Exercices pour l'épreuve pratique de la spécialité NSI. Série 2 (8 à 10)

Exercice 8.1

Écrire une fonction `recherche` qui prend en paramètres `elt` un nombre entier et `tab` un tableau de nombres entiers, et qui renvoie l'indice de la première occurrence de `elt` dans `tab` si `elt` est dans `tab` et `-1` sinon.

Exemples :

```
>>> recherche(1, [2, 3, 4])
-1
>>> recherche(1, [10, 12, 1, 56])
2
>>> recherche(50, [1, 50, 1])
1
>>> recherche(15, [8, 9, 10, 15])
3
```

Exercice 8.2

On considère la fonction `insere` ci-dessous qui prend en argument un entier `a` et un tableau `tab` d'entiers triés par ordre croissant. Cette fonction insère la valeur `a` dans le tableau et renvoie le nouveau tableau. Les tableaux seront représentés sous la forme de listes python.

```
def insere(a, tab):
    l = list(tab) #l contient les mêmes éléments que tab
    l.append(a)
    i = ...
    while a < ... and i >= 0:
      l[i+1] = ...
      l[i] = a
      i = ...
    return l
```

Compléter la fonction `insere` ci-dessus.

Exemples :

```
>>> insere(3,[1,2,4,5])
[1, 2, 3, 4, 5]
>>> insere(10,[1,2,7,12,14,25])
[1, 2, 7, 10, 12, 14, 25]
>>> insere(1,[2,3,4])
[1, 2, 3, 4]
```

Exercice 9.1

Soit un nombre entier supérieur ou égal à 1 :

- s'il est pair, on le divise par 2 ;
- s'il est impair, on le multiplie par 3 et on ajoute 1.

Puis on recommence ces étapes avec le nombre entier obtenu, jusqu'à ce que l'on obtienne la valeur 1.

On définit ainsi la suite (u_n) par

- $u_0 = k$, où k est un entier choisi initialement ;
- $u_{n+1} = u_n / 2$ si u_n est pair ;
- $u_{n+1} = 3 \times u_n + 1$ si u_n est impair.

On admet que, quel que soit l'entier k choisi au départ, la suite finit toujours sur la valeur 1.

Écrire une fonction `calcul` prenant en paramètres un entier n strictement positif et qui renvoie la liste des valeurs u_n , en partant de k et jusqu'à atteindre 1.

Exemple :

```
>>> calcul(7)
[7, 22, 11, 34, 17, 52, 26, 13, 40, 20, 10, 5, 16, 8, 4, 2, 1]
```

Exercice 9.2

On affecte à chaque lettre de l'alphabet un code selon les tableaux ci-dessous :

A	B	C	D	E	F	G	H	I	J	K	L	M
1	2	3	4	5	6	7	8	9	10	11	12	13

N	O	P	Q	R	S	T	U	V	W	X	Y	Z
14	15	16	17	18	19	20	21	22	23	24	25	26

Pour un mot donné, on détermine d'une part son *code alphabétique concaténé*, obtenu par la juxtaposition des codes de chacun de ses caractères, et d'autre part, son *code additionné*, qui est la somme des codes de chacun de ses caractères. Par ailleurs, on dit que ce mot est « *parfait* » si le code additionné divise le code concaténé.

Exemples :

- Pour le mot `"PAUL"`, le code concaténé est la chaîne `1612112`, soit l'entier $1\,612\,112$.

  Son code additionné est l'entier 50 car $16 + 1 + 21 + 12 = 50$.
  50 ne divise pas l'entier $1\,612\,112$; par conséquent, le mot `"PAUL"` n'est pas parfait.

- Pour le mot `"ALAIN"`, le code concaténé est la chaîne `1121914`, soit l'entier $1\,121\,914$.
  Le code additionné est l'entier 37 car $1 + 12 + 1 + 9 + 14 = 37$.
  37 divise l'entier $1\,121\,914$; par conséquent, le mot `"ALAIN"` est parfait.

Compléter la fonction `est_parfait` ci-dessous qui prend comme argument une chaîne de caractères `mot` (en lettres majuscules) et qui renvoie le code alphabétique concaténé, le code additionné de `mot`, ainsi qu'un booléen qui indique si `mot` est parfait ou pas.

```
dico = {"A":1, "B":2, "C":3, "D":4, "E":5, "F":6, "G":7, \
        "H":8, "I":9, "J":10, "K":11, "L":12, "M":13, \
        "N":14, "O":15, "P":16, "Q":17, "R":18, "S":19, \
        "T":20, "U":21,"V":22, "W":23, "X":24, "Y":25, "Z":26}

def est_parfait(mot) :
    #mot est une chaîne de caractères (en lettres majuscules)
    code_c = ""
    code_a = ???
    for c in mot :
        code_c = code_c + ???
        code_a = ???
    code_c = int(code_c)
    if ??? :
        mot_est_parfait = True
    else :
        mot_est_parfait = False
    return [code_a, code_c, mot_est_parfait]
```

Exemples :

```
>>> est_parfait("PAUL")
[50, 1612112, False]
>>> est_parfait("ALAIN")
[37, 1121914, True]
```

Exercice 10.1

L'occurrence d'un caractère dans un phrase est le nombre de fois où ce caractère est présent.

Exemples :

- l'occurrence du caractère 'o' dans 'bonjour' est 2 ;
- l'occurrence du caractère 'b' dans 'Bébé' est 1 ;
- l'occurrence du caractère 'B' dans 'Bébé' est 1 ;
- l'occurrence du caractère ' ' dans 'Hello world !' est 2.

On cherche les occurrences des caractères dans une phrase. On souhaite stocker ces occurrences dans un dictionnaire dont les clefs seraient les caractères de la phrase et les valeurs l'occurrence de ces caractères.

Par exemple : avec la phrase 'Hello world !' le dictionnaire est le suivant :

```
{'H': 1,'e': 1,'l': 3,'o': 2,' ': 2,'w': 1,'r': 1,'d': 1,'!': 1}
```

(l'ordre des clefs n'ayant pas d'importance).

Écrire une fonction `occurence_lettres` avec prenant comme paramètre une variable `phrase` de type `str`. Cette fonction doit renvoyer un dictionnaire de type constitué des occurrences des caractères présents dans la phrase.

Exercice 10.2

La fonction `fusion` prend deux listes `L1, L2` d'entiers triées par ordre croissant et les fusionne en une liste triée `L12` qu'elle renvoie.

Le code Python de la fonction est

```
def fusion(L1,L2):
    n1 = len(L1)
    n2 = len(L2)
    L12 = [0]*(n1+n2)
    i1 = 0
    i2 = 0
    i = 0
    while i1 < n1 and ... :
        if L1[i1] < L2[i2]:
            L12[i] = ...
            i1 = ...
        else:
            L12[i] = L2[i2]

              i2 = ...
          i += 1
      while i1 < n1:
        L12[i] = ...
        i1 = i1 + 1
        i = ...
      while i2 < n2:
        L12[i] = ...
        i2 = i2 + 1
        i = ...
      return L12
```

Compléter le code.

Exemple :

```
>>> fusion([1,6,10],[0,7,8,9])
[0, 1, 6, 7, 8, 9, 10]
```