TP 02

Thème 1 : Recherche séquentielle

Savoirs et compétences :

- *Th. 1 : Recherche séquentielle dans un tableau unidimensionnel. Dictionnaire.*

Consignes

1. Commencez la séance en créant un dossier au nom du TP dans le répertoire dédié à l'informatique de votre compte.
2. Après la séance, vous devez rédiger un compte-rendu de TP et l'envoyer au format électronique à votre enseignant.
3. Essayer d'être le plus autonome possible.
4. Ce TP est à faire en binôme, vous ne rendrez donc qu'un compte-rendu pour deux. Votre fichier portera un nom du type : `tp01_durif_pessoles.py`, où les noms de vos enseignants sont à remplacer par ceux des membres du binôme. Le nom de ce fichier ne devra comporter ni espace, ni accent, ni apostrophe, ni majuscule. Dans ce fichier, vous respecterez les consignes suivantes.
    - Écrivez d'abord en commentaires (ligne débutant par #), le titre du TP, les noms et prénoms des étudiants du groupe.
    - Commencez chaque question par son numéro écrit en commentaires.
    - Les questions demandant une réponse écrite seront rédigées en commentaires.
    - Les questions demandant une réponse sous forme de fonction ou de script respecteront pointilleusement les noms de variables et de fonctions demandés.

Activité 1 – Exercices d'échauffement

Objectif Rechercher séquentiellement un élément dans un tableau unidimensionnel ou dans un dictionnaire.

Recherche d'un nombre dans une liste

Nous allons commencer par rechercher si un nombre est dans un tableau.
Commençons par définir la liste des entiers pairs compris entre 0 et `nb` exclus.

```python
def generate_pair_01(nb: int) -> list :
    """
    Génération d'une liste de nombres pairs compris entre 0 (exclus) et nb (exclus).
    """
    res = []
    for i in range(1,nb//2):
        res.append(2*i)
    return res
```

Recopier la fonction dans un fichier.

Question 1 *Vérifier que la fonction* `generate_pair_01` *fonctionne pour* `nb=0`, `nb=9`, `nb=10`.

Question 2 *Écrire une fonction de signature* `generate_pair_02(nb: int) -> list` *en utilisant une boucle* `while`.

Question 3 *Écrire une fonction de signature* `recherche_nb_01(nb: int, L: list) -> bool` *qui renvoie* `True` *si* `nb` *est dans* `L`, `False` *sinon. On utilisera une boucle* `for`.

Question 4 *Écrire une fonction de signature* `recherche_nb_02(nb: int, L: list) -> bool` *qui renvoie* `True` *si* `nb` *est dans* `L`, `False` *sinon. On utilisera une boucle* `while`.

Question 5 *Écrire une fonction de signature* `recherche_nb_03(nb: int, L: list) -> bool` *qui renvoie* `True` *si* `nb` *est dans* `L`, `False` *sinon. On n'utilisera pas explicitement de boucles* `for` *ou* `while`.

Question 6 *Écrire une fonction de signature* `recherche_first_index_nb_01(nb: int, L: list) -> int` *qui renvoie l'index de la première appartion du nombre* `nb` *dans la liste* `L`. *La fonction renverra* `-1` *si* `nb` *n'est pas dans la liste. On utilisera une boucle* `for`.

Question 7 *Écrire une fonction de signature* `recherche_first_index_nb_02(nb: int, L: list) -> int` *qui renvoie l'index de la première appartion du nombre* `nb` *dans la liste* `L`. *La fonction renverra* `-1` *si* `nb` *n'est pas dans la liste. On utilisera une boucle* `while`.

Question 8 *Écrire une fonction de signature* `recherche_last_index_nb_01(nb: int, L: list) -> int` *qui renvoie l'index de la dernière appartion du nombre* `nb` *dans la liste* `L`. *La fonction renverra* `-1` *si* `nb` *n'est pas dans la liste. On utilisera une boucle* `for`.

Question 9 *Écrire une fonction de signature* `recherche_last_index_nb_02(nb: int, L: list) -> int` *qui renvoie l'index de la dernière appartion du nombre* `nb` *dans la liste* `L`. *La fonction renverra* `-1` *si* `nb` *n'est pas dans la liste. On utilisera une boucle* `while`.

Question 10 *Écrire une fonction de signature* `recherche_index_nb_01(nb: int, L: list) -> list` *qui renvoie la liste des index du nombre* `nb` *dans la liste* `L`. *La fonction renverra une liste vide si* `nb` *n'est pas dans la liste.*

Recherche d'un caractère dans une chaîne (de caractères)

Question 11 *Écrire une fonction de signature* `is_char_in_str_01(lettre: str, mot: str) -> int` *qui renvoie* `True` *si* `lettre` *est dans* `mot`, `False` *sinon. On utilisera une boucle* `for` *ou* `while`.

Question 12 *Écrire une fonction de signature* `is_char_in_str_02(lettre: str, mot: str) -> int` *qui renvoie* `True` *si* `lettre` *est dans* `mot`, `False` *sinon. On n'utilisera ni boucle* `for` *ni* `while` *explicite.*

Question 13 *Écrire une fonction de signature* `compte_lettre_01(lettre: str, mot: str) -> int` *qui renvoie le nombre d'occurrences de* `lettre` *dans le* `mot`.

Les instructions suivantes permettent de charger l'ensemble des mots du dictionnaire dans la variable `dictionnaire`. `dictionnaire` est une liste de mots. Chacune des lettres de l'alphabet sont stockées dans la variable `alphabet`.

```python
alphabet = 'abcdefghijklmnopqrstuvwxyz'
def load_fichier(file):
    fid = open(file,'r')
    mots = fid.readlines()
    fid.close()
    return mots
liste_mots = load_file('liste_francais.txt')
```

Question 14 *Écrire une fonction de signature* `compte_lettre_02(lettre: str, mots: list) -> int` *qui renvoie le nombre d'occurrences de* `lettre` *dans une liste de mots* `mots`.

Question 15 *Quelle consonne apparaît le plus souvent ? Quelle consonne apparaît le moins souvent ? Indiquer le nombre d'occurrences dans chacun des mots*

Question 16 *Écrire une fonction de signature* `mots_plus_long(mots: list) -> str` *qui renvoie le mot le plus long.*

Recherche dans un dictionnaire

Un dictionnaire (`dict`) est un type composite (au même titre que les chaînes, les listes ou les tuples). Les éléments d'un dictionnaire sont constitués d'une **clé** (alphabétique ou numérique par exemple) et d'une valeur. À la différence d'une liste par exemple, les éléments ne sont pas ordonnés.

Exemple Dans le but de faire un comptage du nombre de lettres des mots du dictionnaire, nous allons créer un dictionnaire constitué des lettres de l'alphabet (clés) et de leur nombre d'apparitions (valeurs).

```
nb_lettres = {}
nb_lettres['a']=0
nb_lettres['b']=0
print(nb_lettres)
    {'a': 0, 'b': 0}
```

```
nb_lettres = {}
for lettre in alphabet :
    nb_lettres[lettre]=0
print(nb_lettres["a"])
    0
```

Tester l'appartenance d'une clé à un dictionnaire : `"a" in nb_lettres`.
Supprimer une clé d'un dictionnaire : `del nb_lettres["a"]`.
Parcourir un dictionnaire :

```
for clef in nb_lettres :
    print(clef)
    print(clef,nb_lettres[clef])

for clef,valeur in nb_lettres.items() :
    print(clef,valeur)
```

■

Question 17 *Écrire une fonction* `init_dictionnaire(chaine:str) -> dict` *que crée, initialise et renvoie le dictionnaire dont les clés sont les lettres de l'alphabet et les valeurs sont initialisées à 0.*

Question 18 *Écrire une fonction* `remplir_dictionnaire(dico:dict, liste_mots:list) -> dict` *qui compte le nombre de lettres de chacun des mots du dictionnaire chargé précédemment en incrémentant chacune des valeurs du dictionnaire.*

Question 19 *Écrire une fonction* `cherche_podium(dico:dict) ->list` *qui renvoie la liste des trois lettres les plus utilisées. La liste renvoyée sera sous la forme* `[['a',5],['b',4],['c',3]]`.

Activité 2 – La conjecture de Syracuse

D'après Jean-Pierre Becirspahic, `https://info-llg.fr/`

On doit cette conjecture au mathématicien allemand Lothar Collatz qui, en 1937, proposa à la communauté mathématique le problème suivant : « on part d'un nombre entier strictement positif; s'il est pair on le divise par 2, s'il est impair on le multiplie par 3 et on ajoute 1. On réitère ensuite cette opération.»

Par exemple, à partir de 14 on construit la suite de nombres :

$$14, 7, 22, 11, 34, 17, 52, 26, 13, 40, 20, 10, 5, 16, 8, 4, 2, 1, 4, 2...$$

Après que le nombre 1 ait été atteint, la suite des valeurs (1, 4, 2, 1, 4, 2 ...) se répète indéfiniment en un cycle de longueur 3.

La conjecture de Syracuse[1] est l'hypothèse mathématique selon laquelle n'importe quel entier de départ conduit à la valeur 1 au bout d'un certain temps.

Nous allons expérimenter cette conjecture en programmant l'évolution de la suite $(u_n)_{n\in\mathbb{N}}$ définie par les relations :

$$u_0 = c \text{ et } \forall n \geq 1, u_{n+1} = \begin{cases} \dfrac{u_n}{2} & \text{si } u_n \text{ est pair} \\ 3u_n + 1 & \text{si } u_n \text{ est impair} \end{cases}.$$

Temps de vol et altitude maximale

Le temps de vol d'un entier c et le plus petit entier n (en admettant qu'il existe) pour lequel $u_n = 1$. Par exemple, le temps de vol pour $c = 14$ est égal à 17.

1. Du nom de l'université américaine qui a popularisé ce problème.

Question 20 *Définir une fonction nommée* `tempsdevol` *prenant un paramètre entier* `c` *et retournant le plus petit entier n pour lequel $u_n = 1$.*

De manière tout aussi imagée, on appelle altitude maximale de c la valeur maximale de la suite $(u_n)_{n\in\mathbb{N}}$. Par exemple, l'altitude maximale de $c = 14$ est égale à 52.

Question 21 *Modifier votre algorithme pour définir une fonction nommée* `altitude` *qui calcule cette fois-ci l'altitude maximale pour un entier c donné en paramètre.*

Vérification expérimentale de la conjecture

On désire désormais vérifier la validité de la conjecture pour toute valeur $c \leq 1\,000\,000$. Une première solution consisterait à calculer le temps de vol pour toutes ces valeurs, mais ce calcul est long et il y a mieux à faire en observant que si la conjecture a déjà été vérifiée pour toute valeur $c' < c$, il suffit qu'il existe un rang n pour lequel $u_n < c$ pour être certain que la conjecture sera aussi vérifiée au rang c.

On appelle temps d'arrêt (ou encore temps de vol en altitude) le premier entier n (s'il existe) pour lequel $u_n < c$.

Question 22 *Écrire une fonction* `tempsdarret` *prenant un paramètre entier c et retournant le temps d'arrêt de la suite de Syracuse correspondante.*

Nous souhaitons maintenant mesurer le temps nécessaire pour vérifier la conjecture jusqu'à un paramètre entier m. Pour cela, nous allons utiliser la fonction `time` du module time du même nom, sans argument, qui retourne le temps en secondes depuis une date de référence (qui dépend du système).

Question 23 *À l'aide de cette fonction écrire une fonction* `verification` *qui prend en argument un paramètre entier m et retourne le temps nécessaire pour vérifier que toutes les valeurs $c \in [\![2, m]\!]$ ont bien un temps d'arrêt fini. Quelle durée d'exécution obtient-t-on pour $m = 1\,000\,000$?*

Question 24 *Quel est le temps d'arrêt d'un entier pair? et d'un entier de la forme $c = 4n+1$? En déduire qu'on peut restreindre la recherche aux entiers de la forme $4n+3$, et modifier en conséquence la fonction précédente. Combien de temps gagne-t-on par rapport à la version précédente pour $m = 1\,000\,000$? Vérifier ensuite la conjecture pour $m = 10\,000\,000$.*

Records

Question 25 *Déterminer l'altitude maximale que l'on peut atteindre lorsque $c \in [\![1, 1\,000\,000]\!]$, ainsi que la valeur minimale de c permettant d'obtenir cette altitude.*

Question 26 *Déterminer le temps de vol en altitude (autrement dit le temps d'arrêt) de durée maximale lorsque $c \in [\![1, 1\,000\,000]\!]$ ainsi que la valeur de c correspondante.*

Question 27 *On appelle vol en altitude de durée record un vol dont tous les temps d'arrêt de rangs inférieurs sont plus courts. Par exemple, le vol réalisé pour $c = 7$ est un vol en altitude de durée record (égale à 11) car tous les vols débutant par $c = 1, 2, 3, 4, 5, 6$ ont des temps d'arrêt de durées inférieures à 11. Déterminez tous les vols en altitude de durée record pour $c \leq 1\,000\,000$.*

Affichage du vol

Pour obtenir des graphes, on utilise la fonction plot qui appartient à un module appelé `matplotlib.pyplot` et dédié au tracé de graphes. Vous allez donc commencer par importer celui-ci à l'aide de la commande : `import matplotlib.pyplot as plt` Désormais toutes les fonctions de ce module vous sont accessibles à condition de les préfixer par `plt`. Nous découvrirons progressivement les nombreuses possibilités qu'offre ce module, mais aujourd'hui nous n'aurons besoin que de deux fonctions : `plt.plot` et `plt.show`. Sous sa forme la plus simple, la fonction `plt.plot` n'exige qu'une liste en paramètre : `plt.plot([a0, a1, ..., an])` crée un graphe constitué d'une ligne brisée reliant les points de coordonnées (k, a_k) pour $k \in [\![0, n]\!]$. Une fois votre graphe créé par la fonction `plt.plot`, il reste à le faire apparaître dans une fenêtre annexe à l'aide de l'instruction `plt.show()`.

Question 28 *Définir une fonction nommée graphique qui prend un entier c en paramètre et qui construit le graphe de la suite $(u_n)_{n\in\mathbb{N}}$ durant son temps de vol.*