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*
parcours d'une liste	*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 ».

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*.

Correction de l'exercice 1 – Dérouler une boucle while

— **Initialisation** : $a = 3$ et $b = 1$ ✓
— **Test** : $3 < 50$, vrai ✓
— **Tour 1** : $a = 6$ et $b = 2$ ✓
— **Tour 2** : $a = 12$ et $b = 3$ ✓
— **Tour 3** : $a = 24$ et $b = 4$ ✓
— **Tour 4** : $a = 48$ et $b = 5$ ✓
— **Test suivant** : $48 < 50$, vrai ✓
— **Tour 5** : $a = 96$ et $b = 6$ ✓
— **Test final** : $96 < 50$, **faux** ✓
— **Affichage** :

96 et 6

— **Interprétation de** b : le nombre de tours plus 1, puisqu'il partait de 1 ✓
— **Nombre de tours effectués** : 5 ✓
— **Contrôle** : $a = 3 \times 2^5 = 96$ ✓
— **Contrôle de l'arrêt** : $3 \times 2^4 = 48 < 50$, il fallait bien un cinquième doublement ✓
— **Piège d'analyse** : la boucle ne s'arrête **pas** dès que a dépasse 50, mais au test **suivant** ✓
— **Conséquence** : la valeur finale de a dépasse largement le seuil ✓
— **Avec un seuil de** 100 : a finirait à 192 et b à 7 ✓
— **Contrôle** : $3 \times 2^6 = 192$ ✓

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

Correction de l'exercice 2 – Seuil de population

— **Structure** : un *while* avec la population et un compteur d'années ✓

```
p = 5000
```

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

— **Après** 5 **ans** : ≈ 5796,37 ✓
— **Après** 10 **ans** : ≈ 6719,58 ✓
— **Après** 15 **ans** : ≈ 7789,84 ✓
— **Après** 16 **ans** : ≈ 8023,53 ✓
— **Résultat** :

$$n = 16 \text{ ans, pour } \approx 8024 \text{ habitants}$$

— **Contrôle** : $5000 \times 1{,}03^{16} \approx 8023{,}53$ ✓
— **Contrôle de l'année précédente** : $5000 \times 1{,}03^{15} \approx 7789{,}84 < 8000$ ✓
— **Cohérence** : 16 est bien la première année de dépassement ✓
— **Croissance en** 16 **ans** : $\dfrac{8023{,}53}{5000} \approx 1{,}605$, soit +60,5 % ✓
— **Temps de doublement** : la population atteindrait 10 000 après 24 ans ✓
— **Contrôle** : $5000 \times 1{,}03^{24} \approx 10\,164{,}0$ ✓

— **Contrôle de l'année précédente** : $5000 \times 1{,}03^{23} \approx 9867{,}9 < 10\,000$ ✓
— **Règle des** 70 : le temps de doublement vaut environ $\frac{70}{3} \approx 23{,}3$ ans ✓
— **Cohérence** : l'estimation est excellente ✓
— **Piège du type** : la population affichée est un **décimal**, il faudrait l'arrondir pour un compte d'habitants ✓
— **Instruction utile** : *round(p)* ou *int(p)* ✓

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

Correction de l'exercice 3 – Division euclidienne par soustractions

— **Principe** : soustraire le diviseur tant que c'est possible ✓

```
a = 47
```

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

— **Initialisation** : $a = 47$, $q = 0$ ✓
— **Tour 1** : $a = 41$, $q = 1$ ✓
— **Tour 2** : $a = 35$, $q = 2$ ✓
— **Tour 3** : $a = 29$, $q = 3$ ✓
— **Tour 4** : $a = 23$, $q = 4$ ✓
— **Tour 5** : $a = 17$, $q = 5$ ✓
— **Tour 6** : $a = 11$, $q = 6$ ✓
— **Tour 7** : $a = 5$, $q = 7$ ✓
— **Test suivant** : $5 \geq 6$ est **faux** ✓
— **Affichage** :

$$q = 7 \text{ et reste } 5$$

— **Contrôle de l'égalité euclidienne** : $47 = 6 \times 7 + 5$ ✓
— **Contrôle** : $42 + 5 = 47$ ✓
— **Contrôle du reste** : $0 \leq 5 < 6$ ✓
— **Contrôle par les opérateurs natifs** : *47 // 6* vaut 7 et *47 % 6* vaut 5 ✓
— **Piège de l'inégalité** : avec *while a > b*, on s'arrêterait à $a = 5$ aussi ici, mais pour 48 on obtiendrait un reste de 6, incorrect ✓
— **Contrôle de ce piège** : $48 = 6 \times 7 + 6$ n'est pas une division euclidienne valide ✓
— **Efficacité** : cette méthode fait q tours, elle est **lente** pour de grands nombres ✓
— **Exemple** : diviser 10^9 par 3 demanderait plus de 300 millions de tours ✓

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

Correction de l'exercice 4 – PGCD par soustractions

— **Principe** : le PGCD de a et b égale celui de $a - b$ et b ✓

```
a = 84
```

```
b = 36
while a != b:
    if a > b:
        a = a - b
```

```
    else:
        b = b - a
print(a)
```

— **Tour 1** : $a = 84 > 36$, donc $a = 48$ ✓
— **Tour 2** : $48 > 36$, donc $a = 12$ ✓
— **Tour 3** : $12 < 36$, donc $b = 24$ ✓
— **Tour 4** : $12 < 24$, donc $b = 12$ ✓
— **Test suivant** : $a = b = 12$, la boucle s'arrête ✓
— **Résultat** :

$$\text{PGCD}(84\,;36) = 12$$

— **Contrôle par les décompositions** : $84 = 2^2 \times 3 \times 7$ et $36 = 2^2 \times 3^2$ ✓
— **Facteurs communs** : $2^2 \times 3 = 12$ ✓
— **Contrôle par divisibilité** : $\frac{84}{12} = 7$ et $\frac{36}{12} = 3$ ✓
— **Cohérence** : 7 et 3 sont premiers entre eux ✓
— **PPCM associé** : $\frac{84 \times 36}{12} = 252$ ✓
— **Piège de la condition** : avec *while a > b*, la boucle s'arrêterait trop tôt ✓
— **Piège du cas nul** : si l'un des deux vaut 0, la boucle est infinie ✓
— **Protection** : traiter le cas 0 à part ✓
— **Version plus rapide, par restes** : *while b != 0: a, b = b, a % b* ✓
— **Contrôle de cette version** : 84mod36 = 12, puis 36mod12 = 0, donc PGCD = 12 ✓
— **Efficacité comparée** : 2 tours au lieu de 4 ✓

Réponse. $\text{PGCD}(84\,;36) = 12$, atteint en 4 soustractions.

Correction de l'exercice 5 – Recherche par dichotomie

— **Principe** : couper l'intervalle en deux et garder la moitié contenant la solution ✓
— **Contrôle initial** : $2^3 = 8 < 20$ et $3^3 = 27 > 20$ ✓
— **Étape 1 — milieu** 2,5 : $2{,}5^3 = 15{,}625 < 20$ ✓
— **Nouvel intervalle** : $[2{,}5\,;3]$ ✓
— **Étape 2 — milieu** 2,75 : $2{,}75^3 \approx 20{,}797 > 20$ ✓
— **Nouvel intervalle** : $[2{,}5\,;2{,}75]$ ✓
— **Étape 3 — milieu** 2,625 : $2{,}625^3 \approx 18{,}088 < 20$ ✓
— **Nouvel intervalle** : $[2{,}625\,;2{,}75]$ ✓
— **Étape 4 — milieu** 2,6875 : $2{,}6875^3 \approx 19{,}411 < 20$ ✓
— **Nouvel intervalle** :

$$[2{,}6875\,;2{,}75]$$

— **Amplitude atteinte** : 0,0625, soit $\frac{1}{16}$ ✓
— **Contrôle** : chaque étape divise l'amplitude par 2, donc $\frac{1}{2^4}$ ✓

```
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)
```

— **Valeur exacte** : $\sqrt[3]{20} \approx 2{,}7144$ ✓
— **Cohérence** : elle appartient bien à $[2{,}6875\,;2{,}75]$ ✓
— **Nombre d'étapes pour une précision de** 0,001 : il faut $\dfrac{1}{2^n} < 0{,}001$, soit $n = 10$ ✓
— **Contrôle** : $\dfrac{1}{1024} \approx 0{,}00098 < 0{,}001$ ✓
— **Efficacité** : 10 étapes suffisent, contre 1000 pour un balayage au pas de 0,001 ✓

Réponse. Après quatre étapes : $[2{,}6875\,;2{,}75]$, encadrant $\sqrt[3]{20} \approx 2{,}714$.

Correction de l'exercice 6 – Suite et seuil

— **Structure** : un *while* sur la valeur du terme ✓

```
u = 1
```

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

— **Déroulé** : $u_0 = 1$ ✓
— $u_1 = 5$ ✓
— $u_2 = 17$ ✓
— $u_3 = 53$ ✓
— $u_4 = 161$ ✓
— $u_5 = 485$ ✓
— **Test** : $485 \leq 500$, la boucle continue ✓
— $u_6 = 1457$ ✓
— **Test** : $1457 > 500$, arrêt ✓
— **Résultat** :

$$n = 6, \text{ avec } u_6 = 1457$$

— **Contrôle** : $3 \times 485 + 2 = 1457$ ✓
— **Observation** : le terme **saute** de 485 à 1457 ✓
— **Explication** : la suite croît **géométriquement**, les écarts triplent ✓
— **Forme explicite** : $u_n = 2 \times 3^n - 1$ ✓
— **Contrôle pour** $n = 0$: $2 - 1 = 1$ ✓
— **Contrôle pour** $n = 5$: $2 \times 243 - 1 = 485$ ✓
— **Contrôle pour** $n = 6$: $2 \times 729 - 1 = 1457$ ✓
— **Cohérence** : la formule explicite confirme le déroulé ✓
— **Premier rang dépassant** 10^6 : $n = 13$, car $2 \times 3^{13} - 1 = 3\,188\,645$ ✓

Réponse. $n = 6$, avec $u_6 = 1457$ (la formule explicite est $u_n = 2 \times 3^n - 1$).

Correction de l'exercice 7 – Nombres premiers

— **Principe** : chercher un diviseur ; s'il n'y en a aucun, le nombre est premier ✓
— **Optimisation** : il suffit de tester jusqu'à $\sqrt{n}$ ✓
— **Justification** : si $n = ab$ avec $a \leq b$, alors $a \leq \sqrt{n}$ ✓

```
def estPremier(n):
    d = 2
    while d * d <= n:
        if n % d == 0:
            return False
        d = d + 1
    return True
```

— **Condition de boucle** : d d <= n* évite de calculer une racine carrée ✓
— **Test de** 91 : $d = 2$, reste 1 ✓
— $d = 3$: reste 1 ✓
— $d = 4$: reste 3 ✓
— $d = 5$: reste 1 ✓
— $d = 6$: reste 1 ✓
— $d = 7$: $91 = 7 \times 13$, reste 0 ✓
— **Résultat** : 91 **n'est pas** premier ✓
— **Test de** 97 : les diviseurs testés vont de 2 à 9, car $10^2 = 100 > 97$ ✓
— **Aucun diviseur trouvé** ✓
— **Résultat** :

97 est premier

— **Contrôle** : $\sqrt{97} \approx 9{,}849$ ✓
— **Nombre de tests pour** 97 : 8 diviseurs, de 2 à 9 ✓
— **Sans l'optimisation** : il en faudrait 95 ✓
— **Gain** : un facteur 12 ✓
— **Amélioration supplémentaire** : ne tester que 2 puis les impairs ✓
— **Contrôle** : cela diviserait encore par deux le nombre de tests ✓

Réponse. $91 = 7 \times 13$ n'est pas premier ; 97 est premier.

Correction de l'exercice 8 – Somme de chiffres

— **Principe** : extraire le dernier chiffre, puis le retirer ✓
— **Dernier chiffre** : *n % 10* ✓
— **Retirer ce chiffre** : *n // 10* ✓

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

— **Déroulé** : $n = 4739$, $s = 0$ ✓
— **Tour 1** : $s = 9$ et $n = 473$ ✓
— **Tour 2** : $s = 12$ et $n = 47$ ✓
— **Tour 3** : $s = 19$ et $n = 4$ ✓
— **Tour 4** : $s = 23$ et $n = 0$ ✓
— **Test suivant** : $0 > 0$ est faux, arrêt ✓
— **Résultat** :

23

— **Contrôle** : $4 + 7 + 3 + 9 = 23$ ✓

— **Nombre de tours** : 4, le nombre de chiffres ✓
— **Application — divisibilité par** 3 : 23 n'est pas multiple de 3, donc 4739 non plus ✓
— **Contrôle** : $4739 = 3 \times 1579 + 2$ ✓
— **Application — divisibilité par** 9 : également non ✓
— **Cas de** $n = 0$: la boucle ne tourne pas, le résultat est 0 ✓
— **Cohérence** : la somme des chiffres de 0 vaut bien 0 ✓
— **Piège** : avec *while n >= 0*, la boucle serait **infinie**, n restant à 0 ✓
— **Somme itérée** : $2 + 3 = 5$, c'est la « racine numérique » de 4739 ✓
— **Contrôle** : $4739 \bmod 9 = 5$ ✓

Réponse. La somme des chiffres de 4739 vaut 23.

Correction de l'exercice 9 – Conversion en binaire

— **Principe** : diviser par 2 en notant les restes ✓

```
n = 43
```

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

— **Tour 1** : $43 = 2 \times 21 + 1$, reste 1, puis $n = 21$ ✓
— **Tour 2** : $21 = 2 \times 10 + 1$, reste 1, puis $n = 10$ ✓
— **Tour 3** : $10 = 2 \times 5 + 0$, reste 0, puis $n = 5$ ✓
— **Tour 4** : $5 = 2 \times 2 + 1$, reste 1, puis $n = 2$ ✓
— **Tour 5** : $2 = 2 \times 1 + 0$, reste 0, puis $n = 1$ ✓
— **Tour 6** : $1 = 2 \times 0 + 1$, reste 1, puis $n = 0$ ✓
— **Restes affichés dans l'ordre** : 1, 1, 0, 1, 0, 1 ✓
— **Reconstitution** : on lit les restes **à l'envers** ✓
— **Écriture binaire** :

$$43 = 101011_2$$

— **Contrôle** : $32 + 0 + 8 + 0 + 2 + 1 = 43$ ✓
— **Détail** : $1 \times 32 + 0 \times 16 + 1 \times 8 + 0 \times 4 + 1 \times 2 + 1 \times 1$ ✓
— **Nombre de bits** : 6 ✓
— **Contrôle** : $2^5 = 32 \leq 43 < 64 = 2^6$ ✓
— **Vérification native** : *bin(43)* renvoie *0b101011* ✓
— **Pourquoi lire à l'envers ?** Le premier reste est le bit de **poids faible** ✓
— **Amélioration du programme** : accumuler dans une chaîne, avec *s = str(n % 2) + s* ✓

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

Correction de l'exercice 10 – Convergence d'une suite

— **Structure** : comparer le terme courant au précédent ✓

```
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)
```

— **Déroulé** : $u_0 = 10$ ✓
— $u_1 = 6$, écart 4 ✓
— $u_2 = 4$, écart 2 ✓
— $u_3 = 3$, écart 1 ✓
— $u_4 = 2{,}5$, écart 0,5 ✓
— $u_5 = 2{,}25$, écart 0,25 ✓
— **Observation** : l'écart est **divisé par** 2 à chaque étape ✓
— $u_9 = 2{,}015625$, écart $\approx 0{,}0156$ ✓
— $u_{10} = 2{,}0078125$, écart $\approx 0{,}0078$ ✓
— **Arrêt** : $0{,}0078 < 0{,}01$ ✓
— **Résultat** :

$$n = 10, \text{ avec } u_{10} \approx 2{,}008$$

— **Limite** : la suite converge vers 2 ✓
— **Justification** : la limite ℓ vérifie $\ell = \frac{\ell}{2} + 1$, donc $\ell = 2$ ✓
— **Forme explicite** : $u_n = 2 + \frac{8}{2^n}$ ✓
— **Contrôle pour** $n = 0$: $2 + 8 = 10$ ✓
— **Contrôle pour** $n = 10$: $2 + \frac{8}{1024} \approx 2{,}0078$ ✓
— **Rôle de* abs*** : garantir un écart positif, même si la suite dépassait la limite ✓
— **Ici** : la suite décroît, l'écart serait positif de toute façon ✓

Réponse. $n = 10$ et $u_{10} \approx 2{,}008$; la limite est 2.

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

— **Mécanisme** : chaque mois, on ajoute les intérêts puis on retire la mensualité ✓

```
d = 3000
```

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

— **Après** 1 **mois** : $3000 \times 1{,}005 - 200 = 2815$ € ✓
— **Après** 2 **mois** : $\approx 2629{,}07$ € ✓
— **Après** 5 **mois** : $\approx 2065{,}70$ € ✓
— **Après** 10 **mois** : $\approx 1107{,}82$ € ✓
— **Après** 15 **mois** : $\approx 125{,}74$ € ✓
— **Après** 16 **mois** : $\approx -73{,}63$ € ✓
— **Interprétation du négatif** : le seizième versement de 200 € est **trop élevé** ✓
— **Résultat** :

$$n = 16 \text{ mensualités}$$

— **Dernier versement réel** : $125{,}74 \times 1{,}005 \approx 126{,}37$ € ✓
— **Total remboursé** : $15 \times 200 + 126{,}37 \approx 3126{,}37$ € ✓
— **Coût du crédit** : $3126{,}37 - 3000 = 126{,}37$ € ✓
— **Coût relatif** : $\frac{126{,}37}{3000} \approx 4{,}21\,\%$ du capital ✓

— **Taux annuel équivalent** : $1{,}005^{12} - 1 \approx 6{,}17\,\%$ ✓
— **Contrôle de vraisemblance** : 4,2 % du capital sur 16 mois est cohérent avec un taux annuel de 6,2 % ✓
— **Piège de la condition** : avec *while d >= 200*, on s'arrêterait avant le dernier versement partiel ✓
— **Avec des mensualités de** 150 **€** : il faudrait 22 mois ✓
— **Contrôle** : la dette après 21 mois serait ≈ 18,66 € ✓
— **Dernier versement dans ce cas** : ≈ 18,75 € ✓

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

Correction de l'exercice 12 – Boucles imbriquées

— **Valeurs de** i : 1, 2, 3 ✓
— **Valeurs de** j : 1, 2, 3 pour chaque i ✓
— **Nombre de couples testés** : $3 \times 3 = 9$ ✓
— **Condition** : afficher seulement si $i \neq j$ ✓
— **Couples exclus** : (1 ; 1), (2 ; 2), (3 ; 3) ✓
— **Nombre d'exclusions** : 3 ✓
— **Nombre de lignes affichées** :

$$9 - 3 = 6$$

— **Affichage pour** $i = 1$: (1 ; 2) puis (1 ; 3) ✓
— **Pour** $i = 2$: (2 ; 1) puis (2 ; 3) ✓
— **Pour** $i = 3$: (3 ; 1) puis (3 ; 2) ✓
— **Liste complète** : 12, 13, 21, 23, 31, 32 ✓
— **Interprétation** : ce sont les **couples ordonnés** d'éléments distincts ✓
— **Formule générale** : $n(n-1)$ couples pour n valeurs ✓
— **Contrôle** : $3 \times 2 = 6$ ✓
— **Avec* range(1, 5)*** : $4 \times 3 = 12$ lignes ✓
— **Si l'on voulait des couples non ordonnés** : il faudrait *for j in range(i + 1, 4)* ✓
— **Nombre alors** : 3 lignes, soit $\dfrac{3 \times 2}{2}$ ✓
— **Application** : dénombrer les poignées de main dans un groupe ✓

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

— **Point de départ** : $x_0 = 3$ ✓
— **Étape 1** : $x_1 = \dfrac{1}{2}\left(3 + \dfrac{10}{3}\right)$ ✓
— **Calcul** : $\dfrac{1}{2} \times \dfrac{19}{3} = \dfrac{19}{6} \approx 3{,}166667$ ✓
— **Étape 2** : $x_2 = \dfrac{1}{2}\left(3{,}166667 + \dfrac{10}{3{,}166667}\right)$ ✓
— **Calcul intermédiaire** : $\dfrac{10}{3{,}166667} \approx 3{,}157895$ ✓
— **Résultat** : $x_2 \approx 3{,}162281$ ✓
— **Étape 3** : $\dfrac{10}{3{,}162281} \approx 3{,}162274$ ✓
— **Résultat** :

$$x_3 \approx 3{,}162277$$

— **Valeur exacte** : $\sqrt{10} \approx 3{,}1622777$ ✓
— **Précision atteinte** : 6 décimales exactes en **trois** étapes ✓
— **Erreur à l'étape 1** : $\approx 0{,}0044$ ✓
— **Erreur à l'étape 2** : $\approx 3{,}3 \times 10^{-6}$ ✓
— **Observation** : le nombre de décimales exactes **double** à chaque étape ✓
— **Nom de ce comportement** : convergence **quadratique** ✓

```
a = 10
```

```
x = 3
for i in range(3):
    x = (x + a / x) / 2
print(x)
```

— **Contrôle** : le programme affiche bien $\approx 3{,}1622777$ ✓
— **Comparaison avec la dichotomie** : elle gagne un bit par étape, Héron double les décimales ✓
— **Origine** : cette méthode est attribuée à Héron d'Alexandrie, au I^{er} siècle ✓

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

— **Principe** : tester tous les entiers de 1 à n ✓

```
def nbDiviseurs(n):
```

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

— **Diviseurs de** 36 : 1, 2, 3, 4, 6, 9, 12, 18, 36 ✓
— **Comptage** :

9 diviseurs

— **Contrôle par la décomposition** : $36 = 2^2 \times 3^2$ ✓
— **Formule** : $(2 + 1)(2 + 1) = 9$ ✓
— **Cohérence** : les deux méthodes concordent ✓
— **Remarque** : 9 est **impair**, ce qui caractérise les carrés parfaits ✓
— **Explication** : le diviseur $6 = \sqrt{36}$ est compté une seule fois ✓
— **Diviseurs de** 37 : 1 et 37 ✓
— **Comptage** :

2 diviseurs

— **Conclusion** : 37 est **premier** ✓
— **Caractérisation** : un nombre est premier si et seulement s'il a exactement 2 diviseurs ✓
— **Optimisation** : il suffit de tester jusqu'à $\sqrt{n}$ et de compter par paires ✓
— **Code optimisé** : pour chaque diviseur $d \leq \sqrt{n}$, on compte d et $\frac{n}{d}$ ✓
— **Attention au carré** : si $d = \frac{n}{d}$, on ne compte qu'une fois ✓
— **Somme des diviseurs de** 36 : $1 + 2 + 3 + 4 + 6 + 9 + 12 + 18 + 36 = 91$ ✓

— **Nombre le plus « divisible » sous** 100 : 60, 72, 84 et 90 ont 12 diviseurs ✓

Réponse. 36 a 9 diviseurs, 37 en a 2 (il est donc premier).

Correction de l’exercice 15 – Éviter une boucle infinie

— **Programme A — diagnostic** : aucune **mise à jour** de n ✓
— **Conséquence** : la condition $n < 100$ reste vraie pour toujours ✓
— **Comportement** : le programme affiche 1 indéfiniment ✓
— **Correction** : ajouter $n = n + 1$ dans la boucle ✓
— **Résultat après correction** : les entiers de 1 à 99 ✓
— **Nombre de lignes** : 99 ✓
— **Programme B — diagnostic** : la mise à jour **augmente** i au lieu de la diminuer ✓
— **Conséquence** : la condition $i > 0$ reste vraie pour toujours ✓
— **Comportement** : s croît sans fin, jusqu’à saturation de la mémoire ✓
— **Correction** : écrire $i = i - 1$ ✓
— **Résultat après correction** : $s = 10 + 9 + \cdots + 1$ ✓
— **Calcul** :

$$s = \frac{10 \times 11}{2} = 55$$

— **Règle générale** : dans un *while*, la mise à jour doit **rapprocher** de la condition d’arrêt ✓
— **Vérification à faire systématiquement** : la quantité testée évolue-t-elle dans le bon sens ? ✓
— **Deuxième vérification** : la condition finit-elle par devenir fausse ? ✓
— **Sécurité en pratique** : on peut ajouter un compteur de garde et un *break* au-delà d’un million de tours ✓
— **Alternative pour B** : une boucle *for i in range(10, 0, -1)* est plus sûre ✓
— *Avantage du* for* **: le nombre de tours est** fixé d’avance**, aucune boucle infinie possible ✓

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) **Mécanisme annuel** : intérêts sur le capital, puis versement ✓

```
def capital(n):
    c = 1500
    for i in range(n):
        c = c * 1.025 + 600
    return c
```

— **Ordre des opérations** : les intérêts portent sur le capital **avant** le versement ✓
— **Justification** : le versement a lieu en fin d’année ✓

b) **Après** 1 **an** : $1500 \times 1{,}025 + 600 = 2137{,}50$ € ✓

— **Après** 2 **ans** : $2137{,}50 \times 1{,}025 + 600 = 2790{,}94$ € ✓
— **Après** 3 **ans** : ≈ 3460,71 € ✓
— **Après** 4 **ans** : ≈ 4147,23 € ✓
— **Après** 5 **ans** :

$$\approx 4850{,}91 \text{ €}$$

— **Contrôle** : $4147{,}23 \times 1{,}025 + 600 \approx 4850{,}91$ ✓
— **Total versé en** 5 **ans** : $1500 + 5 \times 600 = 4500$ € ✓
— **Intérêts cumulés** : $4850{,}91 - 4500 \approx 350{,}91$ € ✓

c) **Programme de recherche du seuil** ✓

```
c = 1500
```

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

— **Après** 10 **ans** : ≈ 8642,16 € ✓
— **Après** 11 **ans** : ≈ 9458,21 € ✓
— **Après** 12 **ans** : ≈ 10 294,67 € ✓
— **Résultat** :

$$n = 12 \text{ ans, pour } \approx 10\,295 \text{ €}$$

— **Contrôle de l'année précédente** : $9458{,}21 \leq 10\,000$ ✓
— **Total versé** : $1500 + 12 \times 600 = 8700$ € ✓
— **Intérêts cumulés** : ≈ 1594,67 € ✓
— **Part des intérêts** : $\dfrac{1594{,}67}{10\,294{,}67} \approx 15{,}5\,\%$ ✓

d) **Sans versement annuel** : le capital suit $1500 \times 1{,}025^{n}$ ✓

— **Condition** : $1500 \times 1{,}025^{n} > 10\,000$ ✓
— **Rapport à atteindre** : $\dfrac{10\,000}{1500} \approx 6{,}667$ ✓
— **Essai à** $n = 76$: $1{,}025^{76} \approx 6{,}5315$, soit ≈ 9797,3 € ✓
— **Essai à** $n = 77$: $1{,}025^{77} \approx 6{,}6948$, soit ≈ 10 042,2 € ✓
— **Résultat** :

$$n = 77 \text{ ans}$$

— **Comparaison frappante** : 12 ans avec versements contre 77 ans sans ✓
— **Rapport** : plus de **six fois** plus long ✓
— **Conclusion** : les **versements réguliers** pèsent bien plus que le taux d'intérêt ✓
— **Contrôle de l'ordre de grandeur** : les versements apportent 7200 € sur 12 ans, les intérêts seulement 1595 € ✓
— **Part relative** : les versements représentent 82 % de la croissance ✓
— **Leçon d'épargne** : mieux vaut verser régulièrement que chercher le meilleur taux ✓

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.