

Consécutifs et construction de listes

● Méthode

Exercice flash (10 minutes, sur papier). Écrire une fonction `nb_pairs(L: list) -> int` qui renvoie le nombre d'entiers pairs de la liste `L`. Exemple : `nb_pairs([4, 7, 0, 3, 8])` renvoie 3. On rappelle que `x % 2` est le reste de la division de `x` par 2.

Corrigé.

```
def nb_pairs(L: list) -> int:
    '''Renvoie le nombre d'éléments pairs de L.'''
    n = 0
    for x in L:
        if x % 2 == 0:
            n = n + 1
    return n
```

● Méthode

Pour chaque fonction : **exemple à la main, entrée/sortie, schéma, code puis trace**. Pour les éléments consécutifs, écrire les indices des couples sur l'exemple avant de choisir le `range`. Les fonctions `sum`, `max`, `min`, `count` et la méthode `reverse` sont interdites.

● Exercice 1

Échauffement. Pour chaque fonction, écrire d'abord un exemple d'appel et le résultat attendu.

1. `positifs(L: list) -> list` renvoie la liste des éléments strictement positifs de `L`, dans le même ordre.
2. `carres(L: list) -> list` renvoie la liste des carrés des éléments de `L`.
3. `indices_de(L: list, v) -> list` renvoie la liste des indices `i` tels que `L[i] == v`.
4. `ecarts(L: list) -> list` renvoie la liste des différences `L[i+1] - L[i]`. Quelle est la longueur du résultat?
5. `est_croissante(L: list) -> bool` renvoie `True` si la liste est croissante au sens large et `False` sinon. Que doit renvoyer la fonction pour une liste vide ou à un seul élément?

Corrigé.

```
def positifs(L: list) -> list:
    '''Renvoie la liste des éléments strictement positifs de L.'''
    res = []
    for x in L:
        if x > 0:
            res.append(x)
    return res

def carres(L: list) -> list:
    '''Renvoie la liste des carrés des éléments de L.'''
    res = []
    for x in L:
        res.append(x ** 2)
    return res

def indices_de(L: list, v) -> list:
    '''Renvoie la liste des indices i tels que L[i] == v.'''
    res = []
    for i in range(len(L)):
        # on a besoin de l'indice
```

```

        if L[i] == v:
            res.append(i)
        return res

def ecarts(L: list) -> list:
    '''Renvoie la liste des L[i+1] - L[i] ; elle a len(L) - 1 éléments.'''
    res = []
    for i in range(len(L) - 1):
        res.append(L[i + 1] - L[i])
    return res

def est_croissante(L: list) -> bool:
    '''Renvoie True si L est croissante au sens large, False sinon.'''
    for i in range(len(L) - 1):
        if L[i + 1] < L[i]:
            return False          # un contre-exemple suffit
    return True

```

Pour une liste vide ou à un élément, la boucle ne fait aucun tour et la fonction renvoie **True** : il n'y a aucun contre-exemple.

● Exercice 2

Trouver l'erreur. Pour chaque fonction, faire l'appel à la main sur l'exemple indiqué, expliquer l'erreur et corriger le code.

```

# nombre de hausses : exemple [3, 5, 2, 6]
def hausses1(L):
    n = 0
    for i in range(len(L)):
        if L[i + 1] > L[i]:
            n = n + 1
    return n

# même spécification : exemple [7, 5, 2, 4, 4, 6]
def hausses2(L):
    n = 0
    for i in range(len(L)):
        if L[i] > L[i - 1]:
            n = n + 1
    return n

```

```

# éléments non nuls : exemple [0, 3, 0, 5]
def non_nuls(L):
    res = []
    for x in L:
        if x != 0:
            res = res.append(x)
    return res

# croissante ? exemple [1, 4, 2]
def croissante(L):
    for i in range(len(L) - 1):
        if L[i + 1] < L[i]:
            return False
    else:
        return True

```

Corrigé.

- **hausses1** : au dernier tour, $i = 3$ et $L[4]$ n'existe pas, d'où une **IndexError**. Écrire `range(len(L) - 1)`.
- **hausses2** renvoie 3 au lieu de 2, *sans message d'erreur* : pour $i = 0$, $L[-1]$ vaut 6, le dernier élément, et le test $7 > 6$ compte une hausse fictive. Écrire `range(1, len(L))`.
- **non_nuls** : `append` renvoie **None**, donc `res` vaut **None** dès le premier ajout, et l'ajout suivant provoque une **AttributeError**. Écrire `res.append(x)`.
- **croissante** renvoie **True** sur $[1, 4, 2]$: elle conclut dès le premier couple. Supprimer le **else** et placer `return True` après la boucle. De plus, sur une liste à un élément, la fonction renvoie **None**.

● Exercice 3

Dénivelé d'une randonnée (d'après CCINP PC 2025).

Un itinéraire est une liste non vide de triplets (**latitude**, **longitude**, **altitude**). Entre deux points successifs, si l'altitude augmente, la différence d'altitude est un *dénivelé positif*. Le *dénivelé positif cumulé* est la somme de tous ces dénivelés positifs.

1. Écrire une fonction `liste_altitudes(iti: list) -> list` qui renvoie la liste des altitudes des points de l'itinéraire.

2. Écrire une fonction `denivele_positif(alt: list) -> float` qui renvoie le dénivelé positif cumulé à partir de la liste des altitudes.
3. Le GPS est imprécis. Sur une plage d'altitude constante, il enregistre `alt = [0, -2, 2, -2, 2, -2, 2, -2, 2]`. Que renvoie `denivele_positif(alt)`? Commenter.
4. Pour corriger ce défaut, on *lisse* les altitudes : chaque altitude, sauf la dernière, est remplacée par la moyenne d'elle-même et de la suivante ; la dernière est conservée. Écrire `lissage(alt: list) -> list` qui renvoie la liste lissée, sans modifier `alt`.
5. Calculer la liste lissée de la question 3 et son dénivelé positif cumulé.

Corrigé.

```
def liste_altitudes(iti: list) -> list:
    '''iti : liste de triplets (lat, long, alt).
    Renvoie la liste des altitudes des points de iti.'''
    res = []
    for (lat, long, alt) in iti:
        res.append(alt)
    return res

def denivele_positif(alt: list) -> float:
    '''Renvoie la somme des alt[i+1] - alt[i] positifs.'''
    total = 0
    for i in range(len(alt) - 1):
        d = alt[i + 1] - alt[i]
        if d > 0:
            total = total + d
    return total

def lissage(alt: list) -> list:
    '''Renvoie la liste lissée : moyenne de chaque altitude et de la
    suivante, la dernière altitude étant conservée.'''
    res = []
    for i in range(len(alt) - 1):
        res.append((alt[i] + alt[i + 1]) / 2)
    res.append(alt[len(alt) - 1])    # dernière altitude conservée
    return res
```

3. On obtient 16 (quatre montées de 4 m) alors que le dénivelé réel est nul : de petites erreurs répétées s'additionnent.
5. Liste lissée : `[-1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 2]`, de dénivelé positif $1 + 2 = 3$ m, plus proche de la réalité.

● Exercice 4

Mesures défectueuses dans deux spectres (d'après CCINP PC 2026).

Deux spectres sont représentés par deux listes `S1` et `S2` de même longueur. Une mesure défectueuse est remplacée par la valeur `None`. On teste si une valeur est absente avec `x is None`.

1. Écrire une fonction `positions_a_rejeter(S1: list, S2: list) -> list` qui renvoie la liste des indices `i` pour lesquels `S1[i]` ou `S2[i]` vaut `None`.
Exemple : pour `S1 = [1.0, None, 2.5, 3.0]` et `S2 = [0.5, 1.5, 2.0, None]`, le résultat est `[1, 3]`.
2. Écrire une fonction `valeurs_valides(S: list) -> list` qui renvoie la liste des valeurs de `S` différentes de `None`.
3. Pourquoi ne peut-on pas écrire la question 1 avec un parcours par valeur?

Corrigé.

```
def positions_a_rejeter(S1: list, S2: list) -> list:
    '''S1, S2 : listes de même longueur.
    Renvoie les indices i tels que S1[i] ou S2[i] vaut None.'''
    res = []
    for i in range(len(S1)):
        if S1[i] is None or S2[i] is None:
            res.append(i)
    return res

def valeurs_valides(S: list) -> list:
    '''Renvoie la liste des éléments de S différents de None.'''
    res = []
    for x in S:
        if x is not None:
            res.append(x)
    return res
```

Question 3 : on doit lire les deux listes au même rang et renvoyer ce rang ; l'indice est indispensable.

● Exercice 5

Pour aller plus loin : le plateau d'awalé (d'après CCINP PC 2024).

Le plateau est une liste `plateau` de 12 entiers : `plateau[i]` est le nombre de graines de la case `i`. Les cases 0 à 5 sont celles du joueur dont c'est le tour, les cases 6 à 11 celles de son adversaire. Les cases sont parcourues dans l'ordre 0, 1, ..., 11, puis on revient à la case 0.

1. Écrire une fonction `cases_jouables(plateau: list) -> list` qui renvoie la liste des indices des cases *non vides* du joueur dont c'est le tour.
2. À la fin d'un tour, on échange les deux moitiés du plateau : les cases 6 à 11 deviennent les cases 0 à 5 et inversement. Écrire une fonction `tourner(plateau: list) -> list` qui renvoie le nouveau plateau.
3. Quel est l'indice de la case qui suit la case `i` ? Donner une expression valable pour `i = 11`.
4. *Semer* la case `case` consiste à prendre toutes ses graines (la case devient vide), puis à déposer une graine dans chacune des cases suivantes, en sautant la case de départ si l'on fait un tour complet. Écrire une fonction `deplacer_graines(plateau: list, case: int) -> int` qui modifie `plateau` en place et renvoie l'indice de la case où la dernière graine a été déposée.

Tests : pour `plateau = [4]*12` et `case = 5`, le résultat est 9 ; pour `plateau = [0, 0, 2, 0, 16, 5, 2, 1, 0, 1, 0, 1]` et `case = 4`, le résultat est 9 et le plateau devient `[1, 1, 3, 1, 0, 7, 4, 3, 2, 3, 1, 2]`.

Corrigé.

```
def cases_jouables(plateau: list) -> list:
    '''Renvoie les indices des cases non vides parmi les cases 0 à 5.'''
    res = []
    for i in range(6):
        if plateau[i] > 0:
            res.append(i)
    return res

def tourner(plateau: list) -> list:
    '''Renvoie le plateau dont les deux moitiés sont échangées.'''
    res = []
    for i in range(12):
        res.append(plateau[(i + 6) % 12])
    return res

def deplacer_graines(plateau: list, case: int) -> int:
    '''Sème les graines de la case case (modification en place).
    Renvoie l'indice de la case de la dernière graine semée.'''
```

```

graines = plateau[case]
plateau[case] = 0
i = case
while graines > 0:
    i = (i + 1) % 12          # case suivante, retour à 0 après 11
    if i != case:            # on saute la case de départ
        plateau[i] = plateau[i] + 1
        graines = graines - 1
return i

```

Question 3 : la case suivante est $(i + 1) \% 12$, qui vaut bien 0 pour $i = 11$. Pour **tourner**, on peut aussi écrire **return plateau[6:] + plateau[:6]**.

● Méthode

Travail personnel : deux exercices flash (15 minutes chacun).

1. `sommes_cumulees(L: list) -> list` renvoie la liste dont l'élément d'indice i est la somme des éléments de L d'indices 0 à i . Exemple : `sommes_cumulees([2, 5, 1])` renvoie `[2, 7, 8]`.
2. `nb_changements_signe(L: list) -> int` renvoie le nombre de fois où deux éléments consécutifs de L sont de signes strictement opposés. Exemple : `nb_changements_signe([3, -1, -4, 2, 5])` renvoie 2.

Corrigé.

```

def sommes_cumulees(L: list) -> list:
    '''Renvoie la liste des sommes L[0] + ... + L[i].'''
    res = []
    s = 0                                # deux accumulateurs : un nombre et une liste
    for x in L:
        s = s + x
        res.append(s)
    return res

def nb_changements_signe(L: list) -> int:
    '''Renvoie le nombre d'indices i tels que L[i] et L[i+1] sont de
    signes strictement opposés.'''
    n = 0
    for i in range(len(L) - 1):
        if L[i] * L[i + 1] < 0:
            n = n + 1
    return n

```