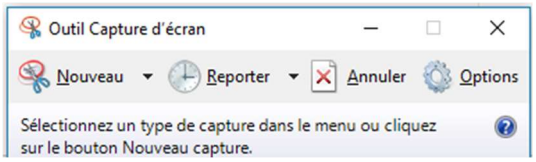
TP2 TRI PAR SELECTION D'UNE LISTE

OBJECTIFS : Les objectifs de ce TP sont :

- d'écrire un algorithme qui permette de trier une liste en utilisant un algorithme du type « *tri par sélection* »,
- d'intégrer cet algorithme dans une fonction,
- de déterminer la durée des différentes opérations réalisées lorsque la liste est de grande taille,
- d'écrire les valeurs des listes triées dans un fichier *.txt* .

DOCUMENT A RENDRE : Ce travail est évalué. Vous en rédigerez un compte-rendu sous le nom « *tp2_nomfamille.doc* » ou « *tp2_nomfamille.odt* » et vous le transférerez en fin d'activité **par l'intermédiaire de l'onglet transfert** du site https://nsibranly.fr/transferts_terminale.php en utilisant comme identifiant : **tp2** et mot de passe : **tp2** . Ce compte-rendu contiendra :

- les réponses aux différentes questions posées,
- les captures d'écran des morceaux de codes et celles des résultats des exécutions données dans le shell. Pour faire ces captures, utiliser *l'Outil Capture d'écran* de Windows.



Pour commencer, ouvrir dans Pyzo un fichier nommé « tri_selection.py » . Importer les bibliothèques random et time :

```
from random import *
import time
```

1. Création de listes

- Créer une liste nommée « *liste_test* » contenant 5 nombres aléatoires compris entre 0 et 10. Afficher le contenu de cette liste.

2. Algorithme de « tri par sélection »

- Découvrez le principe du tri par insertion en utilisant le lien donné dans la rubrique *Documents*.

- Si on applique cette technique à la liste [7, 2 , 1 , 4] , le tri par sélection se fait en 3 étapes :
  - 1ère étape : le min de toute la liste [7, 2 , 1 , 4] est placé au rang 0 : [1 , 2 , 7 , 4]
  - 2ième étape : le min de la liste à droite du minimum précédent [1 , 2 , 7 , 4] est placé au rang 1. Ici, il y est déjà, donc rien ne change : [1 , 2 , 7 , 4]
  - 3ième étape : le min de la liste à droite du minimum précédent [1 , 2 , 7 , 4] est placé au rang 2 : [1 , 2 , 4 , 7]

PRINCIPE A RETENIR : On est certain à chaque étape, que les éléments à gauche de l'élément déplacé sont triés dans l'ordre et sont à leur place définitive.

⇨ Ecrire sur votre compte-rendu les étapes nécessaires pour trier la liste [2, 4, 6, 5, 1]

- Ecrire le code d'une fonction nommée « tri_selection(liste) » qui retourne la liste entrée en argument, triée par un algorithme du type « tri par sélection ».
  Tester cette fonction « liste_test ».

3. Performance de cet algorithme de tri

- Modifier la fonction « *tri_selection(liste)* » afin qu'elle retourne la liste triée **et** le nombre d'opérations réalisées dans ce tri (insérer un compteur).

- En utilisant la fonction *time(),* déterminer les temps de calcul pour trier « *liste_test* ». Compléter finalement le tableau ci-dessous :

	Temps mis pour trier (en s)	Nombre d'opérations réalisées pour effectuer ce tri
liste_test		

- Créer une liste nommée « *grande_liste* » par compréhension qui contient 100 valeurs de nombres entiers aléatoires compris entre 0 et 100 .

- Utiliser la fonction « *tri_selection(liste)* » pour trier cette liste et écrire les 100 valeurs triées dans un fichier nommé « *liste_trie.txt* ». Vérifier que le tri a été effectué correctement. Justifier toujours en joignant des copies d'écran.

- En utilisant la fonction *time(),* déterminer le temps de calcul pour trier « *grande_liste* ». Compléter le tableau ci-dessous :

Taille de « *grande*_liste »	100	1000	10 000	100 000
Durée du tri en s				
Nombre d'opérations nécessaires pour trier				

- Une fonction native de python nommée « *sorted(liste)* » retourne la liste en argument triée. Exécuter dans le shell : *sorted([5,9,1,0])* pour vous en convaincre. Afin d'analyser l'efficacité en temps de calcul, de la fonction « *tri_selection(liste)* » , il est intéressant de comparer les durées précédentes avec celles obtenues en employant « *sorted(liste)* ».
  Compléter le tableau ci-dessous et conclure :

Taille de « *grande*_liste »	100	1 000	10 000	100 000	1 000 000	10 000 000
Durée du tri en s						