Toutes les fonctions à remplir se trouvent dans le fichier **tris.py**. Vous êtes priés d'exécuter cette feuille vie **F5** sous peine d'avoir des petits problèmes d'importation (surtout pour la fin du TD).

I Deux tris

Le but est ici d'implémenter (ie. écrire en python) deux algorithmes qui trient une liste d'entier donnée en paramètre.

1.1 Tri par sélection

Principe : il s'agit ici de trouver le minimum de la liste et de le placer en première position, puis de trouver le minimum de la liste restante (à partir de l'indice 1, donc) et de le placer en deuxième position et de répéter jusqu'à l'avant dernier indice (car évidemment, le seul élément restant sera le maximum de la liste).

Exercice 1
Créer une fonction **echange(L, i, j)** qui échange les éléments d'indices i et j dans la liste L.

Exercice 2
Vous disposez d'une fonction **mini_a_partir** qui reprend les idées vues en cours.
Grâce à cette fonction et à **echange**, compléter la fonction **tri_selection**

1.2 Tri par insertion

Principe : On considère que le début de la liste est triée (c'est vrai si on considère seulement le premier élément de la liste comme début) puis on ajoute le prochain élément (celui d'indice 1, puis celui d'indice 2...) à "la bonne place" en le faisant remonter (par échange successif vers la gauche).

Exercice 3
Finir de remplir la fonction **insert(L, i)** qui prend une liste L et un indice i comme argument. On suppose dans le corps de cette fonction que $L[0 : i]$ (les éléments d'indices $0, 1, \ldots, i-1$) est triée dans l'ordre croissant. Le but est de déplacer successivement l'élément $x = L[i]$ depuis l'indice jusqu'à la "bonne" position qui fait que $L[0 : i+1]$ est triée.

Exercice 4
Compléter la fonction **tri_insertion**.

II Compter les opérations

2.1 Modification de l'existant

Le but est ici de compter le nombre d'affectation (un élément de la liste change de valeur) et de comparaisons (entre deux éléments de la liste) nécessaires à nos fonctions pour trier une liste. On tente de déterminer par la pratique quel est la plus efficace de nos implémentations.

Exercice 5
Copiez vos fonction (qui ont été testées, je vous le rappelle) et modifiez les pour qu'elles aient comme valeur de retour le couple (affectations, comparaisons) des nombres évoqués ci-dessus.
Indication : on devra calculer au brouillon le nombre de comparaisons effectuées par **mini_a_partir(L, i)** en fonction de la longueur de L et de i.

Testez avec une liste facile (comprendre : pour laquelle vous aurez trouvé les nombres d'opérations à la main).

2.2 Graphiques

But Obtenir des graphique d'évolution du nombre d'affectations, de comparaisons et d'opérations (la somme des deux précédents) pour nos deux tris en fonction de la longueur de la liste à trier.

Graphiques de sortie Pour chacune des trois mesure, on veut obtenir des courbe représentant la donnée étudiée pour des listes de longueur $100, 200, \ldots, 2000$ avec le tri par sélection en vert, le tri par insertion en rouge ainsi qu'une légende et un titre évocateur.

```
plt.plot(X, Y, 'nom_de_la_couleur', label='étiquette de la légende')
# puis
plt.legend(loc=2)
plt.title('Ici mon super titre')
plt.savefig('nom_du_fichier.png')
```

Premier outil Complétez la fonction **liste_alea**.

On se lance Compléter la fonction **teste_affectations** puis la tester sous la forme

```
test_affectations(tri_selection, [10, 30, 70])
```

Comprendre ce qui se passe... Puis créer le premier graphique demandé.
Il ne reste plus qu'à remplir notre but.

III Comparaison à d'autres tris

Il se trouve que nos algorithmes de tri sont assez frustres. Ajouter à votre graphique de nombres d'opérations les résultats pour la fonction **trirapide.tri_fusion**.
Retenter l'expérience pour des listes de longueur comprises entre 5 et 100.

Et le temps d'exécution dans tout ça ? Tracer 2 derniers graphiques (pour les listes "grandes", et pour celles qui le sont moins) comparant les temps d'exécution de 4 fonctions : nos 3 tris plus le tri natif de python qui s'obtient par

```
L.sort()  # tri la liste L et ne retourne rien
```

Il faut donc créer une quatrième fonction.