TP PROGRAMMATION ORIENTEE OBJET POO

Objectifs

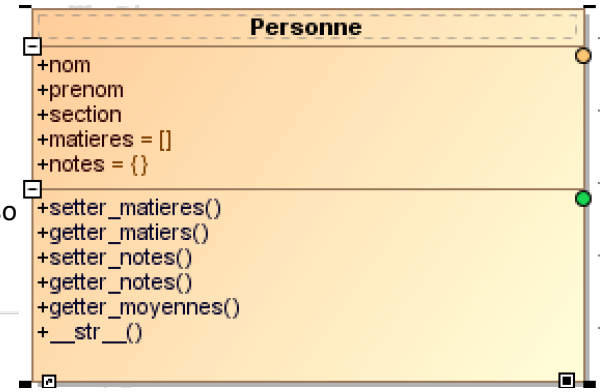
- Créer des objets avec leur attributs et leurs méthodes.
- Utiliser des objets
- Réviser les dictionnaires

On désire créer des objets qui modélise des « Personne » puis des « Eleve » avec le formalisme de la POO dans le but de pouvoir exploiter les informations des différents objets.

On obtiendra des Objet « Eleve » qui contiendront les information suivantes :

GROS Louis tg2 ['maths', 'physique', None, None, None] {'maths': [14.0, 18.0], 'physique': [12.5]} EXT

1. Création d'un Object « Personne »



- Compléter le code de la représentation de l'objet « Perso

```
class Personne() :
    # # attribut de classe
    nb_personnes = 0
    def __init__(self, nom, prenom,section,matieres = [],notes = {}):
        self.nom = ?
        self.prenom = ?
        self.section = ?
        self.matieres = ?
        self.notes = ?
        self.__class__.nb_personnes+=1 # modif  attribut de classe
```

- Créer un premier objet « Personne »

```
Personne1 = Personne("GROS","Louis","tg1")
```

- Accès aux valeurs des attributs de l'objet

En python les attributs de l'objets ne sont pas protégés. On peut y accéder librement dans le programme principal. Ce qui n'est pas le cas d'autres langages comme le C++. Dans le shell expérimenter :

```
>>> Personne1.nom
'GROS'
```

Faites la même chose pour les autres attributs. Vérifier que l'on peut aussi modifier la valeur des attributs.

- Exploitation de l'attribut de classe

Créer une autre Personne et faites afficher la valeur de l'attribut de classe

En règle générale on évite d'accéder directement aux valeurs des attributs. On code des méthodes SET (setter) pour imposer des valeurs et des méthode GET (getter) pour récupérer les valeurs des attributs. Ces méthodes s'appellent **des accesseurs**. Ces méthodes permettent d'accéder aux valeurs des attributs de manière sécurisé.

2. Codage des différentes méthodes

2.1. Création de la méthode __str__

Ajouter la méthode _str_ qui retourne l'affiche de tous les attributs de l'objet « Personne ».

```
16      def __str__(self):
17          return(f" ? ? ? ? ? ")
```

Faites afficher les attributs de Personne1 grâce à la méthode __str__.

2.2. Création de la méthode set_matières

Cette méthode permet à l'utilisateur de valider dans l'attribut « matieres » les différentes spécialités choisie par la « Personne » toujours dans le même ordre et sous la forme d'une liste. La liste comporte l'objet vide « None » quand la matière n'est pas choisie.

Dans l'exemple : ['maths', 'physique', None, None, None]

Compléter le code

```
    def set_matieres(self):
        """
        Un dictionnaire de matières constitue les notes
        Permet le choix des spécialités
        Initialise le dictionnaire des notes pour les spécialités vides

        """
        """
        choix des spécialités
        """
        print(f"Choix des matières pour ",  ? affiche le nom ?," ", ? affiche le prénom ? )
        specialites = ["maths","physique","NSI","SVT","HLP"]
        for elt in specialites :
            print(" Notes en ",elt)
            reponse = input("repondre oui / non ")
            if reponse  == "oui" :
                ???? ajoute la specialité  à la liste de l'attribut matières
            else:
                ???? ajoute l'ojet vide None  à la liste de l'attribut matières

        """
        Initialise les dictionnaires des notes avec des listes vides
        """

        for elt in self.matieres :
            if elt != None :
                ????  = []
```

Vérifier le bon fonctionnement de la méthode avec l'outil environnement présent dans pyzo.

2.3. Création de la méthode set_matières

Créer la méthode « get_matière_ qui retourne la liste des matières. Compléter le code.

```
def get_matieres(self):
    print("les spécialité choisies par ", ? affiche le nom ?" ", §? affiche le prenom ? ," sont :",end = " ")
    for elt in ????  :
        if elt != None:
            print(elt," ",end = " ")

    print()
    return ????
```

Vérifier le bon fonctionnement de la méthode .

2.4. Création de la méthode set_notes

La méthode set_note complète les listes des dictionnaire associés au matière étudiées.

Dans l'exemple {'maths': [14.0, 18.0], 'physique': [12.5]}

```
58      def set_notes(self):
59          for cle in ????????? :
60              ajout = "oui"
61              while ajout == "oui" :
62                  print("Voulez vous ajouter une note en ", cle)
63                  ajout = input( " oui /non ")
64                  if ajout  == "oui" :
65                      note = float(input("entrer votre note "))
66                      ??????????
```

2.5. Création de la méthode get_notes

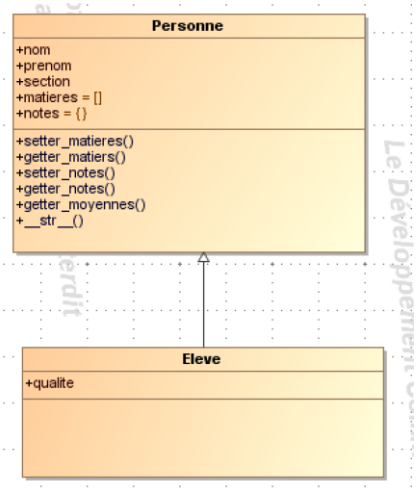
Coder la fonction qui retourne le dictionnaire des notes.

Dans l'exemple {'maths': [14.0, 18.0], 'physique': [12.5]}

2.6. Création de la méthode get_moyenne qui calcul la moyenne des notes pour chaque matière

3. Héritage

A l'objet « Personne » de base on crée une spécialité « Eleve ». Un « Eleve » est une « Personne ». cette affirmation j'ustifie la possibilité de créer une objet enfant qui hérite des caractéristiques de l'objet mère.



Ici « Elve » est une spécialisation de « Personne ». Il possède un attribut en plus qualité qui sera demi-pensionnaire « DP » ou externe « EX »

Compléter le code

```
class Eleve(Personne) :

    def __init__(self,nom,prenom,section,matieres,notes,qualite = ""):
        super().__init__(?,?,?,?,?)
        self.qualite  = ?
    def oui(self) :
        print("oui")

    def regime(self):
        print(? affiche le nom ?," ",? affiche le prenom ?, "est - il pentionnaire ?")
        rep = input("oui / non ")
        if rep == ?:
            self.qualite = "DP"
        else :
            self.qualite = "EXT"

    def __str__(self) :
        return( f" {Personne.__str__(self)} {self.qualite} ")

```

Créer l'objet Eleve2

Eleve2= Eleve("GROS","Louis","tg2",['maths', 'physique', None, None, None],{'maths': [14.0, 18.0], 'physique': [12.5]})

Appliquer lui les différentes méthodes de « Personne » et d' « Eleve »