

Algorithmique et programmation Python

Corrigé des exercices - 1ère

Corrigé détaillé des exercices du chapitre. Vérifie tes réponses étape par étape et corrige-toi.

1 Que fait ce programme ? ★★★

Le programme affiche :

```
2 2 2.4 139
```

- $12 // 5$ est le quotient entier : $12 = 5 \times 2 + 2$, donc 2 ;
- $12 \% 5$ est le reste : 2 ;
- $12 / 5 = 2,4$ (division décimale, affichée 2.4) ;
- $12^2 - 5 = 144 - 5 = 139$.

2 Vrai ou faux ? ★★★

- Faux : `range(4)` donne 0, 1, 2, 3 ; la borne 4 est exclue.
- Faux : `=` est une affectation, le test d'égalité s'écrit `==` (et la ligne écrite provoque une erreur).
- Faux : `return` termine immédiatement la fonction.
- Faux : les indices vont de 0 à 2 ; `L[3]` provoque une erreur, c'est `L[2]` qui vaut 8.
- Faux : $0.1 + 0.2$ vaut 0.30000000000000004 à cause de l'approximation des décimaux.

3 Écrire une fonction ★★★

Programme :

```
def f(x):  
    return 3 * x ** 2 - 5 * x + 2  
  
print(f(4), f(-1))
```

$f(4) = 3 \times 16 - 20 + 2 = 30$ et $f(-1) = 3 \times 1 + 5 + 2 = 10$. Le programme affiche donc :

30 10

4 Parité ★★★

Un entier est pair si son reste dans la division par 2 est nul :

```
def est_pair(n):  
    return n % 2 == 0  
  
print(est_pair(14), est_pair(7))
```

Pour 14 : $14 \% 2 = 0$ donc True. Pour 7 : $7 \% 2 = 1$ donc False. Sortie :

True False

5 Tableau de suivi ★★★

k	1	2	3	4	5
s après le tour	1	5	14	30	55

Le programme calcule $1^2 + 2^2 + 3^2 + 4^2 + 5^2$ et affiche 55.

6 Manipuler une liste ★★★

```
12 5 4  
[8, 0, 12, 5, 20]  
5
```

Premier print : L[2] est le troisième élément (12), L[-1] le dernier (5), la longueur est 4. Puis append ajoute 20 à la fin et L[1] = 0 remplace le 3 par 0 : la liste est [8, 0, 12, 5, 20] et sa longueur est 5.

7 Tarif de cinéma ★★★

```
def tarif(age):  
    if age < 12:  
        return 5  
    elif age <= 25:
```

```

        return 7
    else:
        return 9

print(tarif(10), tarif(12), tarif(30))

```

10 ans : moins de 12, donc 5 €. 12 ans : le premier test est faux ($12 < 12$ est faux) mais $12 \leq 25$ est vrai, donc 7 €. 30 ans : aucun des deux tests, donc 9 €. Sortie :

5 7 9

8 Suivi d'une boucle while ★★★

Tour	0	1	2	3	4
x	100	50	25	12,5	6,25
n	0	1	2	3	4

Après le tour 4, $x = 6,25$ n'est plus strictement supérieur à 10 : la boucle s'arrête.
Affichage :

6.25 4

n est le nombre de fois qu'il faut diviser 100 par 2 pour obtenir un résultat inférieur ou égal à 10.

9 Moyenne d'une liste ★★★

```

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

print(moyenne([12, 15, 9, 18]))

```

Somme : $12 + 15 + 9 + 18 = 54$, longueur 4, moyenne $54 / 4 = 13,5$. Sortie :

13.5

10 Indice du maximum ★★★

```
def indice_max(L):
    imax = 0
    for i in range(len(L)):
        if L[i] > L[imax]:
            imax = i
    return imax

print(indice_max([4, 17, 9, 17, 2]))
```

On compare avec l'inégalité *stricte* $>$: le second 17 ne remplace pas le premier. Le maximum est 17, d'indice 1 :

1

11 Compter les notes ★★★

```
def compte_reussites(notes):
    k = 0
    for x in notes:
        if x >= 10:
            k = k + 1
    return k

print(compte_reussites([8, 12, 10, 15, 6, 9, 11]))
```

Les notes ≥ 10 sont 12, 10, 15 et 11 : il y en a 4. Sortie :

4

12 Une suite récurrente ★★★

a. $u_1 = 2 \times 1 + 3 = 5$; $u_2 = 2 \times 5 + 3 = 13$; $u_3 = 2 \times 13 + 3 = 29$.

```
b. def terme(n):
    u = 1
    for k in range(n):
        u = 2 * u + 3
```

```
return u
```

c. Le programme complet

```
def terme(n):  
    u = 1  
    for k in range(n):  
        u = 2 * u + 3  
    return u  
  
print([terme(n) for n in range(6)])  
print(4 * 2 ** 5 - 3)
```

```
[1, 5, 13, 29, 61, 125]  
125
```

On lit $u_5 = 125$ et, avec la formule, $4 \times 2^5 - 3 = 128 - 3 = 125$. La formule est cohérente pour $n = 5$.

13 Placement à 2 % ★★★

```
c = 2500  
n = 0  
while c <= 3000:  
    c = c * 1.02  
    n = n + 1  
print(n)
```

Il affiche 10. Vérification : $2500 \times 1,02^9 \approx 2987,7$ € (encore sous 3000 €) et $2500 \times 1,02^{10} \approx 3047,5$ € (au-dessus de 3000 €). Réponse : il faut 10 ans.

14 Un village qui se dépeuple ★★★

Baisser de 5 % revient à multiplier par 0,95.

```
p = 12000  
n = 0  
while p >= 8000:  
    p = p * 0.95  
    n = n + 1
```

```
print(n)
```

Le programme affiche 8. Vérification : $12000 \times 0,95^7 \approx 8380$ (encore au-dessus de 8000) et $12000 \times 0,95^8 \approx 7961$ (en dessous). Réponse : 8 ans.

15 Fréquence du 6 ★★★

```
from random import randint

def frequence_six(N):
    c = 0
    for i in range(N):
        if randint(1, 6) == 6:
            c = c + 1
    return c / N

print(frequence_six(1000))
```

La probabilité théorique est $1/6 \approx 0,167$. À chaque exécution on obtient une valeur différente, généralement entre 0,14 et 0,19 pour $N = 1000$. Si on remplace 1000 par 100000, la valeur affichée est plus proche de 0,167 : c'est la loi des grands nombres. Il est normal que la fréquence ne soit jamais exactement $1/6$.

16 Dichotomie à la main ★★★

- a. $f(2) = 4 - 5 = -1 < 0$ et $f(3) = 9 - 5 = 4 > 0$, et f est continue : une solution existe dans $[2 ; 3]$.

Étape	Milieu m	f(m)	Nouvel intervalle
1	2,5	1,2500	$[2 ; 2,5]$
2	2,25	0,0625	$[2 ; 2,25]$
3	2,125	-0,4844	$[2,125 ; 2,25]$

- c. L'amplitude vaut $2,25 - 2,125 = 0,125 = 1/8$, soit $1/2^3$: elle a été divisée par 2 à chaque étape. Donc $2,125 \leq \sqrt{5} \leq 2,25$ (en réalité $\sqrt{5} \approx 2,236$).

17 Compléter un programme ★★★

- a. $h(0) = -1 < 0$ et $h(1) = 1 > 0$; h est continue (c'est un polynôme), donc h s'annule dans $[0 ; 1]$. De plus h est strictement croissante (somme de deux fonctions croissantes), la solution est unique.
- b. Il faut **b - a > e** après while, puis **b = m** et **a = m**. Programme complet :

```
def dichotomie(f, a, b, e):
    n = 0
    while b - a > e:
        m = (a + b) / 2
        if f(a) * f(m) <= 0:
            b = m
        else:
            a = m
        n = n + 1
    return a, b, n

def h(x):
    return x ** 3 + x - 1

print(dichotomie(h, 0, 1, 0.001))
```

c. Sortie :

```
(0.681640625, 0.6826171875, 10)
```

La solution est située entre les deux bornes affichées, et il a fallu 10 étapes : l'amplitude finale vaut $1/2^{10} = 1/1024 \approx 0,00098$, inférieure à 0,001, alors que $1/2^9 = 1/512 \approx 0,00195$ ne l'était pas.

18 Combien d'étapes ? ★★★

- $1/2^n$.
- Il faut $1/2^n \leq 10^{-6}$, c'est-à-dire $2^n \geq 1\,000\,000$. Comme $2^{19} = 524\,288 < 1\,000\,000$ et $2^{20} = 1\,048\,576 \geq 1\,000\,000$, il faut **20 étapes**.
- L'amplitude est $8/2^n \leq 10^{-6}$, c'est-à-dire $2^n \geq 8\,000\,000$. Comme $2^{22} = 4\,194\,304 < 8\,000\,000$ et $2^{23} = 8\,388\,608 \geq 8\,000\,000$, il faut **23 étapes** (trois de plus, car $8 = 2^3$).

19 Tri par sélection à la main ★★★

- Tour 0 : le plus petit élément est 1 (indice 3), on l'échange avec 7 : [1, 3, 9, 7, 5].
- Tour 1 : le plus petit parmi 3, 9, 7, 5 est 3, déjà en place : [1, 3, 9, 7, 5].
- Tour 2 : le plus petit parmi 9, 7, 5 est 5, échangé avec 9 : [1, 3, 5, 7, 9].
- Tour 3 : le plus petit parmi 7, 9 est 7, déjà en place : [1, 3, 5, 7, 9].

La liste est triée après 4 tours ($n - 1$ tours pour $n = 5$ éléments).

20 Coût du tri par sélection ★★★

a. Affichage :

```
10
45
4950
```

b. Au tour i , la boucle intérieure compare $L[i_{\min}]$ avec les éléments d'indices $i + 1$, ..., $n - 1$, soit $n - 1 - i$ comparaisons. Le total est $(n - 1) + (n - 2) + \dots + 1 + 0 = n(n - 1)/2$. Ce nombre ne dépend pas des valeurs : pour $n = 5$, on trouve 10 ; pour $n = 10$, 45 ; pour $n = 100$, 4950, ce qui correspond à l'affichage.

21 Suite de Syracuse ★★★

a. $7 \rightarrow 22 \rightarrow 11 \rightarrow 34 \rightarrow 17 \rightarrow 52 \rightarrow 26 \rightarrow 13 \rightarrow 40 \rightarrow 20 \rightarrow 10 \rightarrow 5 \rightarrow 16 \rightarrow 8 \rightarrow 4 \rightarrow 2 \rightarrow 1$: cela fait 16 étapes.

```
b. def vol(n):
    etapes = 0
    while n != 1:
        if n % 2 == 0:
            n = n // 2
        else:
            n = 3 * n + 1
        etapes = etapes + 1
    return etapes

print(vol(7))
print(max(vol(n) for n in range(1, 31)))
```

Sortie :

```
16
111
```

Le premier affichage confirme 16 pour $n = 7$. Le second donne le plus long temps de vol parmi les entiers de 1 à 30. On utilise un **while** car on ne sait pas à l'avance combien d'étapes il faudra.

22 Estimer π par simulation ★★★

- a. L'aire du carré est 1, celle du quart de disque est $\pi \times 1^2 / 4 = \pi / 4 \approx 0,785$. C'est donc aussi la probabilité cherchée.

```
b. from random import random

def estime_pi(N):
    dedans = 0
    for i in range(N):
        x = random()
        y = random()
        if x * x + y * y <= 1:
            dedans = dedans + 1
    return 4 * dedans / N

print(estime_pi(100000))
```

La fréquence des points dans le quart de disque estime $\pi/4$, donc on multiplie par 4.

- c. Les points sont tirés au hasard : chaque exécution donne une fréquence légèrement différente. Avec $N = 100000$ on obtient en général 3,14 à 0,01 près ; avec N plus grand, l'estimation est en moyenne meilleure, mais elle reste une estimation.

23 Deuxième plus grand ★★★

```
def deuxieme_max(L):
    m1 = max(L[0], L[1])
    m2 = min(L[0], L[1])
    for x in L[2:]:
        if x > m1:
            m2 = m1
            m1 = x
        elif x > m2:
            m2 = x
    return m2

print(deuxieme_max([4, 17, 9, 12, 2]))
print(deuxieme_max([17, 4, 9, 12, 2]))
```

On garde deux variables : m1 (le plus grand vu) et m2 (le deuxième). Quand un nouvel élément dépasse m1, l'ancien m1 devient m2 ; sinon, s'il dépasse seulement m2, il remplace m2. Suivi pour [4, 17, 9, 12, 2] : (m1, m2) = (17, 4), puis 9 → (17, 9), puis 12 → (17, 12), puis 2 ne change rien. Sortie :

12

12

On trouve 12 dans les deux cas : le résultat ne dépend pas de la position du 17.