TP	TP 3.4	TSI1
	Tri par sélection	Période 4
		1h

1) Créer un fichier **TP_3_4_tri_selection.py** et ajouter en début de fichier les instructions permettant d'importer les bibliothèques utiles au TP :

- import de la fonction **randint()** de la bibliothèque **random**,
- import de la fonction **time()** de la bibliothèque **time**.

1 Algorithme du tri par sélection

2) Ecrire une fonction **minimum()** qui admet comme argument une liste **L** d'éléments comparables et qui renvoie l'indice **im** du 1er minimum.

3) Réaliser dans le corps du programme un jeu de test pour vérifier que votre algorithme renvoie bien les valeurs attendues (liste avec quelques éléments quelconques, avec des éléments tous identiques, avec des éléments par ordre décroissant).

Pseudo code fonction tri_selection admettant comme argument une liste L d'éléments comparables.

```
n <- nombre d'éléments de L
Pour i variant de 0 à n-1 inclus répéter
    im <- minimum de l'extraction de la liste L de l'indice i jusqu'à la fin de liste
    Permuter les éléments de L d'indice i et im
Fin Pour
Renvoyer L
Fin de la fonction tri_selection
```

4) Traduire en python la fonction **tri_selection()**, décrite en pseudo-code, qui admet comme argument une liste d'éléments comparables et qui renvoie la liste triée en faisant appel notamment à la fonction **minimum()**.

5) Réaliser dans le corps du programme un jeu de test pour vérifier que votre algorithme ne renvoie pas les valeurs attendues (liste avec quelques éléments quelconques, avec des éléments tous identiques, avec des éléments par ordre décroissant).

6) Identifier dans le pseudo-code, l'erreur qui a été réalisée (au besoin, comparer avec l'algorithme du cours). Vérifier qu'une fois cette erreur corrigée, la fonction renvoie bien le résultat attendu.

2 Performance du tri par sélection

2.1 Stabilité

7) Affecter **100** à la variable **n** correspondant au nombre d'éléments de la liste **lc** que nous aurons à trier.

8) Définir une liste **lc** en compréhension contenant **n** chaines de caractères obtenue par **i** réplications d'une lettre majuscule aléatoire (où **i** est le variant de la boucle **for** qui varie de **1** à **n** inclus) :

- la fonction **randint(a,b)** renvoie un entier aléatoire dans l'intervalle **[a,b]**,
- une lettre majuscule peut s'obtenir à partir de son numéro dans la table UTF8 à l'aide de la fonction **chr()** (**chr(65)** renvoie **'A'** et **chr(90)** renvoie **'Z'**), à l'inverse, **ord('A')** renvoie **65**.
- la réplication d'une lettre peut être obtenue par l'instruction **'A'*3** qui renvoie **'AAA'**.

9) Copier la fonction **minimum()** et la modifier afin qu'elle renvoie l'indice du minimum à partir du premier caractère de chaque chaine de caractères. On la renommera **mini2()** et **tri_selection2()**. (On utilisera l'instruction L.copy() pour ne pas modifier la liste passée en entrée)

10) Affecter à **lct** la liste triée renvoyée par la fonction **tri_selection2()** à partir des éléments de la liste **lc** (on ne souhaite pas que la liste globale **lc** soit triée).

11) A l'aide de l'explorateur de variable, parcourir les éléments de **lct** et vérifier que le tri est effectif.

2.2 Complexité = durée du tri en fonction de la taille de la liste

12) Affecter 2 variables :

- **N = 1000** pour définir le nombre d'éléments de la liste **lc** de chaines de caractères à trier.
- **Nm = 10 000** pour la taille des chaines de caractères constituants les éléments de la liste **lc** à trier.

13) Définir une liste **lc2** en compréhension contenant N chaines de caractères obtenue par **Nm** réplications d'une lettre majuscule aléatoire :

14) A l'aide de la fonction **time()** qui renvoie la valeur en seconde depuis l'origine des dates, déterminer la durée du tri par sélection avec la liste **lc2**.

15) Montrer que la longueur **Nm** des mots de la liste à peu d'effet sur la durée du tri (on pourra multiplier sa valeur par 2 et vérifier que l'impact sur la durée du tri est modéré).

16) Montrer que la taille **N** de la liste à trier a un fort impact sur la durée du tri puisque la multiplication par **2** du nombre d'éléments **n** de la liste **lc2** conduit à une durée de tri **2^2** fois supérieure. Le tri par insertion est bien quadratique.

17) Montrer que le **tri_selection2()** est également quadratique pour des listes déjà triée ou pour des listes triées en sens inverse (pour générer une liste triée en sens inverse, on implémentera une fonction **tri_selection3()** qui trie à l'aide d'une fonction **maximum()**.