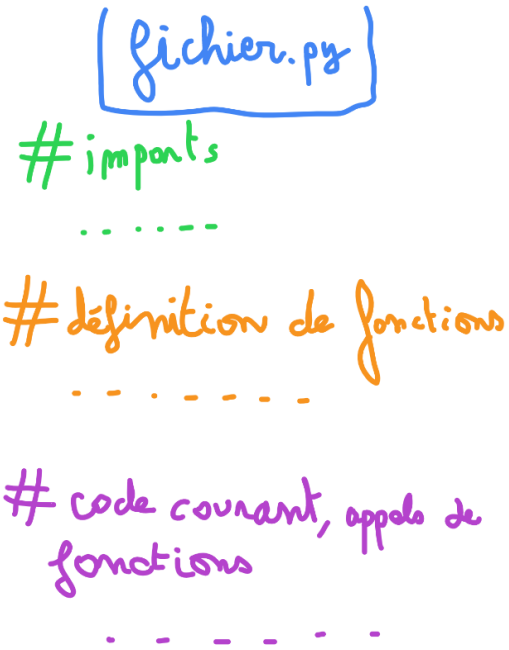
Type de document	Cours et TP	Classe	1re	Durée	2h	Date	20/01/2025
Thème et contenu(s)	Langages et programmation – Algorithmique – Complexité – Boucles – Listes						
Capacités attendues	Comparer deux algorithmes de tri						
Prérequis	Manipulation de listes, boucles non bornées						
Description	Rechercher un élément dans une liste de deux manières différentes						

I) Préambule

Nous nous intéressons ici à la manière la plus efficace de rechercher un élément donné dans une liste **triée**. Nous avons durant les séances précédentes créé de nombreuses fonctions de recherche :

1. Certaines renvoyant True si l'élément était dans la liste, False sinon
2. Certaines renvoyant le nombre d'occurrences de l'élément dans la liste
3. Certaines renvoyant l'indice auquel se trouvait l'élément, -1 sinon



Nous allons pour ce TP recoder une fonction `recherche(element,liste)` qui se comporte comme le cas n°3, puis nous réfléchirons à une manière plus efficace de rechercher un élément dans une liste triée en codant une nouvelle fonction. Enfin nous comparerons les vitesses d'exécution de ces deux fonctions, sur la même liste. N'oubliez pas quand vous codez de respecter l'architecture ci-contre.

II) Créer une liste aléatoirement remplie

On peut créer une liste remplie aléatoirement. Pour ce faire, on peut importer le module `random`.
Cela se fait simplement en écrivant : `import random`
Pour générer un entier entre a et b **inclus**, on peut utiliser `randint` comme suit :
`random.randint(a,b)`
Ainsi, si a vaut 10 et b vaut 20, la ligne ci-dessus génère un nombre aléatoire (en réalité, pseudo-aléatoire) compris entre 10 et 20 inclus.

On veut maintenant créer une liste de 100 éléments compris entre 0 et 100 de manière aléatoire : utilisez ce code pour créer une telle liste (vous pouvez vous aider d'une boucle, tout simplement).

III) Trier la liste

On peut bien sûr chercher un élément dans une liste non triée, mais à ce moment-là il n'y a pas d'autre solution que de parcourir tous les éléments de la liste (l'élément cherché pouvant être, par exemple, en dernier). Le fait de la trier va nous aider à imaginer par la suite un algorithme efficace qui va être en mesure de « savoir » dans quelle partie de la liste chercher.

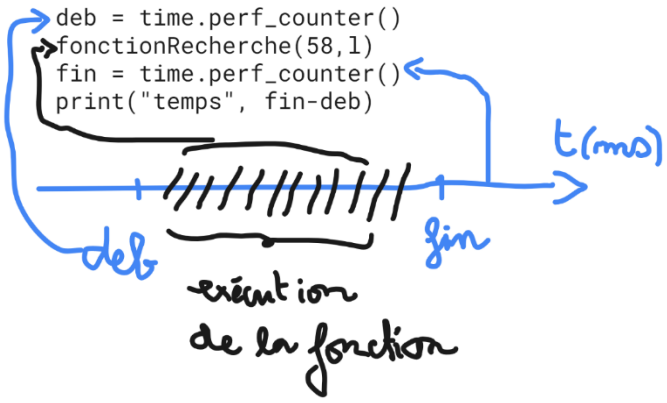
Pour trier la liste, on utilise `l.sort()` où `l` est notre liste : cela classe tous les éléments de la liste par ordre croissant.

IV) Créer une fonction de recherche (déjà vu)

Créer une fonction `recherche(element,liste)` qui renvoie l'indice de l'élément s'il se trouve dans la liste, -1 sinon. Cette fonction doit parcourir la liste élément par élément.

V) Mesurer le temps d'exécution

On va importer la bibliothèque `time`, très utile en Python.
On va ensuite initialiser un timer en utilisant `time.perf_counter()` juste avant d'appeler la fonction, puis appeler la fonction, puis rappeler un timer. La différence entre les deux timers va nous donner le temps d'exécution. Voici un schéma explicatif avec le code associé :



Faire varier la taille de la liste passée en paramètre et mesurer le temps. Effectuer plusieurs mesures. Que remarque-t-on ?

VI) Créer une fonction de recherche dichotomique

Une manière de comprendre la **recherche dichotomique** est de s'imaginer un jeu dans lequel une machine ou un joueur vous fait deviner un nombre entre 0 et 100. Il peut juste vous dire : « trouvé », « plus grand », ou « plus petit ».
Quelle stratégie vous permet de trouver en un minimum de coups ?
Essayez d'appliquer cette stratégie à la création d'un algorithme. La difficulté ici est qu'on ne sait pas si le nombre se trouve dans la liste ou non…

Implémentez maintenant l'algorithme suivant qui recherche l'élément `el` dans `l`:

```
Variable bg = 0
Variable bd = longueur(l) - 1
Tant que bg < bd :
      Variable milieu = (bg+bd)//2
      #ensuite 3 cas
      Si element = l[milieu] : #on a trouvé l'élément !
            retourner milieu
      Sinon si element > l[milieu] : #élément > « milieu », on cherche dans la partie droite
            bg = milieu + 1
      Sinon: #dernier cas, élément < « milieu », on cherche dans la partie gauche
            bd = milieu - 1
retourner -1 #se produira si on n'a jamais eu de return milieu dans le while (l'élément n'y est pas)
```

Comparer ensuite les temps d'exécution des deux fonctions sur la même liste.

Aller plus loin : https://www.codingame.com/ide/puzzle/shadows-of-the-knight-episode-1