ACADÉMIE
DE REIMS
Liberté
Égalité
Fraternité

NUMERIQUE
et
SCIENCES
INFORMATIQUES

Épreuve de l'enseignement de spécialité

Sujet d'entraînement

Partie Pratique

Classe Terminale de la voie générale

Le candidat doit traiter les 2 exercices

Le sujet comporte 4 pages

Exercice 1 (4 points)

Une de vos amies vient de faire deux parties du jeu Poche_démone. Lors de sa première partie :

- dans le chateau, elle a affronté 20 fois le personnage 'Sir_chapeau' et elle a été victorieuse 12 fois,
- dans la forêt, elle a affronté 16 fois le personnage 'Hetre_dit_mot' et elle a gagné 13 fois,
- dans la grotte, elle a affronté 12 fois le personnage 'Ratopnu', dont 2 victoires, et 9 fois le personnage 'Trop_dit_glau', dont 3 victoires.

Lors de sa seconde partie :

- dans le chateau, elle a affronté 6 fois le personnage 'Sir_chapeau', elle a été victorieuse 3 fois, et 3 fois le personnage 'Elfe_lest', dont 1 seule victoire,
- dans la forêt, elle a affronté 7 fois le personnage 'Hetre_dit_mot' et elle a gagné 6 fois,
- dans la grotte, elle a affronté 11 fois le personnage 'Ratopnu', dont 6 victoires, et 14 fois le personnage 'Trop_dit_glau', dont 5 victoires.

On représente les résultats de chaque partie sous forme d'un dictionnaire :

Exemples :

```
partie1 = {
           'Sir_chapeau': ('chateau', 20, 12),
           'Hetre_dit_mot': ('forêt', 16, 13),
           'Ratopnu': ('grotte', 12, 2),
           'Trop_dit_glau': ('grotte', 9, 3)
           }
partie2 = {
           'Sir_chapeau': ('chateau', 6, 3),
           'Hetre_dit_mot': ('forêt', 7, 6),
           'Ratopnu': ('grotte', 11, 6),
           'Trop_dit_glau': ('grotte', 14, 5),
           'Elfe_lest': ('chateau', 3, 1)
           }
```

`partie1` est le dictionnaire contenant les données de la première partie, le second dictionnaire `partie2` résume la seconde partie.

Proposer une fonction `meilleur_adversaire` qui prend en paramètre un dictionnaire contenant les données d'une partie et qui renvoie un tuple contenant le nom de l'adversaire contre lequel votre amie a le plus perdu ainsi que le nombre de défaites obtenues.

```
>>>meilleur_adversaire(partie1)
    ('Ratopnu',10)
>>>meilleur_adversaire(partie2)
    ('Trop_dit_glau',9)
```

Exercice 2 (4 points)

Le but de cet exercice est d'implémenter un arbre binaire en programmation objet.

Vous devez pour cela compléter la classe `Noeud` ci-dessous en respectant les contraintes suivantes :

Le construction `__init__` est tel qu'un objet `Noeud` qui aura 3 attributs :

- L'attribut `valeur` contiendra la valeur (ou étiquette) associée au nœud.
- L'attribut `gauche` contiendra le sous-arbre gauche.
- L'attribut `droit` seront le sous-arbre droit.
  Si il n'y a pas de sous-arbre gauche ou droit, on indiquera la valeur `None` dans les attributs correspondants.

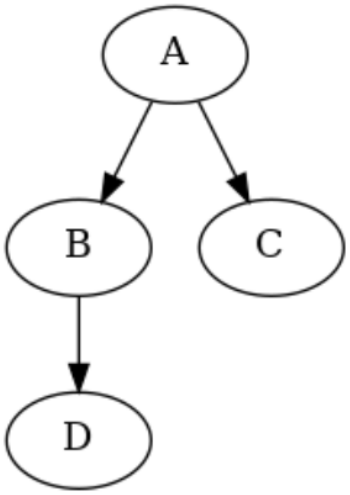
Dans la classe `Noeud`, vous devez compléter les trois méthodes suivante :

- La méthode `est_feuille()` renverra un booléen selon que l'objet est une feuille de l'arbre ou non.
- La méthode `cree_fils_gauche()` prend en paramètre une valeur et crée une feuille à gauche dont la valeur est passée en paramètres. De plus, elle renvoie le nœud fils.

- La méthode `cree_fils_droit()` est construite sur le même modèle que `cree_fils_gauche()`. De même, elle renvoie le nœud fils.

Exemple d'utilisation de la classe `Noeud` :

On considère l'arbre binaire suivant :



En supposant la classe `Noeud` créée, voici comment l'arbre ci-dessus peut être implémenté :

```
>>>arbre = Noeud("A")
>>>sous_arbre_gauche = arbre.cree_fils_gauche("B")
>>>sous_arbre_gauche.cree_fils_gauche("D")
>>>arbre.cree_fils_droit("C")
```

Quelques vérifications possibles :

```
>>>arbre.est_feuille())
    False
>>>arbre.droit.est_feuille())
    True
>>>arbre.gauche.valeur
    "B"
```

Compléter le programme ci-dessous :

```
class Noeud():
    """
    Implémentation d'un arbre binaire
    """

    def __init__(self,valeur):
        """
        Constructeur :
        valeur est une chaîne de caractères ou un nombre entier.
        """
        self.valeur =    # À compléter
        self.gauche =    # À compléter
        self.droit =     # À compléter

    def cree_fils_gauche(self,fils_gauche):
        """
        fils_gauche est une chaîne de caractères ou un nombre entier qui sera rajouté comme fils gauche
        au noeud de l'arbre binaire sur lequel s'applique cette méthode.
        La valeur de ce fils rajouté est renvoyée par la méthode.
        """
        self.gauche =     # À compléter
        return self.gauche

    def cree_fils_droit(self,fils_droit):
        self.droit =     # À compléter
        return self.droit

    def est_feuille(self):
        """
        méthode renvoyant True si le noeud sur lequel s'applique la méthode est une feuille de l'arbre
        et False sinon.
        """
        # À compléter
```