FACULTY
OF INFORMATICS
Masaryk University

Séminaire CRANS: Python partie 2

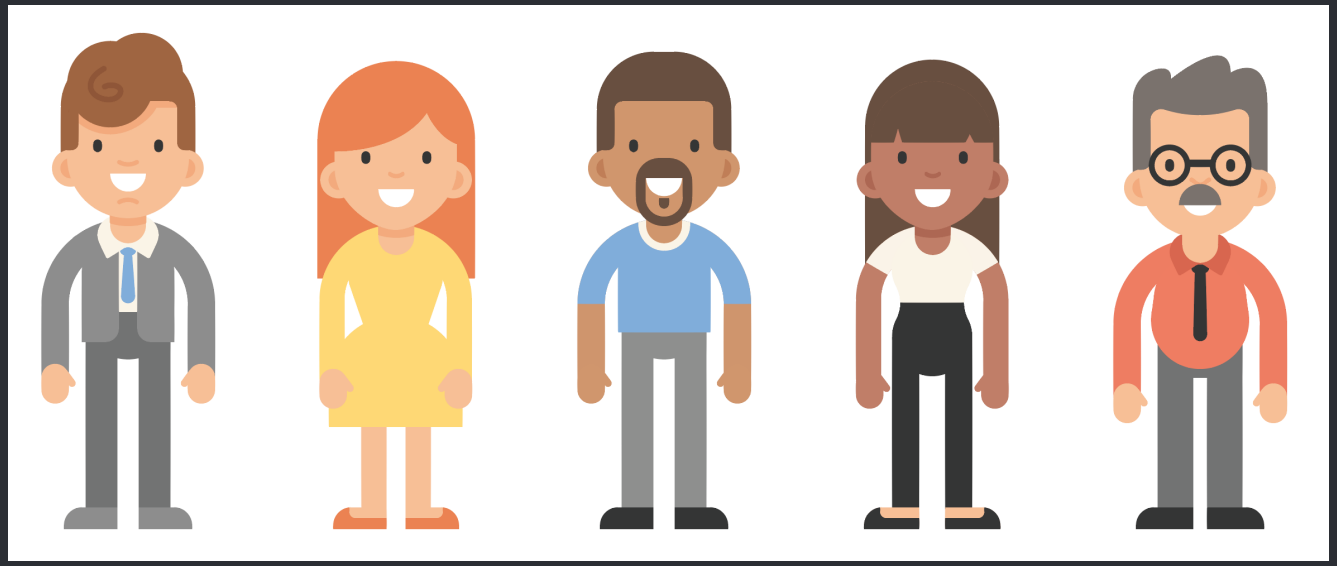
Programmation orientée objet

Grizzly

Paradigme

- **Programmation fonctionnelle:** création et appel de fonctions prenant des arguments et renvoyant (ou non) des sorties.
- **Programmation objet:** création de modèles (class) puis créations d'objets basées sur ces modèles comportant des valeurs (attributs) et des fonctions (méthodes).

Classes de personnage



Attributs:

- Nom: Chaîne de caractère
- PV: Entier de 0 à 100
- Force: Entier de 0 à 100
- Magie: Entier de 0 à 100

Méthodes:

- attaquer(ennemi)
- mourir(message)

Convention de nommage

Convention **CapWord**: Majuscule en début de chaque mot:

- maclasse, Maclasse, Ma_Classe, MACLASSE
- MaClasse, Personnage, SwitchBay

Structure classique d'une classe

```python
class MaClasse():
    """Doc String"""

    attribut1 = "someString"
    attribut2 = 8

    def __init__(self,arg1,arg2):
        """Methode appelés lors de l'initialisation de l'objet
        """
        self.attribut3 = arg1
        self.attribut4 = arg2

    def ma_methode(self,args):
        """Methode quelconque"""
```

Initialisation

```
class Personnage():
    """Un personnage
    """

    def __init__(self,nom,vie,force,magie):
        self.nom=nom
        self.vie=vie
        self.force=force
        self.magie=magie
```

Représentation

```
    def __repr__(self):
        return("{}:{} point de vie".format(self.nom,self.vie))
```

Attaquer et mourir

```
    def attaquer(self,ennemi):
        ennemi.vie=ennemi.vie-10
        if(ennemi.vie<0):
            ennemi.mort("{} killed {}".format(self.nom,ennemi.nom))

    def mort(self,message):
        self.vie=0
        self.force=0
        self.magie=0
        print(message)
```

Instantiation d’un personnage

```
bob = Personnage("bob",100,100,100)
```

Appel des attributs, méthodes

- Variable attribut: bob.vie, bob.force, bob.magie
- Appel d'une méthode: bob.attaquer(bill), bob.mort()

Utilisation des personnages

```
In [1]: from fight import Personnage

In [2]: bob = Personnage('bob', 100,100,100)

In [3]: bill = Personnage('bill',15,100,100)

In [4]: bob.attaquer(bill)

In [5]: bill.vie
Out[5]: 5

In [6]: bob.attaquer(bill)
bob killed bill
```

Classe Arme

```python
class Arme():
    """Une arme qui fait des dégats"""

    def __init__(self,nom,degats):
        self.nom = nom
        self.degats = degats

    def __repr__(self):
        return('{}({}pv)'.format(self.nom,self.degats))
```

Classe Personnage armé

```python
class Personnage():
    """Un personnage
    """

    def __init__(self,nom,vie,force,magie,armes):
        self.nom=nom
        self.vie=vie
        self.force=force
        self.magie=magie
        self.armes=armes

    def __repr__(self):
        return("{}:{} point de vie".format(self.nom,self.vie))

    def attaquer(self,ennemi,arme):
        ennemi.vie=ennemi.vie-arme.degats
        if(ennemi.vie<0):
            ennemi.mort("{} -> {} -> {}".format(self.nom,arme,ennemi.nom))

    def mort(self,message):
        self.vie=0
        self.force=0
        self.magie=0
        print(message)
```

Création et utilisation des armes

```
In [1]: from fight_wp import Personnage, Arme

In [2]: define_armes={'gun':10,'knife':8,'grenade':25}

In [3]: armes=[Arme(nom,define_armes[nom]) for nom in define_armes.keys()]

In [4]: armes
Out[4]: [gun(10pv), knife(8pv), grenade(25pv)]

In [5]: bob = Personnage('bob',100,100,100,armes)

In [6]: bill = Personnage('bill',100,100,100,armes)

In [7]: bob.armes
Out[7]: [gun(10pv), knife(8pv), grenade(25pv)]

In [8]: bob.attaquer(bill,bob.armes[1])

In [9]: bill.vie
Out[9]: 92
```

Héritage simple

Les classes Filles peuvent utiliser toutes les méthodes de toutes les classes dont elles héritent

Heritage simple

```python
class Etudiant():
    """Un Etudiant lambda"""
    def __init__(self,nom,age,ecole):
        self.nom = nom
        self.age = age
        self.ecole = ecole

    def __repr__(self):
        return('{}, etudiant de {} ans a {}'.format(self.nom,self.age,self.ecole))

    def manger(self,nouriture):
        print("{} mange {}".format(self.nom,nouriture))

    def demissionner(self):
        print("{} ne fait plus parti de {}".format(self.nom,self.ecole))

class Normalien(Etudiant):
    """Un etudiant Normalien"""
    ecole = 'ENS'
    def __init__(self,nom,age,departement):
        super().__init__(nom,age,self.ecole)
        self.departement = departement

    def demissionner(self):
        print("{} quitte l'ENS.".format(self.nom))
```

Héritage multiple, parents

```
class Etudiant():
    """Un Etudiant lambda"""
    def __init__(self,nom,age,ecole):
        self.nom = nom
        self.age = age
        self.ecole = ecole

    def __repr__(self):
        return('{}, etudiant de {} ans a {}'.format(self.nom,self.age,self.ecole))

    def manger(self,nouriture):
        print("{} mange {}".format(self.nom,nouriture))

    def demissionner(self):
        print("{} ne fait plus parti de {}".format(self.nom,self.ecole))

class Grimpeur():
    """Un grimpeur"""

    def __init__(self,nom,age,materiel):
        self.nom = nom
        self.age = age
        self.materiel = materiel

    def grimper(self,montagne):
        print("{} grimpe {}".format(self.nom,montagne))
```

Héritage multiple, enfant

```python
class AdherentCaillou(Grimpeur,Etudiant):
    """Un adhérent du club caillou est un grimpeur et un etudiant"""

    def __init__(self,nom,age,materiel,cotisation):
        Etudiant.__init__(self,nom,age,'ENS')
        Grimpeur.__init__(self,nom,age,materiel)
        self.cotisation = cotisation

    def tartiflette(self,nombre):
        print("{} va préparer {} tartiflette pour la gloire du Caillou".format(self.nom,nombre)
```

Assignation dynamique de méthode

TP:

- Créer une classe vide C
- Créer une classe vide D qui hérite de C
- Instancier c et d
- Créer une fonction dummy plop qui affiche le text "plop"
- Créer dynamiquement la methode c.plop = plop
- Utiliser c.plop() et d.plop()

Méthodes spéciales

- PEP8
- "_methode" méthode réputée privée
- "__methode__" méthode spéciale propre à python

Méthodes spéciales

- __init__ création de l'objet
- __del__ destruction de l'objet
- __repr__ représentation de l'objet
- __str__ représentation de l'objet par print (repr sinon)
- __getattr__, __setattr__, __delattr__, recuperation, modification, suppression d'attribut
- __len__ appelée lors de len(instance)
- __add__, __mul__, __truediv__, __pow__,etc appelé par +, *, /, **,etc **[a+b <=> a.__add__(b)]**
- __lt__,__le__,__gt__,__eq__,etc comparaison avec <, <=, >=, =,etc

Exemple

- Objectif: ordonner des livres selon leur nombre de pages sans écrire d'algorithme spécifique.
- Objets: Des Livres qui ont des pages
- Utilisation de sorted()

Tri de livres

```
class Livre():

    def __init__(self,nom,pages):
        self.nom=nom
        self.pages=pages

    def __repr__(self):
        return(self.nom)
```

Tri de livres

```
class Livre():

    def __init__(self,nom,pages):
        self.nom=nom
        self.pages=pages

    def __repr__(self):
        return(self.nom)

    def __lt__(self,autre):
        if(self.pages < autre.pages):
            return(True)
        else:
            return(False)
```

Livres triés

```
In [3]: livres=[Livre('Les Âmes croisée',411),Livre('L\'autre: Tome 1',336),Livre(
'Les mondes d\'Ewilan',384)]

In [4]: livres=sorted(livres)

In [5]: livres
Out[5]: [L'autre: Tome 1, Les mondes d'Ewilan, Les Âmes croisée]
```

Sources

- Doc officielle python
- Sam & Max: Cours en 5+ parties sur le POO python
- irc.crans.org #python