ISN – Informatique et Sciences du Numérique

PROGRAMMATION ORIENTEE OBJET

1 – LA PROGRAMMATION ORIENTEE OBJET

L'idée de la **programmation orientée objet**, est de **manipuler des éléments** que l'on appelle des "**objets**" dans son code source.

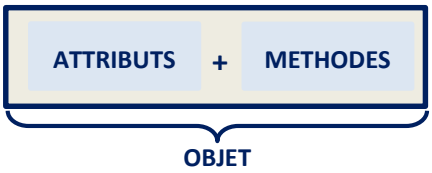
Un des nombreux avantages de la POO, est qu' il existe des **milliers d'objets** (on les appelle des classes) **prêts à être utilisés**. On peut réaliser des programmes extrêmement complexes uniquement en utilisant des **classes (objets) pré-existantes**. Il est possible aussi de créer ses propres classes.

Une **classe** est équivalente à un **nouveau type de données**. On connait déjà **int** ou **str**.

Un **objet** ou une **instance** est un **exemplaire particulier d'une classe**. Par exemple "**truc**" est une **instance de la classe str**.

La plupart des classes **encapsulent** à la fois **les données et les méthodes** (fonctions) applicables aux objets. Un **objet str** contient une **chaîne de caractères** et de nombreuses méthodes comme **upper()** par exemple.

On pourrait définir un objet comme un < **paquet** > **contenant des attributs et des méthodes** :



Les objets ont généralement deux sortes d'attributs : les données nommées simplement **attribut**s et les fonctions applicables appelées **méthodes**. Par exemple un objet de la classe **complex** possède :

- ses attributs : .imag et .real ;
- des méthodes : conjugate() …;
- des méthodes spéciales : +, -, / ...

Une classe peut être une classe fille qui **hérite** alors de tous les attributs (données et méthodes) de sa **super classe**. Comme tous les attributs peuvent être redéfinis, une méthode de la classe fille et de la classe mère peut posséder le même nom mais effectuer des traitements différents (**surcharge**).

2 – CLASSES ET INSTANCIATION D'OBJETS

2.1 - L'instruction class

La création d'une classe en python commence toujours par l'instruction **class**.

Exemple 1 : Création de la classe « personnages »

```
# Défini une classe personnage
class personnage(object):
    # méthode _init_ qui permet initialiser le nombre de vies et
    # le nom du personnage. Il s'agit de la méthode constructeur
    def __init__(self,nbvies,nomperso) :
        self.vie = nbvies
        self.nom = nomperso
    # méthode qui affiche le nombre de vie d'un personnage
    def affVie(self) :
        print("Il reste ",self.vie," points de vie à ",self.nom)
    # méthode qui fait perdre 1 point de vie à un personnage
    def perdVie(self) :
        print(self.nom," a subit une attaque, il perd une vie")
        self.vie = self.vie - 1
# la création de la classe est terminée

# instanciation (création d'instances) et appels de méthodes
gollum = personnage(20,"Gollum")  # On crée l'instance Gollum
gollum.affVie()    # On appelle la méthode affVie pour l'instance Gollum
bilbo = personnage(20,"Bilbo")# On crée l'instance Bilbo
bilbo.affVie()     # On appelle la méthode affVie pour l'instance Bilbo
bilbo.perdVie()    # On appelle la méthode perdVie pour l'instance Bilbo
gollum.affVie()    # On appelle la méthode affVie pour l'instance Gollum
bilbo.affVie()     # On appelle la méthode affVie pour l'instance Bilbo


>>>
Il reste 20 points de vie à Bilbo
Il reste 20 points de vie à Bilbo
Bilbo a subit une attaque, il perd une vie
Il reste 20 points de vie à Bilbo
Il reste 19 points de vie à Bilbo
```

Dans l'exemple ci-dessus, la classe d'objet créée s'appelle « **personnage** ». Cette classe possède **2 attributs** : « **.vie** » et « **.nom** ». Elle possède 3 méthodes : « **__init__** » qui est la méthode constructeur, « **affVie** » et « **perdVie** ».

La méthode constructeur est **exécutée automatiquement** lorsque l'on **instancie un nouvel objet** à partir de la classe. On peut donc y placer tout ce qui semble nécessaire pour **initialiser automatiquement l'objet** que l'on crée.

2.2 – Notion d'héritage

Une classe nouvellement créée peut hériter de **toutes les méthodes et des attributs** d'une autre classe déjà créée. On parle de **classe fille** et **classe parente**.

La classe fille peut avoir des **attributs** et des **méthodes supplémentaires**.

Exemple 12 : Création de la classe « magicien » qui hérite de la classe « personnage »

```
# Défini la classe magicien qui hérite de la classe personnage
class magicien(personnage):
    # création d'un attribut supplémentaire « pointMagie » par rapport
    # à la classe personnage.
    def __init__(self,nbvies,nomperso,pointmagie) :
        # appel de la méthode constructeur de la classe parente
        personnage __init__(self,nbreDeVie,nomperso)
        self.magie = pointmagie
    # méthode qui fait perdre 1 point de magie lors de l'utilisation de celle-ci
    # méthode uniquement disponible pour la classe magicien
    def faireMagie(self) :
        print(self.nom," fait de la magie")
        self.magie = self.magie - 1
    # méthode qui affiche le nombre de points de magie d'un magicien
    def affMagie(self) :
        print(self.nom," a ",self.magie," points de magie")
    # la classe magicien hérite des méthodes de la classe personnage
# instanciations (création d'instances) et appels de méthodes
gandalf = magicien(20,"Gandalf",15)       # On crée l'instance Gandalf
gandalf.affVie()       # On appelle la méthode affVie pour l'instance Gandalf
gandalf.affMagie() # On appelle la méthode affMagie pour l'instance Gandalf
gandalf.faireMagie()   # On appelle la méthode faireMagie pour l'instance Gandalf
gandalf.affMagie() # On appelle la méthode affMagie pour l'instance Gandalf

>>>
Il reste 20 points de vie à Gandalf
Gandalf a 15 points de magie
Gandalf fait de la magie
Gandalf a 14 points de magie
```

La classe **magicien hérite** de la classe **personnage** les attributs « **.vie** » et « **.nom** » et les méthodes « **affVie** » et « **perdVie** ». Elle dispose de l'attribut supplémentaire « **.magie** » et des méthodes supplémentaires « **faireMagie** » et « **affMagie** ».