Corrigé – Listes en Python

Mathématiques – Première spécialité

Logique et raisonnement · exercices ★ à ★★★, pièges et QCM

Pour chaque erreur : relis la ligne du corrigé, trouve l'étape où tu t'es trompé, puis refais le calcul sans regarder.

Exercice 1 – Indices et longueur ★

On a écrit `L = [12, 5, 8, 21, 3, 17]`. Que vaut chaque expression ?

a) L'indice 0 désigne le premier élément.
`L[0]` vaut `12`.

b) Indice 3 : quatrième élément (on compte 0, 1, 2, 3).
`L[3]` vaut `21`.

c) `len` compte les éléments : il y en a 6.
`len(L)` vaut `6`.

d) `len(L) - 1` vaut $6 - 1 = 5$: c'est l'indice du dernier élément.
`L[len(L) - 1]` vaut `17`.

e) *Les indices vont de* 0 *à* 5 *: l'indice* 6 *n'existe pas.*
Python s'arrête sur une erreur (`IndexError`).

f) `L[1]` vaut 5 (deuxième élément) et `L[4]` vaut 3 (cinquième) : $5 + 3 = 8$.
`L[1] + L[4]` vaut `8`.

g) `L[2]` vaut 8 et `L[5]` vaut 17 : $8 \times 17 = 136$.
`L[2] * L[5]` vaut `136`.

h) `in` teste l'appartenance, comme $9 \in L$: 9 n'est pas dans la liste.
`9 in L` vaut `False`.

Exercice 2 – Que contient la liste ? ★

Donne le contenu de la liste à la fin de chaque programme.

a) `k` prend les valeurs 0 à 4 ; à chaque tour, on ajoute $2k$ à la fin.
À la fin, `T` vaut `[0, 2, 4, 6, 8]`.

b) Au tour `k`, on ajoute le triple de l'élément d'indice `k` : 3×1, puis 3×3, 3×9, 3×27.
À la fin, `T` vaut `[1, 3, 9, 27, 81]`.

c) `append` ajoute 9 à la fin, puis `L[0] = 5` remplace le premier élément.
À la fin, `L` vaut `[5, 7, 1, 9]`.

d) `remove(2)` supprime la **première** occurrence de 2 seulement.
À la fin, `L` vaut `[6, 8, 2]`.

e) `del L[1]` supprime l'élément d'**indice** 1, c'est-à-dire le deuxième (1).
À la fin, `L` vaut `[3, 4, 1, 5]`.

f) `x` parcourt directement les éléments ; on ajoute leurs carrés, dans l'ordre.
À la fin, `T` vaut `[4, 1, 25, 0]`.

g) Deux ajouts par tour, pour $k = 1, 2, 3$: k puis $-k$.
À la fin, `T` vaut `[1, -1, 2, -2, 3, -3]`.

h) `i` parcourt les indices 0, 1, 2 ; chaque élément est remplacé par lui-même plus 10.
À la fin, `L` vaut `[15, 18, 11]`.

Exercice 3 – Listes en compréhension ★

Donne la liste obtenue.

a) `k` prend les valeurs 0, 1, 2, 3, 4, 5.
Pour chacune, on calcule `k * k`.
La liste vaut `[0, 1, 4, 9, 16, 25]`.

b) `n` prend les valeurs 1, 2, 3, 4, 5.
Pour chacune, on calcule `3 * n - 2`.
Ce sont les termes u_1 à u_5 de la suite $u_n = 3n - 2$.
La liste vaut `[1, 4, 7, 10, 13]`.

c) `d` prend les valeurs entières de 1 à 20 (la borne 21 est exclue).
On garde celles qui vérifient `20 % d == 0` : 1, 2, 4, 5, 10, 20.
Ce sont les diviseurs de 20 ; la borne 21 est nécessaire pour inclure 20.
La liste vaut `[1, 2, 4, 5, 10, 20]`.

d) `x` prend les valeurs $-3, -2, -1, 0, 1, 2, 3$.
On garde celles qui vérifient `x * x > 4` : $-3, 3$.
La liste vaut `[-3, 3]`.

e) `n` prend les valeurs 0, 1, 2, 3, 4, 5, 6.
Pour chacune, on calcule `2 ** n`.
La liste vaut `[1, 2, 4, 8, 16, 32, 64]`.

f) `k` prend les valeurs entières de 0 à 49 (la borne 50 est exclue).
On garde celles qui vérifient `k % 7 == 0` : 0, 7, 14, 21, 28, 35, 42, 49.
Ce sont les multiples de 7 inférieurs à 50 (0 compris).
La liste vaut `[0, 7, 14, 21, 28, 35, 42, 49]`.

g) `x` prend les valeurs $-1, 0, 2, 5$.
Pour chacune, on calcule `x * x - 3 * x`.
Ce sont les images de $-1, 0, 2, 5$ par $f(x) = x^2 - 3x$.
La liste vaut `[4, 0, -2, 10]`.

h) `n` prend les valeurs entières de 1 à 29 (la borne 30 est exclue).
On garde celles qui vérifient `n % 2 == 1 and n % 3 == 0` : 3, 9, 15, 21, 27.
Les nombres impairs **et** multiples de 3, de 1 à 29.
La liste vaut `[3, 9, 15, 21, 27]`.

Exercice 4 – Écrire une fonction qui parcourt une liste ★★

Écris la fonction décrite, puis donne le résultat du test.

a) Un accumulateur `s` à 0 avant la boucle ; on lui ajoute chaque élément ; on renvoie après la boucle.

```
def somme(L):
    s = 0
    for x in L:
        s = s + x
    return s
```

Test : `somme([4, -2, 7, 1])` renvoie `10`.

b) Même somme, puis on divise par le nombre d'éléments `len(L)`.

Test : $\dfrac{12+15+9}{3} = \dfrac{36}{3} = 12$.

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

Test : `moyenne([12, 15, 9])` renvoie `12.0`.

c) On part du premier élément (pas de 0 : la liste peut ne contenir que des négatifs) ; chaque élément plus grand le remplace.

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

Test : `maxi([-5, -2, -9])` renvoie `-2`.

d) Un compteur à 0, augmenté de 1 seulement quand `x >= s` (le 10 compte : « supérieur **ou égal** »).

```
def compte(L, s):
    c = 0
    for x in L:
        if x >= s:
            c = c + 1
    return c
```

Test : `compte([8, 13, 10, 4, 17], 10)` renvoie `3`.

e) La condition après `if` joue le rôle de « tel que » : $\{x \in L \mid x > 0\}$; 0 n'est pas strictement positif.

```
def positifs(L):
    return [x for x in L if x > 0]
```

Test : `positifs([3, -1, 0, 6, -4])` renvoie `[3, 6]`.

Exercice 5 – Termes d'une suite ★★

Réponds à chaque question.

a)

```
U = [5 * n - 3 for n in range(6)]
```

Pour aller jusqu'à u_5, il faut `range(6)` (n de 0 à 5).
`U` vaut `[-3, 2, 7, 12, 17, 22]`.

b)

```
U = [2]
for n in range(4):
    U.append(3 * U[n] - 1)
```

Au tour `n`, on calcule u_{n+1} à partir de `U[n]`, qui vaut u_n.
4 tours ajoutent u_1 à u_4 : `U` vaut `[2, 5, 14, 41, 122]`.

c)

```
V = [1000]
for n in range(4):
    V.append(V[n] * 0.5)
```

$v_{n+1} = 0{,}5\,v_n$: on multiplie le dernier terme calculé par 0,5.
`V` vaut `[1000, 500.0, 250.0, 125.0, 62.5]` (le produit par `0.5` donne des flottants).

d)

```
U = [5 * n - 3 for n in range(6)]
s = 0
for u in U:
    s = s + u
```

`u` parcourt les termes de la liste.
À la fin, `s` vaut `57` : $-3 + 2 + 7 + 12 + 17 + 22 = 57$.

e) `W` contient w_0 à w_{19} ; la liste en compréhension garde les termes strictement négatifs et `len` les compte.
$w_n = n(n-10) < 0$ pour n de 1 à 9 ($w_0 = w_{10} = 0$ ne comptent pas).
`c` vaut `9` : parmi w_0 à w_{19}, 9 termes sont strictement négatifs.

Exercice 6 – Programmes à trous ★★

Complète les `...` pour que le programme fasse ce qui est annoncé : a) liste des multiples de 3 de 3 à 30 ; b) somme $1^2 + 2^2 + \cdots + n^2$ pour $n = 6$; c) produit des éléments de `L` ; d) liste des entiers de 1 à 40 qui se terminent par 7.

a)

```
T = []
for k in range(1, 11):
    T.append(3 * k)
```

But : les multiples de 3 de 3 à 30.
On les obtient en calculant $3k$ pour k de 1 à 10, donc `range(1, 11)`.
À la fin, `T` vaut `[3, 6, 9, 12, 15, 18, 21, 24, 27, 30]`.

b)

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

But : la somme $1^2 + 2^2 + \cdots + n^2$.

La somme part de 0, avant la boucle ; `range(1, n + 1)` va jusqu'à n compris ; à chaque tour, on ajoute k^2.

Contrôle : $1 + 4 + 9 + 16 + 25 + 36 = 91$.

À la fin, `s` vaut `91`.

c)

```
L = [3, -2, 5, 1, 4]
p = 1
for x in L:
    p = p * x
```

But : le produit des éléments.

Un produit part de 1, pas de 0 (sinon il resterait nul) ; à chaque tour, on multiplie par l'élément.

Contrôle : $3 \times (-2) \times 5 \times 1 \times 4 = -120$.

À la fin, `p` vaut `-120`.

d)

```
D = [d for d in range(1, 41) if d % 10 == 7]
```

But : les entiers de 1 à 40 qui se terminent par 7.

Leur reste dans la division par 10 est 7, d'où le test `d % 10 == 7` ; la borne 41 inclut 40.

À la fin, `D` vaut `[7, 17, 27, 37]`.

Exercice 7 – Les notes ★★★

1. Il y a 8 notes : `len(notes)` vaut `8`.
   `notes[2]` est la troisième note : `15`.
   `notes[7]` est la huitième et dernière : `14`.

2. ```
   notes = [12, 8.5, 15, 9, 17, 11, 6.5, 14]
   s = 0
   for x in notes:
       s = s + x
   m = s / len(notes)
   ```

   Somme : $12 + 8{,}5 + 15 + 9 + 17 + 11 + 6{,}5 + 14 = 93$, puis $93 \div 8 = 11{,}625$.

   `m` vaut `11.625`.

3. ```
   notes = [12, 8.5, 15, 9, 17, 11, 6.5, 14]
   bonnes = [x for x in notes if x >= 10]
   nb_bonnes = len(bonnes)
   ```

   `bonnes` vaut `[12, 15, 17, 11, 14]` et `nb_bonnes` vaut `5`.

4. On parcourt les indices ; `i_max` retient l'indice de la plus grande note vue jusqu'ici.

```
notes = [12, 8.5, 15, 9, 17, 11, 6.5, 14]
i_max = 0
for i in range(len(notes)):
    if notes[i] > notes[i_max]:
        i_max = i
```

`i_max` vaut `4` : la meilleure note, 17, est la cinquième de la liste.

Exercice 8 – Tester une conjecture ★★★

1. `d` parcourt les entiers de 2 à $k-1$.
   Si l'un d'eux divise k (`k % d == 0`), k n'est pas premier : on renvoie `False` tout de suite.
   Si la boucle se termine sans trouver de diviseur, on renvoie `True`.

2. ```
   C = [n for n in range(50) if not premier(n * n + n + 41)]
   ```

   `not` inverse le test.
   `C` vaut `[40, 41, 44, 49]`.

3. Non : la liste `C` n'est pas vide, elle contient des contre-exemples.
   Pour $n = 40$: $40^2 + 40 + 41 = 1600 + 81 = 1681 = 41 \times 41$, qui n'est pas premier.
   Pourtant, la propriété est vraie pour tous les n de 0 à 39 : des exemples, même nombreux, ne démontrent pas un « pour tout ».

4. Non : le programme n'a testé que n de 0 à 49.
   Sur un ensemble infini comme $\mathbb{N}$, un programme peut trouver un contre-exemple, mais jamais démontrer un « pour tout ».

Exercice 9 – Trouve l'erreur ★★

Chaque programme ne fait pas ce qui est annoncé. Trouve l'erreur, puis corrige.

a) Le programme faux : il s'arrête sur une erreur (`IndexError`).
Ligne fautive (l. 2) : `x = L[3]`
Ligne corrigée : `x = L[2]`
La liste a 3 *éléments, d'indices* 0, 1, 2 *: l'indice* 3 *n'existe pas.*
Le dernier est `L[len(L) - 1]`, *ici* `L[2]`.

```
L = [4, 9, 2]
x = L[2]
```

Version corrigée : `x` = 2.

b) Le programme faux : il donne `S` = 17.
Ligne fautive (l. 3) : `for i in range(1, len(L)):`
Ligne corrigée : `for i in range(len(L)):`
`range(1, len(L))` *commence à l'indice* 1 *: le premier élément,* `L[0]`, *est oublié.*

```
L = [5, 3, 8, 6]
S = 0
for i in range(len(L)):
    S = S + L[i]
```

Version corrigée : `S` = 22.
```

c) Le programme faux : il s'arrête sur une erreur (`AttributeError`).

Ligne fautive (l. 3) : `T = T.append(k * k)`

Ligne corrigée : `T.append(k * k)`

`append` modifie la liste et ne renvoie rien (`None`) : après le premier tour, `T` vaut `None` et le deuxième `append` plante.

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

Version corrigée : `T` = `[0, 1, 4, 9, 16]`.

d) Le programme faux : il donne `c` = `1`.

Une ligne est mal placée.

`c = 0` est dans la boucle : le compteur repart de 0 *à chaque tour, il ne reste que le dernier.*

```
L = [14, 8, 21, 11]
c = 0
for x in L:
    if x > 10:
        c = c + 1
```

Version corrigée : `c` = `3`.

e) Le programme faux : il donne `m` = `0`.

Ligne fautive (l. 2) : `m = 0`

Ligne corrigée : `m = L[0]`

Parti de 0*, le maximum n'est jamais dépassé par des négatifs.*

On part du premier élément `L[0]`.

```
L = [-4, -1, -7]
m = L[0]
for x in L:
    if x > m:
        m = x
```

Version corrigée : `m` = `-1`.

f) Le programme faux : il s'arrête sur une erreur (`SyntaxError`).

Ligne fautive (l. 1) : `P = [k for k in range(1, 10) if k % 2 = 0]`

Ligne corrigée : `P = [k for k in range(1, 10) if k % 2 == 0]`

`=` *affecte une valeur ; pour comparer, il faut* **`==`** *(sinon erreur de syntaxe).*

```
P = [k for k in range(1, 10) if k % 2 == 0]
```

Version corrigée : `P` = `[2, 4, 6, 8]`.

QCM – Six questions comme en contrôle

Une seule bonne réponse par question.

1. Avec `L = [7, 3, 9, 4, 6]`, `L[2]` vaut :

Réponse C : `9`

Indices 0, 1, 2 : `L[2]` est le troisième élément.

2. `len([2, 5, 5, 8])` vaut :

Réponse A : `4`

`len` compte tous les éléments, même répétés.

3. `[k * 2 for k in range(1, 4)]` vaut :

Réponse D : `[2, 4, 6]`

k prend les valeurs 1, 2, 3 (on s'arrête avant 4).

4. Avec `L = [10, 20, 30]`, `L[len(L)]` donne :

Réponse B : une erreur

`len(L)` vaut 3, et les indices vont de 0 à 2.

5. Après `T = [1, 2]`, `T.append(5)` et `T.append(1)`, `T` vaut :

Réponse D : `[1, 2, 5, 1]`

`append` ajoute à la fin, dans l'ordre des appels.

6. Pour parcourir tous les indices d'une liste `L`, on écrit :

Réponse A : `for i in range(len(L)):`

Les indices vont de 0 à `len(L) - 1`, exactement ce que donne `range(len(L))`.

Ce document est la propriété intellectuelle de Ben, professeur particulier (benprofparticulier.fr), protégée par le Code de la propriété intellectuelle (art. L. 111-1 et L. 122-4). Il est mis à disposition pour un usage personnel de révision. Toute reproduction, diffusion, mise en ligne ou revente, totale ou partielle, sans autorisation écrite est interdite et constitue une contrefaçon (art. L. 335-2).