— Sujet 4 —

Exercice 4.1

Écrire une fonction qui prend en paramètre un tableau d'entiers non vide et qui renvoie la moyenne de ces entiers. La fonction est spécifiée ci-après et doit passer les assertions fournies.

```
def moyenne (tab):
    '''
    moyenne(list) -> float
    Entrée : un tableau non vide d'entiers
    Sortie : nombre de type float
    Correspondant à la moyenne des valeurs présentes dans le tableau
    '''
assert moyenne([1]) == 1
assert moyenne([1,2,3,4,5,6,7]) == 4
assert moyenne([1,2]) == 1.5
```

Exercice 4.2

Le but de l'exercice est de compléter une fonction qui détermine si une valeur est présente dans un tableau de valeurs triées dans l'ordre croissant.
L'algorithme traite le cas du tableau vide.
L'algorithme est écrit pour que la recherche dichotomique ne se fasse que dans le cas où la valeur est comprise entre les valeurs extrêmes du tableau.
On distingue les trois cas qui renvoient False en renvoyant False,1 , False,2 et False,3.
Compléter l'algorithme de dichotomie donné ci-après.

```
def dichotomie(tab, x):
    """
    tab : tableau trié dans l'ordre croissant
    x : nombre entier
    La fonction renvoie True si tab contient x et False sinon
    """
    # cas du tableau vide
    if ...:
        return False,1
    # cas où x n'est pas compris entre les valeurs extrêmes
    if (x < tab[0]) or ...:
        return False,2
    debut = 0
    fin = len(tab) - 1
    while debut <= fin:
        m = ...
        if x == tab[m]:
            return ...
        if x > tab[m]:
            debut = m + 1
        else:
            fin = ...
    return ...
```

Exemples :

```
>>> dichotomie([15, 16, 18, 19, 23, 24, 28, 29, 31, 33],28)
True
>>> dichotomie([15, 16, 18, 19, 23, 24, 28, 29, 31, 33],27)
(False, 3)
>>> dichotomie([15, 16, 18, 19, 23, 24, 28, 29, 31, 33],1)
(False, 2)
>>> dichotomie([],28)
(False, 1)
```

— Sujet 6 —

Exercice 6.1

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 euros.

Exemples :

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

Exercice 6.2

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 enfile(self,element) :
        nouveau_maillon = Maillon(...)
        nouveau_maillon.suivant = ...
        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
```

— Sujet 15 —

Exercice 15.1

Écrire une fonction `rechercheMinMax` qui prend en paramètre un tableau de nombres non triés tab, et qui renvoie la plus petite et la plus grande valeur du tableau sous la forme d'un dictionnaire à deux clés 'min' et 'max'. Les tableaux seront représentés sous forme de liste Python.

Exemples :

```
>>> tableau = [0, 1, 4, 2, -2, 9, 3, 1, 7, 1]
>>> resultat = rechercheMinMax(tableau)
>>> resultat
{'min': -2, 'max': 9}
>>> tableau = []
>>> resultat = rechercheMinMax(tableau)
>>> resultat
{'min': None, 'max': None}
```

Exercice 15.2

On dispose d'un programme permettant de créer un objet de type PaquetDeCarte, selon les éléments indiqués dans le code ci-dessous.
Compléter ce code aux endroits indiqués par #A compléter, puis ajouter des assertions dans l'initialiseur de Carte, ainsi que dans la méthode getCarteAt().

```
class Carte:
    """Initialise Couleur (entre 1 à 4), et Valeur (entre 1 à 13)"""
    def __init__(self, c, v):
        self.Couleur = c
        self.Valeur = v
    """Renvoie le nom de la Carte As, 2, ... 10, Valet, Dame, Roi"""
    def getNom(self):
        if ( self.Valeur > 1 and self.Valeur < 11):
            return str(self.Valeur)
        elif self.Valeur == 11:
            return "Valet"
        elif self.Valeur == 12:
            return "Dame"
        elif self.Valeur == 13:
            return "Roi"
        else:
            return "As"
    """Renvoie la couleur de la Carte (parmi pique, coeur, carreau, trefle"""
    def getCouleur(self):
        return ['pique', 'coeur', 'carreau', 'trefle'][self.Couleur - 1]

class PaquetDeCarte:
    def __init__(self):
        self.contenu = []
    """Remplit le paquet de cartes"""
    def remplir(self):
        #A compléter
    """Renvoie la Carte qui se trouve à la position donnée"""
    def getCarteAt(self, pos):
        #A compléter
```

Exemple :

```
>>> unPaquet = PaquetDeCarte()
>>> unPaquet.remplir()
>>> uneCarte = unPaquet.getCarteAt(20)
>>> print(uneCarte.getNom() + " de " + uneCarte.getCouleur())
6 de coeur
```