TP n°1 - Bases de la Programmation orientée objet

Jusqu'à présent, le principe de programmation que nous avons utilisé est celui de la **programmation procédurale**. Il s'agit d'utiliser des fonctions et procédures afin de décomposer un problème complexe en sous-problèmes plus simples.

Nous abordons ici une autre approche : la **programmation orientée objet (POO)**.

Il s'agit de définir et de faire interagir entre elles des briques logicielles appelées **objets**.

Chaque objet représente un concept, une idée ou toute entité du monde physique. Il est doté d'**attributs** (= caractéristiques) et de **méthodes** (fonctions qui lui sont propres).

I - Objet et classe d'objets

A - Attributs et méthodes

Prenons un exemple d'objet simple : un personnage de jeu vidéo d'aventure.

Ce personnage aura des caractéristiques qui lui sont propres : ses **attributs**.

On peut par exemple imaginer les attributs suivants : nom, position, points de vie, points d'attaques, points de défense, arme, armure, accessoires.

Il pourra aussi faire des actions via des fonctions qui lui permettent d'interagir avec ses attributs et avec le monde extérieur : ses **méthodes**.

On peut par exemple imaginer les méthodes suivantes : déplacement, attaque, parade, récupération d'accessoires.

B - Classe d'objets

Une manière simple d'aborder la notion de classe est de considérer celle-ci comme une « usine » permettant de fabriquer des objets possédant des caractéristiques (attributs) et des actions exécutables (méthodes) communes.

Ainsi, dans le jeu d'aventure ci-dessus, tout personnage (contrôlé ou non par le joueur), aura les caractéristiques décrites précédemment.

classe Personnage : Attributs : Position Points de Vie Points d'attaque Points de défense Arme Armure Accessoires Méthodes : Déplacement Attaque Parade Récupération d'accessoire

L'objet est donc créé à partir de la classe. On dit que c'est une **instance** de la classe. voici deux instances de la classe Personnage :

Personnage 1 : "Hector" Attributs : Position = (12,5) PV = 50 PA = 15 PD = 8 Arme = ("Hache",3) Armure = ("Plastron en fer", 12) Accessoires = ["Manuel de NSI", "Statuette d'Aphrodite"] Personnage 2 : "Achille" Attributs : Position = (12,5) PV = 30 PA = 22 PD = 13 Arme = ("Épée",2) Armure = ("Plastron en cuir", 5) Accessoires = ["L'Iliade", "Talon de chéquier"]

II - Définir une classe et des objets en Python

Partons cette fois-ci d'exemple de classe d'objets plus simple : un point du plan muni d'un repère orthonormé.

Chaque point comprend forcément deux attributs : ses coordonnées.

On pourra ensuite implémenter des méthodes au fur et à mesure (déplacement, calcul de distance, transformation géométrique, ...).

En Python, créer une classe d'objets se fait via l'instruction **class** .

Pour que les attributs de chaque objet soient initialisés lorsque l'on le créera, il faut faire appel à une méthode spécifique appelé **constructeur** : **__init__**

In [ ]:
```
class Point:
    """ Point dans le repère orthonormé """
    # Constructeur :
    def __init__(self,X,Y):
        self.x = X
        self.y = Y
```

__init__ est automatiquement appelée à la création de l'objet :

In [ ]:
```
A = Point(5,7)
```

On a ici créé le point A de coordonnées $(5;7)$.

`A` est une **instance** de la classe `Point` .

Comment accéder aux attributs du point A ?

In [ ]:
```
print(A)
```

La commande `print` ne nous renvoie qu'une information : il s'agit d'un objet stocké à l'adresse mémoire *0xblablablabla*.

Il faut donc passer par le caractère `.` pour accéder aux attributs ou méthodes de l'objet.

In [ ]:
```
print(A.x)
print(A.y)
```

L'attribut **__dict__** , définit implicitement par Python à la création de tout objet, permet d'accéder à la liste des attributs d'un objet.

In [ ]:
```
A.__dict__
```

Exercice 1 : Changer maintenant la valeur de l'abscisse de A en -7.

In [ ]:
```

```

Exercice 2 : Créer un point B de coordonnées $(-12;6)$

In [ ]:
```
```

Exercice 3 : Rajouter un attribut z valant 3 au point A.

In [ ]:
```
```

Exercice 4 : Rajouter l'attribut z à tout nouvel objet de la classe Point. On modifiera le code de la classe ci-dessous :

In [ ]:
```
class Point:
    """ Point dans le repère orthonormé """
    # Constructeur :
    def __init__(self,X,Y):
        self.x = X
        self.y = Y
```

Revenons sur le constructeur **__init__** .

Il a trois paramètres : `self` , `X` et `Y` .

S'il est aisé de comprendre que `X` et `Y` sont les valeurs que l'utilisateur fournit pour les coordonnées à la création de l'objet, le terme `self` est lui moins évident... et il n'est pas appelé lors de la création d'un objet (on a créé A avec la commande `A = Point(5,7)`).

Il s'agit en fait de préciser à Python que cette méthode **__init__** a besoin de travailler sur l'objet lui-même ("self").

`self.x` crée ainsi l'attribut `x` pour l'instance en cours de création. Cet attribut prend la valeur `X` .

Exercice 5 : Créer une méthode `deplacer` pour la classe `Point` qui décale les coordonnées d'un point de valeurs `dx` et `dy` .

In [ ]:
```
class Point:
    """ Point dans le repère orthonormé """
    # Constructeur :
    def __init__(self,X,Y):
        self.x = X
        self.y = Y

    def deplacer(self,dx,dy):
        pass # À remplacer par les instructions de la fonction
```

Rappel : **pass** est l'instruction qui ne fait rien mais évite à la fonction de renvoyer une erreur.

Elle est utile pour créer des squelettes de projets avec les fonctions (vides) déjà en place, comme ci-dessus.

Exercice 6 : Créer une méthode `coordoMax` pour la classe `Point` qui renvoie le nom ('x' ou 'y') du maximum des deux coordonnées.

(*ainsi pour le point A, la méthode renverra 'y' car 7 > 5*)

In [ ]:
```
# On recopiera ici le code entier de la classe terminé à l'exercice 5.
```

Exercice 7 :

Créer une méthode `distanceALOrigine` pour la classe `Point` qui calcule la distance entre le point et l'origine du repère.

Rappels :

- la distance AB se calcule par la formule : $AB = \sqrt{(x_B - x_A)^2 + (y_B - y_A)^2}$.
- pour le calcul d'une racine carrée, il faut utiliser la fonction `sqrt` de la bibliothèque math que l'on pourra importer au début de la fonction `distanceALOrigine` avec `from math import sqrt` .

In [ ]:
```
# On recopiera ici le code entier de la classe terminé à l'exercice 6.
```

III - Faire interagir deux objets :

Exercice 8 : Créer une méthode `distance` pour la classe `Point` qui renvoie la distance entre le point de l'instance appelée et un autre point.

In [ ]:
```
# On pourra intégrer au code ci-dessous les autres méthodes déjà codées
plus haut.

class Point:
    """ Point dans le repère orthonormé """
    # Constructeur :
    def __init__(self,X,Y):
        self.x = X
        self.y = Y

    def distance(self,autrePoint):
        """
            Distance entre ce point et un autre point
            -> autrePoint est un objet de la classe Point
            Par exemple pour calculer la distance entre A et B, on fera
la saisie : A.distance(B)
        """
        from math import sqrt
        pass # À remplacer par les instructions de la fonction
```

Exercice 9 : Créer une méthode `milieu` pour la classe `Point` qui renvoie les coordonnées du milieu du segment dont les extrémités sont le point de l'instance appelée et un autre point.

In [ ]:
```
# On recopiera ici le code entier de la classe terminé à l'exercice 8.
```

Exercice 10 : Créer la classe `Personnage` comme décrite plus haut :

- on construira soigneusement la fonction **init** pour l'initialisation des attributs.
- on préparera simplement le squelette des méthodes abordées (sans les coder) comme montré ci-dessous :

```
def methode(paramètres):
  pass
```

In [ ]:

Exercice 11 : Créer Hector et Achille. On reprendra les caractéristiques proposées ci-dessus.

In [ ]:

Exercice 12 : Coder la méthode `attaque(self, autrePersonnage)` .

Elle consiste à regarder :

- le nombre de points d'attaque du personnage (PA) et le nombre de points de son arme (W pour weapon)
- le nombre de points de défense (PD) du personnage attaqué et le nombre de points de son armure (A) À faire baisser le nombre de points de vie (PV) du défenseur de N via la formule : $N = PA + W - PD - A$ (mais seulement si N > 0)

In [ ]:

Tester en faisant attaquer Hector ou Achille.

In [ ]:

Attribut d'instance ou attribut de classe :

Passer par la fonction **__init__** est assez contraignant et on pourrait être tenté de faire la simplification suivante :

In [ ]:
```
class Velo:
    couleur = "Rouge"
    diametreRoue = 622
    pressionPneus = 5.5
```

La classe `Velo` ci-dessus créera donc par défaut des vélos rouges, avec des roues de 622 mm de diamètre et des pneus gonflés à 5,5 Bars.

In [ ]:
```
monVelo = Velo()
```

In [ ]:
```
print(monVelo.couleur)
print(monVelo.diametreRoue)
print(monVelo.pressionPneus)
```

In [ ]:
```
tonVelo = Velo()
print(tonVelo.couleur)
print(tonVelo.diametreRoue)
print(tonVelo.pressionPneus)
```

In [ ]:
```
sonVelo = Velo()
print(sonVelo.couleur)
print(sonVelo.diametreRoue)
print(sonVelo.pressionPneus)
```

On serait alors tenté de modifier si besoin l'attribut qui diffère de la norme. Ainsi, si `sonVelo` a une couleur "Noire" contrairement aux deux autres, on peut tenter de modifier l'attribut `couleur` :

In [ ]:
```
sonVelo.couleur = "Noire"
```

In [ ]:
```
print(sonVelo.couleur)
```

Vérifions que cela n'a pas posé de problème ailleurs :

In [ ]:
```
print("couleur de sonVelo : ",sonVelo.couleur)
print("couleur de monVelo : ",monVelo.couleur)
print("couleur de tonVelo : ",tonVelo.couleur)
```

Pour autant, s'il faut tout changer d'un coup (les vélos étaient en réalité de couleur "Verte"), il est possible de modifier l'attribut pour la classe :

In [ ]:
```
Velo.couleur = "Verte"
```

Vérifions ce que cela donne :

In [ ]:
```
print("couleur de sonVelo : ",sonVelo.couleur)
print("couleur de monVelo : ",monVelo.couleur)
print("couleur de tonVelo : ",tonVelo.couleur)
```

Ainsi, l'attribut `couleur` défini dans la classe est un attribut commun à tous les éléments de la classe... sauf s'ils ont été directement modifiés via `objet.couleur` .

On parle alors d'**attribut de classe** par opposition aux **attributs d'instances** créés pour un seul objet par la fonction **__init__** (ou par un appel à la commance `objet.couleur = 'Verte'`).

Un attribut de classe peut être très utile pour compter le nombre d'instances créées pour cette classe.

Ainsi, modifions la définition de la classe des vélos :

In [ ]:
```
class Velo:
    # Attribut de classe
    nbVelos = 0

    # Constructeur
    def __init__(self, Couleur = "Rouge", Diametre = 622, Pression = 5.
5): # On donne les valeurs par défaut
        self.couleur = Couleur
        self.diametreRoue = Diametre
        self.pressionPneus = Pression

        Velo.nbVelos += 1
```

In [ ]:
```
# Création d'un vélo
notreVelo = Velo()
```

In [ ]:
```
# Affichage des données :
print(notreVelo.__dict__)
print("nbVelos", Velo.nbVelos)
```

In [ ]:
```
# Création d'un autre vélo
votreVelo = Velo("Bleu")
```

In [ ]:
```
# Affichage des données :
print(votreVelo.__dict__)
print("nbVelos", Velo.nbVelos)
```

In [ ]:
```
# Création d'un autre vélo
leurVelo = Velo("Bleu",622,3.8)
```

In [ ]:
```
# Affichage des données :
print(leurVelo.__dict__)
print("nbVelos", Velo.nbVelos)
```

On mesure bien ici l'évolution de l'attribut de classe et les valeurs des attributs d'instances pour les trois instances créées.

Bibliographie :

- cours de C. Gerland et D. Latouche, Lycée Saint-Exupéry, Mantes-la-Jolie
- https://docs.python.org/fr/3/tutorial/classes.html (https://docs.python.org/fr/3/tutorial/classes.html)
- https://python.sdv.univ-paris-diderot.fr/19_avoir_la_classe_avec_les_objets/ (https://python.sdv.univ-paris-diderot.fr/19_avoir_la_classe_avec_les_objets/)