

Fiche méthode : maximum et copies

Chercher le meilleur – Copier ou modifier une liste

1. Le schéma « meilleur élément »

On garde en mémoire le **meilleur candidat rencontré jusqu'ici** et on le remplace chaque fois qu'on trouve mieux. Aux concours, `max` et `min` sont souvent interdits.

```
# valeur du maximum (L non vide)
m = L[0]
for x in L:
    if x > m:
        m = x
return m
```

```
# indice du maximum (L non vide)
i_max = 0
for i in range(1, len(L)):
    if L[i] > L[i_max]:
        i_max = i
return i_max
```

● Méthode

- **Initialiser avec un élément de la liste** (le premier), jamais avec 0 : si tous les éléments sont négatifs, 0 ne serait jamais battu.
- Pour renvoyer l'**indice**, ne mémoriser **que l'indice** : la valeur se retrouve avec `L[i_max]`. Deux variables à tenir à jour, c'est deux occasions d'en oublier une.
- > garde le **premier** maximum en cas d'égalité, >= garde le **dernier**.
- Pour un minimum, on peut aussi initialiser avec `float('inf')`, plus grand que tout nombre. C'est l'écriture utilisée dans les sujets pour Dijkstra et Prim.

On cherche...	Comparaison
le point le plus haut d'un itinéraire de triplets	<code>iti[i][2] > iti[i_max][2]</code>
le mot le plus long	<code>len(mots[i]) > len(mots[i_max])</code>
le centre le plus proche de S	<code>distance(S, C[i]) < distance(S, C[i_min])</code>

2. Le meilleur parmi ceux qui vérifient une condition

Aucun élément ne convient peut-être : on initialise avec **None** (« pas encore de candidat »).

```
def plus_petit_positif(L: list) -> float:
    '''Renvoie le plus petit élément strictement positif de L,
    ou None s'il n'y en a pas.'''
    meilleur = None
    for x in L:
        if x > 0 and (meilleur is None or x < meilleur):
            meilleur = x
    return meilleur
```

3. Une variable est une étiquette

```
L = [1, 2, 3]
M = L          # même liste, deux noms
M[0] = 9
print(L)       # [9, 2, 3]
```

```
L = [1, 2, 3]
M = L.copy()   # nouvelle liste
M[0] = 9
print(L)       # [1, 2, 3]
```

● Méthode

- `M = L` ne crée **aucune** liste : `M` et `L` désignent la même. Pour une vraie copie : `L.copy()` ou `L[:]`.
- `.copy()` ne copie qu'un niveau. Pour une liste de listes, il faut copier chaque ligne : `[ligne.copy() for ligne in M]`.
- Une fonction qui reçoit une liste peut la **modifier en place** (`L[i] = ...`, `L.append(...)`) : la modification est visible après l'appel. Si l'énoncé dit « les arguments ne doivent pas être modifiés », travailler sur une copie.

4. Les pièges

● Attention !

1. Initialiser le maximum à 0.
2. Renvoyer la valeur quand l'énoncé demande l'indice (et inversement).
3. Écrire `M = L` en croyant copier, puis modifier `M`.
4. Créer une matrice avec `[[0] * 3] * 2` : les deux lignes sont *la même* liste. Écrire `[[0] * 3 for i in range(2)]`.
5. Utiliser `L.sort()` pour trouver un maximum : c'est interdit aux concours, plus coûteux ($O(n \log n)$) au lieu de $O(n)$, et la liste `L` est modifiée.

5. Exemple rédigé

Écrire une fonction `indice_plus_haut(iti: list) -> int` qui renvoie l'indice du point le plus haut d'un itinéraire non vide de triplets (`lat`, `long`, `alt`).

1. **Exemple** : altitudes 261, 267, 280, 275 → indice 2.
2. **Entrée / sortie** : une liste non vide de triplets ; un entier.
3. **Schéma** : meilleur élément, version indice ; le critère est la composante d'indice 2.
4. **Code et trace** :

```
def indice_plus_haut(iti: list) -> int:
    '''iti : liste non vide de triplets.
    Renvoie l'indice du premier point
    d'altitude maximale.'''
    i_max = 0
    for i in range(1, len(iti)):
        if iti[i][2] > iti[i_max][2]:
            i_max = i
    return i_max
```

i	alt.	max prov.	i_max
-	-	261	0
1	267	267	1
2	280	280	2
3	275	280	2