Programmation orientée objet

TP 1 - Introduction à la Programmation orientée objet

Exercice 1 : Vocabulaire et syntaxe

Voici une classe Individu :

```
class Individu:
    def __init__(self, nom, prenom, annee_naissance):
        self.nom = nom
        self.prenom = prenom
        self.annee_naissance = annee_naissance

    def calculAge(self,annee):
        return annee - self.annee_naissance
```

1. Quel nom donne-t-on à la méthode __init__ ?
2. Quels sont les attributs et méthodes de la classe Individu ?
3. A quoi sert la “fonction” **calculAge(self, annee)** ? Que renvoie-t-elle ?
4. Que représente le mot clé self ?
5. Instanciez un individu avec les valeurs de votre choix. (Instancier signifie créer un objet)
6. Créez une nouvelle méthode **estMajeur(self)** qui renvoie True si l’individu a plus de 18 ans et False sinon.

Exercice 2 : Stylo

Dans cet exercice, vous allez implémenter une classe permettant de représenter des Stylos ayant plusieurs couleurs.

Pour cela :

1. Définissez la classe Stylo
2. Implémentez le **constructeur** avec deux attributs :
    - **liste_couleurs** qui représente la liste des couleurs dont dispose le stylo
    - **couleur_actuelle** qui représente la couleur choisie.

    Lorsque l’on instancie un stylo, il n’est pas nécessaire d’avoir choisi une couleur_actuelle. On peut donc fixer sa valeur à “None” dans le constructeur.

3. Implémentez la méthode **choisirCouleur(self, couleur)** qui va vérifier si la couleur choisie est bien dans la liste des couleurs disponibles. Si oui, la couleur actuelle du stylo va devenir cette couleur. Si non, un message sera affiché : “Couleur non disponible”

4. Implémentez la méthode **fermerStylo(self)** qui permet de rentrer la mine du stylo et donc de réinitialiser sa couleur actuelle.

5. Testez vos méthodes en créant une instance de Stylo.

Exercice 3 : Rectangle

Un rectangle est défini par une longueur et une largeur.

1. Définissez la classe Rectangle

2. Implémentez le **constructeur** de la classe Rectangle

3. Implémentez la méthode **périmetre(self)** qui permet de calculer le périmètre d’un rectangle.

   Rappel : Le périmètre d’un rectangle est la somme de tous ses côtés.

4. Implémentez la méthode **aire(self)** qui permet de calculer l’aire d’un rectangle.

   Rappel : L’aire d’un rectangle est égale à la longueur * la largeur

5. Implémentez la méthode **estCarre(self)** qui renvoie True si le rectangle est un carré et False sinon.

6. Testez vos méthodes en créant une instance de Rectangle

7. Implémentez une nouvelle méthode **traceRectangle(self)** qui va tracer le rectangle grâce à turtle.

Exercice 4 : Création d’une classe JeuVideo

Un jeu vidéo est défini par :

- Un titre (chaîne de caractères == string)
- Une ou plusieurs plateformes (liste de strings), par exemple PC, PS4 et XBOX One
- Le fait qu’il soit jouable ou non en multijoueur (booléen)

1. Complétez le code de la classe ci-dessous modélisant un jeu vidéo.

```
class JeuVideo:
    def __init__(self, titre, plateformes, multijoueur):
        self.adefinir = ...
        self.adefinir = ...
        self.adefinir = ...
```

```
    def addPlateforme(self, ...):
        if not(... in self.adefinir):
            self.adefinir.append(...)

    def removePlateforme(self, ...):
        if ... in self.adefinir:
            self.adefinir.remove(...)

    def isOnPlateforme(self, ...):
        return ... in self.adefinir

    def isMultijoueur(self):
        return self.adefinir
```

2. Testez votre classe et méthodes avec la console. Voici les tests et les résultats attendus :

```
>>> mario64 = JeuVideo("Mario 64", ["N64", "DS", "Switch"], False)
>>> mario64.titre
"Mario 64"

>>> mario64.isMultijoueur()
False

>>> mario64.addPlateforme("PS4")
>>> mario64.plateformes
["N64", "DS", "Switch", "PS4"]

>>> mario64.isOnPlateforme("Switch")
True

>>> mario64.removePlateforme("PS4")
>>> mario64.isOnPlateforme("PS4")
False
```

3. Tentez d'exécuter le code suivant :

```
mario64 = JeuVideo("Mario 64", ["N64", "DS", "Switch"], False)
print(mario64)
```

Qu'observez-vous? Pourquoi ?

4. Maintenant, rajoutez la méthode suivante dans votre classe JeuVideo :

```
def __str__(self):
    texteAffichage = self.titre + "\n"
    texteAffichage += "Multijoueur : " + str(self.isMultijoueur()) + "\n"
    texteAffichage += "Plateformes : " + str(self.plateformes) + "\n"
    return texteAffichage
```

Retentez l'exécution du code précédent, qu'observez vous dorénavant ?

La méthode **__str__()** est une "Méthode magique" de python, pour les classes. Sa signature est équivalente pour n'importe quelle classe. Tentez d'expliquer ce qu'elle permet de faire.

Exercice 5 : Dates

Dans cet exercice, vous allez implémenter une classe permettant de représenter des objets de type "Date" définis à partir d'un jour, d'un mois et d'une année.

Vous aurez besoin de la liste des mois :

```
nomMois = ["Janvier", "Fevrier", "Mars", "Avril", "Mai", "Juin", "Juillet",
"Aout", "Septembre", "Octobre", "Novembre", "Decembre"]
```

Pour cela :

1. Définissez la classe Date
2. Implémentez le **constructeur** (trois attributs de l'objet : Le jour, le mois et l'année, des entiers)
3. Implémentez la méthode **nomMois(self)** qui donne le nom du mois de la date (par exemple pour une date 20/09/2020, cela donne "Septembre")
4. Implémentez la méthode magique **__str__(self)** afin d'avoir une représentation de la date sous la forme : Jour Nom du Mois Année. Par exemple pour le 04/03/2021 : 4 Mars 2021.

Exercice 6 : Fractions

Une fraction désigne la représentation d'un nombre relationnel sous la forme d'un quotient de deux entiers. La fraction a/b désigne le quotient de "a" par "b" où "a" est nommé numérateur et "b" dénominateur.

On se sert également des fractions pour représenter des valeurs exactes (par exemple 1/3 qui ne peut qu'être qu'approché si on l'écrit en tant que nombre à virgule.

Le but de cet exercice va être de créer une classe Fraction sous python et de la manipuler afin de représenter des fractions et de les multiplier.

Pour cela :

1. Définissez la classe Fraction
2. Implémentez le **constructeur** (deux attributs de l'objet : Le numérateur et le dénominateur)
3. Implémentez la méthode **valeur(self)** qui donne la valeur de la division du numérateur par le dénominateur de la fraction.
4. Implémentez la méthode **multiplier(self, fraction_bis)** qui va **renvoyer une nouvelle fraction** issue du produit de la fraction self (sur laquelle la méthode est appelée) et la fraction fraction_bis.

   Pour rappel, pour multiplier deux fractions, on multiplie simplement les numérateurs ensembles et les dénominateurs ensembles.
5. Implémentez la "méthode magique" **__str__(self)** afin d'avoir une représentation de la fraction ainsi : (a / b)

Testez votre classe et méthodes avec la console. Voici les tests et les résultats attendus :

```
>>> f1 = Fraction(1,3)
>>> print(f1)
(1 / 3)

>>> print(f1.valeur())
0.3333333333333

>>> f2 = Fraction(3,2)
>>> print(f2.valeur())
1.5

>>> f3 = f1.multiplier(f2)
>>> print(f3)
(3 / 6)

>>> print(f3.valeur())
0.5
```

Exercice 7 : Personnage

Dans cet exercice vous allez implémenter une classe de personnage de jeu vidéo.

1. Définissez la classe Personnage
2. Implémentez le **constructeur** (un seul attribut : les points de vies du personnage)
3. Implémentez une méthode **perdVie(self, pv_perdus)** qui fait perdre à l'objet courant un nombre de point de vie égal à "pv_perdus"
4. Implémentez une méthode **gagneVie(self, pv_gagne)** qui fait gagner à l'objet courant un nombre de point de vie égal à "pv_gagne"
5. Implémentez une méthode **etat(self)** qui affiche les attributs du personnages (ici les points de vies)

Testez votre classe et méthodes avec la console. Voici les tests et les résultats attendus :

```
>>> Diluc = Personnage(500)
>>> Diluc.perdVie(100)
>>> Diluc.etat()
400

>>> Ganyu = Personnage(500)
>>> Ganyu.gagneVie(50)
>>> Ganyu.etat()
550
```

6. Ajoutez un attribut "attaque" et un attribut "defense" au personnage. Il faudra changer le constructeur de la classe.

7. Modifiez la méthode **perdVie(self,pv_perdus)** pour qu'elle fasse perdre des points de vie au joueur égal à la différence entre les points de vie perdus et la défense du joueur.

   Exemple :
   Mon personnage à 10 de défense. Il subit une attaque et doit perdre 25 points de vie. Il absorbe les dégats grâce à sa défense et ne perd que 25 - 10 soit 15 points de vie

8. Testez vos méthodes en faisant en sorte que 2 personnages définis par vos soins s'attaquent mutuellement. Quand un personnage subit l'attaque d'un autre personnage, celui-ci perd autant de points de vie que l'attaque du personnage attaquant.

9. Implémentez d'autres méthodes/attributs de votre choix sur la classe Personnage (ajout d'attributs, ajouts de sorts, chance d'esquiver une attaque...)

Exercice 8 : Carte

Dans cet exercice vous allez implémenter une carte de jeu.

1. Définissez la classe Carte

2. Implémentez le **constructeur** (deux attributs : la valeur allant de 1 à 13 de la carte et sa couleur (Trèfle Carreau Coeur ou Pique)

3. Implémentez une méthode **estFigure(self)** qui indique si la carte est une figure ou non. Cette méthode renverra True si la carte est une figure et False sinon.
   Une figure est un Valet une Dame ou un Roi (de valeur 11, 12 ou 13)

4. Implémentez une méthode **nomFigure(self)** Elle renverra le nom de la figure de la carte si c'est une figure. (Valet, Dame ou Roi)

5. Implémentez une méthode **afficheCarte(self)** qui affiche la carte proprement.
   Exemple :

```
>> c = Carte(12, ”Coeur”)
>> c.afficheCarte()
Dame de Coeur

>> c = Carte(3, ”Pique”)
>> c.afficheCarte()
3 de Pique
```

6. Implémentez une méthode **plusGrande(self, carte)** qui prend en paramètre une carte et qui renvoie la carte de plus haute valeur.da

---

Exercice 9 : Jeu de Cartes

Dans cet exercice vous allez implémenter un jeu de cartes.

1. Définissez la classe JeuDeCartes

2. Implémentez une méthode **creerPaquet(self)** qui va créer un paquet de cartes. Commencez par définir un tableau vide “paquet” qui contiendra les cartes. Pour chaque couleur, et pour chaque valeur, la méthode va créer une nouvelle carte et l’ajouter à “paquet”.
   La méthode renverra à la fin “paquet”

3. Implémentez le **constructeur** de la classe. Il n’y aura qu’un seul attribut “paquet” qui ne sera pas donné en paramètre mais initialisé dans la constructeur grâce à la méthode **creerPaquet(self)**.

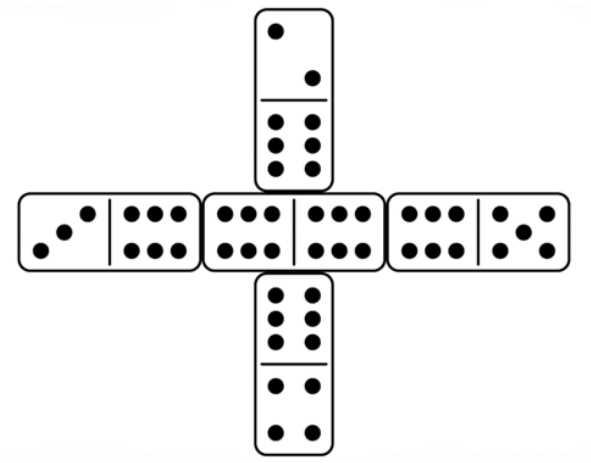
4. Implémentez une méthode **melange(self)** qui va mélanger aléatoirement le paquet. Aidez vous de la méthode **shuffle()** de la bibliothèque random.

5. Implémentez une méthode **afficheJeu(self)** qui affiche proprement chaque carte du jeu.

6. Testez vos méthodes en créant un nouveau paquet de carte, en le mélangeant et en l’affichant.

Exercice 10 : Dominos

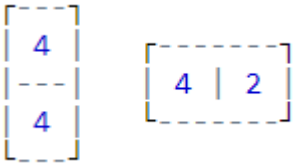
Dans cet exercice, vous allez implémenter une classe permettant de représenter des Dominos.



Un domino possède 2 parties : la partie gauche et la partie droite.
On peut compter le nombre de points d’un dominos et faisant la somme de la partie gauche et de la partie droite.
Il peut y avoir une partie vide. On appelle cela un domino blanc.
Un domino ayant la partie gauche et droite identiques s’appelle un domino double.

1. Définissez la classe Domino et implémentez le constructeur de cette classe.
2. Implémentez une méthode qui permet de calculer le nombre de points d'un domino.
3. Implémentez une méthode qui renvoie True si un domino est blanc et False sinon.
4. Implémentez une méthode pour savoir si un domino est double ou non.
5. Implémentez une méthode qui permet d'afficher un domino. Si le domino est double l'affichage devra être vertical et horizontal sinon.

Exemple :



Pour réaliser cet affichage vous aurez besoin des caractères spéciaux : └ ┘ ┌ ┐

Exercice 11 : Jeu de dominos

Il est maintenant question de créer un jeu de dominos.

Pour cela :

1. Définissez la classe JeuDeDominos
2. Implémentez la méthode **creerJeu(self)** qui va renvoyer une liste de 28 dominos aléatoires.
3. Implémentez le constructeur de cette classe. Il n'y aura qu'un attribut qui contiendra une liste de dominos.
4. Implémentez la méthode **mélanger(self)** qui va mélanger aléatoirement un jeu de dominos. Aidez vous de la méthode shuffle() de la bibliothèque random.
5. Implémentez une méthode permettant d'afficher un jeu de dominos.
6. Implémentez une méthode **distribuer(self)** qui va renvoyer une liste de 7 dominos choisis au hasard dans un jeu de dominos. Il faudra retirer les dominos choisis du jeu de dominos.