Plaquette 3 – Boucles while, seuils et algorithmes classiques

Algorithmique et Python — Seconde

Corrections détaillées

Boîte à outils

Quelle boucle choisir ?

Situation	Boucle
nombre de tours **connu**	*for*
arrêt sur une **condition**	*while*
recherche d'un seuil	*while*

Structure d'une boucle while.

```
initialisation
while condition:
    corps
    mise à jour
```

Les trois parties sont indispensables : sans **mise à jour**, la condition ne change jamais et la boucle est **infinie**.

Les trois erreurs classiques.

Erreur	Conséquence
oublier la mise à jour	boucle infinie
condition inversée	la boucle ne démarre pas
mauvaise initialisation du compteur	résultat décalé de 1

Condition d'arrêt et valeur de sortie. À la sortie d'un *while*, la condition est **fausse**. C'est ce qui permet de conclure : « on sort quand le capital dépasse le seuil, donc la valeur finale le dépasse ». Pour un capital de 2000 € placé à 4 %, le programme cherche le premier n tel que

$$2000 \times 1{,}04^{n} > 3000$$

et l'on contrôle que l'année précédente ne convenait pas encore.

Opérateurs utiles.

Écriture	Sens
a // b	quotient entier
a % b	reste
*a ** n*	puissance
abs(x)	valeur absolue
and, *or*, *not*	et, ou, non

Compter les tours. On ajoute une variable initialisée à 0, incrémentée à chaque passage. Sa valeur finale est le **nombre de tours effectués**, pas le nombre de tests.

Dérouler à la main. On construit un tableau avec une colonne par variable et une ligne par tour : c'est la méthode la plus sûre pour comprendre ou déboguer un *while*.

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 1 – Dérouler une boucle while

On suit a et b tour par tour, en testant la condition $a < 50$ **avant** chaque tour.

Tour	départ	1	2	3	4	5
a	3	6	12	24	48	96
b	1	2	3	4	5	6

Après le quatrième tour, $a = 48$: le test $48 < 50$ est encore vrai, donc la boucle fait un cinquième tour et a passe à 96. Le test $96 < 50$ est faux, la boucle s’arrête et le programme affiche

96 et 6

La boucle ne s’arrête pas au moment où a dépasse 50, mais au test **suivant** : c’est pourquoi la valeur finale dépasse largement le seuil. **Contrôle** : $a = 3 \times 2^5 = 96$, et b vaut le nombre de tours plus 1, puisqu’il partait de 1.

Réponse. Il affiche 96 et 6 (soit 5 tours effectués).

Correction de l’exercice 2 – Seuil de population

Une hausse de 3 % revient à multiplier par 1,03. On ne sait pas combien d’années il faudra : on utilise une boucle *while* avec un compteur d’années.

```
p = 5000
```

```
n = 0
while p <= 8000:
    p = p * 1.03
    n = n + 1
print(n, p)
```

La population vaut environ 5796 habitants après 5 ans, 6720 après 10 ans, 7790 après 15 ans, puis

$$5000 \times 1{,}03^{16} \approx 8023{,}5$$

après 16 ans. Comme $5000 \times 1{,}03^{15} \approx 7789{,}8 \leq 8000$, l’année 16 est bien la première où la population dépasse 8000 habitants. Le programme affiche un nombre décimal ; pour un nombre d’habitants, on l’arrondit avec *round(p)*, ce qui donne 8024.

Réponse. $n = 16$ ans, la population atteignant alors ≈ 8024 habitants.

Correction de l’exercice 3 – Division euclidienne par soustractions

Le quotient de 47 par 6 est le nombre de fois qu’on peut retirer 6 de 47 ; ce qui reste à la fin est le reste.

```
a = 47
```

```
b = 6
q = 0
while a >= b:
    a = a - b
    q = q + 1
print(q, a)
```

a prend les valeurs 41, 35, 29, 23, 17, 11 et 5, pendant que q va de 1 à 7. Le test $5 \geq 6$ est faux : la boucle s’arrête et le programme affiche 7 et 5.

$$47 = 6 \times 7 + 5 \qquad \text{avec} \quad 0 \leq 5 < 6$$

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).

Contrôle : *47 // 6* vaut 7 et *47 % 6* vaut 5. La condition doit être $a >= b$: avec $a > b$, la division de 48 par 6 s'arrêterait sur un « reste » de 6, ce qui n'est pas un vrai reste. Cette méthode fait autant de tours que le quotient : elle est très lente pour de grands nombres.

Réponse. $q = 7$ et reste 5, car $47 = 6 \times 7 + 5$.

Correction de l'exercice 4 – Lister les diviseurs d'un entier

Un entier d divise n quand le reste de la division de n par d est nul, c'est-à-dire quand *n % d == 0*. On teste tous les entiers de 1 à n :

```
n = 36
```

```
compteur = 0
for d in range(1, n + 1):
    if n % d == 0:
        print(d)
        compteur = compteur + 1
print("nombre de diviseurs :", compteur)
```

Pour $n = 36$, le programme affiche 1, 2, 3, 4, 6, 9, 12, 18, 36, puis *nombre de diviseurs : 9*. Pour $n = 84$: 1, 2, 3, 4, 6, 7, 12, 14, 21, 28, 42, 84, soit 12 diviseurs.

Contrôle : les diviseurs vont **par paires** de produit n : 1×36, 2×18, 3×12, 4×9, et 6×6 seul, car 36 est un carré. C'est pourquoi 36 a un nombre **impair** de diviseurs. **Amélioration** : il suffirait de tester d jusqu'à $\sqrt{n}$, puis d'afficher aussi $\frac{n}{d}$.

Réponse. 36 a 9 diviseurs $(1, 2, 3, 4, 6, 9, 12, 18, 36)$; 84 en a 12.

Correction de l'exercice 5 – Recherche par dichotomie

La dichotomie coupe l'intervalle en deux et garde la moitié qui contient la solution. Au départ, $2^3 = 8 < 20$ et $3^3 = 27 > 20$: la solution est bien entre 2 et 3.

Étape	**milieu** m	m^3	**nouvel intervalle**
1	2,5	15,625 < 20	[2,5 ; 3]
2	2,75	≈ 20,797 > 20	[2,5 ; 2,75]
3	2,625	≈ 18,088 < 20	[2,625 ; 2,75]
4	2,6875	≈ 19,411 < 20	[2,6875 ; 2,75]

Quand $m^3 < 20$, la solution est à droite de m et l'on remplace la borne a par m ; sinon on remplace b.

```
a = 2
```

```
b = 3
while b - a > 0.001:
    m = (a + b) / 2
    if m ** 3 < 20:
        a = m
    else:
        b = m
print(a, b)
```

Chaque étape divise la largeur par 2 : après 4 étapes, elle vaut $\frac{1}{16} = 0{,}0625$, et après 10 étapes

$$\frac{1}{2^{10}} = \frac{1}{1024} < 0{,}001$$

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).

Dix étapes suffisent donc pour une précision de 0,001, là où un balayage au pas de 0,001 en demanderait jusqu'à 1000. **Contrôle** : $2{,}714^3 \approx 19{,}99$.

Réponse. Après quatre étapes : [2,6875 ; 2,75], qui contient la solution ≈ 2,714.

Correction de l'exercice 6 – Répéter une opération jusqu'à un seuil

On ne connaît pas le nombre d'étapes : on répète l'opération **tant que** la valeur ne dépasse pas 500, en comptant les étapes.

```
u = 1
n = 0
while u <= 500:
    u = 3 * u + 2
    n = n + 1
print(n, u)
```

Les valeurs successives sont 5, 17, 53, 161, 485. Comme $485 \leq 500$, la boucle fait encore un tour :

$$3 \times 485 + 2 = 1457 > 500$$

Le programme affiche 6 et 1457. La valeur saute de 485 à 1457 : chaque étape multiplie à peu près par 3, si bien que les écarts triplent. **Contrôle** : on remarque que chaque valeur augmentée de 1 est une puissance de 3 multipliée par 2 ; par exemple $1457 + 1 = 1458 = 2 \times 729 = 2 \times 3^6$.

Réponse. 6 étapes, pour la valeur 1457.

Correction de l'exercice 7 – Années bissextiles

La règle se traduit directement par une expression booléenne, avec le reste de la division euclidienne : *a % 4 == 0* signifie « *a* est divisible par 4 ».

```
def bissextile(a):
    return (a % 4 == 0 and a % 100 != 0) or a % 400 == 0

def nombre_bissextiles(debut, fin):
    c = 0
    for a in range(debut, fin + 1):
        if bissextile(a):
            c = c + 1
    return c
```

2024 est divisible par 4 et pas par 100 : bissextile. 2100 est divisible par 100 mais pas par 400 : non bissextile. 2000 est divisible par 400 : bissextile, bien que divisible par 100.

De 2001 à 2100, les multiples de 4 vont de 2004 à 2100, soit 25 années, dont il faut retirer 2100. La fonction renvoie donc

$$25 - 1 = 24$$

Les parenthèses comptent : sans elles, Python évalue *and* avant *or*, ce qui donne ici le même résultat, mais il vaut mieux les écrire pour que la règle soit lisible.

Réponse. 2000 et 2024 sont bissextiles, pas 2100 ; de 2001 à 2100, il y a 24 années bissextiles.

Correction de l'exercice 8 – Somme de chiffres

Le dernier chiffre d'un entier est le reste de sa division par 10, *n % 10* ; on le retire avec *n // 10*. On recommence tant qu'il reste des chiffres.

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 sommeChiffres(n):
    s = 0
    while n > 0:
        s = s + n % 10
        n = n // 10
    return s
```

Pour 4739, les couples $(s\,;n)$ successifs sont $(9\,;473)$, $(12\,;47)$, $(19\,;4)$ puis $(23\,;0)$; la boucle s'arrête quand $n = 0$.

$$4 + 7 + 3 + 9 = 23$$

La boucle fait autant de tours que l'entier a de chiffres. Comme 23 n'est pas multiple de 3, 4739 n'est pas divisible par 3 : en effet $4739 = 3 \times 1579 + 2$. Avec la condition *n >= 0*, la boucle ne s'arrêterait jamais, puisque n resterait égal à 0.

Réponse. La somme des chiffres de 4739 vaut 23.

Correction de l'exercice 9 – Conversion en binaire

On divise par 2 en notant chaque reste, jusqu'à obtenir un quotient nul.

```
n = 43
while n > 0:
    print(n % 2)
    n = n // 2
```

$43 = 2 \times 21 + 1$, $21 = 2 \times 10 + 1$, $10 = 2 \times 5 + 0$, $5 = 2 \times 2 + 1$, $2 = 2 \times 1 + 0$ et $1 = 2 \times 0 + 1$: le programme affiche 1, 1, 0, 1, 0, 1.

Le premier reste est le chiffre des **unités** en binaire : on lit donc les restes à l'envers.

$$43 = 101011_2 = 32 + 8 + 2 + 1$$

Contrôle : $1 \times 32 + 0 \times 16 + 1 \times 8 + 0 \times 4 + 1 \times 2 + 1 \times 1 = 43$, et l'écriture a 6 chiffres car $32 \leq 43 < 64$. Python le confirme : *bin(43)* renvoie *0b101011*.

Réponse. Restes 1, 1, 0, 1, 0, 1, soit $43 = 101011_2$.

Correction de l'exercice 10 – Des valeurs qui se stabilisent

On garde la valeur précédente pour mesurer l'écart avec la nouvelle ; *abs* donne l'écart sans signe.

```
u = 10
n = 0
ecart = 1
while ecart > 0.01:
    v = u / 2 + 1
    ecart = abs(v - u)
    u = v
    n = n + 1
print(n, u)
```

Les valeurs successives sont 6, 4, 3, 2,5, 2,25… et les écarts 4, 2, 1, 0,5, 0,25 : chaque écart est la moitié du précédent. À la neuvième étape, l'écart vaut $\approx 0{,}0156$; à la dixième, il vaut $\approx 0{,}0078 < 0{,}01$ et la boucle s'arrête avec $u \approx 2{,}008$.

Les valeurs se rapprochent du nombre qui ne change pas quand on applique l'opération :

$$x = \frac{x}{2} + 1 \Leftrightarrow \frac{x}{2} = 1 \Leftrightarrow x = 2$$

Contrôle : $\frac{2}{2} + 1 = 2$. Ici l'écart entre la valeur et 2 est divisé par 2 à chaque étape : il

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).

vaut 8 au départ et $\frac{8}{1024} \approx 0{,}008$ après 10 étapes.

Réponse. 10 étapes, pour une valeur ≈ 2,008 ; les valeurs se rapprochent de 2.

Correction de l’exercice 11 – Remboursement d’un prêt

Chaque mois, la dette augmente de 0,5 % (on la multiplie par 1,005), puis on retire la mensualité. On continue tant qu’il reste une dette.

```
d = 3000
```

```
n = 0
while d > 0:
    d = d * 1.005 - 200
    n = n + 1
print(n, d)
```

La dette vaut 2815 € après 1 mois, environ 2629,07 € après 2 mois, 1107,82 € après 10 mois et 125,74 € après 15 mois. Au seizième mois, le calcul donne environ −73,63 € : la dette est soldée, le dernier versement de 200 € étant trop élevé. Il faut donc 16 mensualités, la dernière valant seulement

$$125{,}74 \times 1{,}005 \approx 126{,}37 \text{ €}$$

Au total, on rembourse $15 \times 200 + 126{,}37 = 3126{,}37$ € : le crédit a coûté 126,37 €, soit environ 4,2 % de la somme empruntée. Avec la condition *d >= 200*, la boucle s’arrêterait avant ce dernier versement partiel.

Réponse. $n = 16$ mensualités, la dernière étant de ≈ 126,37 € (coût du crédit ≈ 126,37 €).

Correction de l’exercice 12 – Boucles imbriquées

La boucle sur i prend les valeurs 1, 2, 3 et, pour chacune, la boucle sur j prend aussi les valeurs 1, 2, 3 : on teste $3 \times 3 = 9$ couples. Le test *i != j* écarte les trois couples (1 ; 1), (2 ; 2) et (3 ; 3).

$$9 - 3 = 6 \text{ lignes}$$

Dans l’ordre d’affichage : (1 ; 2), (1 ; 3), (2 ; 1), (2 ; 3), (3 ; 1), (3 ; 2). Ce sont tous les couples **ordonnés** de deux valeurs différentes : pour chacune des 3 valeurs de i, il reste 2 choix pour j, d’où $3 \times 2 = 6$. Avec *range(1, 5)*, on aurait $4 \times 3 = 12$ lignes. Pour ne pas compter à la fois (1 ; 2) et (2 ; 1), on ferait partir j de $i + 1$: il resterait 3 lignes, comme le nombre de poignées de main entre 3 personnes.

Réponse. Six lignes : (1 ; 2), (1 ; 3), (2 ; 1), (2 ; 3), (3 ; 1), (3 ; 2).

Correction de l’exercice 13 – Approximation de la racine carrée

On applique trois fois la formule en partant de $x_0 = 3$.

$$x_1 = \frac{1}{2}\left(3 + \frac{10}{3}\right) = \frac{1}{2} \times \frac{19}{3} = \frac{19}{6} \approx 3{,}166667$$

Ensuite $\frac{10}{3{,}166667} \approx 3{,}157895$, d’où $x_2 \approx \frac{3{,}166667 + 3{,}157895}{2} \approx 3{,}162281$. Puis $\frac{10}{3{,}162281} \approx 3{,}162274$, d’où $x_3 \approx 3{,}162277$.

```
a = 10
```

```
x = 3
for i in range(3):
    x = (x + a / x) / 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).

```
print(x)
```

Or $\sqrt{10} \approx 3{,}1622777$: trois étapes donnent déjà 6 décimales exactes. L'erreur passe d'environ 0,004 après la première étape à environ 0,000003 après la deuxième : le nombre de décimales exactes double à peu près à chaque étape, bien plus vite que la dichotomie. **Contrôle** : $3{,}162277^2 \approx 9{,}99999$.

Réponse. $x_1 \approx 3{,}166667$, $x_2 \approx 3{,}162281$, $x_3 \approx 3{,}162277$ — soit $\sqrt{10}$ à 10^{-6} près.

Correction de l'exercice 14 – Compter les diviseurs

On teste tous les entiers d de 1 à n et l'on compte ceux pour lesquels le reste $n \,\%\, d$ est nul.

```
def nbDiviseurs(n):
    c = 0
    for d in range(1, n + 1):
        if n % d == 0:
            c = c + 1
    return c
```

Les diviseurs de 36 sont 1, 2, 3, 4, 6, 9, 12, 18, 36, soit 9 diviseurs. On peut les grouper par paires de produit 36 : 1×36, 2×18, 3×12, 4×9, et 6×6 qui ne compte qu'une fois. C'est pourquoi un carré parfait a toujours un nombre **impair** de diviseurs.

Les diviseurs de 37 sont seulement 1 et 37 :

9 diviseurs pour 36 2 diviseurs pour 37

Un entier qui, comme 37, n'a que 2 diviseurs n'est divisible que par 1 et par lui-même. Grâce aux paires, on pourrait s'arrêter à $\sqrt{n}$ et compter deux diviseurs à chaque fois, sans oublier le cas du carré.

Réponse. 36 a 9 diviseurs, 37 en a 2 (1 et 37).

Correction de l'exercice 15 – Éviter une boucle infinie

Dans le programme A, n n'est jamais modifié : la condition $n < 100$ reste vraie et le programme affiche 1 sans fin. Il faut ajouter $n = n + 1$ dans la boucle ; il affiche alors les entiers de 1 à 99.

Dans le programme B, la mise à jour $i = i + 1$ éloigne i de la condition d'arrêt : i grandit et $i > 0$ reste vraie pour toujours. Il faut écrire $i = i - 1$. La boucle ajoute alors 10, 9, …, 1 et s'arrête quand $i = 0$:

$$s = 10 + 9 + \cdots + 1 = \frac{10 \times 11}{2} = 55$$

Avant d'écrire un *while*, on vérifie donc deux choses : la quantité testée change à chaque tour, et elle change dans le bon sens, pour que la condition finisse par devenir fausse. Pour B, une boucle *for i in range(10, 0, -1)* serait plus sûre : son nombre de tours est fixé d'avance.

Réponse. A : il manque $n = n + 1$ · B : il faut $i = i - 1$ (la somme vaut alors 55).

Correction de l'exercice 16 – Synthèse : simulation d'épargne

a) Chaque année, on calcule les intérêts sur le capital **avant** le versement, qui a lieu en fin d'année, puis on ajoute 600 €.

```
def capital(n):
    c = 1500
    for i in range(n):
```

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).

```
        c = c * 1.025 + 600
    return c
```

b) Après 1 an : $1500 \times 1{,}025 + 600 = 2137{,}50$ €. Après 2 ans : $2137{,}50 \times 1{,}025 + 600 \approx 2790{,}94$ €. Puis 3460,71 € et 4147,23 €, et après 5 ans :

$$4147{,}23 \times 1{,}025 + 600 \approx 4850{,}91 \text{ €}$$

Sur ces 5 ans, on a versé $1500 + 5 \times 600 = 4500$ € : les intérêts représentent environ 350,91 €.

c) Le nombre d'années est inconnu : on utilise un *while*.

```
c = 1500
```

```
n = 0
while c <= 10000:
    c = c * 1.025 + 600
    n = n + 1
print(n, c)
```

Le capital vaut environ 9458,21 € après 11 ans, puis 10 294,67 € après 12 ans : le programme affiche $n = 12$.

d) Sans versement, le capital vaut $1500 \times 1{,}025^n$ et il doit être multiplié par $\dfrac{10\,000}{1500} \approx 6{,}67$. Comme $1{,}025^{76} \approx 6{,}53$ et $1{,}025^{77} \approx 6{,}69$, il faut 77 ans. Les versements réguliers font donc gagner plus de 60 ans : en 12 ans, ils apportent 7200 € sur les 8795 € gagnés, contre seulement 1595 € pour les intérêts.

Réponse. b) 2137,50 €, 2790,94 € puis ≈ 4850,91 € · c) $n = 12$ ans (≈ 10 295 €) · d) 77 ans sans versement, soit plus de six fois plus long.

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).