ARBRES ET RECURSIVITE – DEVOIR

EXERCICE 1 : UNE PUNITION

On veut créer une fonction `punition` qui écrit `n` fois un certain `texte`.

- `n` est un entier positif
- `texte` est une chaine de caractères

L´appel de cette fonction pour `n=5` doit donner en console :

```
>>> punition(5, "Je dois suivre en classe.")
Je dois suivre en classe.
Je dois suivre en classe.
Je dois suivre en classe.
Je dois suivre en classe.
Je dois suivre en classe.
```

1) Écrire une version itérative de cette fonction
2) Écrire une version récursive de cette fonction

EXERCICE 2 : SOMME

On veut créer une fonction `somme` qui **renvoie** la somme des entiers de `1` à `n` inclus. `n` est un entier.

- Exemple : 1+2+3+4 =10, donc `somme(4)` renvoie `10`.

1) Écrire une version itérative de cette fonction
2) Écrire une version récursive de cette fonction

EXERCICE 3 : COMPTE A REBOURS

On veut créer une fonction `compte` qui écrit le compte à rebours en commençant par `n`.

- L´appel de cette fonction pour `n=5` doit donner en console :

```
>>> compte(5)
5
4
3
2
1
```

1) Écrire une version itérative de cette fonction
2) Écrire une version récursive de cette fonction

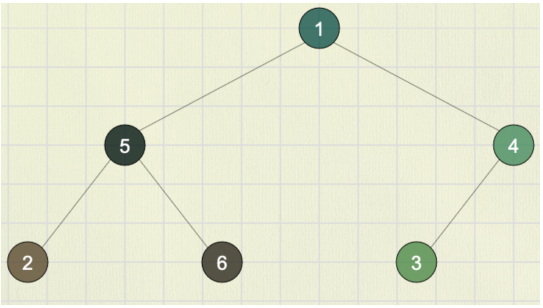
EXERCICE 4 : ARBRES BINAIRE

L´implémentation d´un `arbre binaire` est réalisée à l´aide de la classe `Node` :

```
class Node():
    def __init__(self, val):
        self.val = val
        self.gauche = None
        self.droit = None

    def __repr__(self):
        if self==None:
            return
        return "["+str(self.val)+","+str(self.gauche)+","+str(self.droit)+"]"
```

On souhaite créer l´arbre suivant :



Le code commence par :

```
mon_arbre = Node(1)
mon_arbre.gauche = Node(5)
```

1) Écrire la suite des instructions à donner pour créer cet arbre.

2) Que va afficher la commande `print(mon_arbre)` ?

3) Compléter la fonction récursive `compte_noeud` qui compte le nombre de nœuds d´un arbre binaire.

```
def compte_noeuds(arbre):
    if arbre == None:
        return 0
    else:
        return ..........
```

4) Que calcule cette fonction ? Expliquez votre réponse.
La fonction `max (a, b)` renvoie la plus grande valeur entre `a` et `b`.

```
# code
def fonction(arbre):
    if arbre == None:
        return 0
    else:
        return 1+ max(fonction(arbre.gauche),fonction(arbre.droit))
```