idée : on enlève `b` à `a` tant que c'est possible, et on compte les soustractions.

1. Complétez l'algorithme.

```
PSEUDO-CODE
Lire a
Lire b
q ← 0
Tant que a ≥ . . . Faire
    a ← . . .
    q ← . . .
Fin de Tant que
Écrire "quotient :", q, "reste :", a
```

1. Faites la trace pour `a = 47` et `b = 15`.
2. Écrivez le programme en Python. Vérifiez avec `47 // 15` et `47 % 15`.

Exercice 4.3.6 — Le disque qui se remplit ★★☆

Un serveur stocke 1 Go de journaux le premier jour. Chaque jour, la taille des journaux double : c'est la suite $t_{n+1} = 2 \times t_n$, avec $t_1 = 1$. Le disque fait 1 000 Go.

1. Complétez l'algorithme qui cherche le premier jour où la taille dépasse 1 000 Go.

```
PSEUDO-CODE
taille ← 1
jours ← 1
Tant que taille ≤ . . . Faire
    taille ← . . .
    jours ← jours + 1
Fin de Tant que
Écrire jours, taille
```

1. Qu'affiche-t-il ?
2. Pourquoi une boucle Pour ne convient-elle pas ici ?

★★★ Niveau 3 — Aller plus loin

Exercice 4.3.7 — La boucle qui ne s'arrête pas ★★★

Un stagiaire veut afficher les nombres de 10 à 1, en descendant. Il a écrit :

```
PSEUDO-CODE
n ← 10
Tant que n ≥ 0 Faire
    Écrire n
    n ← n + 1
Fin de Tant que
```

1. Que se passe-t-il quand on lance cet algorithme ?
2. Cet algorithme contient **deux** erreurs. Trouvez-les et corrigez-les.
3. Avec la version corrigée, combien de fois la condition est-elle testée ? Quelle est la valeur de `n` après la boucle ?
4. Quelle valeur limite faut-il vérifier pour repérer la deuxième erreur ?

Exercice 4.3.8 — Le nombre mystère ★★★ Sur machine

Le programme cache le nombre 42. L'utilisateur propose des nombres.

Le programme répond « Plus grand », « Plus petit », ou « Bravo ». Il compte les essais.

1. Quelle boucle choisir ? Pourquoi ?
2. Écrivez l'algorithme en pseudo-code. Ajoutez un en-tête : nom, rôle, entrées, sortie. Ajoutez un commentaire pour chaque variable. Le programme demande un en-tête et des commentaires : ils rendent la copie lisible pour le correcteur.
3. Écrivez le programme en Python, avec les mêmes commentaires, et testez-le avec 50, puis 30, puis 42.

```
PYTHON
secret = 42
essais = 0
# à compléter
```

Exercice 4.3.9 — Le PGCD par soustractions ★★★

Le **PGCD** de deux entiers, pour « plus grand commun diviseur », est le plus grand entier qui divise les deux. Une méthode ancienne le calcule par soustractions : tant que les deux nombres sont différents, on enlève le plus petit au plus grand. Quand ils sont égaux, ils valent le PGCD.

1. Écrivez l'algorithme en pseudo-code. Il faut un `Si` dans la boucle.
2. Faites la trace pour `a = 84` et `b = 36`. Quel est le PGCD ?
3. Pourquoi la boucle s'arrête-t-elle toujours, avec deux entiers positifs ?

4.4 — Construire une boucle de calcul

‹ Revoir le cours 4.4

★☆☆ **Niveau 1 — Appliquer**

Exercice 4.4.1 — Les trois temps ★☆☆

Voici un algorithme qui calcule la moyenne de 4 notes lues au clavier.

```
PSEUDO-CODE
total ← 0
Pour i allant de 1 à 4 Faire
    Lire note
    total ← total + note
Fin de Pour
moyenne ← total / 4
Écrire "Moyenne :", arrondi(moyenne, 2)
```

1. Quelle ligne est la préparation ?
2. Quelles lignes forment l'itération ?
3. Quelles lignes forment la mise en forme finale ?
4. Quel est l'accumulateur ? Pourquoi part-il de 0 ?

Exercice 4.4.2 — La somme des entiers ★☆☆

On veut calculer $\sum_{i=1}^{10} i$, la somme des entiers de 1 à 10.

1. Complétez l'algorithme.

```
PSEUDO-CODE
s ← . . .
Pour i allant de 1 à . . . Faire
    s ← . . .
Fin de Pour
Écrire s
```

1. Quel résultat obtenez-vous ?

Exercice 4.4.3 — La factorielle ★☆☆

Le produit $\prod_{i=1}^{6} i = 1 \times 2 \times 3 \times 4 \times 5 \times 6$ s'appelle la factorielle de 6.

1. Complétez l'algorithme.

```
PSEUDO-CODE
p ← . . .
Pour i allant de 1 à 6 Faire
    p ← . . .
Fin de Pour
Écrire p
```

1. Quel résultat obtenez-vous ?
2. Que se passerait-il avec `p ← 0` au départ ?

★★☆ Niveau 2 — S'entraîner

Exercice 4.4.4 — La moyenne de n notes ★★☆ Sur machine

Le programme lit d'abord le nombre de notes `n`, puis les `n` notes. Il affiche la moyenne arrondie à 2 décimales.

1. Écrivez l'algorithme en pseudo-code. Repérez les trois temps.
2. Complétez le programme Python et testez-le avec 3 notes : 12, 15 et 10.

```
PYTHON
n = int(input("Nombre de notes ? "))
total = 0
# à compléter
```

Exercice 4.4.5 — La somme des impairs ★★☆

On veut calculer $S = \sum_{i=1}^{n} (2i - 1)$. On lit : somme des $2i - 1$, pour i allant de 1 à n.

1. Écrivez l'algorithme en pseudo-code.
2. Calculez S pour n = 5 : écrivez la valeur de S à la fin de chaque tour.
3. Quelle propriété de S est vraie à la fin de chaque tour ? Pensez à l'encadré « Construire la boucle avec une propriété ».
4. Calculez S pour n = 10.

Exercice 4.4.6 — Simuler 600 lancers de dé ★★☆ Sur machine

Un dé bien équilibré donne chaque face avec la même chance : une fois sur six, soit environ 0,167. On veut le vérifier en simulant 600 lancers.

`random.randint(1, 6)` tire un entier au hasard entre 1 et 6. `random.seed(1)` fixe le point de départ du hasard : chacun obtient alors les mêmes tirages, et on peut comparer les résultats.

1. Repérez les trois temps : quelle préparation, quelle mise à jour dans la boucle, quelle mise en forme finale ?
2. Complétez le programme, puis exécutez-le. Combien de 6 sortent ? Quelle est leur fréquence, arrondie au millième ?
3. Remplacez `random.seed(1)` par `random.seed(2)` : le résultat change-t-il ? Reste-t-il proche de 0,167 ?

```python
import random
random.seed(1)
nb6 = 0
for i in range(1, 601):
    de = random.randint(1, 6)
    # à compléter : compter les 6
print(nb6, round(nb6 / 600, 3))
```

★★★ Niveau 3 — Aller plus loin

Exercice 4.4.7 — La puissance sans le signe ^ ★★★

On veut calculer a^n en multipliant `a` par lui-même, sans utiliser `^`.

1. Complétez l'écriture de la puissance sous la forme d'un Π : $a^n = \prod_{i=1}^{n} \ldots$ Chaque facteur du produit est le même nombre.
2. Écrivez l'algorithme qui lit `a` et `n`, puis affiche a^n.
3. Faites la trace pour `a = 3` et `n = 4`.
4. Que donne votre algorithme pour `n = 0` ? Est-ce juste ?

Exercice 4.4.8 — Une somme qui approche π ★★★ Sur machine

La somme $\sum_{k=1}^{n} \frac{1}{k^2}$ s'approche de $\frac{\pi^2}{6} \approx 1{,}6449$ quand n grandit.

1. Écrivez la somme en boucle, en Python.
2. Calculez-la pour n = 10, n = 100 et n = 1000. Arrondissez à 4 décimales.
3. Pourquoi faut-il écrire `1 / k ** 2` et pas `1 / k * 2` ?

```python
n = 10
s = 0
# à compléter
print(round(s, 4))
```

Exercice 4.4.9 — La température la plus haute ★★★

Une sonde relève 5 températures, une par jour, du jour 1 au jour 5. Le programme les lit une par une. Il affiche la plus haute, et le jour où elle a été relevée.

1. Un collègue propose de partir de `maxi ← 0`. Pourquoi est-ce une erreur ? Donnez un jeu de températures qui le montre.
2. Écrivez l'algorithme. Quelles sont les deux variables à mettre à jour quand on trouve une température plus haute ?
3. Faites la trace avec -3, 5, 2, 8, -1.
4. Si deux jours ont la même température la plus haute, quel jour votre algorithme affiche-t-il ? Que faudrait-il changer pour afficher l'autre ?

4.5 — Les structures imbriquées

‹ Revoir le cours 4.5

★☆☆ **Niveau 1 — Appliquer**

Exercice 4.5.1 — Compter les passages ★☆☆

```pseudo-code
compt ← 0
Pour i allant de 1 à 4 Faire
    Pour j allant de 1 à 3 Faire
        compt ← compt + 1
    Fin de Pour
Fin de Pour
Écrire compt
```

1. Combien de fois la ligne `compt ← compt + 1` est-elle exécutée ?
2. Donnez les trois premiers couples (i, j), dans l'ordre.
3. Que vaut `j` quand `i` passe de 1 à 2 ?

Exercice 4.5.2 — Le rectangle d'étoiles ★☆☆ Sur machine

On veut afficher un rectangle de 3 lignes et 5 étoiles par ligne.

1. Complétez l'algorithme.

```
PSEUDO-CODE
Pour i allant de 1 à . . . Faire
    ligne ← ""
    Pour j allant de 1 à . . . Faire
        ligne ← ligne + "*"
    Fin de Pour
    Écrire ligne
Fin de Pour
```

1. Complétez le programme Python et exécutez-le.

```
PYTHON
for i in range(1, 2):    # à corriger
    ligne = ""
    for j in range(1, 2):    # à corriger
        ligne = ligne + "*"
    print(ligne)
```

Exercice 4.5.3 — Les multiples de 3 ★☆☆

On veut compter les multiples de 3 entre 1 et 20.

1. Complétez l'algorithme.

```
PSEUDO-CODE
compt ← 0
Pour i allant de 1 à 20 Faire
    Si . . . Alors
        compt ← compt + 1
    Fin de Si
Fin de Pour
Écrire compt
```

1. Que vaut `compt` à la fin ? Donnez la liste des nombres comptés.

★★☆ **Niveau 2 — S'entraîner**

Exercice 4.5.4 — Les paires de dés ★★☆

On lance deux dés à 6 faces. On veut compter les tirages dont la somme vaut 7.

1. Complétez l'algorithme.

PSEUDO-CODE

```
compt ← 0
Pour i allant de 1 à 6 Faire
    Pour j allant de 1 à 6 Faire
        Si . . . Alors
            . . .
        Fin de Si
    Fin de Pour
Fin de Pour
Écrire compt
```

1. Combien de fois le test `Si` est-il évalué en tout ?
2. Que vaut `compt` à la fin ? Donnez la liste des tirages.

Exercice 4.5.5 — La grille de multiplication ★★☆ Sur machine

On veut afficher la grille des produits de 1 à 4, une ligne par valeur de `i` :

CE QUI S'AFFICHE

```
1 2 3 4
2 4 6 8
3 6 9 12
4 8 12 16
```

1. Écrivez l'algorithme avec deux boucles imbriquées. Chaque ligne est une chaîne qu'on construit par concaténation.
2. Écrivez le programme Python. Il faut `str(...)` pour coller un nombre à une chaîne.

PYTHON

```python
for i in range(1, 5):
    ligne = ""
    # à compléter
    print(ligne)
```

Exercice 4.5.6 — Le triangle de nombres ★★☆

```
PSEUDO-CODE
total ← 0
Pour i allant de 1 à 4 Faire
    ligne ← ""
    Pour j allant de 1 à i Faire
        ligne ← ligne + chaîne(j) + " "
        total ← total + 1
    Fin de Pour
    Écrire ligne
Fin de Pour
Écrire total
```

1. Qu'affiche cet algorithme ? Écrivez les lignes.
2. Combien de fois la boucle intérieure tourne-t-elle quand `i` vaut 3 ?
3. Que vaut `total` à la fin ?

★★★ Niveau 3 — Aller plus loin

Exercice 4.5.7 — Encore des dés ★★★

On reprend les deux dés à 6 faces de l'exercice 4.5.4.

1. Écrivez un seul algorithme, avec une seule double boucle et deux compteurs. Le premier compte les tirages où le premier dé est plus grand que le second. Le second compte les tirages dont la somme est au moins 10.
2. Donnez les deux résultats.
3. Sans ordinateur, expliquez pourquoi le premier compteur vaut 15. Indice : il y a 36 tirages, dont 6 doubles.
4. Combien de tirages vérifient les deux conditions à la fois ? Quelle ligne ajouteriez-vous pour les compter ?

Exercice 4.5.8 — Le produit de deux matrices ★★★ Sur machine

On veut calculer $C = A \times B$ avec

$$A = \begin{pmatrix} 2 & 1 \\ 0 & 3 \end{pmatrix} \qquad B = \begin{pmatrix} 1 & 4 \\ 2 & 5 \end{pmatrix}$$

La formule, donnée comme à l'examen : $c_{ij} = \sum_{k=1}^{2} a_{ik} \times b_{kj}$.

Une matrice s'écrit en Python comme une liste de lignes : `A = [[2, 1], [0, 3]]`. `A[i][k]` est le nombre de la ligne `i`, colonne `k`, et les indices partent de 0. La ligne qui applique la formule vous est donnée : `s = s + A[i][k] * B[k][j]`.

1. Calculez à la main c_{11}, le nombre en haut à gauche de C.
2. Exécutez le programme ci-dessous : le résultat est faux. Corrigez la borne de la boucle sur `k`.
3. Le résultat est encore faux : la ligne `s = 0` est mal placée. Déplacez-la au bon endroit, et expliquez pourquoi.

PYTHON

```
A = [[2, 1], [0, 3]]
B = [[1, 4], [2, 5]]
C = [[0, 0], [0, 0]]
s = 0                        # à déplacer au bon endroit
for i in range(2):
    for j in range(2):
        for k in range(1):    # à corriger : la borne
            s = s + A[i][k] * B[k][j]
        C[i][j] = s
print(C)
```

Exercice 4.5.9 — Les nombres premiers jusqu'à 30 ★★★

On veut afficher tous les nombres premiers de 2 à 30. Un nombre est premier s'il a exactement deux diviseurs : 1 et lui-même.

1. Écrivez l'algorithme avec deux boucles imbriquées : la boucle extérieure choisit `n`, la boucle intérieure compte ses diviseurs.
2. La boucle intérieure dépend-elle de la boucle extérieure ? Expliquez.
3. Quels nombres sont affichés ?
4. Combien de tests de reste votre algorithme fait-il pour le seul `n = 30` ? Comment en faire beaucoup moins ? Relisez l'encadré « Combien de tours ? ».

5.1 — Les tableaux à une dimension

‹ Revoir le cours 5.1

★☆☆ **Niveau 1 — Appliquer**

Exercice 5.1.1 — Lire et écrire dans un tableau ★☆☆

On a `T ← [4, 9, 2, 7, 5]`.

1. Que valent `T[0]`, `T[4]` et `longueur(T)` ?
2. On exécute `T[2] ← T[1] + T[3]`. Écrivez le tableau obtenu.
3. On exécute ensuite `T[0] ← T[0] * 10`. Écrivez le tableau obtenu.
4. Que se passe-t-il avec `T[5] ← 1` ?

Exercice 5.1.2 — La trace d'un parcours ★☆☆

Le tableau `stock` donne le nombre de câbles dans chacun des 4 tiroirs d'une armoire.

```
PSEUDO-CODE
stock ← [12, 0, 7, 3]
total ← 0
Pour i allant de 0 à longueur(stock) - 1 Faire
    total ← total + stock[i]
Fin de Pour
Écrire total
```

1. Combien de tours fait la boucle ?
2. Faites la trace avec les colonnes `i`, `stock[i]` et `total`.
3. Qu'affiche l'algorithme ?

Exercice 5.1.3 — Copie ou deuxième nom ? ★☆☆

Lisez ce programme sans l'exécuter.

```python
a = [5, 6, 7]
b = a
c = a.copy()
b[1] = 0
print(a, b, c)
```

1. Après `b = a`, combien de tableaux existent en mémoire ?
2. Que contient `a` à la fin ?
3. Que contient `c` à la fin ?
4. Écrivez en pseudo-code une copie de `a` élément par élément, sans `.copy()`.

★★☆ **Niveau 2 — S'entraîner**

Exercice 5.1.4 — Compter les notes suffisantes ★★☆ Sur machine

Un professeur veut savoir combien d'élèves ont au moins 10.

1. Écrivez l'algorithme en pseudo-code, avec un compteur `nb`.
2. Complétez le programme Python et exécutez-le.
3. Modifiez-le pour afficher aussi le nombre de notes en dessous de 10, sans écrire une deuxième boucle.

```python
notes = [12, 8, 15, 9, 10, 17, 6]
nb = 0
# à compléter : la boucle
print("Au moins 10 :", nb)
```

Exercice 5.1.5 — La température la plus basse ★★☆

Un capteur relève la température d'une salle serveur pendant 5 nuits : `temp ← [-2, 3, -5, 1, -4]`.

1. On veut la plus petite valeur. Quelle valeur de départ donner à `mini` ?
2. Écrivez l'algorithme en pseudo-code.
3. Faites la trace. Que vaut `mini` à la fin ?
4. Un camarade part de `mini ← 0`. Quel résultat obtient-il avec le tableau `[4, 7, 2]` ? Pourquoi est-ce faux ?

Exercice 5.1.6 — Construire un tableau ★★☆ Sur machine

On veut le tableau des carrés des entiers de 1 à 6 : `[1, 4, 9, 16, 25, 36]`.

1. On part d'un tableau vide `carres ← []`. Quelle instruction ajoute une valeur à la fin ?
2. Écrivez l'algorithme avec une boucle Pour.
3. Complétez le programme Python. Affichez ensuite la longueur du tableau.

```python
carres = []
# à compléter
print(carres)
```

★★★ **Niveau 3 — Aller plus loin**

Exercice 5.1.7 — Trier un tableau ★★★

On veut ranger le tableau `T ← [5, 2, 8, 1, 4]` dans l'ordre croissant. On utilise le **tri par sélection** :

- au tour 0, on cherche l'indice du plus petit élément de tout le tableau, et on l'échange avec la case 0 ;
- au tour 1, on cherche le plus petit parmi les cases 1 à 4, et on l'échange avec la case 1 ;
- et ainsi de suite, jusqu'à l'avant-dernière case.

1. Combien de tours faut-il pour un tableau de 5 éléments ?
2. Pour échanger `T[i]` et `T[imin]`, pourquoi faut-il une variable `temp` ?
3. Complétez l'algorithme.
4. Écrivez le tableau à la fin de chaque tour.

```
PSEUDO-CODE
Pour i allant de 0 à longueur(T) - 2 Faire
    imin ← i
    Pour j allant de i + 1 à longueur(T) - 1 Faire
        Si T[j] < T[imin] Alors
            imin ← . . .
        Fin de Si
    Fin de Pour
    temp ← T[i]
    T[i] ← . . .
    T[imin] ← . . .
Fin de Pour
```

Exercice 5.1.8 — La moyenne sans les valeurs aberrantes ★★★

Sur machine

Un test en ligne enregistre des notes sur 20. Certaines sont des erreurs de saisie : 25, -5, 100.

Ce sont des **valeurs aberrantes** : des valeurs impossibles, qu'on ne doit pas compter. On ne garde que les notes entre 0 et 20 inclus.

Le programme doit :

- calculer la moyenne des notes gardées, arrondie à 2 décimales ;

- afficher -1 si aucune note n'est gardée. Une moyenne ne vaut jamais -1 : on comprend qu'il y a un problème.

1. Écrivez l'algorithme en pseudo-code.
2. Remplissez le tableau des tests, avec le résultat attendu.
3. Complétez le programme Python et vérifiez vos résultats.
4. Modifiez-le pour afficher aussi le nombre de notes gardées.

Tableau	Notes gardées	Somme	Nombre	Résultat attendu
`[10, 12, 18]`				
`[25, -5, 15]`				
`[30, -2, 100]`				
`[8, 9, 12, 21]`				

PYTHON

```
arr = [10, 12, 18]
s = 0
k = 0
# à compléter
```

Exercice 5.1.9 — Le tableau est-il rangé ? ★★★

Un tableau est **trié dans l'ordre croissant** si chaque élément est plus petit ou égal au suivant. Exemple : `[2, 5, 5, 9]` est trié, `[2, 7, 4]` ne l'est pas.

1. Pour un tableau de 5 éléments, combien de paires de voisins faut-il comparer ?
2. Écrivez un algorithme qui range Vrai ou Faux dans le booléen `trie`. Idée : on part de `trie ← Vrai`, et on passe à Faux dès qu'une paire est mal rangée.
3. Améliorez-le avec une boucle Tant que, qui s'arrête dès qu'une paire est mal rangée.
4. Que donnent vos deux versions sur `[2, 5, 5, 9]`, `[2, 7, 4]` et `[8]` ?
5. Pour `[9, 1, 2, 3, 4]`, combien de comparaisons fait la version Pour ? Et la version Tant que ? Quelle version est la plus efficace ?

5.2 — Les tableaux à deux dimensions

‹ Revoir le cours 5.2

★☆☆ **Niveau 1 — Appliquer**

Exercice 5.2.1 — Lire une grille ★☆☆

Un tableau donne le nombre de tickets ouverts au service informatique, par jour (une ligne) et par étage (une colonne).

PYTHON

```
tickets = [[3, 0, 5],
           [1, 4, 2]]
```

1. Combien de lignes et combien de colonnes ?
2. Que vaut `tickets[1][0]` ? Et `tickets[0][2]` ?
3. Quelle case contient 4 ? Donnez la ligne et la colonne.

Exercice 5.2.2 — Construire une grille ★☆☆

On veut une grille `G` de 2 lignes et 3 colonnes, remplie de `0`.

PSEUDO-CODE

```
G ← []
Pour i allant de 0 à . . . Faire
    ligne ← []
    Pour j allant de 0 à . . . Faire
        ajouter(ligne, 0)
    Fin de Pour
    ajouter(G, . . .)
Fin de Pour
```

1. Complétez les trois pointillés.
2. On exécute ensuite `G[1][0] ← 5`. Écrivez la grille obtenue.
3. Pourquoi `G[2][0] ← 1` est-il une erreur ?

Exercice 5.2.3 — Un tableau de stagiaires ★☆☆

Chaque ligne décrit un stagiaire : son nom, son âge, et s'il est présent aujourd'hui.

PYTHON

```
stagiaires = [["Sarah", 22, True],
              ["Tom", 19, False],
              ["Inès", 20, True]]
```

1. Quel est le type de chaque colonne ?
2. Que vaut `stagiaires[2][0]` ? Et `stagiaires[1][2]` ?

3. Quelle expression donne l'âge de Sarah ?

★★☆ Niveau 2 — S'entraîner

Exercice 5.2.4 — Les ventes par magasin ★★☆

Une chaîne a 3 magasins. Le tableau `ventes` donne le nombre d'ordinateurs vendus par magasin (une ligne) et par trimestre (une colonne).

```python
ventes = [[120, 95, 130, 110],
          [80, 105, 90, 100],
          [150, 140, 160, 155]]
```

1. Que vaut `ventes[1][2]` ? Que représente-t-il ?
2. Écrivez, en pseudo-code, une boucle qui calcule le total du magasin d'indice 0, le premier, sur l'année. Quel indice varie ?
3. Écrivez, en pseudo-code, une boucle qui calcule le total du trimestre d'indice 2, le troisième, tous magasins. Quel indice varie ?
4. Écrivez, en pseudo-code, deux boucles imbriquées qui calculent le total général.

Exercice 5.2.5 — La moyenne de chaque stagiaire ★★☆ Sur machine

Le tableau `notes` a une ligne par stagiaire et une colonne par module.

```python
notes = [[12, 15, 9, 18],
         [8, 11, 14, 10],
         [16, 13, 17, 15]]
for i in range(3):
    s = 0
    # à compléter : additionner la ligne i
    print("Stagiaire", i, ":", s / 4)
```

1. Pour la moyenne du stagiaire `i`, quel indice varie dans la boucle intérieure ?
2. Complétez le programme et exécutez-le.
3. Que se passe-t-il si on déplace `s = 0` avant la boucle sur `i` ?

Exercice 5.2.6 — Qui est présent ? ★★☆ Sur machine

On reprend le tableau des stagiaires : nom, âge, présent.

```python
stagiaires = [["Sarah", 22, True],
              ["Tom", 19, False],
              ["Inès", 20, True],
              ["Yanis", 21, True]]
nb = 0
for ligne in stagiaires:
    # à compléter : remplacez la ligne suivante par votre code
    nb = nb + 0
print("Présents :", nb)
```

1. Dans la boucle, quelle case dit si le stagiaire est présent ?
2. Complétez la boucle : affichez le nom de chaque présent, et comptez-les dans `nb`.
3. Modifiez le programme pour afficher aussi l'âge moyen des présents.

★★★ Niveau 3 — Aller plus loin

Exercice 5.2.7 — Le serveur le plus relié ★★★

Trois serveurs sont numérotés 0, 1 et 2. Dans la grille `L`, `L[i][j]` vaut 1 si le serveur `i` envoie des données au serveur `j`, et 0 sinon.

```python
L = [[0, 1, 1],
     [0, 0, 1],
     [1, 0, 0]]
```

Le **degré sortant** d'un serveur est le nombre de serveurs qui reçoivent ses données.

1. Lisez la grille : à quels serveurs le serveur 0 envoie-t-il des données ? Quel est son degré sortant ?
2. Écrivez un algorithme qui calcule le degré sortant du serveur `s`. Quel indice varie ?
3. Écrivez un algorithme qui trouve le serveur qui a le plus grand degré sortant.

Exercice 5.2.8 — Les places libres ★★★ Sur machine

Le plan d'une salle de formation a 3 rangs de 4 places. Une place vaut 1 si elle est prise, 0 si elle est libre.

```python
salle = [[1, 0, 1, 1],
         [0, 0, 1, 0],
         [1, 1, 1, 1]]
# à compléter
```

1. Écrivez un programme qui affiche chaque place libre sous la forme `rang 1 place 3`, en indices.
2. Ajoutez le nombre total de places libres.
3. Ajoutez, pour chaque rang, le nombre de places libres. Où remettre le compteur du rang à `0` ?

Exercice 5.2.9 — Qui peut joindre qui ? ★★★ Sur machine

Quatre serveurs sont numérotés de `0` à 3. Certains envoient des données à d'autres. On dessine chaque envoi par une flèche, appelée un **arc** : un arc de `i` vers `j` veut dire « le serveur `i` envoie au serveur `j` ».

On range ces arcs dans une grille `L`, appelée **matrice d'adjacence**. `L[i][j]` vaut 1 s'il y a un arc de `i` vers `j`, et `0` sinon.

```python
L = [[0, 1, 0, 0],
     [0, 0, 1, 1],
     [1, 0, 0, 0],
     [0, 0, 1, 0]]
```

Les **voisins** d'un serveur sont les serveurs qui reçoivent directement ses données.

1. Quels sont les voisins du serveur 1 ? Lisez sa ligne.
2. Écrivez un algorithme qui affiche les voisins du serveur `s`.
3. Une donnée part du serveur `0` et fait exactement deux arcs. Sur quels serveurs peut-elle arriver ? Trouvez-les à la main.
4. Écrivez un algorithme qui affiche les serveurs atteints en deux arcs depuis `s`. Idée : pour chaque voisin `v` de `s`, on affiche les voisins de `v`.
5. Programmez-le en Python, et testez-le pour `s` = `0` et `s` = 1.

5.3 — Les fonctions et les procédures

‹ Revoir le cours 5.3

★☆☆ **Niveau 1 — Appliquer**

Exercice 5.3.1 — Que renvoie la fonction ? ★☆☆

```
PSEUDO-CODE
Fonction carre(n)
    Renvoyer n * n
Fin de Fonction
```

1. Quel est le paramètre de la fonction ?
2. Que renvoie `carre(4)` ?
3. Que vaut `carre(3) + carre(4)` ?
4. Que vaut `carre(carre(2))` ?

Exercice 5.3.2 — Fonction ou procédure ? ★☆☆

Pour chaque besoin, dites s'il faut une fonction ou une procédure. Justifiez en une phrase.

1. Calculer la TVA d'un prix, pour l'utiliser ensuite dans un total.
2. Afficher un menu de 4 choix à l'écran.
3. Dire si un mot de passe est assez long, Vrai ou Faux.
4. Afficher le bulletin d'un stagiaire, ligne par ligne.

Exercice 5.3.3 — Le plus grand de deux nombres ★☆☆

Complétez la fonction `plus_grand(a, b)`. Elle renvoie le plus grand des deux nombres.

```
PSEUDO-CODE
Fonction plus_grand(a, b)
    Si a > b Alors
        Renvoyer . . .
    Sinon
        Renvoyer . . .
    Fin de Si
Fin de Fonction
```

Que renvoient `plus_grand(3, 8)` et `plus_grand(5, 5)` ?

★★☆ **Niveau 2 — S'entraîner**

Exercice 5.3.4 — Le plus petit et le plus grand ★★☆ Sur machine

Une application de suivi relève les temps de réponse d'un serveur, en millisecondes.

1. Écrivez la fonction `mini_maxi(T)` en pseudo-code. Elle renvoie deux résultats : la plus petite et la plus grande valeur du tableau.
2. Écrivez l'appel qui récupère les deux résultats dans `a` et `b`, en pseudo-code.
3. Complétez la version Python et testez-la sur `[48, 120, 35, 90]`.

PYTHON

```python
def mini_maxi(T):
    mini = T[0]
    maxi = T[0]
    # à compléter
    return mini, maxi

a, b = mini_maxi([48, 120, 35, 90])
print("Plus petit :", a, "Plus grand :", b)
```

Exercice 5.3.5 — Combien de fois ? ★★☆

Un journal de connexion liste les codes d'erreur d'une journée :
`codes ← [404, 200, 404, 500, 404, 200]`.

1. Écrivez la fonction `compter(T, x)`. Elle renvoie le nombre de fois où la valeur `x` apparaît dans le tableau `T`.
2. Que renvoient `compter(codes, 404)` et `compter(codes, 301)` ?
3. Écrivez la ligne qui affiche « Erreurs 404 : » suivi du résultat.

Exercice 5.3.6 — La table de multiplication ★★☆ Sur machine

1. Écrivez la procédure `afficher_table(n)`. Elle affiche la table de `n`, de 1 à 10, sous la forme `7 x 3 = 21`.
2. Appelez-la pour 7, puis pour 9.
3. Un camarade écrit une **fonction** qui renvoie une seule ligne de la table. Pourquoi une procédure convient-elle mieux ici ?

```python
def afficher_table(n):
    # à compléter : remplacez la ligne suivante par votre boucle
    print("table de", n)

afficher_table(7)
```

★★★ Niveau 3 — Aller plus loin

Exercice 5.3.7 — Le bulletin ★★★

On veut afficher la mention de chaque stagiaire.

- La fonction `mention(moy)` renvoie une mention selon la moyenne :
  - `"Très bien"` à partir de 16 ;
  - `"Bien"` à partir de 14 ;
  - `"Assez bien"` à partir de 12 ;
  - `"Passable"` à partir de 10 ;
  - `"Insuffisant"` en dessous.
- La procédure `bulletin(noms, moyennes)` affiche, pour chaque stagiaire, son nom, sa moyenne et sa mention. Elle appelle `mention`.

1. Écrivez la fonction `mention`. Dans quel ordre faut-il tester les seuils ?
2. Écrivez la procédure `bulletin`. Les deux tableaux ont la même longueur : la case `i` de `noms` va avec la case `i` de `moyennes`.
3. Que s'affiche-t-il pour `noms ← ["Léa", "Hugo", "Sarah"]` et `moyennes ← [14.5, 9, 16]` ?
4. Écrivez la version Python **avec un en-tête et des commentaires** : pour chaque sous-programme, son nom, son rôle, ses entrées et sa sortie.

Exercice 5.3.8 — Trouvé, et où ? ★★★

On veut la fonction `chercher(T, x)`. Elle renvoie **deux** valeurs : un booléen qui dit si `x` est dans le tableau `T`, et l'indice de la première case qui contient `x`. Si `x` est absent, l'indice renvoyé vaut `longueur(T)`.

1. Pourquoi une boucle Tant que convient-elle mieux qu'une boucle Pour ?
2. Complétez la fonction.
3. Dans le programme principal, comment récupérer les deux résultats ? Écrivez l'appel pour `T ← [7, 3, 9, 3]` et `x ← 3`.
4. Que renvoie `chercher([7, 3, 9, 3], 5)` ?

PSEUDO-CODE

```
Fonction chercher(T, x)
    k ← 0
    Tant que k < longueur(T) ET . . . Faire
        k ← k + 1
    Fin de Tant que
    trouve ← . . .
    Renvoyer (trouve, k)
Fin de Fonction
```

Exercice 5.3.9 — Convertir des durées ★★★ Sur machine

Un serveur note la durée de chaque sauvegarde en secondes : `[7384, 59, 3600]`.

1. Écrivez la fonction `hms(duree)`. Elle renvoie trois valeurs : les heures, les minutes et les secondes.
2. Écrivez un programme qui affiche chaque durée sous la forme `2 h 3 min 4 s`.
3. Ajoutez le total de toutes les sauvegardes, converti lui aussi.
4. Écrivez la fonction `secondes(h, m, s)`, qui fait le calcul inverse : elle renvoie la durée en secondes. Vérifiez qu'elle redonne 7384 pour 2 h 3 min 4 s.

PYTHON

```python
def hms(duree):
    # à compléter
    return 0, 0, 0

for d in [7384, 59, 3600]:
    h, m, s = hms(d)
    print(h, "h", m, "min", s, "s")
```

5.4 — Paramètres et variables locales

‹ Revoir le cours 5.4

★☆☆ **Niveau 1 — Appliquer**

Exercice 5.4.1 — Locale ou globale ? ★☆☆

```python
taux = 20

def prix_ttc(ht):
    tva = ht * taux / 100
    return ht + tva

prix = prix_ttc(50)
print(prix)
```

1. Classez chaque variable : `taux`, `tva`, `ht`, `prix`. Est-elle locale ou globale ?
2. Qu'affiche le programme ?
3. Peut-on écrire `print(tva)` à la dernière ligne ?

Exercice 5.4.2 — Dans quel sens passent les données ? ★☆☆

Pour chaque fonction ou procédure, dites si le paramètre est en **entrée seule**, en **sortie seule** ou en **entrée et sortie**.

1. `longueur_mot(mot)` : elle renvoie le nombre de caractères du mot.
2. `augmenter(T)` : elle ajoute 1 à chaque case du tableau `T`.
3. `moyenne(T)` : elle renvoie la moyenne des valeurs de `T`.
4. `remplir(T, n)` : elle reçoit un tableau vide `T` et y met n zéros, sans lire ce qu'il contenait. Donnez le sens de `T` et celui de `n`.
5. `ajouter_note(T, n)` : elle ajoute la note `n` à la fin du tableau `T`. Donnez le sens de `T` et celui de `n`.

Exercice 5.4.3 — Deux variables, un seul nom ★☆☆

Lisez ce programme sans l'exécuter.

```python
x = 10

def f():
    x = 5
    return x * 2

y = f()
print(x, y)
```

1. Combien de variables s'appellent `x` ?
2. Que vaut `y` ?
3. Qu'affiche le programme ?

★★☆ Niveau 2 — S'entraîner

Exercice 5.4.4 — Qui modifie quoi ? ★★☆

Lisez ce programme sans l'exécuter.

```python
total = 0

def ajouter(T, x):
    T.append(x)
    total = len(T)
    return total

notes = [12, 15]
n = ajouter(notes, 9)
copie = notes.copy()
copie.append(20)
print(notes, n, total, copie)
```

1. Pour le paramètre `T`, le transfert est-il en entrée seule ou en entrée et sortie ? Et pour `x` ?
2. Que vaut `n` ?
3. Pourquoi la variable `total` affichée vaut-elle encore 0 ?
4. Ajouter 20 à `copie` change-t-il `notes` ?
5. Écrivez ce qu'affiche la dernière ligne, puis vérifiez.

Exercice 5.4.5 — Modifier ou rendre une copie ? ★★☆ Sur machine

On veut ajouter 2 points de bonus à chaque note, sans dépasser 20.

- Version A : la procédure `bonus_sur_place(T)` modifie le tableau qu'on lui donne.
- Version B : la fonction `avec_bonus(T)` renvoie un **nouveau** tableau, et laisse `T` intact.

1. Pour chaque version, le paramètre `T` est-il en entrée seule ou en entrée et sortie ?
2. Complétez les deux versions.
3. Testez avec `[12, 19, 8]`. Affichez le tableau de départ après chaque appel.

PYTHON

```python
def bonus_sur_place(T):
    # à compléter : remplacez la ligne suivante par votre boucle
    T[0] = T[0]

def avec_bonus(T):
    R = T.copy()
    # à compléter
    return R

notes = [12, 19, 8]
nouvelles = avec_bonus(notes)
print(notes, nouvelles)
bonus_sur_place(notes)
print(notes)
```

Exercice 5.4.6 — Une fonction qui dépend d'une globale ★★☆

Un collègue a écrit ce programme.

PYTHON

```python
remise = 10

def prix_final(prix):
    return prix - prix * remise / 100

print(prix_final(80))
remise = 30
print(prix_final(80))
```

1. Qu'affichent les deux lignes `print` ?
2. Pourquoi le même appel `prix_final(80)` donne-t-il deux résultats différents ?
3. Réécrivez la fonction pour que la remise soit un paramètre. Écrivez les deux appels.

★★★ **Niveau 3 — Aller plus loin**

Exercice 5.4.7 — La trace des appels ★★★

Lisez ce programme sans l'exécuter.

```python
def double(n):
    r = n * 2
    return r

def calcul(a, b):
    r = double(a) + b
    return r

r = 1
x = calcul(3, r)
y = calcul(x, 4)
print(r, x, y)
```

1. Combien de variables `r` différentes existent dans ce programme ?
2. Faites la trace de l'appel `calcul(3, r)` : que valent `a`, `b`, puis le `r` de `calcul` ?
3. Que valent `x` et `y` ?
4. Qu'affiche le programme ?

Exercice 5.4.8 — Ramener les notes entre 0 et 20 ★★★ Sur machine

Un tableur a exporté des notes avec des erreurs : `[12, -3, 25, 18]`. On veut la fonction `bornees(T)`.

Elle renvoie un **nouveau** tableau où chaque note négative devient 0 et chaque note au-dessus de 20 devient 20. Le tableau de départ ne doit pas changer.

1. Le paramètre `T` doit-il être en entrée seule ou en entrée et sortie ?
2. Écrivez la fonction en pseudo-code, puis en Python.
3. Affichez `notes` après l'appel. Comment voit-on que le tableau de départ n'a pas changé ?

```python
def bornees(T):
    # à compléter
    return T

notes = [12, -3, 25, 18]
propres = bornees(notes)
print(notes, propres)
```

Exercice 5.4.9 — Découper un programme en fonctions ★★★

Une petite école veut un programme pour ses notes. Il doit :

- calculer la moyenne d'un tableau de notes ;
- compter les notes au moins égales à 10 ;
- afficher un bilan : la moyenne et le nombre de notes suffisantes.

1. Proposez trois sous-programmes. Pour chacun, donnez son nom, s'il s'agit d'une fonction ou d'une procédure, ses paramètres et ce qu'il renvoie.
2. Pour chaque paramètre, dites s'il est en entrée seule ou en entrée et sortie.
3. Écrivez les trois en Python, puis appelez le bilan pour `[12, 8, 15, 9]`.

Entraînement à l'épreuve

Vous passez l'épreuve E3 sous sa forme écrite : **2 heures**, deux ou trois exercices, sur papier, sans ordinateur. La calculatrice est autorisée. L'un des exercices évalue la capacité « programmer ».

Cet exercice vaut souvent **5 points** : comptez 25 à 35 minutes. Vous écrivez vos algorithmes en pseudo-code ou dans un langage, au choix. Python vous sert à vous entraîner, puis à vérifier vos réponses après coup.

Les deux sujets ci-dessous ont la forme des annales. Faites-les sur une feuille, chronomètre en main, avant de lire le corrigé.

Sujet type — Les codes de réduction Type examen

Exercice : 5 points

Une boutique en ligne distribue des codes de réduction. Un code est une chaîne de 8 caractères : 3 lettres majuscules, puis 5 chiffres. Par exemple : `"SIO25401"`.

Un code est **valide** s'il a exactement 8 caractères et si la somme de ses 5 chiffres est un multiple de 7. Un nombre est un multiple de 7 quand le reste de sa division par 7 vaut 0.

On dispose de la fonction `chiffre(c)`, déjà écrite :

- elle reçoit un caractère `c`, qui est un chiffre ;
- elle renvoie l'entier correspondant. Par exemple, `chiffre("7")` renvoie 7.

Les caractères d'un code sont repérés par leur indice, à partir de 0 : dans `"SIO25401"`, le caractère d'indice 3 est `"2"`.

1. On considère la fonction suivante.

```
PSEUDO-CODE
1   Fonction somme_chiffres(code)
2       s ← 0
3       Pour i allant de 3 à 7 Faire
4           s ← s + chiffre(code[i])
5       Fin de Pour
6       Renvoyer s
7   Fin de Fonction
```

1. Que renvoie `somme_chiffres("SIO25401")` ? Justifier. *(0,75 point)*

2. Expliquer en une phrase le rôle de cette fonction. *(0,25 point)*

2. Recopier et compléter les lignes 2, 5 et 6 de la fonction `est_valide`, qui renvoie Vrai si le code est valide et Faux sinon. *(1,5 point)*

```
PSEUDO-CODE
1   Fonction est_valide(code)
2       Si longueur(code) ≠ . . . Alors
3           Renvoyer Faux
4       Fin de Si
5       Si le reste de la division de . . . par 7 est 0 Alors
6           Renvoyer . . .
7       Sinon
8           Renvoyer Faux
9       Fin de Si
10  Fin de Fonction
```

3. Écrire une fonction `nb_valides(T)` qui reçoit un tableau `T` de codes et renvoie le nombre de codes valides. On pourra utiliser la fonction `est_valide`. *(1,5 point)*

4. a. Que renvoie `est_valide("SIO00000")` ? Justifier. *(0,5 point)*

1. La boutique refuse les codes dont la somme des chiffres vaut 0. Réécrire la ligne 5 de `est_valide` en conséquence *(0,5 point)*.

Sujet type — Le serveur qui se remplit Type examen

Exercice court : 3 points

Le disque d'un serveur a déjà 100 Go occupés. Chaque jour, les sauvegardes ajoutent 50 Go. On veut savoir combien de jours il faut pour que l'espace occupé atteigne ou dépasse la capacité du disque.

PSEUDO-CODE

```
1   Fonction jours(capacite)
2       occupe ← 100
3       j ← 0
4       Tant que occupe . . . capacite Faire
5           occupe ← occupe + 50
6           j ← . . .
7       Fin de Tant que
8       Renvoyer j
9   Fin de Fonction
```

1. Recopier et compléter les lignes 4 et 6. *(1 point)*

2. Que renvoie `jours(300)` ? Justifier. *(1,5 point)*

3. Que renvoie `jours(80)` ? Expliquer pourquoi. *(0,5 point)*

ANNALES

Les vrais sujets du BTS

Ces questions viennent des vrais sujets de l'épreuve écrite de 2 heures du BTS SIO. Dans chaque sujet, un exercice évalue la capacité « programmer ».

Les énoncés sont recopiés tels quels, du plus récent au plus ancien. Leur corrigé est dans l'onglet Corrigés.

BTS SIO 2026 — Le nombre de clés de codage Type examen

Source : BTS SIO, session 2026, E3 – Mathématiques pour l'informatique, exercice 2, question 2.

Annexe A4 Le sujet officiel, page 3 du PDF, à la fin de ce document.

Un système de codage affine est un système de chiffrement basé sur la correspondance entre les lettres de l'alphabet et les entiers compris entre 0 et 25.

La table de correspondance est donnée ci-dessous.

A	B	C	D	E	F	G	H	I	J	K	L	M
0	1	2	3	4	5	6	7	8	9	10	11	12

N	O	P	Q	R	S	T	U	V	W	X	Y	Z
13	14	15	16	17	18	19	20	21	22	23	24	25

Le codage consiste tout d'abord à choisir une clé qui est un couple d'entiers $(a ; b)$ avec $1 \leqslant a \leqslant 25$ et $0 \leqslant b \leqslant 25$.

Pour coder une lettre :

- on détermine l'entier x qui lui correspond dans la table de correspondance ;
- on calcule $ax + b$ puis on détermine l'entier y compris entre 0 et 25 tel que $ax + b \equiv y \,[26]$;
- on détermine la lettre associée à y dans la table de correspondance.

Exemple : Codons la lettre H avec la clé $(9 ; 5)$.

- H correspond à $x = 7$;
- $9 \times 7 + 5 = 68$ et $68 \equiv 16 \,[26]$ donc $y = 16$;

- $y = 16$ correspond à la lettre Q donc H est codée en Q.

1. Toutes les clés de codage $(a ; b)$ ne permettent pas de crypter un message convenablement. On admet qu'une clé de codage est acceptable si et seulement si l'entier a est premier avec 26. Il n'y a pas de condition sur b (donc 26 choix possibles pour b).

   Recopier et compléter les lignes numérotées **3**, **4** et **5** de l'algorithme définissant la fonction nbCles() qui renvoie le nombre de clés $(a ; b)$ acceptables et dans laquelle la fonction $\text{pgcd}(u, v)$ renvoie le plus grand commun diviseur des entiers u et v.

PSEUDO-CODE

```
1  Fonction nbCles()
2      nb ← 0
3      Pour a allant de 1 à . . . . Faire
4          Si pgcd(a, 26) = . . . . Alors
5              nb ← . . . .
6          Fin de Si
7      Fin de Pour
8      Renvoyer nb × 26
```

BTS SIO 2026 — La clé de décodage Type examen

Source : BTS SIO, session 2026, E3 – Mathématiques pour l'informatique, exercice 2, question 3.

Annexe A5 Le sujet officiel, pages 3 à 4 du PDF, à la fin de ce document.

Le contexte est celui de la question précédente : le codage affine, avec la clé $(a ; b)$.

Pour déterminer la clé de décodage associée à la clé de codage $(a ; b)$, il est nécessaire de déterminer un entier c compris entre 0 et 25 tel que $ac \equiv 1 \ [26]$.

La fonction *inverse* de paramètre a donnée ci-dessous renvoie le plus petit entier c compris entre 0 et 25 tel que $ac \equiv 1 \ [26]$.

La fonction « reste(u, n) » qui est utilisée ci-dessous renvoie le reste de la division euclidienne d'un entier naturel u par un entier naturel non nul n.

PSEUDO-CODE

```
1  Fonction inverse(a)
2      c ← 1
3      Tant que reste(a × c, 26) . . . . Faire
4          c ← c + 1
5      Renvoyer . . . .
```

1. 1. Recopier et compléter les lignes numérotées **3** et **5** de l'algorithme définissant la fonction *inverse* de paramètre a.

2. Déterminer l'entier renvoyé par inverse(9).

BTS SIO 2025 — Le reste par soustractions Type examen

Source : BTS SIO, Métropole, mai 2025, épreuve obligatoire, exercice 2, question 2.

Annexe A6 Le sujet officiel, page 2 du PDF, à la fin de ce document.

Un codage affine est une méthode de chiffrement utilisée en cryptographie, qui consiste à remplacer chaque lettre du message en clair par une autre lettre de l'alphabet en utilisant une fonction affine. On associe à chaque lettre de l'alphabet une valeur numérique.

1. Pour calculer le reste de la division euclidienne d'un entier N par 26, il suffit de soustraire 26 à N autant de fois que possible, c'est-à-dire tant que la différence demeure supérieure ou égale à 26.

   Exemple : Le reste de la division euclidienne de 80 par 26 s'obtient ainsi :

   - 80 – 26 = 54. Comme 54 ⩾ 26, on soustrait alors 26 à 54.
   - 54 – 26 = 28. Comme 28 ⩾ 26, on soustrait alors 26 à 28.
   - 28 – 26 = 2. Comme 2 < 26, on arrête.
   - Le reste de la division euclidienne de 80 par 26 est 2.

   La fonction « reste_division_par_26 » de paramètre N, ci-après, est écrite en langage naturel et emploie cette méthode pour renvoyer le reste de la division euclidienne de N par 26.

   Exemple : reste_division_par_26(80) renvoie la valeur entière 2.

```
PSEUDO-CODE
Fonction reste_division_par_26(N)
    Tant que N . . . Faire
        N ← . . .
    Fin de Tant que
    Renvoyer (N)
Fin de la fonction
```

Recopier et compléter les deux lignes contenant des pointillés de cette fonction.

BTS SIO 2025 — L'indice d'une lettre Type examen

Source : BTS SIO, Métropole, mai 2025, épreuve obligatoire, exercice 2, question 3.

Annexe A7 Le sujet officiel, pages 2 à 3 du PDF, à la fin de ce document.

1. La fonction indice (lettre, chaine) ci-dessous, renvoie le plus petit indice du caractère « lettre » dans la chaîne de caractères « chaine », lorsqu'il est présent dans « chaine », et elle renvoie le nombre de caractères de « chaine » sinon.

   La documentation de cette fonction précise que :

   - k est un entier.
   - lettre est une chaîne de caractères constituée d'un seul caractère.
   - chaine est une chaîne de caractères.
   - longueur(chaine) renvoie le nombre de caractères de chaine.

   Exemple : indice("S","BTSSIO") renvoie la valeur 2, le plus petit indice de "S" dans "BTSSIO".

```
PSEUDO-CODE
Fonction indice(lettre, chaine)
    k ← 0
    Tant que . . . longueur(chaine) et lettre . . . Faire
        k ← k + 1
    Fin de Tant que
    Renvoyer (k)
Fin de Fonction
```

1. Quelle est la valeur renvoyée par indice("A","BTSSIO") ?

2. Recopier et compléter la ligne contenant les pointillés de la fonction indice.

BTS SIO 2024 — Le bit de parité Type examen

Source : BTS SIO, Métropole, mai 2024, épreuve obligatoire, exercice 2, question 3.

Annexe A8 Le sujet officiel, pages 2 à 3 du PDF, à la fin de ce document.

La norme de codage ASCII (American Standard Code for Information Interchange), défini aux États-Unis en 1963, associe aux caractères les plus utilisés dans les documents en langue anglaise un entier représentable en binaire sur 7 bits.

1. Lors de la transmission des données, pour éviter les erreurs, une méthode consiste à ajouter pour chaque caractère, un bit de parité à la fin du codage en binaire.

   Pour cela, on compte le nombre de 1 apparaissant dans le codage sur 7 bits d'un caractère :

   - si le nombre obtenu est pair, on rajoute `0` à la fin du codage ;
   - sinon on rajoute 1.

Chaque caractère est alors codé par un groupe de 8 bits appelé octet.

Par exemple :

- la lettre "A" est codée par 1000001 en binaire. Ce codage contient un nombre pair de 1, donc on ajoute 0 à la fin, et la lettre "A" sera codée sur 8 bits par 10000010 ;
- la lettre "C" est codée par 1000011 en binaire. Ce codage contient un nombre impair de 1, donc on ajoute 1 à la fin, et la lettre "C" sera codée sur 8 bits par 10000111.

On considère l'algorithme ci-dessous écrit en langage naturel où ajoute_bit_parite est la fonction prenant en paramètre une chaine de caractères code de longueur 7 représentant un codage binaire et qui renvoie le codage obtenu en lui ajoutant le bit de parité à la fin.

Ainsi ajoute_bit_parite("1101100") renvoie "11011000" et ajoute_bit_parite("1100001") renvoie "11000011".

```
PSEUDO-CODE
Fonction ajoute_bit_parite(code)
    compt ← 0
    Pour i allant de 0 à 6 Faire
        Si code[i] est égal à "1"
        Alors

        Fin de Si
    Fin de Pour
    Si le reste de la division de . . . par 2 est 0
    Alors
        code ← code + "0"
    Sinon
        . . . . . .
    Fin de Si
    Renvoyer code
```

Recopier et compléter cet algorithme pour qu'il renvoie le code binaire sous forme de chaine de caractères, complété par le bit de parité.

BTS SIO 2023 — Le degré sortant d'un sommet Type examen

Source : BTS SIO, Métropole, mai 2023, épreuve obligatoire, exercice 2, partie B, question 2.

Annexe A9 Le sujet officiel, pages 2 à 3 du PDF, à la fin de ce document.

Partie A. On considère le graphe orienté G comportant 3 sommets notés A, B et C dont la matrice d'adjacence est P, où

$$P = \begin{pmatrix} 1 & 0 & 1 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{pmatrix}$$

Partie B. Dans un graphe orienté, on définit :

- le degré entrant d'un sommet comme étant le nombre d'arcs menant à ce sommet.
- le degré sortant d'un sommet comme étant le nombre d'arcs issus de ce sommet.

1. On étudie dans cette question les graphes orientés à trois sommets numérotés de 1 à 3.

   On considère l'algorithme ci-dessous écrit en langage naturel où Degré_sortant désigne une fonction de paramètres M et s, M étant une matrice à 3 lignes et 3 colonnes et s un entier compris entre 1 et 3.

   Le coefficient de la matrice M situé ligne i colonne j est noté m_{ij}.

```
PSEUDO-CODE
Fonction Degré_sortant(M, s)
    deg ← 0
    Pour j allant de 1 à 3 Faire
        Si m_sj . . . . . Faire
            . . . . . .
        Fin de Si
    Fin de Pour
    Renvoyer deg
```

Compléter cet algorithme pour que la fonction renvoie le degré sortant du sommet numéroté s dans un graphe dont la matrice d'adjacence est M.

BTS SIO 2022 — Les diviseurs et les nombres premiers Type examen

Source : BTS SIO, Métropole, mai 2022, épreuve obligatoire, exercice 2, question 4.

Annexe A10 Le sujet officiel, page 2 du PDF, à la fin de ce document.

1. On considère l'algorithme ci-dessous écrit en langage naturel où Div désigne une fonction de paramètre Nbre, Nbre étant un nombre entier supérieur ou égal à 2.

   Algorithme en langage naturel :

PSEUDO-CODE

```
Fonction Div(Nbre)
    Test ← 0
    Pour i allant de 1 à Nbre Faire
        Si le reste de la division de Nbre par i est égal à 0 Faire
            Test ← Test + 1
        Fin de Si
    Fin de Pour
    Retourner Test
```

1. Que renvoie Div(6) ? Justifier en expliquant le rôle de cette fonction.

2. Écrire une fonction Prem de paramètre Nbre, où Nbre est un nombre entier supérieur ou égal à 2, qui renvoie Vrai (ou True) si Nbre est premier, Faux (ou False) si Nbre n'est pas premier. On pourra utiliser la fonction Div.

CORRIGÉS

Exercices et annales corrigés

Exercices

1.1 — Du problème au programme

‹ Revoir le cours 1.1

★☆☆ **Niveau 1 — Appliquer**

Exercice 1.1.1 — Remettre dans l'ordre ★☆☆

Un magasin vend un casque 50 € hors taxes. On veut afficher son prix toutes taxes comprises. La TVA est de 20 % du prix hors taxes.

Les instructions ont été mélangées :

PSEUDO-CODE

```
TTC ← HT + TVA
Écrire "Prix TTC :", TTC
HT ← 50
TVA ← HT * 0.2
```

1. Remettez les quatre instructions dans le bon ordre.
2. Qu'affiche l'algorithme ?

CORRIGÉ

La méthode. Une variable doit recevoir une valeur avant d'être utilisée. On part de ce qu'on connaît : le prix HT.

1. Le bon ordre :

PSEUDO-CODE

```
HT ← 50
TVA ← HT * 0.2
TTC ← HT + TVA
Écrire "Prix TTC :", TTC
```

1. L'algorithme affiche le prix TTC : 60.

PYTHON

```
HT = 50
TVA = HT * 0.2
TTC = HT + TVA
print("Prix TTC :", TTC)
```

CE QUI S'AFFICHE

```
Prix TTC : 60.0
```

Python écrit 60.0 et non 60 : `HT * 0.2` est un nombre à virgule, donc la somme aussi.

Exercice 1.1.2 — Pseudo-code ou Python ? ★☆☆

Voici six lignes.

```
a. x ← 3
b. print(x)
c. Écrire x
d. x = 3
e. total = prix * qte
f. total ← prix * qte
```

1. Pour chaque ligne, dites si elle est écrite en pseudo-code ou en Python.
2. Associez chaque ligne de pseudo-code à la ligne Python qui dit la même chose.

CORRIGÉ

La méthode. En pseudo-code, on range avec la flèche `←` et on affiche avec `Écrire`. En Python, on range avec `=` et on affiche avec `print`.

1. Pseudo-code : a, c, f. Python : b, d, e.
2. Les paires : a et d, c et b, f et e.

Exercice 1.1.3 — Une vraie instruction ? ★☆☆

Un élève a écrit ces lignes pour calculer la surface d'une salle de 8 m sur 6 m.

PSEUDO-CODE

```
longueur ← 8
largeur ← 6
calculer la surface
Écrire surface
```

1. Quelle ligne n'est pas une instruction que l'ordinateur peut exécuter ? Pourquoi ?
2. Remplacez-la par une vraie instruction.

CORRIGÉ

La méthode. Une instruction dit exactement **quoi** calculer et **où** ranger le résultat.

1. L'instruction « calculer la surface » ne dit pas comment calculer. L'ordinateur ne devine pas la formule.
2. On écrit le calcul : `surface ← longueur * largeur`. L'algorithme affiche alors 48.

★★☆ Niveau 2 — S'entraîner

Exercice 1.1.4 — Traduire en Python ★★☆ Sur machine

Voici un algorithme en pseudo-code.

PSEUDO-CODE

```
longueur ← 8
largeur ← 6
surface ← longueur * largeur
Écrire "Surface :", surface, "m²"
```

1. Écrivez le même algorithme en Python, ligne pour ligne.
2. Exécutez-le. Qu'affiche-t-il ?
3. Changez la largeur en 7,5. En Python, on écrit `7.5`, avec un point. Que devient l'affichage ?

PYTHON

```
longueur = 8
# à compléter
```

CORRIGÉ

La méthode. On traduit ligne par ligne : la flèche devient `=`, `Écrire` devient `print`. Les noms de variables restent les mêmes.

1. et 2. Le programme et son affichage :

PYTHON

```python
longueur = 8
largeur = 6
surface = longueur * largeur
print("Surface :", surface, "m²")
```

CE QUI S'AFFICHE

```
Surface : 48 m²
```

1. En Python, 7,5 s'écrit `7.5`, avec un point :

PYTHON

```python
longueur = 8
largeur = 7.5
surface = longueur * largeur
print("Surface :", surface, "m²")
```

CE QUI S'AFFICHE

```
Surface : 60.0 m²
```

Exercice 1.1.5 — Écrire l'en-tête ★★☆

Voici un algorithme sans en-tête.

PSEUDO-CODE

```
HT ← 50
TVA ← HT * 0.2
TTC ← HT + TVA
Écrire "Prix TTC :", TTC
```

Écrivez son en-tête en commentaires : le nom, le rôle, les entrées et la sortie.

CORRIGÉ

La méthode. L'en-tête répond à quatre questions : comment s'appelle l'algorithme, à quoi il sert, ce qu'il reçoit, ce qu'il produit.

PSEUDO-CODE

```
// Nom : prix_ttc
// Rôle : calculer le prix toutes taxes comprises d'un article
// Entrée : HT, le prix hors taxes, un nombre
// Sortie : le prix TTC affiché à l'écran
```

D'autres noms sont justes, s'ils disent le rôle : `calcul_ttc`, `prix_avec_tva`.

Exercice 1.1.6 — Le taux de pannes ★★☆ Sur machine

Inès compte 80 postes dans le lycée. Ce matin, 76 marchent. Elle veut connaître le **pourcentage de postes en panne**.

1. Écrivez l'algorithme en pseudo-code. Utilisez les variables `total`, `marche`, `pannes` et `pourcentage`.
2. Traduisez-le en Python et exécutez-le.

Pour le pourcentage, multipliez d'abord par 100, puis divisez par le total.

CORRIGÉ

La méthode. On calcule d'abord le nombre de pannes, puis le pourcentage. Chaque ligne utilise ce qui est au-dessus.

PSEUDO-CODE

```
total ← 80
marche ← 76
pannes ← total - marche
pourcentage ← pannes * 100 / total
Écrire "Postes en panne :", pourcentage, "%"
```

PYTHON

```
total = 80
marche = 76
pannes = total - marche
pourcentage = pannes * 100 / total
print("Postes en panne :", pourcentage, "%")
```

CE QUI S'AFFICHE

```
Postes en panne : 5.0 %
```

Vérification à la main : 80 – 76 = 4 pannes. 4 × 100 / 80 = 5.

★★★ Niveau 3 — Aller plus loin

Exercice 1.1.7 — Le programme qui plante ★★★

Ce programme doit afficher le prix d'un abonnement de 3 mois à 15 € par mois. Il s'arrête sur une erreur.

PYTHON

```python
print("Total :", total)
mois = 3
prix = 15
total = mois * prix
```

1. Quelle ligne provoque l'erreur ? Pourquoi ?
2. Corrigez le programme.
3. Ajoutez deux lignes : l'une calcule le prix pour 12 mois, avec une remise de 20 € sur le total ; l'autre l'affiche.
4. Le prix passe à 18 € par mois. Quelle seule ligne changez-vous ? Qu'affiche alors le programme ? Pourquoi les deux totaux suivent-ils seuls ?

CORRIGÉ

La méthode. On lit de haut en bas, comme l'ordinateur. On cherche la première ligne qui utilise une variable qui n'existe pas encore.

1. L'instruction `print("Total :", total)` affiche `total`, mais `total` n'est calculé que trois lignes plus bas. Python s'arrête et affiche `NameError` : c'est son message pour « variable inconnue ».
2. et 3. On met l'affichage à la fin, puis on ajoute le calcul pour 12 mois :

PYTHON

```python
mois = 3
prix = 15
total = mois * prix
print("Total :", total)
total_an = 12 * prix - 20
print("Total sur 12 mois :", total_an)
```

CE QUI S'AFFICHE

```
Total : 45
Total sur 12 mois : 160
```

1. On change seulement `prix = 15` en `prix = 18`. Les deux totaux sont calculés à partir de `prix` : ils changent seuls.

PYTHON

```python
mois = 3
prix = 18
total = mois * prix
print("Total :", total)
total_an = 12 * prix - 20
print("Total sur 12 mois :", total_an)
```

CE QUI S'AFFICHE

```
Total : 54
Total sur 12 mois : 196
```

C'est l'intérêt des variables : une valeur écrite une seule fois se change à un seul endroit.

Exercice 1.1.8 — L'espace libre du disque ★★★ Sur machine

Un serveur a un disque de 500 Go. 365 Go sont occupés.

Écrivez un algorithme, en pseudo-code puis en Python, qui affiche :

- l'espace libre, en Go ;
- le pourcentage d'espace libre.

Ajoutez son en-tête en commentaires.

CORRIGÉ

La méthode. Trois calculs, dans l'ordre : l'espace libre, puis le pourcentage, puis les affichages.

PSEUDO-CODE

```
// Nom : espace_libre
// Rôle : calculer l'espace libre d'un disque et son pourcentage
// Entrées : taille et occupe, en Go
// Sortie : l'espace libre et le pourcentage, affichés
taille ← 500
occupe ← 365
libre ← taille - occupe
pourcentage ← libre * 100 / taille
Écrire "Libre :", libre, "Go"
Écrire "Soit", pourcentage, "%"
```

PYTHON

```
# Nom : espace_libre
# Rôle : calculer l'espace libre d'un disque et son pourcentage
# Entrées : taille et occupe, en Go
# Sortie : l'espace libre et le pourcentage, affichés
taille = 500
occupe = 365
libre = taille - occupe
pourcentage = libre * 100 / taille
print("Libre :", libre, "Go")
print("Soit", pourcentage, "%")
```

CE QUI S'AFFICHE

```
Libre : 135 Go
Soit 27.0 %
```

Exercice 1.1.9 — Quelles lignes peut-on échanger ? ★★★

PSEUDO-CODE

```
a ← 4
b ← 5
c ← a * b
d ← a + b
Écrire c, d
```

1. Qu'affiche l'algorithme ?
2. Peut-on échanger `a ← 4` et `b ← 5` sans changer le résultat ? Et `c ← a * b` et `d ← a + b` ?

3. Peut-on échanger `b ← 5` et `c ← a * b` ? Pourquoi ?
4. Donnez une règle : quand deux lignes peuvent-elles changer de place ?

CORRIGÉ

La méthode. Pour chaque échange, on regarde si une ligne utilise une variable que l'autre crée.

1. L'algorithme affiche 20 et 9.
2. Oui pour les deux. `a ← 4` et `b ← 5` ne dépendent pas l'une de l'autre. `c ← a * b` et `d ← a + b` non plus : elles utilisent `a` et `b`, déjà rangés.
3. Non. `c ← a * b` utilise `b`, qui serait créé après elle : erreur.
4. Deux lignes peuvent changer de place si aucune n'utilise ce que l'autre range.

1.2 — Les variables et l'affectation

‹ Revoir le cours 1.2

★☆☆ **Niveau 1 — Appliquer**

Exercice 1.2.1 — La valeur finale ★☆☆

PSEUDO-CODE

```
p ← 4
p ← p + 6
p ← p * 3
```

Que vaut `p` après chaque ligne ?

CORRIGÉ

La méthode. On calcule la droite avec la valeur actuelle de `p`, puis on range le résultat dans `p`.

- Après `p ← 4` : `p` vaut 4.
- Après `p ← p + 6` : 4 + 6 = 10, `p` vaut 10.
- Après `p ← p * 3` : 10 × 3 = 30, `p` vaut 30.

Exercice 1.2.2 — Affectation juste ou fausse ? ★☆☆

Pour chaque ligne, dites si l'affectation est correcte. Si elle est fausse, dites pourquoi.

PSEUDO-CODE

```
x ← 5
5 ← x
x ← x + 1
x + 1 ← x
total ← prix * qte
```

CORRIGÉ

La méthode. À gauche de la flèche, il faut **une seule variable**. À droite, une valeur ou un calcul.

- `x ← 5` : juste.
- `5 ← x` : faux. On ne range rien dans le nombre 5.
- `x ← x + 1` : juste. C'est une mise à jour.
- `x + 1 ← x` : faux. `x + 1` est un calcul, pas une variable.
- `total ← prix * qte` : juste.

Exercice 1.2.3 — Le stock de câbles ★☆☆ Sur machine

Un magasin a 40 câbles. Il en vend 15, puis il en reçoit 30.

PSEUDO-CODE

```
stock ← 40
stock ← stock - 15
stock ← stock + 30
Écrire "Stock :", stock
```

1. Que vaut `stock` à la fin ?
2. Écrivez le programme en Python et vérifiez.

CORRIGÉ

La méthode. Deux mises à jour : on part de l'ancienne valeur à chaque fois.

1. 40 − 15 = 25, puis 25 + 30 = 55.
2. En Python :

PYTHON

```python
stock = 40
stock = stock - 15
stock = stock + 30
print("Stock :", stock)
```

CE QUI S'AFFICHE

```
Stock : 55
```

On peut aussi écrire `stock -= 15` et `stock += 30`.

★★☆ **Niveau 2 — S'entraîner**

Exercice 1.2.4 — Échanger deux valeurs ★★☆ Sur machine

On a `a ← 3` et `b ← 8`. On veut que `a` vaille 8 et `b` vaille 3.

PSEUDO-CODE

```
a ← 3
b ← 8
temp ← . . .
a ← . . .
b ← . . .
```

1. Complétez les trois lignes.
2. Vérifiez en Python.

CORRIGÉ

La méthode. On met de côté la valeur de `a` avant de l'écraser.

PSEUDO-CODE

```
a ← 3
b ← 8
temp ← a
a ← b
b ← temp
```

PYTHON

```python
a = 3
b = 8
temp = a
a = b
b = temp
print(a, b)
```

CE QUI S'AFFICHE

```
8 3
```

> **Attention**
>
> Sans `temp`, `a ← b` efface le 3. Les deux variables finissent à 8.

Exercice 1.2.5 — Une copie, pas un lien ★★☆

PSEUDO-CODE

```
x ← 7
y ← x
x ← x + 5
```

1. Que vaut `x` à la fin ?
2. Que vaut `y` à la fin ?
3. Un camarade dit : « `y` vaut 12, car `y` est égal à `x` ». A-t-il raison ?

CORRIGÉ

La méthode. `y ← x` copie la valeur de `x` **à ce moment-là**. Ensuite, les deux variables vivent séparément.

1. `x` vaut 12.
2. `y` vaut 7.
3. Non. `y` a reçu une copie de la valeur 7. Quand `x` change ensuite, `y` ne bouge pas.

Exercice 1.2.6 — Le compteur de visiteurs ★★☆ Sur machine

Une médiathèque compte ses visiteurs. Le compteur part de `0`. Il entre 12 personnes, puis 7, puis 3 personnes sortent.

1. Écrivez l'algorithme avec une seule variable `visiteurs` et des mises à jour.
2. Traduisez-le en Python en utilisant `+=` et `-=`.

CORRIGÉ

La méthode. Une seule variable, mise à jour à chaque événement.

PSEUDO-CODE

```
visiteurs ← 0
visiteurs ← visiteurs + 12
visiteurs ← visiteurs + 7
visiteurs ← visiteurs - 3
Écrire "Présents :", visiteurs
```

PYTHON

```python
visiteurs = 0
visiteurs += 12
visiteurs += 7
visiteurs -= 3
print("Présents :", visiteurs)
```

CE QUI S'AFFICHE

```
Présents : 16
```

★★★ Niveau 3 — Aller plus loin

Exercice 1.2.7 — Faire tourner trois valeurs ★★★ Sur machine

On a `a ← 1`, `b ← 2` et `c ← 3`. On veut obtenir `a` = 2, `b` = 3 et `c` = 1 : chaque valeur avance d'une place.

1. Écrivez les affectations, avec une variable `temp`.
2. Vérifiez en Python.

CORRIGÉ

La méthode. C'est un échange à trois. On met de côté la première valeur, on décale les autres, puis on range la valeur mise de côté à la fin.

PSEUDO-CODE

```
temp ← a
a ← b
b ← c
c ← temp
```

PYTHON

```python
a = 1
b = 2
c = 3
temp = a
a = b
b = c
c = temp
print(a, b, c)
```

CE QUI S'AFFICHE

```
2 3 1
```

Exercice 1.2.8 — Le solde du compte ★★★ Sur machine

Le compte d'une petite association contient 200 €. Elle paie une facture de 45,50 €, puis reçoit un don de 120 €. En Python, 45,50 s'écrit `45.50`, avec un point.

1. Écrivez l'algorithme avec une seule variable `solde`.
2. Traduisez-le en Python avec `-=` et `+=`.
3. Ajoutez une ligne qui retire 15 € de frais. Quel solde s'affiche ?

CORRIGÉ

La méthode. Chaque opération est une mise à jour du même solde.

PSEUDO-CODE

```
solde ← 200
solde ← solde - 45.50
solde ← solde + 120
solde ← solde - 15
Écrire "Solde :", solde
```

PYTHON

```python
solde = 200
solde -= 45.50
solde += 120
solde -= 15
print("Solde :", solde)
```

CE QUI S'AFFICHE

```
Solde : 259.5
```

Vérification à la main : 200 − 45,50 = 154,50. Puis 154,50 + 120 = 274,50. Puis 274,50 − 15 = 259,50.

Exercice 1.2.9 — Doubler encore et encore ★★★

PSEUDO-CODE

```
x ← 1
x ← x * 2
x ← x * 2
x ← x * 2
x ← x * 2
x ← x * 2
```

1. Que vaut `x` à la fin ?
2. Combien de lignes `x ← x * 2` faut-il, après `x ← 1`, pour que `x` vaille 1 024 ?
3. Quel lien voyez-vous avec les puissances de 2 ?
4. On part de `x ← 2` et on répète `x ← x * x`. Après combien de lignes `x` dépasse-t-il 1 000 ? Combien de lignes `x ← x * 2` faudrait-il, en partant aussi de 2 ? Quelle méthode est la plus rapide ?

CORRIGÉ

La méthode. On suit `x` ligne par ligne : 1, 2, 4, 8, 16, 32.

1. `x` vaut 32.
2. Il faut 10 lignes, car $2^{10} = 1024$.
3. Après n lignes `x ← x * 2`, `x` vaut 2^n.
4. Avec `x ← x * x` : 2, puis 4, puis 16, puis 256, puis 65 536. Il faut 4 lignes pour dépasser 1 000. Avec `x ← x * 2` en partant de 2, il en faut 9 pour atteindre 1 024. La mise au carré va beaucoup plus vite : chaque ligne double le nombre de chiffres binaires.

PYTHON

```python
x = 1
x = x * 2
x = x * 2
x = x * 2
x = x * 2
x = x * 2
print(x)
y = 2
y = y * y
y = y * y
y = y * y
y = y * y
print(y)
```

CE QUI S'AFFICHE

```
32
65536
```

1.3 — Lire et écrire

‹ Revoir le cours 1.3

★★★ **Niveau 1 — Appliquer**

Exercice 1.3.1 — Le bon type à la lecture ★★★

On lit au clavier les valeurs suivantes. Pour chacune, dites s'il faut écrire `int(input(...))`, `float(input(...))` ou seulement `input(...)`.

1. L'âge d'un client.
2. Le prix d'un article, par exemple 12,99 €.
3. Le prénom d'un client.
4. Le nombre d'élèves d'une classe.
5. La température d'une salle, par exemple 21,5 °C.

CORRIGÉ

La méthode. `input` donne du texte. On le change en entier pour un nombre sans virgule, en réel pour un nombre à virgule. Un texte reste un texte.

1. `int(input(...))`

2. `float(input(...))`
3. `input(...)`
4. `int(input(...))`
5. `float(input(...))`

Exercice 1.3.2 — Dire bonjour ★☆☆ Sur machine

Écrivez un programme qui demande le prénom de l'utilisateur, puis affiche « Bonjour » suivi du prénom.

PYTHON

```python
prenom = input("Prénom ? ")
# à compléter
```

CORRIGÉ

La méthode. On lit le prénom, puis on l'affiche. Un prénom reste du texte : pas de `int`.

PSEUDO-CODE

```
Écrire "Prénom ?"
Lire prenom
Écrire "Bonjour", prenom
```

PYTHON

```python
prenom = input("Prénom ? ")
print("Bonjour", prenom)
```

CE QUI S'AFFICHE

```
Prénom ? Inès
Bonjour Inès
```

Exercice 1.3.3 — L'âge de l'an prochain ★☆☆

Ce programme doit afficher l'âge de l'utilisateur l'an prochain. Il s'arrête sur une erreur.

PYTHON

```python
age = input("Âge ? ")
print("L'an prochain :", age + 1)
```

1. Pourquoi ce programme plante-t-il ?

2. Corrigez-le.

CORRIGÉ

La méthode. On regarde le type de ce que donne `input`.

1. `input` renvoie du texte. Python refuse d'ajouter le nombre 1 à du texte.
2. On change le texte en entier dès la lecture :

PYTHON

```
age = int(input("Âge ? "))
print("L'an prochain :", age + 1)
```

CE QUI S'AFFICHE

```
Âge ? 17
L'an prochain : 18
```

★★☆ **Niveau 2 — S'entraîner**

Exercice 1.3.4 — Celsius vers Fahrenheit ★★☆ Sur machine

Un technicien lit la température d'une salle serveur en degrés Celsius. Le fabricant de la climatisation donne ses seuils en degrés Fahrenheit. La formule est :

$$F = C \times \frac{9}{5} + 32$$

On lit : F égale C fois 9 cinquièmes, plus 32.

1. Écrivez la formule sur une ligne, avec les variables `c` et `f`. Une fraction devient une division : $\frac{9}{5}$ s'écrit `9 / 5`.
2. Complétez le programme : il lit une température en Celsius et affiche la température en Fahrenheit.
3. Testez-le avec 25 °C. Vous devez obtenir 77.0 °F : Python écrit `77.0`, car `c` est un réel.

PYTHON

```
c = float(input("Température en °C ? "))
f = 0      # à compléter
print(c, "°C =", f, "°F")
```

CORRIGÉ

La méthode. On traduit la fraction en division, et on garde l'ordre des calculs de la formule.

1. Sur une ligne : `f ← c * 9 / 5 + 32`. La multiplication et la division passent avant l'addition : pas besoin de parenthèses.
2. Le programme complet :

PYTHON

```python
c = float(input("Température en °C ? "))
f = c * 9 / 5 + 32
print(c, "°C =", f, "°F")
```

CE QUI S'AFFICHE

```
Température en °C ? 25
25.0 °C = 77.0 °F
```

1. Le test donne bien 77. Python écrit `77.0`, car `c` est un réel. En pseudo-code :

PSEUDO-CODE

```
Écrire "Température en °C ?"
Lire c
f ← c * 9 / 5 + 32
Écrire c, "°C =", f, "°F"
```

Attention

Sans `float(...)`, `c` serait du texte, et `c * 9` répéterait le texte 9 fois au lieu de calculer.

Exercice 1.3.5 — Le prix avec remise ★★☆ Sur machine

Une boutique en ligne calcule le prix d'une commande. Le client tape le prix d'un article, la quantité et le taux de remise en pourcentage. Le programme doit :

1. calculer le montant brut : prix × quantité ;
2. calculer la remise : montant brut × taux / 100 ;
3. calculer le montant net : montant brut – remise ;
4. afficher le montant net.

Écrivez l'algorithme en pseudo-code, puis en Python. Testez-le avec un prix de 12,5 €, 4 articles et 10 % de remise.

CORRIGÉ

La méthode. On lit les trois valeurs, on suit les quatre étapes dans l'ordre, puis on affiche.

PSEUDO-CODE

```
Écrire "Prix unitaire ?"
Lire prix
Écrire "Quantité ?"
Lire qte
Écrire "Remise en % ?"
Lire taux
brut ← prix * qte
remise ← brut * taux / 100
net ← brut - remise
Écrire "Montant net :", net, "€"
```

PYTHON

```python
prix = float(input("Prix unitaire ? "))
qte = int(input("Quantité ? "))
taux = float(input("Remise en % ? "))
brut = prix * qte
remise = brut * taux / 100
net = brut - remise
print("Montant net :", net, "€")
```

CE QUI S'AFFICHE

```
Prix unitaire ? 12.5
Quantité ? 4
Remise en % ? 10
Montant net : 45.0 €
```

Vérification à la main : 12,5 × 4 = 50. La remise vaut 50 × 10 / 100 = 5. Le net vaut 50 – 5 = 45.

> **Attention**
>
> En Python, on tape `12.5` avec un point. Avec `12,5`, `float` s'arrête sur une erreur.

Exercice 1.3.6 — Le périmètre d'une salle ★★☆

On veut le périmètre d'une salle rectangulaire. L'utilisateur tape la longueur et la largeur, en mètres. Le périmètre vaut 2 × (longueur + largeur).

1. Écrivez l'algorithme en pseudo-code : deux lectures, un calcul, un affichage.
2. Écrivez-le en Python. Testez avec 8 m et 6 m.

CORRIGÉ

La méthode. On lit, on calcule, on affiche. Les parenthèses font l'addition avant la multiplication.

PSEUDO-CODE

```
Écrire "Longueur ?"
Lire longueur
Écrire "Largeur ?"
Lire largeur
perimetre ← 2 * (longueur + largeur)
Écrire "Périmètre :", perimetre
```

PYTHON

```python
longueur = float(input("Longueur ? "))
largeur = float(input("Largeur ? "))
perimetre = 2 * (longueur + largeur)
print("Périmètre :", perimetre)
```

CE QUI S'AFFICHE

```
Longueur ? 8
Largeur ? 6
Périmètre : 28.0
```

> **Attention**
>
> Sans parenthèses, `2 * longueur + largeur` donne 22 : on ne double que la longueur.

★★★ **Niveau 3 — Aller plus loin**

Exercice 1.3.7 — Passer de 3 notes à 4 notes ★★★

Cet algorithme calcule la moyenne de 3 notes.

PSEUDO-CODE

```
Lire n1
Lire n2
Lire n3
somme ← n1 + n2 + n3
moyenne ← somme / 3
Écrire "Moyenne :", moyenne
```

1. Qu'affiche-t-il pour les notes 12, 15 et 9 ?
2. Modifiez-le pour qu'il calcule la moyenne de 4 notes.
3. Qu'affiche votre version pour 12, 15, 9 et 14 ?

CORRIGÉ

La méthode. On repère chaque endroit où le nombre de notes compte : la lecture, la somme, la division.

1. La somme vaut 12 + 15 + 9 = 36. La moyenne vaut 36 / 3 = 12.
2. Trois changements : une lecture de plus, `n4` dans la somme, une division par 4.

PSEUDO-CODE

```
Lire n1
Lire n2
Lire n3
Lire n4
somme ← n1 + n2 + n3 + n4
moyenne ← somme / 4
Écrire "Moyenne :", moyenne
```

1. La vérification en Python :

PYTHON

```python
n1 = float(input("Note 1 ? "))
n2 = float(input("Note 2 ? "))
n3 = float(input("Note 3 ? "))
n4 = float(input("Note 4 ? "))
somme = n1 + n2 + n3 + n4
moyenne = somme / 4
print("Moyenne :", moyenne)
```

CE QUI S'AFFICHE

```
Note 1 ? 12
Note 2 ? 15
Note 3 ? 9
Note 4 ? 14
Moyenne : 12.5
```

Attention

L'erreur fréquente : ajouter `n4` mais garder `/ 3`. Le programme tourne sans erreur, mais la moyenne est fausse.

Exercice 1.3.8 — Heures et minutes en minutes ★★★ Sur machine

Une vidéo dure 2 h 35 min. On veut un programme qui lit les heures et les minutes, puis affiche :

- la durée totale en minutes ;
- la durée totale en secondes.

1. Écrivez l'algorithme en pseudo-code.
2. Écrivez-le en Python et testez avec 2 h 35 min.

CORRIGÉ

La méthode. Une heure fait 60 minutes, une minute fait 60 secondes. On calcule les minutes, puis les secondes à partir des minutes.

PSEUDO-CODE

```
Écrire "Heures ?"
Lire h
Écrire "Minutes ?"
Lire m
total ← h * 60 + m
secondes ← total * 60
Écrire "Total :", total, "minutes"
Écrire "Soit :", secondes, "secondes"
```

PYTHON

```python
h = int(input("Heures ? "))
m = int(input("Minutes ? "))
total = h * 60 + m
secondes = total * 60
print("Total :", total, "minutes")
print("Soit :", secondes, "secondes")
```

CE QUI S'AFFICHE

```
Heures ? 2
Minutes ? 35
Total : 155 minutes
Soit : 9300 secondes
```

Exercice 1.3.9 — La facture du magasin ★★★ Sur machine

Un magasin vend trois articles : une clé USB à 8 €, un câble à 5 €, une souris à 12 €. Le client tape la quantité de chaque article.

Le programme affiche le total hors taxes, la TVA à 20 % et le total TTC. Calculez la TVA avec `HT * 20 / 100`.

1. Écrivez l'algorithme en pseudo-code, **avec son en-tête** : nom, rôle, entrées, sortie. Ajoutez un commentaire `//` pour le rôle de chaque variable calculée.
2. Écrivez-le en Python, avec le même en-tête en commentaires `#`. Testez avec 2 clés, 3 câbles et 1 souris.

CORRIGÉ

La méthode. On lit les trois quantités. Le total HT est la somme des trois prix fois leur quantité. Puis la TVA, puis le TTC.

PSEUDO-CODE

```
// Nom : facture_magasin
// Rôle : calculer le total HT, la TVA et le total TTC d'une commande
// Entrées : les quantités de clés, de câbles et de souris, des entiers
// Sortie : les trois montants, affichés
Écrire "Clés USB ?"
Lire cles
Écrire "Câbles ?"
Lire cables
Écrire "Souris ?"
Lire souris
HT ← cles * 8 + cables * 5 + souris * 12   // total hors taxes, en €
TVA ← HT * 20 / 100                        // taxe à 20 %
TTC ← HT + TVA                             // total à payer
Écrire "Total HT :", HT
Écrire "TVA :", TVA
Écrire "Total TTC :", TTC
```

PYTHON

```
# Nom : facture_magasin
# Rôle : calculer le total HT, la TVA et le total TTC d'une commande
# Entrées : les quantités de clés, de câbles et de souris
# Sortie : les trois montants, affichés
cles = int(input("Clés USB ? "))
cables = int(input("Câbles ? "))
souris = int(input("Souris ? "))
HT = cles * 8 + cables * 5 + souris * 12
TVA = HT * 20 / 100
TTC = HT + TVA
print("Total HT :", HT)
print("TVA :", TVA)
print("Total TTC :", TTC)
```

CE QUI S'AFFICHE

```
Clés USB ? 2
Câbles ? 3
Souris ? 1
Total HT : 43
TVA : 8.6
Total TTC : 51.6
```

Vérification à la main : 2 × 8 + 3 × 5 + 1 × 12 = 16 + 15 + 12 = 43. La TVA vaut 43 × 20 / 100 = 8,6.

1.4 — La trace d'un algorithme

‹ Revoir le cours 1.4

★☆☆ **Niveau 1 — Appliquer**

Exercice 1.4.1 — La trace d'un calcul ★☆☆

PSEUDO-CODE

```
x ← 7
y ← x - 2
x ← x + y
y ← x * 2
```

1. Recopiez le tableau et faites la trace.
2. Donnez les valeurs de `x` et de `y` à la fin.

Étape	Instruction	x	y
1	`x ← 7`		
2	`y ← x - 2`		
3	`x ← x + y`		
4	`y ← x * 2`		

CORRIGÉ

La méthode. On lit une ligne à la fois. À chaque ligne, on prend les valeurs de la ligne juste au-dessus.

Étape	Instruction	x	y
1	`x ← 7`	7	–
2	`y ← x - 2`	7	5
3	`x ← x + y`	12	5
4	`y ← x * 2`	12	24

À la fin, `x` vaut 12 et `y` vaut 24.

PYTHON

```python
x = 7
y = x - 2
x = x + y
y = x * 2
print("x =", x, "y =", y)
```

CE QUI S'AFFICHE

```
x = 12 y = 24
```

Attention

À l'étape 4, on utilise la **nouvelle** valeur de `x`, 12, pas l'ancienne, 7.

Exercice 1.4.2 — Trois variables ★☆☆

PSEUDO-CODE

```
a ← 2
b ← a + 4
c ← a * b
a ← c - b
```

Faites la trace dans ce tableau.

Étape	Instruction	a	b	c
1	`a ← 2`			
2	`b ← a + 4`			
3	`c ← a * b`			
4	`a ← c - b`			

CORRIGÉ

La méthode. Une colonne par variable. On recopie les valeurs qui ne changent pas.

Étape	Instruction	a	b	c
1	`a ← 2`	2	–	–
2	`b ← a + 4`	2	6	–
3	`c ← a * b`	2	6	12
4	`a ← c - b`	6	6	12

À la fin : `a` = 6, `b` = 6, `c` = 12.

Exercice 1.4.3 — Une seule variable ★☆☆

PSEUDO-CODE

```
s ← 0
s ← s + 5
s ← s + 8
s ← s - 3
```

Faites la trace de `s`. Que vaut `s` à la fin ?

Étape	Instruction	s
1	`s ← 0`	
2	`s ← s + 5`	
3	`s ← s + 8`	
4	`s ← s - 3`	

CORRIGÉ

La méthode. À chaque ligne, on part de la valeur de la ligne au-dessus.

Étape	Instruction	s
1	`s ← 0`	0
2	`s ← s + 5`	5
3	`s ← s + 8`	13
4	`s ← s - 3`	10

À la fin, `s` vaut 10.

★★☆ **Niveau 2 — S'entraîner**

Exercice 1.4.4 — Une trace avec une lecture ★★☆

PSEUDO-CODE

```
Lire x
y ← x * x
x ← y - x
Écrire x, y
```

1. L'utilisateur tape 6. Faites la trace.
2. Qu'affiche l'algorithme ?
3. Qu'afficherait-il si l'utilisateur tapait 3 ?

Étape	Instruction	x	y
1	`Lire x`		
2	`y ← x * x`		
3	`x ← y - x`		

CORRIGÉ

La méthode. `Lire x` range la valeur tapée dans `x`. Ensuite, on trace comme d'habitude.

1. La trace avec 6 :

Étape	Instruction	x	y
1	`Lire x`	6	–
2	`y ← x * x`	6	36
3	`x ← y - x`	30	36

1. L'algorithme affiche 30 et 36.
2. Avec 3 : `y` vaut 9, puis `x` vaut 9 – 3 = 6. Il affiche 6 et 9.

PYTHON

```python
x = int(input("x ? "))
y = x * x
x = y - x
print(x, y)
```

CE QUI S'AFFICHE

```
x ? 6
30 36
```

Exercice 1.4.5 — La moyenne fausse ★★☆

Cet algorithme doit calculer la moyenne de deux notes, 12 et 15.

PSEUDO-CODE

```
a ← 12
b ← 15
m ← a + b / 2
Écrire m
```

1. Faites la trace. Que vaut `m` ?
2. La moyenne attendue est 13,5. D'où vient l'erreur ?
3. Corrigez l'instruction qui calcule `m`.

CORRIGÉ

La méthode. La trace montre la valeur réelle de `m`. On la compare à la valeur attendue.

1. La division passe avant l'addition : `b / 2` vaut 7,5, puis 12 + 7,5 = 19,5. `m` vaut 19,5.
2. L'ordinateur suit l'ordre des opérations : il divise seulement `b` par 2.
3. On ajoute des parenthèses : `m ← (a + b) / 2`.

PYTHON

```python
a = 12
b = 15
m = (a + b) / 2
print(m)
```

CE QUI S'AFFICHE

```
13.5
```

Exercice 1.4.6 — Trouver l'erreur dans une trace ★★☆ Sur machine

PSEUDO-CODE

```
p ← 3
q ← p * 4
p ← q - p
q ← q + p
```

Un élève a rempli cette trace :

Étape	p	q
1	3	–
2	3	12
3	9	12
4	9	15

1. À quelle étape l'élève s'est-il trompé ? Quelle erreur a-t-il faite ?
2. Écrivez le programme en Python avec un `print` à la fin pour vérifier.

CORRIGÉ

La méthode. On refait chaque ligne avec les valeurs de la ligne juste au-dessus.

1. L'erreur est à l'étape 4. L'élève a calculé 12 + 3 = 15, avec l'**ancienne** valeur de `p`. Or `p` vaut 9 depuis l'étape 3. Le bon calcul est 12 + 9 = 21.
2. La vérification :

PYTHON

```python
p = 3
q = p * 4
p = q - p
q = q + p
print("p =", p, "q =", q)
```

CE QUI S'AFFICHE

```
p = 9 q = 21
```

★★★ **Niveau 3 — Aller plus loin**

Exercice 1.4.7 — Que fait cet algorithme ? ★★★

PSEUDO-CODE

```
a ← 2
b ← 3
a ← a + b
b ← a - b
a ← a - b
```

1. Faites la trace de l'algorithme.
2. Comparez les valeurs de départ et d'arrivée de `a` et de `b`. Que fait l'algorithme, en une phrase ?
3. Refaites la trace avec `a ← 10` et `b ← 4` pour confirmer.

CORRIGÉ

La méthode. On fait la trace, puis on compare le début et la fin.

1. La trace :

Étape	a	b
`a ← 2`	2	–
`b ← 3`	2	3
`a ← a + b`	5	3
`b ← a - b`	5	2
`a ← a - b`	3	2

1. Au départ, `a` vaut 2 et `b` vaut 3. À la fin, `a` vaut 3 et `b` vaut 2. L'algorithme **échange** les valeurs de `a` et de `b`, sans troisième variable.

2. Avec 10 et 4 : `a` devient 14, puis `b` devient 10, puis `a` devient 4. L'échange marche encore.

PYTHON

```python
a = 10
b = 4
a = a + b
b = a - b
a = a - b
print("a =", a, "b =", b)
```

CE QUI S'AFFICHE

```
a = 4 b = 10
```

Exercice 1.4.8 — Le prix change après coup ★★★

PSEUDO-CODE

```
prix ← 20
qte ← 3
total ← prix * qte
prix ← 25
Écrire total
```

1. Faites la trace.
2. Qu'affiche l'algorithme : 60 ou 75 ? Justifiez avec la trace.
3. On veut afficher le total avec le nouveau prix. Quelle ligne faut-il ajouter, et où ?

CORRIGÉ

La méthode. Une affectation calcule **une fois**, au moment où elle s'exécute. Elle ne se refait pas seule quand une variable change ensuite.

1. La trace :

Étape	prix	qte	total
`prix ← 20`	20	–	–
`qte ← 3`	20	3	–
`total ← prix * qte`	20	3	60
`prix ← 25`	25	3	60

1. Il affiche 60. `total` a été calculé quand `prix` valait 20. Changer `prix` après ne change pas `total`.

2. On refait le calcul après le changement de prix : `total ← prix * qte`, juste avant `Écrire total`. Il affiche alors 75.

PYTHON

```python
prix = 20
qte = 3
total = prix * qte
prix = 25
total = prix * qte
print(total)
```

CE QUI S'AFFICHE

```
75
```

Exercice 1.4.9 — Retrouver la valeur de départ ★★★

PSEUDO-CODE

```
Lire x
x ← x + 4
x ← x * 3
Écrire x
```

1. L'utilisateur a tapé un nombre, et l'algorithme a affiché 27. Quel nombre a-t-il tapé ? Expliquez votre raisonnement. Pour démarrer : juste après `x ← x * 3`, `x` vaut 27. Que valait `x` juste avant cette instruction ?
2. Vérifiez en faisant la trace avec ce nombre.

CORRIGÉ

La méthode. On remonte la trace à l'envers, en faisant l'opération inverse à chaque ligne.

1. À la fin, `x` vaut 27. Avant `x ← x * 3`, `x` valait 27 / 3 = 9. Avant `x ← x + 4`, `x` valait 9 – 4 = 5. L'utilisateur a tapé 5.
2. La trace avec 5 : `x` vaut 5, puis 9, puis 27.

PYTHON

```python
x = int(input("x ? "))
x = x + 4
x = x * 3
print(x)
```

CE QUI S'AFFICHE

```
x ? 5
27
```

2.1 — Les types simples

‹ Revoir le cours 2.1

★☆☆ **Niveau 1 — Appliquer**

Exercice 2.1.1 — Choisir le type ★☆☆

Pour chaque donnée, choisissez le type le plus adapté : entier naturel, entier relatif, réel, booléen ou chaîne de caractères.

1. Le nombre d'utilisateurs connectés à un serveur.
2. La température relevée par un capteur dans une chambre froide, en degrés entiers.
3. L'utilisation du processeur, affichée en pourcentage, par exemple 73,4.
4. « Le serveur web est-il en ligne ? »
5. L'âge d'un client.

CORRIGÉ

La méthode. On se pose trois questions.

- Peut-il être négatif ?
- A-t-il une virgule ?
- Fait-on des calculs dessus ?

1. **Entier naturel** : un nombre de personnes est entier et jamais négatif.
2. **Entier relatif** : une chambre froide descend sous 0 °C.
3. **Réel** : il y a une virgule.
4. **Booléen** : la réponse est oui ou non.
5. **Entier naturel** : un âge est entier et jamais négatif.

Exercice 2.1.2 — Vrai ou faux ? ★☆☆

Dites si chaque phrase est vraie ou fausse. Justifiez en une phrase.

1. Le mot de passe `admin123` peut être rangé dans un entier.
2. En Python, `3 > 5` est un entier.
3. Une adresse IP, l'adresse d'un ordinateur sur un réseau, comme 192.168.0.1, est faite de 4 entiers naturels.
4. Le numéro 443, qui désigne le service web sur un serveur, est un entier naturel.
5. En Python, `type(7.0)` et `type(7)` donnent le même type.

CORRIGÉ

La méthode. On regarde ce que la donnée contient vraiment, et ce qu'on fait avec.

1. **Faux.** `admin123` contient des lettres : c'est une chaîne de caractères.
2. **Faux.** Une comparaison donne un booléen. Ici, `False`.
3. **Vrai.** Chacun des 4 nombres est entier et jamais négatif.
4. **Vrai.** C'est un nombre entier, jamais négatif.
5. **Faux.** `7.0` est un réel, `float`. `7` est un entier, `int`.

PYTHON

```python
print(type(3 > 5), type(7.0), type(7))
```

CE QUI S'AFFICHE

```
<class 'bool'> <class 'float'> <class 'int'>
```

Exercice 2.1.3 — Le type en Python ★☆☆

Donnez le type Python de chaque variable : `int`, `float` ou `bool`.

1. `users = 57`
2. `temp = -7`
3. `cpu = 73.4`
4. `connecte = True`
5. `prix = 12.0`

CORRIGÉ

La méthode. Un nombre sans point est un `int`, avec un point un `float`. `True` et `False` sont des `bool`.

1. `int` : entier naturel.
2. `int` : entier relatif. Python utilise le même type pour les deux.
3. `float` : réel.
4. `bool` : booléen.
5. `float` : il y a un point, même si la partie après vaut `0`.

★★☆ **Niveau 2 — S'entraîner**

Exercice 2.1.4 — La fiche d'un produit ★★☆

Un magasin d'informatique décrit chaque produit avec cinq données :

- son nom ;
- son prix en euros ;
- le nombre en stock ;
- s'il est en promotion ;
- l'écart de stock depuis l'inventaire, qui peut être négatif.

1. Donnez le type de chaque donnée.
2. Écrivez en Python les cinq affectations pour une souris à 19,90 €, 12 en stock, en promotion, avec un écart de -3.

CORRIGÉ

La méthode. On choisit un type par donnée. Puis on écrit la valeur dans la bonne forme : guillemets pour le texte, point pour les réels, `True` ou `False` pour les booléens.

1. Les types :
    - nom : chaîne de caractères ;
    - prix : réel ;
    - stock : entier naturel ;
    - promotion : booléen ;
    - écart : entier relatif.
2. En Python :

PYTHON

```python
nom = "Souris"
prix = 19.90
stock = 12
promo = True
ecart = -3
print(type(nom), type(prix), type(stock), type(promo), type(ecart))
```

CE QUI S'AFFICHE

```
<class 'str'> <class 'float'> <class 'int'> <class 'bool'> <class 'int'>
```

Attention

`prix = 19,90` n'est pas un réel en Python. La virgule fabrique deux valeurs, 19 et 90. Écrivez toujours un point.

Exercice 2.1.5 — Prévoir le type ★★☆ Sur machine

1. Sans ordinateur, écrivez ce qu'affiche chaque ligne.
2. Exécutez le programme pour vérifier.

PYTHON

```python
print(type(7))
print(type("7"))
print(type(7.0))
print(type(7 > 3))
print(type("True"))
```

CORRIGÉ

La méthode. Des guillemets donnent toujours une chaîne, même autour d'un nombre ou du mot `True`.

PYTHON

```python
print(type(7))
print(type("7"))
print(type(7.0))
print(type(7 > 3))
print(type("True"))
```

CE QUI S'AFFICHE

```
<class 'int'>
<class 'str'>
<class 'float'>
<class 'bool'>
<class 'str'>
```

`str` est le type des chaînes de caractères, pour *string* en anglais.

Exercice 2.1.6 — Le mauvais type ★★☆

Expliquez en une ou deux phrases le problème posé par chaque choix.

1. On range une moyenne de 12,67 dans un entier.
2. On range « le client a payé » dans une chaîne, `"oui"` ou `"non"`.
3. On range le code postal `01000` dans un entier.

CORRIGÉ

La méthode. On cherche ce qui se perd ou ce qui peut mal tourner.

1. La partie après la virgule est perdue : la moyenne devient 12.
2. Une chaîne peut valoir `"Oui"`, `"OUI"`, `"o"`. Un test sur `"oui"` échoue alors. Un booléen n'a que deux valeurs, sans faute de frappe.
3. Le `0` du début disparaît : le code devient `1000`.

★★★ Niveau 3 — Aller plus loin

Exercice 2.1.7 — La fiche d'inscription ★★★ Sur machine

Yanis écrit la fiche d'inscription d'une salle de sport. Elle demande le prénom, l'âge et le poids du client, puis les affiche.

1. Donnez le type de chaque donnée.
2. Complétez le programme. Le poids peut avoir une virgule.
3. Testez avec Léa, 19 ans, 58,5 kg. En Python, on tape `58.5`, avec un point.
4. La fiche doit aussi garder « le client a payé ». Ajoutez une variable `paye` qui vaut Vrai, sans la lire au clavier.
5. Ajoutez une ligne qui affiche le type des quatre variables. Comparez avec vos réponses à la question 1.

PYTHON

```python
prenom = input("Prénom ? ")
age = 0      # à compléter
poids = 0    # à compléter
print(prenom, age, poids)
```

CORRIGÉ

La méthode. `input` renvoie du texte. On garde le texte pour le prénom, on le change en entier pour l'âge, en réel pour le poids.

1. Prénom : chaîne. Âge : entier naturel. Poids : réel.
2. et 3. Le programme complet :

PYTHON

```python
prenom = input("Prénom ? ")
age = int(input("Âge ? "))
poids = float(input("Poids ? "))
paye = True
print(prenom, age, poids, paye)
print(type(prenom), type(age), type(poids), type(paye))
```

CE QUI S'AFFICHE

```
Prénom ? Léa
Âge ? 19
Poids ? 58.5
Léa 19 58.5 True
<class 'str'> <class 'int'> <class 'float'> <class 'bool'>
```

1. `paye` est un booléen : on écrit `True`, sans guillemets. Avec `"True"`, ce serait une chaîne.
2. Les quatre types correspondent : `str` pour le prénom, `int` pour l'âge, `float` pour le poids, `bool` pour le paiement.

Exercice 2.1.8 — Type et valeur d'une expression ★★★

Pour chaque expression Python, donnez sa valeur et son type.

1. `7 * 2`
2. `7 / 2`
3. `8 / 2`
4. `3 > 2`
5. `2.0 * 3`

CORRIGÉ

La méthode. La division `/` donne toujours un réel, même quand elle tombe juste. Un calcul avec un réel donne un réel.

Expression	Valeur	Type
`7 * 2`	14	`int`
`7 / 2`	3.5	`float`
`8 / 2`	4.0	`float`
`3 > 2`	True	`bool`
`2.0 * 3`	6.0	`float`

PYTHON

```python
print(7 * 2, 7 / 2, 8 / 2, 3 > 2, 2.0 * 3)
```

CE QUI S'AFFICHE

```
14 3.5 4.0 True 6.0
```

Exercice 2.1.9 — Une application de livraison ★★★

Une application suit des colis. Un développeur a écrit ces affectations :

PYTHON

```python
nb_colis = 4
poids = 2.5
livre = "non"
code_suivi = 12
temp_camion = -18
```

Le code de suivi réel est `FR0012AB`.

1. Donnez le bon type de chaque donnée.
2. Trouvez les deux affectations mal choisies, et corrigez-les.

CORRIGÉ

La méthode. Pour chaque donnée, on revient à la question : que contient-elle, et que fait-on avec ?

1. Les types :
   - `nb_colis` : entier naturel ;
   - `poids` : réel ;
   - `livre` : booléen ;
   - `code_suivi` : chaîne, il contient des lettres ;
   - `temp_camion` : entier relatif.
2. Deux erreurs :
   - `livre = "non"` : la réponse est oui ou non, c'est un booléen. On écrit `livre = False` .
   - `code_suivi = 12` : le vrai code a des lettres et des 0 au début. On écrit `code_suivi = "FR0012AB"` .

PYTHON

```python
livre = False
code_suivi = "FR0012AB"
print(type(livre), type(code_suivi))
```

CE QUI S'AFFICHE

```
<class 'bool'> <class 'str'>
```

2.2 — Les limites de la machine

‹ Revoir le cours 2.2

★★★ **Niveau 1 — Appliquer**

Exercice 2.2.1 — Combien de valeurs ? ★★★

Complétez le tableau. Utilisez le tableau des puissances de 2 des rappels.

Nombre de bits	Nombre de valeurs	Plus grand entier naturel
4		
8		
10		

CORRIGÉ

La méthode. Avec n bits, on a 2^n valeurs. On commence à 0, donc le plus grand est $2^n - 1$.

Nombre de bits	Nombre de valeurs	Plus grand entier naturel
4	16	15
8	256	255
10	1 024	1 023

Exercice 2.2.2 — Les grandes tailles ★☆☆ Sur machine

1. Quelle est la plus grande valeur d'un entier naturel sur 16 bits ?
2. Et sur 32 bits ?
3. Exécutez ce programme pour vérifier vos deux réponses. En Python, la puissance s'écrit `**` : `2 ** 16` veut dire 2^{16}.

PYTHON

```python
print(2 ** 16 - 1)
print(2 ** 32 - 1)
```

CORRIGÉ

La méthode. On calcule $2^n - 1$ avec le tableau des puissances de 2.

1. $2^{16} - 1 = 65\,535$.
2. $2^{32} - 1 = 4\,294\,967\,295$.
3. En Python :

PYTHON

```python
print(2 ** 16 - 1)
print(2 ** 32 - 1)
```

CE QUI S'AFFICHE

```
65535
4294967295
```

Exercice 2.2.3 — Exact ou approché ? ★☆☆

Dans la machine, chaque valeur est-elle rangée exactement, ou à peu près ?

1. L'entier 42.
2. Le réel 0,1.
3. L'entier 1024.
4. Le réel 0,3.
5. Le booléen Vrai.

CORRIGÉ

La méthode. Un entier dans son domaine est exact. Un booléen aussi : il n'a que deux valeurs. Un réel comme 0,1 ou 0,3 n'a pas d'écriture binaire finie : il est approché.

1. Exact.
2. Approché.
3. Exact.
4. Approché.
5. Exact.

★★☆ Niveau 2 — S'entraîner

Exercice 2.2.4 — Vérifier des bornes ★★☆

1. Un entier relatif sur 16 bits va de -32 768 à 32 767. Vérifiez ces deux bornes avec la règle du cours.
2. La distance de la Terre au Soleil fait environ 150 000 000 km. Tient-elle dans un entier naturel sur 32 bits ?
3. Un compteur est un entier naturel sur 8 bits. Après 255, il repart à 0 : 255 + 1 donne 0, 255 + 2 donne 1. Il vaut 250. Que donne 250 + 10 ?

CORRIGÉ

La méthode. Un entier relatif sur n bits va de -2^{n-1} à $2^{n-1} - 1$. Un entier naturel va de 0 à $2^n - 1$.

1. Avec n = 16 : $2^{15} = 32\,768$. On va donc de -32 768 à 32 767.
2. Oui. $2^{32} - 1 = 4\,294\,967\,295$, bien plus que 150 000 000.

3. 250 + 10 = 260 dépasse 255 : c'est un dépassement de capacité. On compte : 255 + 1 donne 0, donc 250 + 6 donne 0. Il reste 4 à ajouter : on obtient 4. C'est aussi 260 – 256, car le compteur a 256 valeurs.

La vérification en Python utilise `**`, la puissance, et `<=`, « inférieur ou égal ». Vous les verrez au chapitre 3 : ici, lisez seulement le résultat.

PYTHON

```python
print(-2 ** 15, 2 ** 15 - 1)
print(150000000 <= 2 ** 32 - 1)
print(250 + 10 - 256)
```

CE QUI S'AFFICHE

```
-32768 32767
True
4
```

Exercice 2.2.5 — L'addition qui tombe faux ★★☆ Sur machine

1. Sans ordinateur, que devrait afficher chaque ligne de ce programme ?
2. Exécutez-le. Notez ce qui s'affiche vraiment.
3. Expliquez la différence.
4. Modifiez la dernière ligne pour que le test affiche `True`.

PYTHON

```python
print(0.2 + 0.4)
s = 0.1 + 0.1 + 0.1
print(s)
print(s == 0.3)
```

CORRIGÉ

La méthode. On compare ce que disent les maths et ce que range la machine.

1. En maths : 0,6, puis 0,3, puis `True`.
2. La machine affiche :

PYTHON

```python
print(0.2 + 0.4)
s = 0.1 + 0.1 + 0.1
print(s)
print(s == 0.3)
```

CE QUI S'AFFICHE

```
0.6000000000000001
0.30000000000000004
False
```

1. Les nombres 0,1, 0,2 et 0,4 n'ont pas d'écriture binaire finie. La machine les range à peu près. Les petites erreurs s'ajoutent.
2. On ne teste jamais l'égalité exacte de deux réels. On arrondit d'abord :

PYTHON

```python
s = 0.1 + 0.1 + 0.1
print(round(s, 2) == 0.3)
```

CE QUI S'AFFICHE

```
True
```

Exercice 2.2.6 — Combien de bits ? ★★☆

Pour chaque donnée, choisissez le plus petit nombre de bits qui suffit : 8, 16 ou 32. Dites aussi si c'est un entier naturel ou relatif.

1. L'âge d'une personne, de 0 à 130 ans.
2. Le nombre de visiteurs d'un site web en un jour, jusqu'à 50 000.
3. La population de la France, environ 68 millions.
4. Une température extérieure, de -40 à 60 °C.

CORRIGÉ

La méthode. On compare la plus grande valeur avec les tableaux du cours. Un entier relatif sert seulement si la donnée peut être négative.

1. 8 bits, entier naturel : 130 ≤ 255.
2. 16 bits, entier naturel : 50 000 ≤ 65 535, mais dépasse 255.
3. 32 bits, entier naturel : 68 000 000 dépasse 65 535.
4. 8 bits, entier relatif : de -128 à 127 suffit pour -40 à 60.

★★★ **Niveau 3 — Aller plus loin**

Exercice 2.2.7 — Compter en centimes ★★★ Sur machine

Une caisse additionne trois articles : 4,35 €, 2,10 € et 0,55 €.

1. Calculez le total à la main.
2. Exécutez le programme. Que remarquez-vous ?
3. Beaucoup de logiciels de caisse comptent en centimes, dans des entiers. Pourquoi ?

PYTHON

```python
total = 4.35 + 2.10 + 0.55
print(total)
total_centimes = 435 + 210 + 55
print(total_centimes)
```

CORRIGÉ

La méthode. On compare le calcul avec des réels et le même calcul avec des entiers.

1. 4,35 + 2,10 + 0,55 = 7,00 €.
2. Le programme affiche :

PYTHON

```python
total = 4.35 + 2.10 + 0.55
print(total)
total_centimes = 435 + 210 + 55
print(total_centimes)
```

CE QUI S'AFFICHE

```
6.999999999999999
700
```

Avec des réels, le total n'est pas exactement 7 : les centimes sont rangés à peu près. Avec des entiers, 700 centimes est exact.

1. Les entiers sont exacts dans leur domaine. En comptant en centimes, la caisse ne fait jamais d'erreur d'arrondi. Elle divise par 100 seulement pour afficher.

Exercice 2.2.8 — Une valeur trop grande ★★★

Un programme range une vitesse dans un entier relatif sur 16 bits. Sur un nouveau modèle de drone, la vitesse mesurée peut atteindre 40 000.

1. Quelle est la plus grande valeur d'un entier relatif sur 16 bits ?
2. La valeur 40 000 tient-elle ? Que se passe-t-il sinon ?
3. Pour un entier relatif, quel nombre de bits faut-il choisir : 16 ou 32 ?
4. Quelle vérification aurait évité le problème ?

CORRIGÉ

La méthode. On compare la valeur avec la borne du type.

1. $2^{15} - 1 = 32\,767$.
2. Non. 40 000 dépasse 32 767 : c'est un dépassement de capacité. La valeur rangée devient fausse.
3. 32 bits : un entier relatif va alors jusqu'à 2 147 483 647.
4. Vérifier que la valeur reste dans le domaine du type avant de la ranger. C'est le défaut qui a détruit la fusée Ariane 5 en 1996.

Exercice 2.2.9 — Comparer deux prix ★★★ Sur machine

Un site vérifie que le prix de trois câbles à 0,10 € fait bien 0,30 €.

1. Exécutez le programme. Le test est-il réussi ?
2. Expliquez le résultat.
3. Corrigez la ligne du test pour qu'il réussisse.

PYTHON

```python
prix = 0.1 * 3
print(prix)
print(prix == 0.3)
```

CORRIGÉ

La méthode. On ne compare jamais deux réels avec `==`. On arrondit d'abord au centime.

1. Le test échoue :

PYTHON

```python
prix = 0.1 * 3
print(prix)
print(prix == 0.3)
```

CE QUI S'AFFICHE

```
0.30000000000000004
False
```

1. 0,1 est rangé à peu près. Multiplié par 3, la petite erreur se voit.
2. On arrondit à 2 décimales avant de comparer :

PYTHON

```python
prix = 0.1 * 3
print(round(prix, 2) == 0.3)
```

CE QUI S'AFFICHE

```
True
```

2.3 — Les chaînes de caractères

‹ Revoir le cours 2.3

★☆☆ **Niveau 1 — Appliquer**

Exercice 2.3.1 — Le code d'un badge ★☆☆

Dans un lycée, chaque badge porte un code comme `badge ← "B3-2041"`. Le premier caractère est le bâtiment. Le deuxième est l'étage.

1. Quelle est la longueur du code ?
2. Quelle instruction extrait le bâtiment ? Que vaut-elle ?
3. Quelle instruction extrait l'étage ? Que vaut-elle ?
4. Écrivez l'instruction pseudo-code qui affiche `Bâtiment B étage 3`, avec `Écrire` et des virgules.

CORRIGÉ

La méthode. On extrait un caractère avec son indice. Le premier caractère a l'indice `0`.

1. 7 caractères : le tiret compte.
2. `badge[0]`, qui vaut `B`.
3. `badge[1]`, qui vaut `3`. C'est le caractère `"3"`, pas le nombre 3.
4. `Écrire "Bâtiment", badge[0], "étage", badge[1]`

PYTHON

```python
badge = "B3-2041"
print(len(badge))
print("Bâtiment", badge[0], "étage", badge[1])
```

CE QUI S'AFFICHE

```
7
Bâtiment B étage 3
```

Exercice 2.3.2 — Compter les caractères ★☆☆

Donnez la longueur de chaque chaîne.

1. `"BTS SIO"`
2. `"192.168.1.10"`
3. `"Bonjour !"`
4. `""`

CORRIGÉ

La méthode. On compte tous les caractères : les espaces et la ponctuation comptent aussi.

1. 7 : l'espace compte.
2. 12 : les points comptent.
3. 9 : l'espace et le point d'exclamation comptent.
4. `0` : c'est la chaîne vide, elle n'a aucun caractère.

PYTHON

```python
print(len("BTS SIO"), len("192.168.1.10"), len("Bonjour !"), len(""))
```

CE QUI S'AFFICHE

```
7 12 9 0
```

Exercice 2.3.3 — Texte ou nombre ? ★☆☆

Pour chaque valeur, dites si c'est une chaîne ou un nombre, et si on peut calculer avec.

1. `"123"`
2. `123`
3. `"12.5"`
4. `12.5`

CORRIGÉ

La méthode. Les guillemets font une chaîne, quel que soit leur contenu.

1. Chaîne de 3 caractères : on ne calcule pas avec.
2. Nombre entier : on peut calculer.
3. Chaîne de 4 caractères : on ne calcule pas avec.
4. Nombre réel : on peut calculer.

★★☆ Niveau 2 — S'entraîner

Exercice 2.3.4 — Une adresse mail ★★☆ Sur machine

On a `mail = "lea.martin@lycee.fr"`.

1. Quelle est la longueur de `mail` ?
2. À quel indice est le caractère `@` ? Comptez à la main.
3. Écrivez les instructions Python qui affichent la longueur, le premier caractère, le dernier et le caractère d'indice 10.
4. Exécutez pour vérifier.

CORRIGÉ

La méthode. On compte à partir de 0. Le dernier caractère a l'indice `len(mail) - 1`.

1. 19 caractères.
2. Le `@` est à l'indice 10 :

Caractère	l	e	a	.	m	a	r	t	i	n	@
Indice	0	1	2	3	4	5	6	7	8	9	10

1. et 4. En Python :

PYTHON

```python
mail = "lea.martin@lycee.fr"
print(len(mail))
print(mail[0])
print(mail[len(mail) - 1])
print(mail[10])
```

CE QUI S'AFFICHE

```
19
l
r
@
```

Exercice 2.3.5 — La trace d'une chaîne ★★☆

Faites la trace de cet algorithme. Donnez la valeur de chaque variable à la fin.

PSEUDO-CODE

```
mot ← "RESEAU"
n ← longueur(mot)
premier ← mot[0]
dernier ← mot[n - 1]
quatrieme ← mot[3]
```

CORRIGÉ

La méthode. Une colonne par variable. On compte les indices de `RESEAU` : R=0, E=1, S=2, E=3, A=4, U=5.

Étape	mot	n	premier	dernier	quatrieme
1	"RESEAU"	–	–	–	–
2	"RESEAU"	6	–	–	–
3	"RESEAU"	6	"R"	–	–
4	"RESEAU"	6	"R"	"U"	–
5	"RESEAU"	6	"R"	"U"	"E"

PYTHON

```python
mot = "RESEAU"
n = len(mot)
premier = mot[0]
dernier = mot[n - 1]
quatrieme = mot[3]
print(n, premier, dernier, quatrieme)
```

CE QUI S'AFFICHE

```
6 R U E
```

Exercice 2.3.6 — Deux erreurs d'indice ★★☆

Un élève veut afficher la première et la dernière lettre de `"Python"`. Il écrit :

PYTHON

```python
mot = "Python"
print(mot[1])
print(mot[len(mot)])
```

1. Que va afficher le premier `print` ? Est-ce la première lettre ?
2. Que se passe-t-il avec le deuxième `print` ?
3. Corrigez les deux lignes.

CORRIGÉ

La méthode. Le premier indice est 0. Le dernier est la longueur moins 1.

1. `mot[1]` vaut `y` : c'est la deuxième lettre.
2. `len(mot)` vaut 6, et l'indice 6 n'existe pas. Python s'arrête sur une erreur `IndexError`.

3. La correction :

PYTHON

```python
mot = "Python"
print(mot[0])
print(mot[len(mot) - 1])
```

CE QUI S'AFFICHE

```
P
n
```

★★★ **Niveau 3 — Aller plus loin**

Exercice 2.3.7 — Corriger un prénom ★★★

Une fiche contient `prenom ← "lea"`. On veut une majuscule au début. Un élève écrit :

PSEUDO-CODE

```
prenom ← "lea"
prenom[0] ← "L"
```

1. Que se passe-t-il à la deuxième ligne ? Pourquoi ?
2. Comment obtenir `"Lea"` dans la variable `prenom`, avec une seule affectation ?
3. Expliquez la différence entre « modifier un caractère » et « ranger une nouvelle chaîne dans la variable ».

CORRIGÉ

La méthode. Une chaîne n'est pas un tableau : on lit ses caractères, on ne les modifie pas. Mais la variable, elle, peut recevoir une nouvelle valeur.

1. C'est une erreur. On ne modifie pas un caractère d'une chaîne. En Python, le message est `TypeError`.
2. `prenom ← "Lea"` : on range une nouvelle chaîne entière dans la variable. L'ancienne valeur est perdue, comme pour toute affectation.
3. Modifier un caractère voudrait changer l'intérieur de la chaîne : c'est impossible. Ranger une nouvelle chaîne remplace toute la valeur de la variable : c'est une affectation normale.

PYTHON

```python
prenom = "lea"
prenom = "Lea"
print(prenom, len(prenom))
```

CE QUI S'AFFICHE

```
Lea 3
```

Exercice 2.3.8 — Première et dernière lettre ★★★ Sur machine

1. Écrivez un programme qui lit un mot au clavier, puis affiche sa longueur, sa première lettre et sa dernière lettre. Écrivez d'abord le pseudo-code, puis le Python. Testez avec `Python`.
2. Testez avec un mot d'une seule lettre, `A`. Que valent la première et la dernière lettre ? Pourquoi ?
3. L'utilisateur appuie sur Entrée sans rien taper. Que vaut la longueur ? Que se passe-t-il à l'affichage ? Pourquoi ?

CORRIGÉ

La méthode. On lit une chaîne : pas de conversion. Le dernier indice vaut la longueur moins 1.

PSEUDO-CODE

```
Écrire "Mot ?"
Lire mot
n ← longueur(mot)
Écrire n, mot[0], mot[n - 1]
```

PYTHON

```python
mot = input("Mot ? ")
n = len(mot)
print(n, mot[0], mot[n - 1])
```

CE QUI S'AFFICHE

```
Mot ? Python
6 P n
```

1. Avec `A`, la longueur vaut 1. L'indice du premier et du dernier caractère est le même : 0. Les deux valent `A`.

2. La chaîne vide `""` a une longueur 0. Il n'y a aucun caractère : `mot[0]` n'existe pas. Python s'arrête sur une erreur `IndexError`. Un programme sérieux vérifie la longueur avant d'extraire un caractère : vous saurez le faire au chapitre 4.

Exercice 2.3.9 — L'avant-dernier caractère ★★★ Sur machine

Un code d'accès a toujours au moins deux caractères, par exemple `4F7A2C`. Le service informatique veut afficher son avant-dernier caractère.

1. Pour `4F7A2C`, quel est l'avant-dernier caractère ? À quel indice est-il ?
2. Donnez l'indice de l'avant-dernier caractère pour une chaîne quelconque.
3. Écrivez le programme Python qui lit un code et affiche son avant-dernier caractère. Testez avec `4F7A2C`, puis avec `XY`.

CORRIGÉ

La méthode. Le dernier est à l'indice `longueur - 1`. L'avant-dernier est juste avant, à `longueur - 2`.

1. `2`, à l'indice 4. La chaîne a 6 caractères : 6 – 2 = 4.
2. `longueur(code) - 2`.
3. En Python :

PYTHON

```
code = input("Code ? ")
print(code[len(code) - 2])
```

CE QUI S'AFFICHE

```
Code ? 4F7A2C
2
```

Avec `XY`, la longueur vaut 2 : l'indice 0 donne `X`.

2.4 — Changer de type : le transtypage

‹ Revoir le cours 2.4

★☆☆ **Niveau 1 — Appliquer**

Exercice 2.4.1 — Les conversions de base ★☆☆

Donnez le résultat de chaque conversion.

1. `int("42")`
2. `float("3.5")`
3. `str(42)`
4. `int(3.9)`
5. `int(True)`

CORRIGÉ

La méthode. `int` donne un entier, `float` un réel, `str` une chaîne. `int` coupe la partie après la virgule, il n'arrondit pas.

1. L'entier 42.
2. Le réel 3.5.
3. La chaîne `"42"`.
4. L'entier 3.
5. L'entier 1.

PYTHON

```python
print(int("42"), float("3.5"), str(42), int(3.9), int(True))
```

CE QUI S'AFFICHE

```
42 3.5 42 3 1
```

À l'écran, `str(42)` s'affiche comme 42 : `print` n'affiche pas les guillemets.

Exercice 2.4.2 — Quelle conversion après input ? ★☆☆

Pour chaque donnée lue avec `input`, dites quelle conversion il faut : aucune, `int` ou `float`.

1. L'âge.
2. Le prix d'un article, par exemple 12.99.
3. Le prénom.
4. Le code postal.
5. Le nombre d'articles commandés.

CORRIGÉ

La méthode. `input` renvoie toujours du texte. On convertit seulement si on va calculer.

1. `int` : l'âge est un entier.
2. `float` : le prix a une virgule.
3. Aucune : c'est du texte.
4. Aucune : on ne calcule pas sur un code postal, et on garde le `0` du début.
5. `int`.

Exercice 2.4.3 — Additionner ou coller ? ★☆☆

Que donne chaque expression en Python ?

1. `"10" + "5"`
2. `10 + 5`
3. `int("10") + int("5")`

CORRIGÉ

La méthode. Avec deux chaînes, `+` les colle. Avec deux nombres, il les additionne.

1. `"105"` : deux chaînes collées.
2. `15`.
3. `15` : on convertit d'abord les deux chaînes en entiers.

PYTHON

```python
print("10" + "5", 10 + 5, int("10") + int("5"))
```

CE QUI S'AFFICHE

```
105 15 15
```

★★☆ Niveau 2 — S'entraîner

Exercice 2.4.4 — Prévoir un transtypage ★★☆ Sur machine

1. Pour chaque ligne, écrivez ce que Python va afficher, sans l'exécuter.
2. Exécutez le programme pour vérifier.
3. Que donnerait `int("3.5")` ? Pourquoi ?

PYTHON

```python
print(int("42") + 8)
print("42" + "8")
print(float("3.5") * 2)
print(int(9.99))
print(str(7) + "7")
print(int(True) + 1)
```

CORRIGÉ

La méthode. Pour chaque `+`, on regarde le type des deux côtés : deux nombres s'additionnent, deux chaînes se collent.

PYTHON

```python
print(int("42") + 8)
print("42" + "8")
print(float("3.5") * 2)
print(int(9.99))
print(str(7) + "7")
print(int(True) + 1)
```

CE QUI S'AFFICHE

```
50
428
7.0
9
77
2
```

- Ligne 1 : `int("42")` est le nombre 42. On ajoute 8.
- Ligne 2 : deux chaînes, elles sont collées.
- Ligne 4 : `int` coupe la partie après la virgule. Il n'arrondit pas.
- Ligne 5 : `str(7)` est la chaîne `"7"`. On colle deux chaînes.
- Ligne 6 : `True` devient 1.

1. `int("3.5")` s'arrête sur une erreur `ValueError`. `int` n'accepte que l'écriture d'un entier.

Exercice 2.4.5 — Le total d'une commande ★★☆ Sur machine

Complétez le programme. Il lit le prix d'un article et la quantité. Il affiche ensuite un seul message, collé avec `+`, par exemple `Total : 37.5 €`.

Testez avec 12.5 et 3.

PYTHON

```python
prix = input("Prix ? ")       # à corriger
qte = input("Quantité ? ")    # à corriger
total = 0                     # à remplacer par le calcul
print("Total :", total)       # à remplacer par un message collé avec +
```

CORRIGÉ

La méthode. On convertit le texte lu en nombres pour calculer. Puis on reconvertit le résultat en texte pour le coller dans le message.

PYTHON

```python
prix = float(input("Prix ? "))
qte = int(input("Quantité ? "))
total = prix * qte
print("Total : " + str(total) + " €")
```

CE QUI S'AFFICHE

```
Prix ? 12.5
Quantité ? 3
Total : 37.5 €
```

Attention

`"Total : " + total` s'arrête sur une erreur : on ne colle pas une chaîne et un nombre. Il faut `str(total)`.

Exercice 2.4.6 — Corriger les conversions ★★☆

Chaque ligne s'arrête sur une erreur. Expliquez pourquoi, puis corrigez-la.

PYTHON

```python
print("Total : " + 25)
print(float("3,5"))
print(int("12.5"))
```

CORRIGÉ

La méthode. On lit le message d'erreur : il dit quel type pose problème.

1. On colle une chaîne et un nombre. On écrit `"Total : " + str(25)`.

2. Python attend un point, pas une virgule. On écrit `float("3.5")`.
3. `int` n'accepte pas un texte avec un point. On passe par un réel : `int(float("12.5"))`.

PYTHON

```python
print("Total : " + str(25))
print(float("3.5"))
print(int(float("12.5")))
```

CE QUI S'AFFICHE

```
Total : 25
3.5
12
```

★★★ Niveau 3 — Aller plus loin

Exercice 2.4.7 — Minutes depuis minuit ★★★ Sur machine

Un badge de pointage lit l'heure et les minutes d'arrivée sous forme de texte, par exemple `"14"` et `"35"`. On veut le nombre de minutes écoulées depuis minuit.

1. Calculez-le à la main pour 14 h 35.
2. Écrivez l'algorithme en pseudo-code, avec deux `Lire`.
3. Écrivez le programme Python, et testez avec 14 et 35.

CORRIGÉ

La méthode. Une heure fait 60 minutes. On convertit les deux textes en entiers, puis on calcule heures × 60 + minutes.

1. 14 × 60 + 35 = 840 + 35 = 875 minutes.
2. En pseudo-code :

PSEUDO-CODE

```
Lire h
Lire m
total ← entier(h) * 60 + entier(m)
Écrire total
```

1. En Python :

PYTHON

```python
h = input("Heure ? ")
m = input("Minutes ? ")
total = int(h) * 60 + int(m)
print(total)
```

CE QUI S'AFFICHE

```
Heure ? 14
Minutes ? 35
875
```

Exercice 2.4.8 — La note coupée ★★★ Sur machine

Une note est tapée au clavier, par exemple `15.75`. Le logiciel doit afficher sa partie entière, ici 15.

1. Pourquoi `int(input("Note ? "))` s'arrête-t-il sur une erreur avec `15.75` ?
2. Complétez le programme en trois lignes : on lit le texte, on le change en réel, puis le réel en entier.
3. `int` arrondit-il, ou coupe-t-il ? Testez avec `15.75`.

PYTHON

```python
texte = input("Note ? ")
reel = 0     # à compléter
note = 0     # à compléter
print(note)
```

CORRIGÉ

La méthode. `int` n'accepte pas le texte `"15.75"`. On le change d'abord en réel, puis le réel en entier.

1. `input` renvoie le texte `"15.75"`, et `int` n'accepte que l'écriture d'un entier.
2. `reel = float(texte)`, puis `note = int(reel)`.
3. Il coupe : 15,75 donne 15, pas 16.

PYTHON

```python
texte = input("Note ? ")
reel = float(texte)
note = int(reel)
print(note)
```

CE QUI S'AFFICHE

```
Note ? 15.75
15
```

Exercice 2.4.9 — Le code postal qui perd son zéro ★★★

Un développeur range les codes postaux dans des entiers. Pour Bourg-en-Bresse, il écrit `cp = int("01000")`.

1. Que vaut `cp` ?
2. Que donne ensuite `str(cp)` ? Est-ce le bon code postal ?
3. Quel type faut-il choisir ? Pourquoi ?
4. Le même développeur range l'âge en texte : `age = "25"`. Que donne `age + "1"` ? Et `int(age) + 1` ?
5. Écrivez une règle qui dit quand choisir une chaîne et quand choisir un nombre.

CORRIGÉ

La méthode. On suit la valeur à chaque conversion.

1. `cp` vaut 1000 : un entier n'a pas de 0 au début.
2. `str(cp)` donne `"1000"`. Ce n'est plus le bon code : le 0 est perdu pour toujours.
3. Une chaîne de caractères. On ne calcule jamais sur un code postal, et la chaîne garde le 0.
4. `age + "1"` colle deux chaînes : `"251"`. `int(age) + 1` donne 26.
5. Si on calcule avec la donnée, on choisit un nombre. Si on ne fait que la lire ou l'afficher, comme un code postal ou un numéro de téléphone, on choisit une chaîne.

PYTHON

```python
cp = int("01000")
print(cp, str(cp))
age = "25"
print(age + "1", int(age) + 1)
```

CE QUI S'AFFICHE

```
1000 1000
251 26
```

3.1 — Les opérateurs numériques

‹ Revoir le cours 3.1

★★★ **Niveau 1 — Appliquer**

Exercice 3.1.1 — Quotient et reste ★★★

1. Calculez `47 div 5` et `47 mod 5`.
2. Calculez `100 mod 7`.
3. Le nombre 2026 est-il pair ? Écrivez le test en pseudo-code.

CORRIGÉ

La méthode. On écrit la division euclidienne : dividende = diviseur × quotient + reste, avec un reste plus petit que le diviseur.

1. 47 = 5 × 9 + 2. Donc `47 div 5` vaut 9 et `47 mod 5` vaut 2.
2. 100 = 7 × 14 + 2. Donc `100 mod 7` vaut 2.
3. Le test est `2026 mod 2 = 0`. Il est vrai : 2026 est pair.

PYTHON

```python
print(47 // 5, 47 % 5)
print(100 % 7)
print(2026 % 2 == 0)
```

CE QUI S'AFFICHE

```
9 2
2
True
```

Exercice 3.1.2 — Divisible ou pas ★★★

Donnez la valeur, Vrai ou Faux, de chaque test.

1. `27 mod 3 = 0`
2. `28 mod 3 = 0`
3. `2025 mod 2 = 0`
4. `343 mod 7 = 0`

CORRIGÉ

La méthode. On calcule le reste. Le test est Vrai seulement si le reste vaut 0.

1. 27 = 3 × 9 + 0 : Vrai. 27 est divisible par 3.
2. 28 = 3 × 9 + 1 : Faux.
3. 2025 = 2 × 1012 + 1 : Faux. 2025 est impair.
4. 343 = 7 × 49 + 0 : Vrai.

PYTHON

```
print(27 % 3 == 0, 28 % 3 == 0, 2025 % 2 == 0, 343 % 7 == 0)
```

CE QUI S'AFFICHE

```
True False False True
```

Exercice 3.1.3 — Écrire une formule en ligne ★☆☆

1. Écrivez en ligne, avec les opérateurs du cours :
   1. $\dfrac{a+b}{a-b}$
   2. $3x^2+1$
2. Écrivez en maths l'expression en ligne `a / b + c`.

CORRIGÉ

La méthode. Chaque trait de fraction devient une division, avec des parenthèses autour du haut et du bas s'ils contiennent un calcul.

1. Les écritures en ligne :
   1. `(a + b) / (a - b)`
   2. `3 * x ^ 2 + 1` en pseudo-code, `3 * x ** 2 + 1` en Python
2. La division passe avant l'addition : c'est $\dfrac{a}{b}+c$, et non $\dfrac{a}{b+c}$.

Attention

`a + b / a - b` n'est pas la formule a. Sans parenthèses, Python calcule d'abord `b / a`.

★★☆ **Niveau 2 — S'entraîner**

Exercice 3.1.4 — Convertir une durée ★★☆ Sur machine

Un serveur affiche depuis combien de secondes il tourne. On veut l'afficher en heures, minutes et secondes.

1. Complétez le programme. Il lit un nombre de secondes et affiche les heures, les minutes et les secondes.
2. Testez-le avec 4000 secondes.

PYTHON

```
duree = int(input("Durée en secondes ? "))
heures = 0      # à compléter
reste = 0       # à compléter
minutes = 0     # à compléter
secondes = 0    # à compléter
print(heures, "h", minutes, "min", secondes, "s")
```

CORRIGÉ

La méthode. Une heure fait 3 600 secondes, une minute fait 60 secondes. On prend le quotient pour les heures, puis on travaille sur le reste.

PYTHON

```
duree = int(input("Durée en secondes ? "))
heures = duree // 3600
reste = duree % 3600
minutes = reste // 60
secondes = reste % 60
print(heures, "h", minutes, "min", secondes, "s")
```

CE QUI S'AFFICHE

```
Durée en secondes ? 4000
1 h 6 min 40 s
```

Le calcul : 4000 = 3600 × 1 + 400, puis 400 = 60 × 6 + 40.

Exercice 3.1.5 — La moyenne pondérée ★★☆

À un contrôle, la note `a` compte 3 fois, et la note `b` compte 2 fois. On dit que `a` a le **coefficient** 3, et `b` le coefficient 2. Pour la moyenne, on multiplie chaque note par son coefficient, on additionne, puis on divise par 3 + 2 = 5.

1. Écrivez la moyenne en ligne.
2. Calculez-la pour `a = 6` et `b = 2`, puis pour `a = 12` et `b = 15`.
3. Un élève écrit `3 * a + 2 * b / 5`. Que calcule-t-il vraiment pour `a = 12` et `b = 15` ?

CORRIGÉ

La méthode. La division porte sur toute la somme : il faut des parenthèses autour de la somme.

1. `(3 * a + 2 * b) / 5`
2. Pour 6 et 2 : (18 + 4) / 5 = 4,4. Pour 12 et 15 : (36 + 30) / 5 = 13,2.
3. Sans parenthèses, la division passe d'abord : 36 + 30 / 5 = 36 + 6 = 42.

PYTHON

```
a = 12
b = 15
print((3 * a + 2 * b) / 5)
print(3 * a + 2 * b / 5)
```

CE QUI S'AFFICHE

```
13.2
42.0
```

Exercice 3.1.6 — Quel jour dans 100 jours ? ★★☆

On numérote les jours : 0 pour lundi, 1 pour mardi, 2 pour mercredi, 3 pour jeudi, 4 pour vendredi, 5 pour samedi, 6 pour dimanche. Aujourd'hui, `jour ← 0` : c'est lundi.

1. Pourquoi le reste de la division par 7 donne-t-il le jour de la semaine ?
2. Écrivez l'expression qui donne le numéro du jour dans `n` jours.
3. Quel jour serons-nous dans 100 jours ? Dans 31 jours ?

CORRIGÉ

La méthode. Tous les 7 jours, on retombe sur le même jour. Seul le reste de la division par 7 compte.

1. 7 jours plus tard, c'est encore lundi. On peut donc retirer autant de semaines entières qu'on veut : il ne reste que le reste.
2. `(jour + n) mod 7`
3. Dans 100 jours : 100 = 7 × 14 + 2, le reste vaut 2 : mercredi. Dans 31 jours : 31 = 7 × 4 + 3, le reste vaut 3 : jeudi.

PYTHON

```
jour = 0
print((jour + 100) % 7, (jour + 31) % 7)
```

CE QUI S'AFFICHE

```
2 3
```

★★★ **Niveau 3 — Aller plus loin**

Exercice 3.1.7 — Les chiffres d'un nombre ★★★ Sur machine

On lit un entier de 3 chiffres, par exemple 472. On veut ses chiffres, sans le transformer en chaîne.

1. Quelle expression donne le chiffre des unités ?
2. Quelle expression donne le chiffre des dizaines ?
3. Quelle expression donne le chiffre des centaines ?
4. Écrivez le programme Python qui affiche les trois chiffres, puis leur somme. Testez-le avec 472.

CORRIGÉ

La méthode. Le reste de la division par 10 donne le dernier chiffre. Le quotient par 10 « efface » le dernier chiffre.

1. Unités : `n mod 10`. Pour 472 : 2.
2. Dizaines : `(n div 10) mod 10`. 472 div 10 = 47, puis 47 mod 10 = 7.
3. Centaines : `n div 100`. Pour 472 : 4.
4. Le programme :

PYTHON

```python
n = int(input("Nombre à 3 chiffres ? "))
unites = n % 10
dizaines = (n // 10) % 10
centaines = n // 100
print(centaines, dizaines, unites)
print("Somme :", centaines + dizaines + unites)
```

CE QUI S'AFFICHE

```
Nombre à 3 chiffres ? 472
4 7 2
Somme : 13
```

Exercice 3.1.8 — La trace d'un calcul avec div et mod ★★★

PSEUDO-CODE

```
a ← 47
b ← 5
q ← a div b
r ← a mod b
a ← b
b ← r
q ← a div b
r ← a mod b
```

1. Faites la trace de cet algorithme : une colonne pour `a`, `b`, `q`, `r`.
2. Que valent les quatre variables à la fin ?
3. Vérifiez la première division : `5 × q + r` redonne-t-il 47 ?

CORRIGÉ

La méthode. On suit les lignes une par une. À chaque ligne, on utilise les valeurs de la ligne juste au-dessus.

Étape	a	b	q	r
1	47	–	–	–
2	47	5	–	–
3	47	5	9	–
4	47	5	9	2
5	5	5	9	2
6	5	2	9	2
7	5	2	2	2
8	5	2	2	1

1. À la fin : `a` vaut 5, `b` vaut 2, `q` vaut 2, `r` vaut 1.
2. Avec q = 9 et r = 2 : 5 × 9 + 2 = 47. Oui.

PYTHON

```python
a = 47
b = 5
q = a // b
r = a % b
a = b
b = r
q = a // b
r = a % b
print(a, b, q, r)
```

CE QUI S'AFFICHE

```
5 2 2 1
```

Exercice 3.1.9 — Coder une lettre ★★★ Sur machine

Cet exercice reprend le début d'un sujet de BTS. Chaque lettre a un **rang** : son numéro dans l'alphabet, en partant de 0.

A	B	C	D	E	F	G	H	I	J	K	L	M
0	1	2	3	4	5	6	7	8	9	10	11	12

N	O	P	Q	R	S	T	U	V	W	X	Y	Z
13	14	15	16	17	18	19	20	21	22	23	24	25

Pour coder la lettre de rang `n`, on calcule le reste de la division de $7n + 5$ par 26. Ce reste est le rang de la lettre codée.

1. Écrivez l'expression en ligne du rang codé, avec `mod`.
2. Codez la lettre C, de rang 2.
3. Codez la lettre T, de rang 19.
4. Vérifiez en Python.

CORRIGÉ

La méthode. On calcule $7n + 5$, puis on prend le reste de la division par 26.

1. `(7 * n + 5) mod 26`
2. Pour C : 7 × 2 + 5 = 19, et 19 = 26 × 0 + 19. Le rang 19 est la lettre T. C est codé par T.
3. Pour T : 7 × 19 + 5 = 138, et 138 = 26 × 5 + 8. Le rang 8 est la lettre I. T est codé par I.

PYTHON

```
print((7 * 2 + 5) % 26, (7 * 19 + 5) % 26)
```

CE QUI S'AFFICHE

```
19 8
```

Attention

Écrivez `(7 * n + 5) mod 26`, avec les parenthèses. Sinon, le `mod` ne porte que sur 5.

3.2 — Les fonctions mathématiques usuelles

‹ Revoir le cours 3.2

★☆☆ **Niveau 1 — Appliquer**

Exercice 3.2.1 — Trois arrondis ★☆☆ Sur machine

On a `x ← 7.456`.

1. Que donne l'arrondi au plus près, à 2 décimales ?
2. Que donne l'arrondi par défaut, à l'entier ?
3. Que donne l'arrondi par excès, à l'entier ?
4. Exécutez ce programme pour vérifier vos trois réponses.

PYTHON

```python
import math
x = 7.456
print(round(x, 2), math.floor(x), math.ceil(x))
```

CORRIGÉ

La méthode. Au plus près : on regarde le chiffre suivant. Par défaut : on va vers le bas. Par excès : on va vers le haut.

1. 7,46, car le chiffre suivant, 6, fait monter.
2. 7.
3. 8.

PYTHON

```python
import math
x = 7.456
print(round(x, 2), math.floor(x), math.ceil(x))
```

CE QUI S'AFFICHE

```
7.46 7 8
```

Exercice 3.2.2 — L'écart de température ★☆☆

Un capteur mesure -3 °C la nuit et 5 °C le jour.

1. Quel calcul donne l'écart entre les deux, sans signe ?
2. Que donne `abs(12 - 20)` ? Et `abs(20 - 12)` ?

CORRIGÉ

La méthode. La valeur absolue enlève le signe. L'écart ne dépend pas de l'ordre.

1. `abs(-3 - 5)`, qui vaut `abs(-8)`, soit 8 degrés.
2. Les deux valent 8.

PYTHON

```python
print(abs(-3 - 5), abs(12 - 20), abs(20 - 12))
```

CE QUI S'AFFICHE

```
8 8 8
```

Exercice 3.2.3 — Des racines carrées ★☆☆

1. Une salle carrée a une aire de 49 m². Quelle instruction Python donne la longueur de son côté ?
2. Que donne `math.sqrt(144)` ?
3. Que donne `round(math.sqrt(2), 2)` ?

CORRIGÉ

La méthode. Le côté d'un carré est la racine carrée de son aire. `math.sqrt` renvoie toujours un réel.

PYTHON

```python
import math
print(math.sqrt(49))
print(math.sqrt(144))
print(round(math.sqrt(2), 2))
```

CE QUI S'AFFICHE

```
7.0
12.0
1.41
```

1. Le côté mesure 7 m. 2. 12.0. 3. 1.41.

★★☆ **Niveau 2 — S'entraîner**

Exercice 3.2.4 — Arrondir au bon sens ★★☆ Sur machine

Un technicien range 130 câbles dans des cartons de 50. Un article coûte 19,987 € avant arrondi.

1. Combien de cartons faut-il ? Quelle fonction utiliser ?
2. Quel prix affiche-t-on, arrondi au centime le plus proche ?
3. Que donne l'arrondi par défaut de 19,987, à l'entier ?

Exécutez ce programme et comparez avec vos réponses :

PYTHON

```python
import math
print(math.ceil(130 / 50))
print(round(19.987, 2))
print(math.floor(19.987))
```

CORRIGÉ

La méthode. On se demande dans quel sens arrondir : vers le haut, vers le bas, ou au plus près.

1. 130 / 50 = 2,6. Un carton incomplet compte quand même : on arrondit **par excès**. Il faut 3 cartons, avec `math.ceil`.
2. On arrondit **au plus près** : 19,99 €, avec `round(19.987, 2)`.
3. L'arrondi **par défaut** donne 19, avec `math.floor`.

PYTHON

```python
import math
print(math.ceil(130 / 50))
print(round(19.987, 2))
print(math.floor(19.987))
```

CE QUI S'AFFICHE

```
3
19.99
19
```

Exercice 3.2.5 — Les pages d'un rapport ★★☆ Sur machine

Un rapport affiche les résultats d'une recherche, 20 lignes par page. Dans cet exercice, le nombre de lignes n'est jamais un multiple de 20 : la dernière page n'est jamais pleine.

1. Pour 53 lignes, combien faut-il de pages ?
2. Combien de lignes y a-t-il sur la dernière page ?
3. Écrivez un programme qui lit le nombre de lignes et affiche ces deux nombres. Testez avec 53.

CORRIGÉ

La méthode. Une page incomplète compte : on arrondit par excès. Les lignes de la dernière page sont le reste de la division par 20.

1. 53 / 20 = 2,65, arrondi par excès : 3 pages.
2. 53 = 20 × 2 + 13 : 13 lignes sur la dernière page.

PYTHON

```python
import math
lignes = int(input("Nombre de lignes ? "))
print("Pages :", math.ceil(lignes / 20))
print("Lignes sur la dernière page :", lignes % 20)
```

CE QUI S'AFFICHE

```
Nombre de lignes ? 53
Pages : 3
Lignes sur la dernière page : 13
```

> **Attention**
>
> Pour 60 lignes, un multiple de 20, le reste vaut 0, alors que la dernière page est pleine avec 20 lignes. C'est pour ça que l'énoncé écarte ce cas.

Exercice 3.2.6 — Le prix TTC ★★☆ Sur machine

Un article coûte 19,99 € hors taxes. Le prix TTC, toutes taxes comprises, est le prix hors taxes multiplié par 1,2.

1. Calculez à la main 19,99 × 1,2. Puis exécutez `print(19.99 * 1.2)` en Python et notez ce qui s'affiche. Pourquoi est-ce différent ?
2. Quel prix TTC affiche-t-on, au centime le plus proche ?

3. Un élève coupe après 2 décimales avec `math.floor(prix * 100) / 100`. Quel prix obtient-il ? Est-ce un arrondi au plus près ?

PYTHON

```python
import math
prix = 19.99 * 1.2
print(prix)
```

CORRIGÉ

La méthode. Le calcul sur les réels laisse une petite erreur. On arrondit à l'affichage.

PYTHON

```python
import math
prix = 19.99 * 1.2
print(prix)
print(round(prix, 2))
print(math.floor(prix * 100) / 100)
```

CE QUI S'AFFICHE

```
23.987999999999996
23.99
23.98
```

1. On attend 23,988. Python affiche 23.987999999999996 : c'est la petite erreur des réels.
2. Au plus près : 23,99 €.
3. 23,98 € : c'est un arrondi **par défaut**, pas au plus près. On a coupé au lieu d'arrondir.

★★★ **Niveau 3 — Aller plus loin**

Exercice 3.2.7 — Simuler le hasard ★★★ Sur machine

Un **port** est le numéro qui désigne un service sur un ordinateur, entre 0 et 65 535 : 443 pour le web sécurisé, par exemple.

1. Quelle instruction tire 0 ou 1, comme pile ou face ?
2. Un serveur choisit un port au hasard entre 1024 et 65535. Quelle instruction ?
3. On simule le temps de réponse d'un serveur : un réel au hasard entre 0,5 et 2 secondes. Quelle instruction ?
4. Exécutez le programme ci-dessous trois fois. Les résultats sont-ils les mêmes ?

5. Un élève simule un dé avec `round(random.uniform(1, 6))`. Le réel tiré va de 1 à 6. Quels réels donnent 1 une fois arrondis ? Quels réels donnent 3 ? Le tirage du dé est-il encore uniforme ?

```python
import random
print(random.randint(0, 1), random.randint(1024, 65535))
print(random.uniform(0.5, 2))
```

CORRIGÉ

La méthode. `random.randint(a, b)` tire un entier entre `a` et `b`, les deux bornes comprises. `random.uniform(a, b)` tire un réel entre `a` et `b`.

1. `random.randint(0, 1)`.
2. `random.randint(1024, 65535)`.
3. `random.uniform(0.5, 2)`.
4. Non : chaque exécution tire de nouveaux nombres. Ce hasard vient d'un calcul, il est pseudo-aléatoire, mais on ne peut pas le prévoir.
5. Arrondis au plus près, les réels de 1 à 1,5 donnent 1 : un intervalle de longueur 0,5. Les réels de 2,5 à 3,5 donnent 3 : un intervalle de longueur 1. Le 3 sort donc deux fois plus souvent que le 1. Le tirage n'est plus uniforme : pour un dé, on garde `random.randint(1, 6)`.

> **Attention**
>
> Un tirage uniforme donne la même chance à chaque valeur. Arrondir un réel tiré au hasard casse cette égalité aux deux bouts de l'intervalle.

Exercice 3.2.8 — La distance entre deux points ★★★

Sur un plan, la distance entre le point A de coordonnées (x_1, y_1) et le point B de coordonnées (x_2, y_2) est :

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

1. Écrivez cette formule en ligne, en Python.
2. Calculez la distance entre A(1 ; 2) et B(4 ; 6).
3. Un élève écrit `math.sqrt(x2 - x1 ** 2 + y2 - y1 ** 2)`. Quelle erreur fait-il ?

CORRIGÉ

La méthode. La racine porte sur toute la somme. Chaque différence est élevée au carré : il faut des parenthèses autour.

PYTHON

```python
import math
x1 = 1
y1 = 2
x2 = 4
y2 = 6
d = math.sqrt((x2 - x1) ** 2 + (y2 - y1) ** 2)
print(d)
```

CE QUI S'AFFICHE

```
5.0
```

1. `math.sqrt((x2 - x1) ** 2 + (y2 - y1) ** 2)`
2. $(4 - 1)^2 + (6 - 2)^2 = 9 + 16 = 25$, et la racine de 25 vaut 5.
3. La puissance passe avant la soustraction : il calcule `x1 ** 2` seul, et non la différence au carré.

Exercice 3.2.9 — Arrondir au quart d'heure ★★★ Sur machine

Un technicien facture ses interventions par quarts d'heure entamés. 37 minutes de travail sont facturées 45 minutes.

1. Combien de quarts d'heure entamés dans 37 minutes ?
2. Écrivez l'expression qui donne les minutes facturées pour `m` minutes.
3. Écrivez le programme qui lit `m` et affiche les minutes facturées. Testez avec 37, puis avec 60.

CORRIGÉ

La méthode. On compte les quarts d'heure entamés avec un arrondi par excès, puis on multiplie par 15.

1. 37 / 15 = 2,47, arrondi par excès : 3 quarts d'heure.
2. `math.ceil(m / 15) * 15`

PYTHON

```python
import math
m = int(input("Minutes de travail ? "))
print("Minutes facturées :", math.ceil(m / 15) * 15)
```

CE QUI S'AFFICHE

```
Minutes de travail ? 37
Minutes facturées : 45
```

1. Avec 60, 60 / 15 = 4 tout juste : on facture 60 minutes.

3.3 — Comparer et combiner des conditions

‹ Revoir le cours 3.3

★★★ **Niveau 1 — Appliquer**

Exercice 3.3.1 — Vrai ou faux ? ★★★

Donnez la valeur de chaque comparaison, en Python.

1. `7 > 3`
2. `10 != 10`
3. `"abc" < "abd"`
4. `4 <= 4`

CORRIGÉ

La méthode. Une comparaison vaut toujours `True` ou `False`.

1. `True`. 2. `False` : 10 est égal à 10. 3. `True` : les deux premiers caractères sont égaux, puis `c` vient avant `d`. 4. `True` : « plus petit ou égal » accepte l'égalité.

PYTHON

```python
print(7 > 3, 10 != 10, "abc" < "abd", 4 <= 4)
```

CE QUI S'AFFICHE

```
True False True True
```

Exercice 3.3.2 — Les quatre opérateurs ★☆☆

On a `a ← Vrai` et `b ← Faux`. Donnez la valeur de :

1. `NON a`
2. `a ET b`
3. `a OU b`
4. `a OUX b`

CORRIGÉ

La méthode. On applique la table de vérité du cours, ligne a = Vrai, b = Faux.

1. Faux. 2. Faux : il faut les deux vrais. 3. Vrai : un seul suffit. 4. Vrai : exactement un des deux est vrai.

PYTHON

```python
a = True
b = False
print(not a, a and b, a or b, a != b)
```

CE QUI S'AFFICHE

```
False False True True
```

Exercice 3.3.3 — Majeur ? ★☆☆

On a `majeur ← age ≥ 18`.

1. Que vaut `majeur` pour `age = 17` ? Pour `age = 18` ?
2. Écrivez cette ligne en Python.

CORRIGÉ

La méthode. La comparaison donne un booléen, qu'on range dans la variable.

1. Pour 17 : Faux. Pour 18 : Vrai, car `≥` accepte l'égalité.
2. `majeur = age >= 18`

PYTHON

```python
age = 18
majeur = age >= 18
print(majeur)
```

CE QUI S'AFFICHE

```
True
```

★★☆ **Niveau 2 — S'entraîner**

Exercice 3.3.4 — Traduire une condition ★★☆

Traduisez chaque règle en expression booléenne, en pseudo-code puis en Python. Puis donnez sa valeur pour les données indiquées.

1. « L'âge est entre 18 et 25 ans inclus. » Avec `age = 30`.
2. « Le jour est samedi ou dimanche. » Avec `jour = "dimanche"`.
3. « Le stock n'est pas vide et le produit est encore vendu. » Avec `stock = 0` et `vendu = True`.

CORRIGÉ

La méthode. On découpe la règle en petites comparaisons, puis on les relie avec ET, OU ou NON.

Règle	Pseudo-code	Python	Valeur
1	`age ≥ 18 ET age ≤ 25`	`age >= 18 and age <= 25`	Faux
2	`jour = "samedi" OU jour = "dimanche"`	`jour == "samedi" or jour == "dimanche"`	Vrai
3	`stock ≠ 0 ET vendu`	`stock != 0 and vendu`	Faux

- Règle 1 : 30 ≥ 18 est vrai, mais 30 ≤ 25 est faux. Avec ET, tout est faux.
- Règle 3 : le stock vaut 0, donc `stock ≠ 0` est faux. Avec ET, tout est faux.

PYTHON

```python
age = 30
jour = "dimanche"
stock = 0
vendu = True
print(age >= 18 and age <= 25)
print(jour == "samedi" or jour == "dimanche")
print(stock != 0 and vendu)
```

CE QUI S'AFFICHE

```
False
True
False
```

Attention

`jour == "samedi" or "dimanche"` est une erreur : la seconde partie n'est pas une comparaison, et la condition devient vraie même un lundi. Écrivez `jour ==` des deux côtés.

Exercice 3.3.5 — Le mot de passe accepté ★★☆ Sur machine

Un site accepte un mot de passe si ces trois règles sont vraies :

- il fait au moins 8 caractères ;
- il est différent de l'identifiant ;
- il est différent du mot de passe interdit `motdepasse`.

1. Écrivez la condition `accepte` en pseudo-code, avec `mdp`, `login` et `longueur`.
2. Complétez le programme Python et testez-le avec l'identifiant `ines` et le mot de passe `Sio2026!`.
3. Sans ordinateur, que vaut `accepte` pour les mots de passe `ines` et `motdepasse` ?

PYTHON

```python
login = "ines"
mdp = input("Mot de passe ? ")
accepte = False     # à compléter
print(accepte)
```

CORRIGÉ

La méthode. Une règle donne une comparaison. Les trois doivent être vraies : on les relie avec ET.

1. En pseudo-code :

```
PSEUDO-CODE
accepte ← longueur(mdp) ≥ 8 ET mdp ≠ login ET mdp ≠ "motdepasse"
```

1. En Python :

```python
PYTHON
login = "ines"
mdp = input("Mot de passe ? ")
accepte = len(mdp) >= 8 and mdp != login and mdp != "motdepasse"
print(accepte)
```

```
CE QUI S'AFFICHE
Mot de passe ? Sio2026!
True
```

1. `ines` : Faux, il est trop court et égal à l'identifiant. `motdepasse` : Faux. Il fait bien 10 caractères, mais c'est le mot de passe interdit.

Exercice 3.3.6 — L'année bissextile ★★☆

Une année est bissextile, avec un 29 février, si elle est divisible par 4 mais pas par 100, ou si elle est divisible par 400.

1. Traduisez cette règle en expression booléenne, en pseudo-code, avec la variable `an`. Mettez des parenthèses autour de chaque morceau.
2. Donnez sa valeur pour 2024, 1900, 2000 et 2026.

CORRIGÉ

La méthode. « Divisible par 4 » s'écrit `an mod 4 = 0`. On met des parenthèses autour de la partie avec ET.

1. `(an mod 4 = 0 ET an mod 100 ≠ 0) OU an mod 400 = 0`
2. 2024 : divisible par 4, pas par 100 : Vrai. 1900 : divisible par 100, pas par 400 : Faux. 2000 : divisible par 400 : Vrai. 2026 : pas divisible par 4 : Faux.

PYTHON

```python
an = 1900
print((an % 4 == 0 and an % 100 != 0) or an % 400 == 0)
an = 2000
print((an % 4 == 0 and an % 100 != 0) or an % 400 == 0)
```

CE QUI S'AFFICHE

```
False
True
```

★★★ **Niveau 3 — Aller plus loin**

Exercice 3.3.7 — L'ordre des tests compte ★★★

On a `mot ← "BTSSIO"` et `k ← 6`.

1. Que vaut `k < longueur(mot)` ?
2. Que fait Python avec `k < len(mot) and mot[k] == "S"` ? Avec quel résultat ?
3. Que fait Python avec `mot[k] == "S" and k < len(mot)` ?
4. Quelle règle en tirez-vous ?

CORRIGÉ

La méthode. Un ET s'arrête dès que le premier test est faux. La suite n'est pas lue.

1. 6 < 6 est faux.
2. Le premier test est faux : Python s'arrête et donne `False`. Il ne lit jamais `mot[6]`.

PYTHON

```python
mot = "BTSSIO"
k = 6
print(k < len(mot) and mot[k] == "S")
```

CE QUI S'AFFICHE

```
False
```

1. Python lit d'abord `mot[6]`. Cet indice n'existe pas : le programme s'arrête sur une erreur `IndexError`.

PYTHON

```python
mot = "BTSSIO"
k = 6
print(mot[k] == "S" and k < len(mot))
```

1. On met d'abord le test qui protège, `k < longueur(mot)`, puis celui qui lit la case.

Exercice 3.3.8 — Le contraire d'une condition ★★★

Un club donne accès à la salle si `age ≥ 18 ET inscrit`.

Une règle vous aide : le contraire de `A ET B` s'écrit `NON A OU NON B`. L'accès est refusé dès qu'une des deux conditions manque.

1. Écrivez la condition « accès refusé » avec NON devant toute la condition.
2. Écrivez-la avec la règle, sans NON devant la parenthèse. Le contraire de `age ≥ 18` est `age < 18`.
3. Vérifiez vos deux écritures pour `age = 16` et `inscrit = Vrai`, puis pour `age = 20` et `inscrit = Faux`.

CORRIGÉ

La méthode. L'accès est refusé dès qu'une des deux conditions manque. Le contraire d'un ET devient un OU, et chaque morceau est inversé.

1. `NON (age ≥ 18 ET inscrit)`
2. `age < 18 OU NON inscrit`
3. Pour 16 ans et inscrit : refusé, Vrai pour les deux écritures. Pour 20 ans, pas inscrit : refusé aussi, Vrai pour les deux.

PYTHON

```python
age = 16
inscrit = True
print(not (age >= 18 and inscrit), age < 18 or not inscrit)
age = 20
inscrit = False
print(not (age >= 18 and inscrit), age < 18 or not inscrit)
```

CE QUI S'AFFICHE

```
True True
True True
```

Attention

Le contraire de `age ≥ 18` est `age < 18`, et non `age ≤ 18`. 18 ans donne l'accès.

Exercice 3.3.9 — Le magasin est-il ouvert ? ★★★ Sur machine

Un magasin est ouvert de 9 h à 12 h et de 14 h à 18 h, sauf le dimanche.

À 9 h, il est ouvert. À 12 h, il est fermé. De même, à 14 h il est ouvert, et à 18 h il est fermé.

1. Écrivez la condition `ouvert` avec les variables `h`, une heure entière, et `jour`. Mettez des parenthèses autour de chaque morceau.
2. Complétez le programme et testez-le avec 15 h un mardi.
3. Sans ordinateur : que vaut `ouvert` à 12 h un lundi ? À 10 h un dimanche ?

PYTHON

```
h = int(input("Heure ? "))
jour = input("Jour ? ")
ouvert = False      # à compléter
print(ouvert)
```

CORRIGÉ

La méthode. On écrit chaque créneau avec ET, on relie les deux créneaux avec OU, puis on ajoute la règle du dimanche avec ET. Les parenthèses gardent cet ordre.

1. `((h ≥ 9 ET h < 12) OU (h ≥ 14 ET h < 18)) ET jour ≠ "dimanche"`

PYTHON

```
h = int(input("Heure ? "))
jour = input("Jour ? ")
ouvert = ((h >= 9 and h < 12) or (h >= 14 and h < 18)) and jour != "dimanche"
print(ouvert)
```

CE QUI S'AFFICHE

```
Heure ? 15
Jour ? mardi
True
```

1. À 12 h, le magasin ferme : `h < 12` est faux, Faux. Le dimanche : Faux, quelle que soit l'heure.

3.4 — Les opérateurs bit à bit

‹ Revoir le cours 3.4

★☆☆ **Niveau 1 — Appliquer**

Exercice 3.4.1 — Rang par rang ★☆☆

Calculez à la main, rang par rang :

1. `1100 ET 1010`
2. `1100 OU 1010`
3. `1100 OUX 1010`

CORRIGÉ

La méthode. On écrit les deux nombres l'un sous l'autre. On applique l'opérateur à chaque colonne.

	bit 3	bit 2	bit 1	bit 0
x	1	1	0	0
y	1	0	1	0
ET	1	0	0	0
OU	1	1	1	0
OUX	0	1	1	0

1. `1000`. 2. `1110`. 3. `0110`.

PYTHON

```
print(bin(0b1100 & 0b1010), bin(0b1100 | 0b1010), bin(0b1100 ^ 0b1010))
```

CE QUI S'AFFICHE

```
0b1000 0b1110 0b110
```

Python n'écrit pas le `0` de tête : `0b110` est bien `0110`.

Exercice 3.4.2 — Repérer les bits ★☆☆

On a le nombre binaire `10110`.

1. Que vaut le bit 0 ? Le bit 1 ? Le bit 4 ?
2. Quel masque désigne le bit 3, sur 5 bits ?

CORRIGÉ

La méthode. On numérote les bits de droite à gauche, en partant de 0.

1. `10110` se lit, de droite à gauche : bit 0 = 0, bit 1 = 1, bit 2 = 1, bit 3 = 0, bit 4 = 1.
2. `01000` : un seul 1, au rang 3.

Exercice 3.4.3 — Tester un bit ★☆☆

On a `x = 0b1011`.

1. Quel masque teste le bit 2 ?
2. Que vaut `x & masque` ? Le bit 2 vaut-il 1 ?

CORRIGÉ

La méthode. On fait un ET avec un masque qui a un seul 1, au rang testé. Si le résultat vaut 0, le bit vaut 0.

1. `0b0100`.
2. `1011 ET 0100` donne `0000`. Le résultat vaut 0 : le bit 2 vaut 0.

PYTHON

```
x = 0b1011
print(x & 0b0100)
```

CE QUI S'AFFICHE

```
0
```

★★☆ **Niveau 2 — S'entraîner**

Exercice 3.4.4 — Les droits d'un fichier ★★☆

Les droits d'un fichier tiennent sur 3 bits : le bit 2 pour lire, le bit 1 pour écrire, le bit 0 pour exécuter. Un fichier a les droits `droits = 0b110`.

1. Quels droits a-t-il ?
2. Avec quel masque et quel opérateur teste-t-on le droit d'exécuter ? Quel est le résultat ?
3. Comment ajouter le droit d'exécuter ? Quel résultat ?
4. Comment retirer le droit d'écrire, en partant de `0b110` ? Quel résultat ?
5. Comment inverser le droit de lire, en partant de `0b110` ? Quel résultat ?

CORRIGÉ

La méthode. On place un 1 dans le masque au rang du bit qu'on veut toucher. Puis on choisit l'opérateur selon l'action.

1. `110` : lire oui, écrire oui, exécuter non.
2. Masque `0b001`, opérateur ET : `110 ET 001` donne 0. Le droit d'exécuter est absent.
3. OU avec le masque `001` : `110 OU 001` donne `111`.
4. On inverse d'abord le masque `010`, bit par bit. Le 0 du bit 2 devient 1, le 1 du bit 1 devient 0, le 0 du bit 0 devient 1. On obtient `101`. Puis `110 ET 101` donne `100`.
5. OUX avec le masque `100` : `110 OUX 100` donne `010`.

PYTHON

```
droits = 0b110
print(droits & 0b001)
print(bin(droits | 0b001))
print(bin(droits & ~0b010))
print(bin(droits ^ 0b100))
```

CE QUI S'AFFICHE

```
0
0b111
0b100
0b10
```

Python écrit `0b10` sans le 0 de tête : c'est bien `010`.

Exercice 3.4.5 — Éteindre un bit avec l'inverse ★★☆

On a `x = 0b1111`, et on veut mettre le bit 2 à 0.

1. Quel est le masque du bit 2, sur 4 bits ?
2. Écrivez l'inverse de ce masque, bit par bit.
3. Calculez `x ET inverse`.
4. Pourquoi `x ET masque` ne convient pas ?

CORRIGÉ

La méthode. L'inverse du masque a des 1 partout, sauf au rang à éteindre. Le ET garde alors tous les bits, sauf celui-là.

1. `0100`.
2. On inverse chaque bit : bit 3 : 0 devient 1, bit 2 : 1 devient 0, bit 1 : 0 devient 1, bit 0 : 0 devient 1. L'inverse est `1011`.
3. `1111 ET 1011` donne `1011`.
4. `1111 ET 0100` donne `0100` : on éteint tous les autres bits et on garde seulement le bit 2. C'est le contraire de ce qu'on veut.

PYTHON

```
x = 0b1111
print(bin(x & ~0b0100))
print(bin(x & 0b0100))
```

CE QUI S'AFFICHE

```
0b1011
0b100
```

Exercice 3.4.6 — Le clignotant ★★☆ Sur machine

Quatre voyants sont rangés dans les 4 bits de `voyants`. Un 1 veut dire allumé. Au départ, `voyants = 0b0101`.

1. Quel masque et quel opérateur inversent les 4 voyants à la fois ?
2. Que vaut `voyants` après une inversion ? Après une deuxième ?
3. Exécutez le programme pour vérifier.
4. Ajoutez à la fin une ligne qui éteint seulement le voyant du bit 3, et une ligne qui l'affiche. Que s'affiche-t-il ?

PYTHON

```
voyants = 0b0101
voyants = voyants ^ 0b1111
print(bin(voyants))
voyants = voyants ^ 0b1111
print(bin(voyants))
```

CORRIGÉ

La méthode. OUX avec 1 inverse le bit. Avec le masque `1111`, on inverse les quatre bits d'un coup.

1. Le masque `1111`, avec OUX.
2. `0101` OUX `1111` donne `1010`. On inverse encore : on revient à `0101`.
3. Inverser deux fois ramène au départ : c'est le principe d'un clignotant.
4. Pour éteindre un bit, on fait ET avec l'inverse de son masque : `voyants & ~0b1000`. Le bit 3 vaut déjà 0 dans `0101` : rien ne change.

PYTHON

```
voyants = 0b0101
voyants = voyants ^ 0b1111
print(bin(voyants))
voyants = voyants ^ 0b1111
print(bin(voyants))
voyants = voyants & ~0b1000
print(bin(voyants))
```

CE QUI S'AFFICHE

```
0b1010
0b101
0b101
```

★★★ **Niveau 3 — Aller plus loin**

Exercice 3.4.7 — Le réseau d'une adresse ★★★ Sur machine

Une **adresse IP** est l'adresse d'un ordinateur sur un réseau. Elle s'écrit avec 4 nombres de 0 à 255, comme `192.168.1.198`.

Un réseau découpe ces nombres avec un masque : les bits à 1 du masque désignent le réseau, les bits à 0 désignent l'ordinateur. Deux ordinateurs sont dans le même réseau si leur nombre ET le masque donne le même résultat.

On regarde seulement le dernier nombre, avec le masque `11000000`, soit 192.

1. L'ordinateur A a le nombre 198, soit `11000110`. Calculez `11000110 ET 11000000`, rang par rang, puis en décimal.
2. L'ordinateur B a le nombre 230, soit `11100110`. Calculez `230 ET 192`.
3. L'ordinateur C a le nombre 70, soit `01000110`. Calculez `70 ET 192`.
4. Quels ordinateurs sont dans le même réseau que A ? Vérifiez vos calculs en Python.

PYTHON

```python
print(198 & 192, 230 & 192, 70 & 192)
```

CORRIGÉ

La méthode. Le ET avec le masque garde les bits où le masque vaut 1, et met les autres à 0. On compare ensuite les résultats.

1. `11000110 ET 11000000` donne `11000000`, soit 128 + 64 = 192.
2. `11100110 ET 11000000` donne `11000000`, soit 192.
3. `01000110 ET 11000000` donne `01000000`, soit 64.
4. A et B donnent 192 : ils sont dans le même réseau. C donne 64 : il est dans un autre réseau.

PYTHON

```python
print(198 & 192, 230 & 192, 70 & 192)
```

CE QUI S'AFFICHE

```
192 192 64
```

Le masque garde les 2 bits de gauche, qui désignent le réseau. Il efface les 6 bits de droite, qui changent d'un ordinateur à l'autre.

Exercice 3.4.8 — Tester deux bits à la fois ★★★

Un registre d'alarmes tient sur 8 bits. Le bit 1 signale « porte ouverte », le bit 2 signale « mouvement ».

1. Quel masque désigne les bits 1 et 2 ensemble ?
2. Pour `x = 0b10110110`, que vaut `x ET masque` ? Et pour `x = 0b10110010` ?
3. Écrivez l'expression booléenne « les deux alarmes sont allumées », avec `x` et le masque.
4. Écrivez l'expression « au moins une des deux alarmes est allumée ».
5. Donnez la valeur des deux expressions pour les deux valeurs de `x`.

CORRIGÉ

La méthode. Le ET avec le masque garde seulement les bits 1 et 2. Les deux sont à 1 si le résultat est égal au masque. Au moins un est à 1 si le résultat n'est pas nul.

1. `00000110`, soit `0b110` : un 1 au bit 1 et au bit 2.
2. `10110110 ET 00000110` donne `00000110`. `10110010 ET 00000110` donne `00000010`.
3. `(x ET 0b110) = 0b110`
4. `(x ET 0b110) ≠ 0`
5. Pour `10110110` : les deux sont vraies. Pour `10110010` : « les deux » est faux, car seul le bit 1 est à 1. « Au moins une » est vrai.

PYTHON

```
x = 0b10110110
print((x & 0b110) == 0b110, (x & 0b110) != 0)
x = 0b10110010
print((x & 0b110) == 0b110, (x & 0b110) != 0)
```

CE QUI S'AFFICHE

```
True True
False True
```

Attention

En Python, les parenthèses autour de `x & 0b110` sont obligatoires. Sans elles, Python compare d'abord `0b110 == 0b110`, puis fait le ET : le résultat est faux.

Exercice 3.4.9 — Les options d'un objet connecté ★★★ Sur machine

Une montre range ses options sur 4 bits : bit 0 = wifi, bit 1 = bluetooth, bit 2 = GPS, bit 3 = NFC. NFC veut dire *Near Field Communication* : c'est le paiement sans contact. Au départ, `options = 0`.

Répondez d'abord sur papier.

1. On allume le wifi et le GPS d'un seul coup. Quel masque, quel opérateur ?
2. On teste le GPS. Quel calcul ?
3. On éteint le wifi. Quel calcul ?
4. On inverse le NFC. Quel calcul ?
5. Faites la trace de `options` en binaire.

6. Complétez le programme : remplacez chaque `0` marqué « à compléter » par le bon masque. Exécutez-le et comparez avec votre trace.

PYTHON

```python
options = 0
options = options | 0      # masque à compléter
print(bin(options))
print(options & 0)         # masque à compléter
options = options & ~0     # masque à compléter
print(bin(options))
options = options ^ 0      # masque à compléter
print(bin(options))
```

CORRIGÉ

La méthode. Un masque peut avoir plusieurs 1 : il touche alors plusieurs bits à la fois.

1. OU avec `0101`, qui a un 1 au bit 0 et au bit 2.
2. `options ET 0100`. Le résultat n'est pas nul : le GPS est allumé.
3. ET avec l'inverse de `0001`, soit `1110`.
4. OUX avec `1000`.

Étape	options
départ	0000
wifi et GPS	0101
wifi éteint	0100
NFC inversé	1100

PYTHON

```python
options = 0
options = options | 0b0101
print(bin(options))
print(options & 0b0100)
options = options & ~0b0001
print(bin(options))
options = options ^ 0b1000
print(bin(options))
```

CE QUI S'AFFICHE

```
0b101
4
0b100
0b1100
```

Le test du GPS affiche 4 : un résultat différent de `0` veut dire que le bit est à 1.

3.5 — Les opérateurs de chaînes

‹ Revoir le cours 3.5

★☆☆ **Niveau 1 — Appliquer**

Exercice 3.5.1 — Coller des chaînes ★☆☆

Que vaut chaque expression ?

1. `"Ines" + " " + "Martin"`
2. `"BTS" + "SIO"`
3. `"3" + "4"`

CORRIGÉ

La méthode. Le `+` entre deux chaînes les colle dans l'ordre, sans ajouter d'espace.

PYTHON

```python
print("Ines" + " " + "Martin")
print("BTS" + "SIO")
print("3" + "4")
```

CE QUI S'AFFICHE

```
Ines Martin
BTSSIO
34
```

Au 3, ce sont deux chaînes : on obtient `"34"`, pas 7.

Exercice 3.5.2 — Extraire un morceau ★☆☆

On a `s = "BTS SIO 2026"`. Que vaut :

1. `s[:3]`
2. `s[4:7]`
3. `s[-4:]`

CORRIGÉ

La méthode. Dans `s[i:j]`, l'indice `i` est pris, l'indice `j` est exclu. Un indice négatif compte depuis la fin.

PYTHON

```
s = "BTS SIO 2026"
print(s[:3])
print(s[4:7])
print(s[-4:])
```

CE QUI S'AFFICHE

```
BTS
SIO
2026
```

Au 2, on prend les indices 4, 5 et 6 : S, I, O.

Exercice 3.5.3 — Chercher un motif ★☆☆

Que vaut :

1. `"bonjour".find("j")`
2. `"bonjour".find("z")`
3. `"a@b.fr".find("@")`

CORRIGÉ

La méthode. `find` donne l'indice de la première apparition du motif, ou -1 s'il est absent.

1. 3 : b = 0, o = 1, n = 2, j = 3. 2. -1 : il n'y a pas de z. 3. 1.

PYTHON

```
print("bonjour".find("j"), "bonjour".find("z"), "a@b.fr".find("@"))
```

CE QUI S'AFFICHE

```
3 -1 1
```

★★☆ **Niveau 2 — S'entraîner**

Exercice 3.5.4 — Découper une adresse mail ★★☆ Sur machine

On a `mail = "ines.martin@lycee-sio.fr"`.

1. Quelle instruction donne la position de `@` ? Quelle est sa valeur ?
2. Avec cette position `p`, extrayez l'identifiant, avant `@`.
3. Extrayez le domaine, après `@`.
4. Extrayez les 2 dernières lettres, l'extension.
5. Construisez le message `Domaine : lycee-sio.fr` par concaténation.

PYTHON

```python
mail = "ines.martin@lycee-sio.fr"
p = 0       # à compléter : la position de @
print(p)
# à compléter
```

CORRIGÉ

La méthode. On cherche d'abord le `@`. Tout ce qui est avant est l'identifiant, tout ce qui est après est le domaine.

PYTHON

```python
mail = "ines.martin@lycee-sio.fr"
p = mail.find("@")
print(p)
identifiant = mail[:p]
domaine = mail[p + 1:]
print(identifiant)
print(domaine)
print(mail[-2:])
print("Domaine : " + domaine)
```

CE QUI S'AFFICHE

```
11
ines.martin
lycee-sio.fr
fr
Domaine : lycee-sio.fr
```

- `mail[:p]` prend les indices 0 à 10 : le `@`, à l'indice 11, est exclu.
- `mail[p + 1:]` commence juste après le `@` et va jusqu'à la fin.

Attention

`mail[p:]` donnerait `@lycee-sio.fr`, avec le `@`. Il faut partir de `p + 1`.

Exercice 3.5.5 — Le nom et l'extension d'un fichier ★★☆ Sur machine

On lit un nom de fichier, comme `rapport.pdf`. Il contient un seul point.

1. Comment trouver la position `p` du point ?
2. Comment extraire le nom, avant le point ? L'extension, après ?
3. Complétez le programme et testez-le avec `rapport.pdf`.

PYTHON

```python
fichier = input("Nom du fichier ? ")
p = 0               # à compléter : la position du point
nom = ""            # à compléter
extension = ""      # à compléter
print(nom, extension)
```

CORRIGÉ

La méthode. Même idée que pour une adresse mail : le point sépare deux morceaux.

1. `p = fichier.find(".")`.
2. Le nom : `fichier[:p]`. L'extension : `fichier[p + 1:]`.

PYTHON

```python
fichier = input("Nom du fichier ? ")
p = fichier.find(".")
nom = fichier[:p]
extension = fichier[p + 1:]
print(nom, extension)
```

CE QUI S'AFFICHE

```
Nom du fichier ? rapport.pdf
rapport pdf
```

Exercice 3.5.6 — Construire un message ★★☆

On a `prenom = "Tom"` et `total = 42.456`.

1. Construisez par concaténation le message `Tom doit 42.46 €`.
2. Pourquoi faut-il `str` ?
3. Pourquoi faut-il `round` ?

CORRIGÉ

La méthode. On ne colle que des chaînes. On arrondit le nombre, puis on le change en chaîne.

PYTHON

```python
prenom = "Tom"
total = 42.456
print(prenom + " doit " + str(round(total, 2)) + " €")
```

CE QUI S'AFFICHE

```
Tom doit 42.46 €
```

1. `+` refuse de coller une chaîne et un nombre : `str` change le nombre en texte.
2. Sans `round`, on afficherait toutes les décimales : 42.456.

★★★ Niveau 3 — Aller plus loin

Exercice 3.5.7 — Retourner une date ★★★ Sur machine

Une date arrive au format `18/10/2026`, toujours avec 2 chiffres pour le jour et le mois. Un logiciel attend le format `2026-10-18`.

1. Quels indices donnent le jour ? Le mois ? L'année ?
2. Construisez la date au nouveau format par concaténation.
3. Le mois doit aussi servir à un calcul. Comment obtenir le mois sous forme d'entier ?
4. Complétez le programme et testez-le avec `18/10/2026`.

PYTHON

```python
date = input("Date (jj/mm/aaaa) ? ")
nouvelle = ""       # à compléter
mois = 0            # à compléter
print(nouvelle)
print("Mois suivant :", mois + 1)
```

CORRIGÉ

La méthode. Le format est fixe : chaque morceau est toujours au même endroit. On extrait avec des tranches.

1. Jour : indices 0 et 1, `date[:2]`. Mois : indices 3 et 4, `date[3:5]`. Année : indices 6 à 9, `date[6:]`.
2. `date[6:] + "-" + date[3:5] + "-" + date[:2]`
3. `int(date[3:5])` : on change le texte `"10"` en entier 10.

PYTHON

```python
date = input("Date (jj/mm/aaaa) ? ")
nouvelle = date[6:] + "-" + date[3:5] + "-" + date[:2]
mois = int(date[3:5])
print(nouvelle)
print("Mois suivant :", mois + 1)
```

CE QUI S'AFFICHE

```
Date (jj/mm/aaaa) ? 18/10/2026
2026-10-18
Mois suivant : 11
```

Exercice 3.5.8 — Une adresse du réseau local ? ★★★

Une adresse IP commence par `192.168.` quand elle appartient à un réseau local. On a `ip = "192.168.1.25"`.

1. Écrivez un test avec une tranche de `ip`.
2. Un élève écrit `ip.find("192.168.") != -1`. Que vaut ce test pour `"10.192.168.5"` ? Est-ce juste ?
3. Écrivez un test avec `find` qui vérifie que l'adresse **commence** par `192.168.`.

CORRIGÉ

La méthode. « Commence par » veut dire : le motif est à l'indice 0. Être présent ailleurs ne suffit pas.

1. `ip[:8] == "192.168."` : `192.168.` fait 8 caractères.
2. Le motif est présent, à l'indice 3 : le test vaut Vrai. C'est faux : cette adresse ne commence pas par `192.168.`.
3. `ip.find("192.168.") == 0`.

PYTHON

```
ip = "192.168.1.25"
autre = "10.192.168.5"
print(ip[:8] == "192.168.", ip.find("192.168.") == 0)
print(autre.find("192.168.") != -1, autre.find("192.168.") == 0)
```

CE QUI S'AFFICHE

```
True True
True False
```

Exercice 3.5.9 — Fabriquer un identifiant ★★★ Sur machine

Un lycée fabrique l'identifiant d'un élève ainsi : la première lettre du prénom, puis les 7 premières lettres du nom, au plus. Il ajoute `@lycee.fr` pour l'adresse mail.

1. Quelle instruction donne la première lettre du prénom ?
2. Que donne `nom[:7]` pour `martin` ? Pour `fontaine-dupont` ?
3. Écrivez le programme qui lit le prénom et le nom, puis affiche l'adresse mail. Testez avec `ines` et `fontaine-dupont`.
4. Ajoutez la variable `court`, qui vaut Vrai si l'identifiant fait au plus 6 caractères.

PYTHON

```
prenom = input("Prénom ? ")
nom = input("Nom ? ")
identifiant = ""    # à compléter
print(identifiant + "@lycee.fr")
```

CORRIGÉ

La méthode. On extrait les morceaux avec des tranches, puis on les colle. Une tranche qui dépasse la fin s'arrête simplement au dernier caractère.

1. `prenom[0]`.

2. `martin` a 6 lettres : `nom[:7]` donne `martin` en entier. `fontaine-dupont` donne `fontain`.

PYTHON

```python
prenom = input("Prénom ? ")
nom = input("Nom ? ")
identifiant = prenom[0] + nom[:7]
court = len(identifiant) <= 6
print(identifiant + "@lycee.fr")
print(court)
```

CE QUI S'AFFICHE

```
Prénom ? ines
Nom ? fontaine-dupont
ifontain@lycee.fr
False
```

1. `ifontain` a 8 caractères : `court` vaut Faux.

4.1 — Faire un choix : Si... Alors... Sinon

‹ Revoir le cours 4.1

★☆☆ **Niveau 1 — Appliquer**

Exercice 4.1.1 — Pair ou impair ★☆☆ Sur machine

Un programme lit un entier et dit s'il est pair ou impair.

1. Complétez l'algorithme.

PSEUDO-CODE

```
Lire n
Si n mod 2 = . . . Alors
    Écrire "pair"
Sinon
    Écrire . . .
Fin de Si
```

1. Qu'affiche-t-il pour 14 ? Pour 7 ?
2. Complétez le programme Python, puis testez-le avec 7.

PYTHON

```python
n = int(input("Un entier ? "))
if n % 2 == 0:
    print("pair")
```

CORRIGÉ

La méthode. Un nombre est pair quand le reste de sa division par 2 vaut 0.

1. L'algorithme complet :

PSEUDO-CODE

```
Lire n
Si n mod 2 = 0 Alors
    Écrire "pair"
Sinon
    Écrire "impair"
Fin de Si
```

1. 14 mod 2 = 0 : « pair ». 7 mod 2 = 1 : « impair ».
2. Il manque le `else` :

PYTHON

```python
n = int(input("Un entier ? "))
if n % 2 == 0:
    print("pair")
else:
    print("impair")
```

CE QUI S'AFFICHE

```
Un entier ? 7
impair
```

Exercice 4.1.2 — Suivre un Si ★☆☆

On considère cet algorithme :

PSEUDO-CODE

```
Si x > 10 Alors
    y ← x - 10
Sinon
    y ← x + 10
Fin de Si
Écrire y
```

1. Que vaut `y` si `x` vaut 12 ?
2. Que vaut `y` si `x` vaut 4 ?
3. Que vaut `y` si `x` vaut 10 ? Attention au signe `>`.

CORRIGÉ

La méthode. On évalue la condition, puis on exécute un seul des deux blocs.

1. 12 > 10 est Vrai : `y ← 12 - 10`, donc `y` vaut 2.
2. 4 > 10 est Faux : `y ← 4 + 10`, donc `y` vaut 14.
3. 10 > 10 est Faux, car 10 n'est pas strictement plus grand que 10 : `y` vaut 20.

PYTHON

```python
x = 10
if x > 10:
    y = x - 10
else:
    y = x + 10
print(y)
```

CE QUI S'AFFICHE

```
20
```

Exercice 4.1.3 — Le digicode ★☆☆

La porte de la salle serveur s'ouvre avec le code `"4721"`. Le programme lit un code tapé par l'utilisateur.

1. Écrivez l'algorithme en pseudo-code : s'il tape le bon code, on affiche « Porte ouverte », sinon « Code refusé ».
2. Pourquoi le code est-il rangé dans une chaîne, et pas dans un entier ?

CORRIGÉ

La méthode. Une seule condition, deux cas : un `Si … Alors … Sinon`.

1. L'algorithme :

```
PSEUDO-CODE
Lire code
Si code = "4721" Alors
    Écrire "Porte ouverte"
Sinon
    Écrire "Code refusé"
Fin de Si
```

1. On ne calcule jamais sur un code. Et un code comme `"0472"` perdrait son `0` du début dans un entier.

```
PYTHON
code = input("Code ? ")
if code == "4721":
    print("Porte ouverte")
else:
    print("Code refusé")
```

```
CE QUI S'AFFICHE
Code ? 4712
Code refusé
```

★★☆ **Niveau 2 — S'entraîner**

Exercice 4.1.4 — La mention ★★☆

On veut afficher la mention d'un étudiant selon sa moyenne :

- 16 ou plus : « Très bien » ;
- 14 ou plus : « Bien » ;
- 12 ou plus : « Assez bien » ;
- 10 ou plus : « Passable » ;
- sinon : « Non admis ».

1. Écrivez l'algorithme en pseudo-code avec `Si … Sinon si … Sinon`.
2. Qu'affiche-t-il pour une moyenne de 13,5 ? Et de 16 ?
3. Un collègue a écrit les tests dans l'ordre inverse, en commençant par `moyenne ≥ 10`. Qu'affiche sa version pour 16 ? Pourquoi ?

CORRIGÉ

La méthode. On teste le seuil le plus haut d'abord. Le premier test vrai gagne.

1. L'algorithme :

PSEUDO-CODE

```
Lire moyenne
Si moyenne ≥ 16 Alors
    Écrire "Très bien"
Sinon si moyenne ≥ 14 Alors
    Écrire "Bien"
Sinon si moyenne ≥ 12 Alors
    Écrire "Assez bien"
Sinon si moyenne ≥ 10 Alors
    Écrire "Passable"
Sinon
    Écrire "Non admis"
Fin de Si
```

1. Pour 13,5 : les deux premiers tests sont faux, le troisième est vrai. On affiche « Assez bien ». Pour 16 : « Très bien ».
2. Sa version affiche « Passable » pour 16. Le test `16 ≥ 10` est vrai, et c'est le premier : les autres ne sont jamais lus.

PYTHON

```python
moyenne = float(input("Moyenne ? "))
if moyenne >= 16:
    print("Très bien")
elif moyenne >= 14:
    print("Bien")
elif moyenne >= 12:
    print("Assez bien")
elif moyenne >= 10:
    print("Passable")
else:
    print("Non admis")
```

CE QUI S'AFFICHE

```
Moyenne ? 13.5
Assez bien
```

Exercice 4.1.5 — Le prix du billet ★★☆ Sur machine

Un cinéma fixe le prix du billet selon l'âge :

- moins de 12 ans : 6 € ;
- 65 ans ou plus : 7 € ;
- les autres : 10 €.

1. Écrivez l'algorithme en pseudo-code. Il lit l'âge et affiche le prix.
2. Écrivez le programme en Python.
3. Préparez un jeu d'essai : une valeur par branche et les valeurs limites 11, 12, 64, 65, avec le prix attendu. Puis testez.

PYTHON

```python
age = int(input("Âge ? "))
# à compléter
```

CORRIGÉ

La méthode. Trois cas : un `Si`, un `Sinon si`, un `Sinon`. Le cas « les autres » va dans le `Sinon`.

1. L'algorithme :

PSEUDO-CODE

```
Lire age
Si age < 12 Alors
    prix ← 6
Sinon si age ≥ 65 Alors
    prix ← 7
Sinon
    prix ← 10
Fin de Si
Écrire "Prix :", prix, "€"
```

1. En Python, avec 70 ans :

PYTHON

```python
age = int(input("Âge ? "))
if age < 12:
    prix = 6
elif age >= 65:
    prix = 7
else:
    prix = 10
print("Prix :", prix, "€")
```

CE QUI S'AFFICHE

```
Âge ? 70
Prix : 7 €
```

Avec 8 ans, on obtient 6 €. Avec 30 ans, 10 €.

1. Le jeu d'essai :

Âge	Branche	Prix attendu
8	moins de 12 ans	6 €
11	limite : dernier âge à 6 €	6 €
12	limite : premier âge à 10 €	10 €
30	les autres	10 €
64	limite : dernier âge à 10 €	10 €
65	limite : premier âge à 7 €	7 €
70	65 ans ou plus	7 €

Les valeurs limites vérifient les signes `<` et `≥` : avec `age ≤ 12`, l'âge 12 donnerait 6 € au lieu de 10 €.

Exercice 4.1.6 — Le plus grand de trois ★★☆

On a trois nombres `a`, `b` et `c`. On veut ranger le plus grand dans `grand`.

1. Complétez l'algorithme.

```
PSEUDO-CODE
Si a ≥ b ET a ≥ c Alors
    grand ← a
Sinon si . . . Alors
    grand ← b
Sinon
    grand ← . . .
Fin de Si
```

1. Que vaut `grand` pour `a = 4`, `b = 9`, `c = 2` ? Quel bloc est exécuté ?
2. Et pour `a = 5`, `b = 5`, `c = 1` ?

CORRIGÉ

La méthode. `a` est le plus grand s'il dépasse les deux autres. Sinon, on teste `b` de la même façon. Sinon, c'est `c`.

1. L'algorithme complet :

```
PSEUDO-CODE
Si a ≥ b ET a ≥ c Alors
    grand ← a
Sinon si b ≥ c Alors
    grand ← b
Sinon
    grand ← c
Fin de Si
```

Au `Sinon si`, on sait déjà que `a` n'est pas le plus grand. Il suffit donc de comparer `b` et `c`.

1. `4 ≥ 9` est faux : le premier test est faux. `9 ≥ 2` est vrai : `grand` vaut 9.
2. `5 ≥ 5 ET 5 ≥ 1` est vrai : `grand` vaut 5. Avec `≥`, une égalité ne pose pas de problème.

PYTHON

```python
a = 4
b = 9
c = 2
if a >= b and a >= c:
    grand = a
elif b >= c:
    grand = b
else:
    grand = c
print(grand)
```

CE QUI S'AFFICHE

```
9
```

★★★ **Niveau 3 — Aller plus loin**

Exercice 4.1.7 — L'année bissextile ★★★

Une année est bissextile si elle est divisible par 4 mais pas par 100, ou si elle est divisible par 400.

1. Traduisez cette règle en une condition, avec `mod`, `ET`, `OU`.
2. Écrivez l'algorithme qui lit une année et affiche « bissextile » ou « non bissextile ».
3. Donnez le résultat pour 2024, 1900 et 2000, dans un tableau.

CORRIGÉ

La méthode. On découpe la phrase : « divisible par 4 » devient `annee mod 4 = 0`, « pas par 100 » devient `annee mod 100 ≠ 0`.

1. La condition :

PSEUDO-CODE

```
(annee mod 4 = 0 ET annee mod 100 ≠ 0) OU annee mod 400 = 0
```

1. L'algorithme :

PSEUDO-CODE

```
Lire annee
Si (annee mod 4 = 0 ET annee mod 100 ≠ 0) OU annee mod 400 = 0 Alors
    Écrire "bissextile"
Sinon
    Écrire "non bissextile"
Fin de Si
```

1. Les trois années :

Année	mod 4 = 0	mod 100 ≠ 0	mod 400 = 0	Résultat
2024	Vrai	Vrai	Faux	bissextile
1900	Vrai	Faux	Faux	non bissextile
2000	Vrai	Faux	Vrai	bissextile

PYTHON

```python
annee = int(input("Année ? "))
if (annee % 4 == 0 and annee % 100 != 0) or annee % 400 == 0:
    print("bissextile")
else:
    print("non bissextile")
```

CE QUI S'AFFICHE

```
Année ? 1900
non bissextile
```

Attention

Les parenthèses comptent. Elles disent que le ET se calcule d'abord, puis le OU.

Exercice 4.1.8 — Les frais de livraison ★★★ Sur machine

Un site de vente calcule les frais de livraison :

- commande de moins de 50 € : 4,90 € de frais ;
- de 50 € à moins de 100 € : 2,50 € ;
- 100 € ou plus : livraison gratuite ;
- un client « premium » a toujours la livraison gratuite.

Le programme lit le montant, puis la réponse à « Client premium ? (o/n) ».

1. Écrivez l'algorithme en pseudo-code. Quel cas faut-il tester en premier ?
2. Écrivez le programme Python. Testez-le avec 60 € pour un client premium, puis 60 € pour un client normal.

PYTHON

```python
montant = float(input("Montant ? "))
premium = input("Client premium ? (o/n) ")
# à compléter
```

CORRIGÉ

La méthode. Le client premium passe avant tout : on le teste en premier. Ensuite, on teste les montants du plus petit au plus grand.

1. L'algorithme :

PSEUDO-CODE

```
Lire montant
Lire premium
Si premium = "o" OU montant ≥ 100 Alors
    frais ← 0
Sinon si montant < 50 Alors
    frais ← 4.90
Sinon
    frais ← 2.50
Fin de Si
Écrire "Frais :", frais, "€"
```

Au `Sinon`, on sait que le montant est entre 50 et 100 : les deux autres cas ont été écartés.

1. En Python :

PYTHON

```python
montant = float(input("Montant ? "))
premium = input("Client premium ? (o/n) ")
if premium == "o" or montant >= 100:
    frais = 0
elif montant < 50:
    frais = 4.90
else:
    frais = 2.50
print("Frais :", frais, "€")
```

CE QUI S'AFFICHE

```
Montant ? 60
Client premium ? (o/n) o
Frais : 0 €
```

Pour un client normal, on tape `n` : le programme affiche `Frais : 2.5 €`. Python n'écrit pas le zéro de fin : 2,50 € s'affiche `2.5`, et 4,90 € s'affiche `4.9`.

Exercice 4.1.9 — Le type de triangle ★★★

Un programme lit les trois côtés `a`, `b`, `c` d'un triangle. Il affiche :

- « équilatéral » si les trois côtés sont égaux ;
- « isocèle » si exactement deux côtés sont égaux ;
- « quelconque » sinon.

Voici la version d'un stagiaire :

PSEUDO-CODE

```
Si a = b OU b = c OU a = c Alors
    Écrire "isocèle"
Sinon si a = b ET b = c Alors
    Écrire "équilatéral"
Sinon
    Écrire "quelconque"
Fin de Si
```

1. Qu'affiche ce programme pour `a = 5`, `b = 5`, `c = 5` ? Est-ce juste ?
2. Expliquez l'erreur.
3. Corrigez l'algorithme.

CORRIGÉ

La méthode. On teste chaque cas à la main. Le premier test vrai gagne : l'ordre des tests compte.

1. Pour 5, 5, 5 : `a = b` est vrai, donc le premier test est vrai. Le programme affiche « isocèle ». C'est faux : le triangle est équilatéral.
2. Un triangle équilatéral a aussi deux côtés égaux : il passe le premier test. Le cas « équilatéral » n'est jamais atteint.
3. On met le cas le plus précis en premier :

PSEUDO-CODE

```
Si a = b ET b = c Alors
    Écrire "équilatéral"
Sinon si a = b OU b = c OU a = c Alors
    Écrire "isocèle"
Sinon
    Écrire "quelconque"
Fin de Si
```

PYTHON

```
a = 5
b = 5
c = 5
if a == b and b == c:
    print("équilatéral")
elif a == b or b == c or a == c:
    print("isocèle")
else:
    print("quelconque")
```

CE QUI S'AFFICHE

```
équilatéral
```

4.2 — Répéter un nombre connu de fois : la boucle Pour

‹ Revoir le cours 4.2

★☆☆ **Niveau 1 — Appliquer**

Exercice 4.2.1 — La table de multiplication ★☆☆ Sur machine

On veut afficher la table de 7, de 7 × 1 à 7 × 10, une ligne par calcul : `7 x 1 = 7`.

1. Complétez l'algorithme.

PSEUDO-CODE

```
Pour i allant de . . . à . . . Faire
    Écrire 7, "x", i, "=", . . .
Fin de Pour
```

1. Corrigez le programme Python, puis exécutez-le.

PYTHON

```python
for i in range(1, 2):    # à corriger
    print(7, "x", i, "=", 0)    # à compléter
```

CORRIGÉ

La méthode. On connaît le nombre de tours, 10 : c'est une boucle Pour.

1. `Pour i allant de 1 à 10 Faire`, et on affiche `7 * i`.
2. En Python, la borne de droite est exclue : on écrit `range(1, 11)`.

PYTHON

```python
for i in range(1, 11):
    print(7, "x", i, "=", 7 * i)
```

CE QUI S'AFFICHE

```
7 x 1 = 7
7 x 2 = 14
7 x 3 = 21
7 x 4 = 28
7 x 5 = 35
7 x 6 = 42
7 x 7 = 49
7 x 8 = 56
7 x 9 = 63
7 x 10 = 70
```

> **Attention**
>
> Avec `range(1, 10)`, la ligne `7 x 10` manque : la borne 10 est exclue.

Exercice 4.2.2 — Trace d'une boucle Pour ★☆☆

PSEUDO-CODE

```
s ← 0
Pour i allant de 1 à 4 Faire
    s ← s + 2 * i
Fin de Pour
Écrire s
```

1. Faites la trace : une ligne par tour, avec `i` et `s`.
2. Qu'affiche l'algorithme ?

CORRIGÉ

La méthode. Une ligne de départ, puis une ligne par tour. À chaque tour, on lit `s` dans la ligne du dessus.

1. La trace :

Étape	i	s
départ	–	0
tour 1	1	2
tour 2	2	6
tour 3	3	12
tour 4	4	20

1. L'algorithme affiche 20.

PYTHON

```python
s = 0
for i in range(1, 5):
    s = s + 2 * i
    print("i =", i, "s =", s)
```

CE QUI S'AFFICHE

```
i = 1 s = 2
i = 2 s = 6
i = 3 s = 12
i = 4 s = 20
```

Exercice 4.2.3 — Combien de tours ? ★☆☆

Pour chaque boucle, donnez le nombre de tours et les valeurs prises par le compteur.

1. `Pour i allant de 3 à 8 Faire`
2. `for i in range(2, 10):`
3. `for i in range(5):`
4. `Pour k allant de 0 à longueur("BTSSIO") - 1 Faire`

CORRIGÉ

La méthode. En pseudo-code, les deux bornes sont prises. En Python, `range` exclut la borne de droite, et part de `0` si on ne donne qu'un nombre.

1. 6 tours : 3, 4, 5, 6, 7, 8.
2. 8 tours : 2, 3, 4, 5, 6, 7, 8, 9.
3. 5 tours : `0`, 1, 2, 3, 4.
4. 6 tours : `0`, 1, 2, 3, 4, 5. Ce sont les indices de la chaîne.

PYTHON

```python
for i in range(5):
    print(i)
```

CE QUI S'AFFICHE

```
0
1
2
3
4
```

★★☆ **Niveau 2 — S'entraîner**

Exercice 4.2.4 — Compter une lettre ★★☆

On veut compter combien de fois la lettre `e` apparaît dans le mot `"developpement"`.

1. Complétez l'algorithme.

PSEUDO-CODE

```
mot ← "developpement"
compt ← . . .
Pour i allant de 0 à . . . Faire
    Si mot[i] = "e" Alors
        . . .
    Fin de Si
Fin de Pour
Écrire compt
```

1. Que vaut `compt` à la fin ? Donnez les indices des `e`.

CORRIGÉ

La méthode. C'est un compteur : il part de 0 avant la boucle, et on ajoute 1 à chaque `e` trouvé. On lit tous les indices du mot.

1. L'algorithme complet :

```pseudo
mot ← "developpement"
compt ← 0
Pour i allant de 0 à longueur(mot) - 1 Faire
    Si mot[i] = "e" Alors
        compt ← compt + 1
    Fin de Si
Fin de Pour
Écrire compt
```

1. Les `e` sont aux indices 1, 3, 8 et 10 : `compt` vaut 4.

```python
mot = "developpement"
compt = 0
for i in range(len(mot)):
    if mot[i] == "e":
        compt = compt + 1
print(compt)
```

```
4
```

Exercice 4.2.5 — Compter les voyelles ★★☆ Sur machine

On veut compter les voyelles d'un mot lu au clavier. Les voyelles sont a, e, i, o, u, y, en minuscules ou en majuscules.

L'idée : on range toutes les voyelles dans une chaîne `voyelles`. Pour chaque caractère du mot, on le cherche dans cette chaîne. `position(voyelles, mot[i])` renvoie -1 si le caractère n'est pas dans `voyelles` : ce n'est pas une voyelle.

1. Complétez les pointillés.

PSEUDO-CODE

```
Lire mot
voyelles ← "aeiouyAEIOUY"
cpt ← . . .
Pour i allant de 0 à . . . Faire
    Si position(voyelles, mot[i]) ≠ -1 Alors
        . . .
    Fin de Si
Fin de Pour
Écrire cpt
```

1. Que donne l'algorithme pour `Algorithmique` ? Listez les voyelles trouvées.
2. Écrivez le programme en Python. Testez-le avec `Algorithmique`, `Python`, `SIO` et `Crypt`.

CORRIGÉ

La méthode. `position(voyelles, mot[i]) ≠ -1` veut dire : le caractère est dans la liste des voyelles. On compte ces caractères.

1. L'algorithme complet :

PSEUDO-CODE

```
Lire mot
voyelles ← "aeiouyAEIOUY"
cpt ← 0
Pour i allant de 0 à longueur(mot) - 1 Faire
    Si position(voyelles, mot[i]) ≠ -1 Alors
        cpt ← cpt + 1
    Fin de Si
Fin de Pour
Écrire cpt
```

1. Dans `Algorithmique`, on trouve A, o, i, i, u, e : 6 voyelles.
2. En Python :

PYTHON

```python
mot = input("Mot ? ")
voyelles = "aeiouyAEIOUY"
cpt = 0
for i in range(len(mot)):
    if voyelles.find(mot[i]) != -1:
        cpt = cpt + 1
print(cpt)
```

CE QUI S'AFFICHE

```
Mot ? Algorithmique
6
```

Mot	Voyelles	Résultat
Algorithmique	A, o, i, i, u, e	6
Python	y, o	2
SIO	I, O	2
Crypt	y	1

Exercice 4.2.6 — Le mot à l'envers ★★☆

On veut écrire un mot à l'envers : `"SIO"` devient `"OIS"`.

Idée : on part d'une chaîne vide. À chaque tour, on colle la lettre lue **devant** ce qu'on a déjà.

1. Complétez l'algorithme.

PSEUDO-CODE

```
mot ← "SIO"
envers ← ""
Pour i allant de 0 à longueur(mot) - 1 Faire
    envers ← . . .
Fin de Pour
Écrire envers
```

1. Faites la trace pour `"SIO"`.
2. Que donne l'algorithme pour `"radar"` ? Qu'a ce mot de particulier ?

CORRIGÉ

La méthode. `mot[i] + envers` colle la nouvelle lettre devant. La dernière lettre lue se retrouve donc en premier.

1. `envers ← mot[i] + envers`
2. La trace :

Étape	i	mot[i]	envers
départ	–	–	`""`
tour 1	0	S	`"S"`
tour 2	1	I	`"IS"`
tour 3	2	O	`"OIS"`

1. `"radar"` donne `"radar"` : ce mot se lit pareil dans les deux sens.

PYTHON

```python
mot = "SIO"
envers = ""
for i in range(len(mot)):
    envers = mot[i] + envers
print(envers)
```

CE QUI S'AFFICHE

```
OIS
```

Attention

`envers ← envers + mot[i]` colle la lettre derrière : on retrouve le mot dans le même ordre.

★★★ **Niveau 3 — Aller plus loin**

Exercice 4.2.7 — Le bit de parité ★★★ Sur machine

Pour détecter une erreur de transmission, on ajoute un bit à la fin d'un code binaire :

- si le code contient un nombre pair de « 1 », on ajoute `"0"` ;
- sinon, on ajoute `"1"`.

1. Écrivez l'algorithme qui part de `code ← "1011001"` et affiche le code complété.
2. Quel est le code complété pour `"1011001"` ? Pour `"1100000"` ?
3. Écrivez-le en Python et testez-le.
4. À l'arrivée, le récepteur vérifie le code de 8 bits : il doit contenir un nombre pair de « 1 ». Il reçoit `"10110011"`. Le code est-il arrivé intact ? Avec quel calcul de votre algorithme le savez-vous ?

CORRIGÉ

La méthode. Deux temps. D'abord, une boucle compte les « 1 ».

Ensuite, le reste de la division du compteur par 2 dit s'il est pair. On colle le bon bit au bout.

1. L'algorithme :

PSEUDO-CODE

```
code ← "1011001"
compt ← 0
Pour i allant de 0 à longueur(code) - 1 Faire
    Si code[i] = "1" Alors
        compt ← compt + 1
    Fin de Si
Fin de Pour
Si compt mod 2 = 0 Alors
    code ← code + "0"
Sinon
    code ← code + "1"
Fin de Si
Écrire code
```

1. `"1011001"` contient quatre « 1 », un nombre pair : on obtient `"10110010"`. `"1100000"` contient deux « 1 » : `"11000000"`.
2. En Python :

PYTHON

```python
code = input("Code sur 7 bits ? ")
compt = 0
for i in range(len(code)):
    if code[i] == "1":
        compt = compt + 1
if compt % 2 == 0:
    code = code + "0"
else:
    code = code + "1"
print(code)
```

CE QUI S'AFFICHE

```
Code sur 7 bits ? 1011001
10110010
```

1. On compte les « 1 » de `"10110011"` : 1, 0, 1, 1, 0, 0, 1, 1, soit cinq « 1 ». Cinq est impair : un bit a changé pendant le transport. On réutilise la boucle de comptage et le test `compt mod 2 = 0`, sans ajouter de bit.

Le sujet du BTS 2024 pose exactement cette question : voyez l'onglet Annales.

Exercice 4.2.8 — Le mot de passe sécurisé ★★★ Sur machine

Un mot de passe est valide s'il a au moins 8 caractères, au moins une majuscule et au moins un chiffre.

Pour « au moins une majuscule », on utilise un **drapeau** : un booléen qui vaut Faux au départ. Dès qu'on trouve une majuscule, il passe à Vrai, et il le reste.

Les caractères se comparent par leur code : les chiffres d'abord, puis les majuscules, puis les minuscules. Les majuscules de `"A"` à `"Z"` se suivent.

Donc un caractère `ch` est une majuscule si `ch ≥ "A" ET ch ≤ "Z"`. C'est un chiffre si `ch ≥ "0" ET ch ≤ "9"`.

1. Complétez l'algorithme.

PSEUDO-CODE

```
Lire m
maj ← Faux
chiffre ← Faux
Pour i allant de 0 à longueur(m) - 1 Faire
    ch ← m[i]
    Si ch ≥ "A" ET ch ≤ "Z" Alors
        . . .
    Fin de Si
    Si . . . Alors
        chiffre ← Vrai
    Fin de Si
Fin de Pour
valide ← longueur(m) ≥ 8 ET . . .
Écrire valide
```

1. Que donne l'algorithme pour `Sio2026!`, `motdepasse1` et `ABCD12` ? Présentez les résultats dans un tableau.

2. Écrivez le programme en Python et testez-le.

CORRIGÉ

La méthode. Un drapeau par règle « au moins un ». La boucle peut les faire passer à Vrai. À la fin, on relie toutes les règles avec ET.

1. L'algorithme complet :

PSEUDO-CODE

```
Lire m
maj ← Faux
chiffre ← Faux
Pour i allant de 0 à longueur(m) - 1 Faire
    ch ← m[i]
    Si ch ≥ "A" ET ch ≤ "Z" Alors
        maj ← Vrai
    Fin de Si
    Si ch ≥ "0" ET ch ≤ "9" Alors
        chiffre ← Vrai
    Fin de Si
Fin de Pour
valide ← longueur(m) ≥ 8 ET maj ET chiffre
Écrire valide
```

1. Les trois mots de passe :

Mot de passe	8 caractères	Majuscule	Chiffre	Valide
`Sio2026!`	oui	oui	oui	Vrai
`motdepasse1`	oui	non	oui	Faux
`ABCD12`	non	oui	oui	Faux

1. En Python :

PYTHON

```python
m = input("Mot de passe ? ")
maj = False
chiffre = False
for i in range(len(m)):
    ch = m[i]
    if ch >= "A" and ch <= "Z":
        maj = True
    if ch >= "0" and ch <= "9":
        chiffre = True
valide = len(m) >= 8 and maj and chiffre
print(valide)
```

CE QUI S'AFFICHE

```
Mot de passe ? Sio2026!
True
```

Attention

N'écrivez pas `Sinon maj ← Faux` dans la boucle : une seule lettre minuscule effacerait la majuscule trouvée avant.

Exercice 4.2.9 — Compter les diviseurs ★★★

Un **diviseur** de `n` est un entier `i` tel que `n mod i = 0`. Un nombre premier est un entier qui a exactement deux diviseurs : 1 et lui-même.

1. Écrivez un algorithme qui lit `n`, compte ses diviseurs entre 1 et `n`, puis affiche ce nombre.
2. Faites la trace pour `n = 6`. Que vaut le compteur à la fin ?
3. Ajoutez à la fin un test qui affiche « premier » ou « pas premier ».
4. Que donne l'algorithme pour 13 ?

CORRIGÉ

La méthode. On essaie chaque `i` de 1 à `n`. Si `n mod i = 0`, c'est un diviseur : on compte. Le test « premier » se fait après la boucle, une seule fois.

1. et 3. L'algorithme :

```
PSEUDO-CODE
Lire n
compt ← 0
Pour i allant de 1 à n Faire
    Si n mod i = 0 Alors
        compt ← compt + 1
    Fin de Si
Fin de Pour
Écrire compt
Si compt = 2 Alors
    Écrire "premier"
Sinon
    Écrire "pas premier"
Fin de Si
```

1. La trace pour 6 :

i	6 mod i	compt
1	0	1
2	0	2
3	0	3
4	2	3
5	1	3
6	0	4

Le compteur vaut 4 : 6 a quatre diviseurs, 1, 2, 3 et 6. Il n'est pas premier.

1. 13 a deux diviseurs, 1 et 13 : l'algorithme affiche 2, puis « premier ».

```
PYTHON
n = int(input("n ? "))
compt = 0
for i in range(1, n + 1):
    if n % i == 0:
        compt = compt + 1
print(compt)
if compt == 2:
    print("premier")
else:
    print("pas premier")
```

CE QUI S'AFFICHE

```
n ? 13
2
premier
```

Le sujet du BTS 2022 pose cette question avec la fonction `Div` : voyez l'onglet Annales.

4.3 — Répéter tant que : les boucles Tant que et Répéter

‹ Revoir le cours 4.3

★☆☆ **Niveau 1 — Appliquer**

Exercice 4.3.1 — Trace d'une boucle Tant que ★☆☆

PSEUDO-CODE

```
n ← 100
tours ← 0
Tant que n > 1 Faire
    n ← n div 3
    tours ← tours + 1
Fin de Tant que
Écrire n, tours
```

1. Faites la trace : une ligne par test de la condition.
2. Qu'affiche l'algorithme ?

CORRIGÉ

La méthode. On teste la condition avant chaque tour. On s'arrête dès qu'elle est fausse.

1. La trace :

Étape	n	tours	`n > 1 ?`
départ	100	0	Vrai
tour 1	33	1	Vrai
tour 2	11	2	Vrai
tour 3	3	3	Vrai
tour 4	1	4	Faux : on sort

1. L'algorithme affiche `1 4`.

PYTHON

```python
n = 100
tours = 0
while n > 1:
    n = n // 3
    tours = tours + 1
print(n, tours)
```

CE QUI S'AFFICHE

```
1 4
```

Exercice 4.3.2 — Un Pour écrit avec Tant que ★☆☆

PSEUDO-CODE

```
Pour i allant de 1 à 3 Faire
    Écrire i * 10
Fin de Pour
```

1. Réécrivez cette boucle avec un Tant que. N'oubliez ni la valeur de départ de `i`, ni la ligne qui le fait avancer.
2. Faites la trace des deux versions. Affichent-elles la même chose ?

CORRIGÉ

La méthode. On suit l'encadré du cours : `i` part de 1 avant la boucle, la condition est `i ≤ 3`, et `i ← i + 1` termine chaque tour.

1. La version Tant que :

```
PSEUDO-CODE
i ← 1
Tant que i ≤ 3 Faire
    Écrire i * 10
    i ← i + 1
Fin de Tant que
```

1. La trace de la version Tant que :

Étape	i	i ≤ 3 ?	Affiché
départ	1	Vrai	
tour 1	2	Vrai	10
tour 2	3	Vrai	20
tour 3	4	Faux : on sort	30

La boucle Pour affiche aussi 10, 20, 30 : les deux versions font la même chose. Seule différence : après le Tant que, `i` vaut 4.

```
PYTHON
i = 1
while i <= 3:
    print(i * 10)
    i = i + 1
```

```
CE QUI S'AFFICHE
10
20
30
```

Attention

Sans `i ← i + 1`, `i` reste à 1 : la condition est toujours vraie, c'est une boucle infinie.

Exercice 4.3.3 — Une note contrôlée ★☆☆ Sur machine

On veut lire une note. Tant qu'elle n'est pas entre 0 et 20, on la redemande.

1. Complétez l'algorithme.

PSEUDO-CODE

```
Répéter
    Écrire "Note sur 20 ?"
    Lire note
Jusqu'à note ≥ 0 ET . . .
Écrire "Note enregistrée :", note
```

1. Complétez la condition du programme Python, puis testez-le en tapant 25, puis -3, puis 14.

PYTHON

```python
while True:
    note = float(input("Note sur 20 ? "))
    if note >= 0:    # à compléter
        break
print("Note enregistrée :", note)
```

CORRIGÉ

La méthode. Répéter s'arrête quand la condition est **vraie** : on écrit la condition d'une bonne note.

1. `Jusqu'à note ≥ 0 ET note ≤ 20`
2. En Python :

PYTHON

```python
while True:
    note = float(input("Note sur 20 ? "))
    if note >= 0 and note <= 20:
        break
print("Note enregistrée :", note)
```

CE QUI S'AFFICHE

```
Note sur 20 ? 25
Note sur 20 ? -3
Note sur 20 ? 14
Note enregistrée : 14.0
```

Attention

Avec seulement `note >= 0`, la note 25 serait acceptée. Les deux bornes doivent être testées.

★★☆ **Niveau 2 — S'entraîner**

Exercice 4.3.4 — Chercher une lettre ★★☆

Cette fonction cherche la première position d'une lettre dans une chaîne. Si la lettre est absente, elle renvoie la longueur de la chaîne.

PSEUDO-CODE

```
Fonction chercher(lettre, chaine)
    k ← 0
    Tant que k < longueur(chaine) ET chaine[k] ≠ lettre Faire
        k ← k + 1
    Fin de Tant que
    Renvoyer k
Fin de Fonction
```

1. Faites la trace de `chercher("I", "BTSSIO")`. Que renvoie-t-elle ?
2. Que renvoie `chercher("A", "BTSSIO")` ? Pourquoi la boucle s'arrête-t-elle ?
3. Que se passerait-il sans la condition `k < longueur(chaine)` ?

CORRIGÉ

La méthode. On fait une ligne de trace par test de la condition.

1. La trace :

k	chaine[k]	chaine[k] ≠ "I"	On continue ?
0	B	Vrai	oui
1	T	Vrai	oui
2	S	Vrai	oui
3	S	Vrai	oui
4	I	Faux	non

La fonction renvoie 4.

1. `A` n'est jamais trouvé. `k` monte jusqu'à 6. Alors `k < 6` est faux, la boucle s'arrête : la fonction renvoie 6, la longueur.
2. Sans ce test, on lirait `chaine[6]`, qui n'existe pas : erreur `IndexError`.

On vérifie avec le même calcul, sans fonction :

PYTHON

```python
lettre = "A"
chaine = "BTSSIO"
k = 0
while k < len(chaine) and chaine[k] != lettre:
    k = k + 1
print(k)
```

CE QUI S'AFFICHE

```
6
```

Attention

L'ordre des deux tests compte. Le ET s'arrête dès que `k < len(chaine)` est faux : `chaine[k]` n'est jamais lu hors de la chaîne.

Exercice 4.3.5 — Quotient et reste par soustractions ★★☆ Sur machine

On veut le quotient et le reste de la division de `a` par `b` sans utiliser `div` ni `mod`. Idée : on enlève `b` à `a` tant que c'est possible, et on compte les soustractions.

1. Complétez l'algorithme.

PSEUDO-CODE

```
Lire a
Lire b
q ← 0
Tant que a ≥ . . . Faire
    a ← . . .
    q ← . . .
Fin de Tant que
Écrire "quotient :", q, "reste :", a
```

1. Faites la trace pour `a = 47` et `b = 15`.
2. Écrivez le programme en Python. Vérifiez avec `47 // 15` et `47 % 15`.

CORRIGÉ

La méthode. Chaque tour enlève une fois `b` : le nombre de tours est le quotient. Ce qui reste à la fin est le reste.

1. L'algorithme complet :

```
PSEUDO-CODE
Lire a
Lire b
q ← 0
Tant que a ≥ b Faire
    a ← a - b
    q ← q + 1
Fin de Tant que
Écrire "quotient :", q, "reste :", a
```

1. La trace :

Étape	a	q	a ≥ 15 ?
départ	47	0	Vrai
tour 1	32	1	Vrai
tour 2	17	2	Vrai
tour 3	2	3	Faux : on sort

Quotient 3, reste 2 : $47 = 15 \times 3 + 2$.

1. En Python :

```
PYTHON
a = int(input("a ? "))
b = int(input("b ? "))
print(a // b, a % b)
q = 0
while a >= b:
    a = a - b
    q = q + 1
print("quotient :", q, "reste :", a)
```

```
CE QUI S'AFFICHE
a ? 47
b ? 15
3 2
quotient : 3 reste : 2
```

Exercice 4.3.6 — Le disque qui se remplit ★★☆

Un serveur stocke 1 Go de journaux le premier jour. Chaque jour, la taille des journaux double : c'est la suite $t_{n+1} = 2 \times t_n$, avec $t_1 = 1$. Le disque fait 1 000 Go.

1. Complétez l'algorithme qui cherche le premier jour où la taille dépasse 1 000 Go.

PSEUDO-CODE

```
taille ← 1
jours ← 1
Tant que taille ≤ . . . Faire
    taille ← . . .
    jours ← jours + 1
Fin de Tant que
Écrire jours, taille
```

1. Qu'affiche-t-il ?
2. Pourquoi une boucle Pour ne convient-elle pas ici ?

CORRIGÉ

La méthode. On double la taille à chaque tour, et on compte les jours, tant qu'on ne dépasse pas 1 000.

1. `Tant que taille ≤ 1000 Faire` et `taille ← taille * 2`. Cette ligne calcule le terme suivant de la suite.
2. Les tailles sont 1, 2, 4, 8, … 512, puis 1 024. Il affiche `11 1024` : le 11e jour est le premier où la taille dépasse 1 000 Go.
3. On ne sait pas d'avance combien de jours il faudra : c'est le résultat qu'on cherche.

PYTHON

```python
taille = 1
jours = 1
while taille <= 1000:
    taille = taille * 2
    jours = jours + 1
print(jours, taille)
```

CE QUI S'AFFICHE

```
11 1024
```

★★★ **Niveau 3 — Aller plus loin**

Exercice 4.3.7 — La boucle qui ne s'arrête pas ★★★

Un stagiaire veut afficher les nombres de 10 à 1, en descendant. Il a écrit :

PSEUDO-CODE

```
n ← 10
Tant que n ≥ 0 Faire
    Écrire n
    n ← n + 1
Fin de Tant que
```

1. Que se passe-t-il quand on lance cet algorithme ?
2. Cet algorithme contient **deux** erreurs. Trouvez-les et corrigez-les.
3. Avec la version corrigée, combien de fois la condition est-elle testée ? Quelle est la valeur de `n` après la boucle ?
4. Quelle valeur limite faut-il vérifier pour repérer la deuxième erreur ?

CORRIGÉ

La méthode. On regarde si le bloc rapproche la condition de Faux. Puis on vérifie la dernière valeur affichée : c'est là que se cachent les erreurs de borne.

1. C'est une boucle infinie : `n` grandit, donc `n ≥ 0` reste vrai. L'algorithme affiche 10, 11, 12… sans jamais s'arrêter.
2. Première erreur : `n ← n + 1` fait monter `n`. Il faut `n ← n - 1`. Deuxième erreur : avec `n ≥ 0`, la boucle afficherait aussi 0. Il faut `n > 0`.
3. La condition est testée 11 fois : 10 fois vraie, pour n = 10 à 1, puis une fois fausse, pour n = 0. Après la boucle, `n` vaut 0.
4. On regarde la valeur limite : le dernier nombre affiché doit être 1. Une trace des derniers tours, avec n = 2, 1, 0, montre que la version avec `≥` affiche 0.

PYTHON

```python
n = 10
while n > 0:
    print(n)
    n = n - 1
print("après la boucle :", n)
```

CE QUI S'AFFICHE

```
10
9
8
7
6
5
4
3
2
1
après la boucle : 0
```

Exercice 4.3.8 — Le nombre mystère ★★★ Sur machine

Le programme cache le nombre 42. L'utilisateur propose des nombres.

Le programme répond « Plus grand », « Plus petit », ou « Bravo ». Il compte les essais.

1. Quelle boucle choisir ? Pourquoi ?
2. Écrivez l'algorithme en pseudo-code. Ajoutez un en-tête : nom, rôle, entrées, sortie. Ajoutez un commentaire pour chaque variable. Le programme demande un en-tête et des commentaires : ils rendent la copie lisible pour le correcteur.
3. Écrivez le programme en Python, avec les mêmes commentaires, et testez-le avec 50, puis 30, puis 42.

PYTHON

```python
secret = 42
essais = 0
# à compléter
```

CORRIGÉ

La méthode. On ne sait pas combien d'essais il faudra, mais il en faut au moins un : Répéter … Jusqu'à. Dans la boucle, un `Si` choisit le message.

1. **Répéter … Jusqu'à**, car on propose au moins un nombre, sans savoir combien.
2. L'algorithme :

PSEUDO-CODE

```
// Nom : nombre_mystere
// Rôle : faire deviner un nombre caché, en guidant le joueur
// Entrées : les nombres proposés au clavier
// Sortie : un message par essai, puis le nombre d'essais
secret ← 42      // le nombre à deviner
essais ← 0       // compteur des propositions
Répéter
    Lire n       // n : la proposition du joueur
    essais ← essais + 1
    Si n < secret Alors
        Écrire "Plus grand"
    Sinon si n > secret Alors
        Écrire "Plus petit"
    Fin de Si
Jusqu'à n = secret
Écrire "Bravo, trouvé en", essais, "essais"
```

1. En Python :

PYTHON

```python
# Nom : nombre_mystere
# Rôle : faire deviner un nombre caché, en guidant le joueur
# Entrées : les nombres proposés au clavier
# Sortie : un message par essai, puis le nombre d'essais
secret = 42      # le nombre à deviner
essais = 0       # compteur des propositions
while True:
    n = int(input("Votre nombre ? "))    # la proposition du joueur
    essais = essais + 1
    if n < secret:
        print("Plus grand")
    elif n > secret:
        print("Plus petit")
    if n == secret:
        break
print("Bravo, trouvé en", essais, "essais")
```

CE QUI S'AFFICHE

```
Votre nombre ? 50
Plus petit
Votre nombre ? 30
Plus grand
Votre nombre ? 42
Bravo, trouvé en 3 essais
```

Pour un vrai jeu, on tire le secret au hasard : `secret = random.randint(1, 100)`.

Exercice 4.3.9 — Le PGCD par soustractions ★★★

Le **PGCD** de deux entiers, pour « plus grand commun diviseur », est le plus grand entier qui divise les deux. Une méthode ancienne le calcule par soustractions : tant que les deux nombres sont différents, on enlève le plus petit au plus grand. Quand ils sont égaux, ils valent le PGCD.

1. Écrivez l'algorithme en pseudo-code. Il faut un `Si` dans la boucle.
2. Faites la trace pour `a = 84` et `b = 36`. Quel est le PGCD ?
3. Pourquoi la boucle s'arrête-t-elle toujours, avec deux entiers positifs ?

CORRIGÉ

La méthode. La condition de la boucle est `a ≠ b`. Dans la boucle, un `Si` choisit le nombre qui diminue.

1. L'algorithme :

```
PSEUDO-CODE
Lire a
Lire b
Tant que a ≠ b Faire
    Si a > b Alors
        a ← a - b
    Sinon
        b ← b - a
    Fin de Si
Fin de Tant que
Écrire "PGCD :", a
```

1. La trace :

Étape	a	b	a ≠ b ?
départ	84	36	Vrai
tour 1	48	36	Vrai
tour 2	12	36	Vrai
tour 3	12	24	Vrai
tour 4	12	12	Faux : on sort

Le PGCD de 84 et 36 est 12.

1. À chaque tour, un des deux nombres diminue, et il reste positif. Ils ne peuvent pas diminuer pour toujours : ils finissent par devenir égaux.

PYTHON

```python
a = 84
b = 36
while a != b:
    if a > b:
        a = a - b
    else:
        b = b - a
print("PGCD :", a)
```

CE QUI S'AFFICHE

```
PGCD : 12
```

Le sujet du BTS 2026 utilise une fonction `pgcd` toute faite : voyez l'onglet Annales.

4.4 — Construire une boucle de calcul

‹ Revoir le cours 4.4

★☆☆ **Niveau 1 — Appliquer**

Exercice 4.4.1 — Les trois temps ★☆☆

Voici un algorithme qui calcule la moyenne de 4 notes lues au clavier.

PSEUDO-CODE

```
total ← 0
Pour i allant de 1 à 4 Faire
    Lire note
    total ← total + note
Fin de Pour
moyenne ← total / 4
Écrire "Moyenne :", arrondi(moyenne, 2)
```

1. Quelle ligne est la préparation ?
2. Quelles lignes forment l'itération ?
3. Quelles lignes forment la mise en forme finale ?

4. Quel est l'accumulateur ? Pourquoi part-il de 0 ?

CORRIGÉ

La méthode. La préparation est avant la boucle, l'itération dedans, la mise en forme après.

1. Préparation : `total ← 0`.
2. Itération : la boucle Pour, avec `Lire note` et `total ← total + note`.
3. Mise en forme finale : `moyenne ← total / 4` et l'affichage arrondi.
4. L'accumulateur est `total`. Il part de 0 car 0 ne change pas une somme.

Exercice 4.4.2 — La somme des entiers ★☆☆

On veut calculer $\sum_{i=1}^{10} i$, la somme des entiers de 1 à 10.

1. Complétez l'algorithme.

PSEUDO-CODE

```
s ← . . .
Pour i allant de 1 à . . . Faire
    s ← . . .
Fin de Pour
Écrire s
```

1. Quel résultat obtenez-vous ?

CORRIGÉ

La méthode. Un Σ devient une boucle Pour avec un accumulateur qui part de 0.

1. L'algorithme complet :

PSEUDO-CODE

```
s ← 0
Pour i allant de 1 à 10 Faire
    s ← s + i
Fin de Pour
Écrire s
```

1. On obtient 55.

PYTHON

```python
s = 0
for i in range(1, 11):
    s = s + i
print(s)
```

CE QUI S'AFFICHE

```
55
```

Exercice 4.4.3 — La factorielle ★☆☆

Le produit $\prod_{i=1}^{6} i = 1 \times 2 \times 3 \times 4 \times 5 \times 6$ s'appelle la factorielle de 6.

1. Complétez l'algorithme.

PSEUDO-CODE

```
p ← . . .
Pour i allant de 1 à 6 Faire
    p ← . . .
Fin de Pour
Écrire p
```

1. Quel résultat obtenez-vous ?
2. Que se passerait-il avec `p ← 0` au départ ?

CORRIGÉ

La méthode. Un Π devient une boucle Pour avec un accumulateur qui part de 1.

1. `p ← 1` et `p ← p * i`.
2. On obtient 720.
3. Le produit resterait 0 à chaque tour : 0 multiplié par n'importe quoi donne 0.

PYTHON

```python
p = 1
for i in range(1, 7):
    p = p * i
print(p)
```

CE QUI S'AFFICHE

```
720
```

★★☆ **Niveau 2 — S'entraîner**

Exercice 4.4.4 — La moyenne de n notes ★★☆ Sur machine

Le programme lit d'abord le nombre de notes `n`, puis les `n` notes. Il affiche la moyenne arrondie à 2 décimales.

1. Écrivez l'algorithme en pseudo-code. Repérez les trois temps.
2. Complétez le programme Python et testez-le avec 3 notes : 12, 15 et 10.

PYTHON

```python
n = int(input("Nombre de notes ? "))
total = 0
# à compléter
```

CORRIGÉ

La méthode. On connaît le nombre de tours une fois `n` lu : c'est une boucle Pour de 1 à `n`. La division se fait après la boucle.

1. L'algorithme :

PSEUDO-CODE

```
Lire n
total ← 0
Pour i allant de 1 à n Faire
    Lire note
    total ← total + note
Fin de Pour
Écrire "Moyenne :", arrondi(total / n, 2)
```

Préparation : `Lire n` et `total ← 0`. Itération : la boucle. Mise en forme finale : la division, l'arrondi, l'affichage.

1. En Python :

PYTHON

```python
n = int(input("Nombre de notes ? "))
total = 0
for i in range(1, n + 1):
    note = float(input("Note ? "))
    total = total + note
print("Moyenne :", round(total / n, 2))
```

CE QUI S'AFFICHE

```
Nombre de notes ? 3
Note ? 12
Note ? 15
Note ? 10
Moyenne : 12.33
```

Exercice 4.4.5 — La somme des impairs ★★☆

On veut calculer $S = \sum_{i=1}^{n} (2i - 1)$. On lit : somme des $2i - 1$, pour i allant de 1 à n.

1. Écrivez l'algorithme en pseudo-code.
2. Calculez S pour n = 5 : écrivez la valeur de S à la fin de chaque tour.
3. Quelle propriété de S est vraie à la fin de chaque tour ? Pensez à l'encadré « Construire la boucle avec une propriété ».
4. Calculez S pour n = 10.

CORRIGÉ

La méthode. On traduit le Σ en boucle Pour, et le terme $2i - 1$ en `2 * i - 1`.

1. L'algorithme :

PSEUDO-CODE

```
Lire n
S ← 0
Pour i allant de 1 à n Faire
    S ← S + (2 * i - 1)
Fin de Pour
Écrire S
```

1. Pour n = 5 :

Tour i	Terme ajouté	S à la fin du tour
1	1	1
2	3	4
3	5	9
4	7	16
5	9	25

1. À la fin du tour i, S vaut i^2 : 1, 4, 9, 16, 25. C'est vrai au départ, avec S = 0 = 0^2. On le vérifie sur la trace : 0 + 1 = 1, 1 + 3 = 4, 4 + 5 = 9. Chaque tour garde la propriété vraie.
2. Pour n = 10 : 100, soit 10^2. La somme des n premiers nombres impairs vaut n^2.

PYTHON

```python
n = 10
S = 0
for i in range(1, n + 1):
    S = S + (2 * i - 1)
print(S, n ** 2)
```

CE QUI S'AFFICHE

```
100 100
```

Exercice 4.4.6 — Simuler 600 lancers de dé ★★☆ Sur machine

Un dé bien équilibré donne chaque face avec la même chance : une fois sur six, soit environ 0,167. On veut le vérifier en simulant 600 lancers.

`random.randint(1, 6)` tire un entier au hasard entre 1 et 6. `random.seed(1)` fixe le point de départ du hasard : chacun obtient alors les mêmes tirages, et on peut comparer les résultats.

1. Repérez les trois temps : quelle préparation, quelle mise à jour dans la boucle, quelle mise en forme finale ?
2. Complétez le programme, puis exécutez-le. Combien de 6 sortent ? Quelle est leur fréquence, arrondie au millième ?
3. Remplacez `random.seed(1)` par `random.seed(2)` : le résultat change-t-il ? Reste-t-il proche de 0,167 ?

PYTHON

```python
import random
random.seed(1)
nb6 = 0
for i in range(1, 601):
    de = random.randint(1, 6)
    # à compléter : compter les 6
print(nb6, round(nb6 / 600, 3))
```

CORRIGÉ

La méthode. C'est un compteur dans une boucle Pour de 600 tours. La fréquence se calcule après la boucle : c'est la mise en forme finale.

1. Préparation : `nb6 ← 0`. Itération : à chaque tour, on tire le dé et on ajoute 1 à `nb6` si le dé vaut 6. Mise en forme finale : la fréquence `nb6 / 600`, arrondie.
2. Le programme complet :

PYTHON

```python
import random
random.seed(1)
nb6 = 0
for i in range(1, 601):
    de = random.randint(1, 6)
    if de == 6:
        nb6 = nb6 + 1
print(nb6, round(nb6 / 600, 3))
```

CE QUI S'AFFICHE

```
84 0.14
```

Avec `random.seed(1)`, on obtient 84 six, soit une fréquence de 0,14.

1. Avec une autre graine, les tirages changent, et le nombre de 6 aussi. Il reste autour de 100, soit une fréquence proche de 0,167. Plus on fait de lancers, plus la fréquence s'approche de 1/6.

★★★ **Niveau 3 — Aller plus loin**

Exercice 4.4.7 — La puissance sans le signe ^ ★★★

On veut calculer a^n en multipliant `a` par lui-même, sans utiliser `^`.

1. Complétez l'écriture de la puissance sous la forme d'un Π : $a^n = \prod_{i=1}^{n} \ldots$ Chaque facteur du produit est le même nombre.
2. Écrivez l'algorithme qui lit `a` et `n`, puis affiche a^n.
3. Faites la trace pour `a = 3` et `n = 4`.
4. Que donne votre algorithme pour `n = 0` ? Est-ce juste ?

CORRIGÉ

La méthode. a^n est un produit de `n` facteurs tous égaux à `a`. Un produit part de 1.

1. $a^n = \prod_{i=1}^{n} a$.
2. L'algorithme :

PSEUDO-CODE

```
Lire a
Lire n
p ← 1
Pour i allant de 1 à n Faire
    p ← p * a
Fin de Pour
Écrire p
```

1. La trace :

Étape	i	p
départ	–	1
tour 1	1	3
tour 2	2	9
tour 3	3	27
tour 4	4	81

1. Pour `n = 0`, la boucle ne fait aucun tour : on affiche 1. C'est juste, car $a^0 = 1$.

PYTHON

```python
a = 3
n = 4
p = 1
for i in range(1, n + 1):
    p = p * a
print(p, a ** n)
```

CE QUI S'AFFICHE

```
81 81
```

Exercice 4.4.8 — Une somme qui approche π ★★★ Sur machine

La somme $\sum_{k=1}^{n} \frac{1}{k^2}$ s'approche de $\frac{\pi^2}{6} \approx 1{,}6449$ quand n grandit.

1. Écrivez la somme en boucle, en Python.
2. Calculez-la pour n = 10, n = 100 et n = 1000. Arrondissez à 4 décimales.
3. Pourquoi faut-il écrire `1 / k ** 2` et pas `1 / k * 2` ?

PYTHON

```python
n = 10
s = 0
# à compléter
print(round(s, 4))
```

CORRIGÉ

La méthode. Un Σ devient une boucle Pour qui part de 1, avec un accumulateur à 0. Le terme $\frac{1}{k^2}$ s'écrit `1 / k ** 2`.

1. et 2. En Python :

PYTHON

```python
n = 10
s = 0
for k in range(1, n + 1):
    s = s + 1 / k ** 2
print(round(s, 4))
```

CE QUI S'AFFICHE

```
1.5498
```

On obtient 1,5498 pour n = 10, 1,635 pour n = 100, et 1,6439 pour n = 1 000. On se rapproche de 1,6449.

1. La puissance passe avant la division : `1 / k ** 2` calcule bien $\frac{1}{k^2}$. Avec `1 / k * 2`, on calcule $\frac{1}{k} \times 2$.

Exercice 4.4.9 — La température la plus haute ★★★

Une sonde relève 5 températures, une par jour, du jour 1 au jour 5. Le programme les lit une par une. Il affiche la plus haute, et le jour où elle a été relevée.

1. Un collègue propose de partir de `maxi ← 0`. Pourquoi est-ce une erreur ? Donnez un jeu de températures qui le montre.
2. Écrivez l'algorithme. Quelles sont les deux variables à mettre à jour quand on trouve une température plus haute ?
3. Faites la trace avec -3, 5, 2, 8, -1.
4. Si deux jours ont la même température la plus haute, quel jour votre algorithme affiche-t-il ? Que faudrait-il changer pour afficher l'autre ?

CORRIGÉ

La méthode. L'accumulateur garde la plus grande valeur vue. Il part d'une vraie valeur : la première lue. Quand on trouve plus haut, on garde aussi le numéro du jour.

1. Si toutes les températures sont négatives, par exemple -5, -2, -8, -4, -6, le maximum trouvé serait 0. Or 0 n'est pas une des valeurs.
2. Les deux variables sont `maxi` et `jour_maxi`. L'algorithme :

PSEUDO-CODE

```
Lire t
maxi ← t
jour_maxi ← 1
Pour i allant de 2 à 5 Faire
    Lire t
    Si t > maxi Alors
        maxi ← t
        jour_maxi ← i
    Fin de Si
Fin de Pour
Écrire "Maximum :", maxi, "le jour", jour_maxi
```

1. La trace :

Étape	t	maxi	jour_maxi
départ	-3	-3	1
i = 2	5	5	2
i = 3	2	5	2
i = 4	8	8	4
i = 5	-1	8	4

1. Avec `t > maxi`, une égalité ne change rien : on garde le premier jour. Avec `t ≥ maxi`, on garde le dernier.

PYTHON

```python
t = float(input("Température ? "))
maxi = t
jour_maxi = 1
for i in range(2, 6):
    t = float(input("Température ? "))
    if t > maxi:
        maxi = t
        jour_maxi = i
print("Maximum :", maxi, "le jour", jour_maxi)
```

CE QUI S'AFFICHE

```
Température ? -3
Température ? 5
Température ? 2
Température ? 8
Température ? -1
Maximum : 8.0 le jour 4
```

4.5 — Les structures imbriquées

‹ Revoir le cours 4.5

★☆☆ **Niveau 1 — Appliquer**

Exercice 4.5.1 — Compter les passages ★☆☆

PSEUDO-CODE

```
compt ← 0
Pour i allant de 1 à 4 Faire
    Pour j allant de 1 à 3 Faire
        compt ← compt + 1
    Fin de Pour
Fin de Pour
Écrire compt
```

1. Combien de fois la ligne `compt ← compt + 1` est-elle exécutée ?
2. Donnez les trois premiers couples (i, j), dans l'ordre.
3. Que vaut `j` quand `i` passe de 1 à 2 ?

CORRIGÉ

La méthode. La boucle intérieure fait tous ses tours à chaque tour de la boucle extérieure.

1. 4 × 3 = 12 fois. L'algorithme affiche 12.
2. (1, 1), puis (1, 2), puis (1, 3).
3. `j` repart à 1 : la boucle intérieure recommence depuis le début.

PYTHON

```python
compt = 0
for i in range(1, 5):
    for j in range(1, 4):
        compt = compt + 1
print(compt)
```

CE QUI S'AFFICHE

```
12
```

Exercice 4.5.2 — Le rectangle d'étoiles ★☆☆ Sur machine

On veut afficher un rectangle de 3 lignes et 5 étoiles par ligne.

1. Complétez l'algorithme.

PSEUDO-CODE

```
Pour i allant de 1 à . . . Faire
    ligne ← ""
    Pour j allant de 1 à . . . Faire
        ligne ← ligne + "*"
    Fin de Pour
    Écrire ligne
Fin de Pour
```

1. Complétez le programme Python et exécutez-le.

PYTHON

```python
for i in range(1, 2):    # à corriger
    ligne = ""
    for j in range(1, 2):    # à corriger
        ligne = ligne + "*"
    print(ligne)
```

CORRIGÉ

La méthode. La boucle extérieure compte les lignes, la boucle intérieure les étoiles d'une ligne.

1. `Pour i allant de 1 à 3` et `Pour j allant de 1 à 5`.
2. En Python, la borne de droite est exclue :

PYTHON

```python
for i in range(1, 4):
    ligne = ""
    for j in range(1, 6):
        ligne = ligne + "*"
    print(ligne)
```

CE QUI S'AFFICHE

```
*****
*****
*****
```

Attention

`ligne ← ""` est dans la boucle extérieure : chaque ligne repart vide. Placé avant toutes les boucles, il ferait grandir une seule ligne.

Exercice 4.5.3 — Les multiples de 3 ★☆☆

On veut compter les multiples de 3 entre 1 et 20.

1. Complétez l'algorithme.

```
PSEUDO-CODE
compt ← 0
Pour i allant de 1 à 20 Faire
    Si . . . Alors
        compt ← compt + 1
    Fin de Si
Fin de Pour
Écrire compt
```

1. Que vaut `compt` à la fin ? Donnez la liste des nombres comptés.

CORRIGÉ

La méthode. Un `Si` dans une boucle : la boucle passe sur chaque nombre, le `Si` décide s'il compte.

1. `Si i mod 3 = 0 Alors`.
2. `compt` vaut 6 : 3, 6, 9, 12, 15, 18.

```python
compt = 0
for i in range(1, 21):
    if i % 3 == 0:
        compt = compt + 1
print(compt)
```

```
CE QUI S'AFFICHE
6
```

★★☆ Niveau 2 — S'entraîner

Exercice 4.5.4 — Les paires de dés ★★☆

On lance deux dés à 6 faces. On veut compter les tirages dont la somme vaut 7.

1. Complétez l'algorithme.

PSEUDO-CODE

```
compt ← 0
Pour i allant de 1 à 6 Faire
    Pour j allant de 1 à 6 Faire
        Si . . . Alors
            . . .
        Fin de Si
    Fin de Pour
Fin de Pour
Écrire compt
```

1. Combien de fois le test `Si` est-il évalué en tout ?
2. Que vaut `compt` à la fin ? Donnez la liste des tirages.

CORRIGÉ

La méthode. Deux boucles imbriquées parcourent tous les couples (i, j). Le test s'exécute une fois par couple.

1. Le test est `i + j = 7`, et on fait `compt ← compt + 1`.
2. 6 × 6 = 36 fois.
3. `compt` vaut 6 : (1, 6), (2, 5), (3, 4), (4, 3), (5, 2), (6, 1).

PYTHON

```python
compt = 0
for i in range(1, 7):
    for j in range(1, 7):
        if i + j == 7:
            compt = compt + 1
print(compt)
```

CE QUI S'AFFICHE

```
6
```

Attention

`compt ← 0` va **avant** les deux boucles. Placé dans une boucle, il remettrait le compte à zéro en cours de route.

Exercice 4.5.5 — La grille de multiplication ★★☆ Sur machine

On veut afficher la grille des produits de 1 à 4, une ligne par valeur de `i` :

CE QUI S'AFFICHE

```
1 2 3 4
2 4 6 8
3 6 9 12
4 8 12 16
```

1. Écrivez l'algorithme avec deux boucles imbriquées. Chaque ligne est une chaîne qu'on construit par concaténation.
2. Écrivez le programme Python. Il faut `str(...)` pour coller un nombre à une chaîne.

PYTHON

```python
for i in range(1, 5):
    ligne = ""
    # à compléter
    print(ligne)
```

CORRIGÉ

La méthode. La boucle extérieure choisit la ligne `i`. La boucle intérieure ajoute les produits `i × j` à la chaîne de la ligne.

1. L'algorithme :

PSEUDO-CODE

```
Pour i allant de 1 à 4 Faire
    ligne ← ""
    Pour j allant de 1 à 4 Faire
        ligne ← ligne + chaîne(i * j) + " "
    Fin de Pour
    Écrire ligne
Fin de Pour
```

1. En Python :

PYTHON

```python
for i in range(1, 5):
    ligne = ""
    for j in range(1, 5):
        ligne = ligne + str(i * j) + " "
    print(ligne)
```

CE QUI S'AFFICHE

```
1 2 3 4
2 4 6 8
3 6 9 12
4 8 12 16
```

Chaque ligne se termine par un espace en trop. Il ne se voit pas à l'écran.

Exercice 4.5.6 — Le triangle de nombres ★★☆

PSEUDO-CODE

```
total ← 0
Pour i allant de 1 à 4 Faire
    ligne ← ""
    Pour j allant de 1 à i Faire
        ligne ← ligne + chaîne(j) + " "
        total ← total + 1
    Fin de Pour
    Écrire ligne
Fin de Pour
Écrire total
```

1. Qu'affiche cet algorithme ? Écrivez les lignes.
2. Combien de fois la boucle intérieure tourne-t-elle quand `i` vaut 3 ?
3. Que vaut `total` à la fin ?

CORRIGÉ

La méthode. La boucle intérieure dépend de la boucle extérieure : elle va jusqu'à `i`. Elle fait donc 1 tour, puis 2, puis 3, puis 4.

1. Les lignes affichées :

CE QUI S'AFFICHE

```
1
1 2
1 2 3
1 2 3 4
10
```

1. Quand `i` vaut 3, la boucle intérieure fait 3 tours.
2. `total` vaut 1 + 2 + 3 + 4 = 10.

PYTHON

```python
total = 0
for i in range(1, 5):
    ligne = ""
    for j in range(1, i + 1):
        ligne = ligne + str(j) + " "
        total = total + 1
    print(ligne)
print(total)
```

CE QUI S'AFFICHE

```
1
1 2
1 2 3
1 2 3 4
10
```

★★★ **Niveau 3 — Aller plus loin**

Exercice 4.5.7 — Encore des dés ★★★

On reprend les deux dés à 6 faces de l'exercice 4.5.4.

1. Écrivez un seul algorithme, avec une seule double boucle et deux compteurs. Le premier compte les tirages où le premier dé est plus grand que le second. Le second compte les tirages dont la somme est au moins 10.
2. Donnez les deux résultats.
3. Sans ordinateur, expliquez pourquoi le premier compteur vaut 15. Indice : il y a 36 tirages, dont 6 doubles.
4. Combien de tirages vérifient les deux conditions à la fois ? Quelle ligne ajouteriez-vous pour les compter ?

CORRIGÉ

La méthode. On garde les deux boucles. On change le test, ou on ajoute un deuxième compteur avec son propre `Si`.

1. L'algorithme :

PSEUDO-CODE

```
plus_grand ← 0
grande_somme ← 0
Pour i allant de 1 à 6 Faire
    Pour j allant de 1 à 6 Faire
        Si i > j Alors
            plus_grand ← plus_grand + 1
        Fin de Si
        Si i + j ≥ 10 Alors
            grande_somme ← grande_somme + 1
        Fin de Si
    Fin de Pour
Fin de Pour
Écrire plus_grand, grande_somme
```

1. 15 tirages où le premier dé est plus grand. 6 tirages avec une somme d'au moins 10 : (4, 6), (5, 5), (5, 6), (6, 4), (6, 5), (6, 6).
2. Sur 36 tirages, 6 sont des doubles, où les deux dés sont égaux. Les 30 autres se partagent en deux moitiés égales : premier dé plus grand, ou second dé plus grand. 30 ÷ 2 = 15.
3. Deux tirages : (6, 4) et (6, 5). On ajouterait `Si i > j ET i + j ≥ 10 Alors les_deux ← les_deux + 1 Fin de Si`, avec `les_deux ← 0` avant les boucles.

PYTHON

```python
plus_grand = 0
grande_somme = 0
for i in range(1, 7):
    for j in range(1, 7):
        if i > j:
            plus_grand = plus_grand + 1
        if i + j >= 10:
            grande_somme = grande_somme + 1
print(plus_grand, grande_somme)
```

CE QUI S'AFFICHE

```
15 6
```

Attention

Ce sont deux `Si` séparés, pas un `Sinon si` : un même tirage, comme (6, 4), compte dans les deux.

Exercice 4.5.8 — Le produit de deux matrices ★★★ Sur machine

On veut calculer $C = A \times B$ avec

$$A = \begin{pmatrix} 2 & 1 \\ 0 & 3 \end{pmatrix} \qquad B = \begin{pmatrix} 1 & 4 \\ 2 & 5 \end{pmatrix}$$

La formule, donnée comme à l'examen : $c_{ij} = \sum_{k=1}^{2} a_{ik} \times b_{kj}$.

Une matrice s'écrit en Python comme une liste de lignes : `A = [[2, 1], [0, 3]]`. `A[i][k]` est le nombre de la ligne `i`, colonne `k`, et les indices partent de 0. La ligne qui applique la formule vous est donnée : `s = s + A[i][k] * B[k][j]`.

1. Calculez à la main c_{11}, le nombre en haut à gauche de C.
2. Exécutez le programme ci-dessous : le résultat est faux. Corrigez la borne de la boucle sur `k`.
3. Le résultat est encore faux : la ligne `s = 0` est mal placée. Déplacez-la au bon endroit, et expliquez pourquoi.

PYTHON

```
A = [[2, 1], [0, 3]]
B = [[1, 4], [2, 5]]
C = [[0, 0], [0, 0]]
s = 0                          # à déplacer au bon endroit
for i in range(2):
    for j in range(2):
        for k in range(1):     # à corriger : la borne
            s = s + A[i][k] * B[k][j]
        C[i][j] = s
print(C)
```

CORRIGÉ

La méthode. Trois boucles : `i` pour la ligne de C, `j` pour la colonne, `k` pour la somme de la formule.

1. $c_{11} = a_{11} \times b_{11} + a_{12} \times b_{21} = 2 \times 1 + 1 \times 2 = 4$.
2. La somme a deux termes, k vaut 0 puis 1 : il faut `range(2)`.
3. Le programme complet :

PYTHON

```python
A = [[2, 1], [0, 3]]
B = [[1, 4], [2, 5]]
C = [[0, 0], [0, 0]]
for i in range(2):
    for j in range(2):
        s = 0
        for k in range(2):
            s = s + A[i][k] * B[k][j]
        C[i][j] = s
print(C)
```

CE QUI S'AFFICHE

```
[[4, 13], [6, 15]]
```

Donc $C = \begin{pmatrix} 4 & 13 \\ 6 & 15 \end{pmatrix}$.

`s = 0` se place juste avant la boucle sur `k`, dans la boucle sur `j`. Chaque case de C a sa propre somme. Placé avant toutes les boucles, `s` additionne les sommes de toutes les cases : seule la première case est juste.

> **Attention**
>
> Dans la formule, les indices vont de 1 à 2. En Python, ils vont de 0 à 1 : c_{11} est `C[0][0]`.

Exercice 4.5.9 — Les nombres premiers jusqu'à 30 ★★★

On veut afficher tous les nombres premiers de 2 à 30. Un nombre est premier s'il a exactement deux diviseurs : 1 et lui-même.

1. Écrivez l'algorithme avec deux boucles imbriquées : la boucle extérieure choisit `n`, la boucle intérieure compte ses diviseurs.
2. La boucle intérieure dépend-elle de la boucle extérieure ? Expliquez.
3. Quels nombres sont affichés ?
4. Combien de tests de reste votre algorithme fait-il pour le seul `n = 30` ? Comment en faire beaucoup moins ? Relisez l'encadré « Combien de tours ? ».

CORRIGÉ

La méthode. Pour chaque `n`, on recompte ses diviseurs à partir de 0. Le test « premier » se fait après la boucle intérieure, dans la boucle extérieure.

1. L'algorithme :

PSEUDO-CODE

```
Pour n allant de 2 à 30 Faire
    compt ← 0
    Pour i allant de 1 à n Faire
        Si n mod i = 0 Alors
            compt ← compt + 1
        Fin de Si
    Fin de Pour
    Si compt = 2 Alors
        Écrire n
    Fin de Si
Fin de Pour
```

1. Oui : la boucle intérieure va jusqu'à `n`, la valeur de la boucle extérieure. Elle fait 2 tours pour `n = 2`, 30 tours pour `n = 30`.
2. On affiche 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, un par ligne.
3. Pour `n = 30`, la boucle intérieure fait 30 tests de reste. Pour en faire moins, on cherche un diviseur entre 2 et `n - 1`, et on s'arrête au premier trouvé. Pour 30, le premier test, avec 2, suffit. C'est la version B de l'encadré « Combien de tours ? ».

PYTHON

```python
for n in range(2, 31):
    compt = 0
    for i in range(1, n + 1):
        if n % i == 0:
            compt = compt + 1
    if compt == 2:
        print(n)
```

CE QUI S'AFFICHE

```
2
3
5
7
11
13
17
19
23
29
```

Attention

`compt ← 0` va dans la boucle extérieure, avant la boucle intérieure. Sinon, les diviseurs de tous les nombres s'additionnent.

5.1 — Les tableaux à une dimension

‹ Revoir le cours 5.1

★☆☆ **Niveau 1 — Appliquer**

Exercice 5.1.1 — Lire et écrire dans un tableau ★☆☆

On a `T ← [4, 9, 2, 7, 5]`.

1. Que valent `T[0]`, `T[4]` et `longueur(T)` ?
2. On exécute `T[2] ← T[1] + T[3]`. Écrivez le tableau obtenu.
3. On exécute ensuite `T[0] ← T[0] * 10`. Écrivez le tableau obtenu.
4. Que se passe-t-il avec `T[5] ← 1` ?

CORRIGÉ

La méthode. On repère chaque case par son indice, à partir de 0.

1. `T[0]` vaut 4, `T[4]` vaut 5, `longueur(T)` vaut 5.
2. `T[1] + T[3]` vaut 9 + 7 = 16. On range 16 à l'indice 2 : `[4, 9, 16, 7, 5]`.
3. `[40, 9, 16, 7, 5]`.
4. L'indice 5 n'existe pas : les indices vont de 0 à 4. C'est une erreur. Pour ajouter à la fin, on écrit `ajouter(T, 1)`.

PYTHON

```
T = [4, 9, 2, 7, 5]
print(T[0], T[4], len(T))
T[2] = T[1] + T[3]
print(T)
T[0] = T[0] * 10
print(T)
```

CE QUI S'AFFICHE

```
4 5 5
[4, 9, 16, 7, 5]
[40, 9, 16, 7, 5]
```

Exercice 5.1.2 — La trace d'un parcours ★☆☆

Le tableau `stock` donne le nombre de câbles dans chacun des 4 tiroirs d'une armoire.

PSEUDO-CODE

```
stock ← [12, 0, 7, 3]
total ← 0
Pour i allant de 0 à longueur(stock) - 1 Faire
    total ← total + stock[i]
Fin de Pour
Écrire total
```

1. Combien de tours fait la boucle ?
2. Faites la trace avec les colonnes `i`, `stock[i]` et `total`.
3. Qu'affiche l'algorithme ?

CORRIGÉ

La méthode. On lit chaque case dans l'ordre des indices, et on l'ajoute à l'accumulateur.

1. 4 tours : `i` va de 0 à 3.
2. La trace :

Étape	i	stock[i]	total
départ	–	–	0
tour 1	0	12	12
tour 2	1	0	12
tour 3	2	7	19
tour 4	3	3	22

1. L'algorithme affiche 22.

PYTHON

```python
stock = [12, 0, 7, 3]
total = 0
for i in range(len(stock)):
    total = total + stock[i]
print(total)
```

CE QUI S'AFFICHE

```
22
```

Exercice 5.1.3 — Copie ou deuxième nom ? ★☆☆

Lisez ce programme sans l'exécuter.

PYTHON

```python
a = [5, 6, 7]
b = a
c = a.copy()
b[1] = 0
print(a, b, c)
```

1. Après `b = a`, combien de tableaux existent en mémoire ?
2. Que contient `a` à la fin ?
3. Que contient `c` à la fin ?
4. Écrivez en pseudo-code une copie de `a` élément par élément, sans `.copy()`.

CORRIGÉ

La méthode. `b = a` donne un deuxième nom au même tableau. `.copy()` fabrique un nouveau tableau.

1. Un seul tableau, avec deux noms : `a` et `b`. Puis `c` en crée un deuxième.
2. `a` contient `[5, 0, 7]` : modifier `b` a modifié `a`.
3. `c` contient `[5, 6, 7]` : c'est une vraie copie, faite avant la modification.
4. On part d'un tableau vide, et on ajoute chaque valeur :

PSEUDO-CODE

```
copie ← []
Pour chaque x dans a Faire
    ajouter(copie, x)
Fin de Pour
```

PYTHON

```python
a = [5, 6, 7]
b = a
c = a.copy()
b[1] = 0
print(a, b, c)
```

CE QUI S'AFFICHE

```
[5, 0, 7] [5, 0, 7] [5, 6, 7]
```

★★☆ **Niveau 2 — S'entraîner**

Exercice 5.1.4 — Compter les notes suffisantes ★★☆ Sur machine

Un professeur veut savoir combien d'élèves ont au moins 10.

1. Écrivez l'algorithme en pseudo-code, avec un compteur `nb`.
2. Complétez le programme Python et exécutez-le.
3. Modifiez-le pour afficher aussi le nombre de notes en dessous de 10, sans écrire une deuxième boucle.

PYTHON

```python
notes = [12, 8, 15, 9, 10, 17, 6]
nb = 0
# à compléter : la boucle
print("Au moins 10 :", nb)
```

CORRIGÉ

La méthode. Un compteur qui part de 0. Dans la boucle, un Si ajoute 1 quand la note convient.

PSEUDO-CODE

```
nb ← 0
Pour i allant de 0 à longueur(notes) - 1 Faire
    Si notes[i] ≥ 10 Alors
        nb ← nb + 1
    Fin de Si
Fin de Pour
Écrire "Au moins 10 :", nb
```

Pour la question 3, on n'a pas besoin d'une autre boucle. Les autres notes sont toutes celles qui restent : `longueur(notes) - nb`.

PYTHON

```python
notes = [12, 8, 15, 9, 10, 17, 6]
nb = 0
for i in range(len(notes)):
    if notes[i] >= 10:
        nb = nb + 1
print("Au moins 10 :", nb)
print("En dessous :", len(notes) - nb)
```

CE QUI S'AFFICHE

```
Au moins 10 : 4
En dessous : 3
```

Attention

Avec `notes[i] > 10`, la note 10 ne serait pas comptée. « Au moins 10 » veut dire `≥ 10`.

Exercice 5.1.5 — La température la plus basse ★★☆

Un capteur relève la température d'une salle serveur pendant 5 nuits : `temp ← [-2, 3, -5, 1, -4]`.

1. On veut la plus petite valeur. Quelle valeur de départ donner à `mini` ?
2. Écrivez l'algorithme en pseudo-code.
3. Faites la trace. Que vaut `mini` à la fin ?
4. Un camarade part de `mini ← 0`. Quel résultat obtient-il avec le tableau `[4, 7, 2]` ? Pourquoi est-ce faux ?

CORRIGÉ

La méthode. C'est la méthode du maximum, retournée : on part du premier élément, et on garde une valeur seulement si elle est plus petite.

1. `mini ← temp[0]`, soit -2.
2. L'algorithme :

PSEUDO-CODE

```
temp ← [-2, 3, -5, 1, -4]
mini ← temp[0]
Pour i allant de 1 à longueur(temp) - 1 Faire
    Si temp[i] < mini Alors
        mini ← temp[i]
    Fin de Si
Fin de Pour
Écrire mini
```

1. La trace :

i	temp[i]	temp[i] < mini ?	mini
départ	–	–	-2
1	3	Faux	-2
2	-5	Vrai	-5
3	1	Faux	-5
4	-4	Faux	-5

`mini` vaut -5.

1. Il obtient 0 : aucune valeur de `[4, 7, 2]` n'est plus petite que 0, donc `mini` reste à 0. Or 0 n'est même pas dans le tableau : la bonne réponse est 2. On part toujours d'un vrai élément du tableau.

PYTHON

```python
temp = [-2, 3, -5, 1, -4]
mini = temp[0]
for i in range(1, len(temp)):
    if temp[i] < mini:
        mini = temp[i]
print(mini)
```

CE QUI S'AFFICHE

```
-5
```

Exercice 5.1.6 — Construire un tableau ★★☆ Sur machine

On veut le tableau des carrés des entiers de 1 à 6 : `[1, 4, 9, 16, 25, 36]`.

1. On part d'un tableau vide `carres ← []`. Quelle instruction ajoute une valeur à la fin ?
2. Écrivez l'algorithme avec une boucle Pour.
3. Complétez le programme Python. Affichez ensuite la longueur du tableau.

PYTHON

```
carres = []
# à compléter
print(carres)
```

CORRIGÉ

La méthode. On part d'un tableau vide. À chaque tour, on calcule une valeur et on l'ajoute à la fin.

1. `ajouter(carres, valeur)`. En Python : `carres.append(valeur)`.
2. L'algorithme :

PSEUDO-CODE

```
carres ← []
Pour i allant de 1 à 6 Faire
    ajouter(carres, i * i)
Fin de Pour
Écrire carres
```

PYTHON

```
carres = []
for i in range(1, 7):
    carres.append(i * i)
print(carres)
print(len(carres))
```

CE QUI S'AFFICHE

```
[1, 4, 9, 16, 25, 36]
6
```

Attention

`carres[i] = i * i` ne marche pas sur un tableau vide : la case d'indice `i` n'existe pas encore. Pour un tableau qui grandit, on ajoute à la fin.

★★★ Niveau 3 — Aller plus loin

Exercice 5.1.7 — Trier un tableau ★★★

On veut ranger le tableau `T ← [5, 2, 8, 1, 4]` dans l'ordre croissant. On utilise le **tri par sélection** :

- au tour 0, on cherche l'indice du plus petit élément de tout le tableau, et on l'échange avec la case 0 ;
- au tour 1, on cherche le plus petit parmi les cases 1 à 4, et on l'échange avec la case 1 ;
- et ainsi de suite, jusqu'à l'avant-dernière case.

1. Combien de tours faut-il pour un tableau de 5 éléments ?
2. Pour échanger `T[i]` et `T[imin]`, pourquoi faut-il une variable `temp` ?
3. Complétez l'algorithme.
4. Écrivez le tableau à la fin de chaque tour.

PSEUDO-CODE

```
Pour i allant de 0 à longueur(T) - 2 Faire
    imin ← i
    Pour j allant de i + 1 à longueur(T) - 1 Faire
        Si T[j] < T[imin] Alors
            imin ← . . .
        Fin de Si
    Fin de Pour
    temp ← T[i]
    T[i] ← . . .
    T[imin] ← . . .
Fin de Pour
```

CORRIGÉ

La méthode. Deux boucles imbriquées. La boucle intérieure cherche l'indice du minimum, comme la méthode du maximum, retournée. Puis on échange deux cases, avec une variable de secours.

1. 4 tours : `i` va de `0` à 3. Quand les 4 premières cases sont rangées, la dernière l'est forcément.
2. Sans `temp`, la ligne `T[i] ← T[imin]` effacerait l'ancienne valeur de `T[i]`. On la garde d'abord dans `temp`, comme pour échanger deux variables.
3. L'algorithme complet :

```
PSEUDO-CODE
Pour i allant de 0 à longueur(T) - 2 Faire
    imin ← i
    Pour j allant de i + 1 à longueur(T) - 1 Faire
        Si T[j] < T[imin] Alors
            imin ← j
        Fin de Si
    Fin de Pour
    temp ← T[i]
    T[i] ← T[imin]
    T[imin] ← temp
Fin de Pour
```

1. Le tableau à la fin de chaque tour :

Tour i	Plus petit, à partir de la case i	imin	T à la fin du tour
départ	–	–	`[5, 2, 8, 1, 4]`
0	1	3	`[1, 2, 8, 5, 4]`
1	2	1	`[1, 2, 8, 5, 4]`
2	4	4	`[1, 2, 4, 5, 8]`
3	5	3	`[1, 2, 4, 5, 8]`

Au tour 1 et au tour 3, le plus petit est déjà à sa place : on l'échange avec lui-même, rien ne change.

PYTHON

```python
T = [5, 2, 8, 1, 4]
for i in range(len(T) - 1):
    imin = i
    for j in range(i + 1, len(T)):
        if T[j] < T[imin]:
            imin = j
    temp = T[i]
    T[i] = T[imin]
    T[imin] = temp
    print("Tour", i, ":", T)
```

CE QUI S'AFFICHE

```
Tour 0 : [1, 2, 8, 5, 4]
Tour 1 : [1, 2, 8, 5, 4]
Tour 2 : [1, 2, 4, 5, 8]
Tour 3 : [1, 2, 4, 5, 8]
```

Exercice 5.1.8 — La moyenne sans les valeurs aberrantes ★★★

Sur machine

Un test en ligne enregistre des notes sur 20. Certaines sont des erreurs de saisie : 25, -5, 100.

Ce sont des **valeurs aberrantes** : des valeurs impossibles, qu'on ne doit pas compter. On ne garde que les notes entre 0 et 20 inclus.

Le programme doit :

- calculer la moyenne des notes gardées, arrondie à 2 décimales ;
- afficher -1 si aucune note n'est gardée. Une moyenne ne vaut jamais -1 : on comprend qu'il y a un problème.

1. Écrivez l'algorithme en pseudo-code.
2. Remplissez le tableau des tests, avec le résultat attendu.
3. Complétez le programme Python et vérifiez vos résultats.
4. Modifiez-le pour afficher aussi le nombre de notes gardées.

Tableau	Notes gardées	Somme	Nombre	Résultat attendu
`[10, 12, 18]`				
`[25, -5, 15]`				
`[30, -2, 100]`				
`[8, 9, 12, 21]`				

PYTHON

```python
arr = [10, 12, 18]
s = 0
k = 0
# à compléter
```

CORRIGÉ

La méthode. Une somme et un compteur, avec un Si dans la boucle pour filtrer. La division se fait après la boucle, et seulement si le compteur n'est pas nul.

PSEUDO-CODE

```
s ← 0
k ← 0
Pour i allant de 0 à longueur(arr) - 1 Faire
    Si arr[i] ≥ 0 ET arr[i] ≤ 20 Alors
        s ← s + arr[i]
        k ← k + 1
    Fin de Si
Fin de Pour
Si k = 0 Alors
    Écrire -1
Sinon
    Écrire arrondi(s / k, 2)
Fin de Si
```

Tableau	Notes gardées	Somme	Nombre	Résultat attendu
`[10, 12, 18]`	10, 12, 18	40	3	13,33
`[25, -5, 15]`	15	15	1	15
`[30, -2, 100]`	aucune	0	0	-1
`[8, 9, 12, 21]`	8, 9, 12	29	3	9,67

Pour la question 4, on affiche aussi `k` : dans le pseudo-code, la dernière ligne devient `Écrire arrondi(s / k, 2), k`.

Le programme Python, avec la question 4 :

PYTHON

```python
arr = [10, 12, 18]
s = 0
k = 0
for i in range(len(arr)):
    if arr[i] >= 0 and arr[i] <= 20:
        s = s + arr[i]
        k = k + 1
if k == 0:
    print(-1)
else:
    print(round(s / k, 2), k)
```

CE QUI S'AFFICHE

```
13.33 3
```

Pour tester les autres lignes du tableau des tests, changez la première ligne, puis relancez.

Avec `[25, -5, 15]`, Python écrit `15.0 1` : une division donne toujours un réel. Avec `[30, -2, 100]`, il écrit `-1`. Avec `[8, 9, 12, 21]`, il écrit `9.67 3`.

Attention

Sans le test `k = 0`, le tableau `[30, -2, 100]` provoque une division par zéro : Python s'arrête sur une erreur `ZeroDivisionError`.

Exercice 5.1.9 — Le tableau est-il rangé ? ★★★

Un tableau est **trié dans l'ordre croissant** si chaque élément est plus petit ou égal au suivant. Exemple : `[2, 5, 5, 9]` est trié, `[2, 7, 4]` ne l'est pas.

1. Pour un tableau de 5 éléments, combien de paires de voisins faut-il comparer ?
2. Écrivez un algorithme qui range Vrai ou Faux dans le booléen `trie`. Idée : on part de `trie ← Vrai`, et on passe à Faux dès qu'une paire est mal rangée.
3. Améliorez-le avec une boucle Tant que, qui s'arrête dès qu'une paire est mal rangée.
4. Que donnent vos deux versions sur `[2, 5, 5, 9]`, `[2, 7, 4]` et `[8]` ?
5. Pour `[9, 1, 2, 3, 4]`, combien de comparaisons fait la version Pour ? Et la version Tant que ? Quelle version est la plus efficace ?

CORRIGÉ

La méthode. On compare chaque élément `T[i]` à son voisin de droite `T[i + 1]`. Le dernier élément n'a pas de voisin de droite : `i` s'arrête à `longueur(T) - 2`.

1. 4 paires : (0, 1), (1, 2), (2, 3), (3, 4).
2. Avec une boucle Pour :

PSEUDO-CODE

```
trie ← Vrai
Pour i allant de 0 à longueur(T) - 2 Faire
    Si T[i] > T[i + 1] Alors
        trie ← Faux
    Fin de Si
Fin de Pour
```

1. Avec Tant que, on s'arrête au premier défaut. La garde `i < longueur(T) - 1` vient en premier : avec ET, si elle est fausse, on ne lit pas `T[i + 1]`.

PSEUDO-CODE

```
i ← 0
Tant que i < longueur(T) - 1 ET T[i] ≤ T[i + 1] Faire
    i ← i + 1
Fin de Tant que
trie ← (i = longueur(T) - 1)
```

On sort de la boucle pour deux raisons. Soit on est arrivé au bout : tout est rangé. Soit une paire est mal rangée : `i` n'est pas au bout.

PYTHON

```python
T = [2, 7, 4]
i = 0
while i < len(T) - 1 and T[i] <= T[i + 1]:
    i = i + 1
trie = (i == len(T) - 1)
print(trie)
```

CE QUI S'AFFICHE

```
False
```

1. Les deux versions donnent les mêmes résultats :

- `[2, 5, 5, 9]` : Vrai. Chaque élément est plus petit ou égal au suivant.
- `[2, 7, 4]` : Faux. La paire 7 et 4 est mal rangée.

- `[8]` : Vrai. La boucle Pour va de 0 à -1 : elle ne fait aucun tour, et `trie` reste Vrai. La boucle Tant que s'arrête tout de suite, avec `i` = 0 = `longueur(T) - 1`.

Pour tester un autre tableau en Python, changez la première ligne et relancez.

1. Pour `[9, 1, 2, 3, 4]`, la version Pour compare toujours les 4 paires. La version Tant que s'arrête dès la première paire, 9 et 1 : une seule comparaison. Elle est plus efficace : se demander combien de tours fait un algorithme, c'est s'interroger sur son **efficacité**.

5.2 — Les tableaux à deux dimensions

‹ Revoir le cours 5.2

★☆☆ **Niveau 1 — Appliquer**

Exercice 5.2.1 — Lire une grille ★☆☆

Un tableau donne le nombre de tickets ouverts au service informatique, par jour (une ligne) et par étage (une colonne).

```python
tickets = [[3, 0, 5],
           [1, 4, 2]]
```

1. Combien de lignes et combien de colonnes ?
2. Que vaut `tickets[1][0]` ? Et `tickets[0][2]` ?
3. Quelle case contient 4 ? Donnez la ligne et la colonne.

CORRIGÉ

La méthode. On donne toujours la ligne d'abord, puis la colonne, en comptant à partir de 0.

1. 2 lignes et 3 colonnes.
2. `tickets[1][0]` vaut 1. `tickets[0][2]` vaut 5.
3. Le 4 est à la ligne 1, colonne 1 : `tickets[1][1]`.

PYTHON

```python
tickets = [[3, 0, 5],
           [1, 4, 2]]
print(len(tickets), len(tickets[0]))
print(tickets[1][0], tickets[0][2], tickets[1][1])
```

CE QUI S'AFFICHE

```
2 3
1 5 4
```

Exercice 5.2.2 — Construire une grille ★☆☆

On veut une grille `G` de 2 lignes et 3 colonnes, remplie de `0`.

PSEUDO-CODE

```
G ← []
Pour i allant de 0 à . . . Faire
    ligne ← []
    Pour j allant de 0 à . . . Faire
        ajouter(ligne, 0)
    Fin de Pour
    ajouter(G, . . .)
Fin de Pour
```

1. Complétez les trois pointillés.
2. On exécute ensuite `G[1][0] ← 5`. Écrivez la grille obtenue.
3. Pourquoi `G[2][0] ← 1` est-il une erreur ?

CORRIGÉ

La méthode. La boucle extérieure fabrique les lignes. La boucle intérieure remplit une ligne. On ajoute chaque ligne terminée à la grille.

1. `i` va de `0` à 1 : 2 lignes. `j` va de `0` à 2 : 3 colonnes. On ajoute `ligne` à `G`.

PSEUDO-CODE

```
G ← []
Pour i allant de 0 à 1 Faire
    ligne ← []
    Pour j allant de 0 à 2 Faire
        ajouter(ligne, 0)
    Fin de Pour
    ajouter(G, ligne)
Fin de Pour
```

1. La ligne 1, case 0 devient 5 : `[[0, 0, 0], [5, 0, 0]]`.
2. Il n'y a que 2 lignes, d'indices 0 et 1. La ligne 2 n'existe pas.

PYTHON

```python
G = []
for i in range(2):
    ligne = []
    for j in range(3):
        ligne.append(0)
    G.append(ligne)
G[1][0] = 5
print(G)
```

CE QUI S'AFFICHE

```
[[0, 0, 0], [5, 0, 0]]
```

Exercice 5.2.3 — Un tableau de stagiaires ★☆☆

Chaque ligne décrit un stagiaire : son nom, son âge, et s'il est présent aujourd'hui.

PYTHON

```python
stagiaires = [["Sarah", 22, True],
              ["Tom", 19, False],
              ["Inès", 20, True]]
```

1. Quel est le type de chaque colonne ?
2. Que vaut `stagiaires[2][0]` ? Et `stagiaires[1][2]` ?
3. Quelle expression donne l'âge de Sarah ?

CORRIGÉ

La méthode. Dans ce tableau, chaque colonne a son type. Les lignes ont toutes la même forme.

1. Colonne 0 : une chaîne. Colonne 1 : un entier. Colonne 2 : un booléen.
2. `stagiaires[2][0]` vaut `"Inès"`. `stagiaires[1][2]` vaut `False` : Tom est absent.
3. `stagiaires[0][1]`, qui vaut 22.

PYTHON

```
stagiaires = [["Sarah", 22, True],
              ["Tom", 19, False],
              ["Inès", 20, True]]
print(stagiaires[2][0], stagiaires[1][2], stagiaires[0][1])
```

CE QUI S'AFFICHE

```
Inès False 22
```

★★☆ **Niveau 2 — S'entraîner**

Exercice 5.2.4 — Les ventes par magasin ★★☆

Une chaîne a 3 magasins. Le tableau `ventes` donne le nombre d'ordinateurs vendus par magasin (une ligne) et par trimestre (une colonne).

PYTHON

```
ventes = [[120, 95, 130, 110],
          [80, 105, 90, 100],
          [150, 140, 160, 155]]
```

1. Que vaut `ventes[1][2]` ? Que représente-t-il ?
2. Écrivez, en pseudo-code, une boucle qui calcule le total du magasin d'indice 0, le premier, sur l'année. Quel indice varie ?
3. Écrivez, en pseudo-code, une boucle qui calcule le total du trimestre d'indice 2, le troisième, tous magasins. Quel indice varie ?
4. Écrivez, en pseudo-code, deux boucles imbriquées qui calculent le total général.

CORRIGÉ

La méthode. Une ligne : on fixe la ligne et on fait varier la colonne. Une colonne : l'inverse. Tout le tableau : deux boucles.

1. `ventes[1][2]` vaut 90 : les ventes du magasin d'indice 1 au trimestre d'indice 2.
2. On fait varier `j`, la colonne, de 0 à 3 : `s ← s + ventes[0][j]`. Le total vaut 455.
3. On fait varier `i`, la ligne, de 0 à 2 : `s ← s + ventes[i][2]`. Le total vaut 380.
4. Le total général :

PSEUDO-CODE

```
total ← 0
Pour i allant de 0 à 2 Faire
    Pour j allant de 0 à 3 Faire
        total ← total + ventes[i][j]
    Fin de Pour
Fin de Pour
```

PYTHON

```
ventes = [[120, 95, 130, 110],
          [80, 105, 90, 100],
          [150, 140, 160, 155]]
print(ventes[1][2])
s0 = 0
for j in range(4):
    s0 = s0 + ventes[0][j]
t2 = 0
for i in range(3):
    t2 = t2 + ventes[i][2]
total = 0
for i in range(3):
    for j in range(4):
        total = total + ventes[i][j]
print(s0, t2, total)
```

CE QUI S'AFFICHE

```
90
455 380 1435
```

Exercice 5.2.5 — La moyenne de chaque stagiaire ★★☆ Sur machine

Le tableau `notes` a une ligne par stagiaire et une colonne par module.

PYTHON

```python
notes = [[12, 15, 9, 18],
         [8, 11, 14, 10],
         [16, 13, 17, 15]]
for i in range(3):
    s = 0
    # à compléter : additionner la ligne i
    print("Stagiaire", i, ":", s / 4)
```

1. Pour la moyenne du stagiaire `i`, quel indice varie dans la boucle intérieure ?
2. Complétez le programme et exécutez-le.
3. Que se passe-t-il si on déplace `s = 0` avant la boucle sur `i` ?

CORRIGÉ

La méthode. Deux boucles. La boucle extérieure choisit le stagiaire, la boucle intérieure additionne sa ligne.

La somme repart de 0 pour chaque stagiaire.

1. L'indice de colonne `j`, de 0 à 3. La ligne `i` reste fixe.
2. Le programme complet :

PYTHON

```python
notes = [[12, 15, 9, 18],
         [8, 11, 14, 10],
         [16, 13, 17, 15]]
for i in range(3):
    s = 0
    for j in range(4):
        s = s + notes[i][j]
    print("Stagiaire", i, ":", s / 4)
```

CE QUI S'AFFICHE

```
Stagiaire 0 : 13.5
Stagiaire 1 : 10.75
Stagiaire 2 : 15.25
```

1. La somme ne repart plus de 0. Le stagiaire 1 reçoit aussi les notes du stagiaire 0 : seule la première moyenne est juste.

Exercice 5.2.6 — Qui est présent ? ★★☆ Sur machine

On reprend le tableau des stagiaires : nom, âge, présent.

PYTHON

```python
stagiaires = [["Sarah", 22, True],
              ["Tom", 19, False],
              ["Inès", 20, True],
              ["Yanis", 21, True]]
nb = 0
for ligne in stagiaires:
    # à compléter : remplacez la ligne suivante par votre code
    nb = nb + 0
print("Présents :", nb)
```

1. Dans la boucle, quelle case dit si le stagiaire est présent ?
2. Complétez la boucle : affichez le nom de chaque présent, et comptez-les dans `nb`.
3. Modifiez le programme pour afficher aussi l'âge moyen des présents.

CORRIGÉ

La méthode. `for ligne in stagiaires` donne chaque ligne à son tour. `ligne[2]` est le booléen « présent ». On garde un compteur, et une somme des âges pour la question 3.

1. `ligne[2]`.
2. et 3. Le programme :

PYTHON

```python
stagiaires = [["Sarah", 22, True],
              ["Tom", 19, False],
              ["Inès", 20, True],
              ["Yanis", 21, True]]
nb = 0
somme_ages = 0
for ligne in stagiaires:
    if ligne[2]:
        print(ligne[0])
        nb = nb + 1
        somme_ages = somme_ages + ligne[1]
print("Présents :", nb)
print("Âge moyen :", somme_ages / nb)
```

CE QUI S'AFFICHE

```
Sarah
Inès
Yanis
Présents : 3
Âge moyen : 21.0
```

`if ligne[2]:` suffit : `ligne[2]` est déjà un booléen. `if ligne[2] == True:` marche aussi.

★★★ Niveau 3 — Aller plus loin

Exercice 5.2.7 — Le serveur le plus relié ★★★

Trois serveurs sont numérotés 0, 1 et 2. Dans la grille `L`, `L[i][j]` vaut 1 si le serveur `i` envoie des données au serveur `j`, et 0 sinon.

PYTHON

```
L = [[0, 1, 1],
     [0, 0, 1],
     [1, 0, 0]]
```

Le **degré sortant** d'un serveur est le nombre de serveurs qui reçoivent ses données.

1. Lisez la grille : à quels serveurs le serveur 0 envoie-t-il des données ? Quel est son degré sortant ?
2. Écrivez un algorithme qui calcule le degré sortant du serveur `s`. Quel indice varie ?
3. Écrivez un algorithme qui trouve le serveur qui a le plus grand degré sortant.

CORRIGÉ

La méthode. Le degré sortant du serveur `s` est la somme de sa ligne. Pour trouver le plus grand, on calcule le degré de chaque ligne, et on applique la méthode du maximum.

1. La ligne 0 est `[0, 1, 1]` : le serveur 0 envoie aux serveurs 1 et 2. Son degré sortant vaut 2.
2. On fixe la ligne `s` et on fait varier la colonne `j` :

PSEUDO-CODE

```
deg ← 0
Pour j allant de 0 à 2 Faire
    Si L[s][j] = 1 Alors
        deg ← deg + 1
    Fin de Si
Fin de Pour
```

1. Deux boucles imbriquées, et un maximum :

PSEUDO-CODE

```
meilleur ← 0
deg_max ← -1
Pour i allant de 0 à 2 Faire
    deg ← 0
    Pour j allant de 0 à 2 Faire
        deg ← deg + L[i][j]
    Fin de Pour
    Si deg > deg_max Alors
        deg_max ← deg
        meilleur ← i
    Fin de Si
Fin de Pour
Écrire meilleur, deg_max
```

PYTHON

```
L = [[0, 1, 1],
     [0, 0, 1],
     [1, 0, 0]]
meilleur = 0
deg_max = -1
for i in range(3):
    deg = 0
    for j in range(3):
        deg = deg + L[i][j]
    print("Serveur", i, ": degré", deg)
    if deg > deg_max:
        deg_max = deg
        meilleur = i
print("Le plus relié :", meilleur)
```

CE QUI S'AFFICHE

```
Serveur 0 : degré 2
Serveur 1 : degré 1
Serveur 2 : degré 1
Le plus relié : 0
```

Le sujet du BTS 2023 pose la même question, avec des sommets numérotés de 1 à 3. Voyez l'onglet Annales.

Exercice 5.2.8 — Les places libres ★★★ Sur machine

Le plan d'une salle de formation a 3 rangs de 4 places. Une place vaut 1 si elle est prise, 0 si elle est libre.

PYTHON

```python
salle = [[1, 0, 1, 1],
         [0, 0, 1, 0],
         [1, 1, 1, 1]]
# à compléter
```

1. Écrivez un programme qui affiche chaque place libre sous la forme `rang 1 place 3`, en indices.
2. Ajoutez le nombre total de places libres.
3. Ajoutez, pour chaque rang, le nombre de places libres. Où remettre le compteur du rang à `0` ?

CORRIGÉ

La méthode. Deux boucles imbriquées parcourent tout le plan. Un Si repère les `0`. Le compteur du rang repart de `0` à chaque nouveau rang, donc juste avant la boucle sur les places.

PSEUDO-CODE

```
total ← 0
Pour i allant de 0 à 2 Faire
    libres_rang ← 0
    Pour j allant de 0 à 3 Faire
        Si salle[i][j] = 0 Alors
            Écrire "rang", i, "place", j
            libres_rang ← libres_rang + 1
            total ← total + 1
        Fin de Si
    Fin de Pour
    Écrire "Rang", i, ":", libres_rang, "libre(s)"
Fin de Pour
Écrire "Total :", total
```

PYTHON

```python
salle = [[1, 0, 1, 1],
         [0, 0, 1, 0],
         [1, 1, 1, 1]]
total = 0
for i in range(3):
    libres_rang = 0
    for j in range(4):
        if salle[i][j] == 0:
            print("rang", i, "place", j)
            libres_rang = libres_rang + 1
            total = total + 1
    print("Rang", i, ":", libres_rang, "libre(s)")
print("Total :", total)
```

CE QUI S'AFFICHE

```
rang 0 place 1
Rang 0 : 1 libre(s)
rang 1 place 0
rang 1 place 1
rang 1 place 3
Rang 1 : 3 libre(s)
Rang 2 : 0 libre(s)
Total : 4
```

`print` met un espace entre chaque morceau : on obtient `rang 0 place 1`.

Exercice 5.2.9 — Qui peut joindre qui ? ★★★ Sur machine

Quatre serveurs sont numérotés de 0 à 3. Certains envoient des données à d'autres. On dessine chaque envoi par une flèche, appelée un **arc** : un arc de `i` vers `j` veut dire « le serveur `i` envoie au serveur `j` ».

On range ces arcs dans une grille `L`, appelée **matrice d'adjacence**. `L[i][j]` vaut 1 s'il y a un arc de `i` vers `j`, et 0 sinon.

PYTHON

```python
L = [[0, 1, 0, 0],
     [0, 0, 1, 1],
     [1, 0, 0, 0],
     [0, 0, 1, 0]]
```

Les **voisins** d'un serveur sont les serveurs qui reçoivent directement ses données.

1. Quels sont les voisins du serveur 1 ? Lisez sa ligne.

2. Écrivez un algorithme qui affiche les voisins du serveur `s`.
3. Une donnée part du serveur 0 et fait exactement deux arcs. Sur quels serveurs peut-elle arriver ? Trouvez-les à la main.
4. Écrivez un algorithme qui affiche les serveurs atteints en deux arcs depuis `s`. Idée : pour chaque voisin `v` de `s`, on affiche les voisins de `v`.
5. Programmez-le en Python, et testez-le pour `s` = 0 et `s` = 1.

CORRIGÉ

La méthode. Les voisins de `s` se lisent sur la ligne `s` : ce sont les colonnes `j` où `L[s][j]` vaut 1. Pour deux arcs, on fait la même chose depuis chaque voisin : deux boucles imbriquées.

1. La ligne 1 est `[0, 0, 1, 1]` : les voisins du serveur 1 sont les serveurs 2 et 3.
2. On fixe la ligne `s` et on fait varier la colonne `j` :

```
PSEUDO-CODE
Pour j allant de 0 à 3 Faire
    Si L[s][j] = 1 Alors
        Écrire j
    Fin de Si
Fin de Pour
```

1. Le serveur 0 envoie au serveur 1. Le serveur 1 envoie aux serveurs 2 et 3. En deux arcs, on arrive sur 2 ou sur 3.
2. Pour chaque voisin `v` de `s`, on parcourt la ligne `v` :

```
PSEUDO-CODE
Pour v allant de 0 à 3 Faire
    Si L[s][v] = 1 Alors
        Pour j allant de 0 à 3 Faire
            Si L[v][j] = 1 Alors
                Écrire j
            Fin de Si
        Fin de Pour
    Fin de Si
Fin de Pour
```

1. Le programme :

PYTHON

```python
L = [[0, 1, 0, 0],
     [0, 0, 1, 1],
     [1, 0, 0, 0],
     [0, 0, 1, 0]]
for s in range(2):
    print("Depuis", s, ":")
    for v in range(4):
        if L[s][v] == 1:
            for j in range(4):
                if L[v][j] == 1:
                    print("  serveur", j, "par le serveur", v)
```

CE QUI S'AFFICHE

```
Depuis 0 :
  serveur 2 par le serveur 1
  serveur 3 par le serveur 1
Depuis 1 :
  serveur 0 par le serveur 2
  serveur 2 par le serveur 3
```

Depuis le serveur 1, on atteint le serveur 0 en passant par 2, et le serveur 2 en passant par 3. Cet algorithme est un premier **parcours de graphe**.

5.3 — Les fonctions et les procédures

‹ Revoir le cours 5.3

★☆☆ **Niveau 1 — Appliquer**

Exercice 5.3.1 — Que renvoie la fonction ? ★☆☆

PSEUDO-CODE

```
Fonction carre(n)
    Renvoyer n * n
Fin de Fonction
```

1. Quel est le paramètre de la fonction ?
2. Que renvoie `carre(4)` ?
3. Que vaut `carre(3) + carre(4)` ?
4. Que vaut `carre(carre(2))` ?

CORRIGÉ

La méthode. On remplace le paramètre par la valeur donnée à l'appel, puis on calcule ce qui suit `Renvoyer`.

1. Le paramètre est `n`.
2. 16.
3. 9 + 16 = 25.
4. On calcule d'abord l'appel intérieur : `carre(2)` vaut 4. Puis `carre(4)` vaut 16.

PYTHON

```python
def carre(n):
    return n * n

print(carre(4), carre(3) + carre(4), carre(carre(2)))
```

CE QUI S'AFFICHE

```
16 25 16
```

Exercice 5.3.2 — Fonction ou procédure ? ★☆☆

Pour chaque besoin, dites s'il faut une fonction ou une procédure. Justifiez en une phrase.

1. Calculer la TVA d'un prix, pour l'utiliser ensuite dans un total.
2. Afficher un menu de 4 choix à l'écran.
3. Dire si un mot de passe est assez long, Vrai ou Faux.
4. Afficher le bulletin d'un stagiaire, ligne par ligne.

CORRIGÉ

La méthode. On se demande : est-ce qu'on a besoin d'un résultat pour la suite ? Si oui, c'est une fonction, qui renvoie. Si on fait seulement une action, c'est une procédure.

1. Fonction : on renvoie la TVA pour la réutiliser dans un calcul.
2. Procédure : on affiche, on ne renvoie rien.
3. Fonction : elle renvoie un booléen.
4. Procédure : elle affiche le bulletin.

Exercice 5.3.3 — Le plus grand de deux nombres ★☆☆

Complétez la fonction `plus_grand(a, b)`. Elle renvoie le plus grand des deux nombres.

PSEUDO-CODE

```
Fonction plus_grand(a, b)
    Si a > b Alors
        Renvoyer . . .
    Sinon
        Renvoyer . . .
    Fin de Si
Fin de Fonction
```

Que renvoient `plus_grand(3, 8)` et `plus_grand(5, 5)` ?

CORRIGÉ

La méthode. Deux cas, deux `Renvoyer`. Un seul est exécuté : `Renvoyer` termine la fonction.

PSEUDO-CODE

```
Fonction plus_grand(a, b)
    Si a > b Alors
        Renvoyer a
    Sinon
        Renvoyer b
    Fin de Si
Fin de Fonction
```

`plus_grand(3, 8)` renvoie 8. `plus_grand(5, 5)` renvoie 5 : `5 > 5` est faux, on renvoie `b`, qui vaut aussi 5.

PYTHON

```python
def plus_grand(a, b):
    if a > b:
        return a
    else:
        return b

print(plus_grand(3, 8), plus_grand(5, 5))
```

CE QUI S'AFFICHE

```
8 5
```

★★☆ **Niveau 2 — S'entraîner**

Exercice 5.3.4 — Le plus petit et le plus grand ★★☆ Sur machine

Une application de suivi relève les temps de réponse d'un serveur, en millisecondes.

1. Écrivez la fonction `mini_maxi(T)` en pseudo-code. Elle renvoie deux résultats : la plus petite et la plus grande valeur du tableau.
2. Écrivez l'appel qui récupère les deux résultats dans `a` et `b`, en pseudo-code.
3. Complétez la version Python et testez-la sur `[48, 120, 35, 90]`.

PYTHON

```python
def mini_maxi(T):
    mini = T[0]
    maxi = T[0]
    # à compléter
    return mini, maxi

a, b = mini_maxi([48, 120, 35, 90])
print("Plus petit :", a, "Plus grand :", b)
```

CORRIGÉ

La méthode. La méthode du maximum et celle du minimum, dans la même boucle. On part du premier élément pour les deux.

PSEUDO-CODE

```
Fonction mini_maxi(T)
    mini ← T[0]
    maxi ← T[0]
    Pour i allant de 1 à longueur(T) - 1 Faire
        Si T[i] < mini Alors
            mini ← T[i]
        Fin de Si
        Si T[i] > maxi Alors
            maxi ← T[i]
        Fin de Si
    Fin de Pour
    Renvoyer (mini, maxi)
Fin de Fonction
```

1. Deux variables à gauche, dans l'ordre du `Renvoyer` : `(a, b) ← mini_maxi([48, 120, 35, 90])`.

PYTHON

```python
def mini_maxi(T):
    mini = T[0]
    maxi = T[0]
    for i in range(1, len(T)):
        if T[i] < mini:
            mini = T[i]
        if T[i] > maxi:
            maxi = T[i]
    return mini, maxi

a, b = mini_maxi([48, 120, 35, 90])
print("Plus petit :", a, "Plus grand :", b)
print(mini_maxi([48, 120, 35, 90]))
```

CE QUI S'AFFICHE

```
Plus petit : 35 Plus grand : 120
(35, 120)
```

La dernière ligne montre ce qui se passe sans les deux variables : Python affiche les deux valeurs entre parenthèses.

> **Attention**
>
> Les deux tests sont deux `Si` séparés, pas un `Si … Sinon`. Une même valeur peut changer `mini`, puis une autre changer `maxi`, dans des tours différents.

Exercice 5.3.5 — Combien de fois ? ★★☆

Un journal de connexion liste les codes d'erreur d'une journée : `codes ← [404, 200, 404, 500, 404, 200]`.

1. Écrivez la fonction `compter(T, x)`. Elle renvoie le nombre de fois où la valeur `x` apparaît dans le tableau `T`.
2. Que renvoient `compter(codes, 404)` et `compter(codes, 301)` ?
3. Écrivez la ligne qui affiche « Erreurs 404 : » suivi du résultat.

CORRIGÉ

La méthode. Un compteur qui part de 0, et un Si dans la boucle. Les deux paramètres entrent, le compteur sort par `Renvoyer`.

PSEUDO-CODE

```
Fonction compter(T, x)
    nb ← 0
    Pour i allant de 0 à longueur(T) - 1 Faire
        Si T[i] = x Alors
            nb ← nb + 1
        Fin de Si
    Fin de Pour
    Renvoyer nb
Fin de Fonction
```

1. `compter(codes, 404)` renvoie 3. `compter(codes, 301)` renvoie 0.
2. `Écrire "Erreurs 404 :", compter(codes, 404)`.

PYTHON

```python
def compter(T, x):
    nb = 0
    for i in range(len(T)):
        if T[i] == x:
            nb = nb + 1
    return nb

codes = [404, 200, 404, 500, 404, 200]
print("Erreurs 404 :", compter(codes, 404))
print(compter(codes, 301))
```

CE QUI S'AFFICHE

```
Erreurs 404 : 3
0
```

Exercice 5.3.6 — La table de multiplication ★★☆ Sur machine

1. Écrivez la procédure `afficher_table(n)`. Elle affiche la table de `n`, de 1 à 10, sous la forme `7 x 3 = 21`.
2. Appelez-la pour 7, puis pour 9.
3. Un camarade écrit une **fonction** qui renvoie une seule ligne de la table. Pourquoi une procédure convient-elle mieux ici ?

PYTHON

```python
def afficher_table(n):
    # à compléter : remplacez la ligne suivante par votre boucle
    print("table de", n)

afficher_table(7)
```

CORRIGÉ

La méthode. Une boucle Pour de 1 à 10 dans la procédure. Le paramètre `n` change d'un appel à l'autre.

PSEUDO-CODE

```
Procédure afficher_table(n)
    Pour i allant de 1 à 10 Faire
        Écrire n, "x", i, "=", n * i
    Fin de Pour
Fin de Procédure
```

PYTHON

```python
def afficher_table(n):
    for i in range(1, 11):
        print(n, "x", i, "=", n * i)

afficher_table(7)
```

CE QUI S'AFFICHE

```
7 x 1 = 7
7 x 2 = 14
7 x 3 = 21
7 x 4 = 28
7 x 5 = 35
7 x 6 = 42
7 x 7 = 49
7 x 8 = 56
7 x 9 = 63
7 x 10 = 70
```

Pour la table de 9, on écrit seulement `afficher_table(9)`.

1. Ce qu'on demande est une **action** : afficher. On ne réutilise aucun résultat dans un calcul. Une procédure fait exactement ça. Une fonction servirait si on voulait garder la table pour la réutiliser ensuite.

★★★ **Niveau 3 — Aller plus loin**

Exercice 5.3.7 — Le bulletin ★★★

On veut afficher la mention de chaque stagiaire.

- La fonction `mention(moy)` renvoie une mention selon la moyenne :
  - `"Très bien"` à partir de 16 ;
  - `"Bien"` à partir de 14 ;
  - `"Assez bien"` à partir de 12 ;
  - `"Passable"` à partir de 10 ;
  - `"Insuffisant"` en dessous.
- La procédure `bulletin(noms, moyennes)` affiche, pour chaque stagiaire, son nom, sa moyenne et sa mention. Elle appelle `mention`.

1. Écrivez la fonction `mention`. Dans quel ordre faut-il tester les seuils ?
2. Écrivez la procédure `bulletin`. Les deux tableaux ont la même longueur : la case `i` de `noms` va avec la case `i` de `moyennes`.
3. Que s'affiche-t-il pour `noms ← ["Léa", "Hugo", "Sarah"]` et `moyennes ← [14.5, 9, 16]` ?
4. Écrivez la version Python **avec un en-tête et des commentaires** : pour chaque sous-programme, son nom, son rôle, ses entrées et sa sortie.

CORRIGÉ

La méthode. La fonction fait un seul calcul et renvoie un texte. La procédure parcourt les tableaux et appelle la fonction à chaque tour. On teste le seuil le plus haut d'abord.

1. Du plus haut au plus bas, sinon 17 donnerait « Passable » :

PSEUDO-CODE

```
Fonction mention(moy)
    Si moy ≥ 16 Alors
        Renvoyer "Très bien"
    Sinon si moy ≥ 14 Alors
        Renvoyer "Bien"
    Sinon si moy ≥ 12 Alors
        Renvoyer "Assez bien"
    Sinon si moy ≥ 10 Alors
        Renvoyer "Passable"
    Sinon
        Renvoyer "Insuffisant"
    Fin de Si
Fin de Fonction
```

1. La procédure :

PSEUDO-CODE

```
Procédure bulletin(noms, moyennes)
    Pour i allant de 0 à longueur(noms) - 1 Faire
        Écrire noms[i], moyennes[i], mention(moyennes[i])
    Fin de Pour
Fin de Procédure
```

1. La version Python, avec en-tête et commentaires :

PYTHON

```python
# Nom : mention
# Rôle : donner la mention qui correspond à une moyenne
# Entrée : moy, un réel entre 0 et 20
# Sortie : la mention, une chaîne
def mention(moy):
    # du seuil le plus haut au plus bas
    if moy >= 16:
        return "Très bien"
    elif moy >= 14:
        return "Bien"
    elif moy >= 12:
        return "Assez bien"
    elif moy >= 10:
        return "Passable"
    else:
        return "Insuffisant"

# Nom : bulletin
# Rôle : afficher le nom, la moyenne et la mention de chaque stagiaire
# Entrées : noms et moyennes, deux tableaux de même longueur
# Sortie : rien, la procédure affiche
def bulletin(noms, moyennes):
    # la case i de noms va avec la case i de moyennes
    for i in range(len(noms)):
        print(noms[i], moyennes[i], mention(moyennes[i]))

bulletin(["Léa", "Hugo", "Sarah"], [14.5, 9, 16])
```

CE QUI S'AFFICHE

```
Léa 14.5 Bien
Hugo 9 Insuffisant
Sarah 16 Très bien
```

Exercice 5.3.8 — Trouvé, et où ? ★★★

On veut la fonction `chercher(T, x)`. Elle renvoie **deux** valeurs : un booléen qui dit si `x` est dans le tableau `T`, et l'indice de la première case qui contient `x`. Si `x` est absent, l'indice renvoyé vaut `longueur(T)`.

1. Pourquoi une boucle Tant que convient-elle mieux qu'une boucle Pour ?
2. Complétez la fonction.
3. Dans le programme principal, comment récupérer les deux résultats ? Écrivez l'appel pour `T ← [7, 3, 9, 3]` et `x ← 3`.
4. Que renvoie `chercher([7, 3, 9, 3], 5)` ?

PSEUDO-CODE

```
Fonction chercher(T, x)
    k ← 0
    Tant que k < longueur(T) ET . . . Faire
        k ← k + 1
    Fin de Tant que
    trouve ← . . .
    Renvoyer (trouve, k)
Fin de Fonction
```

CORRIGÉ

La méthode. On avance tant qu'on n'est pas au bout et qu'on n'a pas trouvé. À la sortie, on a trouvé si on n'est pas arrivé au bout.

1. On veut s'arrêter dès qu'on trouve `x`. On ne sait pas d'avance combien de tours il faut.
2. La fonction complète :

PSEUDO-CODE

```
Fonction chercher(T, x)
    k ← 0
    Tant que k < longueur(T) ET T[k] ≠ x Faire
        k ← k + 1
    Fin de Tant que
    trouve ← (k < longueur(T))
    Renvoyer (trouve, k)
Fin de Fonction
```

La garde `k < longueur(T)` est écrite en premier. Quand elle est fausse, le ET s'arrête : on ne lit pas `T[k]`, qui n'existe pas.

1. Deux variables à gauche, dans l'ordre du `Renvoyer` : `(trouve, k) ← chercher(T, 3)`. En Python : `trouve, k = chercher(T, 3)`. On obtient Vrai et 1 : le premier 3 est à l'indice 1.
2. `(Faux, 4)` : 5 est absent, et `k` est arrivé à la longueur du tableau.

PYTHON

```python
def chercher(T, x):
    k = 0
    while k < len(T) and T[k] != x:
        k = k + 1
    trouve = k < len(T)
    return trouve, k

T = [7, 3, 9, 3]
trouve, k = chercher(T, 3)
print(trouve, k)
print(chercher(T, 5))
```

CE QUI S'AFFICHE

```
True 1
(False, 4)
```

Exercice 5.3.9 — Convertir des durées ★★★ Sur machine

Un serveur note la durée de chaque sauvegarde en secondes : `[7384, 59, 3600]`.

1. Écrivez la fonction `hms(duree)`. Elle renvoie trois valeurs : les heures, les minutes et les secondes.
2. Écrivez un programme qui affiche chaque durée sous la forme `2 h 3 min 4 s`.
3. Ajoutez le total de toutes les sauvegardes, converti lui aussi.
4. Écrivez la fonction `secondes(h, m, s)`, qui fait le calcul inverse : elle renvoie la durée en secondes. Vérifiez qu'elle redonne 7384 pour 2 h 3 min 4 s.

PYTHON

```python
def hms(duree):
    # à compléter
    return 0, 0, 0

for d in [7384, 59, 3600]:
    h, m, s = hms(d)
    print(h, "h", m, "min", s, "s")
```

CORRIGÉ

La méthode. Dans la fonction : quotient et reste, deux fois. Dans le programme : on récupère les trois valeurs dans trois variables. Le total se calcule en secondes, puis on le convertit avec la même fonction.

PSEUDO-CODE

```
Fonction hms(duree)
    h ← duree div 3600
    reste ← duree mod 3600
    m ← reste div 60
    s ← reste mod 60
    Renvoyer (h, m, s)
Fin de Fonction
```

PYTHON

```python
def hms(duree):
    h = duree // 3600
    reste = duree % 3600
    m = reste // 60
    s = reste % 60
    return h, m, s

durees = [7384, 59, 3600]
total = 0
for d in durees:
    h, m, s = hms(d)
    print(h, "h", m, "min", s, "s")
    total = total + d
h, m, s = hms(total)
print("Total :", h, "h", m, "min", s, "s")

def secondes(h, m, s):
    return h * 3600 + m * 60 + s

print(secondes(2, 3, 4))
```

CE QUI S'AFFICHE

```
2 h 3 min 4 s
0 h 0 min 59 s
1 h 0 min 0 s
Total : 3 h 4 min 3 s
7384
```

La fonction `hms` est écrite une seule fois et sert quatre fois. C'est tout l'intérêt d'une fonction. La fonction `secondes` fait le chemin inverse : 2 × 3600 + 3 × 60 + 4 = 7384.

5.4 — Paramètres et variables locales

‹ Revoir le cours 5.4

★☆☆ **Niveau 1 — Appliquer**

Exercice 5.4.1 — Locale ou globale ? ★☆☆

PYTHON

```python
taux = 20

def prix_ttc(ht):
    tva = ht * taux / 100
    return ht + tva

prix = prix_ttc(50)
print(prix)
```

1. Classez chaque variable : `taux`, `tva`, `ht`, `prix`. Est-elle locale ou globale ?
2. Qu'affiche le programme ?
3. Peut-on écrire `print(tva)` à la dernière ligne ?

CORRIGÉ

La méthode. Une variable créée dans la fonction, ou un paramètre, est locale. Une variable créée dans le programme principal est globale.

1. `taux` et `prix` : globales. `tva` : locale. `ht` : c'est un paramètre, il est local lui aussi.
2. 50 × 20 / 100 = 10, et 50 + 10 = 60. Le programme affiche `60.0`.
3. Non. `tva` n'existe que pendant l'appel : Python s'arrête sur une erreur `NameError`.

PYTHON

```python
taux = 20

def prix_ttc(ht):
    tva = ht * taux / 100
    return ht + tva

prix = prix_ttc(50)
print(prix)
```

CE QUI S'AFFICHE

```
60.0
```

Exercice 5.4.2 — Dans quel sens passent les données ? ★☆☆

Pour chaque fonction ou procédure, dites si le paramètre est en **entrée seule**, en **sortie seule** ou en **entrée et sortie**.

1. `longueur_mot(mot)` : elle renvoie le nombre de caractères du mot.
2. `augmenter(T)` : elle ajoute 1 à chaque case du tableau `T`.
3. `moyenne(T)` : elle renvoie la moyenne des valeurs de `T`.
4. `remplir(T, n)` : elle reçoit un tableau vide `T` et y met n zéros, sans lire ce qu'il contenait. Donnez le sens de `T` et celui de `n`.
5. `ajouter_note(T, n)` : elle ajoute la note `n` à la fin du tableau `T`. Donnez le sens de `T` et celui de `n`.

CORRIGÉ

La méthode. On se pose deux questions. La procédure lit-elle ce qu'elle reçoit ? Le modifie-t-elle ?

- Lu seulement : entrée seule.
- Rempli sans être lu : sortie seule.
- Lu et modifié : entrée et sortie.

1. `mot` : entrée seule. Le résultat sort par `Renvoyer`.
2. `T` : entrée et sortie. On lit chaque case pour lui ajouter 1.
3. `T` : entrée seule. On lit les valeurs, sans les changer.
4. `T` : sortie seule. La procédure ne lit pas `T`, elle le remplit. `n` : entrée seule.
5. `T` : entrée et sortie, car il grandit à partir de ce qu'il contient déjà. `n` : entrée seule.

Exercice 5.4.3 — Deux variables, un seul nom ★☆☆

Lisez ce programme sans l'exécuter.

PYTHON

```python
x = 10

def f():
    x = 5
    return x * 2

y = f()
print(x, y)
```

1. Combien de variables s'appellent `x` ?

2. Que vaut `y` ?
3. Qu'affiche le programme ?

CORRIGÉ

La méthode. Le `x` créé dans la fonction est local. Il n'a rien à voir avec le `x` global, même s'il porte le même nom.

1. Deux : le `x` global, qui vaut 10, et le `x` local à `f`, qui vaut 5.
2. `y` vaut 5 × 2 = 10.
3. Le programme affiche `10 10`. Le `x` global n'a pas changé.

PYTHON

```python
x = 10

def f():
    x = 5
    return x * 2

y = f()
print(x, y)
```

CE QUI S'AFFICHE

```
10 10
```

★★☆ **Niveau 2 — S'entraîner**

Exercice 5.4.4 — Qui modifie quoi ? ★★☆

Lisez ce programme sans l'exécuter.

PYTHON

```python
total = 0

def ajouter(T, x):
    T.append(x)
    total = len(T)
    return total

notes = [12, 15]
n = ajouter(notes, 9)
copie = notes.copy()
copie.append(20)
print(notes, n, total, copie)
```

1. Pour le paramètre `T`, le transfert est-il en entrée seule ou en entrée et sortie ? Et pour `x` ?
2. Que vaut `n` ?
3. Pourquoi la variable `total` affichée vaut-elle encore `0` ?
4. Ajouter `20` à `copie` change-t-il `notes` ?
5. Écrivez ce qu'affiche la dernière ligne, puis vérifiez.

CORRIGÉ

La méthode. On sépare ce qui est créé dans la fonction, local, de ce qui est créé dans le programme principal, global.

1. `T` : **entrée et sortie**. La fonction reçoit le tableau et le modifie avec `append`. `x` : **entrée seule**, il est lu, pas modifié.
2. Après l'ajout, le tableau a 3 éléments : `n` vaut 3.
3. Le `total` de la fonction est une variable **locale**, créée pendant l'appel. Le `total` global n'a jamais été touché : il vaut `0`.
4. Non. `copie` est une vraie copie, faite avec `.copy()`.
5. La sortie :

PYTHON

```python
total = 0

def ajouter(T, x):
    T.append(x)
    total = len(T)
    return total

notes = [12, 15]
n = ajouter(notes, 9)
copie = notes.copy()
copie.append(20)
print(notes, n, total, copie)
```

CE QUI S'AFFICHE

```
[12, 15, 9] 3 0 [12, 15, 9, 20]
```

Attention

Avec `copie = notes`, sans `.copy()`, ajouter 20 à `copie` l'aurait aussi ajouté à `notes`.

Exercice 5.4.5 — Modifier ou rendre une copie ? ★★☆ Sur machine

On veut ajouter 2 points de bonus à chaque note, sans dépasser 20.

- Version A : la procédure `bonus_sur_place(T)` modifie le tableau qu'on lui donne.
- Version B : la fonction `avec_bonus(T)` renvoie un **nouveau** tableau, et laisse `T` intact.

1. Pour chaque version, le paramètre `T` est-il en entrée seule ou en entrée et sortie ?
2. Complétez les deux versions.
3. Testez avec `[12, 19, 8]`. Affichez le tableau de départ après chaque appel.

PYTHON

```python
def bonus_sur_place(T):
    # à compléter : remplacez la ligne suivante par votre boucle
    T[0] = T[0]

def avec_bonus(T):
    R = T.copy()
    # à compléter
    return R

notes = [12, 19, 8]
nouvelles = avec_bonus(notes)
print(notes, nouvelles)
bonus_sur_place(notes)
print(notes)
```

CORRIGÉ

La méthode. Dans la version A, on écrit dans `T`. Dans la version B, on écrit dans une copie `R`, qu'on renvoie. Pour ne pas dépasser 20, un Si : si `note + 2` dépasse 20, on met 20, sinon on met `note + 2`.

1. Version A : entrée et sortie. Version B : entrée seule, le résultat sort par `Renvoyer`.
2. Les deux versions :

PYTHON

```python
def bonus_sur_place(T):
    for i in range(len(T)):
        if T[i] + 2 > 20:
            T[i] = 20
        else:
            T[i] = T[i] + 2

def avec_bonus(T):
    R = T.copy()
    for i in range(len(R)):
        if R[i] + 2 > 20:
            R[i] = 20
        else:
            R[i] = R[i] + 2
    return R

notes = [12, 19, 8]
nouvelles = avec_bonus(notes)
print(notes, nouvelles)
bonus_sur_place(notes)
print(notes)
```

CE QUI S'AFFICHE

```
[12, 19, 8] [14, 20, 10]
[14, 20, 10]
```

En pseudo-code, la ligne de la boucle devient : `Si T[i] + 2 > 20 Alors T[i] ← 20 Sinon T[i] ← T[i] + 2 Fin de Si`.

Attention

Sans `.copy()` dans la version B, `R` serait le même tableau que `T`, et `notes` serait modifié aussi.

Exercice 5.4.6 — Une fonction qui dépend d'une globale ★★☆

Un collègue a écrit ce programme.

PYTHON

```python
remise = 10

def prix_final(prix):
    return prix - prix * remise / 100

print(prix_final(80))
remise = 30
print(prix_final(80))
```

1. Qu'affichent les deux lignes `print` ?
2. Pourquoi le même appel `prix_final(80)` donne-t-il deux résultats différents ?
3. Réécrivez la fonction pour que la remise soit un paramètre. Écrivez les deux appels.

CORRIGÉ

La méthode. La fonction lit une variable globale. Si la globale change, le résultat change, sans que l'appel le montre. On fait entrer la remise par un paramètre.

1. `72.0`, puis `56.0`.
2. La fonction lit `remise`, une variable globale. Entre les deux appels, le programme l'a changée.
3. La remise devient un paramètre : tout ce dont la fonction a besoin entre par ses paramètres.

PYTHON

```python
def prix_final(prix, remise):
    return prix - prix * remise / 100

print(prix_final(80, 10))
print(prix_final(80, 30))
```

CE QUI S'AFFICHE

```
72.0
56.0
```

Maintenant, chaque appel dit lui-même la remise utilisée. On peut tester la fonction seule.

★★★ **Niveau 3 — Aller plus loin**

Exercice 5.4.7 — La trace des appels ★★★

Lisez ce programme sans l'exécuter.

PYTHON

```python
def double(n):
    r = n * 2
    return r

def calcul(a, b):
    r = double(a) + b
    return r

r = 1
x = calcul(3, r)
y = calcul(x, 4)
print(r, x, y)
```

1. Combien de variables `r` différentes existent dans ce programme ?
2. Faites la trace de l'appel `calcul(3, r)` : que valent `a`, `b`, puis le `r` de `calcul` ?
3. Que valent `x` et `y` ?
4. Qu'affiche le programme ?

CORRIGÉ

La méthode. Chaque appel a ses propres variables locales. On suit un appel à la fois, et on remplace chaque paramètre par la valeur reçue.

1. Trois : le `r` global, le `r` de `double` et le `r` de `calcul`.
2. `calcul(3, r)` : `a` vaut 3 et `b` vaut 1, la valeur du `r` global. `double(3)` renvoie 6. Le `r` de `calcul` vaut 6 + 1 = 7.
3. `x` vaut 7. Puis `calcul(7, 4)` : `double(7)` vaut 14, et 14 + 4 = 18. `y` vaut 18.
4. Le `r` global n'a jamais changé : le programme affiche `1 7 18`.

PYTHON

```python
def double(n):
    r = n * 2
    return r

def calcul(a, b):
    r = double(a) + b
    return r

r = 1
x = calcul(3, r)
y = calcul(x, 4)
print(r, x, y)
```

CE QUI S'AFFICHE

```
1 7 18
```

Exercice 5.4.8 — Ramener les notes entre 0 et 20 ★★★ Sur machine

Un tableur a exporté des notes avec des erreurs : `[12, -3, 25, 18]`. On veut la fonction `bornees(T)`.

Elle renvoie un **nouveau** tableau où chaque note négative devient 0 et chaque note au-dessus de 20 devient 20. Le tableau de départ ne doit pas changer.

1. Le paramètre `T` doit-il être en entrée seule ou en entrée et sortie ?
2. Écrivez la fonction en pseudo-code, puis en Python.
3. Affichez `notes` après l'appel. Comment voit-on que le tableau de départ n'a pas changé ?

PYTHON

```python
def bornees(T):
    # à compléter
    return T

notes = [12, -3, 25, 18]
propres = bornees(notes)
print(notes, propres)
```

CORRIGÉ

La méthode. On fabrique un nouveau tableau, case par case, avec `ajouter`. On ne touche pas à `T` : il est en entrée seule.

1. En entrée seule : la fonction lit `T` et renvoie un autre tableau.
2. La fonction :

PSEUDO-CODE

```
Fonction bornees(T)
    R ← []
    Pour i allant de 0 à longueur(T) - 1 Faire
        Si T[i] < 0 Alors
            ajouter(R, 0)
        Sinon si T[i] > 20 Alors
            ajouter(R, 20)
        Sinon
            ajouter(R, T[i])
        Fin de Si
    Fin de Pour
    Renvoyer R
Fin de Fonction
```

PYTHON

```python
def bornees(T):
    R = []
    for i in range(len(T)):
        if T[i] < 0:
            R.append(0)
        elif T[i] > 20:
            R.append(20)
        else:
            R.append(T[i])
    return R

notes = [12, -3, 25, 18]
propres = bornees(notes)
print(notes, propres)
```

CE QUI S'AFFICHE

```
[12, -3, 25, 18] [12, 0, 20, 18]
```

1. La sortie affiche `notes` après l'appel : il vaut toujours `[12, -3, 25, 18]`. Les notes -3 et 25 sont encore là : le tableau de départ n'a pas changé.

Exercice 5.4.9 — Découper un programme en fonctions ★★★

Une petite école veut un programme pour ses notes. Il doit :

- calculer la moyenne d'un tableau de notes ;
- compter les notes au moins égales à 10 ;
- afficher un bilan : la moyenne et le nombre de notes suffisantes.

1. Proposez trois sous-programmes. Pour chacun, donnez son nom, s'il s'agit d'une fonction ou d'une procédure, ses paramètres et ce qu'il renvoie.
2. Pour chaque paramètre, dites s'il est en entrée seule ou en entrée et sortie.
3. Écrivez les trois en Python, puis appelez le bilan pour `[12, 8, 15, 9]`.

CORRIGÉ

La méthode. Une tâche par sous-programme. Les deux calculs renvoient un résultat : ce sont des fonctions. L'affichage est une action : c'est une procédure, qui appelle les deux fonctions.

1. et 2. Une proposition :

Nom	Sorte	Paramètre	Sens	Renvoie
`moyenne(T)`	fonction	`T`	entrée seule	la moyenne
`nb_suffisantes(T)`	fonction	`T`	entrée seule	un entier
`bilan(T)`	procédure	`T`	entrée seule	rien : elle affiche

1. Le programme :

PYTHON

```python
def moyenne(T):
    s = 0
    for x in T:
        s = s + x
    return s / len(T)

def nb_suffisantes(T):
    nb = 0
    for x in T:
        if x >= 10:
            nb = nb + 1
    return nb

def bilan(T):
    print("Moyenne :", moyenne(T))
    print("Notes suffisantes :", nb_suffisantes(T))

bilan([12, 8, 15, 9])
```

CE QUI S'AFFICHE

```
Moyenne : 11.0
Notes suffisantes : 2
```

Aucune variable globale n'est utilisée dans les fonctions : chacune peut se tester seule.

Attention

Pour un tableau vide, `moyenne([])` diviserait par zéro : Python s'arrêterait sur une erreur `ZeroDivisionError`. Le bilan suppose un tableau non vide. Sinon, on teste `len(T) == 0` avant de diviser.

Entraînement à l'épreuve

Vous passez l'épreuve E3 sous sa forme écrite : **2 heures**, deux ou trois exercices, sur papier, sans ordinateur. La calculatrice est autorisée. L'un des exercices évalue la capacité « programmer ».

Cet exercice vaut souvent **5 points** : comptez 25 à 35 minutes. Vous écrivez vos algorithmes en pseudo-code ou dans un langage, au choix. Python vous sert à vous entraîner, puis à vérifier vos réponses après coup.

Les deux sujets ci-dessous ont la forme des annales. Faites-les sur une feuille, chronomètre en main, avant de lire le corrigé.

Sujet type — Les codes de réduction Type examen

Exercice : 5 points

Une boutique en ligne distribue des codes de réduction. Un code est une chaîne de 8 caractères : 3 lettres majuscules, puis 5 chiffres. Par exemple : `"SIO25401"`.

Un code est **valide** s'il a exactement 8 caractères et si la somme de ses 5 chiffres est un multiple de 7. Un nombre est un multiple de 7 quand le reste de sa division par 7 vaut 0.

On dispose de la fonction `chiffre(c)`, déjà écrite :

- elle reçoit un caractère `c`, qui est un chiffre ;
- elle renvoie l'entier correspondant. Par exemple, `chiffre("7")` renvoie 7.

Les caractères d'un code sont repérés par leur indice, à partir de 0 : dans `"SIO25401"`, le caractère d'indice 3 est `"2"`.

1. On considère la fonction suivante.

```
PSEUDO-CODE
1   Fonction somme_chiffres(code)
2       s ← 0
3       Pour i allant de 3 à 7 Faire
4           s ← s + chiffre(code[i])
5       Fin de Pour
6       Renvoyer s
7   Fin de Fonction
```

1. Que renvoie `somme_chiffres("SIO25401")` ? Justifier. *(0,75 point)*

2. Expliquer en une phrase le rôle de cette fonction. *(0,25 point)*

2. Recopier et compléter les lignes 2, 5 et 6 de la fonction `est_valide`, qui renvoie Vrai si le code est valide et Faux sinon. *(1,5 point)*

PSEUDO-CODE

```
1   Fonction est_valide(code)
2       Si longueur(code) ≠ . . . Alors
3           Renvoyer Faux
4       Fin de Si
5       Si le reste de la division de . . . par 7 est 0 Alors
6           Renvoyer . . .
7       Sinon
8           Renvoyer Faux
9       Fin de Si
10  Fin de Fonction
```

3. Écrire une fonction `nb_valides(T)` qui reçoit un tableau `T` de codes et renvoie le nombre de codes valides. On pourra utiliser la fonction `est_valide`. *(1,5 point)*

4. a. Que renvoie `est_valide("SIO00000")` ? Justifier. *(0,5 point)*

1. La boutique refuse les codes dont la somme des chiffres vaut 0. Réécrire la ligne 5 de `est_valide` en conséquence *(0,5 point)*.

CORRIGÉ

1. a. On suit la boucle : `i` prend les valeurs 3, 4, 5, 6 et 7. Ce sont les indices des 5 chiffres.

Étape	i	code[i]	s
départ	–	–	0
tour 1	3	"2"	2
tour 2	4	"5"	7
tour 3	5	"4"	11
tour 4	6	"0"	11
tour 5	7	"1"	12

La fonction renvoie **12**.

Ce qui rapporte les points : la trace, ou au moins la valeur de `s` à chaque tour, et la valeur finale 12. Une réponse sans justification ne rapporte pas tous les points.

1. b. La fonction renvoie la somme des 5 chiffres du code.

Ce qui rapporte les points : une phrase qui dit « somme des chiffres », et pas une paraphrase ligne par ligne.

2. L'algorithme complet :

```
PSEUDO-CODE
1   Fonction est_valide(code)
2       Si longueur(code) ≠ 8 Alors
3           Renvoyer Faux
4       Fin de Si
5       Si le reste de la division de somme_chiffres(code) par 7 est 0 Alors
6           Renvoyer Vrai
7       Sinon
8           Renvoyer Faux
9       Fin de Si
10  Fin de Fonction
```

Ce qui rapporte les points : les trois trous, 0,5 point chacun. À la ligne 5, il faut la somme des chiffres, donc l'appel `somme_chiffres(code)`, et non `code` lui-même.

3. C'est un compteur : il part de 0, et on ajoute 1 à chaque code valide.

```
PSEUDO-CODE
Fonction nb_valides(T)
    nb ← 0
    Pour chaque code dans T Faire
        Si est_valide(code) Alors
            nb ← nb + 1
        Fin de Si
    Fin de Pour
    Renvoyer nb
Fin de Fonction
```

On peut aussi parcourir les indices : `Pour i allant de 0 à longueur(T) - 1 Faire`, puis `est_valide(T[i])`.

Ce qui rapporte les points :

- l'initialisation `nb ← 0` avant la boucle ;
- une boucle qui passe par tous les codes ;
- le test avec `est_valide` et la mise à jour du compteur ;
- `Renvoyer nb` après la boucle.

4. a. La longueur vaut 8. La somme des chiffres vaut 0 + 0 + 0 + 0 + 0 = 0.

Le reste de la division de 0 par 7 est 0 : la fonction renvoie **Vrai**.

Ce qui rapporte les points : le calcul de la somme, et la remarque que 0 est un multiple de 7.

4. b. On ajoute une seconde condition, avec ET :

```
PSEUDO-CODE
5       Si le reste de la division de somme_chiffres(code) par 7 est 0 ET
somme_chiffres(code) ≠ 0 Alors
```

Ce qui rapporte les points : les deux conditions reliées par ET. Avec OU, un code de somme 12 serait accepté à tort.

Vérifier sur machine après l'épreuve. En Python, `chiffre(c)` s'écrit avec `int(c)`. On teste plusieurs codes, dont les cas limites : un code trop court, un code de somme 0.

```
PYTHON
def chiffre(c):
    return int(c)

def somme_chiffres(code):
    s = 0
    for i in range(3, 8):
        s = s + chiffre(code[i])
    return s

def est_valide(code):
    if len(code) != 8:
        return False
    if somme_chiffres(code) % 7 == 0:
        return True
    else:
        return False

def nb_valides(T):
    nb = 0
    for code in T:
        if est_valide(code):
            nb = nb + 1
    return nb

print(somme_chiffres("SIO25401"))
codes = ["SIO25401", "BTS00007", "WEB34000", "ABC1234", "NET12346"]
for c in codes:
    print(c, est_valide(c))
print("Codes valides :", nb_valides(codes))
print(est_valide("SIO00000"))
```

CE QUI S'AFFICHE

```
12
SIO25401 False
BTS00007 True
WEB34000 True
ABC1234 False
NET12346 False
Codes valides : 2
True
```

La dernière ligne confirme la question 4 a : sans la modification, le code `"SIO00000"` est accepté.

Sujet type — Le serveur qui se remplit Type examen

Exercice court : 3 points

Le disque d'un serveur a déjà 100 Go occupés. Chaque jour, les sauvegardes ajoutent 50 Go. On veut savoir combien de jours il faut pour que l'espace occupé atteigne ou dépasse la capacité du disque.

PSEUDO-CODE

```
1   Fonction jours(capacite)
2       occupe ← 100
3       j ← 0
4       Tant que occupe . . . capacite Faire
5           occupe ← occupe + 50
6           j ← . . .
7       Fin de Tant que
8       Renvoyer j
9   Fin de Fonction
```

1. Recopier et compléter les lignes 4 et 6. *(1 point)*

2. Que renvoie `jours(300)` ? Justifier. *(1,5 point)*

3. Que renvoie `jours(80)` ? Expliquer pourquoi. *(0,5 point)*

CORRIGÉ

1. On continue tant que le disque n'est pas plein, et on compte un jour par tour :

```
PSEUDO-CODE
4       Tant que occupe < capacite Faire
6           j ← j + 1
```

Ce qui rapporte les points : 0,5 point par ligne. Avec `≤`, la boucle ferait un tour de trop quand l'espace occupé tombe pile sur la capacité.

2. La trace de `jours(300)` :

Étape	occupe	j	occupe < 300 ?
départ	100	0	Vrai
tour 1	150	1	Vrai
tour 2	200	2	Vrai
tour 3	250	3	Vrai
tour 4	300	4	Faux : on sort

La fonction renvoie **4**.

Ce qui rapporte les points : la trace avec le test de la condition à chaque tour, et la valeur 4.

3. Au départ, `occupe` vaut 100, et 100 < 80 est Faux. La boucle ne fait aucun tour : la fonction renvoie **0**. Le disque est déjà plein au départ.

Ce qui rapporte les points : dire que la condition est fausse dès le départ, donc 0 tour.

Vérifier sur machine après l'épreuve.

```python
PYTHON
def jours(capacite):
    occupe = 100
    j = 0
    while occupe < capacite:
        occupe = occupe + 50
        j = j + 1
    return j

print(jours(300), jours(80), jours(320))
```

```
CE QUI S'AFFICHE
4 0 5
```

Avec 320 Go, il faut 5 jours : après 4 jours, les 300 Go occupés n'atteignent pas encore 320.

Annales

Ces questions viennent des vrais sujets de l'épreuve écrite de 2 heures du BTS SIO. Dans chaque sujet, un exercice évalue la capacité « programmer ».

Les énoncés sont recopiés tels quels, du plus récent au plus ancien. Leur corrigé est dans l'onglet Corrigés.

BTS SIO 2026 — Le nombre de clés de codage Type examen

Source : BTS SIO, session 2026, E3 – Mathématiques pour l'informatique, exercice 2, question 2.

Annexe A4 Le sujet officiel, page 3 du PDF, à la fin de ce document.

Un système de codage affine est un système de chiffrement basé sur la correspondance entre les lettres de l'alphabet et les entiers compris entre 0 et 25.

La table de correspondance est donnée ci-dessous.

A	B	C	D	E	F	G	H	I	J	K	L	M
0	1	2	3	4	5	6	7	8	9	10	11	12

N	O	P	Q	R	S	T	U	V	W	X	Y	Z
13	14	15	16	17	18	19	20	21	22	23	24	25

Le codage consiste tout d'abord à choisir une clé qui est un couple d'entiers $(a ; b)$ avec $1 \leqslant a \leqslant 25$ et $0 \leqslant b \leqslant 25$.

Pour coder une lettre :

- on détermine l'entier x qui lui correspond dans la table de correspondance ;
- on calcule $ax + b$ puis on détermine l'entier y compris entre 0 et 25 tel que $ax + b \equiv y\,[26]$;
- on détermine la lettre associée à y dans la table de correspondance.

Exemple : Codons la lettre H avec la clé $(9 ; 5)$.

- H correspond à $x = 7$;
- $9 \times 7 + 5 = 68$ et $68 \equiv 16\,[26]$ donc $y = 16$;

- $y = 16$ correspond à la lettre Q donc H est codée en Q.

1. Toutes les clés de codage $(a ; b)$ ne permettent pas de crypter un message convenablement. On admet qu'une clé de codage est acceptable si et seulement si l'entier a est premier avec 26. Il n'y a pas de condition sur b (donc 26 choix possibles pour b).

   Recopier et compléter les lignes numérotées **3**, **4** et **5** de l'algorithme définissant la fonction nbCles() qui renvoie le nombre de clés $(a ; b)$ acceptables et dans laquelle la fonction $\text{pgcd}(u, v)$ renvoie le plus grand commun diviseur des entiers u et v.

PSEUDO-CODE

```
1  Fonction nbCles()
2      nb ← 0
3      Pour a allant de 1 à . . . . Faire
4          Si pgcd(a, 26) = . . . . Alors
5              nb ← . . . .
6          Fin de Si
7      Fin de Pour
8      Renvoyer nb × 26
```

CORRIGÉ

La méthode. La boucle passe en revue toutes les valeurs possibles de `a`. Le Si garde celles qui sont premières avec 26, c'est-à-dire dont le PGCD avec 26 vaut 1.

`nb` les compte. À la fin, chaque `a` acceptable va avec 26 valeurs de `b` : on multiplie par 26.

PSEUDO-CODE

```
1  Fonction nbCles()
2      nb ← 0
3      Pour a allant de 1 à 25 Faire
4          Si pgcd(a, 26) = 1 Alors
5              nb ← nb + 1
6          Fin de Si
7      Fin de Pour
8      Renvoyer nb × 26
```

- Ligne 3 : `25`, car $1 \leqslant a \leqslant 25$.
- Ligne 4 : `1`. Deux entiers sont premiers entre eux quand leur PGCD vaut 1.
- Ligne 5 : `nb + 1`. C'est un compteur.

La vérification. Les valeurs de `a` premières avec 26 sont les nombres impairs, sauf 13 : il y en a 12. La fonction renvoie 12 × 26 = 312.

PYTHON

```python
import math

def nbCles():
    nb = 0
    for a in range(1, 26):
        if math.gcd(a, 26) == 1:
            nb = nb + 1
    return nb * 26

print(nbCles())
```

CE QUI S'AFFICHE

```
312
```

En Python, la fonction PGCD s'appelle `math.gcd`.

Ce qui rapporte les points : les trois lignes complètes, et `nb ← nb + 1` écrit en entier. `nb + 1` seul, sans affectation, ne compte pas.

BTS SIO 2026 — La clé de décodage Type examen

Source : BTS SIO, session 2026, E3 – Mathématiques pour l'informatique, exercice 2, question 3.

Annexe A5 Le sujet officiel, pages 3 à 4 du PDF, à la fin de ce document.

Le contexte est celui de la question précédente : le codage affine, avec la clé $(a ; b)$.

Pour déterminer la clé de décodage associée à la clé de codage $(a ; b)$, il est nécessaire de déterminer un entier c compris entre 0 et 25 tel que $ac \equiv 1 \ [26]$.

La fonction *inverse* de paramètre a donnée ci-dessous renvoie le plus petit entier c compris entre 0 et 25 tel que $ac \equiv 1 \ [26]$.

La fonction « reste(u, n) » qui est utilisée ci-dessous renvoie le reste de la division euclidienne d'un entier naturel u par un entier naturel non nul n.

```
PSEUDO-CODE
1  Fonction inverse(a)
2      c ← 1
3      Tant que reste(a × c, 26) . . . . Faire
4          c ← c + 1
5      Renvoyer . . . .
```

1. 1. Recopier et compléter les lignes numérotées **3** et **5** de l'algorithme définissant la fonction *inverse* de paramètre a.

   2. Déterminer l'entier renvoyé par inverse(9).

CORRIGÉ

La méthode. On essaie `c = 1`, puis 2, puis 3… On continue **tant que** le reste n'est **pas** 1. Dès qu'il vaut 1, on sort et on renvoie `c`.

a.

```
PSEUDO-CODE
1  Fonction inverse(a)
2      c ← 1
3      Tant que reste(a × c, 26) ≠ 1 Faire
4          c ← c + 1
5      Renvoyer c
```

b. La trace de `inverse(9)` :

c	9 × c	reste de la division par 26	≠ 1 ?
1	9	9	Vrai : on continue
2	18	18	Vrai : on continue
3	27	1	Faux : on sort

La fonction renvoie 3. On vérifie : $9 \times 3 = 27 = 26 + 1$, donc $9 \times 3 \equiv 1\ [26]$.

PYTHON

```python
def inverse(a):
    c = 1
    while (a * c) % 26 != 1:
        c = c + 1
    return c

print(inverse(9))
```

CE QUI S'AFFICHE

```
3
```

Attention

Si `a` n'est pas premier avec 26, par exemple `a = 2`, aucun `c` ne convient. La boucle Tant que ne s'arrête jamais. C'est pour ça que seules les clés de la question 2 sont acceptables.

Ce qui rapporte les points : la condition `≠ 1` à la ligne 3, `Renvoyer c` à la ligne 5, et pour b. le résultat 3 justifié par les essais successifs.

BTS SIO 2025 — Le reste par soustractions

Type examen

Source : BTS SIO, Métropole, mai 2025, épreuve obligatoire, exercice 2, question 2.

Annexe A6 Le sujet officiel, page 2 du PDF, à la fin de ce document.

Un codage affine est une méthode de chiffrement utilisée en cryptographie, qui consiste à remplacer chaque lettre du message en clair par une autre lettre de l'alphabet en utilisant une fonction affine. On associe à chaque lettre de l'alphabet une valeur numérique.

1. Pour calculer le reste de la division euclidienne d'un entier N par 26, il suffit de soustraire 26 à N autant de fois que possible, c'est-à-dire tant que la différence demeure supérieure ou égale à 26.

   Exemple : Le reste de la division euclidienne de 80 par 26 s'obtient ainsi :

   - 80 – 26 = 54. Comme 54 ⩾ 26, on soustrait alors 26 à 54.
   - 54 – 26 = 28. Comme 28 ⩾ 26, on soustrait alors 26 à 28.
   - 28 – 26 = 2. Comme 2 < 26, on arrête.
   - Le reste de la division euclidienne de 80 par 26 est 2.

La fonction « reste_division_par_26 » de paramètre N, ci-après, est écrite en langage naturel et emploie cette méthode pour renvoyer le reste de la division euclidienne de N par 26.

Exemple : reste_division_par_26(80) renvoie la valeur entière 2.

PSEUDO-CODE

```
Fonction reste_division_par_26(N)
    Tant que N . . . Faire
        N ← . . .
    Fin de Tant que
    Renvoyer (N)
Fin de la fonction
```

Recopier et compléter les deux lignes contenant des pointillés de cette fonction.

CORRIGÉ

La méthode. L'énoncé dit tout : on soustrait 26 **tant que** N reste supérieur ou égal à 26.

PSEUDO-CODE

```
Fonction reste_division_par_26(N)
    Tant que N ≥ 26 Faire
        N ← N - 26
    Fin de Tant que
    Renvoyer (N)
Fin de la fonction
```

La vérification. Avec N = 80, on passe par 54, puis 28, puis 2. On compare avec l'opérateur `%` de Python :

PYTHON

```python
def reste_division_par_26(N):
    while N >= 26:
        N = N - 26
    return N

print(reste_division_par_26(80), 80 % 26)
print(reste_division_par_26(138), 138 % 26)
```

CE QUI S'AFFICHE

```
2 2
8 8
```

Attention

`Tant que N > 26` est faux pour N = 26 : la fonction renverrait 26 au lieu de 0. Il faut `≥`.

Ce qui rapporte les points : la condition `N ≥ 26`, et la mise à jour `N ← N - 26` écrite avec la flèche.

BTS SIO 2025 — L'indice d'une lettre Type examen

Source : BTS SIO, Métropole, mai 2025, épreuve obligatoire, exercice 2, question 3.

Annexe A7 Le sujet officiel, pages 2 à 3 du PDF, à la fin de ce document.

1. La fonction indice (lettre, chaine) ci-dessous, renvoie le plus petit indice du caractère « lettre » dans la chaîne de caractères « chaine », lorsqu'il est présent dans « chaine », et elle renvoie le nombre de caractères de « chaine » sinon.

   La documentation de cette fonction précise que :

   - k est un entier.
   - lettre est une chaîne de caractères constituée d'un seul caractère.
   - chaine est une chaîne de caractères.
   - longueur(chaine) renvoie le nombre de caractères de chaine.

   Exemple : indice("S","BTSSIO") renvoie la valeur 2, le plus petit indice de "S" dans "BTSSIO".

```
PSEUDO-CODE
Fonction indice(lettre, chaine)
    k ← 0
    Tant que . . . longueur(chaine) et lettre . . . Faire
        k ← k + 1
    Fin de Tant que
    Renvoyer (k)
Fin de Fonction
```

1. Quelle est la valeur renvoyée par indice("A","BTSSIO") ?

2. Recopier et compléter la ligne contenant les pointillés de la fonction indice.

CORRIGÉ

La méthode. On avance `k` tant qu'on n'est pas au bout de la chaîne **et** qu'on n'a pas trouvé la lettre.

a. La lettre A n'est pas dans `"BTSSIO"`. La boucle avance jusqu'au bout de la chaîne : la fonction renvoie le nombre de caractères, **6**.

b.

PSEUDO-CODE

```
Fonction indice(lettre, chaine)
    k ← 0
    Tant que k < longueur(chaine) et lettre ≠ chaine[k] Faire
        k ← k + 1
    Fin de Tant que
    Renvoyer (k)
Fin de Fonction
```

La vérification :

PYTHON

```python
def indice(lettre, chaine):
    k = 0
    while k < len(chaine) and lettre != chaine[k]:
        k = k + 1
    return k

print(indice("S", "BTSSIO"), indice("A", "BTSSIO"))
```

CE QUI S'AFFICHE

```
2 6
```

> **Attention**
>
> Le test `k < longueur(chaine)` doit venir **avant** `lettre ≠ chaine[k]`. Quand `k` vaut 6, le premier test est faux, et on ne lit pas `chaine[6]`, qui n'existe pas.

Ce qui rapporte les points : pour a., la valeur 6 justifiée par l'absence de A. Pour b., les deux morceaux : `k <` et `≠ chaine[k]`.

BTS SIO 2024 — Le bit de parité Type examen

Source : BTS SIO, Métropole, mai 2024, épreuve obligatoire, exercice 2, question 3.

Annexe A8 Le sujet officiel, pages 2 à 3 du PDF, à la fin de ce document.

La norme de codage ASCII (American Standard Code for Information Interchange), défini aux États-Unis en 1963, associe aux caractères les plus utilisés dans les documents en langue anglaise un entier représentable en binaire sur 7 bits.

1. Lors de la transmission des données, pour éviter les erreurs, une méthode consiste à ajouter pour chaque caractère, un bit de parité à la fin du codage en binaire.

   Pour cela, on compte le nombre de 1 apparaissant dans le codage sur 7 bits d'un caractère :

   - si le nombre obtenu est pair, on rajoute 0 à la fin du codage ;
   - sinon on rajoute 1.

   Chaque caractère est alors codé par un groupe de 8 bits appelé octet.

   Par exemple :

   - la lettre "A" est codée par 1000001 en binaire. Ce codage contient un nombre pair de 1, donc on ajoute 0 à la fin, et la lettre "A" sera codée sur 8 bits par 10000010 ;
   - la lettre "C" est codée par 1000011 en binaire. Ce codage contient un nombre impair de 1, donc on ajoute 1 à la fin, et la lettre "C" sera codée sur 8 bits par 10000111.

   On considère l'algorithme ci-dessous écrit en langage naturel où ajoute_bit_parite est la fonction prenant en paramètre une chaine de caractères code de longueur 7 représentant un codage binaire et qui renvoie le codage obtenu en lui ajoutant le bit de parité à la fin.

   Ainsi ajoute_bit_parite("1101100") renvoie "11011000" et ajoute_bit_parite("1100001") renvoie "11000011".

PSEUDO-CODE

```
Fonction ajoute_bit_parite(code)
    compt ← 0
    Pour i allant de 0 à 6 Faire
        Si code[i] est égal à "1"
        Alors

        Fin de Si
    Fin de Pour
    Si le reste de la division de . . . par 2 est 0
    Alors
        code ← code + "0"
    Sinon
        . . . . . .
    Fin de Si
    Renvoyer code
```

Recopier et compléter cet algorithme pour qu'il renvoie le code binaire sous forme de chaine de caractères, complété par le bit de parité.

CORRIGÉ

La méthode. Deux temps.

D'abord, une boucle compte les « 1 » avec le compteur `compt`. Ensuite, le reste de `compt` par 2 dit si ce nombre est pair. On colle le bon bit au bout de la chaîne.

PSEUDO-CODE

```
Fonction ajoute_bit_parite(code)
    compt ← 0
    Pour i allant de 0 à 6 Faire
        Si code[i] est égal à "1"
        Alors
            compt ← compt + 1
        Fin de Si
    Fin de Pour
    Si le reste de la division de compt par 2 est 0
    Alors
        code ← code + "0"
    Sinon
        code ← code + "1"
    Fin de Si
    Renvoyer code
```

Il y avait **trois** trous : la ligne vide après `Alors`, le mot `compt`, et la ligne du `Sinon`.

La vérification. Les indices vont de 0 à 6 : ce sont les 7 bits.

PYTHON

```python
def ajoute_bit_parite(code):
    compt = 0
    for i in range(0, 7):
        if code[i] == "1":
            compt = compt + 1
    if compt % 2 == 0:
        code = code + "0"
    else:
        code = code + "1"
    return code

print(ajoute_bit_parite("1101100"))
print(ajoute_bit_parite("1100001"))
print(ajoute_bit_parite("1000001"), ajoute_bit_parite("1000011"))
```

CE QUI S'AFFICHE

```
11011000
11000011
10000010 10000111
```

On retrouve les quatre exemples de l'énoncé.

> **Attention**
>
> Le `+` colle deux chaînes : `code + "0"` ajoute le caractère 0 à la fin. Avec `code + 0`, sans guillemets, on mélangerait une chaîne et un nombre.

Ce qui rapporte les points : la ligne `compt ← compt + 1` dans le Si, `compt` dans le test de parité, et `code ← code + "1"` avec les guillemets.

BTS SIO 2023 — Le degré sortant d'un sommet

Type examen

Source : BTS SIO, Métropole, mai 2023, épreuve obligatoire, exercice 2, partie B, question 2.

Annexe A9 Le sujet officiel, pages 2 à 3 du PDF, à la fin de ce document.

Partie A. On considère le graphe orienté G comportant 3 sommets notés A, B et C dont la matrice d'adjacence est P, où

$$P = \begin{pmatrix} 1 & 0 & 1 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{pmatrix}$$

Partie B. Dans un graphe orienté, on définit :

- le degré entrant d'un sommet comme étant le nombre d'arcs menant à ce sommet.
- le degré sortant d'un sommet comme étant le nombre d'arcs issus de ce sommet.

1. On étudie dans cette question les graphes orientés à trois sommets numérotés de 1 à 3.

   On considère l'algorithme ci-dessous écrit en langage naturel où Degré_sortant désigne une fonction de paramètres M et s, M étant une matrice à 3 lignes et 3 colonnes et s un entier compris entre 1 et 3.

   Le coefficient de la matrice M situé ligne i colonne j est noté m_{ij}.

PSEUDO-CODE

```
Fonction Degré_sortant(M, s)
    deg ← 0
    Pour j allant de 1 à 3 Faire
        Si m_sj . . . . . Faire
            . . . . . .
        Fin de Si
    Fin de Pour
    Renvoyer deg
```

Compléter cet algorithme pour que la fonction renvoie le degré sortant du sommet numéroté s dans un graphe dont la matrice d'adjacence est M.

CORRIGÉ

La méthode. Dans une matrice d'adjacence, la ligne `s` décrit les arcs qui **partent** du sommet `s`. Un 1 en colonne `j` veut dire : il y a un arc de `s` vers `j`.

Le degré sortant est donc le nombre de 1 de la ligne `s`. On fixe la ligne `s` et on fait varier la colonne `j`.

PSEUDO-CODE

```
Fonction Degré_sortant(M, s)
    deg ← 0
    Pour j allant de 1 à 3 Faire
        Si m_sj > 0 Faire
            deg ← deg + 1
        Fin de Si
    Fin de Pour
    Renvoyer deg
```

On peut aussi écrire `Si m_sj = 1`. Dans une matrice d'adjacence, les coefficients valent 0 ou 1.

La vérification. Le sujet numérote les sommets et les colonnes de 1 à 3. Python numérote à partir de 0 : on lit donc `M[s - 1][j - 1]`.

PYTHON

```python
def degre_sortant(M, s):
    deg = 0
    for j in range(1, 4):
        if M[s - 1][j - 1] > 0:
            deg = deg + 1
    return deg

P = [[1, 0, 1],
     [0, 0, 1],
     [1, 0, 0]]
print(degre_sortant(P, 1), degre_sortant(P, 2), degre_sortant(P, 3))
```

CE QUI S'AFFICHE

```
2 1 1
```

Le sommet 1, A, a deux arcs sortants : vers A, une boucle, et vers C. Le sommet 3, C, en a un : vers A.

> **Attention**
>
> L'erreur fréquente : parcourir la **colonne** `s`, avec `m_js`. On compterait alors les arcs qui arrivent en `s` : c'est le degré entrant.

Ce qui rapporte les points : la condition sur m_{sj}, et l'incrémentation `deg ← deg + 1` dans le Si.

BTS SIO 2022 — Les diviseurs et les nombres premiers Type examen

Source : BTS SIO, Métropole, mai 2022, épreuve obligatoire, exercice 2, question 4.

Annexe A10 Le sujet officiel, page 2 du PDF, à la fin de ce document.

1. On considère l'algorithme ci-dessous écrit en langage naturel où Div désigne une fonction de paramètre Nbre, Nbre étant un nombre entier supérieur ou égal à 2.

   Algorithme en langage naturel :

PSEUDO-CODE

```
Fonction Div(Nbre)
    Test ← 0
    Pour i allant de 1 à Nbre Faire
        Si le reste de la division de Nbre par i est égal à 0 Faire
            Test ← Test + 1
        Fin de Si
    Fin de Pour
    Retourner Test
```

1. Que renvoie Div(6) ? Justifier en expliquant le rôle de cette fonction.

2. Écrire une fonction Prem de paramètre Nbre, où Nbre est un nombre entier supérieur ou égal à 2, qui renvoie Vrai (ou True) si Nbre est premier, Faux (ou False) si Nbre n'est pas premier. On pourra utiliser la fonction Div.

CORRIGÉ

La méthode. Pour a., on fait la trace : `i` parcourt les entiers de 1 à 6, et `Test` compte ceux qui divisent 6. Pour b., on se sert du résultat : un nombre premier a exactement deux diviseurs, 1 et lui-même.

a. La trace de `Div(6)` :

i	reste de 6 par i	Test
départ	-	0
1	0	1
2	0	2
3	0	3
4	2	3
5	1	3
6	0	4

`Div(6)` renvoie 4. La fonction **compte le nombre de diviseurs** de Nbre. Les diviseurs de 6 sont 1, 2, 3 et 6.

b. Un entier supérieur ou égal à 2 est premier si et seulement s'il a exactement 2 diviseurs.

PSEUDO-CODE

```
Fonction Prem(Nbre)
    Si Div(Nbre) = 2 Alors
        Renvoyer Vrai
    Sinon
        Renvoyer Faux
    Fin de Si
Fin de Fonction
```

On peut l'écrire en une ligne : `Renvoyer Div(Nbre) = 2`.

La vérification. On teste un nombre premier, 7, et deux nombres qui ne le sont pas, 9 et 623 = 7 × 89.

PYTHON

```python
def Div(Nbre):
    Test = 0
    for i in range(1, Nbre + 1):
        if Nbre % i == 0:
            Test = Test + 1
    return Test

def Prem(Nbre):
    return Div(Nbre) == 2

print(Div(6))
print(Prem(7), Prem(9), Prem(623))
```

CE QUI S'AFFICHE

```
4
True False False
```

Attention

Le corrigé officiel de ce sujet écrit `Afficher Vrai` et `Afficher Faux`. C'est une erreur de forme : l'énoncé demande une fonction qui **renvoie** Vrai ou Faux.

Une fonction qui affiche ne rend rien qu'on puisse réutiliser. Écrivez `Renvoyer`.

Ce qui rapporte les points : pour a., la valeur 4 **et** le rôle de la fonction, justifiés par les restes nuls. Pour b., l'appel à `Div` et le test `= 2`, avec un résultat renvoyé.

FICHE

Tout le cours en résumé

Chapitre 1 — Écrire un algorithme

- **Algorithme** : une suite d'instructions qui résout un problème, en un nombre fini d'étapes.
- **Exécution séquentielle** : les instructions s'exécutent dans l'ordre, de haut en bas.
- **Variable** : une case de la mémoire, avec un nom et une valeur.
- **Affectation** `x ← 5` : on calcule la droite, puis on range le résultat dans `x`. L'ancienne valeur est perdue.
- **Mise à jour** `x ← x + 1` : la nouvelle valeur dépend de l'ancienne.
- **Lire** : demander une valeur au clavier. **Écrire** : afficher à l'écran.
- **Trace** : un tableau avec une colonne par variable et une ligne par étape.
- **Écrire le pseudo-code** : un bloc décalé de 4 espaces, chaque structure fermée par son « Fin de ».
- **En-tête** : nom, rôle, entrées, sortie. Un commentaire `//` dit le rôle d'une variable.

Action	Pseudo-code	Python
Affecter	`x ← 5`	`x = 5`
Mettre à jour	`n ← n + 1`, `n ← n - 1`	`n += 1`, `n -= 1`
Lire un entier	`Lire n`	`n = int(input("n ? "))`
Écrire	`Écrire "Total :", t`	`print("Total :", t)`
Commentaire	`// …`	`# …`

Un exemple résolu : la trace de `a ← 3`, `b ← a + 4`, `a ← a * b`.

Étape	a	b
1	3	–
2	3	7
3	21	7

Les pièges :

- ✘ **Utiliser une variable avant de la calculer** → écrivez les lignes dans l'ordre où on en a besoin.
- ✘ **Échanger deux valeurs avec `a ← b` puis `b ← a`** → passez par une variable `temp`.
- ✘ **Donner le résultat sans trace** → à l'examen, la trace est la justification.

Chapitre 2 — Les types de données

- **Entier naturel** : entier positif ou nul. **Entier relatif** : entier positif ou négatif. En Python : `int`.
- **Réel** : nombre à virgule, écrit avec un point en Python. En Python : `float`. La division `/` donne toujours un réel : `8 / 2` vaut `4.0`.
- **Booléen** : Vrai ou Faux. En Python : `bool`, avec `True` et `False`.
- **Chaîne de caractères** : un texte entre guillemets, d'un seul bloc. En Python : `str`. `chaine[i]` extrait le caractère d'indice `i`. On ne modifie pas un caractère : `mot[0] ← "X"` est une erreur.
- **Domaine de valeurs** : sur n bits, on écrit 2^n valeurs. Un entier naturel va de 0 à $2^n - 1$.
- **Dépassement de capacité** : le calcul sort du domaine, le résultat devient faux.
- Les entiers sont **exacts**. Les réels sont **approchés** : `0.1 + 0.2` affiche `0.30000000000000004`.
- **Indice** : la position d'un caractère, à partir de 0. Le dernier est à `longueur(s) - 1`.
- **Transtypage** : changer le type d'une valeur.

Bits	Entier naturel	Entier relatif
8	0 à 255	-128 à 127
16	0 à 65 535	-32 768 à 32 767

Conversion	Python	Résultat
texte → entier	`int("42")`	42
texte → réel	`float("3.5")`	3.5
nombre → texte	`str(42)`	`"42"`
réel → entier	`int(3.9)`	3

Un exemple résolu : l'âge tapé au clavier.

PYTHON

```python
age = "25"
print(age + "1")
print(int(age) + 1)
```

CE QUI S'AFFICHE

```
251
26
```

Les pièges :

- ✗ **Tester `x == 0.3` avec des réels** → arrondissez avec `round`, ne testez pas l'égalité exacte.
- ✗ **Calculer avec ce que renvoie `input`** → convertissez d'abord avec `int` ou `float`.
- ✗ **Prendre `mot[1]` pour le premier caractère** → le premier est `mot[0]`.

Chapitre 3 — Les opérateurs

- **div** et **mod** : quotient et reste de la division euclidienne. 17 = 5 × 3 + 2.
- `n mod 2 = 0` : `n` est pair.
- Une formule en ligne garde l'ordre des calculs avec des **parenthèses** : `(a + b) / 2`.
- **Fonctions usuelles** : `abs`, `round`, `math.floor`, `math.ceil`, `math.sqrt`. Hasard **uniforme** : `random.randint(a, b)` pour un entier, `random.uniform(a, b)` pour un réel.
- **Comparaison** : donne un booléen. « Différent » s'écrit `≠`, `<>` ou `!=`. Les caractères se comparent par leur code : chiffres, puis majuscules, puis minuscules.
- **ET** : les deux vrais. **OU** : au moins un vrai. **OUX** : exactement un vrai. **NON** : inverse. ET passe avant OU : mettez des parenthèses.
- **Masque** : un nombre dont les 1 désignent les bits à toucher. Le bit 0 est le plus à droite.
- **Concaténation** : coller deux chaînes avec `+`. **Motif** : un texte qu'on cherche.

Ce qu'on veut	Pseudo-code	Python
Quotient, reste	`a div b`, `a mod b`	`a // b`, `a % b`
Puissance	`a ^ b`	`a ** b`
Égal, différent	`=`, `≠`	`==`, `!=`
Logique	`ET`, `OU`, `NON`	`and`, `or`, `not`
Tester, allumer, inverser un bit	ET, OU, OUX avec un masque	`x & m`, `x | m`, `x ^ m`
Éteindre un bit	ET avec l'inverse du masque	`x & ~m`
Morceau de chaîne	`sous_chaine(s, i, j)`	`s[i:j]`, `j` exclu
Chercher un motif	`position(s, motif)`	`s.find(motif)`, -1 si absent

Un exemple résolu : convertir 7 384 secondes.

PYTHON

```python
duree = 7384
heures = duree // 3600
reste = duree % 3600
print(heures, "h", reste // 60, "min", reste % 60, "s")
```

CE QUI S'AFFICHE

```
2 h 3 min 4 s
```

Les pièges :

- ✘ **`a + b / 2` pour une moyenne** → écrivez `(a + b) / 2`.
- ✘ **`jour == "samedi" or "dimanche"`** → écrivez chaque comparaison en entier.
- ✘ **`=` pour tester en Python** → `=` range, `==` teste.
- ✘ **`s[4:6]` pour trois caractères** → la borne de droite est exclue, écrivez `s[4:7]`.

Chapitre 4 — Les structures de contrôle

- **Si… Alors… Sinon** : un seul bloc s'exécute, celui de la première condition vraie.
- **Pour** : on connaît le nombre de tours. `range(1, n + 1)` va de 1 à n. Si la fin est plus petite que le début : 0 tour.
- **Pour et Répéter s'écrivent avec Tant que** : `i ← 1`, `Tant que i ≤ n Faire … i ← i + 1`.
- **Tant que** : on teste avant chaque tour. Le bloc doit faire avancer la condition, sinon c'est une **boucle infinie**.
- **Répéter… Jusqu'à** : au moins un tour, on s'arrête quand la condition devient vraie.
- **Boucle de calcul** : préparation, itération, mise en forme finale. On la construit avec une **propriété vraie à la fin de chaque tour** (raisonnement par récurrence) : « après le tour i, `s` contient 1 + … + i ».
- **Accumulateur** : part de 0 pour une somme, de 1 pour un produit.
- **Σ** devient une boucle Pour qui additionne. **Π** devient une boucle Pour qui multiplie.
- **Structures imbriquées** : la boucle intérieure fait tous ses tours à chaque tour de la boucle extérieure.
- **Jeu d'essai** : une valeur par branche, les valeurs limites, un cas d'erreur, et le résultat attendu pour chacune.
- **Combien de tours ?** : pour comparer deux algorithmes, on compte les tests de la boucle. S'arrêter dès qu'on a trouvé fait souvent beaucoup moins de tours.

Structure	Pseudo-code	Python
Si	`Si … Alors` / `Sinon si` / `Sinon` / `Fin de Si`	`if …:` / `elif …:` / `else:`
Pour	`Pour i allant de 1 à n Faire` / `Fin de Pour`	`for i in range(1, n + 1):`
Tant que	`Tant que … Faire` / `Fin de Tant que`	`while …:`
Répéter	`Répéter` / `Jusqu'à …`	`while True:` puis `if …: break`

Un exemple résolu : $\sum_{i=1}^{4} i^2$ et $\prod_{i=1}^{4} i$.

PYTHON

```python
s = 0
p = 1
for i in range(1, 5):
    s = s + i ** 2
    p = p * i
print(s, p)
```

CE QUI S'AFFICHE

```
30 24
```

Les pièges :

- ✘ **`range(1, 10)` pour aller de 1 à 10** → écrivez `range(1, 11)`.
- ✘ **Initialiser l'accumulateur dans la boucle** → mettez `s ← 0` avant la boucle.
- ✘ **Diviser pour la moyenne dans la boucle** → la mise en forme se fait après la boucle.
- ✘ **Tester le cas faible avant le cas fort** → dans un `Sinon si`, le premier cas vrai gagne.

Chapitre 5 — Les tableaux et les fonctions

- **Tableau** : plusieurs valeurs sous un seul nom, repérées par un indice à partir de 0.
- **Construire** : `T ← []` crée un tableau vide, `ajouter(T, v)` ajoute à la fin. Une grille se construit avec deux boucles.
- **Parcourir** : une boucle sur les indices, de 0 à `longueur(T) - 1`, ou directement les valeurs : `Pour chaque x dans T`, `for x in T`.
- **Maximum** : on part de `T[0]`, puis on compare avec chacun des suivants.

- **Copier** : élément par élément, en partant de `c ← []` et en ajoutant chaque `x`. En Python : `b = a.copy()`. Avec `b = a`, les deux noms désignent le même tableau.
- **Tableau à deux dimensions** : `M[ligne][colonne]`. Une ligne : `i` fixe, `j` varie. Une colonne : `j` fixe, `i` varie.
- **Fonction** : reçoit des **paramètres** et renvoie un résultat avec `Renvoyer`. **Procédure** : agit, ne renvoie rien. Plusieurs résultats : `(total, produit) ← f(T)`, en Python `total, produit = f(T)`.
- **Variable locale** : n'existe que dans la fonction. **Variable globale** : visible partout. La modifier dans une fonction (`global` en Python) est à éviter.
- **Entrée seule** : le paramètre est lu, pas modifié. **Sortie seule** : il est rempli sans être lu, comme dans `remplir(T, n)`. **Entrée et sortie** : il est lu puis modifié.

Action	Pseudo-code	Python
Créer	`T ← [12, 15, 9]`	`T = [12, 15, 9]`
Lire, écrire une case	`T[1]`, `T[1] ← 14`	`T[1]`, `T[1] = 14`
Ajouter à la fin	`ajouter(T, 8)`	`T.append(8)`
Fonction	`Fonction f(x)` / `Renvoyer …` / `Fin de Fonction`	`def f(x):` / `return …`

Un exemple résolu : la moyenne d'un tableau, dans une fonction.

PYTHON

```python
def moyenne(T):
    s = 0
    for i in range(len(T)):
        s = s + T[i]
    return s / len(T)

print(moyenne([12, 15, 9, 18]))
```

CE QUI S'AFFICHE

```
13.5
```

Les pièges :

- ✘ `range(len(T) + 1)` → le dernier indice est `len(T) - 1`.
- ✘ **Partir de 0 pour un maximum** → partez de `T[0]`.
- ✘ `Écrire` **à la place de** `Renvoyer` **dans une fonction** → la fonction doit rendre son résultat.
- ✘ **Utiliser hors de la fonction une variable créée dedans** → elle est locale, elle n'existe plus.

QCM

Réviser vite, en boucle

Chapitre 1 — Écrire un algorithme

1. Qu'est-ce qu'un algorithme ?

- **A.** Un langage de programmation
- **B.** Une suite d'instructions qui résout un problème en un nombre fini d'étapes
- **C.** Un ordinateur qui calcule seul

2. Dans quel ordre l'ordinateur exécute-t-il les instructions d'un algorithme sans boucle ni condition ?

- **A.** Dans l'ordre qu'il juge le plus rapide
- **B.** De bas en haut
- **C.** De haut en bas, une par une

3. Quelle est la différence entre le pseudo-code et Python ?

- **A.** Le pseudo-code tourne plus vite que Python
- **B.** Il n'y en a aucune
- **C.** Python s'écrit en français
- **D.** Le pseudo-code ne tourne sur aucune machine, Python tourne et sert à tester

4. Un programme commence par `Écrire reste`, puis `reste ← 40 - 15`. Que se passe-t-il ?

- **A.** Il affiche 25
- **B.** Une erreur : `reste` n'existe pas encore au moment de l'afficher
- **C.** Il affiche `0`

5. Dans un en-tête d'algorithme, que trouve-t-on ?

- **A.** Seulement le résultat final
- **B.** La liste de toutes les traces
- **C.** Le nom, le rôle, les entrées et la sortie

6. Que vaut `n` après `n ← 10 ; n ← n - 3 ; n ← n * 2` ?

- **A.** 4
- **B.** 10
- **C.** 14
- **D.** 17

7. Que fait l'instruction `x ← x + 2` ?

- **A.** Elle teste si `x` est égal à `x + 2`
- **B.** Elle lit la valeur de `x`, ajoute 2, et range le résultat dans `x`
- **C.** Elle est impossible, comme en maths

8. On exécute `a ← 3 ; b ← 8 ; a ← b ; b ← a`. Que valent `a` et `b` ?

A. `a` vaut 8 et `b` vaut 3
B. `a` vaut 8 et `b` vaut 8
C. `a` vaut 3 et `b` vaut 8

9. Quelle instruction est fausse en pseudo-code ?

A. `prix ← prix * 2`
B. `x ← 5`
C. `total ← a + b`
D. `5 ← x`

10. On a `x ← 4 ; y ← x ; x ← 10`. Que vaut `y` ?

A. 10
B. 14
C. 4

11. Comment écrit-on `compteur ← compteur + 1` en Python ?

A. `compteur == compteur + 1`
B. `compteur + 1 = compteur`
C. `compteur += 1`

12. En Python, quel est le type de ce que renvoie `input("Âge ? ")` quand on tape 17 ?

A. Un entier, 17
B. Une chaîne de caractères, `"17"`
C. Un réel, 17.0

13. Quelle ligne lit un entier au clavier en Python ?

A. `n = input(int("n ? "))`
B. `n = int(input("n ? "))`
C. `n = input("n ? ")`

14. Qu'affiche `print("Bonjour", "Inès")` ?

A. `BonjourInès`
B. `Bonjour Inès`
C. `"Bonjour", "Inès"`

15. Quelle est l'erreur dans `print("Bonjour" prenom)` ?

A. Il faut écrire `Écrire` au lieu de `print`
B. `prenom` doit être entre guillemets
C. Il manque une virgule entre les deux morceaux

16. Pour lire un prix comme 12.5, quelle conversion utilise-t-on ?

A. `int(input("Prix ? "))`
B. `str(input("Prix ? "))`
C. `float(input("Prix ? "))`

17. Qu'est-ce que la trace d'un algorithme ?

A. Un tableau qui montre la valeur de chaque variable après chaque instruction
B. Le message d'erreur de Python
C. La liste des instructions sans les valeurs

18. Faites la trace de `x ← 4 ; y ← x + 3 ; x ← x * y`. Que vaut `x` à la fin ?

A. 12
B. 28
C. 7
D. 16

19. Après `a ← 5 ; b ← a * 2 ; a ← b - a`, que valent `a` et `b` ?

A. `a` vaut 5 et `b` vaut 10
B. `a` vaut 10 et `b` vaut 5
C. `a` vaut -5 et `b` vaut 10

20. Dans une trace, qu'écrit-on pour une variable qui n'existe pas encore ?

A. 0
B. La valeur qu'elle aura à la fin
C. Un tiret `–`

21. Pourquoi faire une trace à l'examen quand on demande « Que renvoie l'algorithme ? Justifier » ?

A. Parce que le résultat seul est interdit
B. Pour remplir la copie
C. Parce que la trace prouve qu'on a suivi le chemin, sans deviner

22. Comment se ferme une boucle Pour dans un pseudo-code bien écrit ?

A. Par une accolade `}`
B. Par `Fin de Pour`
C. On ne la ferme pas, l'indentation suffit

23. Que contient l'en-tête d'un algorithme ?

A. Le résultat déjà calculé
B. La trace complète
C. Le nom, le rôle, les entrées et la sortie

24. Que vaut `n` après `n = 10` puis `n -= 3` ?

A. 13
B. 7
C. -3

Chapitre 2 — Les types de données

25. Quel type choisir pour le nombre de fichiers dans un dossier ?

A. Entier naturel
B. Réel
C. Booléen

26. Quel type choisir pour une température de -7 °C ?

A. Entier naturel
B. Entier relatif
C. Booléen

27. Quel est le type de `3 > 5` ?

A. Entier
B. Chaîne
C. Booléen

28. Pourquoi ranger « a payé » dans un booléen plutôt que dans la chaîne `"oui"` ?

A. Le booléen n'a que deux valeurs, sans erreur de frappe possible
B. Une chaîne prend moins de place
C. Python refuse les chaînes `"oui"`

29. Comment écrit-on le réel 12,67 en Python ?

A. `12,67`
B. `"12,67"`
C. `12.67`

30. Que perd-on si on range une moyenne de 12,67 dans un entier ?

A. La partie après la virgule
B. Rien
C. Le signe

31. Combien de valeurs différentes peut-on écrire sur 8 bits ?

A. 8
B. 255
C. 256
D. 128

32. Quelle est la plus grande valeur d'un entier naturel sur 10 bits ?

A. 1024
B. 1023
C. 100

33. Un compteur entier naturel sur 8 bits vaut 255. On ajoute 1. Que se passe-t-il ?

A. Il vaut 256
B. Il reste à 255
C. Dépassement de capacité : il repasse à 0

34. Qu'affiche `print(0.1 + 0.2)` en Python ?

A. `0.3`
B. `0.30000000000000004`
C. `0.2`

35. Comment comparer correctement deux réels ?

A. Avec `=` en pseudo-code, c'est toujours sûr
B. En les changeant en chaînes
C. Jamais par égalité exacte : on arrondit avant

36. Quel est le domaine d'un entier relatif sur 16 bits ?

A. -32 768 à 32 767
B. 0 à 65 535
C. -65 535 à 65 535

37. Faut-il 8 ou 16 bits pour compter 1 000 visiteurs avec un entier naturel ?

A. 16 bits, car 1 000 dépasse 255
B. 8 bits suffisent
C. Il en faut 1 000

38. Que vaut `"Python"[2]` ?

A. `y`
B. `t`
C. `h`

39. Quelle est la longueur de `"BTS SIO"` ?

A. 6
B. 5
C. 7

40. Quel est l'indice du dernier caractère d'une chaîne `mot` ?

A. `longueur(mot)`
B. `longueur(mot) - 1`
C. `0`

41. Que donne `mot[len(mot)]` en Python ?

A. Le dernier caractère
B. Une chaîne vide
C. Une erreur `IndexError`

42. Le numéro de téléphone `06 12 34 56 78` doit être rangé comme :

A. une chaîne de caractères
B. un entier naturel
C. un réel

43. Que vaut `"3" + "4"` en Python ?

A. `7`
B. Une erreur
C. `"34"`

44. Que vaut `int(7.8)` ?

A. 7
B. 8
C. 7.8

45. Que donne `int("12.5")` en Python ?

A. 12
B. 13
C. Une erreur

46. Que donne `float("3,5")` en Python ?

A. 3.5
B. Une erreur `ValueError`
C. 3

47. Comment afficher « Total : 25 » en collant du texte et le nombre `t` qui vaut 25 ?

A. `"Total : " + str(t)`
B. `"Total : " + t`
C. `"Total : " + int(t)`

48. Qu'affiche `print(8 / 2)` en Python ?

A. `4`
B. `"4"`
C. `4.0`

49. On exécute `mot = "Lea"` puis `mot[0] = "B"`. Que se passe-t-il ?

A. `mot` devient `"Bea"`
B. Une erreur : on ne modifie pas un caractère d'une chaîne
C. `mot` devient `"B"`

Chapitre 3 — Les opérateurs

50. Que valent `23 div 4` et `23 mod 4` ?

A. 5 et 75
B. 6 et 1
C. 5 et 3

51. En Python, que vaut `17 // 5` ?

A. 3.4
B. 3
C. 2

52. Comment tester qu'un entier `n` est pair ?

A. `n div 2 = 0`
B. `n / 2 = 1`
C. `n mod 2 = 0`

53. Comment écrire $\frac{a+b}{2}$ en ligne ?

A. `(a + b) / 2`
B. `a + b / 2`
C. `a + (b / 2)`

54. En Python, que vaut `2 ** 10` ?

A. 20
B. 12
C. 1024

55. Combien de minutes entières dans 2 800 secondes, et combien de secondes restent ?

A. 47 min et 0 s
B. 46 min et 40 s
C. 46 min et 4 s

56. Comment écrire $3x^2 - 5$ en Python ?

- **A.** `3x^2 - 5`
- **B.** `(3 * x) ^ 2 - 5`
- **C.** `3 * x ** 2 - 5`

57. Que donne `round(3.678, 2)` ?

- **A.** 3.68
- **B.** 3.67
- **C.** 3.7

58. Combien de cartons de 50 faut-il pour ranger 130 câbles ?

- **A.** `round(130 / 50)`
- **B.** `math.ceil(130 / 50)`, soit 3
- **C.** `math.floor(130 / 50)`, soit 2

59. Que donne `math.floor(-2.5)` ?

- **A.** -2
- **B.** 2
- **C.** -3

60. Quelle instruction Python simule un dé à 6 faces ?

- **A.** `random.randint(0, 6)`
- **B.** `random.randint(1, 7)`
- **C.** `random.randint(1, 6)`

61. Que manque-t-il pour que `math.sqrt(49)` marche ?

- **A.** La ligne `import math` en haut du programme
- **B.** Des guillemets autour de 49
- **C.** Rien, Python connaît `sqrt` seul

62. Pourquoi dit-on qu'un ordinateur tire des nombres « pseudo-aléatoires » ?

- **A.** Parce qu'ils sont toujours les mêmes
- **B.** Parce qu'un calcul les produit, ils ne viennent pas d'un vrai hasard
- **C.** Parce qu'ils sont faux

63. Que vaut `(3 > 2) and (5 < 1)` ?

- **A.** `False`
- **B.** `True`
- **C.** Une erreur

64. Quand l'expression `a OU b` est-elle fausse ?

A. Quand `a` est Faux

B. Quand un seul des deux est Faux

C. Seulement quand `a` et `b` sont Faux tous les deux

65. Que vaut `Vrai OUX Vrai` ?

A. Vrai

B. Faux

C. Une erreur, on ne combine pas deux Vrai

66. Comment traduire « l'âge est entre 18 et 25 inclus » ?

A. `age >= 18 or age <= 25`

B. `age > 18 and age < 25`

C. `age >= 18 and age <= 25`

67. Quelle écriture Python teste correctement « le jour est samedi ou dimanche » ?

A. `jour == "samedi" or jour == "dimanche"`

B. `jour == "samedi" or "dimanche"`

C. `jour == "samedi" and jour == "dimanche"`

68. En Python, quelle est la différence entre `x = 5` et `x == 5` ?

A. Aucune

B. `x = 5` range 5 dans `x`, `x == 5` teste si `x` vaut 5

C. `x = 5` teste, `x == 5` range

69. Que vaut `"Alice" < "Bob"` ?

A. `True`, car le premier caractère `A` vient avant `B`

B. `False`, car « Alice » est plus long

C. Une erreur, on ne compare pas des chaînes

70. Traduisez « le stock est vide ou le produit n'est plus vendu ».

A. `stock == 0 and not vendu`

B. `stock = 0 or vendu`

C. `stock == 0 or not vendu`

71. Dans le nombre binaire `0b101`, que vaut le bit `0` ?

A. `0`

B. 1

C. 5

72. Quel opérateur et quel masque pour tester le bit 2 de `x` ?

- **A.** `x & 0b100`
- **B.** `x | 0b100`
- **C.** `x & 0b010`

73. Que vaut `0b1000 | 0b0001` ?

- **A.** `0b0000`
- **B.** `0b1000`
- **C.** `0b1001`

74. Avec quel opérateur change-t-on l'état d'un bit avec un masque ?

- **A.** OUX `^`
- **B.** ET `&`
- **C.** OU `|`

75. On a `droits = 0b101`. Que vaut `droits & ~0b001` ?

- **A.** `0b101`
- **B.** `0b001`
- **C.** `0b100`

76. Pourquoi `x & 0b0100` ne sert pas à mettre le bit 2 à 0 ?

- **A.** Parce qu'il éteint tous les autres bits et garde seulement le bit 2
- **B.** Parce qu'il allume le bit 2
- **C.** Parce que `&` n'existe pas en Python

77. Que vaut `1011 ET 0010`, rang par rang ?

- **A.** `0010`
- **B.** `1011`
- **C.** `1001`

78. Que vaut `"BTS" + " SIO"` ?

- **A.** `"BTSSIO"`
- **B.** `"BTS SIO"`
- **C.** Une erreur

79. Que vaut `"BTS SIO"[4:7]` ?

- **A.** `" SI"`
- **B.** `"SIO "`
- **C.** `"SIO"`

80. Que vaut `"192.168.1.10"[:3]` ?

A. `"192"`
B. `"192."`
C. `"19"`

81. Que vaut `"bonjour".find("j")` ?

A. 4
B. 3
C. -1

82. Que renvoie `s.find(motif)` si le motif est absent ?

A. -1
B. `0`
C. Une erreur

83. On a `s = "@moi"`. Pourquoi `s.find("@")` seul ne suffit pas pour dire si `@` est présent ?

A. Parce que `find` ne marche pas avec `@`
B. Parce que `find` renvoie `0` quand le motif est au début, et non Vrai
C. Parce qu'il faut des crochets

84. Comment obtenir l'extension `"pdf"` de `nom = "rapport.pdf"` ?

A. `nom[:3]`
B. `nom[3]`
C. `nom[-3:]`

85. Que vaut `"a" < "B"` en Python ?

A. `True`, car a vient avant b dans l'alphabet
B. `False`, car les majuscules viennent avant les minuscules
C. Une erreur

86. Dans certains sujets, comment s'écrit aussi « différent » ?

A. `><`
B. `=!`
C. `<>`

87. Que vaut `True or True and False` en Python ?

A. `True`
B. `False`
C. Une erreur

88. Quelle instruction tire un nombre réel au hasard entre `0` et 1 ?

A. `random.randint(0, 1)`
B. `random.uniform(0, 1)`
C. `round(0.5)`

89. Que veut dire un tirage « uniforme » ?

A. Le même nombre sort à chaque fois
B. Les grands nombres sortent plus souvent
C. Chaque valeur de l'intervalle a la même chance de sortir

Chapitre 4 — Les structures de contrôle

90. Dans un `Si … Sinon si … Sinon`, combien de blocs sont exécutés ?

A. Tous ceux dont la condition est vraie
B. Un seul, celui de la première condition vraie
C. Aucun si la première condition est fausse

91. On a `x ← -3`. Qu'affiche `Si x > 0 Alors Écrire "positif" Sinon Écrire "négatif ou nul" Fin de Si` ?

A. `positif`
B. Rien
C. `négatif ou nul`

92. Un poste a 7 ans. Le programme teste `age ≥ 4` puis `Sinon si age ≥ 6`. Qu'affiche-t-il ?

A. Le message du cas `age ≥ 6`
B. Le message du cas `age ≥ 4`
C. Les deux messages

93. En Python, qu'est-ce qui marque le bloc d'un `if` ?

A. Les deux-points et l'indentation de 4 espaces
B. Le mot `Fin de Si`
C. Des accolades

94. Comment écrit-on `Sinon si` en Python ?

A. `else if`
B. `elseif`
C. `elif`

95. Combien de fois s'exécute le bloc de `Pour i allant de 3 à 7 Faire` ?

A. 4 fois
B. 5 fois
C. 7 fois

96. Quelles valeurs donne `range(4)` ?

A. 1, 2, 3, 4
B. 0, 1, 2, 3, 4
C. 0, 1, 2, 3

97. Comment faire aller `i` de 1 à 10 en Python ?

A. `for i in range(1, 11):`
B. `for i in range(1, 10):`
C. `for i in range(10):`

98. Que vaut `somme` après
`somme ← 0 ; Pour i allant de 1 à 4 Faire somme ← somme + i Fin de Pour` ?

A. 4
B. 6
C. 10

99. On écrit `compt ← 0` à l'intérieur de la boucle Pour. Que se passe-t-il ?

A. Rien de grave
B. Le compteur repart à 0 à chaque tour, le résultat est faux
C. La boucle ne s'arrête plus

100. Combien de « 1 » compte une boucle qui parcourt `"1011001"` ?

A. 3
B. 7
C. 4

101. Quand utilise-t-on une boucle Tant que plutôt qu'une boucle Pour ?

A. Quand on ne connaît pas le nombre de tours à l'avance
B. Quand on connaît le nombre de tours
C. Jamais, Pour fait tout

102. Qu'affiche `n ← 80 ; Tant que n ≥ 26 Faire n ← n - 26 Fin de Tant que ; Écrire n` ?

A. 28
B. 2
C. 0

103. Qu'affiche
`n ← 100 ; Tant que n > 10 Faire n ← n div 2 Fin de Tant que ; Écrire n` ?

A. 6
B. 12
C. 5

104. Quelle boucle fait toujours au moins un tour ?

A. Tant que

B. Aucune

C. Répéter … Jusqu'à

105. Pourquoi `Tant que n ≥ 26 Faire Écrire n Fin de Tant que` ne s'arrête-t-il jamais si `n` vaut 30 ?

A. Parce que 30 est trop grand

B. Parce que le bloc ne change pas `n` : la condition reste vraie

C. Parce qu'il manque `Faire`

106. Comment imite-t-on Répéter … Jusqu'à en Python ?

A. `repeat … until`

B. `for i in range(1):`

C. `while True:` puis `if condition: break`

107. On cherche une lettre dans `mot` avec `Tant que mot[k] ≠ lettre Faire k ← k + 1`. Quel est le risque ?

A. Aucun

B. Si la lettre est absente, `k` dépasse le dernier indice : erreur

C. La boucle s'arrête au premier tour

108. Quelle valeur de départ pour un accumulateur de somme ?

A. 0

B. 1

C. La première valeur à additionner, toujours

109. Quelle valeur de départ pour calculer $\prod_{i=1}^{5} i$?

A. 1

B. 0

C. 5

110. Que vaut $\prod_{i=1}^{5} i$?

A. 15

B. 120

C. 25

111. Que vaut $\sum_{i=1}^{4} i^2$?

A. 10

B. 16

C. 30

112. Que vaut $\sum_{k=1}^{10} 2k$?

A. 20
B. 55
C. 110

113. Pour une moyenne, où écrit-on `moyenne ← total / 5` ?

A. Dans la boucle, à chaque tour
B. Après la boucle, une seule fois
C. Avant la boucle

114. Quels sont les trois temps d'une boucle de calcul ?

A. Préparation, itération, mise en forme finale
B. Lire, écrire, tester
C. Début, milieu, erreur

115. `Pour i allant de 1 à 4` contient `Pour j allant de 1 à 3`. Combien de fois s'exécute le bloc intérieur ?

A. 7
B. 12
C. 3

116. Quand `i` passe de 1 à 2 dans deux boucles imbriquées, que fait `j` ?

A. Il continue où il était
B. Il s'arrête
C. Il repart à sa première valeur

117. Combien de lignes affiche le programme des étoiles avec `Pour i allant de 1 à 5`, la boucle intérieure allant de 1 à `i` ?

A. 15
B. 1
C. 5

118. Dans le produit de deux matrices, où remettre `s ← 0` ?

A. Une seule fois, avant toutes les boucles
B. Juste avant la boucle sur k, pour chaque case (i, j)
C. Dans la boucle sur k

119. Dans une boucle Pour de 1 à 10, on fait `Si i mod 2 = 0 Alors compt ← compt + 1`. Que vaut `compt` à la fin, s'il partait de 0 ?

A. 5
B. 10
C. 4

120. Combien de tours fait `Pour i allant de 5 à 3 Faire` ?

A. `0`
B. 3
C. 2

121. Comment écrire `Pour i allant de 1 à n` avec un Tant que ?

A. `Tant que i < n Faire`, sans rien d'autre
B. `i ← 1`, puis `Tant que i ≤ n Faire`, et `i ← i + 1` à la fin du bloc
C. `i ← 0`, puis `Tant que i ≤ n Faire`, sans faire avancer `i`

122. Dans `s ← 0`, puis `Pour i allant de 1 à n Faire s ← s + i`, quelle propriété est vraie après le tour i ?

A. `s` vaut i
B. `s` vaut n
C. `s` vaut 1 + 2 + … + i

123. On part de `t ← 1`, puis on fait 4 tours de `t ← 2 * t`. Que vaut `t` ?

A. 8
B. 16
C. 32

124. Quel jeu d'essai teste bien `Si age ≥ 18 Alors Écrire "majeur" Sinon Écrire "mineur"` ?

A. 20 et 30
B. 18 seulement
C. 17, 18 et 30

125. Pour savoir si 91 est premier, pourquoi la boucle qui s'arrête au premier diviseur fait-elle beaucoup moins de tours ?

A. Elle s'arrête dès qu'elle trouve 7, qui divise 91
B. Elle saute les nombres pairs
C. Elle teste seulement 2 et 3

Chapitre 5 — Les tableaux et les fonctions

126. Quel est l'indice du premier élément d'un tableau ?

A. 1
B. 0
C. -1

127. On a `T ← [5, 3, 8]`. Que vaut `T[longueur(T) - 1]` ?

A. 3
B. 5
C. 8

128. On a `notes = [12, 15, 9, 18]`, puis `notes[2] = 11`. Que vaut `notes` ?

A. `[12, 11, 9, 18]`
B. `[12, 15, 9, 11]`
C. `[12, 15, 11, 18]`

129. Quelle valeur de départ pour chercher le maximum d'un tableau `T` ?

A. `0`
B. `T[0]`
C. `longueur(T)`

130. Après `a = [1, 2]`, `b = a`, `b.append(3)`, que vaut `a` ?

A. `[1, 2, 3]`
B. `[1, 2]`
C. Une erreur

131. Comment faire une vraie copie du tableau `a` en Python ?

A. `b = a`
B. `b = a.copy()`
C. `b == a`

132. Pourquoi `for i in range(len(notes) + 1):` provoque-t-il une erreur ?

A. Parce que `range` n'accepte pas `+`
B. Parce qu'il manque un tour
C. Parce que le dernier indice vaut `len(notes)`, qui n'existe pas

133. Comment ajoute-t-on 14 à la fin du tableau `notes` en Python ?

A. `notes + 14`
B. `notes.append(14)`
C. `notes[len(notes)] = 14`

134. Dans `M[i][j]`, que désigne `i` ?

A. La ligne
B. La colonne
C. La valeur

135. Avec `notes = [[12, 15, 9, 18], [8, 11, 14, 10], [16, 13, 17, 15]]`, que vaut `notes[1][2]` ?

A. 15
B. 9
C. 14

136. Pour additionner toute la ligne 1 d'un tableau `M`, quel indice reste fixe ?

A. L'indice de ligne, `i = 1`
B. L'indice de colonne
C. Aucun

137. Comment écrit-on l'élément ligne 2, colonne 1 en Python ?

A. `M[2, 1]`
B. `M[2][1]`
C. `M[1][2]`

138. Un tableau de 3 lignes : quels indices de ligne existent ?

A. 0, 1, 2
B. 1, 2, 3
C. 0, 1, 2, 3

139. Quelle est la différence entre une fonction et une procédure ?

A. La fonction renvoie un résultat, la procédure fait une action sans rien renvoyer
B. La procédure renvoie un résultat, la fonction non
C. Il n'y en a aucune

140. Que renvoie `moyenne([10, 20])` ?

A. 30
B. 15.0
C. 10

141. Que se passe-t-il pour les lignes écrites après `Renvoyer` dans le même bloc ?

A. Elles s'exécutent après
B. Elles s'exécutent avant
C. Elles ne s'exécutent jamais

142. Une fonction écrit son résultat avec `Écrire` au lieu de `Renvoyer`. Quel est le problème ?

A. Aucun
B. La fonction plante
C. On ne peut pas réutiliser le résultat dans le programme

143. Quel est le bloc de la fonction `est_pair(n)` ?

A. `Écrire n mod 2`
B. `Renvoyer n mod 2 = 0`
C. `Renvoyer n div 2`

144. Comment s'appellent les valeurs qu'une fonction reçoit ?

A. Les paramètres
B. Les résultats
C. Les variables globales

145. Où existe une variable locale ?

A. Partout dans le programme
B. Seulement dans la fonction, pendant l'appel
C. Seulement dans le programme principal

146. On écrit `def f(): y = 5`, puis `print(y)` hors de la fonction. Que se passe-t-il ?

A. Le programme affiche 5
B. Le programme affiche 0
C. Une erreur, car `y` est locale à `f`

147. On a `x = 10` dans le programme principal et `def g(): return x + 1`. Que renvoie `g()` ?

A. 11
B. 1
C. Une erreur

148. Une procédure ajoute 1 point à chaque note du tableau qu'elle reçoit. Quel est le transfert ?

A. Entrée seule
B. Sortie seule
C. Entrée et sortie

149. Pourquoi vaut-il mieux passer une valeur en paramètre plutôt que de lire une variable globale ?

A. La fonction devient indépendante : on la teste seule et on la réutilise ailleurs
B. Parce que Python interdit les variables globales
C. Parce que c'est plus rapide

150. Que contient `T` après `T ← []`, `ajouter(T, 4)`, `ajouter(T, 9)` ?

A. `[9, 4]`
B. `[4, 9]`
C. `[]`

151. On a `total = 0`, puis `for x in [3, 5, 2]: total += x`. Que vaut `total` ?

A. 3
B. 5
C. 10

152. On copie `a = [1, 2]` élément par élément dans `c`, puis on fait `c[0] = 9`. Que vaut `a` ?

A. `[1, 2]`
B. `[9, 2]`
C. `[9]`

153. On construit avec deux boucles une grille de 2 lignes de 3 zéros. Qu'affiche `print(len(grille), len(grille[0]))` ?

A. `3 2`
B. `2 3`
C. `6 6`

154. En pseudo-code, comment récupérer les deux résultats de `bornes(T)` ?

A. `mini ← bornes(T)`, puis `maxi ← bornes(T)`
B. `bornes(T) ← (mini, maxi)`
C. `(mini, maxi) ← bornes(T)`

155. La procédure `remplir(T, n)` met n zéros dans un tableau vide reçu, sans le lire. Quel est le transfert pour `T` ?

A. Entrée seule
B. Sortie seule
C. Entrée et sortie

156. Pourquoi évite-t-on qu'une fonction modifie une variable globale avec `global` ?

A. Python l'interdit
B. Le changement se voit partout, et l'erreur est difficile à trouver
C. La fonction devient plus lente

Corrigé du QCM

N°	Réponse	Pourquoi
1	B	Un algorithme est une suite d'instructions précises. Il s'écrit en pseudo-code ou dans un langage comme Python.
2	C	C'est l'exécution séquentielle : chaque ligne s'exécute après celle du dessus.
3	D	Le pseudo-code sert à écrire et lire l'algorithme. Python est un vrai langage : on l'exécute pour tester.
4	B	Les lignes s'exécutent dans l'ordre. On ne peut pas afficher une variable avant de l'avoir calculée.
5	C	L'en-tête dit à quoi sert l'algorithme, ce qu'il reçoit et ce qu'il produit.
6	C	On calcule 10 - 3 = 7, puis 7 × 2 = 14. Chaque ligne part de la valeur de la ligne précédente.
7	B	La droite est calculée d'abord, puis le résultat remplace l'ancienne valeur de `x`.
8	B	Après `a ← b`, le 3 est perdu. Puis `b ← a` recopie 8. Pour échanger, il faut une variable `temp`.
9	D	La flèche va toujours vers une variable. On ne peut rien ranger dans le nombre 5.
10	C	`y ← x` copie la valeur une fois. Quand `x` change ensuite, `y` ne bouge pas.
11	C	En Python, `compteur = compteur + 1` ou sa forme courte `compteur += 1` fait la mise à jour.
12	B	`input` renvoie toujours du texte. Pour calculer, on le convertit avec `int` ou `float`.
13	B	On lit du texte avec `input`, puis on le change aussitôt en entier avec `int`.
14	B	Dans `print`, la virgule sépare les morceaux et ajoute un espace entre eux.
15	C	Sans virgule, Python ne sait pas séparer le texte et la variable : erreur de syntaxe.
16	C	Un prix peut avoir une virgule : on le convertit en réel avec `float`.
17	A	Une colonne par variable, une ligne par étape : on suit le chemin de l'algorithme.
18	B	`y` vaut 7, puis `x` vaut 4 × 7 = 28.
19	A	`b` vaut 10, puis `a` vaut 10 - 5 = 5.

N°	Réponse	Pourquoi
20	C	Une variable pas encore créée n'a pas de valeur. Écrire 0 serait faux.
21	C	La trace est la justification attendue : elle montre chaque étape du calcul.
22	B	Chaque structure se ferme par son « Fin de » : `Fin de Si`, `Fin de Pour`, `Fin de Tant que`.
23	C	L'en-tête dit à quoi sert l'algorithme, avant même de le lire.
24	B	`n -= 3` veut dire `n = n - 3`, soit 10 - 3 = 7.
25	A	Un nombre de fichiers est entier et jamais négatif.
26	B	Une température peut être négative : c'est un entier relatif.
27	C	Une comparaison vaut Vrai ou Faux. Ici, elle vaut Faux.
28	A	Une chaîne peut valoir « Oui », « OUI », « o »… Le booléen vaut seulement Vrai ou Faux.
29	C	Python attend un point. Avec des guillemets, ce serait une chaîne.
30	A	Un entier n'a pas de virgule : la moyenne devient 12.
31	C	Avec n bits, on écrit 2^n valeurs. $2^8 = 256$.
32	B	$2^{10} - 1 = 1023$, car on commence à 0.
33	C	256 ne tient pas sur 8 bits. Comme un compteur kilométrique, il repasse à 0.
34	B	0,1 et 0,2 sont rangés en binaire de façon approchée. Les deux petites erreurs s'ajoutent.
35	C	L'égalité exacte entre réels échoue souvent, par exemple `0.1 + 0.2 == 0.3` vaut `False`.
36	A	Les 65 536 valeurs sont partagées entre négatifs et positifs, 0 compris.
37	A	Sur 8 bits, on monte seulement à 255. Sur 16 bits, on monte à 65 535.
38	B	On compte à partir de 0 : P = 0, y = 1, t = 2.
39	C	L'espace est un caractère : B, T, S, espace, S, I, O.
40	B	Les indices vont de 0 à longueur - 1. L'indice `longueur(mot)` n'existe pas.
41	C	Cet indice est un cran trop loin : il n'existe pas.
42	A	On ne calcule jamais dessus, et un nombre perdrait le 0 du début.
43	C	Ce sont deux chaînes : le `+` les colle.

N°	Réponse	Pourquoi
44	A	`int` coupe la partie après la virgule. Il n'arrondit pas.
45	C	`int` n'accepte que l'écriture d'un entier. Pour un texte avec un point, on utilise `float`.
46	B	Python attend un point comme séparateur. La virgule n'est pas acceptée.
47	A	On ne colle pas un texte et un nombre. On change d'abord le nombre en texte avec `str`.
48	C	La division `/` donne toujours un réel, même quand elle tombe juste.
49	B	Une chaîne n'est pas un tableau. On peut lire `mot[0]`, pas le changer. Python s'arrête sur `TypeError`.
50	C	23 = 4 × 5 + 3 : le quotient est 5, le reste est 3.
51	B	`//` donne le quotient entier. `17 / 5` donnerait 3.4.
52	C	Un nombre pair a un reste nul dans la division par 2.
53	A	Sans parenthèses, la division passe avant l'addition.
54	C	`**` est la puissance : 2 multiplié 10 fois par lui-même.
55	B	`2800 div 60` = 46 et `2800 mod 60` = 40.
56	C	La multiplication s'écrit avec `*` et la puissance avec `**`. La puissance passe avant la multiplication.
57	A	On arrondit au plus près, avec 2 chiffres après la virgule.
58	B	On arrondit par excès : un carton incomplet compte quand même.
59	C	L'arrondi par défaut va vers le plus petit entier : -3 est plus petit que -2,5.
60	C	`randint(a, b)` inclut les deux bornes : 1 et 6 peuvent sortir.
61	A	`sqrt` est dans le module `math`. On l'importe avant de s'en servir.
62	B	Une formule produit les nombres. Ils semblent tirés au hasard, ce qui suffit pour un jeu.
63	A	La seconde condition est fausse. Avec ET, il faut que les deux soient vraies.
64	C	OU est vrai dès qu'au moins un des deux est vrai.
65	B	Le ou exclusif est vrai quand exactement un des deux est vrai. Ici, les deux le sont.

N°	Réponse	Pourquoi
66	C	Les deux conditions doivent être vraies : ET. Et « inclus » donne `>=` et `<=`.
67	A	Chaque comparaison s'écrit en entier. Et un jour ne peut pas être les deux à la fois.
68	B	En pseudo-code, la flèche `←` range et le `=` teste.
69	A	Les chaînes se comparent caractère par caractère, par leur code. `A` vient avant `B` : la comparaison est vraie.
70	C	« ou » donne `or`, « n'est plus vendu » donne `not vendu`, et le test d'égalité s'écrit `==`.
71	B	Les bits se numérotent de droite à gauche à partir de 0. Le bit le plus à droite vaut 1.
72	A	On teste avec ET et un masque qui a un seul 1, au rang 2. Un résultat non nul veut dire que le bit vaut 1.
73	C	OU met à 1 chaque rang où au moins un des deux vaut 1.
74	A	OUX avec 1 inverse le bit, OUX avec 0 le laisse comme il est.
75	C	ET avec l'inverse du masque éteint seulement le bit 0. Les autres bits ne bougent pas.
76	A	Pour éteindre seulement le bit 2, on écrit `x & ~0b0100`.
77	A	Seul le rang 1 vaut 1 dans les deux nombres.
78	B	La concaténation colle les deux chaînes telles quelles, espace compris.
79	C	On prend les indices 4, 5 et 6. La borne 7 est exclue.
80	A	`[:3]` prend les caractères d'indices 0, 1 et 2.
81	B	b = 0, o = 1, n = 2, j = 3.
82	A	-1 veut dire « pas trouvé ». 0 voudrait dire « trouvé au début ».
83	B	`find` renvoie une position. On écrit la comparaison `s.find("@") != -1`, qui vaut Vrai ou Faux.
84	C	`[-3:]` prend les 3 derniers caractères.
85	B	Les caractères se comparent par leur code : les chiffres, puis les majuscules, puis les minuscules.
86	C	« Différent » s'écrit `≠`, `<>` ou, en Python, `!=`.

N°	Réponse	Pourquoi
87	A	ET passe avant OU : `True and False` vaut `False`, puis `True or False` vaut `True`. Mettez des parenthèses pour être clair.
88	B	`randint` donne un entier, 0 ou 1. `uniform` donne un réel.
89	C	Avec un dé, le 6 sort aussi souvent que le 1.
90	B	Les conditions sont testées dans l'ordre. Le premier cas vrai gagne, les suivants sont ignorés.
91	C	-3 n'est pas strictement positif : on exécute le bloc Sinon.
92	B	7 ≥ 4 est vrai, donc le premier cas gagne. Il faut tester le cas le plus fort d'abord.
93	A	Python n'a pas de `Fin de Si`. Le retour à gauche ferme le bloc.
94	C	`elif` est l'abréviation de « else if ».
95	B	`i` prend les valeurs 3, 4, 5, 6 et 7.
96	C	`range(n)` va de 0 à n - 1.
97	A	La borne de droite est exclue : pour aller jusqu'à 10, on écrit 11.
98	C	1 + 2 + 3 + 4 = 10.
99	B	La préparation se fait une seule fois, avant la boucle.
100	C	Les caractères sont 1, 0, 1, 1, 0, 0, 1 : il y a quatre « 1 ».
101	A	On répète tant qu'une condition est vraie, sans savoir combien de fois.
102	B	On passe par 54, 28, puis 2. Comme 2 < 26, on sort.
103	A	On passe par 50, 25, 12, puis 6. Comme 6 n'est pas supérieur à 10, on sort.
104	C	Répéter exécute le bloc d'abord, puis teste la condition.
105	B	C'est une boucle infinie. Le bloc doit faire avancer la condition.
106	C	Python n'a pas de Répéter. `break` force la sortie quand la condition est vraie.
107	B	Il faut aussi tester `k < longueur(mot)` dans la condition.
108	A	0 ne change pas une somme. C'est l'élément neutre de l'addition.
109	A	Avec 0, le produit vaudrait toujours 0. 1 est l'élément neutre de la multiplication.
110	B	1 × 2 × 3 × 4 × 5 = 120.

N°	Réponse	Pourquoi
111	C	1 + 4 + 9 + 16 = 30.
112	C	On double chaque terme de 1 + 2 + … + 10 = 55. On obtient 2 × 55 = 110.
113	B	C'est la mise en forme finale. Dans la boucle, le total n'est pas encore complet.
114	A	On prépare l'accumulateur, on le met à jour à chaque tour, puis on présente le résultat.
115	B	La boucle intérieure fait 3 tours à chacun des 4 tours extérieurs : 4 × 3 = 12.
116	C	La boucle intérieure recommence depuis le début à chaque tour de la boucle extérieure.
117	C	Une ligne par tour de la boucle extérieure. La 1re a 1 étoile, la 5e en a 5.
118	B	Chaque case du résultat a sa propre somme.
119	A	On compte les nombres pairs : 2, 4, 6, 8 et 10.
120	A	La fin est plus petite que le début : la boucle ne fait aucun tour.
121	B	Le compteur part de 1, il est testé avant chaque tour, et il avance de 1 à la fin du tour.
122	C	C'est vrai au départ, et chaque tour le garde vrai : c'est le raisonnement par récurrence qui construit la boucle.
123	B	La suite double à chaque tour : 2, 4, 8, 16.
124	C	Une valeur par branche, plus la valeur limite 18.
125	A	91 = 7 × 13. La version qui teste tous les diviseurs fait 91 tests, celle qui s'arrête en fait 6.
126	B	Comme pour les chaînes, les indices commencent à 0.
127	C	`longueur(T) - 1` vaut 2 : c'est l'indice du dernier élément.
128	C	L'indice 2 est la 3e case : 9 est remplacé par 11.
129	B	On part d'une vraie valeur du tableau. Avec 0, un tableau de nombres négatifs donnerait un faux maximum.
130	A	`b = a` ne copie pas : `a` et `b` sont le même tableau.
131	B	`.copy()` crée un nouveau tableau, indépendant de `a`.
132	C	Les indices vont de 0 à `len(notes) - 1`.

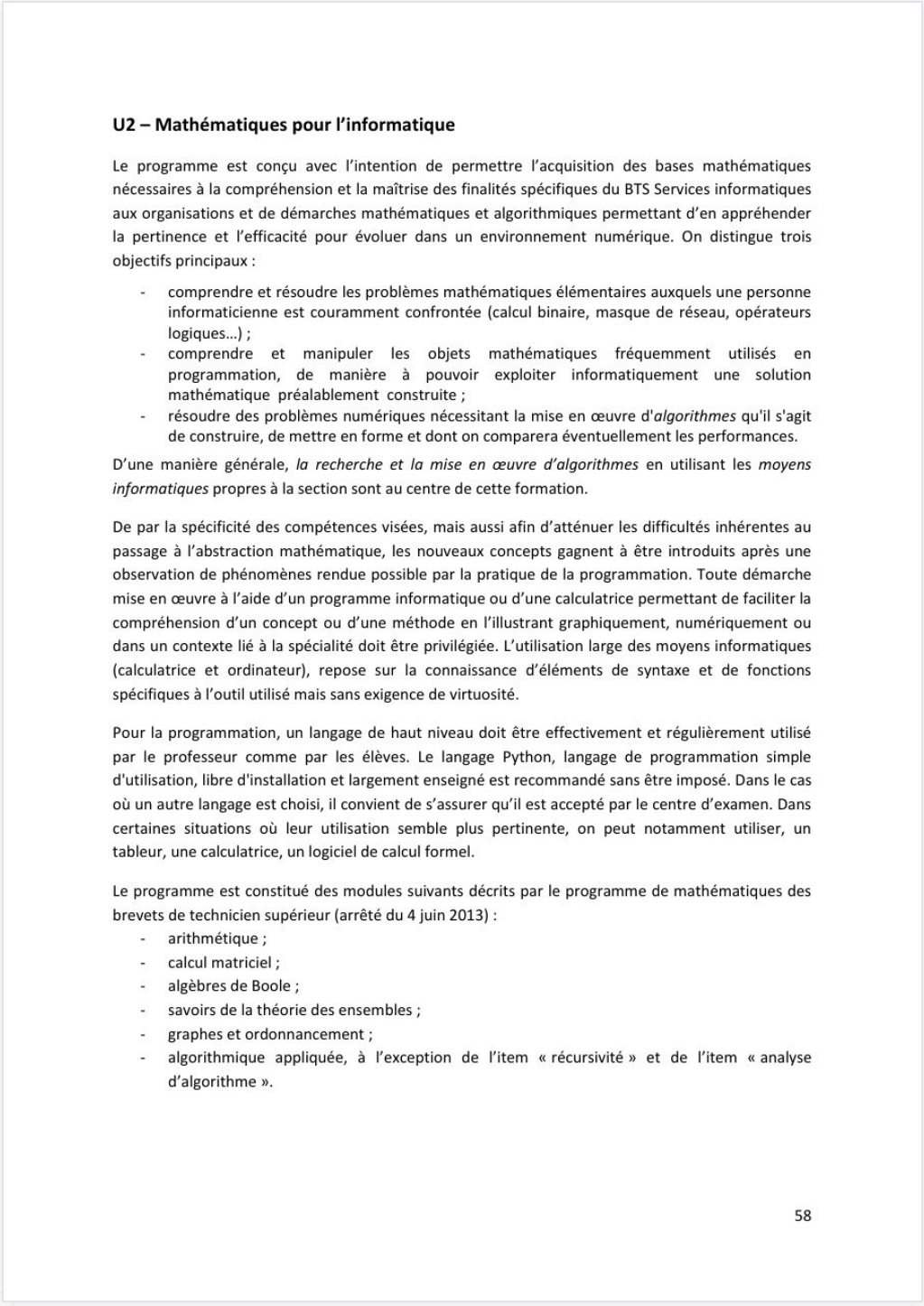
N°	Réponse	Pourquoi
133	B	`append` ajoute un élément à la fin. L'indice `len(notes)` n'existe pas encore.
134	A	L'ordre est toujours ligne, puis colonne.
135	C	Ligne d'indice 1 : `[8, 11, 14, 10]`. Colonne d'indice 2 : 14.
136	A	On fixe la ligne et on fait varier `j`, l'indice de colonne.
137	B	Python attend deux paires de crochets, la ligne d'abord.
138	A	Les indices commencent à 0 et s'arrêtent au nombre de lignes moins 1.
139	A	Une procédure peut afficher, par exemple. Une fonction rend une valeur qu'on peut réutiliser.
140	B	(10 + 20) / 2 = 15.0. La division `/` donne un réel.
141	C	`Renvoyer` termine la fonction.
142	C	La fonction affiche mais ne rend rien. Le programme ne peut pas ranger le résultat dans une variable.
143	B	La fonction renvoie un booléen : Vrai si le reste de la division par 2 est nul.
144	A	Les paramètres s'écrivent entre parenthèses, après le nom de la fonction.
145	B	Elle disparaît à la fin de l'appel.
146	C	`y` n'existe pas en dehors de la fonction.
147	A	Une fonction peut lire une variable globale.
148	C	Elle reçoit le tableau et le modifie.
149	A	Tout ce dont la fonction a besoin entre par ses paramètres, tout ce qu'elle produit sort par `Renvoyer`.
150	B	`ajouter` met chaque valeur à la fin du tableau.
151	C	`for x in T` parcourt directement les valeurs : 3 + 5 + 2 = 10.
152	A	La copie élément par élément crée un nouveau tableau : modifier `c` ne change pas `a`.
153	B	`len(grille)` compte les lignes, `len(grille[0])` les cases d'une ligne.
154	C	On écrit les deux variables entre parenthèses, à gauche de la flèche, dans le même ordre que `Renvoyer`.
155	B	`T` reçoit un résultat sans que la procédure lise ce qu'il contenait.

N°	Réponse	Pourquoi
156	B	Bonne habitude : tout entre par les paramètres et tout sort par `Renvoyer`.

Les sujets officiels

Annexe A1 — 1. Le référentiel du BTS SIO : les modules de maths

Sont joints à ce PDF : les pages citées de « Referentiel BTS SIO - version 2024 (arrete 8 juillet 2024).pdf ». Ouvrez-les depuis le trombone des pièces jointes de votre lecteur PDF.



U2 – Mathématiques pour l'informatique

Le programme est conçu avec l'intention de permettre l'acquisition des bases mathématiques nécessaires à la compréhension et la maîtrise des finalités spécifiques du BTS Services informatiques aux organisations et de démarches mathématiques et algorithmiques permettant d'en appréhender la pertinence et l'efficacité pour évoluer dans un environnement numérique. On distingue trois objectifs principaux :

- comprendre et résoudre les problèmes mathématiques élémentaires auxquels une personne informaticienne est couramment confrontée (calcul binaire, masque de réseau, opérateurs logiques…) ;
- comprendre et manipuler les objets mathématiques fréquemment utilisés en programmation, de manière à pouvoir exploiter informatiquement une solution mathématique préalablement construite ;
- résoudre des problèmes numériques nécessitant la mise en œuvre d'*algorithmes* qu'il s'agit de construire, de mettre en forme et dont on comparera éventuellement les performances.

D'une manière générale, *la recherche et la mise en œuvre d'algorithmes* en utilisant les *moyens informatiques* propres à la section sont au centre de cette formation.

De par la spécificité des compétences visées, mais aussi afin d'atténuer les difficultés inhérentes au passage à l'abstraction mathématique, les nouveaux concepts gagnent à être introduits après une observation de phénomènes rendue possible par la pratique de la programmation. Toute démarche mise en œuvre à l'aide d'un programme informatique ou d'une calculatrice permettant de faciliter la compréhension d'un concept ou d'une méthode en l'illustrant graphiquement, numériquement ou dans un contexte lié à la spécialité doit être privilégiée. L'utilisation large des moyens informatiques (calculatrice et ordinateur), repose sur la connaissance d'éléments de syntaxe et de fonctions spécifiques à l'outil utilisé mais sans exigence de virtuosité.

Pour la programmation, un langage de haut niveau doit être effectivement et régulièrement utilisé par le professeur comme par les élèves. Le langage Python, langage de programmation simple d'utilisation, libre d'installation et largement enseigné est recommandé sans être imposé. Dans le cas où un autre langage est choisi, il convient de s'assurer qu'il est accepté par le centre d'examen. Dans certaines situations où leur utilisation semble plus pertinente, on peut notamment utiliser, un tableur, une calculatrice, un logiciel de calcul formel.

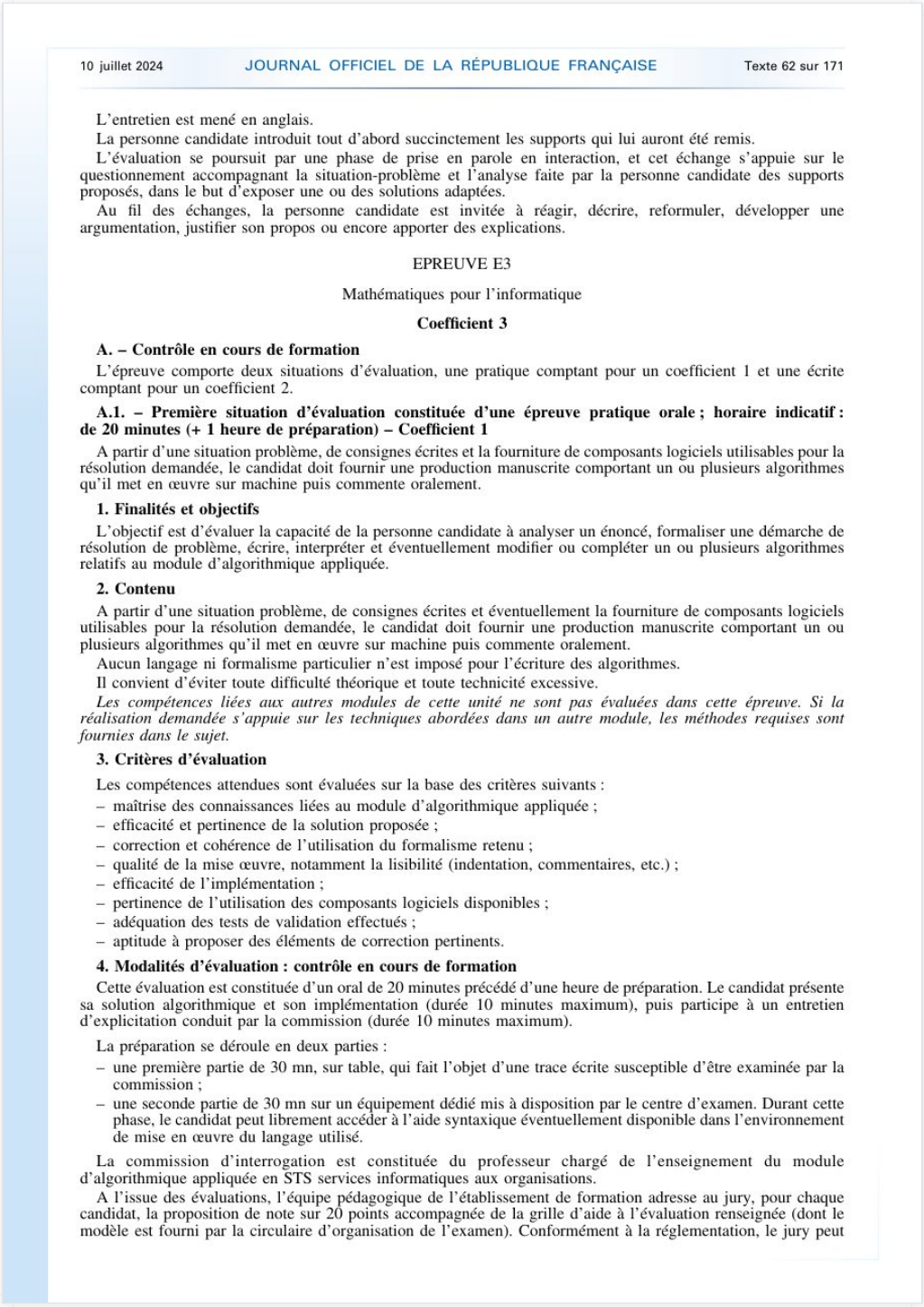
Le programme est constitué des modules suivants décrits par le programme de mathématiques des brevets de technicien supérieur (arrêté du 4 juin 2013) :

- arithmétique ;
- calcul matriciel ;
- algèbres de Boole ;
- savoirs de la théorie des ensembles ;
- graphes et ordonnancement ;
- algorithmique appliquée, à l'exception de l'item « récursivité » et de l'item « analyse d'algorithme ».

58

Annexe A2 — 2. L'épreuve E3 — Mathématiques pour l'informatique

Sont joints à ce PDF : les pages citées de « Referentiel BTS SIO - version 2024 (arrete 8 juillet 2024).pdf ». Ouvrez-les depuis le trombone des pièces jointes de votre lecteur PDF.



L'entretien est mené en anglais.

La personne candidate introduit tout d'abord succinctement les supports qui lui auront été remis.

L'évaluation se poursuit par une phase de prise en parole en interaction, et cet échange s'appuie sur le questionnement accompagnant la situation-problème et l'analyse faite par la personne candidate des supports proposés, dans le but d'exposer une ou des solutions adaptées.

Au fil des échanges, la personne candidate est invitée à réagir, décrire, reformuler, développer une argumentation, justifier son propos ou encore apporter des explications.

EPREUVE E3

Mathématiques pour l'informatique

Coefficient 3

A. – Contrôle en cours de formation

L'épreuve comporte deux situations d'évaluation, une pratique comptant pour un coefficient 1 et une écrite comptant pour un coefficient 2.

A.1. – Première situation d'évaluation constituée d'une épreuve pratique orale ; horaire indicatif : de 20 minutes (+ 1 heure de préparation) – Coefficient 1

A partir d'une situation problème, de consignes écrites et la fourniture de composants logiciels utilisables pour la résolution demandée, le candidat doit fournir une production manuscrite comportant un ou plusieurs algorithmes qu'il met en œuvre sur machine puis commente oralement.

1. Finalités et objectifs

L'objectif est d'évaluer la capacité de la personne candidate à analyser un énoncé, formaliser une démarche de résolution de problème, écrire, interpréter et éventuellement modifier ou compléter un ou plusieurs algorithmes relatifs au module d'algorithmique appliquée.

2. Contenu

A partir d'une situation problème, de consignes écrites et éventuellement la fourniture de composants logiciels utilisables pour la résolution demandée, le candidat doit fournir une production manuscrite comportant un ou plusieurs algorithmes qu'il met en œuvre sur machine puis commente oralement.

Aucun langage ni formalisme particulier n'est imposé pour l'écriture des algorithmes.

Il convient d'éviter toute difficulté théorique et toute technicité excessive.

Les compétences liées aux autres modules de cette unité ne sont pas évaluées dans cette épreuve. Si la réalisation demandée s'appuie sur les techniques abordées dans un autre module, les méthodes requises sont fournies dans le sujet.

3. Critères d'évaluation

Les compétences attendues sont évaluées sur la base des critères suivants :

- maîtrise des connaissances liées au module d'algorithmique appliquée ;
- efficacité et pertinence de la solution proposée ;
- correction et cohérence de l'utilisation du formalisme retenu ;
- qualité de la mise œuvre, notamment la lisibilité (indentation, commentaires, etc.) ;
- efficacité de l'implémentation ;
- pertinence de l'utilisation des composants logiciels disponibles ;
- adéquation des tests de validation effectués ;
- aptitude à proposer des éléments de correction pertinents.

4. Modalités d'évaluation : contrôle en cours de formation

Cette évaluation est constituée d'un oral de 20 minutes précédé d'une heure de préparation. Le candidat présente sa solution algorithmique et son implémentation (durée 10 minutes maximum), puis participe à un entretien d'explicitation conduit par la commission (durée 10 minutes maximum).

La préparation se déroule en deux parties :

- une première partie de 30 mn, sur table, qui fait l'objet d'une trace écrite susceptible d'être examinée par la commission ;
- une seconde partie de 30 mn sur un équipement dédié mis à disposition par le centre d'examen. Durant cette phase, le candidat peut librement accéder à l'aide syntaxique éventuellement disponible dans l'environnement de mise en œuvre du langage utilisé.

La commission d'interrogation est constituée du professeur chargé de l'enseignement du module d'algorithmique appliquée en STS services informatiques aux organisations.

A l'issue des évaluations, l'équipe pédagogique de l'établissement de formation adresse au jury, pour chaque candidat, la proposition de note sur 20 points accompagnée de la grille d'aide à l'évaluation renseignée (dont le modèle est fourni par la circulaire d'organisation de l'examen). Conformément à la réglementation, le jury peut

Page 11

demander à avoir communication des dossiers d'évaluation des candidats. Ces documents sont tenus à la disposition du jury et de l'autorité rectorale pour la session considérée jusqu'à la session suivante.

A.2. – Deuxième situation d'évaluation constituée d'une évaluation écrite ; horaire indicatif : 55 minutes – Coefficient 2

1. Finalités et objectifs

Cette épreuve vise à évaluer les compétences acquises par la personne candidate dans le cadre de l'unité de Mathématiques pour l'informatique. Elle a pour objectifs d'apprécier les connaissances et leur mobilisation dans des situations variées, de vérifier l'aptitude au raisonnement et les capacités à analyser un problème, à justifier l'emploi des méthodes utilisées et à interpréter les résultats et d'apprécier la rigueur et la précision de son expression écrite et orale.

2. Contenu

L'épreuve s'appuie sur un sujet comportant un ou deux exercices couvrant une partie significative des modules autres que l'algorithmique appliquée de l'unité de mathématiques pour l'informatique. Chaque exercice présente une situation concrète, en relation avec les activités professionnelles visées par le diplôme.

3. Critères d'évaluation

Les compétences attendues sont évaluées sur la base des critères suivants :

– maîtrise des connaissances figurant au programme ;
– pertinence des sources d'information mobilisées ;
– adaptation de la stratégie choisie au problème à résoudre ;
– efficacité dans la mise en œuvre de cette stratégie ;
– rigueur et pertinence dans l'utilisation des savoir-faire figurant au programme de mathématiques ;
– cohérence de l'argumentation employée ;
– aptitude à analyser un résultat avec pertinence ;
– qualité d'expression écrite ou orale.

4. Modalités d'évaluation : contrôle en cours de formation

Le contrôle en cours de formation comporte une situation d'évaluation qui se déroule au cours du deuxième semestre de la deuxième année. L'épreuve dure cinquante-cinq minutes.

Elle est constituée d'un ou deux exercices avec des questions de difficulté progressive.

L'emploi de la calculatrice est autorisé selon la législation en vigueur. En fonction des besoins et afin de ne pas introduire de discriminations liées aux performances de la calculatrice employée, certaines formules de base peuvent être rappelées en tête du sujet (relations fonctionnelles, suites arithmétiques et géométriques, etc.).

La correction est assurée par le professeur de mathématiques enseignant en STS services informatiques aux organisations.

A l'issue des évaluations, l'équipe pédagogique de l'établissement de formation adresse au jury, pour chaque candidat, la proposition de note sur 20 points accompagnée de la grille d'aide à l'évaluation renseignée (dont le modèle est fourni par la circulaire d'organisation de l'examen). Conformément à la réglementation, le jury peut demander à avoir communication des dossiers d'évaluation des candidats. Ces documents sont tenus à la disposition du jury et de l'autorité rectorale pour la session considérée jusqu'à la session suivante.

B. – Forme ponctuelle : épreuve écrite de 2 heures

Pour les candidats non éligibles au CCF, l'épreuve ponctuelle prend la forme d'une épreuve écrite de 2 heures sur un sujet comportant deux ou trois exercices couvrant une partie significative du programme de l'unité, pour un total de 20 points. L'un des exercices évalue la capacité « programmer » en lien avec le module d'algorithmique appliquée.

EPREUVE E4

Culture économique juridique et managériale pour l'informatique

Coefficient 3

1. Finalités et objectifs

Cette épreuve vise à évaluer les compétences acquises par la personne candidate dans le cadre du bloc « Culture économique, juridique et managériale » ainsi que sa capacité à les mobiliser pour éclairer des problématiques de gestion rencontrées par des prestataires de services informatiques à travers un questionnement croisant les champs disciplinaires économique, juridique et managérial.

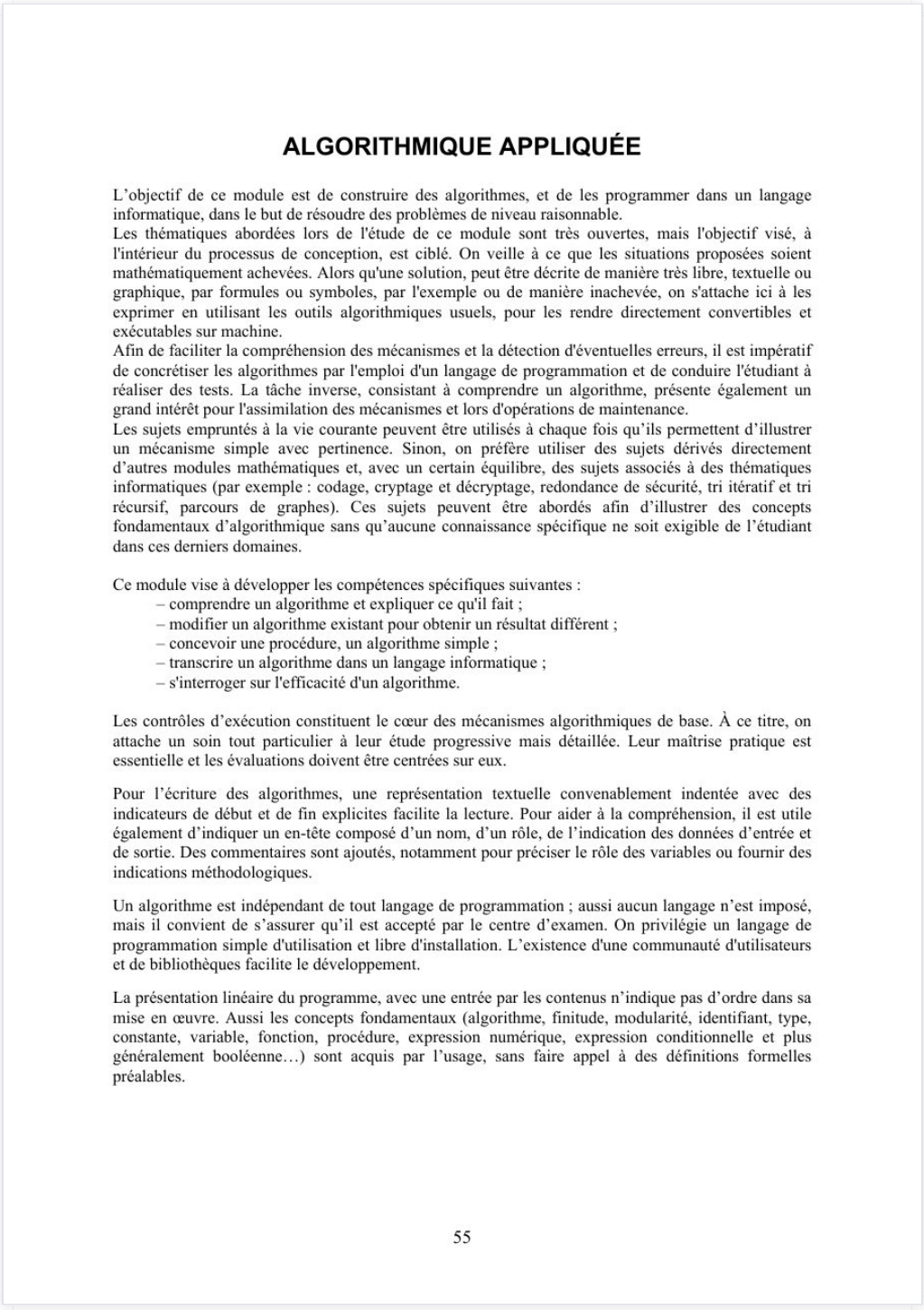
2. Modalités et critères d'évaluation

L'épreuve vise à évaluer les compétences de la personne candidate à :

– analyser des situations auxquelles un prestataire de services informatiques est confronté ;
– exploiter une base documentaire économique, juridique et managériale ;

Annexe A3 — 3. Le programme du module « Algorithmique appliquée »

Sont joints à ce PDF : le document complet (BTS_ProgrammeMathematiques.pdf). Ouvrez-les depuis le trombone des pièces jointes de votre lecteur PDF.



ALGORITHMIQUE APPLIQUÉE

L'objectif de ce module est de construire des algorithmes, et de les programmer dans un langage informatique, dans le but de résoudre des problèmes de niveau raisonnable.
Les thématiques abordées lors de l'étude de ce module sont très ouvertes, mais l'objectif visé, à l'intérieur du processus de conception, est ciblé. On veille à ce que les situations proposées soient mathématiquement achevées. Alors qu'une solution, peut être décrite de manière très libre, textuelle ou graphique, par formules ou symboles, par l'exemple ou de manière inachevée, on s'attache ici à les exprimer en utilisant les outils algorithmiques usuels, pour les rendre directement convertibles et exécutables sur machine.
Afin de faciliter la compréhension des mécanismes et la détection d'éventuelles erreurs, il est impératif de concrétiser les algorithmes par l'emploi d'un langage de programmation et de conduire l'étudiant à réaliser des tests. La tâche inverse, consistant à comprendre un algorithme, présente également un grand intérêt pour l'assimilation des mécanismes et lors d'opérations de maintenance.
Les sujets empruntés à la vie courante peuvent être utilisés à chaque fois qu'ils permettent d'illustrer un mécanisme simple avec pertinence. Sinon, on préfère utiliser des sujets dérivés directement d'autres modules mathématiques et, avec un certain équilibre, des sujets associés à des thématiques informatiques (par exemple : codage, cryptage et décryptage, redondance de sécurité, tri itératif et tri récursif, parcours de graphes). Ces sujets peuvent être abordés afin d'illustrer des concepts fondamentaux d'algorithmique sans qu'aucune connaissance spécifique ne soit exigible de l'étudiant dans ces derniers domaines.

Ce module vise à développer les compétences spécifiques suivantes :
- comprendre un algorithme et expliquer ce qu'il fait ;
- modifier un algorithme existant pour obtenir un résultat différent ;
- concevoir une procédure, un algorithme simple ;
- transcrire un algorithme dans un langage informatique ;
- s'interroger sur l'efficacité d'un algorithme.

Les contrôles d'exécution constituent le cœur des mécanismes algorithmiques de base. À ce titre, on attache un soin tout particulier à leur étude progressive mais détaillée. Leur maîtrise pratique est essentielle et les évaluations doivent être centrées sur eux.

Pour l'écriture des algorithmes, une représentation textuelle convenablement indentée avec des indicateurs de début et de fin explicites facilite la lecture. Pour aider à la compréhension, il est utile également d'indiquer un en-tête composé d'un nom, d'un rôle, de l'indication des données d'entrée et de sortie. Des commentaires sont ajoutés, notamment pour préciser le rôle des variables ou fournir des indications méthodologiques.

Un algorithme est indépendant de tout langage de programmation ; aussi aucun langage n'est imposé, mais il convient de s'assurer qu'il est accepté par le centre d'examen. On privilégie un langage de programmation simple d'utilisation et libre d'installation. L'existence d'une communauté d'utilisateurs et de bibliothèques facilite le développement.

La présentation linéaire du programme, avec une entrée par les contenus n'indique pas d'ordre dans sa mise en œuvre. Aussi les concepts fondamentaux (algorithme, finitude, modularité, identifiant, type, constante, variable, fonction, procédure, expression numérique, expression conditionnelle et plus généralement booléenne…) sont acquis par l'usage, sans faire appel à des définitions formelles préalables.

55

CONTENUS	CAPACITÉS ATTENDUES	COMMENTAIRES
Types de données Types simples : entier naturel, entier relatif, réel, booléen. Chaîne de caractères.	• Connaître les modes de codage et gérer les différences entre données mathématiques et informatiques : – domaine de valeurs ; – représentation exacte ou approchée, précision.	Il n'est pas nécessaire de connaître la représentation exacte des données en machine, notamment en ce qui concerne les flottants. On évite de considérer une chaîne comme un tableau de caractères.
Tableaux de données : – de type homogène à une ou deux dimensions ; – à deux dimensions dans lequel, soit les lignes soit les colonnes, peuvent être de types différents. Procédure et fonction : – paramètres d'entrée ; – valeur(s) retournée(s) par une fonction ; – variables globales ou locales.	• Construire un tableau. • Traiter les données d'un tableau : – accéder à ses différents éléments en lecture et en écriture ; – traiter les éléments d'une ligne ou d'une colonne ; – copier un tableau. • Gérer les transferts en entrée seule, en sortie seule, en entrée et sortie. • Utiliser les variables globales et locales.	On adapte la construction et l'exploitation de ces tableaux aux possibilités de l'outil informatique utilisé. Les structures de données et les objets ne sont pas au programme de mathématiques : ils figurent dans ceux des enseignements professionnels. Sans aborder la programmation objet, les concepts de modularité et d'interface doivent être connus.
Instructions élémentaires Lecture, écriture. Affectation, affectation récursive. **Opérateurs** Opérateurs numériques : addition, soustraction, multiplication, division, exponentiation, quotient et reste de la division entière, signe. Fonctions mathématiques usuelles.	• Saisir une donnée depuis le clavier, manipuler les variables et afficher sur l'écran. • Gérer la chronologie des valeurs contenues dans les variables et produire la trace d'un algorithme. • Transformer une expression mathématique en expression numérique en ligne et réciproquement.	Les fichiers ne sont pas au programme de mathématiques. En standard, la mise en forme des affichages est limitée à l'utilisation de séparateurs et de changements de ligne. Sinon, les outils nécessaires sont fournis et décrits. La gestion des pointeurs n'est pas au programme. « Fonctions mathématiques usuelles » doit être entendu au sens informatique et inclure les fonctions d'arrondi, ainsi qu'un générateur de nombres pseudo-aléatoires uniforme dans un intervalle.

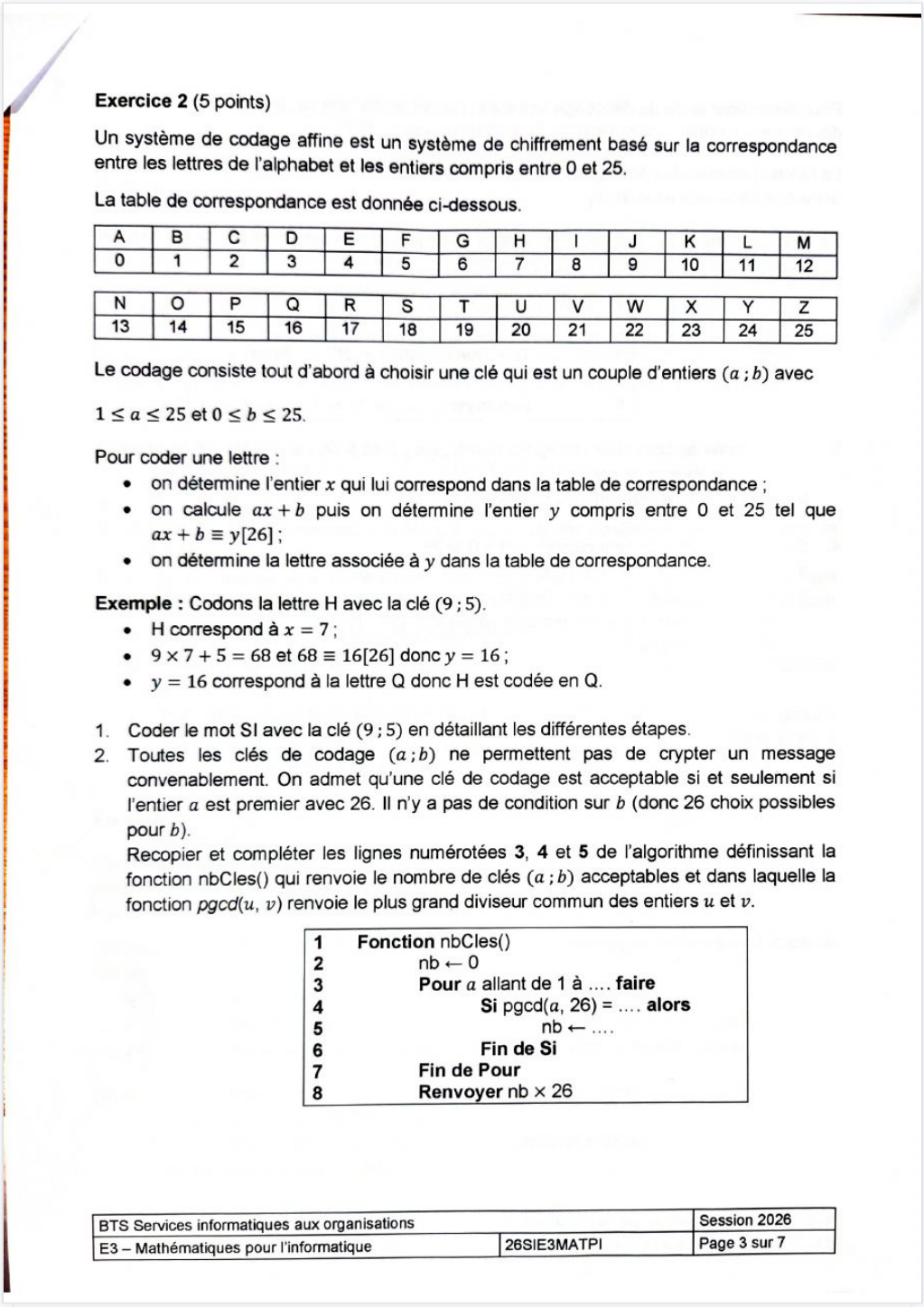
Opérateurs de comparaison : =, <> ou !=, <, <=, >, >=. Opérateurs booléens : non, et, ou, oux.	• Traduire la condition (éventuellement composée) d'une itération (tant que / répéter jusqu'à ce que) ou d'une alternative (si) sous forme d'expression booléenne.	On compare soit des valeurs numériques, soit des chaînes de caractères.
Opérateurs booléens bit à bit.	• Appliquer des opérations booléennes sur les bits.	On interprète notamment en termes de masque, de mise à un, de mise à zéro, de changement d'état.
Opérateur de chaînes : concaténation. Fonctions permettant l'extraction en début, milieu ou fin, la recherche d'un motif.	• Construire et manipuler des chaînes et construire les messages affichés à l'écran.	L'usage d'expressions régulières simples est possible, mais l'étude des expressions régulières est hors programme.
Transtypage	• Gérer les différents types de données et effectuer des conversions d'un type vers un autre.	D'autres instructions, fonctions ou procédures peuvent être introduites dans l'écriture d'algorithmes. Les descriptions sémantiques et syntaxiques précises sont alors mises à disposition de l'étudiant.
Structures de contrôle et d'exécution. Exécution séquentielle. Exécution à structure conditionnelle (si-alors-sinon). Exécution à structure itérative (pour) et (tant que / répéter jusqu'à ce que).	• Mettre en œuvre ces structures.	Afin d'en maîtriser le fonctionnement, les structures d'exécution sont elles-mêmes présentées sous forme d'algorithmes.
Construction des structures itératives : raisonnement par récurrence, initialisation, mise à jour itérative, calcul itératif, mise en forme finale.	• Gérer une itération en distinguant la préparation, l'itération elle-même et la mise en forme finale.	Le raisonnement par récurrence n'a pas à être évalué pour des démonstrations. Il est introduit pour servir de base à une construction des itérations. Le calcul itératif est souvent récursif

Symboles $\sum_{i=}$ et $\prod_{i=}$, traduction algorithmique.	• Gérer une somme ou un produit d'un nombre variable d'opérandes dépendant d'un paramètre.	Généralement la variable affectée récursivement est initialisée à l'élément neutre de l'opération utilisée, avant d'entrer dans l'itération.
Structures imbriquées	• Gérer des structures imbriquées.	On traite également des exemples où les éléments de contrôle des structures internes dépendent de ceux des structures externes. Le nombre d'imbrications n'est pas limité, sauf pour les itérations en dépendance, où on se limite à deux. On évite les excès de complexité, ainsi que les constructions ne correspondant pas à un besoin concret.
	• Traiter un produit matriciel.	La formule de calcul des coefficients d'un produit est donnée.
Récursivité. Nécessité d'un test. Nécessité de cas particuliers résolus sans appel à la récursivité. Finitude.	• Mettre en œuvre ou exploiter une fonction ou une procédure récursive simple.	On peut traiter des exemples de récursivité terminale, de conversion en algorithme non récursif, de récursivité non terminale. On ne présente pas de récursivité mutuelle entre plusieurs procédures.
Analyse d'algorithmes		Ce paragraphe se veut être une simple sensibilisation aux notions de complexité, de correction, de recherche d'erreur et de finalité d'un algorithme. En cas d'évaluation portant sur l'un de ces thèmes, on apporte suffisamment d'indications pour limiter les prérequis.
Notions de complexité temporelle et spatiale.	• Calculer un nombre minimal ou maximal d'opérations significatives ou la taille globale de données choisies.	On présente des variantes produisant les mêmes résultats avec des complexités très différentes. Aucune connaissance théorique n'est exigible.
Validation et débogage.	• Procéder à un suivi de variables par la production d'une trace ou l'utilisation de jeux d'essai pour le test d'un algorithme et la recherche d'erreur.	On propose des algorithmes délibérément erronés dont les défauts sont repérés puis débogués. Selon le langage choisi, on peut montrer les fonctions permettant

		le suivi de variables, le débogage pas à pas. Afin de mieux sensibiliser à certains risques, on peut présenter des cas d'effets indésirables (effets de bord, évaluation partielle lors de calcul d'expressions booléennes, débordements ou approximations numériques, transtypage, utilisation d'indices hors domaine,…) et leurs conséquences spectaculaires. Aucune théorie n'est au programme.
Interprétation d'algorithmes.	• Savoir reconstituer le rôle d'un algorithme.	L'ajout d'une ou plusieurs procédures ou fonctions à un ensemble interdépendant peut constituer une excellente base. On se limite à des cas simples.

Annexe A4 — BTS SIO 2026 — Le nombre de clés de codage

Sont joints à ce PDF : les pages citées de « bts_sio_2026.pdf ». Ouvrez-les depuis le trombone des pièces jointes de votre lecteur PDF.



Exercice 2 (5 points)

Un système de codage affine est un système de chiffrement basé sur la correspondance entre les lettres de l'alphabet et les entiers compris entre 0 et 25.

La table de correspondance est donnée ci-dessous.

A	B	C	D	E	F	G	H	I	J	K	L	M
0	1	2	3	4	5	6	7	8	9	10	11	12

N	O	P	Q	R	S	T	U	V	W	X	Y	Z
13	14	15	16	17	18	19	20	21	22	23	24	25

Le codage consiste tout d'abord à choisir une clé qui est un couple d'entiers $(a\,;b)$ avec

$1 \leq a \leq 25$ et $0 \leq b \leq 25$.

Pour coder une lettre :

- on détermine l'entier x qui lui correspond dans la table de correspondance ;
- on calcule $ax+b$ puis on détermine l'entier y compris entre 0 et 25 tel que $ax+b \equiv y[26]$;
- on détermine la lettre associée à y dans la table de correspondance.

Exemple : Codons la lettre H avec la clé $(9\,;5)$.

- H correspond à $x=7$;
- $9\times 7+5=68$ et $68 \equiv 16[26]$ donc $y=16$;
- $y=16$ correspond à la lettre Q donc H est codée en Q.

1. Coder le mot SI avec la clé $(9\,;5)$ en détaillant les différentes étapes.
2. Toutes les clés de codage $(a\,;b)$ ne permettent pas de crypter un message convenablement. On admet qu'une clé de codage est acceptable si et seulement si l'entier a est premier avec 26. Il n'y a pas de condition sur b (donc 26 choix possibles pour b).
   Recopier et compléter les lignes numérotées **3**, **4** et **5** de l'algorithme définissant la fonction nbCles() qui renvoie le nombre de clés $(a\,;b)$ acceptables et dans laquelle la fonction *pgcd*(u, v) renvoie le plus grand diviseur commun des entiers u et v.

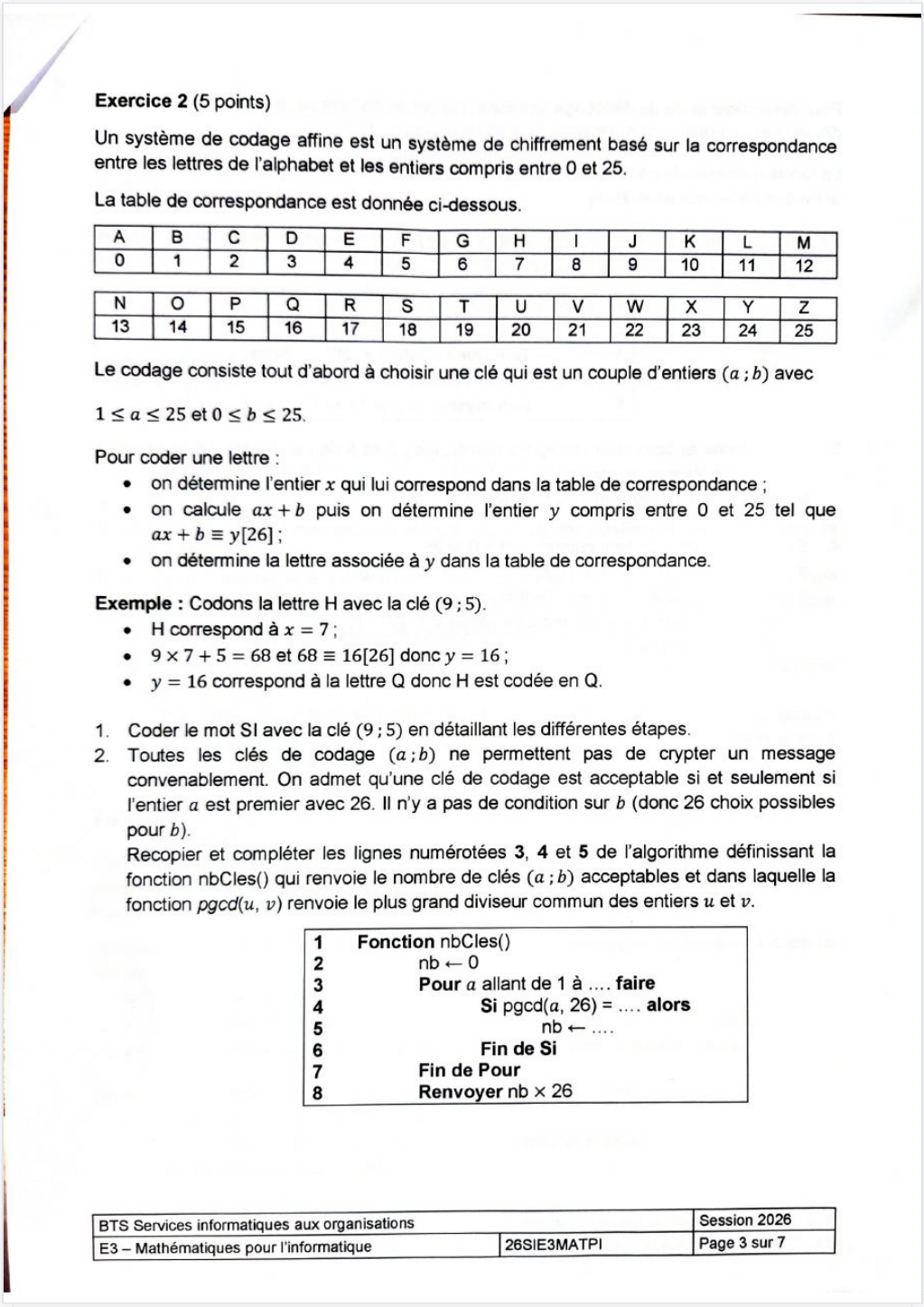
```
1  Fonction nbCles()
2      nb ← 0
3      Pour a allant de 1 à .... faire
4          Si pgcd(a, 26) = .... alors
5              nb ← ....
6          Fin de Si
7      Fin de Pour
8      Renvoyer nb × 26
```

BTS Services informatiques aux organisations		Session 2026
E3 – Mathématiques pour l'informatique	26SIE3MATPI	Page 3 sur 7

Page 3

Annexe A5 — BTS SIO 2026 — La clé de décodage

Sont joints à ce PDF : les pages citées de « bts_sio_2026.pdf ». Ouvrez-les depuis le trombone des pièces jointes de votre lecteur PDF.



Exercice 2 (5 points)

Un système de codage affine est un système de chiffrement basé sur la correspondance entre les lettres de l'alphabet et les entiers compris entre 0 et 25.

La table de correspondance est donnée ci-dessous.

A	B	C	D	E	F	G	H	I	J	K	L	M
0	1	2	3	4	5	6	7	8	9	10	11	12

N	O	P	Q	R	S	T	U	V	W	X	Y	Z
13	14	15	16	17	18	19	20	21	22	23	24	25

Le codage consiste tout d'abord à choisir une clé qui est un couple d'entiers $(a\,;b)$ avec

$1 \leq a \leq 25$ et $0 \leq b \leq 25$.

Pour coder une lettre :

- on détermine l'entier x qui lui correspond dans la table de correspondance ;
- on calcule $ax+b$ puis on détermine l'entier y compris entre 0 et 25 tel que $ax+b \equiv y[26]$;
- on détermine la lettre associée à y dans la table de correspondance.

Exemple : Codons la lettre H avec la clé $(9\,;5)$.

- H correspond à $x=7$;
- $9\times 7+5=68$ et $68 \equiv 16[26]$ donc $y=16$;
- $y=16$ correspond à la lettre Q donc H est codée en Q.

1. Coder le mot SI avec la clé $(9\,;5)$ en détaillant les différentes étapes.
2. Toutes les clés de codage $(a\,;b)$ ne permettent pas de crypter un message convenablement. On admet qu'une clé de codage est acceptable si et seulement si l'entier a est premier avec 26. Il n'y a pas de condition sur b (donc 26 choix possibles pour b).
   Recopier et compléter les lignes numérotées **3**, **4** et **5** de l'algorithme définissant la fonction nbCles() qui renvoie le nombre de clés $(a\,;b)$ acceptables et dans laquelle la fonction *pgcd*(u, v) renvoie le plus grand diviseur commun des entiers u et v.

```
1  Fonction nbCles()
2      nb ← 0
3      Pour a allant de 1 à .... faire
4          Si pgcd(a, 26) = .... alors
5              nb ← ....
6          Fin de Si
7      Fin de Pour
8      Renvoyer nb × 26
```

BTS Services informatiques aux organisations		Session 2026
E3 – Mathématiques pour l'informatique	26SIE3MATPI	Page 3 sur 7

Page 3

Pour déterminer la clé de décodage associée à la clé de codage $(a\,;b)$, il est nécessaire de déterminer un entier c compris entre 0 et 25 tel que $ac \equiv 1[26]$.

La fonction *inverse* de paramètre a donnée ci-dessous renvoie le plus petit entier c compris entre 0 et 25 tel que $ac \equiv 1[26]$.

La fonction « reste$(u,\ n)$ » qui est utilisée ci-dessous renvoie le reste de la division euclidienne d'un entier naturel u par un entier naturel non nul n.

```
1  Fonction inverse(a)
2        c ← 1
3        Tant que reste(a × c, 26) .... faire
4              c ← c + 1
5        Renvoyer ....
```

3. a. Recopier et compléter les lignes numérotées **3** et **5** de l'algorithme définissant la fonction *inverse* de paramètre a.
   b. Déterminer l'entier renvoyé par *inverse*(9).

4. Soient x et y deux entiers compris entre 0 et 25.
   a. Montrer que si $9x + 5 \equiv y[26]$ alors $x + 15 \equiv 3y[26]$.
   b. On admet que si y est l'entier associé à une lettre codée, alors l'entier x correspondant à la lettre décodée vérifie $x \equiv 3y + 11[26]$.
   Décoder le mot ZA.

Annexe A6 — BTS SIO 2025 — Le reste par soustractions

Sont joints à ce PDF : le document complet (bts_sio_metropole_2025.pdf) et le corrigé officiel. Ouvrez-les depuis le trombone des pièces jointes de votre lecteur PDF.

Exercice 2 — 5 points

Un codage affine est une méthode de chiffrement utilisée en cryptographie, qui consiste à remplacer chaque lettre du message en clair par une autre lettre de l'alphabet en utilisant une fonction affine. On associe à chaque lettre de l'alphabet une valeur numérique.
Seules les lettres majuscules sont utilisées dans cet exercice.
Voici le tableau de correspondance des lettres à leur rang.

lettre	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
rang	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25

Ensuite, on définit la fonction affine de codage de la lettre de rang n.

Soit f la fonction définie pour tout entier n compris entre 0 et 25 par :

$$f(n) \equiv 7n+5 \ [26] \text{ avec } 0 \leqslant f(n) \leqslant 25.$$

Exemple : La lettre E a pour rang 4 et

$7 \times 4 + 5 = 33 = 1 \times 26 + 7$ donc $7 \equiv 7 \times 4 + 5 \ [26]$. Comme $0 \leqslant 7 \leqslant 25$,
on a $f(4) = 7$ qui est le rang de la lettre H, donc E sera codé par H.

1. Coder ainsi les lettres C et T en détaillant les étapes.
2. Pour calculer le reste de la division euclidienne d'un entier N par 26, il suffit de soustraire 26 à N autant de fois que possible, c'est-à-dire tant que la différence demeure supérieure ou égale à 26.

   Exemple :

   Le reste de la division euclidienne de 80 par 26 s'obtient ainsi :

   $80 - 26 = 54$ Comme $54 \geqslant 26$, on soustrait alors 26 à 54.
   $54 - 26 = 28$ Comme $28 \geqslant 26$, on soustrait alors 26 à 28.
   $28 - 26 = 2$ Comme $2 < 26$, on arrête.
   Le reste de la division euclidienne de 80 par 26 est 2.

   La fonction « `reste_division_par_26` » de paramètre `N`, ci-après, est écrite en langage naturel et emploie cette méthode pour renvoyer le reste de la division euclidienne de `N` par 26.

   Exemple : `reste_division_par_26(80)` renvoie la valeur entière 2.

   ```
   Fonction reste_division_par_26(N)
       Tant que N ... Faire
           N← ...
       Fin de Tant que
       Renvoyer (N)
   Fin de la fonction
   ```

   Recopier et compléter les deux lignes contenant des pointillés de cette fonction.
3. La fonction `indice (lettre, chaine)` ci-dessous, renvoie le plus petit indice du caractère « `lettre` »dans la chaîne de caractères « `chaine` », lorsqu'il est présent dans « `chaine` », et elle renvoie le nombre de caractères de « `chaine` » sinon.

Annexe A7 — BTS SIO 2025 — L'indice d'une lettre

Sont joints à ce PDF : le document complet (bts_sio_metropole_2025.pdf) et le corrigé officiel. Ouvrez-les depuis le trombone des pièces jointes de votre lecteur PDF.

Exercice 2 — 5 points

Un codage affine est une méthode de chiffrement utilisée en cryptographie, qui consiste à remplacer chaque lettre du message en clair par une autre lettre de l'alphabet en utilisant une fonction affine. On associe à chaque lettre de l'alphabet une valeur numérique.
Seules les lettres majuscules sont utilisées dans cet exercice.
Voici le tableau de correspondance des lettres à leur rang.

lettre	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
rang	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25

Ensuite, on définit la fonction affine de codage de la lettre de rang n.

Soit f la fonction définie pour tout entier n compris entre 0 et 25 par :

$$f(n) \equiv 7n + 5 \ [26] \text{ avec } 0 \leqslant f(n) \leqslant 25.$$

Exemple : La lettre E a pour rang 4 et

$7 \times 4 + 5 = 33 = 1 \times 26 + 7$ donc $7 \equiv 7 \times 4 + 5 \ [26]$. Comme $0 \leqslant 7 \leqslant 25$,
on a $f(4) = 7$ qui est le rang de la lettre H, donc E sera codé par H.

1. Coder ainsi les lettres C et T en détaillant les étapes.
2. Pour calculer le reste de la division euclidienne d'un entier N par 26, il suffit de soustraire 26 à N autant de fois que possible, c'est-à-dire tant que la différence demeure supérieure ou égale à 26.

   Exemple :

   Le reste de la division euclidienne de 80 par 26 s'obtient ainsi :

   $80 - 26 = 54$ Comme $54 \geqslant 26$, on soustrait alors 26 à 54.
   $54 - 26 = 28$ Comme $28 \geqslant 26$, on soustrait alors 26 à 28.
   $28 - 26 = 2$ Comme $2 < 26$, on arrête.
   Le reste de la division euclidienne de 80 par 26 est 2.

   La fonction « `reste_division_par_26` » de paramètre `N`, ci-après, est écrite en langage naturel et emploie cette méthode pour renvoyer le reste de la division euclidienne de `N` par 26.

   Exemple : `reste_division_par_26(80)` renvoie la valeur entière 2.

```
Fonction reste_division_par_26(N)
    Tant que N ...   Faire
        N← ...
    Fin de Tant que
    Renvoyer (N)
Fin de la fonction
```

   Recopier et compléter les deux lignes contenant des pointillés de cette fonction.
3. La fonction `indice (lettre, chaine)` ci-dessous, renvoie le plus petit indice du caractère « `lettre` »dans la chaîne de caractères « `chaine` », lorsqu'il est présent dans « `chaine` », et elle renvoie le nombre de caractères de « `chaine` » sinon.

La documentation de cette fonction précise que :

- `k` est un entier.
- `lettre` est une chaîne de caractères constituée d'un seul caractère.
- `chaine` est une chaîne de caractères.
- `longueur(chaine)` renvoie le nombre de caractères de `chaine`.

Exemple : `indice("S","BTSSIO")` renvoie la valeur 2, le plus petit indice de `"S"` dans `"BTSSIO"`.

```
Fonction indice(lettre, chaine)
    k ← 0
    Tant que ... longueur(chaine) et lettre ... Faire
        k ← k + 1
    Fin de Tant que
    Renvoyer (k)
Fin de Fonction
```

a. Quelle est la valeur renvoyée par `indice("A","BTSSIO")` ?

b. Recopier et compléter la ligne contenant les pointillés de la fonction `indice`.

Exercice 3 — 10 points

Partie A

Les étudiants d'une classe de BTS SIO doivent fabriquer avec une imprimante 3D trois pièces : un boîtier, son couvercle ainsi qu'un support mural.
Ces trois pièces seront respectivement désignées dans la suite du sujet par la pièce P_1, la pièce P_2 et la pièce P_3.
Le projet de conception et de fabrication des prototypes de ces trois pièces est détaillé dans le tableau ci-dessous :

Tâches	Description	Durées en min	Prédécesseurs
A	Création du fichier .stl de la pièce P_1.	120	Aucun
B	Création du fichier .stl de la pièce P_2.	75	Aucun
C	Création du fichier .stl de la pièce P_3.	150	Aucun
D	Paramétrage de l'impression des 3 pièces.	25	A, B, C
E	Impression des pièces P_1 et P_2.	128	D
F	Finition de la pièce P_1.	15	E
G	Finition de la pièce P_2.	5	E
H	Impression de la pièce P_3.	103	D
I	Finition de la pièce P_3.	15	H

1. a. Quels sont les successeurs de la tâche E ?

b. On admet que le graphe associé à ce projet peut être ordonnancé.
Déterminer le niveau de chaque tâche.

Annexe A8 — BTS SIO 2024 — Le bit de parité

Sont joints à ce PDF : le document complet (bts_sio_metropole_2024.pdf) et le corrigé officiel. Ouvrez-les depuis le trombone des pièces jointes de votre lecteur PDF.

Exercice 2 **5 points**

La norme de codage ASCII (American Standard Code for Information Interchange), défini aux États-Unis en 1963, associe aux caractères les plus utilisés dans les documents en langue anglaise un entier représentable en binaire sur 7 bits.

1. Combien de caractères peut-on ainsi encoder ?
2. Aux lettres majuscules A, B, …, Z sont associés les nombres de 65 à 90, et aux lettres minuscules a, b, …, z, ceux de 97 à 122.
   a. Donner l'écriture binaire associée à la lettre "d",
   b. Quel caractère correspond à l'écriture binaire 1101101 ?
3. Lors de la transmission des données, pour éviter les erreurs, une méthode consiste à ajouter pour chaque caractère, un bit de parité à la fin du codage en binaire.
   Pour cela, on compte le nombre de 1 apparaissant dans le codage sur 7 bits d'un caractère :
   — si le nombre obtenu est pair, on rajoute 0 à la fin du codage ;
   — sinon on rajoute 1.
   Chaque caractère est alors codé par un groupe de 8 bits appelé octet.
   Par exemple :
   — la lettre "A" est codée par 1000001 en binaire. Ce codage contient un nombre pair de 1, donc on ajoute 0 à la fin, et la lettre "A" sera codée sur 8 bits par 10000010 ;
   — la lettre "C" est codée par 1000011 en binaire. Ce codage contient un nombre impair de 1, donc on ajoute 1 à 'a fin, et la lettre "C" sera codée sur 8 bits par 10000111.
   On considère l'algorithme ci-dessous écrit en langage naturel où ajoute_bit_parite est la fonction prenant en paramètre une chaine de caractères code de longueur 7 représentant un codage binaire et qui renvoie le codage obtenu en lui ajoutant le bit de parité à la fin.
   Ainsi ajoute_bit_parite("11011 00") renvoie "11011000" et ajoute_bit_parite("1100001") renvoie "11000011".

```
Fonction ajoute_bit_parite(code)
  compt← 0
  Pour i allant de 0 à 6 Faire
      Si code[i] est égal à "1"
      Alors

      Fin de Si
  Fin de Pour
  Si le reste de la division de …par 2 est 0
  Alors
      code ← code +"0"
  Sinon
      ……
  Fin de Si
  Renvoyer code
```

Recopier et compléter cet algorithme pour qu'il renvoie le code binaire sous forme de chaine de caractères, complété par le bit de parité.

Exercice 3 **10 points**

Partie A

La directrice d'une entreprise doit recruter une personne pour son équipe. Ce poste requiert des compétences professionnelles et humaines.
L'évaluation du candidat attribue une note sur 10 points à ses compétences professionnelles et une note sur 10 points à ses compétences humaines. Le total des points du candidat forme ainsi une note sur 20 appelée note finale.
Pour qu'un candidat soit sélectionné, il faut qu'au moins un des critères suivants soit respecté.

— le candidat a obtenu une note finale supérieure ou égale à 12 et il a eu au moins 5 points aux compétences humaines ;
— le candidat a obtenu une note finale inférieure strictement à 12 et il a obtenu 10 points aux compétences professionnelles ;
— le candidat n'a pas obtenu au moins 5 points aux compétences humaines et il a obtenu 10 points aux compétences professionnelles.

On définit les trois variables booléennes a, b, c de la façon suivante :

- a lorsque le candidat a obtenu une note finale supérieure ou égale à 12 $\overline{a}$ sinon ;
- b lorsque le candidat a obtenu au moins 5 points aux compétences humaines, $\overline{b}$ sinon :
- c lorsque le candidat a obtenu 10 points aux compétences professionnelles, $\overline{c}$ sinon.

1. On considère la proposition « Tous les candidats ont obtenu une note finale supérieure ou égale à 12 ».
Donner la négation de cette proposition.

2. On note E l'expression booléenne correspondant aux critères de sélection d'un candidat

a. Exprimer E en fonction des variables booléennes a, b, c.

b. Représenter l'expression E dans un tableau de Karnaugh.
En déduire une expression simplifiée de E sous la forme d'une somme de deux termes

c. Traduire la forme simplifiée de E à l'aide d'une phrase.

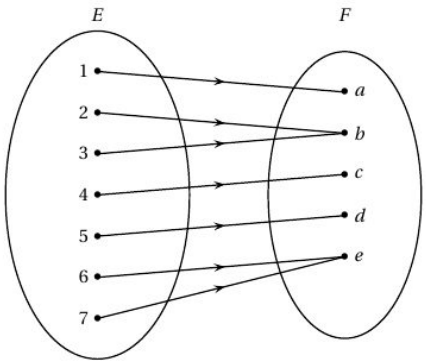
d. Un candidat a obtenu au moins 5 points aux compétences humaines, et il n'a pas obtenu 10 points aux compétences professionnelles.
Sa candidature peut-elle être retenue? Justifier.

Partie B

Pour équiper ses bureaux, l'entreprise a besoin de tables, d'armoires et de chaises.
Ayant demandé un devis à trois fournisseurs notés A, B, C, l'entreprise a obtenu les renseignements suivants, où les prix sont en euros :

Annexe A9 — BTS SIO 2023 — Le degré sortant d'un sommet

Sont joints à ce PDF : le document complet (bts_sio_metropole_2023.pdf) et le corrigé officiel. Ouvrez-les depuis le trombone des pièces jointes de votre lecteur PDF.



L'application ainsi définie est

A : injective et non surjective	B surjective et non injective	C : bijective

Exercice 2 **5 points**

Partie A

On considère le graphe orienté G comportant 3 sommets notés A, B et C dont la matrice d'adjacence est P, où $P = \begin{pmatrix} 1 & 0 & 1 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{pmatrix}$.

1. Dessiner une représentation du graphe G.
2. **a.** Calculer la matrice P^2.
   b. Combien de chemins de longueur 2 ont pour origine B ?
3. Déterminer la matrice d'adjacence $\hat{P}$ et le graphe de la fermeture transitive de G

Partie B

Dans un graphe orienté, on définit :

— le **degré entrant** d'un sommet comme étant le nombre d'arcs menant à ce sommet.
— le **degré sortant** d'un sommet comme étant le nombre d'arcs issus de ce sommet.

1. **a.** Calculer le degré entrant du sommet C du graphe G défini dans la partie A.
   b. Calculer le degré sortant du sommet C du graphe G défini dans la partie A.
2. On étudie dans cette question les graphes orientés à trois sommets numérotés de 1 à 3.
   On considère l'algorithme ci-dessous écrit en langage naturel où Degré_sortant désigne une fonction de paramètres M et s, M étant une matrice à 3 lignes et 3 colonnes et s un entier compris entre 1 et 3.
   Le coefficient de la matrice M situé ligne i colonne j est noté m_{ij}.

```
Fonction Degré_sortant (M, s)
    deg ← 0
    Pour j allant de 1 à 3 Faire
        Si m_sj ……Faire
        ……
        Fin de Si
    Fin de Pour
Retourner deg
```

Compléter cet algorithme pour que la fonction renvoie le degré sortant du sommet numéroté s dans un graphe dont la matrice d'adjacence est M.

Exercice 3 **10 points**

Une entreprise décide de mettre en place une authentification à plusieurs étapes permettant à ses employés d'accéder aux services en ligne qu'elle propose.

Partie A

La première authentification consiste à utiliser un mot de passe.
À la première connexion, l'utilisateur doit créer un mot de passe de 8 à 16 caractères. Ces caractères peuvent être des lettres majuscules de l'alphabet ou des chiffres ou des caractères spéciaux (?,&, etc.).
Pour être valide, un mot de passe doit remplir au moins l'une des trois conditions suivantes :

— il contient au moins trois chiffres et au moins deux caractères spéciaux ;

— il contient moins de trois chiffres, au moins deux caractères spéciaux et au moins dix lettres ;

— il contient moins de deux caractères spéciaux et au moins dix lettres.

1. Les mots de passe suivants sont-ils valides? Justifier.

ABCDABCD?# STU27ABCABCDE&

On définit les variables booléennes a, b et c de la manière suivante :

- a lorsque le mot de passe contient au moins trois chiffres, $\overline{a}$ sinon ;
- b lorsque le mot de passe contient au moins deux caractères spéciaux, $\overline{b}$ sinon ;
- c lorsque le mot de passe contient au moins dix lettres, $\overline{c}$ sinon.

2. **a.** On appelle E l'expression booléenne qui traduit la validité d'un mot de passe.
Traduire chacune des conditions de validité d'un mot de passe à l'aide des variables a, b et c, puis en déduire une expression de E.

b. Représenter E dans un tableau de Karnaugh, puis en déduire une expression simplifiée de E sous la forme d'une somme de deux termes.

c. Traduire par une phrase l'expression simplifiée de E.

3. Déterminer l'expression booléenne $\overline{E}$ négation de E.

Annexe A10 — BTS SIO 2022 — Les diviseurs et les nombres premiers

Sont joints à ce PDF : le document complet (bts_sio_metropole_2022.pdf) et le corrigé officiel. Ouvrez-les depuis le trombone des pièces jointes de votre lecteur PDF.

Question 4

Soit n un entier relatif.

On considère l'égalité matricielle : $\begin{pmatrix} 3 & 4 & 5 \\ -2 & 1 & 2 \end{pmatrix} \times \begin{pmatrix} 1 & 4 \\ -1 & n \\ 3 & -2 \end{pmatrix} = \begin{pmatrix} 14 & -22 \\ 3 & -18 \end{pmatrix}$.

Elle est vérifiée pour :

A : $n = -3$ **B :** $n = 4$ **C :** $n = -6$ **D :** $n = 5$

Question 5

Soit P la proposition : « *Si la télévision est allumée alors quelqu'un la regarde.* »
Parmi les expressions suivantes laquelle est équivalente à P ?

A : Si la télévision n'est pas allumée alors personne ne la regarde.

B : Si la télévision est allumée alors personne ne la regarde

C : Si personne ne regarde la télévision alors la télévision n'est pas allumée.

D : Si personne ne regarde la télévision alors la télévision est allumée.

Exercice 2 **5 points**

1. Donner tous les nombres premiers inférieurs à 25.
2. Le nombre 623 est-il un nombre premier? Justifier.
3. Donner tous les diviseurs de 105.
4. On considère l'algorithme ci-dessous écrit en langage naturel où Div désigne une fonction de paramètre Nbre, Nbre étant un nombre entier supérieur ou égal à 2.

   Algorithme en langage naturel :

```
Fonction Div(Nbre)
    Test ← 0
    Pour i allant de 1 à Nbre Faire
      Si le reste de la division de Nbre par i est égal à 0 Faire
        Test ← Test + 1
      Fin de Si
    Fin de Pour
    Retourner Test
```

   a. Que renvoie Div(6) ? Justifier en expliquant le rôle de cette fonction.

   b. Écrire une fonction Prem de paramètre Nbre, où Nbre est un nombre entier supérieur ou égal à 2, qui renvoie Vrai (ou True) si Nbre est premier, Faux (ou False) si Nbre n'est pas premier.

   On pourra utiliser la fonction Div.