Chapitre 3 - Tri par sélection

On s'intéresse dans ce chapitre à un algorithme classique, utilisé pour trier les éléments d'une liste.

1- Travail preliminaire : fonction qui echange la position de 2 elements d'une liste.

Exercice : On donne le code incomplet ci-contre.

```
# script des fonctions
def echange(liste , i , j) :









# programme principal
l = [10,11,12,13,14,15]
retour = echange(l,0,5)
print(retour,l)

l = [10,11,12,13,14,15]
retour = echange(l,7,5)
print(retour,l)

l = [10,11,12,13,14,15]
retour = echange(l,-2,5)
print(retour,l)

l = [10,11,12,13,14,15]
retour = echange(l,4,1)
print(retour,l)
```

L'exécution de ce code donne dans la console, donne :

```
>>> (executing file "tris.py")
True [15, 11, 12, 13, 14, 10]
False [10, 11, 12, 13, 14, 15]
False [10, 11, 12, 13, 14, 15]
True [10, 14, 12, 13, 11, 15]
```

Ce code permet ainsi d'échanger 2 éléments d'une liste.

⇨ Compléter le script de la fonction *echange()*

dans un fichier nommé *selection.py* .

Remarque : ne pas utiliser l'opération : `liste[i] , liste[j] = liste[j] , liste[i]`

qui est certes valide en python, mais ne l'est pas dans la plupart des langages de programmation.

2- TRI PAR SELECTION D'UNE LISTE

Principe du TRI **SELECTION** pour une liste de n valeurs :

1 ⇨ On recherche le minimum des n valeurs et on le met à sa place en 1ère position, en l'échangeant.

2 ⇨ On recherche le minimum des $n-1$ valeurs restantes et on le met à sa place en 2ième position, en l'échangeant.

3 ⇨ On recherche le minimum des $n-2$ valeurs restantes et on le met à sa place en 3ième position, en l'échangeant.

….. etc, jusqu'à ce que tous les éléments soient à leur place.

Exemple : Tri sélection de la liste `[30 , -7 , 1 , 6 , 4 ]`

	Etat de la liste	
Etape 1		Echange des indices :
Etape 2		
Etape 3		
Etape 4		

Exercice : ⇨ Compléter le fichier *selection.py* en y écrivant le script d'une fonction nommé *triSelection()* qui trie la liste mise en argument en utilisant le principe du tri sélection. Pour échanger les éléments de la liste, vous utiliserez la fonction *echange()* mis au point dans la question précédente. L'exécution du programme principal ci-contre, donnera dans la console :

```
# programme principal
l = [30 , -7 , 1 , 6 , 4 ]
triSelection(l)
print('Sélection :',l)
```

```
>>> (executing file "selection.py"
Sélection : [-7, 1, 4, 6, 30]
```

3- COMPLEXITE DE CET ALGORITHME :

a. EXPERIMENTATION :

A l'image de ce qui a été fait pour le code qui recherchait le minimum d'une liste, on complète le code du fichier *selection.py* en important les fonctions *randint()* et *time()* et en utilisant le programme principal ci-contre (copié et modifié à partir du fichier *minimum.py*).

```
# programme principal
N =10
liste = [randint(-5*N , 5*N) for i in range(N)]
if N < 100 : print('liste non triée : \n',liste)
deb = time()
triSelection(liste)
fin = time()
if N < 100 : print('liste triée : \n', liste)
print(f"""liste de taille {N}
temps de recherche = {fin-deb} s
""")
```

⇨ Réaliser les exécutions qui permettent de compléter le tableau ci-dessous (à noter aussi en commentaire dans le fichier *selection.py*)

Tri d'une liste de 10 valeurs	Tri d'une liste de 100 valeurs	Tri d'une liste de 1000 valeurs	Tri d'une liste de 10000 valeurs	Tri d'une liste de 20000 valeurs	Tri d'une liste de 40000 valeurs
Temps de calcul en s :	Temps de calcul en s :	Temps de calcul en s :	Temps de calcul en s :	Temps de calcul en s :	Temps de calcul en s :

⇨ Uploader le fichier nommé *selection.py* .

b. COMPLEXITE :

Sur le script de tri par sélection d'une liste de taille n , le nombre d'opérations élémentaires est :

- Appel fonction + initialisation des variables au départ : opérations
- Pour chaque valeur de la liste : initialisation variable
  et pour chaque valeur restants à droite : comparaison
  + affectation éventuelle opérations
- Retour fonction : opération

Total : opérations

Point Cours :

Pour trier 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 $\mathcal{O}(n^2)$ ou aussi que la complexité de cet algorithme est *quadratique*. Si on multiplie la taille de la liste à traiter par 10 par exemple, les temps de calcul seront « *à peu près* » multipliés par $10^2 = 100$.

4- APPLICATION :

⇨ Télécharger le fichier *prenoms.zip* proposé sur *nsibranly.fr*. Décompresser et copier les 2 fichiers qu'il contient, dans votre répertoire de travail sur le tri par sélection.

⇨ Ouvrir le fichier *prenomFrance.csv* avec un tableur, *Excel* par exemple. Il contient des centaines de milliers de lignes qui indiquent chacune 4 données : sexe (1 pour garçon, 2 pour fille), prénom en lettres majuscule, année de naissance et nombre de nouveaux nés en France avec ce prénom sur l'année indiquée. Ce fichier a été téléchargé sur le site de l'INSEE.

```
[1, 'JOSÉ-MANUEL', 1967, 26]
[1, 'JOSÉ-MANUEL', 1968, 19]
[1, 'JOSÉ-MANUEL', 1969, 27]
[1, 'JOSÉ-MANUEL', 1970, 38]
[1, 'JOSÉ-MANUEL', 1971, 57]
[1, 'JOSÉ-MANUEL', 1972, 22]
[1, 'JOSÉ-MANUEL', 1973, 38]
[1, 'JOSÉ-MANUEL', 1974, 51]
[1, 'JOSÉ-MANUEL', 1975, 37]
[1, 'JOSÉ-MANUEL', 1976, 40]
```

⇨ Exécuter le code de fichier *prenom.py* . Il permet de lire le fichier csv et de copier toutes les données contenues dans une liste double. Pour l'instant le script affiche un extrait de 10 éléments de cette liste, dans la console :

Exercice : On souhaite compléter ce code :

- en créant une fonction *recherche()* qui renvoie une liste ℓ contenant uniquement les éléments de la liste complète qui concerne le prénom mis en argument (ici 'ANAIS')

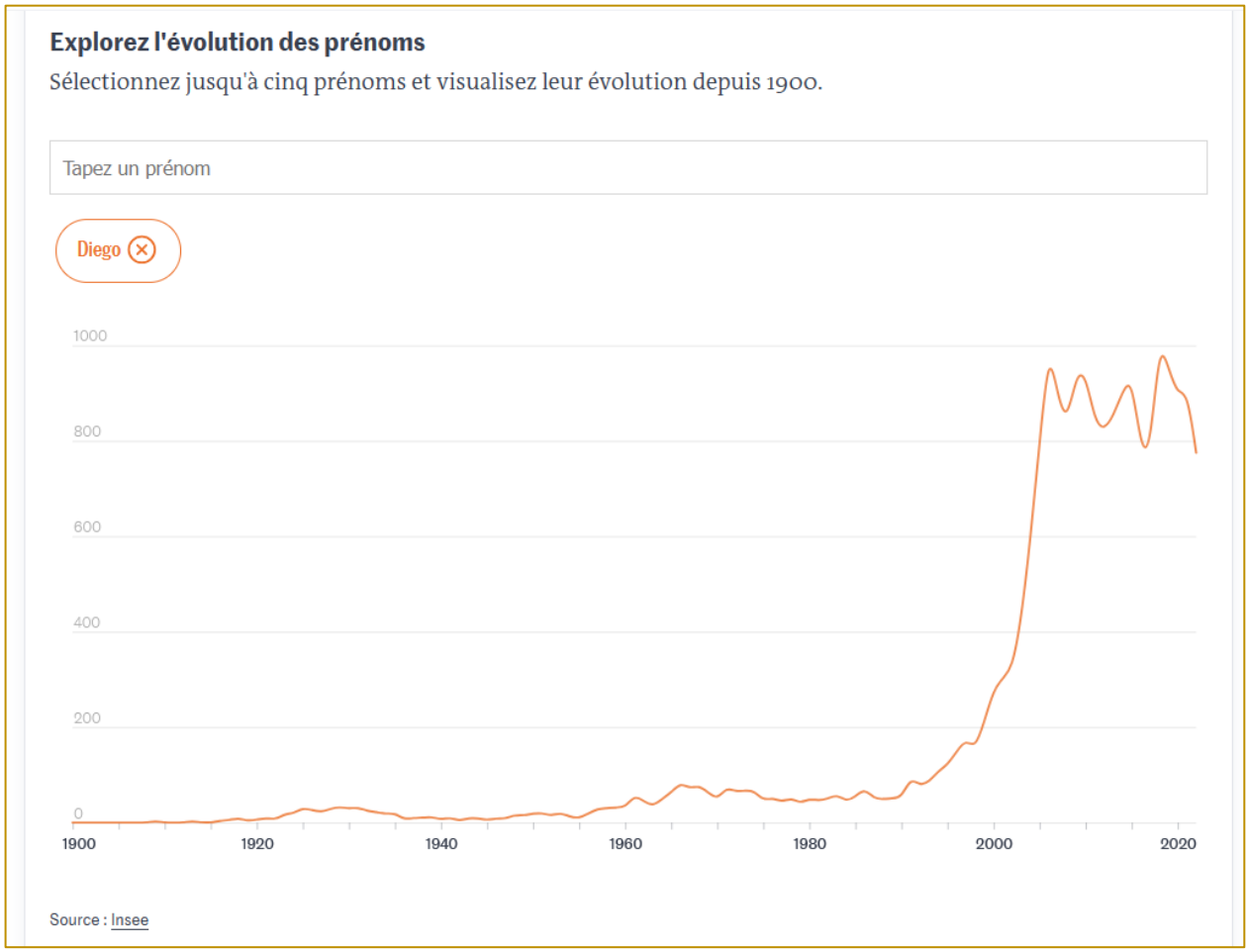
```
# programme principal
listeComplete = prenoms()
l = recherche('ANAIS',listeComplete)
triSelection(l)
for elt in l :
    print(elt)
```

- en reprenant la fonction *triSelection()* du fichier *selection.py* et en la modifiant afin qu'elle puisse réaliser l'opération de tri par ordre décroissant cette-fois ci, sur le nombre de naissance de l'année. Il s'agira d'adapter le script afin de le rendre compatible avec un tri de liste double.

On obtiendra normalement avec ce prénom 'ANAIS' le résultat suivant :

```
>>> (executing file "prenomsCorrige.py")
[2, 'ANAIS', 2010, 1014]
[2, 'ANAIS', 2011, 944]
[2, 'ANAIS', 2012, 421]
[2, 'ANAIS', 2015, 315]
[2, 'ANAIS', 2014, 311]
[2, 'ANAIS', 2016, 222]
[2, 'ANAIS', 2013, 152]
[2, 'ANAIS', 2017, 118]
[2, 'ANAIS', 2019, 68]
[2, 'ANAIS', 2022, 68]
[2, 'ANAIS', 2018, 65]
[2, 'ANAIS', 2020, 62]
[2, 'ANAIS', 2021, 57]
[2, 'ANAIS', 1994, 7]
```

On prolonge l'exploitation de ces données très intéressantes, en traçant des courbes de popularité de prénom, un peu à l'image de ce qui a été fait dans un article du Monde daté du 15 juillet 2023 : https://www.lemonde.fr/les-decodeurs/article/2023/07/15/explorez-la-popularite-des-prenoms-en-france-depuis-1900_6182081_4355770.html



On se propose de tracer les mêmes courbes, mais ici en utilisant la bibliothèque *matplotlib* qui permet de visualiser facilement des courbes. Jetez un coup d'œil sur le site suivant : https://matplotlib.org/ dans la rubrique *Exemples* ce qu'il est possible de faire. Pour ne pas perdre trop de temps, on donne ici un script basique qui permet déjà de tracer le type de courbes qui nous intéressent … essayez-le.

```
from matplotlib.pyplot import plot,grid,show # importation fonctions

X=[2010,2014,2015,2021]  # liste des abscisses
Y1=[510,1400,1800,640]  # liste des ordonnées
Y2=[470,1200,300,100]   # autre liste des ordonnées si on veut supperposer plusieurs
courbes

plot(X,Y1,'r:x') # ajout du nuage de points X,Y1
plot(X,Y2,'b:o') # ajout du nuage de points X,Y2

grid(True) # ajout d'une grille
show() # affichage du tracé dans une fenêtre
```

⇨ Améliorer votre script pour y intégrer ce tracé de courbes. …. Une fois fini, uploader le fichier *prenom.py* .