Plaquette 2 – Variables, boucles bornées et fonctions

Algorithmique et Python — Première spé maths

Corrections détaillées

Boîte à outils

Affectation. $x = 5$ place la valeur 5 dans la variable x. L'affectation $x = x + 1$ **lit** l'ancienne valeur puis **écrit** la nouvelle.

Boucle bornée.

```
for k in range(n):
    instruction
```

range(n) parcourt 0, 1, …, $n-1$ — soit n valeurs, et **pas** $n+1$.

Écriture	Valeurs parcourues	Nombre
range(5)	$0, 1, 2, 3, 4$	5
range(1, 5)	$1, 2, 3, 4$	4
range(1, 6)	$1, 2, 3, 4, 5$	5
range(0, 10, 2)	$0, 2, 4, 6, 8$	5

Contrôler un accumulateur. La somme des entiers sert de vérification :

$$1 + 2 + \cdots + n = \frac{n(n+1)}{2}$$

Fonction.

```
def aire(L, l):
    return L * l
```

Le mot-clé *return* **renvoie** une valeur et **quitte** la fonction. Une fonction sans *return* renvoie *None*.

Opérateurs utiles. *//* division entière · *%* reste · ** puissance · *abs()* valeur absolue.

Tracer une exécution. On dresse un tableau avec une colonne par variable et une ligne par tour de boucle — c'est la méthode la plus sûre pour prévoir un affichage.

Trois erreurs à traquer.

1. Confondre *range(n)* (n valeurs, de 0 à $n-1$) et une boucle allant de 1 à n. 2. Placer le *print* **dans** la boucle au lieu d'après (ou l'inverse). 3. Oublier d'**initialiser** un accumulateur avant la boucle.

Correction de l'exercice 1 – Trace d'exécution

range(5) fait prendre à k les valeurs 0, 1, 2, 3 et 4 : la boucle fait cinq tours, et 5 n'est pas atteint. À chaque tour, on ajoute k à l'ancienne valeur de s. On suit l'évolution des variables dans un tableau d'exécution :

mathlvl.fr — Tous droits réservés. Toute reproduction, diffusion, revente ou exploitation commerciale, même partielle, est strictement interdite.
Tout contrevenant s'expose à des poursuites en vertu du Code de la propriété intellectuelle (art. L.335-2 et suivants).

tour	k	s après
début	·	0
1	0	0
2	1	1
3	2	3
4	3	6
5	4	10

Le script calcule donc

$$0 + 1 + 2 + 3 + 4 = 10$$

Piège : croire que k monte jusqu'à 5. Avec *range(1, 6)*, on obtiendrait $1 + 2 + 3 + 4 + 5 = 15$. Le script affiche 10.

Réponse. $s = 10$.

Correction de l'exercice 2 – Compter les tours

range(a, b, p) part de a et avance de p tant que la valeur reste **strictement inférieure** à b ; *range(b)* part de 0 avec un pas de 1.

a) k prend les valeurs de 0 à 9 : 10 tours.

b) k prend les valeurs de 3 à 9, soit $9 - 3 + 1 = 7$ tours.

c) k prend les valeurs 0, 5, 10 et 15 ; la suivante, 20, n'est pas strictement inférieure à 20 : 4 tours.

d) On part de 10, qui n'est pas inférieur à 3 : la boucle ne s'exécute **jamais**, 0 tour.

Le cas d) ne provoque aucune erreur, Python exécute simplement une boucle vide : c'est ce qui le rend dangereux. Pour descendre de 10 à 4, il faudrait un pas négatif, *range(10, 3, -1)*. Réponses : 10, 7, 4 et 0.

Réponse. a) 10 · b) 7 · c) 4 · d) 0 (boucle vide).

Correction de l'exercice 3 – Écrire une fonction

On utilise un **accumulateur** s, initialisé à 0 avant la boucle, auquel on ajoute chaque entier de 1 à n. Pour inclure n, la boucle doit aller jusqu'à $n + 1$, puisque *range* exclut sa borne de fin.

```
def somme(n):
    s = 0
    for k in range(1, n + 1):
        s = s + k
    return s
```

On teste d'abord sur de petites valeurs dont on connaît le résultat : *somme(3)* renvoie 6 et *somme(4)* renvoie 10. Pour $n = 100$, on compare avec la formule de la somme des entiers :

$$1 + 2 + \cdots + 100 = \frac{100 \times 101}{2} = 5050$$

Conclusion : *somme(100)* renvoie 5050. On pourrait aussi écrire *return n * (n + 1) // 2*, sans boucle ; la division entière // donne un entier, alors que / donnerait 5050.0.

Réponse. *somme(100)* = 5050 (ou directement $n(n+1)//2$).

mathlvl.fr — Tous droits réservés. Toute reproduction, diffusion, revente ou exploitation commerciale, même partielle, est strictement interdite.
Tout contrevenant s'expose à des poursuites en vertu du Code de la propriété intellectuelle (art. L.335-2 et suivants).

Correction de l'exercice 4 – Boucle imbriquée

La boucle intérieure est entièrement exécutée à **chaque** tour de la boucle extérieure. Pour chacune des 4 valeurs de i, la variable j prend 3 valeurs, donc le *print* s'exécute

$$4 \times 3 = 12 \text{ fois}$$

Les couples affichés sont $(0,0)$, $(0,1)$, $(0,2)$, puis $(1,0)$, et ainsi de suite jusqu'à $(3,2)$: c'est j qui varie le plus vite.

Plus généralement, deux boucles imbriquées de n et m tours exécutent $n \times m$ fois leur contenu. C'est ainsi qu'on énumère par exemple les 36 résultats de deux dés, avec deux boucles *range(1, 7)*. Le *print* est exécuté 12 **fois**.

Réponse. 12 fois (4×3).

Correction de l'exercice 5 – Accumuler un produit

On calcule le produit $1 \times 2 \times \cdots \times n$ avec un accumulateur p. Il faut l'initialiser à 1, l'élément neutre de la multiplication : initialisé à 0, il resterait nul. Un accumulateur de somme, lui, s'initialise à 0.

```
def factorielle(n):
    p = 1
    for k in range(1, n + 1):
        p = p * k
    return p
```

Pour $n = 6$, p prend successivement les valeurs 1, 2, 6, 24, 120 et 720 :

$$6! = 1 \times 2 \times 3 \times 4 \times 5 \times 6 = 720$$

Pour $n = 0$, la boucle ne tourne pas et la fonction renvoie 1, ce qui correspond à la convention $0! = 1$. La factorielle grandit très vite : $10! = 3\,628\,800$. **Conclusion** : $6! = 720$.

Réponse. $6! = 720$ (accumulateur initialisé à 1).

Correction de l'exercice 6 – Compter selon une condition

On parcourt les entiers de 1 à 100 et l'on augmente un compteur chaque fois que l'entier est divisible par 3, c'est-à-dire quand le reste de sa division par 3, *k % 3*, est nul.

```
c = 0
for k in range(1, 101):
    if k % 3 == 0:
        c = c + 1
print(c)
```

Les multiples de 3 entre 1 et 100 sont 3, 6, …, $99 = 3 \times 33$: il y en a 33, ce que donne aussi directement la division entière *100 // 3*.

Piège : écrire *if k % 3 = 0* avec un seul signe égal provoque une erreur de syntaxe, car = est une affectation et non un test. Le script affiche 33.

Réponse. 33 multiples (ou directement *100 // 3*).

Correction de l'exercice 7 – Maximum d'une liste

On parcourt la liste en gardant en mémoire le plus grand élément rencontré jusque-là. On initialise cette mémoire avec le **premier élément** de la liste, et non avec 0 : pour une liste de nombres tous négatifs, comme $[-5, -2, -9]$, l'initialisation à 0 renverrait 0, qui n'est même pas dans la liste.

mathlvl.fr — Tous droits réservés. Toute reproduction, diffusion, revente ou exploitation commerciale, même partielle, est strictement interdite.
Tout contrevenant s'expose à des poursuites en vertu du Code de la propriété intellectuelle (art. L.335-2 et suivants).

```
def maximum(L):
    m = L[0]
    for x in L:
        if x > m:
            m = x
    return m
```

Sur $[3, 7, 2, 9, 5]$, *m* vaut successivement 3, 7 puis 9 : la fonction renvoie 9. Elle fait une comparaison par élément ; on ne peut pas faire mieux, puisqu'un élément non lu pourrait être le maximum.

Pour une liste vide, *L[0]* provoque une erreur : il faut alors traiter ce cas à part. **Contrôle** : *maximum([-5, -2, -9])* renvoie bien -2.

Réponse. Parcours avec mémorisation, initialisée à *L[0]* ; coût linéaire.

Correction de l'exercice 8 – Somme des carrés

On accumule les carrés k^2 pour k allant de 1 à 20 ; la puissance s'écrit *k**2* en Python.

```
s = 0
for k in range(1, 21):
    s = s + k**2
print(s)
```

Le script affiche 2870. On vérifie avec la formule donnée, pour $n = 20$:

$$\frac{20 \times 21 \times 41}{6} = \frac{17\,220}{6} = 2870$$

Piège : *k^2* n'est pas une puissance en Python, c'est une opération sur les bits ; *3^2* vaut 1 et non 9.

Contrôle sur un petit cas : pour $n = 3$, le script donnerait $1 + 4 + 9 = 14$, et la formule $\frac{3 \times 4 \times 7}{6} = 14$. La somme vaut 2870.

Réponse. 2870, conforme à $\frac{20 \times 21 \times 41}{6}$.

Correction de l'exercice 9 – Fonction à deux paramètres

Le terme de rang n d'une suite arithmétique a une forme explicite, $u_n = u_0 + nr$: la fonction se contente de la calculer.

```
def terme(u0, r, n):
    return u0 + n * r
```

Pour $u_0 = 7$ et $r = -3$:

$$u_{20} = 7 + 20 \times (-3) = 7 - 60 = -53$$

On peut aussi partir de u_0 et ajouter n fois la raison avec une boucle *for k in range(n): u = u + r*. Cette version est plus longue, mais c'est la seule possible pour une suite récurrente sans forme explicite simple, comme $u_{n+1} = 1{,}05u_n + 3$.

Contrôle : *terme(7, -3, 1)* renvoie 4 et *terme(7, -3, 2)* renvoie 1, comme $7 - 3$ et $7 - 6$. On trouve $u_{20} = -53$.

Réponse. $u_{20} = -53$.

mathlvl.fr — Tous droits réservés. Toute reproduction, diffusion, revente ou exploitation commerciale, même partielle, est strictement interdite.
Tout contrevenant s'expose à des poursuites en vertu du Code de la propriété intellectuelle (art. L.335-2 et suivants).

Correction de l’exercice 10 – Moyenne d’une liste

On additionne les éléments avec un accumulateur, puis on divise par le nombre d’éléments, *len(L)*.

```
def moyenne(L):
    s = 0
    for x in L:
        s = s + x
    return s / len(L)
```

Pour $[12, 15, 9, 20, 14]$, la somme vaut 70 et l’effectif 5 :

$$\overline{x} = \frac{70}{5} = 14$$

On utilise la division `/`, qui donne un nombre décimal : la division entière `//` tronquerait les moyennes non entières, $[1, 2]$ donnerait 1 au lieu de 1,5. Pour une liste vide, on diviserait par 0 et Python signalerait une erreur.

Contrôle : 14 est bien compris entre le minimum, 9, et le maximum, 20. La moyenne vaut 14.

Réponse. Moyenne = 14 (somme 70 divisée par 5).

Correction de l’exercice 11 – Tableau de valeurs

On définit la fonction f en Python, puis on l’évalue pour $x = 0, 1, \ldots, 5$ avec une boucle *range(6)*.

```
def f(x):
    return x*x - 3*x + 1

for k in range(6):
    print(k, f(k))
```

Le script affiche le tableau suivant :

x	0	1	2	3	4	5
$f(x)$	1	-1	-1	1	5	11

Par exemple, $f(5) = 25 - 15 + 1 = 11$. Le tableau montre deux changements de signe, entre 0 et 1 puis entre 2 et 3 : f a deux racines, $\frac{3 \pm \sqrt{5}}{2}$, soit environ 0,38 et 2,62, car son discriminant vaut

$$\Delta = 9 - 4 = 5$$

Pour un pas de 0,5, il faudrait écrire $x = k * 0.5$ dans la boucle : *range* n’accepte que des entiers.

Réponse. 1 ; -1 ; -1 ; 1 ; 5 ; 11 pour $x = 0$ à 5.

Correction de l’exercice 12 – Corriger un script

Première erreur : $s = 0$ est placé **dans** la boucle, donc la somme est remise à zéro à chaque tour. À la fin, s ne contient que la dernière valeur de k, et le script affiche 9.

Seconde erreur : *range(10)* parcourt les entiers de 0 à 9 ; il faut *range(1, 11)* pour aller jusqu’à 10.

```
s = 0
for k in range(1, 11):
```

mathlvl.fr — Tous droits réservés. Toute reproduction, diffusion, revente ou exploitation commerciale, même partielle, est strictement interdite.
Tout contrevenant s’expose à des poursuites en vertu du Code de la propriété intellectuelle (art. L.335-2 et suivants).

```
    s = s + k
print(s)
```

Le script corrigé affiche 55, conforme à la formule :

$$1 + 2 + \cdots + 10 = \frac{10 \times 11}{2} = 55$$

Règle : un accumulateur s'initialise **avant** la boucle, jamais dedans. Tester un script sur un cas dont on connaît la réponse, par exemple $1 + 2 + 3 = 6$, révèle immédiatement ce genre d'erreur.

Réponse. *s = 0* doit être **avant** la boucle, et *range(10)* doit devenir *range(1, 11)* : le résultat est 55.

Correction de l'exercice 13 – Fonction avec condition

Pour deux nombres, un seul test suffit. Si $a > b$, l'instruction *return a* quitte la fonction ; la ligne suivante n'est donc atteinte que si $a \leq b$, et le *else* est inutile.

```
def max2(a, b):
    if a > b:
        return a
    return b

def max3(a, b, c):
    return max2(max2(a, b), c)
```

Pour trois nombres, on **réutilise** *max2* : le plus grand de a et b est comparé à c. Cela fait deux comparaisons, et si *max2* est juste, *max3* l'est aussi : c'est l'intérêt de découper un programme en fonctions.

$$\max(3, 7, 5) = \max(\max(3, 7), 5) = \max(7, 5) = 7$$

Contrôle : *max3(9, 2, 4)* renvoie 9 et *max3(1, 1, 1)* renvoie 1.

Réponse. *max2* par un test, puis *max3(a, b, c) = max2(max2(a, b), c)*.

Correction de l'exercice 14 – Simulation simple

On simule chaque lancer avec *randint(1, 6)*, qui renvoie un entier entre 1 et 6, **bornes incluses**. On compte les 6, puis on divise par le nombre de lancers.

```
from random import randint

c = 0
for i in range(1000):
    if randint(1, 6) == 6:
        c = c + 1
print(c / 1000)
```

La probabilité d'obtenir 6 vaut $\frac{1}{6} \approx 0{,}167$: le script affiche une fréquence proche de cette valeur, différente à chaque exécution. Sur des échantillons de taille $n = 1000$, la fréquence s'écarte rarement de plus de

$$2 \times \sqrt{\frac{\frac{1}{6} \times \frac{5}{6}}{1000}} \approx 0{,}024$$

de $\frac{1}{6}$: on obtient presque toujours une valeur entre 0,143 et 0,190.

Erreur classique : *randint(0, 6)* simulerait un dé à sept faces.

mathlvl.fr — Tous droits réservés. Toute reproduction, diffusion, revente ou exploitation commerciale, même partielle, est strictement interdite.
Tout contrevenant s'expose à des poursuites en vertu du Code de la propriété intellectuelle (art. L.335-2 et suivants).

Réponse. Environ 0,167, typiquement entre 0,143 et 0,190.

Correction de l'exercice 15 – Table de multiplication

On calcule chaque produit $7 \times k$, on l'affiche **dans** la boucle et on l'ajoute à une somme ; le total s'affiche **après** la boucle.

```
s = 0
```

```
for k in range(1, 11):
    p = 7 * k
    print(7, "x", k, "=", p)
    s = s + p
print("somme :", s)
```

Les produits sont 7, 14, 21, …, 70. Leur somme se calcule sans tout additionner, en mettant 7 en facteur :

$$7 \times (1 + 2 + \cdots + 10) = 7 \times 55 = 385$$

Contrôle : $7 + 14 + 21 + 28 + 35 + 42 + 49 + 56 + 63 + 70 = 385$. Le script affiche les dix produits, puis la somme 385.

Réponse. Les dix produits, de somme $7 \times 55 = 385$.

Correction de l'exercice 16 – Synthèse : étude d'une suite

a) On part de $u_0 = 2$ et l'on applique n fois la relation de récurrence :

```
def u(n):
```

```
    x = 2
    for k in range(n):
        x = 3*x - 4
    return x
```

b) $u_1 = 3 \times 2 - 4 = 2$, puis $u_2 = 2$, et ainsi de suite : la suite est **constante**, égale à 2. En effet, 2 est un **point fixe** de la relation : si $u_n = 2$, alors $u_{n+1} = 3 \times 2 - 4 = 2$. On le retrouve en résolvant

$$3x - 4 = x \Leftrightarrow x = 2$$

Partir d'une autre valeur change tout : avec $u_0 = 3$, on obtient 5, 11, 29, 83, 245.

c) On ajoute chaque terme **avant** de calculer le suivant, pour compter u_0 et s'arrêter à u_{10} :

```
s = 0
```

```
x = 2
for k in range(11):
    s = s + x
    x = 3*x - 4
print(s)
```

Les 11 termes valent 2 : la somme vaut 22.

Réponse. b) la suite est **constante** égale à 2 (point fixe) · c) somme = 22.

mathlvl.fr — Tous droits réservés. Toute reproduction, diffusion, revente ou exploitation commerciale, même partielle, est strictement interdite.
Tout contrevenant s'expose à des poursuites en vertu du Code de la propriété intellectuelle (art. L.335-2 et suivants).