# Chapitre 2 - Recherche du minimum d'une liste

On aborde dans ce chapitre la notion de complexité appliqué à un algorithme de recherche du minimum d'une liste.

## 1- Rappels sur les listes :

Point Cours vu en 1 NSI :

- Créer une liste vide : `nomListe = []`
- Créer une liste non vide : `nomListe = [5,"n",True,3.14]`
- Ajouter un élément en fin de liste : `nomListe.append("s")`
- Taille d'une liste : `len(nomListe)`
- Accéder à la valeur à l'index 2 pour lire ou modifier : `nomListe[2]`
- Supprimer l'élément à l'index 2 : `del(nomListe[2])`
- Savoir si un élément est dans la liste : `test = 5 in nomListe`
- Parcourir la liste par éléments :

```
for elt in nomListe :
    print(elt)
```

- Parcourir la liste par index :

```
for i in range(len(nomListe)) :
    print(nomliste[i])
```

Complément pour la terminale :

- Créer une liste par COMPREHENSION :

```
nomListe = [i**2 for i in range(5)]
```

ou

```
nomListe = [i**2 for i in range(5) if i**2 < 4]
```

- Méthode pop() : `element = nomListe.pop(1)` , supprime l'élément qui est à l'index 1 et renvoie sa valeur.

- Si `nomListe = [10,11,12,13,14]` , alors :

```
>>> nomListe[1:]
[11, 12, 13, 14]
```

```
>>> nomListe[:3]
[10, 11, 12]
```

```
>>> nomListe[1:3]
[11, 12]
```

Point Cours :

On peut utiliser les outils dédiés aux listes pour lire les caractères d'un string.
ATTENTION, seuls les outils de **lecture** sont accessibles. Pas possible d'utiliser ces outils pour modifier, ajouter, ou supprimer les caractères du string.

## 2- ALGORITHME DE RECHERCHE DU MINIMUM D'UNE LISTE :

Exercice : On donne le code incomplet ci-contre.

```
from random import randint
from time import time

# script des fonctions
def minimumListe(liste) :
    iMin = 0
    for i in range(1,len(liste)) :
        if liste[i] < liste[iMin] :
            iMin = i
    return iMin , liste[iMin]


# programme principal
N = 10
liste = [randint(-5*N , 5*N) for i in range(N)]
deb = time()
iMin,vMin = minimumListe(liste)
fin = time()
if N < 100 : print(liste)
print(f"""liste de taille {N}
temps de recherche = {fin-deb} s
minimum = {vMin} pour l'index {iMin}
""")
```

CORRIGE

Une exécution de ce code donne dans la console :

```
>>> (executing file "minimum.py")
[-6, -4, -37, 13, 41, 44, -47, -21, -29, -30]
liste de taille 10
temps de recherche = 0.0 s
minimum = -47 pour l'index 6
```

⇨ Compléter ce code en écrivant le script de la fonction *minimumListe()*. L'enregistrer dans un fichier nommé *minimum.py* .

## *3-* Notion de complexite pour cet algorithme :

### *a.* Experimentation :

On s'intéresse à présent au temps de calcul pour des listes de taille importante.

⇨ Utiliser le script mis au point pour des listes de taille 1 million, 10 millions, 100 millions. Noter les temps de calcul dans le tableau ci-contre (à noter aussi en commentaire dans le fichier *minimum.py*)

| Recherche du min d'une liste de 1 million de valeurs | Recherche du min d'une liste de 2 millions de valeurs | Recherche du min d'une liste de 10 millions de valeurs | Recherche du min d'une liste de 100 millions de valeurs |
|---|---|---|---|
| Temps de calcul en s : | Temps de calcul en s : | Temps de calcul en s : | Temps de calcul en s : |

⇨ Uploader le fichier nommé *minimum.py* .

### *b.* Complexite :

Le calcul de la complexité d'un algorithme, permet de mesurer sa performance. C'est un outil qui permet comparer l'**efficacité** d'algorithmes résolvant le même problème.

Réaliser un calcul de complexité en temps revient à compter le nombre d'**opérations élémentaires** (affectation, calcul arithmétique ou logique, comparaison…) effectuées par l'algorithme. Par soucis de simplicité, on fera l'hypothèse que toutes les opérations élémentaires réalisées sont à **égalité de coût**, soit 1 « unité » de temps. Le calcul de cette complexité dépend de la taille des données traitées : plus ces données seront volumineuses, plus il faudra d'opérations élémentaires pour les traiter. On notera $n$ le nombre de données à traiter.

Concrètement, sur le script de recherche du minimum d'une liste de taille $n$ :

- Appel fonction + initialisation des variables au départ : opérations
- Pour chaque valeur de la liste : test valeur + actualisation possible de 2 variables : opérations
- Retour fonction : opération

Total : opérations

Par soucis de simplification, on donne uniquement un ordre de grandeur asymptotique, noté $O$

Point Cours :

Pour rechercher le minimum d'une liste de taille $n$ , on estime que le nombre d'opérations élémentaires réalisées dans le pire des cas est égal à

On dit que cet algorithme a une complexité de classe $O(n)$ ou aussi que la complexité de cet algorithme est *linéaire*. Si on multiplie la taille de la liste à traiter par 10 par exemple, les temps de calcul seront « *à peu près* » multipliés par 10 également.

Il existe d'autres classes de complexité :

## Classes de complexité

| $\mathcal{O}$ | **Type de complexité** |
| --- | --- |
| $\mathcal{O}(1)$ | constante |
| $\mathcal{O}(log(n))$ | logarithmique |
| $\mathcal{O}(n)$ | linéaire |
| $\mathcal{O}(n \times log(n))$ | quasi-linéaire |
| $\mathcal{O}(n^2)$ | quadratique |
| $\mathcal{O}(n^3)$ | cubique |
| $\mathcal{O}(2^n)$ | exponentielle |
| $\mathcal{O}(n!)$ | factorielle |