

Devoir d'informatique — Python et listes

Lundi 28 septembre 2026

Durée : une heure. Répondez exclusivement sur le document réponse fourni, en indiquant votre nom et votre prénom. Ne cornez pas les feuilles et ne les agrafez pas : elles seront numérisées. N'utilisez ni stylo effaçable de type FriXion, ni correcteur blanc, ni crayon à papier. Les calculatrices sont interdites.

Les questions 1 à 18 portent chacune sur une fonction Python syntaxiquement correcte. Une fonction peut néanmoins ne pas satisfaire l'objectif annoncé. Lorsque plusieurs appels se succèdent, ils sont exécutés dans l'ordre indiqué. On suppose que les listes d'entiers mentionnées dans les objectifs ne contiennent que des entiers.

Question 1.

Que renvoie `mystere1([4, 1, 7])` ?

```
1 def mystere1(L):
2     s = 0
3     for x in L:
4         s = s + x
5     return s
```

Démonstration. La fonction additionne les éléments : elle renvoie $4 + 1 + 7 = 12$. □

Question 2.

Que renvoie `mystere2([5, 2, 5, 1])` ?

```
1 def mystere2(L):
2     c = 0
3     for x in L:
4         if x == L[0]:
5             c = c + 1
6     return c
```

Démonstration. Elle compte les éléments égaux au premier élément, ici 5. Elle renvoie donc 2. □

Question 3.

On souhaite savoir si `x` figure dans `L`. Que renvoie `mystere3([2, 4, 6], 4)` ? La fonction répond-elle à l'objectif pour toute liste ? Justifiez brièvement.

```
1 def mystere3(L, x):
2     for y in L:
3         if y == x:
4             return True
5         else:
6             return False
7     return False
```

Démonstration. Elle renvoie **False**. Elle s'arrête dès le premier élément, 2, différent de 4. Elle ne convient donc pas en général. Il faut retirer le **else** et ne renvoyer **False** qu'après la boucle. ☐

Question 4.

Que renvoie `mystere4([3, 1, 4])` ? La liste passée en argument est-elle modifiée ?

```
1 def mystere4(L):
2     R = []
3     for x in L:
4         R.append(x)
5     R[0] = 9
6     return R[0] + L[0]
```

Démonstration. La nouvelle liste R vaut [9, 1, 4] ; L reste [3, 1, 4]. La fonction renvoie donc $9 + 3 = 12$. La liste passée en argument n'est pas modifiée. ☐

Question 5.

Que valent a et b après ces instructions ?

```
1 def mystere5(L):
2     L.append(8)
3     return len(L)
4
5 a = [1, 2]
6 b = mystere5(a)
```

Démonstration. On obtient `a = [1, 2, 8]` et `b = 3`. La méthode `append` modifie la liste reçue. ☐

Question 6.

Même question pour a et b.

```
1 def mystere6(L):
2     R = []
3     for x in L:
4         R.append(x)
5     L = R
6     L.append(8)
7     return len(L)
8
9 a = [1, 2]
10 b = mystere6(a)
```

Démonstration. On obtient `a = [1, 2]` et `b = 3`. La variable locale L désigne la nouvelle liste R avant l'ajout de 8. ☐

Question 7.

Que renvoie `mystere7([4, 7, 2, 9])` ?

```
1 def mystere7(L):
2     R = []
3     for i in range(1, len(L)):
4         R.append(L[i] - L[i - 1])
5     return R
```

Démonstration. Les différences successives sont $7-4=3$, $2-7=-5$ et $9-2=7$. La fonction renvoie `[3, -5, 7]`. $\square$

Question 8.

Que renvoie `mystere8([2, 2, 3, 3, 3])` ? Que compte cette fonction ?

```
1 def mystere8(L):
2     c = 0
3     for i in range(len(L) - 1):
4         if L[i] == L[i + 1]:
5             c = c + 1
6     return c
```

Démonstration. La fonction renvoie 3 : les paires d'indices (0,1), (2,3) et (3,4) contiennent deux éléments égaux. Elle compte les paires d'éléments voisins égaux, et non le nombre de valeurs présentes plusieurs fois. $\square$

Question 9.

Que renvoie `mystere13([2, 5, 1, 4])` ? La liste donnée en argument est-elle modifiée ?

```
1 def mystere13(L):
2     R = []
3     for i in range(len(L)):
4         R.append(L[len(L) - 1 - i])
5     return R
```

Démonstration. La fonction renvoie `[4, 1, 5, 2]`. Elle lit les éléments de `L` en partant de la fin et les ajoute à une nouvelle liste. La liste donnée en argument n'est pas modifiée. $\square$

Question 10.

L'objectif est de renvoyer une liste des éléments de `L` qui sont pairs. Que se passe-t-il lors de `mystere10([1, 2, 3, 4])` ? Expliquez l'erreur.

```
1 def mystere10(L):
2     R = []
3     for x in L:
4         if x % 2 == 0:
5             R = R.append(x)
6     return R
```

Démonstration. Au passage sur 2, `R.append(2)` ajoute 2 à la liste, mais renvoie `None` : `R` devient alors `None`. Au passage sur 4, l'appel `R.append(4)` provoque une erreur `AttributeError`. Il fallait écrire simplement `R.append(x)`. $\square$

Question 11.

Que renvoie `mystere11([4, 1, 3, 2])` ?

```

1 def mystere11(L):
2     c = 0
3     for i in range(len(L) - 1):
4         if L[i] > L[i + 1]:
5             c = c + 1
6     return c

```

Démonstration. Les descentes sont $4 > 1$ et $3 > 2$. La fonction renvoie 2. □

Question 12.

Que renvoie `mystere12([1, 3, 5], [2, 4])` ?

```

1 def mystere12(A, B):
2     R = []
3     for x in A:
4         R.append(x)
5     for x in B:
6         R.append(x)
7     return R

```

Démonstration. La fonction ajoute d'abord les éléments de A, puis ceux de B. Elle renvoie `[1, 3, 5, 2, 4]`. □

Question 13.

Que vaut `a` après ces instructions ? Indiquez les indices des éléments échangés.

```

1 def mystere15(L):
2     for i in range(len(L) - 1):
3         if L[i] > L[i + 1]:
4             tampon = L[i]
5             L[i] = L[i + 1]
6             L[i + 1] = tampon
7
8 a = [4, 2, 3, 1]
9 mystere15(a)

```

Démonstration. Les échanges portent successivement sur les indices (0, 1), (1, 2) et (2, 3). On obtient `a = [2, 3, 1, 4]`. □

Question 14.

L'objectif est de renvoyer le minimum d'une liste non vide. Pour quelle liste cet appel échoue-t-il : `mystere14([2, 3])` ou `mystere14([-3, -1])` ? Donnez la correction d'une seule ligne.

```

1 def mystere14(L):
2     m = 0
3     for x in L:
4         if x < m:
5             m = x
6     return m

```

Démonstration. L'appel `mystere14([2, 3])` renvoie 0 au lieu de 2; celui sur `[-3, -1]` renvoie bien `-3`. Pour une liste non vide, il faut remplacer `m = 0` par `m = L[0]`. □

Question 15.

Que vaut `a` après ces instructions ? Indiquez les indices des éléments échangés.

```
1 def mystere15(L):
2     for i in range(len(L) - 1):
3         if L[i] > L[i + 1]:
4             L[i], L[i + 1] = L[i + 1], L[i]
5
6 a = [4, 2, 3, 1]
7 mystere15(a)
```

Démonstration. Après les échanges des indices (0,1), (1,2), puis (2,3), on obtient successivement `[2, 4, 3, 1]`, `[2, 3, 4, 1]` et finalement `a = [2, 3, 1, 4]`. □

Question 16.

Pour une liste de longueur n , combien de fois la comparaison `L[i] > L[j]` est-elle exécutée ? Donnez une expression en fonction de n , puis l'ordre de grandeur du temps d'exécution.

```
1 def mystere16(L):
2     c = 0
3     for i in range(len(L)):
4         for j in range(i):
5             if L[i] > L[j]:
6                 c = c + 1
7     return c
```

Démonstration. Pour chaque i de 0 à $n - 1$, la comparaison est exécutée i fois. Le nombre total vaut $0 + 1 + \dots + (n - 1) = \frac{n(n-1)}{2}$. Le temps d'exécution est en $\Theta(n^2)$ (comparaisons et opérations élémentaires en temps constant). □

Question 17.

Après `a = [1, 2, 3]` puis `b = mystere17(a, 4)`, que valent `a` et `b` ?

```
1 def mystere17(L, x):
2     R = L
3     R.append(x)
4     return len(R)
```

Démonstration. On obtient `a = [1, 2, 3, 4]` et `b = 4` : `R` et `L` désignent la même liste. □

Question 18.

Que renvoie `mystere18([2, 4, 6])` ? Que renvoie-t-elle sur la liste vide ?

```
1 def mystere18(L):
2     for x in L:
3         if x % 2 != 0:
4             return False
5     return True
```

Démonstration. Elle renvoie `True` sur `[2, 4, 6]`. Elle renvoie aussi `True` sur la liste vide, puisque la boucle ne s'exécute pas. □

Tri à bulle. Une passe parcourt les paires voisines de gauche à droite et échange deux éléments lorsque celui de gauche est plus grand. Après une passe sur une liste non vide, le plus grand élément se trouve en dernière position. On répète les passes sur une partie de plus en plus courte, jusqu'à obtenir une liste croissante. On ne demande pas d'utiliser de compréhension de liste.

Question 19.

Écrivez `une_passe(L, k)`, qui compare et échange si nécessaire les paires d'indices $(0, 1), (1, 2), \dots, (k - 1, k)$, dans cet ordre. On suppose $0 \leq k < \text{len}(L)$. La liste `L` doit être modifiée sur place ; la fonction ne renvoie rien. Pour $k = 0$, aucune comparaison n'est effectuée. Par exemple, avec `L = [4, 2, 3, 1]`, l'appel `une_passe(L, 3)` transforme `L` en `[2, 3, 1, 4]`.

Démonstration. Code

```
1 def une_passe(L, k):
2     for i in range(k):
3         if L[i] > L[i + 1]:
4             L[i], L[i + 1] = L[i + 1], L[i]
```

□

Question 20.

Écrivez `tri(L)`, qui trie `L` sur place dans l'ordre croissant à l'aide de `une_passe`. Elle ne renvoie rien et doit fonctionner aussi sur les listes vides et à un élément. Précisez les valeurs successives de k utilisées.

Démonstration. Les valeurs de k sont $n - 1, n - 2, \dots, 1$, où n est la longueur de la liste. Pour $n \leq 1$, la boucle ne s'exécute pas.

```
1 def tri(L):
2     for i in range(len(L)-1):
3         k = len(L)-1-i
4         une_passe(L, k)
```

□

Question 21.

On considère une liste L constituée uniquement de 0 et de 1. Écrivez une fonction `tri_2_valeurs(L)` qui place tous les 0 avant les 1, en modifiant la liste sur place. Votre fonction doit utiliser une seule boucle, sans liste auxiliaire, et ne rien renvoyer.

Démonstration. On maintient deux indices : avant i , il n'y a que des 0 ; après j , il n'y a que des 1. La partie située entre les deux indices reste à examiner.

```
1 def tri_2_valeurs(L):
2     i = 0
3     j = len(L) - 1
4     while i <= j:
5         if L[i] == 0:
6             i = i + 1
7         else:
8             tampon = L[i]
9             L[i] = L[j]
10            L[j] = tampon
11            j = j - 1
```

Après un échange, on ne fait pas avancer i : l'élément arrivé à cette position doit encore être examiné.

Invariant : tout ce qui est strictement à gauche de i vaut 0, ce qui est strictement à droite de j vaut 1. □