

Exercices pour l'épreuve pratique de la spécialité NSI. Série (11 à 13)

Exercice 11.1

Écrire une fonction `recherche` qui prend en paramètres un tableau `tab` de nombres entiers triés par ordre croissant et un nombre entier `n`, et qui effectue une recherche dichotomique du nombre entier `n` dans le tableau non vide `tab`.

Cette fonction doit renvoyer un indice correspondant au nombre cherché s'il est dans le tableau, -1 sinon.

Exemples:

```
>>> recherche([2, 3, 4, 5, 6], 5)
3
>>> recherche([2, 3, 4, 6, 7], 5)
-1
```

Exercice 11.2

Le codage de César transforme un message en changeant chaque lettre en la décalant dans l'alphabet.

Par exemple, avec un décalage de 3, le A se transforme en D, le B en E, ..., le X en A, le Y en B et le Z en C. Les autres caractères ('!', ' '?...) ne sont pas codés.

La fonction `position_alphabet` ci-dessous prend en paramètre un caractère `lettre` et renvoie la position de `lettre` dans la chaîne de caractères `ALPHABET` s'il s'y trouve et -1 sinon.

La fonction `cesar` prend en paramètre une chaîne de caractères `message` et un nombre entier `decalage` et renvoie le nouveau message codé avec le codage de César utilisant le décalage `decalage`.

```
ALPHABET='ABCDEFGHIJKLMNOPQRSTUVWXYZ'
```

```
def position_alphabet(lettre):
    return ALPHABET.find(lettre)
```

```
def cesar(message, decalage):
    resultat = ""
    for ... in message :
        if lettre in ALPHABET :
            indice = ( ... )%26
            resultat = resultat + ALPHABET[indice]
        else:
            resultat = ...
    return resultat
```

Compléter la fonction `cesar`.

Exemples :

```
>>> cesar('BONJOUR A TOUS. VIVE LA MATIERE NSI !',4)
'FSRNSYV E XSYW. ZMZI PE QEXMIVI RWM !'
>>> cesar('GTSOTZW F YTZX. ANAJ QF RFYNJWJ SXN !',-5)
'BONJOUR A TOUS. VIVE LA MATIERE NSI !'
```

Exercice 12.1

Programmer la fonction `moyenne` prenant en paramètre un tableau d'entiers `tab` (type `list`) qui renvoie la moyenne de ses éléments si le tableau est non vide et affiche `'erreur'` si le tableau est vide.

Exemples :

```
>>> moyenne([5,3,8])
5.333333333333333
>>> moyenne([1,2,3,4,5,6,7,8,9,10])
5.5
>>> moyenne([])
'erreur'
```

Exercice 12.2

On considère un tableau d'entiers `tab` (type `list` dont les éléments sont des 0 ou des 1). On se propose de trier ce tableau selon l'algorithme suivant : à chaque étape du tri, le tableau est constitué de trois zones consécutives, la première ne contenant que des 0, la seconde n'étant pas triée et la dernière ne contenant que des 1.

Zone de 0	Zone non triée	Zone de 1
-----------	----------------	-----------

Tant que la zone non triée n'est pas réduite à un seul élément, on regarde son premier élément :

- ☐ si cet élément vaut 0, on considère qu'il appartient désormais à la zone ne contenant que des 0 ;
- ☐ si cet élément vaut 1, il est échangé avec le dernier élément de la zone non triée et on considère alors qu'il appartient à la zone ne contenant que des 1.

Dans tous les cas, la longueur de la zone non triée diminue de 1.

Recopier sous Python en la complétant la fonction `tri` suivante :

```
def tri(tab):
    #i est le premier indice de la zone non triee, j le dernier indice.
    #Au debut, la zone non triee est le tableau entier.
    i= ...
    j= ...
    while i != j :
        if tab[i]== 0:
            i= ...
        else :
            valeur = tab[j]
            tab[j] = ...
            ...
            j= ...
    ...
```

Exemple :

```
>>> tri([0,1,0,1,0,1,0,1,0])
[0, 0, 0, 0, 0, 1, 1, 1, 1]
```

Exercice 13.1

On s'intéresse au problème du rendu de monnaie. On suppose qu'on dispose d'un nombre infini de billets de 5 euros, de pièces de 2 euros et de pièces de 1 euro.

Le but est d'écrire une fonction nommée `rendu` dont le paramètre est un entier positif non nul `somme_a_rendre` et qui retourne une liste de trois entiers `n1`, `n2` et `n3` qui correspondent aux nombres de billets de 5 euros (`n1`) de pièces de 2 euros (`n2`) et de pièces de 1 euro (`n3`) à rendre afin que le total rendu soit égal à `somme_a_rendre`.

On utilisera un algorithme glouton : on commencera par rendre le nombre maximal de billets de 5 euros, puis celui des pièces de 2 euros et enfin celui des pièces de 1 euro.

Exemples :

```
>>> rendu(13)
[2,1,1]
>>> rendu(64)
[12,2,0]
>>> rendu(89)
[17,2,0]
```

Exercice 13.2

On veut écrire une classe pour gérer une file à l'aide d'une liste chaînée. On dispose d'une classe `Maillon` permettant la création d'un maillon de la chaîne, celui-ci étant constitué d'une valeur et d'une référence au maillon suivant de la chaîne :

```
class Maillon :
    def __init__(self,v) :
        self.valeur = v
        self.suivant = None
```

Compléter la classe `File` suivante où l'attribut `dernier_file` contient le maillon correspondant à l'élément arrivé en dernier dans la file :

```
class File :
    def __init__(self) :
        self.dernier_file = None

    def enqueue(self,element) :
        nouveau_maillon = Maillon(...)
        nouveau_maillon.suivant = self.dernier_file
        self.dernier_file = ...

    def est_vide(self) :
        return self.dernier_file == None

    def affiche(self) :
        maillon = self.dernier_file
        while maillon != ... :
            print(maillon.valeur)
            maillon = ...
```

```

def defile(self) :
    if not self.est_vide() :
        if self.dernier_file.suivant == None :
            resultat = self.dernier_file.valeur
            self.dernier_file = None
            return resultat
        maillon = ...
        while maillon.suivant.suivant != None :
            maillon = maillon.suivant
        resultat = ...
        maillon.suivant = None
        return resultat
    return None

```

On pourra tester le fonctionnement de la classe en utilisant les commandes suivantes dans la console Python :

```

>>> F = File()
>>> F.est_vide()
True
>>> F.enfile(2)
>>> F.affiche()
2
>>> F.est_vide()
False
>>> F.enfile(5)
>>> F.enfile(7)
>>> F.affiche()
7
5
2
>>> F.defile()
2
>>> F.defile()
5
>>> F.affiche()
7

```