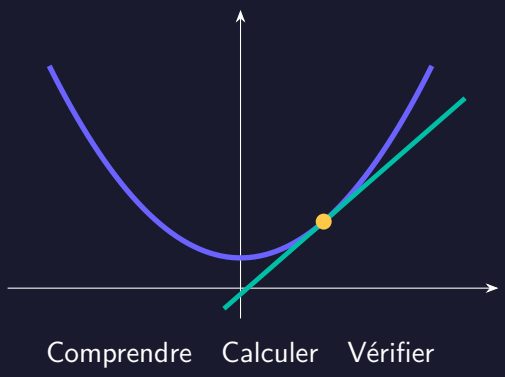
FICHE 08

Algorithmique, Python et tableur

Algorithmique ▪ Comprendre et s'entraîner



Première STMG et STI2D — commun

Année scolaire 2026–2027

Cours · 18 exercices · Problème guidé · Corrigés

Table des matières

1 Pourquoi ce chapitre ?

Un programme répète une démarche que tu sais expliquer. Les variables et les boucles deviennent utiles pour suivre un stock et trouver un seuil. **Ton parcours.** Enseignement commun de mathématiques pour les séries STMG et STI2D. Les situations de gestion et les situations techniques mobilisent ici les mêmes notions du programme.

- Variables, affectations et fonctions.
- Choisir avec une condition.
- Répéter un nombre connu de fois.
- Répéter jusqu'à un objectif.
- Listes, données et contrôles numériques.
- Construire et recopier une formule de tableur.

2 L'idée avant la formule

Joue d'abord le rôle de l'ordinateur sur trois tours. Une affectation remplace une valeur ; la trace rend les mises à jour visibles.

3 Le cours formel

3.1 Variables, affectations et fonctions

Intuition | Construire la notion pas à pas

Partir d'une tâche que l'on sait faire à la main. Un atelier facture 15 euros de préparation et 3 euros par pièce. Pour quatre pièces, tu calcules d'abord le coût des pièces, puis tu ajoutes la préparation. Programmer consiste à décrire exactement ces opérations pour qu'elles puissent être refaites avec une autre quantité.
Donner un nom à une donnée. Écrire `n = 4` range le nombre 4 sous le nom `n`. La lettre n'est pas une inconnue à résoudre : elle possède déjà une valeur. Dans `cout = 15 + 3*n`, Python lit cette valeur, calcule $15 + 3 \times 4$, puis mémorise 27 sous le nom `cout`. L'ordre est donc : lire, calculer, ranger.
Comprendre ce qui reste en mémoire. Une variable conserve un résultat, pas une relation vivante. Si l'on écrit ensuite `n = 10`, le coût déjà mémorisé reste 27. Il faut exécuter à nouveau la formule pour obtenir le coût correspondant à dix pièces.
Rendre le calcul réutilisable. Une fonction reçoit une donnée d'entrée, effectue les instructions de son bloc et renvoie une sortie. Le paramètre de la fonction est un nom local : il reçoit la valeur fournie lors de l'appel. Les espaces au début des lignes indiquent quelles instructions appartiennent à cette fonction. Les deux-points annoncent le bloc.

Définition | Comprendre la notion

Un programme décompose un calcul en instructions. En Python, `x = 5` affecte la valeur 5 à la variable `x` ; ce n'est pas une équation. Une nouvelle affectation remplace l'ancienne valeur. Les opérations utilisent `+`, `-`, `*`, `/` et `**` pour une puissance. Les décimales s'écrivent avec un point. `==` teste une égalité, alors que `=` affecte. Une fonction définie avec `def` reçoit des paramètres et renvoie une valeur avec `return`. `print` affiche une information mais ne remplace pas le retour d'une fonction. Les lignes indentées appartiennent au même bloc.

Exemple | Exemple commenté

Objectif : calculer le coût de n pièces sans recopier la formule à chaque fois.

```
def cout(n):
    return 15 + 3*n

prix = cout(4)
print(prix)
```

1. `def cout(n):` définit une recette nommée `cout`. Cette ligne ne commande pas encore une fabrication et ne calcule pas le coût de quatre pièces.
2. L'appel `cout(4)` donne la valeur 4 au paramètre `n`. La multiplication est prioritaire : $3 \times 4 = 12$.
3. L'addition donne $15 + 12 = 27$. `return` transmet ce nombre à l'endroit où la fonction a été appelée.
4. L'affectation `prix = cout(4)` range donc 27 dans `prix`. Enfin, `print(prix)` affiche 27.

Contrôle indépendant. Quatre pièces à 3 euros coûtent 12 euros ; avec les 15 euros fixes, le total est bien 27 euros. Pour zéro pièce, la formule renvoie 15 : cela correspond aux frais fixes du modèle. Si aucun frais n'est réellement facturé en l'absence de commande, il faut modifier le modèle, pas accuser Python.
Pourquoi distinguer afficher et renvoyer ? On peut écrire `cout(4) + 5` pour ajouter des frais, car `cout(4)` fournit un nombre. Une fonction qui se contente de l'afficher ne fournit pas automatiquement ce nombre au calcul suivant.

Attention | Comprendre une erreur fréquente

En Python, `x = x + 1` n'est pas une égalité à résoudre. Avec x égal à 4, le membre droit vaut 5 ; on remplace alors 4 par 5. Confondre cette affectation avec $x = x + 1$ en mathématiques conduit à déclarer impossible une instruction parfaitement valide. Pour comparer deux valeurs, Python utilise `==`.

À toi d'essayer : Pause 1 — Mets la notion à l'épreuve

En Python, le carré de x s'écrit...

1. `x**2`
2. `x^2`
3. `2x`

Tu peux chercher, prendre un indice ou lire l'explication, à ton rythme.

Indices, p. 31 Explication, p. 33

3.2 Choisir avec une condition

Intuition | Construire la notion pas à pas

Du règlement au choix. Une livraison est gratuite à partir de 50 euros, sinon elle coûte 5 euros. Avant d'écrire un programme, formule une question qui possède deux réponses : « le total est-il supérieur ou égal à 50 ? ». Si oui, les frais valent zéro ; si non, ils valent cinq.
Un test produit une valeur logique. Pour un total de 42, `total >= 50` est faux. Pour 50 ou 70, il est vrai. Le signe égal dans « supérieur ou égal » décide précisément de ce qui arrive à la frontière : il ne s'agit pas d'un détail d'écriture.
Des branches, pas plusieurs calculs obligatoires. Dans une structure `if/elif/else`, Python évalue les conditions dans l'ordre et exécute la première branche dont la condition est vraie. Il ignore les branches suivantes. `else` couvre tous les cas qui n'ont pas été retenus ; cette dernière branche n'a donc pas besoin d'un nouveau test.
Combiner des exigences. « Avoir au moins 16 ans et présenter une autorisation » exige les deux informations. « Payer en espèces ou par carte » accepte au moins l'un des moyens. En logique, ce « ou » n'exclut pas que les deux conditions soient vraies. Cette lecture précise permet de choisir entre `and` et `or`.

Définition | Comprendre la notion

Une condition est évaluée comme vraie ou fausse. `if`, `elif` et `else` choisissent un bloc à exécuter. Les tests utilisent `<`, `<=`, `>`, `>=`, `==`, `!=`. `and` demande deux conditions vraies ; `or` en demande au moins une ; `not` inverse une condition. L'ordre des branches est important : dans une chaîne `if/elif/else`, seule la première condition vraie est retenue. Teste toujours les valeurs aux frontières et juste de chaque côté. Une comparaison approchée sur un nombre décimal calculé peut nécessiter une tolérance.

Exemple | Exemple commenté

Traduire la règle de livraison.

```
def livraison(total):
    if total >= 50:
        return 0
    else:
        return 5
```

Pour `livraison(49)`, le test $49 \geq 50$ est faux : Python va dans `else`, puis renvoie 5. Pour `livraison(50)`, $50 \geq 50$ est vrai : il renvoie 0. Pour 51, la même première branche fournit 0. Les trois essais contrôlent les deux côtés et la frontière.

Pourquoi l'indentation compte. Les deux instructions `return` appartiennent à des branches différentes. Leur décalage indique leur rôle. Aligner `return 5` à l'intérieur de la première branche rendrait le programme différent ; copier un code sans ses espaces peut donc modifier ou empêcher son exécution.

Ajouter une troisième tranche. Supposons 8 euros de frais sous 20 euros, 5 euros entre 20 euros inclus et 50 euros exclus, puis zéro. On teste d'abord `total < 20`, ensuite `total < 50`, enfin `else`. Dans le deuxième test, on sait déjà que le total n'est pas inférieur à 20 : l'échec du premier test apporte cette information. Pour vérifier cette variante, essaie 19, 20, 49 et 50.

Attention | Comprendre une erreur fréquente

Deux instructions `if` séparées ne sont pas une chaîne `if/elif`. Avec deux `if`, les deux blocs peuvent s'exécuter. Pour x égal à 12, « x positif » et « x supérieur à 10 » sont tous deux vrais. Choisis une chaîne lorsque les catégories doivent être exclusives.

À toi d'essayer : Pause 2 — Mets la notion à l'épreuve

Pour tester une égalité en Python, on utilise...

1. `=`
2. `:=`
3. `==`

Tu peux chercher, prendre un indice ou lire l'explication, à ton rythme.

Indices, p. 31 Explication, p. 33

3.3 Répéter un nombre connu de fois

Intuition | Construire la notion pas à pas

Repérer ce qui se répète. Pour additionner les cinq premiers entiers positifs, tu peux écrire $1+2+3+4+5$. Pour cinq mille entiers, cette écriture devient peu pratique. L'opération répétée est « ajouter le prochain entier à la somme déjà obtenue ».

Séparer deux rôles. La variable k indique quel entier on est en train d'ajouter. La variable s conserve la somme des entiers déjà parcourus. Avant tout ajout, cette somme vaut zéro : c'est l'initialisation. Après chaque passage, s doit conserver son nouveau résultat pour le passage suivant.

Comprendre une borne exclue. Dans `range(1, 6)`, le premier entier est 1 et l'on s'arrête avant 6. Les valeurs sont donc 1, 2, 3, 4, 5. Il y a cinq passages. Dans `range(5)`, le début implicite est zéro : on obtient 0, 1, 2, 3, 4, toujours cinq passages. Le nombre de passages et la dernière valeur parcourue sont deux informations différentes.

La boucle exprime une répétition précise. Les instructions indentées sont exécutées pour chaque valeur. L'initialisation se place avant la boucle, car on ne veut pas effacer le travail déjà fait à chaque passage. Une fois la boucle terminée, la variable conserve le dernier résultat.

Définition | Comprendre la notion

Une boucle `for` parcourt une suite de valeurs. `range(n)` produit les entiers de 0 à $n-1$; `range(a,b)` s'arrête avant b. Pour une somme, initialise un accumulateur à 0 ; pour un produit, à 1. La mise à jour doit rester dans la boucle. Un compteur mesure un nombre d'actions, alors qu'un accumulateur additionne leurs valeurs. Une trace sur un petit exemple permet de vérifier les bornes et l'indentation. Dans un tableur, la recopie de formules effectue aussi des calculs répétés, mais les références relatives et absolues doivent être contrôlées.

Exemple | Exemple commenté

Additionner les entiers de 1 à 5.

```
s = 0
for k in range(1, 6):
    s = s + k
print(s)
```

Avant le premier passage, s vaut 0. Pour k égal à 1, le membre droit vaut $0+1=1$, donc s devient 1. Pour k égal à 2, on utilise cette nouvelle valeur : $1+2=3$. Pour k égal à 3, on calcule $3+3=6$. Les deux derniers passages donnent $6+4=10$, puis $10+5=15$.

k	1	2	3	4	5
s après l'ajout	1	3	6	10	15

Le `print` n'est pas indenté dans la boucle : il affiche seulement le résultat final 15. S'il était dans le bloc, les cinq sommes intermédiaires seraient affichées. Ce changement ne modifierait pas les calculs de s, seulement les affichages.

Contrôle. Le calcul direct $1+2+3+4+5=15$ retrouve le résultat. Pour sommer de 1 à n, la borne finale devient `n+1`. Le cas n égal à zéro donne une suite vide de valeurs et conserve la somme initiale zéro.

Attention | Comprendre une erreur fréquente

Placer `s = 0` dans le bloc de la boucle remet la somme à zéro à chaque tour. Pour les termes 1, 2 et 3, on obtiendrait successivement 1, puis 2, puis 3, au lieu de 1, 3, 6. L'initialisation est exécutée une seule fois avant de commencer à accumuler.

À toi d'essayer : Pause 3 — Mets la notion à l'épreuve

`range(5)` parcourt...

1. 0, 1, 2, 3, 4, 5
2. 1, 2, 3, 4, 5
3. 0, 1, 2, 3, 4

Tu peux chercher, prendre un indice ou lire l'explication, à ton rythme.

Indices, p. 31 Explication, p. 33

3.4 Répéter jusqu'à un objectif

Intuition | Construire la notion pas à pas

Quand le nombre de passages est inconnu. Un stock reçoit chaque jour une livraison jusqu'à atteindre une quantité souhaitée. Tu connais la règle d'évolution, mais le nombre de jours n'est pas toujours donné. Il faut répéter tant que l'objectif reste non atteint.
Séparer l'objectif et la condition de continuation. Pour atteindre au moins 100 unités, on s'arrête quand la réserve est supérieure ou égale à 100. On continue donc tant qu'elle est strictement inférieure à 100. Écrire la condition d'arrêt à la place de celle de continuation empêcherait souvent la boucle de commencer.
La chronologie d'un tour. On initialise les variables. On teste la condition. Si elle est vraie, on exécute le bloc, puis on revient tester avec les nouvelles valeurs. Si elle est fausse, on passe à l'instruction qui suit la boucle. Le test se fait avant le premier passage : zéro passage est une possibilité normale.
Expliquer pourquoi la boucle finit. Une instruction correcte doit faire progresser vers un état où le test devient faux. Ajouter une quantité positive à un stock initialement sous un seuil fini y conduit. Retirer une quantité positive peut au contraire éloigner indéfiniment du seuil. Une limite de sécurité protège un calcul expérimental, mais ne dispense pas de comprendre la progression.

Définition | Comprendre la notion

Une boucle `while` teste une condition avant chaque passage. Si elle est fausse dès le départ, le bloc ne s'exécute pas. À l'intérieur, une mise à jour doit faire évoluer la situation vers la fin recherchée. Pour un seuil, le test exprime « l'objectif n'est pas encore atteint ». Justifie la terminaison à partir du modèle ; ajoute une limite d'itérations si l'atteinte du seuil n'est pas garantie. Un balayage numérique peut aussi utiliser cette boucle. L'accumulation d'erreurs décimales invite à compter avec des rangs entiers et calculer une abscisse à partir du rang plutôt qu'à additionner indéfiniment un pas.

Exemple | Exemple commenté

Trouver le premier entier naturel dont le carré dépasse 50.

```
n = 0
while n*n <= 50:
    n = n + 1
print(n)
```

Au départ, $0^2 \leq 50$ est vrai : on remplace n par 1. Le nouveau test porte sur 1^2, pas sur l'ancien zéro. Le processus continue ainsi jusqu'à n égal à 7. À cet instant $7^2 = 49 \leq 50$ est encore vrai ; un passage supplémentaire donne n égal à 8. Le test $8^2 = 64 \leq 50$ est faux et la boucle s'arrête.
Justifier le mot « premier ». Le dernier entier refusé comme réponse est 7 : son carré ne dépasse pas 50. L'entier 8 convient. Comme les carrés des entiers naturels augmentent, aucun entier naturel plus petit que 8 ne convient. Vérifier seulement $8^2 > 50$ prouverait que 8 convient, sans prouver à lui seul qu'il est le premier.
Changer l'objectif. Pour rechercher un carré supérieur ou égal à 49, la continuation serait `n*n < 49`. Le résultat serait 7. Pour rechercher un carré strictement supérieur à 49, on conserverait `<= 49` et le résultat serait 8. L'égalité au seuil doit donc être traitée avant d'écrire le code.

Attention | Comprendre une erreur fréquente

Une boucle peut s'arrêter sans donner la bonne réponse si le test ne correspond pas à l'énoncé. Pour atteindre *au moins* 100, continuer avec `u <= 100` impose un tour supplémentaire lorsque u vaut exactement 100. Tester la frontière est aussi important que vérifier l'absence de boucle infinie.

À toi d'essayer : Pause 4 — Mets la notion à l'épreuve

La condition d'une boucle while est testée...

1. Avant chaque passage
2. Seulement après le premier passage
3. Une seule fois au début

Tu peux chercher, prendre un indice ou lire l'explication, à ton rythme.

Indices, p. 31 Explication, p. 33

3.5 Listes, données et contrôles numériques

Intuition | Construire la notion pas à pas

Conserver plusieurs résultats. Une seule variable peut contenir la température actuelle, mais elle ne garde pas automatiquement les mesures précédentes. Une liste permet de conserver une suite ordonnée de données : `temperatures = [12, 15, 14]` contient trois nombres distinctement accessibles.
Distinguer numéro d'ordre et indice. Le premier élément possède l'indice 0, le deuxième l'indice 1, et ainsi de suite. Pour une liste de trois éléments, le dernier indice est donc 2. `len` donne le nombre d'éléments, pas le dernier indice. Cette convention doit être prise en compte quand on parcourt la liste.
Parcourir les valeurs ou les positions. `for x in L` donne successivement les valeurs contenues dans L. Si l'on a besoin de leur position, on peut parcourir les indices. Ne remplace pas un indice par une donnée : dans une liste de températures, 15 est une mesure, pas forcément une position disponible.
Préserver le sens des données. Lorsque deux listes contiennent des instants et des mesures, les éléments de même indice forment une observation. Changer l'ordre d'une seule liste modifie ces observations. Les opérations informatiques doivent respecter l'organisation des données, leur unité et les informations éventuellement manquantes.

Définition | Comprendre la notion

Une liste conserve plusieurs valeurs ordonnées. Pour `L = [4, 7, 9]`, les indices sont 0, 1, 2 ; `L[0]` vaut 4 et `len(L)` vaut 3. On parcourt les valeurs avec `for x in L`. `append` ajoute un élément en fin de liste. Une moyenne s'obtient par `sum(L)/len(L)` pour une liste non vide. Des listes peuvent stocker une table de fonction, les termes d'une suite ou des fréquences simulées. Garde les couples de données associés. Vérifie les entrées vides, les unités et les cas simples dont le résultat est connu avant d'interpréter une sortie.

Exemple | Exemple commenté

Construire la liste des carrés de 0 à 4.

```
carres = []
for k in range(5):
    carres.append(k*k)
```

`[]` crée une liste vide. `append` ajoute une nouvelle valeur à la fin sans effacer les précédentes. Les valeurs de k sont 0, 1, 2, 3 et 4.

1. Pour k égal à 0, le carré vaut 0. La liste devient `[0]`.
2. Pour k égal à 1, le carré vaut 1. Elle devient `[0, 1]`.
3. Pour k égal à 2, on ajoute 4 : `[0, 1, 4]`.
4. Les passages suivants ajoutent 9 et 16 : la liste finale est `[0, 1, 4, 9, 16]`.

On contrôle la longueur : cinq valeurs de k ont été parcourues et un élément a été ajouté à chaque tour, donc `len(carres)` vaut 5. L'élément d'indice 2 vaut 4 ; celui d'indice 4 vaut 16.
Calculer une moyenne. La somme de cette liste vaut $0+1+4+9+16=30$; son effectif est 5. Sa moyenne est donc $30/5=6$. En Python, `sum(carres)/len(carres)` traduit exactement cette formule. La liste doit être non vide : sinon son effectif est nul et le quotient n'est pas défini.

Attention | Comprendre une erreur fréquente

Avec trois données, la longueur vaut 3 mais les indices valides sont 0, 1 et 2. Demander `L[3]` dépasse donc la liste. Avant de lire une valeur, vérifie que l'indice demandé est inférieur à la longueur et qu'il correspond à la donnée recherchée.

À toi d'essayer : Pause 5 — Mets la notion à l'épreuve

Dans `[3, 8, 11]`, l'élément d'indice 1 vaut...

1. 3
2. 11
3. 8

Tu peux chercher, prendre un indice ou lire l'explication, à ton rythme.

Indices, p. 32 Explication, p. 33

3.6 Construire et recopier une formule de tableur

Intuition | Construire la notion pas à pas

Organiser les données avant les formules. Pour une facture, prépare une colonne de quantités, une de prix unitaires et une de montants. Mets les titres et les unités sur la première ligne : une valeur 12 seule ne dit pas si elle signifie douze pièces ou douze euros. Chaque ligne suivante doit correspondre à un même article.
Passer d'un calcul à une référence. Le montant de quatre pièces à 12 euros vaut $4 \times 12 = 48$. Si 4 se trouve en A2 et 12 en B2, `=A2*B2` indique au tableur où lire les données. Changer A2 modifie automatiquement le résultat. Écrire simplement 48 perdrait cette relation ; écrire `=4*12` garderait le calcul mais ne suivrait pas les cellules.
Comprendre une recopie. Une formule placée en C2 qui lit A2 et B2 utilise les deux cellules de sa ligne. Recopiée en C3, elle devient `=A3*B3`. Ce déplacement est utile pour traiter l'article suivant. En revanche, un taux commun placé en D2 ne doit pas devenir D3 : il faut bloquer sa référence.
Lire un pourcentage enregistré. Une cellule affichant 20 % contient le nombre $0{,}20$. Ajouter une TVA de 20 % revient à multiplier le prix HT par $1 + 0{,}20 = 1{,}20$. Une cellule contenant simplement 20 n'est donc pas interchangeable avec une cellule contenant 20 %.

Définition | Comprendre la notion

Une cellule se repère par une lettre de colonne et un numéro de ligne : B2 est à l'intersection de la colonne B et de la ligne 2. Une formule commence par le signe égal et utilise les adresses des cellules contenant les données.
Dans une recopie, une référence relative comme B2 se déplace avec la formule. Le signe dollar bloque une coordonnée : `$D$2` bloque la colonne D et la ligne 2. La formule calcule automatiquement avec les nouvelles données des cellules référencées.
Une plage comme `C2:C5` comprend toutes les cellules de C2 à C5. Dans un tableur configuré en français, `SOMME` additionne une plage, `MOYENNE` calcule sa moyenne et `SI(test;valeur_si_vrai;valeur_si_faux)` exprime un choix. Les noms et séparateurs peuvent varier selon la langue du logiciel.

Exemple | Exemple commenté

Préparer une petite facture avec un taux commun. A2 contient 4, B2 contient 12 et D2 contient 20 %. En C2, on calcule le montant HT par `=A2*B2`, soit $4 \times 12 = 48$ euros. En E2, on écrit `=C2*(1+$D$2)`, soit $48 \times 1{,}20 = 57{,}60$ euros TTC.

Pour un second article, A3 contient 3 et B3 contient 10. La recopie de C2 en C3 donne `=A3*B3`, soit 30 euros HT. La recopie de E2 en E3 donne `=C3*(1+$D$2)`, soit 36 euros TTC. Le prix change de ligne, le taux reste en D2.
Contrôler le total de deux façons. `=SOMME(E2:E3)` donne $57{,}60 + 36 = 93{,}60$ euros. Comme les deux lignes utilisent le même taux, on peut aussi calculer le total HT $48 + 30 = 78$ euros, puis $78 \times 1{,}20 = 93{,}60$. L'accord valide ici la construction des références.
Une formule de décision. Pour afficher « à contrôler » lorsque le montant d'une ligne dépasse 50 euros, on peut écrire `=SI(E2>50;"à contrôler";"valide")`. Avec 57,60, le test est vrai ; avec 36, il est faux. Pour 50 exactement, il serait faux car le signe est strict.

Attention | Comprendre une erreur fréquente

Une formule correcte sur la première ligne peut devenir fausse après recopie. Si un taux commun se trouve en D2, la référence relative `D2` devient `D3` en descendant d'une ligne. Contrôle toujours la deuxième formule. Le signe dollar bloque une adresse ; il n'indique pas une monnaie.

À toi d'essayer : Pause 6 — Mets la notion à l'épreuve

Une formule de tableur commence par…

1. Le mot calcul
2. Le signe égal
3. Un point

Tu peux chercher, prendre un indice ou lire l'explication, à ton rythme.

Indices, p. 32 Explication, p. 34

4 Boîte à outils

Méthode | Variables, affectations et fonctions

1. Définis les entrées, leur unité et la sortie attendue. Pour une formule de coût, indique si les frais fixes sont toujours facturés.
2. Écris les opérations avec les symboles de Python : `*` pour multiplier, `**` pour une puissance, un point pour un nombre décimal.
3. Exécute les instructions dans l'ordre. Après chaque affectation, recopie les valeurs inchangées dans une table de trace.
4. Pour une fonction, distingue sa définition et son appel. Vérifie que `return` fournit bien la valeur dont le reste du programme a besoin.
5. Teste une entrée simple à la main et une valeur frontière, comme zéro si cette valeur appartient au domaine.

Méthode | Choisir avec une condition

1. Traduis la règle en français et repère les mots « au moins », « strictement », « entre », « et » et « ou ».

2. Écris une condition vraie exactement pour les cas souhaités ; vérifie chaque borne séparément.
3. Classe les branches de manière à ce que les premiers tests ne capturent pas à tort les cas des branches suivantes.
4. Teste une valeur dans chaque branche et chaque valeur frontière. Explique le chemin suivi avant d'observer la sortie.

Méthode | Répéter un nombre connu de fois

1. Écris la liste des valeurs produites par `range` sur un petit cas. Vérifie le début et la borne exclue.
2. Définis le rôle de chaque variable : indice, compteur ou résultat accumulé.
3. Initialise avant la boucle : zéro pour une somme, un pour un produit. Justifie ce choix par l'opération répétée.
4. À chaque tour, calcule le membre droit avec les valeurs actuelles puis range le nouveau résultat.
5. Compare la trace à un calcul manuel et contrôle ce qui est placé après la boucle.

Méthode | Répéter jusqu'à un objectif

1. Écris la condition d'arrêt en français, puis sa négation pour obtenir la condition du `while`.
2. Fixe l'état initial et vérifie si le test est déjà faux. Si oui, explique pourquoi zéro passage est correct.
3. Identifie la mise à jour de toutes les variables nécessaires, y compris le compteur de passages.
4. Fais une trace contenant l'état initial, chaque modification et le premier test faux.
5. Pour un premier rang, contrôle le rang retenu et son prédécesseur. Explique la terminaison.

Méthode | Listes, données et contrôles numériques

1. Nomme les données et précise leur ordre. Si deux colonnes vont ensemble, conserve leur association.
2. Compte les éléments pour connaître la longueur, puis numérote leurs indices à partir de zéro.
3. Pour construire une liste, initialise-la avant la boucle et ajoute les résultats au fur et à mesure.
4. Pour une moyenne, calcule séparément somme et effectif et vérifie que l'effectif est strictement positif.
5. Contrôle une première valeur, une dernière valeur et la longueur totale avant d'interpréter le résultat.

Méthode | Construire et recopier une formule de tableur

1. Donne une signification et une unité à chaque colonne. Vérifie que les données d'une ligne concernent le même article ou la même observation.
2. Écris le calcul en mots, puis remplace les nombres par leurs adresses de cellules. Commence la formule par =.

3. Pour chaque référence, décide si elle doit se déplacer à la recopie. Bloque un paramètre fixe avec les signes dollar nécessaires.
4. Contrôle la formule de la première ligne, puis celle de la deuxième après recopie. Ne te contente pas d'un résultat numérique plausible.
5. Pour une fonction SI, teste un cas vrai, un cas faux et l'égalité à la frontière. Pour une somme, vérifie que la plage comprend toutes les lignes utiles.

5 Exercices progressifs

Les exercices 1 à 6 consolident les bases ; 7 à 12 demandent plusieurs étapes ; 13 à 18 t'invitent à choisir, justifier et contrôler. Cherche avant de consulter les indices.

Exercice 1 — Suivre une affectation ★☆☆

Après `x = 4`, `y = x + 3`, puis `x = 10`, quelles sont les valeurs finales ?

Exercice 2 — Inclure une borne ★☆☆

Quel test Python traduit $3 \leq x < 8$?

Exercice 3 — Les bornes de range ★☆☆

Quels entiers parcourt `range(2, 6)` ? Combien y en a-t-il ?

Exercice 4 — Zéro passage ★☆☆

On initialise `x = 12`, puis exécute `while x < 10: x = x + 1`. Quelle est la valeur finale ?

Exercice 5 — Lire un indice ★☆☆

Pour `L = [12, 8, 15, 10]`, que valent `L[2]` et `len(L)` ?

Exercice 6 — Un montant calculé ★☆☆

A2 contient une quantité de 4 et B2 un prix unitaire de 12 euros. Quelle formule écrire en C2 pour calculer le montant ? Que devient ce montant si A2 passe à 5 ?

Exercice 7 — Coder une formule ★★☆

Écris une fonction Python qui renvoie $2x^2 - 5x + 1$, puis donne sa valeur en 3.

Exercice 8 — Le cas frontière ★★☆

Un code renvoie « accepté » si `x > 10`, sinon « refusé ». Que renvoie-t-il pour 10 et comment inclure 10 ?

Exercice 9 — Un accumulateur ★★☆

On initialise `s = 0` puis on ajoute `2*k` pour `k` dans `range(1, 4)`. Quelle valeur finale obtient-on ?

Exercice 10 — Compter les passages ★★☆

On part de $x = 3$ et on ajoute 4 tant que $x < 15$. Combien de passages ont lieu et quelle est la valeur finale ?

Exercice 11 — Une moyenne dans une liste ★★☆

Calcule le résultat de `sum(L)/len(L)` pour `L = [6, 10, 14]`.

Exercice 12 — Recopier avec un taux fixe ★★☆

B2 contient 80 euros et D2 contient un taux de TVA de 20 %. On veut recopier vers le bas une formule en C2 calculant le montant TTC. Écris la formule et sa version après recopie en C3.

Exercice 13 — Afficher ou renvoyer ⭐⭐⭐

Une fonction contient seulement `print(2*x)`. Pourquoi ne peut-on pas utiliser directement son résultat comme un nombre dans un autre calcul ?

Exercice 14 — L'ordre des tests ⭐⭐⭐

Un code teste d'abord `note >= 10` puis, dans `elif`, `note >= 16`. Pourquoi la seconde branche ne s'exécute-t-elle jamais ?

Exercice 15 — Un produit mal initialisé ⭐⭐⭐

Un programme veut calculer $1 \times 2 \times 3 \times 4$ mais commence par `p = 0`. Explique l'erreur et corrige-la.

Exercice 16 — Une mise à jour dans le mauvais sens ⭐⭐⭐

Un code commence avec $x = 5$ et répète $x = x - 1$ tant que $x < 10$. Pourquoi est-il incorrect pour atteindre 10 ?

Exercice 17 — Garder les couples ⭐⭐⭐

On a les abscisses `[1, 2, 3]` et les ordonnées associées `[30, 10, 20]`. Pourquoi trier seulement les ordonnées est-il une erreur ?

Exercice 18 — Tester un seuil et totaliser ⭐⭐⭐

C2 :C5 contient les montants 12, 18, 25 et 15 euros. Calcule le total avec une formule. Une cellule F2 doit afficher « gratuit » si ce total est au moins égal à 70, sinon « payant ». Propose la formule et contrôle le cas frontière.

6 Problème — Pour aller plus loin

Automatiser une planification

Une réserve commence à 20, gagne 7 unités par jour et doit atteindre au moins 100.

1. Nomme les variables et leurs valeurs initiales.
2. Écris une fonction Python renvoyant le jour et la réserve atteinte.
3. Fais la trace de trois tours puis calcule le résultat final.
4. Explique la condition de boucle et le risque d'un ajout nul.
5. Propose un tableur à deux colonnes et explique comment mémoriser les réserves dans une liste.

7 Indices des exercices

Exercice 1

Indice 1. L'affectation mémorise une valeur calculée à cet instant.

Indice 2. Changer x ensuite ne recalcule pas automatiquement y.

Exercice 2

Indice 1. Les deux conditions doivent être vraies simultanément.

Indice 2. Le signe large concerne seulement la borne 3.

Exercice 3

Indice 1. La borne supérieure n'est jamais parcourue.

Indice 2. Liste les entiers avant de les compter.

Exercice 4

Indice 1. Le test a lieu avant le bloc.

Indice 2. Une boucle while peut s'exécuter zéro fois.

Exercice 5

Indice 1. Le premier indice est 0.

Indice 2. Le nombre d'éléments n'est pas le dernier indice.

Exercice 6

Indice 1. Le montant est quantité multipliée par prix unitaire.

Indice 2. Référence les cellules plutôt que recopier les nombres dans la formule.

Exercice 7

Indice 1. La multiplication doit être écrite explicitement.

Indice 2. Le carré utilise `x**2`.

Exercice 8

Indice 1. Une égalité ne satisfait pas un signe strict.

Indice 2. Utilise le signe supérieur ou égal.

Exercice 9

Indice 1. Écris les trois termes ajoutés.

Indice 2. Le rang 4 n'est pas inclus.

Exercice 10

Indice 1. Écris une trace incluant l'état initial.

Indice 2. Compte les flèches entre les valeurs, pas les valeurs.

Exercice 11

Indice 1. Calcule séparément somme et longueur.

Indice 2. La longueur est l'effectif.

Exercice 12

Indice 1. Le prix dépend de la ligne, le taux doit rester dans D2.

Indice 2. Bloque les deux coordonnées de la cellule du taux.

Exercice 13

Indice 1. Afficher à l'écran et fournir un résultat sont deux actions distinctes.

Indice 2. Une fonction sans retour explicite renvoie `None`.

Exercice 14

Indice 1. Une chaîne s'arrête à la première condition vraie.

Indice 2. Le seuil 16 est contenu dans le seuil 10.

Exercice 15

Indice 1. Que devient 0 quand on le multiplie ?

Indice 2. Le nombre qui ne modifie pas un produit est 1.

Exercice 16

Indice 1. La mise à jour doit rapprocher de la sortie.

Indice 2. Les valeurs décroissantes restent sous 10.

Exercice 17

Indice 1. Chaque ordonnée appartient à une observation précise.

Indice 2. Trier une seule colonne modifie les observations.

Exercice 18

Indice 1. La plage comprend quatre cellules.

Indice 2. « Au moins 70 » impose le signe supérieur ou égal.

8 Corrigés détaillés

Corrigé 1

Démonstration | Solution expliquée

Comprendre la question. Il faut suivre trois instructions successives. On cherche l'état final des deux variables, et non une relation valable pour toutes leurs futures valeurs.
Construire la réponse.

1. Après `x = 4`, la valeur de `x` est 4. La variable `y` n'a pas encore reçu de valeur.
2. Pour `y = x + 3`, on calcule d'abord le membre droit avec la valeur actuelle de `x` : $4+3=7$. On mémorise donc 7 dans `y`.
3. La dernière ligne `x = 10` remplace uniquement la valeur de `x`. Aucune instruction ne redonne une valeur à `y` : elle reste à 7.

Vérifier. La table de trace contient les états $(4;\text{non définie})$, $(4;7)$ puis $(10;7)$. Obtenir $y=13$ supposerait d'exécuter une seconde fois `y = x + 3`, ce que l'énoncé ne demande pas.
Conclure. Les valeurs finales sont `x = 10` et `y = 7`.

Corrigé 2

Démonstration | Solution expliquée

Comprendre la question. La valeur x doit satisfaire en même temps une borne inférieure incluse et une borne supérieure exclue.
Construire la réponse.

1. La condition $3 \leq x$ signifie que x est au moins égal à 3 : elle s'écrit `x >= 3`.
2. La condition $x < 8$ signifie que 8 n'est pas accepté : elle s'écrit `x < 8`.
3. Le mot « et » impose les deux contraintes. On écrit `x >= 3 and x < 8`. Python permet aussi la comparaison chaînée `3 <= x < 8`.

Vérifier. Pour 3, les tests $3 \geq 3$ et $3 < 8$ sont vrais. Pour 8, le second devient faux. Pour 2, le premier est faux. Remplacer `and` par `or` accepterait 2 parce que $2 < 8$, ce qui contredirait l'énoncé.
Conclure. Les deux écritures proposées représentent exactement l'intervalle $[3;8[$.

Corrigé 3

Démonstration | Solution expliquée

Comprendre la question. On cherche les valeurs effectivement produites, puis leur nombre. La seconde borne de `range` est une limite exclue.
Construire la réponse.

1. La première valeur est 2. On augmente ensuite de 1 : viennent 3, 4 et 5.
2. La valeur suivante serait 6, mais elle est égale à la borne d'arrêt. Elle n'appartient donc pas aux valeurs parcourues.
3. On compte les quatre valeurs de la liste $2, 3, 4, 5$. On peut aussi calculer $6-2=4$ pour cette progression de pas 1.

Vérifier. La première valeur est bien 2, la dernière est inférieure à 6 et il n'y a aucun entier oublié entre elles. Compter $6-2+1$ inclurait à tort la borne 6.
Conclure. `range(2, 6)` effectue quatre passages, pour k égal à 2, 3, 4 et 5.

Corrigé 4

Démonstration | Solution expliquée

Comprendre la question. La variable commence à 12 et la boucle ne doit fonctionner que tant qu'elle est strictement inférieure à 10.
Construire la réponse.

1. L'affectation initiale donne x égal à 12.
2. Avant toute addition, Python évalue `x < 10`, donc la proposition $12 < 10$. Elle est fausse.
3. Le bloc `x = x + 1` est donc ignoré. Python passe directement à la suite du programme.

Vérifier. Une valeur finale de 13 signifierait qu'on a exécuté le bloc avant de tester. Ce serait un autre mécanisme que celui de `while`. Ici aucune instruction ne modifie la valeur initiale.
Conclure. La boucle effectue zéro passage et la valeur finale de x est 12.

Corrigé 5

Démonstration | Solution expliquée

Comprendre la question. Il faut lire une valeur à une position donnée et compter le nombre total de valeurs. Ces deux opérations ne donnent pas nécessairement le même nombre.
Construire la réponse.

1. La liste contient, dans l'ordre, 12, 8, 15 et 10.
2. Les indices correspondants sont respectivement 0, 1, 2 et 3. L'indice 2 désigne donc le troisième élément, dont la valeur est 15.
3. On compte quatre éléments ; `len(L)` vaut donc 4. La dernière position disponible a l'indice 3.

Vérifier. La relation « dernier indice = longueur moins un » donne $4 - 1 = 3$, cohérente avec la numérotation. Choisir 8 pour `L[2]` reviendrait à commencer les indices à 1, contrairement à Python.
Conclure. `L[2]` vaut 15 et `len(L)` vaut 4.

Corrigé 6

Démonstration | Solution expliquée

Comprendre la question. Le montant correspond au nombre de pièces multiplié par le prix d'une pièce. La formule doit utiliser les données des cellules pour suivre une modification.
Construire la réponse.

1. La quantité se trouve en A2 et le prix unitaire en B2. On écrit donc en C2 `=A2*B2`. Le signe égal annonce une formule et l'astérisque la multiplication.
2. Avec les données initiales, le tableur calcule $4 \times 12 = 48$. La cellule C2 affiche donc un montant de 48 euros.
3. Quand A2 passe à 5, la formule n'est pas changée mais elle lit la nouvelle valeur. Elle calcule $5 \times 12 = 60$ euros.

Vérifier. La quantité augmente d'une pièce, donc le montant doit augmenter de 12 euros : $60 - 48 = 12$. Écrire directement 48 dans C2 aurait empêché cette mise à jour automatique.
Conclure. La formule est `=A2*B2`. Le montant passe de 48 à 60 euros.

Corrigé 7

Démonstration | Solution expliquée

Comprendre la question. On doit traduire $2x^2 - 5x + 1$ en une fonction qui renvoie un nombre, puis tester cette fonction en 3.
Construire la réponse.

1. Le carré de x se traduit par `x**2`. Les multiplications implicites de l'écriture mathématique doivent devenir explicites : `2*x**2` et `5*x`.
2. On écrit :

```
def f(x):
    return 2*x**2 - 5*x + 1
```

   La ligne indentée appartient à la fonction. `return` fournit le résultat à l'appelant.
3. Lors de `f(3)`, Python utilise $x = 3$. Le carré vaut $3^2 = 9$, puis $2 \times 9 = 18$ et $5 \times 3 = 15$. Enfin $18 - 15 + 1 = 3 + 1 = 4$.

Vérifier. Le calcul manuel doit retrouver la sortie 4. Une écriture comme `2x` n'est pas une multiplication reconnue par Python ; `x*2` calcule le double et non le carré.
Conclure. La fonction ci-dessus renvoie 4 pour l'entrée 3.

Corrigé 8

Démonstration | Solution expliquée

Comprendre la question. On doit d'abord exécuter la règle existante pour la valeur 10, puis modifier la frontière acceptée.
Construire la réponse.

1. Le test est `x > 10`. En remplaçant x par 10, on obtient $10 > 10$, ce qui est faux : un nombre n'est pas strictement supérieur à lui-même.
2. La branche de réussite n'est donc pas exécutée. Le programme suit le cas « sinon » et renvoie « refusé ».
3. Pour inclure la valeur 10, on veut « supérieur ou égal ». On remplace `>` par `>=`, sans changer les messages ni les autres branches.

Vérifier. Avec le test corrigé, 9 reste refusé, 10 devient accepté et 11 reste accepté. Le seul changement à la frontière correspond bien à la demande.
Conclure. Le code initial refuse 10. Le test `x >= 10` permet de l'accepter.

Corrigé 9

Démonstration | Solution expliquée

Comprendre la question. s est une somme cumulée. À chaque passage, on ajoute $2k$ à sa valeur actuelle ; on ne remplace pas s simplement par $2k$.
Construire la réponse.

1. `range(1, 4)` fournit 1, 2 et 3. La valeur initiale de s est 0.
2. Premier tour : k vaut 1, donc le terme ajouté est $2 \times 1 = 2$. La somme devient $0 + 2 = 2$.
3. Deuxième tour : k vaut 2, donc on ajoute $2 \times 2 = 4$ à la somme précédente. Elle devient $2 + 4 = 6$.

4. Troisième tour : k vaut 3, donc on ajoute $2 \times 3 = 6$. La somme finale est $6 + 6 = 12$.

Vérifier. Le calcul direct des trois termes donne $2 + 4 + 6 = 12$. Les états $0, 2, 6, 12$ contiennent quatre valeurs, mais seulement trois passages : la première est l'initialisation.
Conclure. La valeur finale de s est 12.

Corrigé 10

Démonstration | Solution expliquée

Comprendre la question. On cherche deux informations distinctes : le nombre d'additions et la valeur de x lorsque le test devient faux.
Construire la réponse.

1. État initial : $x = 3$. Le test $3 < 15$ est vrai, donc un premier ajout de 4 donne $x = 7$.
2. Le test $7 < 15$ est encore vrai : deuxième ajout, $x = 11$.
3. Le test $11 < 15$ est vrai : troisième ajout, $x = 15$.
4. On teste de nouveau : $15 < 15$ est faux. Il n'y a pas de quatrième ajout.

Vérifier. Après n passages, $x = 3 + 4n$. Pour n égal à 3, on trouve 15 ; pour n égal à 2, on trouve 11, encore sous le seuil. La liste $3, 7, 11, 15$ comporte quatre états mais trois changements.
Conclure. La boucle effectue trois passages et s'arrête avec x égal à 15.

Corrigé 11

Démonstration | Solution expliquée

Comprendre la question. L'expression demandée est une moyenne arithmétique : elle partage la somme des données entre les trois valeurs.
Construire la réponse.

1. La liste `[6, 10, 14]` contient trois éléments, donc `len(L)` vaut 3.
2. La fonction `sum(L)` additionne les éléments : $6 + 10 = 16$, puis $16 + 14 = 30$.
3. Le quotient est donc $30/3 = 10$. En Python, la division avec `/` peut afficher ce résultat sous la forme `10.0`.

Vérifier. La moyenne 10 est entre la plus petite donnée 6 et la plus grande 14. Les écarts à 10 sont -4, 0 et 4 : ils se compensent. Ces deux contrôles confirment le calcul.
Conclure. Le résultat est 10, la moyenne des trois données.

Corrigé 12

Démonstration | Solution expliquée

Comprendre la question. Le prix HT varie selon la ligne tandis que le taux commun doit toujours être lu en D2.
Construire la réponse.

1. Un taux de 20 % vaut $0{,}20$. Le prix TTC est le prix HT plus sa TVA : $HT + 0{,}20HT = HT(1 + 0{,}20)$.
2. Le prix est en B2 et le taux en D2. On écrit `=B2*(1+$D$2)` dans C2. Les dollars empêchent la colonne D et la ligne 2 de changer lors d'une recopie.

3. En recopiant une ligne plus bas, B2 devient B3 car sa référence est relative. La formule en C3 est donc `=B3*(1+$D$2)`.
4. Pour la première ligne, on obtient $80 \times (1 + 0{,}20) = 80 \times 1{,}20 = 96$ euros.

Vérifier. La TVA seule vaut $80 \times 0{,}20 = 16$ euros ; $80 + 16 = 96$ confirme le TTC. Sans blocage, D2 deviendrait D3 et la formule pourrait lire une cellule vide ou un autre taux.
Conclure. La formule initiale calcule 96 euros TTC et conserve le taux fixe pendant la recopie.

Corrigé 13

Démonstration | Solution expliquée

Comprendre la question. On cherche pourquoi une valeur visible à l'écran ne peut pas forcément être utilisée dans une autre opération.
Construire la réponse.

1. `print(2*x)` calcule le double de x puis l'affiche. Son rôle est d'écrire à l'écran, pas de transmettre le résultat comme sortie de la fonction.
2. Une fonction qui termine sans `return` explicite renvoie la valeur spéciale `None`. Par exemple, pour x égal à 3, on voit 6, mais la sortie récupérée par l'appelant est `None`.
3. Si l'appelant tente d'ajouter 1 à cette sortie, il tente de calculer `None + 1`. Cette opération n'est pas définie comme une addition de nombres.
4. La correction consiste à écrire `return 2*x` dans la fonction. L'appelant peut ensuite mémoriser cette sortie, l'ajouter à une autre valeur, ou l'afficher avec `print`.

Vérifier. Avec le retour corrigé, l'entrée 3 donne le nombre 6 ; lui ajouter 1 donne 7. Ce test distingue la donnée transmise du texte affiché.
Conclure. Pour réutiliser le double dans un calcul, la fonction doit le renvoyer avec `return`.

Corrigé 14

Démonstration | Solution expliquée

Comprendre la question. Il faut contrôler l'ordre des branches : une note peut satisfaire plusieurs tests, mais seule la première branche vraie d'une chaîne est exécutée.
Construire la réponse.

1. Pour une note de 18, le test `note >= 10` est déjà vrai. La première branche est donc choisie immédiatement.
2. Python ne poursuit pas la recherche dans le `elif note >= 16`, même si $18 \geq 16$ est vrai. Toute note qui atteint 16 atteint nécessairement 10 ; aucune valeur ne peut donc arriver dans la seconde branche.
3. On place d'abord `if note >= 16`, puis `elif note >= 10`, et enfin le cas restant. Après l'échec du premier test, le deuxième ne concerne plus que $10 \leq$ `note` < 16.

Vérifier. Essaie 18 : première branche ; 12 : deuxième branche ; 8 : cas restant. Les notes frontières 16 et 10 sont incluses dans les branches correspondantes grâce au signe >=.
Conclure. Le test le plus restrictif doit précéder celui qui englobe ses valeurs.

Corrigé 15

Démonstration | Solution expliquée

Comprendre la question. Le programme doit accumuler un produit. Il faut choisir une valeur initiale qui ne fausse pas la multiplication.
Construire la réponse.

1. En partant de zéro, le premier calcul vaut $0 \times 1 = 0$. Le suivant vaut encore $0 \times 2 = 0$, puis $0 \times 3 = 0$ et $0 \times 4 = 0$. Tous les facteurs sont absorbés par zéro.
2. La valeur initiale correcte est 1, car multiplier par 1 conserve une valeur. On écrit donc `p = 1` avant la boucle.
3. Les états après multiplication sont $1 \times 1 = 1$, $1 \times 2 = 2$, $2 \times 3 = 6$, puis $6 \times 4 = 24$.
4. Le parcours doit être `range(1, 5)` pour inclure le facteur 4 et exclure 5.

Vérifier. Le calcul direct $1 \times 2 \times 3 \times 4 = 24$ confirme la sortie. Avec une somme on choisirait zéro, mais le choix doit dépendre de l'opération répétée.
Conclure. Initialiser p à 1 permet d'obtenir le produit attendu, 24.

Corrigé 16

Démonstration | Solution expliquée

Comprendre la question. Le programme doit atteindre 10 depuis 5, mais sa mise à jour fait diminuer la valeur. Il faut déterminer si le test pourra devenir faux.
Construire la réponse.

1. Au départ, $5 < 10$ est vrai. Le premier passage donne $5 - 1 = 4$, qui reste inférieur à 10.
2. Les passages suivants donnent 3, 2, 1, puis des valeurs de plus en plus petites. Après n passages, $x = 5 - n$.
3. Pour tout entier $n \geq 0$, $5 - n \leq 5 < 10$. La condition est donc toujours vraie : la boucle ne s'arrête pas dans ce modèle entier.
4. Pour atteindre 10 en augmentant depuis 5, on remplace la mise à jour par `x = x + 1`. Les états deviennent 5, 6, 7, 8, 9, 10.

Vérifier. À 9, le test reste vrai ; à 10, il devient faux. La boucle corrigée fait cinq passages, ce que confirme $5 + 5 = 10$.
Conclure. La mise à jour initiale provoque une boucle infinie ; ajouter 1 au lieu de le retrancher résout le problème posé.

Corrigé 17

Démonstration | Solution expliquée

Comprendre la question. Chaque abscisse et l'ordonnée de même position décrivent une observation. On doit conserver ces couples pour ne pas modifier les données.
Construire la réponse.

1. Les couples initiaux sont $(1; 30)$, $(2; 10)$ et $(3; 20)$. Le premier peut représenter une mesure 30 à l'instant 1.
2. Trier seulement les ordonnées donne `[10, 20, 30]`, tandis que les abscisses restent `[1, 2, 3]`. Les couples deviennent $(1; 10)$, $(2; 20)$ et $(3; 30)$.

3. Le premier couple n'a plus la même mesure, et aucun des trois couples nouveaux ne correspond au couple initial de même abscisse. On a créé artificiellement une tendance croissante.
4. Pour réordonner les observations, on trie des couples complets, en déplaçant toujours ensemble les deux coordonnées. Par exemple, trier selon l'ordonnée donnerait $(2;10)$, $(3;20)$, $(1;30)$.

Vérifier. Après le tri correct, les trois couples d'origine sont toujours présents, seul leur ordre a changé. C'est ce contrôle qu'un tri d'une seule colonne ne satisfait pas.
Conclure. Il faut conserver les couples associés ; trier une seule colonne falsifie la relation étudiée.

Corrigé 18

Démonstration / Solution expliquée

Comprendre la question. Il faut additionner quatre montants puis appliquer une condition qui inclut exactement le seuil 70.
Construire la réponse.

1. La plage `C2:C5` comprend C2, C3, C4 et C5. La formule `=SOMME(C2:C5)` donne $12+18+25+15=30+40=70$ euros.
2. « Au moins 70 » se traduit par `>=70`. Un test strict `>70` refuserait à tort l'égalité.
3. On peut écrire en F2 `=SI(SOMME(C2:C5)>=70;"gratuit";"payant")`. Le premier argument est le test, le deuxième le texte si le test est vrai et le troisième le texte sinon.
4. Avec le total actuel, $70 \geq 70$ est vrai. La cellule doit donc afficher « gratuit ».

Vérifier. Pour un total de 69, le résultat serait « payant » ; pour 70 et 71, il serait « gratuit ». Ces trois essais vérifient les deux branches et la frontière.
Conclure. Le total est 70 euros et la condition de gratuité est satisfaite.

9 Corrigé du problème

1. **Comprendre la question.** Les variables doivent distinguer le temps écoulé, la quantité actuelle et les paramètres.

   Construire la réponse.

   (a) Au départ, aucun jour n'est écoulé : jour vaut 0. La réserve initiale vaut 20.

   (b) L'ajout quotidien 7 et l'objectif 100 sont deux paramètres fixes : ils ne doivent pas être modifiés dans la boucle.

   (c) Après chaque tour, jour compte les jours terminés et reserve contient la quantité disponible à ce moment.

   Vérifier. Cette convention associe bien le rang zéro à la réserve de départ ; elle évite un décalage d'un jour.

   Conclure. Initialisation : jour $= 0$, reserve $= 20$, ajout $= 7$, objectif $= 100$.

2. **Comprendre la question.** La boucle doit continuer tant que la réserve n'a pas encore atteint le seuil.

   Construire la réponse.

   (a)

```python
def seuil(depart, ajout, objectif):
    if ajout <= 0 and depart < objectif:
        raise ValueError("Seuil inaccessible")
    jour, reserve = 0, depart
    while reserve < objectif:
        reserve += ajout
        jour += 1
    return jour, reserve
```

   (b) Le contrôle initial rejette un ajout nul ou négatif lorsque le départ ne suffit pas : le seuil ne pourrait jamais être atteint.

   (c) La condition while est strictement inférieure au seuil. À chaque tour, on ajoute la quantité quotidienne puis on augmente le jour de 1.

   (d) Le return est placé après la boucle : la fonction renvoie le premier jour où la condition devient fausse et la réserve correspondante.

   Vérifier. Si la réserve initiale suffit, la boucle effectue zéro tour et renvoie le jour zéro. Le compteur avance une seule fois par ajout.

   Conclure. L'appel avec les paramètres (20, 7, 100) calcule le premier jour où la réserve vaut au moins 100.

3. **Comprendre la question.** Une trace manuelle vérifie le programme, puis la formule de suite contrôle le résultat final.

   Construire la réponse.

   (a) Après les jours 1, 2 et 3, les réserves valent 27, 34 et 41.

   (b) Après n jours, $u_n = 20 + 7n$. La condition $u_n \geq 100$ donne $n \geq (100 - 20)/7 = 11{,}4286$.

   (c) Le plus petit entier adapté est 12. La réserve vaut alors 104.

   Vérifier. La veille, au jour 11, elle vaut 97, encore inférieure à 100. Le jour 12 respecte le seuil.

   Conclure. La fonction renvoie le jour 12 et une réserve de 104.

4. **Comprendre la question.** La condition de continuation est le contraire de l'objectif d'arrêt.

   Construire la réponse.

   (a) Atteindre au moins 100 signifie reserve supérieure ou égale au seuil. On continue donc tant que reserve est strictement inférieure.

(b) Si l'on utilisait une condition inférieure ou égale, une égalité exacte provoquerait un tour supplémentaire inutile.

(c) Avec un ajout nul et un départ insuffisant, reserve ne changerait jamais : la condition resterait vraie indéfiniment. Un ajout négatif éloignerait également du seuil.

Vérifier. Le test placé avant la boucle couvre ces cas inaccessibles. Il ne rejette pas un départ déjà suffisant, qui ne nécessite aucune progression.

Conclure. La condition stricte permet le bon arrêt ; le contrôle d'accessibilité évite une boucle sans fin.

5. **Comprendre la question.** Le tableur et la liste conservent l'historique au lieu de ne garder que la dernière valeur.

   Construire la réponse.

   (a) En tableur, A2 contient 0 et B2 contient 20. A3 reçoit `=A2+1` et B3 `=B2+7`, puis on recopie vers le bas.

   (b) Chaque formule utilise la ligne précédente : les références doivent donc rester relatives.

   (c) En Python, on initialise `historique = [depart]`. Après chaque mise à jour de la réserve, on exécute `historique.append(reserve)`.

   (d) La valeur au rang n de la liste correspond alors à la réserve après n jours ; le départ occupe déjà le premier élément, d'indice zéro.

   Vérifier. Au dernier jour 12, la liste contient 13 valeurs, car elle inclut aussi le jour zéro.

   Conclure. Les deux outils rendent visibles le départ, chaque progression et le premier franchissement.

10 Variante autonome et bilan

Une réserve de 90 perd 7 unités par jour. Adapte la boucle pour trouver quand elle est au plus égale à 40.

À toi d'essayer : Vérifie ta démarche

Résous la variante, puis explique pourquoi ta méthode s'applique.

Tu peux chercher, prendre un indice ou lire l'explication, à ton rythme.

Indices, p. 32 Explication, p. 34

- Je peux expliquer : variables, affectations et fonctions.
- Je peux expliquer : choisir avec une condition.
- Je peux expliquer : répéter un nombre connu de fois.
- Je peux expliquer : répéter jusqu'à un objectif.
- Je peux expliquer : listes, données et contrôles numériques.
- Je peux expliquer : construire et recopier une formule de tableur.

11 Fiche-mémoire

Variables, affectations et fonctions.

1. Définis les entrées, leur unité et la sortie attendue. Pour une formule de coût, indique si les frais fixes sont toujours facturés.
2. Écris les opérations avec les symboles de Python : `*` pour multiplier, `**` pour une puissance, un point pour un nombre décimal.
3. Exécute les instructions dans l'ordre. Après chaque affectation, recopie les valeurs inchangées dans une table de trace.
4. Pour une fonction, distingue sa définition et son appel. Vérifie que `return` fournit bien la valeur dont le reste du programme a besoin.
5. Teste une entrée simple à la main et une valeur frontière, comme zéro si cette valeur appartient au domaine.

Choisir avec une condition.

1. Traduis la règle en français et repère les mots « au moins », « strictement », « entre », « et » et « ou ».
2. Écris une condition vraie exactement pour les cas souhaités ; vérifie chaque borne séparément.
3. Classe les branches de manière à ce que les premiers tests ne capturent pas à tort les cas des branches suivantes.
4. Teste une valeur dans chaque branche et chaque valeur frontière. Explique le chemin suivi avant d'observer la sortie.

Répéter un nombre connu de fois.

1. Écris la liste des valeurs produites par `range` sur un petit cas. Vérifie le début et la borne exclue.
2. Définis le rôle de chaque variable : indice, compteur ou résultat accumulé.
3. Initialise avant la boucle : zéro pour une somme, un pour un produit. Justifie ce choix par l'opération répétée.
4. À chaque tour, calcule le membre droit avec les valeurs actuelles puis range le nouveau résultat.
5. Compare la trace à un calcul manuel et contrôle ce qui est placé après la boucle.

Répéter jusqu'à un objectif.

1. Écris la condition d'arrêt en français, puis sa négation pour obtenir la condition du `while`.
2. Fixe l'état initial et vérifie si le test est déjà faux. Si oui, explique pourquoi zéro passage est correct.

3. Identifie la mise à jour de toutes les variables nécessaires, y compris le compteur de passages.
4. Fais une trace contenant l'état initial, chaque modification et le premier test faux.
5. Pour un premier rang, contrôle le rang retenu et son prédécesseur. Explique la terminaison.

Listes, données et contrôles numériques.

1. Nomme les données et précise leur ordre. Si deux colonnes vont ensemble, conserve leur association.
2. Compte les éléments pour connaître la longueur, puis numérote leurs indices à partir de zéro.
3. Pour construire une liste, initialise-la avant la boucle et ajoute les résultats au fur et à mesure.
4. Pour une moyenne, calcule séparément somme et effectif et vérifie que l'effectif est strictement positif.
5. Contrôle une première valeur, une dernière valeur et la longueur totale avant d'interpréter le résultat.

Construire et recopier une formule de tableur.

1. Donne une signification et une unité à chaque colonne. Vérifie que les données d'une ligne concernent le même article ou la même observation.
2. Écris le calcul en mots, puis remplace les nombres par leurs adresses de cellules. Commence la formule par =.
3. Pour chaque référence, décide si elle doit se déplacer à la recopie. Bloque un paramètre fixe avec les signes dollar nécessaires.
4. Contrôle la formule de la première ligne, puis celle de la deuxième après recopie. Ne te contente pas d'un résultat numérique plausible.
5. Pour une fonction SI, teste un cas vrai, un cas faux et l'égalité à la frontière. Pour une somme, vérifie que la plage comprend toutes les lignes utiles.

12 Indices des pauses et des preuves

Chercher avec un indice fait partie du travail. Tu peux revenir à l'énoncé avec le lien sous chaque aide.

Pause 1 — Mets la notion à l'épreuve

Indice 1 — Une piste.

Le symbole de multiplication et celui de puissance sont différents en Python.

Indice 2 — Un pas de plus.

Le carré signifie x multiplié par lui-même ; essaie les propositions sur x égal à 3.

Revenir à la recherche, p. 5

Pause 2 — Mets la notion à l'épreuve

Indice 1 — Une piste.

On demande de comparer deux valeurs, pas d'en mémoriser une.

Indice 2 — Un pas de plus.

Le signe unique sert déjà à l'affectation.

Revenir à la recherche, p. 6

Pause 3 — Mets la notion à l'épreuve

Indice 1 — Une piste.

Quand une seule borne est donnée, le début est zéro.

Indice 2 — Un pas de plus.

Écris les entiers en t'arrêtant juste avant 5.

Revenir à la recherche, p. 8

Pause 4 — Mets la notion à l'épreuve

Indice 1 — Une piste.

Demande-toi si le bloc est exécuté lorsque la condition est fausse dès le départ.

Indice 2 — Un pas de plus.

Python examine le test une première fois avant d'entrer, puis de nouveau après chaque mise à jour.

Revenir à la recherche, p. 9

Pause 5 — Mets la notion à l'épreuve

Indice 1 — Une piste.

Le premier élément a l'indice zéro.

Indice 2 — Un pas de plus.

Numérote les trois éléments 0, 1 et 2 avant de lire la position demandée.

Revenir à la recherche, p. 11

Pause 6 — Mets la notion à l'épreuve

Indice 1 — Une piste.

Le tableur doit distinguer une donnée saisie d'une expression à calculer.

Indice 2 — Un pas de plus.

Compare une cellule contenant le texte 2+3 à une cellule contenant une formule.

Revenir à la recherche, p. 12

Vérifie ta démarche

Indice 1 — Une piste.

Repère les données utiles et l'inconnue. Reviens à la méthode correspondante dans la boîte à outils.

Indice 2 — Un pas de plus.

Contrôle les unités, les bornes et la plausibilité du résultat avant de lire la correction.

Revenir à la recherche, p. 29

13 Explications des pauses et des preuves

Lis une étape, puis essaie de poursuivre avant de lire la suivante. Les calculs ci-dessous complètent le cours.

Pause 1 — Mets la notion à l'épreuve

La bonne réponse est `x**2`. Pour x égal à 3, elle donne $3^2 = 9$. `2*x` donnerait 6, tandis que `2x` n'est pas une syntaxe valide. Le symbole ^ ne désigne pas une puissance en Python.

Revenir à la recherche, p. 5

Pause 2 — Mets la notion à l'épreuve

Le test d'égalité s'écrit ==. Ainsi `x == 5` produit vrai ou faux selon la valeur actuelle de x. `x = 5` change cette valeur ; il ne pose pas une question. L'opérateur `:=` concerne une affectation dans une expression et ne remplace pas ce test.

Revenir à la recherche, p. 6

Pause 3 — Mets la notion à l'épreuve

`range(5)` fournit 0, 1, 2, 3 et 4 : cinq valeurs au total. La liste 1 à 5 correspondrait à `range(1, 6)`. Inclure à la fois 0 et 5 donnerait six valeurs, ce qui ne correspond pas à la commande.

Revenir à la recherche, p. 8

Pause 4 — Mets la notion à l'épreuve

La condition est testée avant chaque passage. Une boucle dont le test initial est faux fait zéro passage. Si le test est vrai, le bloc s'exécute puis un nouveau test décide du passage suivant.

Revenir à la recherche, p. 9

Pause 5 — Mets la notion à l'épreuve

Dans `[3, 8, 11]`, l'indice 0 désigne 3, l'indice 1 désigne 8 et l'indice 2 désigne 11. La réponse est donc 8. Le nombre 1 est une position ; ce n'est ni le nombre d'éléments, ni une valeur à chercher dans la liste.

Revenir à la recherche, p. 11

Pause 6 — Mets la notion à l'épreuve

Une formule commence par le signe =. Ainsi =2+3 calcule 5. Pour qu'elle s'adapte aux données, on utilise des références, par exemple `=A2+B2`.

Revenir à la recherche, p. 12

Vérifie ta démarche

Comprendre la question. La réserve diminue : la condition et la mise à jour doivent toutes deux être adaptées.

Construire la réponse.

1. On initialise `jour = 0` et `reserve = 90`.
2. On continue `while reserve > 40:`, puis on effectue `reserve -= 7` et `jour += 1`. L'égalité à 40 doit déjà arrêter la boucle.
3. Après n jours, $u_n = 90 - 7n$. L'objectif $90 - 7n \leq 40$ donne $-7n \leq -50$, puis $n \geq 50/7 \approx 7{,}143$.
4. Le premier entier est 8. La veille, $u_7 = 41 > 40$; au jour 8, $u_8 = 34 \leq 40$.

Vérifier. La baisse de 7 est strictement positive en valeur absolue : tant que le seuil n'est pas atteint, chaque tour rapproche la réserve de l'objectif.

Conclure. Le premier jour adapté est le jour 8, avec une réserve de 34 unités.

Revenir à la recherche, p. 29