# 4.1 Récursivité

## Exercice 1 (6 points) - Polynésie J1 - 2024

Cet exercice porte sur la récursivité.
Au rugby, une équipe peut marquer, pour simplifier :

- soit 3 points (pénalité) ;
- soit 5 points (essai non transformé) ;
- soit 7 points (essai transformé).

### Partie A

On souhaite savoir s'il est possible d'obtenir un score donné avec uniquement des pénalités, c'est-à-dire avec une succession de « coups » à 3 points.

1. Écrire une fonction `possible_avec_penalites_seules` qui prend en paramètre un score et qui renvoie `True` si le score passé en paramètre peut être marqué uniquement avec des pénalités.
   Exemple:

```
>>> possible_avec_penalites_seules(15)
True
>>> possible_avec_penalites_seules(10)
False
```

2. Recopier et compléter le tableau suivant qui précise, pour les scores de 0 à 10, les évolutions du score menant à un total donné et le nombre de façons différentes d'obtenir ce total.
   Par exemple, pour obtenir un score de 8, en partant de 0, il y a 2 possibilités :
   — Soit l'équipe marque un essai non transformé, atteignant 5 points, puis une pénalité, atteignant 8 points ;
   — Soit l'équipe marque une pénalité, atteignant 3 points, puis un essai non transformé, atteignant 8 points.

| Score | Liste des solutions | Nombre de solutions |
|---|---|---|
| 0 | [0] | 1 |
| 1 | [] | 0 |
| 2 | | |
| 3 | | |
| 4 | | |
| 5 | | |
| 6 | | |
| 7 | | |
| 8 | [0, 5, 8], [0, 3, 8] | 2 |
| 9 | | 1 |
| 10 | [0,3,10] | 3 |

   En notant $f(n)$ le nombre de possibilités d'obtenir le score $n$, on admet que pour $n > 6$, on a la relation suivante :

$$f(n) = f(n-3) + f(n-5) + f(n-7)$$

3. Vérifier cette relation pour $n = 10$ à l'aide du tableau établi à la question 2.

   On veut écrire une fonction récursive `nb_solutions`, qui prend en paramètre un entier positif quelconque correspondant à un score, et qui renvoie le nombre de façons d'obtenir ce score donné.

4. Déterminer tous les cas de base, pour chaque entier $n$ de 0 à 6, de cette fonction récursive.
5. Écrire la fonction récursive `nb_solutions`, qui prend en paramètre un entier positif quelconque correspondant à un score, et qui renvoie le nombre de possibilités d'obtenir ce score donné.

6. Lors de l'appel de `nb_solutions(score)`, on se rend compte que le nombre d'appels récursifs augmente très rapidement lorsque `score` augmente. Nommer une méthode algorithmique optimisant le nombre d'appels récursifs. (On verra cette question plus tard dans l'année)

   On veut écrire une fonction récursive `solutions_possibles`, qui prend en paramètre un entier positif quelconque correspondant à un score, et qui renvoie la liste composée de toutes les listes représentant les possibilités d'obtenir ce score.

   Par exemple :

```
>>> solutions_possibles(8)
[[0, 5, 8], [0, 3, 8]]
```

7. Déterminer quelles lignes du tableau permettent de construire rapidement la liste renvoyée par l'appel `solutions_possibl`
8. Recopier et compléter la fonction `solutions_possibles` suivante, qui prend en paramètre un score et qui renvoie la liste des possibilités d'obtenir ce score :

```
def solutions_possibles(score):
    if score < 0:
        resultat = []
    elif score == 0:
        resultat = ...
    else:
        resultat = []
        for coup in [..., 5, ...]:
            liste = solutions_possibles(score - coup)
                for solution in liste:
                    solution.append(...)
                resultat.append(...)
    return resultat
```

   Le code à trou tel qu'il est proposé ne peut pas fonctionner

```
def solutions_possibles(score):
    if score < 0:
        resultat = []
    elif score == 0:
        resultat = [[0]]
    else:
        resultat = []
        for coup in [3, 5, 7]:
            liste = solutions_possibles(score - coup)
            solution = []
            for solution in liste:
                solution.append(score)
            if solution != []:
                resultat.append(solution)
    return resultat
```

## Exercice 2 (6 points) - Centres-Étrangers J2 - 2024

Cet exercice porte sur la programmation en Python en général, la programmation orientée objet et la récursivité.
On se déplace dans une grille rectangulaire. On s'intéresse aux chemins dont le départ est sur la case en haut à gauche et l'arrivée en bas à droite. Les seuls déplacements autorisés sont composés de déplacements élémentaires d'une case vers le bas ou d'une case vers la droite.
Un itinéraire est noté sous la forme d'une suite de lettres :

— D pour un déplacement vers la droite d'une case ;

— B pour un déplacement vers le bas d'une case.

Le nombre de caractères D est la longueur de l'itinéraire, et le nombre de caractères B est sa largeur. Par exemple, l'itinéraire `'DDBDBBDDDDB'` a pour longueur 7 et pour largeur 4. Sa représentation graphique est :

```
S * *
* *
*
* * * * *
E
```

- S représente la case de départ (start). Ses coordonnées sont (0, 0) ;
- * représente les cases visitées ;
- E représente la case d'arrivée (end).

**Partie A – Programmation orientée objet**

On représente un itinéraire avec la classe `Chemin` suivante :

```
class Chemin:
    def __init__(self, itineraire):
        self.itineraire = itineraire
        longueur, largeur = 0, 0
        for direction in self.itineraire:
            if direction == "D":
                longueur += 1
            if direction == "B":
                largeur += 1
        self.longueur = longueur
        self.largeur = largeur
        self.grille = [['.' for i in range(longueur+1)] for j in range(largeur+1)]

def remplir_grille(self):
    i, j = 0, 0  # Position initiale
    self.grille[0][0] = 'S'  # Case de départ marquee d'un S
    for direction in self.itineraire:
        if direction == 'D':
            j += 1  # Deplacement vers la droite
        elif direction == 'B':
            i += 1  # Déplacement vers le bas
        self.grille[i][j] = '*'  # Marquer le chemin avec '*'
    self.grille[self.largeur][self.longueur] = 'E'  # Case d'arrivée marquée d'un E
```

1. Donner un attribut et une méthode de la classe `Chemin`.
2. On exécute le code suivant dans la console Python :

```
chemin_1 = Chemin("DDBDBBDDDDB")
a = chemin_1.largeur
b = chemin_1.longueur
```

   Préciser les valeurs contenues dans chacune des variables `a` et `b`.
3. Recopier et compléter la méthode `remplir_grille` qui remplace les `'.'` par des `'*'` pour signifier que le déplacement est passé par cette cellule du tableau.
4. Écrire une méthode `get_dimensions` de la classe `Chemin` qui renvoie la longueur et la largeur de l'itinéraire sous la forme d'un tuple.
5. Écrire une méthode `tracer_chemin` de la classe `Chemin` qui affiche une représentation graphique d'un itinéraire.

**Partie B – Génération aléatoire d'itinéraires**

On souhaite créer des chemins de façon aléatoire. Pour cela, on utilise la méthode `choice` de la bibliothèque `random` dont voici la documentation :

```
random.choice(sequence: list)
Renvoie un élément choisi dans une liste non vide.
Si la population est vide, lève IndexError.
```

L'algorithme proposé est le suivant :

- on initialise :
  - une variable `itineraire` comme une chaîne de caractères vide,
  - les variables `i` et `j` à 0 ;
- tant que l'on n'est pas sur la dernière ligne ou la dernière colonne du tableau :
  - on tire au sort entre un déplacement à droite ou en bas,
  - le déplacement est concaténé à la chaîne de caractères `itineraire`,
  - si le déplacement est vers la droite, alors `j` est incrémenté de 1,
  - si le déplacement est vers le bas, alors `i` est incrémenté de 1 ;
- il reste à terminer le chemin en complétant par des déplacements afin d'atteindre la cellule en bas à droite.

6. Écrire les lignes manquantes dans le code ci-dessous. Le nombre de lignes effacées dans le code n'est pas indicatif.

```
from random import choice

def itineraire_aleatoire(m, n):
    itineraire = ''
    i, j = 0, 0
    while i != m and j != n:
        ... # il y a plusieurs lignes
    if i == m:
        itineraire = itineraire + 'D'*(n-j)
    if j == n:
        itineraire = itineraire + 'B'*(m-i)
    return itineraire
```

### Partie C – Calcul du nombre de chemins possibles

Soient $m$ et $n$ deux entiers naturels non nuls. On se place dans le contexte d'un itinéraire de longueur $m$ et de largeur $n$ de dimension $m \times n$.
On note $N(m, n)$ le nombre de chemins distincts respectant les contraintes de l'exercice.

7. Pour un itinéraire de dimension $1 \times n$, justifier, éventuellement à l'aide d'un exemple, qu'il y a un seul chemin, c'est-à-dire que, quel que soit $n$, on a $N(1, n) = 1$.

   De même, $N(m, 1) = 1$.
8. Justifier que $N(m, n) = N(m-1, n) + N(m, n-1)$.
9. En utilisant les questions précédentes, écrire une fonction récursive `nombre_chemins(m, n)` qui renvoie le nombre de chemins possibles dans une grille rectangulaire de dimension $m \times n$.