Exercices pour l'épreuve pratique de la spécialité NSI. Série 1 +

Exercice 2.1

Soit le couple `(note,coefficient)` :

- `note` est un nombre de type flottant (*`float`) compris entre 0 et 20 ;
- coefficient est un nombre entier positif.

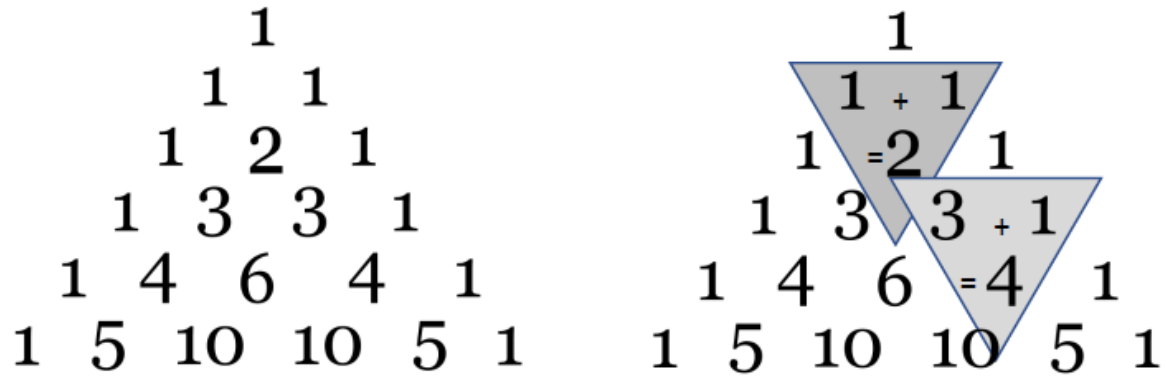
Les résultats aux évaluations d'un élève sont regroupés dans une liste composée de couples `(note,coefficient)`.
Écrire une fonction `moyenne` qui renvoie la moyenne pondérée de cette liste donnée en paramètre.
Par exemple, l'expression `moyenne([(15,2),(9,1),(12,3)])` devra renvoyer le résultat du calcul suivant : $\frac{2\times15+1\times9+3\times12}{2+1+3} = 12{,}5$

```
def moyenne(liste):
    somme = 0
    coefficients = 0
    for couple in liste:
        somme = somme + couple[0]*couple[1]
        coefficients = coefficients + couple[1]
    return somme/coefficients

print(moyenne([(15,2),(9,1),(12,3)]))              # 12,5
```

Exercice 2.2

On cherche à déterminer les valeurs du triangle de Pascal. Dans ce tableau de forme triangulaire, chaque ligne commence et se termine par le nombre 1. Par ailleurs, la valeur qui occupe une case située à l'intérieur du tableau s'obtient en ajoutant les valeurs des deux cases situées juste au-dessus, comme l'indique la figure suivante :



Compléter la fonction `pascal` ci-après. Elle doit renvoyer une liste correspondant au triangle de Pascal de la ligne 1 à la ligne n où `n` est un nombre entier supérieur ou égal à 2 (le tableau sera contenu dans la variable `C`). La variable `Ck` doit, quant à elle, contenir, à l'étape numéro k, la k-ième ligne du tableau.

```
def pascal(n):
    C= [[1]]
    for k in range(1,...):
        Ck = [...]
        for i in range(1,k):
            Ck.append(C[...][i-1]+C[...][...] )
        Ck.append(...)
        C.append(Ck)
    return C
```

Pour n = 4, voici ce que l'on devra obtenir :

```
>> pascal(4)
[[1], [1, 1], [1, 2, 1], [1, 3, 3, 1], [1, 4, 6, 4, 1]]
```

Et pour n = 5, voici ce que l'on devra obtenir :

```
>> pascal(5)
[[1], [1, 1], [1, 2, 1], [1, 3, 3, 1], [1, 4, 6, 4, 1], [1, 5, 10, 10, 5,
1]]
```

```python
def pascal(n):
    C= [[1]]
    for k in range(1,n+1):
        Ck = [1]
        for i in range(1,k):
            Ck.append(C[k-1][i-1]+C[k-1][i] )
        Ck.append(1)
        C.append(Ck)
    return C


print(pascal(4))       # [[1], [1, 1], [1, 2, 1], [1, 3, 3, 1], [1, 4, 6, 4, 1]]
print(pascal(5))       # [[1], [1, 1], [1, 2, 1], [1, 3, 3, 1], [1, 4, 6, 4, 1], [1, 5, 10, 10, 5, 1]]
```

Exercice 4.2

Soit une image binaire représentée dans un tableau à 2 dimensions. Les éléments `M[i][j]`, appelés pixels, sont égaux soit à `0` soit à `1`.

Une composante d'une image est un sous-ensemble de l'image constitué uniquement de `1` et de `0` qui sont côte à côte, soit horizontalement soit verticalement.

Par exemple, les composantes de

M =

0	0	1	0
0	1	0	1
1	1	1	0
0	1	1	0

sont

M =

0	0	1	0
0	1	0	1
1	1	1	0
0	1	1	0

On souhaite, à partir d'un pixel égal à 1 dans une image M, donner la valeur `val` à tous les pixels de la composante à laquelle appartient ce pixel.

La fonction `propager` prend pour paramètre une image `M`, deux entiers `i` et `j` et une valeur entière `val`. Elle met à la valeur `val` tous les pixels de la composante du pixel `M[i][j]` s'il vaut `1` et ne fait rien s'il vaut `0`.

Par exemple, `propager(M,2,1,3)` donne

M =

0	0	1	0
0	3	0	1
3	**3**	3	0
0	3	3	0

Compléter le code récursif de la fonction `propager` donné ci-dessous

```
def propager(M, i, j, val):
    if M[i][j]== ...:
        return

    M[i][j]=val

    # l'élément en haut fait partie de la composante
    if ((i-1) >= 0 and M[i-1][j] == ...):
        propager(M, i-1, j, val)

    # l'élément en bas fait partie de la composante
    if ((...) < len(M) and M[i+1][j] == 1):
        propager(M, ..., j, val)

    # l'élément à gauche fait partie de la composante
    if ((...) >= 0 and M[i][j-1] == 1):
        propager(M, i, ..., val)

    # l'élément à droite fait partie de la composante
    if ((...) < len(M) and M[i][j+1] == 1):
        propager(M, i, ..., val)
```

Exemple :

```
>>> M = [[0,0,1,0],[0,1,0,1],[1,1,1,0],[0,1,1,0]]
>>> propager(M,2,1,3)
>>> M
[[0, 0, 1, 0], [0, 3, 0, 1], [3, 3, 3, 0], [0, 3, 3, 0]]
```

```
def propager(M, i, j, val):
    if M[i][j]== 0:
        return

    M[i][j]=val

    # l'élément en haut fait partie de la composante
    if ((i-1) >= 0 and M[i-1][j] == 1):
        propager(M, i-1, j, val)

    # l'élément en bas fait partie de la composante
    if ((i+1) < len(M) and M[i+1][j] == 1):
        propager(M, i+1, j, val)

    # l'élément à gauche fait partie de la composante
    if ((j-1) >= 0 and M[i][j-1] == 1):
        propager(M, i, j-1, val)

    # l'élément à droite fait partie de la composante
    if ((j+1) < len(M) and M[i][j+1] == 1):
        propager(M, i, j+1, val)


M = [[0,0,1,0],[0,1,0,1],[1,1,1,0],[0,1,1,0]]
propager(M,2,1,3)
print(M)          # [[0, 0, 1, 0], [0, 3, 0, 1], [3, 3, 3, 0], [0, 3, 3, 0]]
```

Exercice 5.2

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
```

Exemple :

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

6 de coeur
```

```python
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):
        for coul in range(1,5):
            for val in range(1,14):
                self.contenu.append(Carte(coul,val))

    """Renvoie la Carte qui se trouve à la position donnée"""
    def getCarteAt(self, pos):
        return self.contenu[pos-2]


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