

INTRODUCTION À LA PROGRAMMATION ORIENTÉE OBJET

Vendredi 12 décembre

Option Informatique
Ecole Alsacienne

PLAN

1. La programmation orientée objet
2. Manipuler des objets
3. Créer des objets
4. Quelques méthodes particulières
5. Mini-TD

AVANT DE COMMENCER

- TD à rendre :
 - Listes
 - Tours d'Hanoi
 - Introduction à la programmation orientée objet
- Date limite : vendredi 2 janvier 2015 à 23h59
- N'hésitez pas à poser des questions, ou envoyer une première version non finalisée pour qu'on en discute
- Un des derniers TD sur les notions de base
 - Il est très important que tout le monde maîtrise ces notions de base (variables, types, tableaux et listes)

LA PROGRAMMATION ORIENTÉE OBJET

LA PROGRAMMATION ORIENTÉE OBJET

- *Qu'est-ce que la programmation orientée objet ?*
- Une question qui a fait couler beaucoup d'encre
 - Une "façon de coder"
 - Une "philosophie"
 - Un "paradigme de programmation"
- Le premier langage orienté objet a été imaginé dans les années 60 par Ole-Johan Dahl et Kristen Nygaard
 - Prix Turing en 2001
- POO (en anglais OOP, pour *Object-oriented programming*)

QU'EST CE QU'UN OBJET ?

- Le principe clef de la programmation orientée objet est de définir... des objets !
- Un **objet** est une brique d'un programme qui représente une entité, un concept, ou une idée.
- Exemples :
 - Une voiture
 - Une personne
 - Une couleur
 - Une thématique
 - etc.
- Chaque objet possède une structure interne et sait effectuer un certain nombre de choses
 - Les informations stockées dans l'objet sont appelées les **attributs** de cet objet
 - Les interactions possibles avec un objet sont appelées les **méthodes** de cet objet

STRUCTURER L'INFORMATION

- L'utilisation d'objets permet notamment de structurer l'information et la rendre plus facile à manipuler

- Exemple :

- Sans objet

```
john = ["John", "Doe", 17]
age_john = john[2]
print(type(john))
list
```

- Avec un objet

```
john = Personne("John", "Doe", 17)
age_john = john.age
print(type(john))
Personne
```

OBJETS ET CLASSES

- En programmation orientée objet, on parle souvent d' "objets" et de "classes"
- Une **classe** correspond à un certain type, à un certain modèle.
 - C'est dans la classe que sont définis les attributs et les méthodes que posséderont les objets créés sur ce modèle.
- Un **objet** est un exemple concret d'une classe
 - Un objet dispose des attributs et des méthodes définis dans la classe correspondante.
- Exemple :
 - Classe : Personne
 - Objets : Barack Obama, Bill Gates, etc.
- On dit parfois qu'un objet est une "instance" d'une classe.

DIFFÉRENCES ENTRE LANGAGES

- La notion de programmation orientée objet n'est pas la même dans tous les langages :
 - Il y a des langages où elle n'existe tout simplement pas (ex : C)
 - Il y a des langages où elle est imposée au point que tout code doit être écrit à l'intérieur d'une classe (ex : Java)
 - Il y a des langages qui sont basés sur des objets, mais qui peuvent être utilisés sans se soucier de ces objets
- Python fait partie de cette dernière catégorie
 - En Python, tout est objet
 - Cependant, on peut utiliser Python sans se préoccuper des objets ni créer de nouvelles classes

MANIPULER DES OBJETS

EN PYTHON, TOUT EST OBJET !

- La vérité est que...
 - Vous manipulez déjà des objets sans trop y prendre garde !
- Exemples
 - Les listes/tableaux sont des objets de type `list`
 - Les chaînes de caractères sont des objets de type `str`
- Les fichiers sources mis à vos dispositions lors des TD contiennent souvent des objets créés pour l'occasion.

LES LISTES

- Nous avons vu que le type `list` permet de représenter des tableaux en Python
- Les fonctions que nous avons vues sur ces tableaux sont en fait les méthodes mises à disposition pour la classe `list` :
 - `append`
 - `pop`
 - `extend`
 - `reverse`
 - `etc.`

LES CHÂÎNES DE CARACTÈRES

- Les chaînes de caractères sont aussi des objets, créés selon le modèle défini dans la classe `str`.
- Là encore, ces objets possèdent des méthodes qui vous permettent d'interagir avec ces chaînes de caractères
 - `upper` : Renvoie une copie de la chaîne, entièrement en majuscules
 - `lower` : Renvoie une copie de la chaîne, entièrement en minuscules
 - `capitalize` : Renvoie une copie de la chaîne, avec la première lettre en majuscule
 - `center` : Renvoie une nouvelle chaîne de caractère de longueur déterminée, avec la chaîne initiale en son centre
- Remarques :
 - Ces méthodes renvoient des chaînes de caractères
 - C'est-à-dire de nouveaux objets basés sur la classe `str` !
 - Les méthodes de la classe `str` sont donc disponibles pour ces résultats
 - On peut donc "enchaîner" les méthodes :

```
texte = "hello"  
super_texte = texte.capitalize().center(15)  
print(super_texte)  
Hello
```

LA MÉTHODE **FORMAT**

- Mini-exercice :
 - On dispose de deux variables `prenom` et `age`
 - On souhaite afficher à l'écran le texte
 - Je m'appelle [prenom] et j'ai [age] ans.
 - *Comment faire ?*

- Réponse naïve :

```
print("Je m'appelle ", end = "")
print(prenom)
print(" et j'ai ")
print(age)
print(" ans.")
```

- La méthode `format` de la classe `str` permet d'injecter des données dans une chaîne de caractères
 - Syntaxe avec des numéros :

```
print("Je m'appelle {0} et j'ai {1} ans".format(prenom, age))
```
 - Syntaxe avec des noms :

```
print("Je m'appelle {p} et j'ai {a} ans".format(p = prenom, a = age))
```

CRÉER UN NOUVEL OBJET

- Sans forcément le savoir, vous avez déjà créé de nouveaux objets en Python
- Créer une liste vide :
 - `[]`
 - `list()`
- Créer une chaîne de caractères vide :
 - `""`
 - `str()`
- L'instruction `nomDeLaClasse()` revient à appeler le "constructeur" de la classe `nomDeLaClasse` sans argument supplémentaire.

LES OBJETS EN PYTHON

- Créer un objet
 - Syntaxe : `nomObjet = nomDeLaClasse()`
 - Variante : `nomObjet = nomDeLaClasse(arg1, arg2, ..., argN)`
- Accéder à un attribut
 - Syntaxe : `nomObjet.nomAttribut`
 - Stockage dans une variable : `x = nomObjet.nomAttribut`
- Utiliser une méthode
 - Syntaxe : `nomObjet.nomMethode()`
 - Variante : `nomObjet.nomMethode(arg1, arg2, ..., argN)`
 - Stockage dans une variable : `x = nomObjet.nomMethode(arg1, arg2, ..., argN)`
- Convention d'écriture "CamelCase" (en français "casse de chameau")
 - Premier mot en minuscule
 - Majuscule au début des mots liés
 - Pas d'espace entre les mots

CRÉER DES OBJETS

POURQUOI CRÉER DES OBJETS ?

- *Que peut-on représenter dans un programme Python ?*
- Réponse : tout !
- Il suffit de créer les classes qui correspondent aux objets que l'on veut utiliser.
- Exemple :
 - Eleve
 - Graphe
 - Plateau de jeu
 - etc.

DÉCLARER UNE NOUVELLE CLASSE

- Pour déclarer une nouvelle classe, la syntaxe de base est la suivante :

```
class NomDeLaClasse:
```

```
    def __init__(self):
```

```
        [instructions à exécuter quand on crée un objet]
```

- La première ligne contient le mot-clef `class`, suivi du nom de la classe à créer, et des deux points à ne pas oublier.
- On trouve ensuite une fonction `__init__` :
 - Cette fonction sera exécutée à chaque fois qu'on créera un objet de type `NomDeLaClasse`.
 - Cette fonction est appelée le "constructeur" de cette classe
 - Le premier argument de cette fonction est nommé `self`.

PREMIER EXEMPLE

- Création d'une classe `Personne` :

```
class Personne:
```

```
    def __init__(self):  
        self.prenom = "John"
```

```
x = Personne()  
print(type(x))
```

- Résultat :

```
<class '__main__.Personne'>
```

DÉCLARER DES ATTRIBUTS

- C'est dans le constructeur que l'on va définir la liste des attributs disponibles pour cette classe

- Exemple

```
class Personne:
```

```
    def __init__(self):  
        self.prenom = "John"  
        self.nom = "Doe"
```

```
x = Personne()  
print(x.prenom)
```

- Résultat
John

MODIFIER UN ATTRIBUT

- A partir du moment où un attribut est déclaré pour une classe, il est possible d'y accéder et même de le modifier :

- Exemple

```
class Personne:  
  
    def __init__(self):  
        self.prenom = "John"  
        self.nom = "Doe"
```

```
x = Personne()  
print(x.prenom)  
x.prenom = "Jane"  
print(x.prenom)
```

- Résultat

```
John  
Jane
```

AMÉLIORER LE CONSTRUCTEUR

- Il est possible d'ajouter des arguments au constructeur d'une classe, qui seront requis pour créer un objet de ce type.

- Exemple

```
class Personne:
```

```
    def __init__(self, p, n):  
        self.prenom = p  
        self.nom = n
```

```
x = Personne("Jack", "Smith")  
print(x.prenom)
```

- Résultat

Jack

AJOUTER DES MÉTHODES

- On peut ensuite définir des méthodes pour interagir avec les objets de ce type.
- Il s'agit en quelque sorte de fonctions propres à cette classe.
- Toutes ces méthodes doivent prendre au moins un argument, qui est nommé `self` :
 - Cet argument correspond à l'objet lui-même
 - Ce doit toujours être le premier argument
 - Grâce à cet argument, on peut accéder aux attributs de l'objets, grâce à la syntaxe `self.nomAttribut`

EXEMPLE

- Définition d'une classe avec un constructeur et une méthode :

```
class Personne :
```

```
    def __init__(self, p, n):  
        self.prenom = p  
        self.nom = n
```

```
    def sePresenter(self):  
        s = "Bonjour, je m'appelle " + self.prenom + " " + self.nom + "!"  
        print(s)
```

```
x = Personne("Jack", "Smith")  
x.sePresenter()
```

- Résultat :
Bonjour, je m'appelle Jack Smith!
- Remarque importante : Lorsqu'on fait appel à une méthode d'un objet, il est inutile de renseigner le paramètre `self`.

EXEMPLE

- Ajout d'une méthode avec un argument supplémentaire :

```
class Personne :
```

```
    def __init__(self, p, n):  
        self.prenom = p  
        self.nom = n
```

```
    def sePresenter(self):  
        s = "Bonjour, je m'appelle " + self.prenom + " " + self.nom + "!"  
        print(s)
```

```
    def changerDeNom(self, nouveauNom):  
        self.nom = nouveauNom
```

```
x = Personne("Jack", "Smith")  
x.sePresenter()  
x.changerDeNom("Doe")  
x.sePresenter()
```

- Résultat :
Bonjour, je m'appelle Jack Smith!
Bonjour, je m'appelle Jack Doe!

QUELQUES MÉTHODES PARTICULIÈRES

ET LA FONCTION `PRINT` ?

- Exemple :

```
class Personne :  
  
    def __init__(self, p, n):  
        self.prenom = p  
        self.nom = n
```

```
x = Personne("Jack", "Smith")  
print(x)
```

- Résultat :

```
<__main__.Personne object at 0x029E0990>
```

- Par défaut, la fonction `print` ne sait pas quoi faire lorsqu'on l'utilise sur un objet de type inconnu.

LA MÉTHODE __REPR__

- Pour "expliquer" à Python comment utiliser la fonction `print` sur votre classe, il vous suffit de définir une méthode `__repr__` pour votre classe
- Cette fonction doit renvoyer une chaîne de caractères
 - C'est cette chaîne qui sera affichée quand on utilisera la fonction `print`
- Cette fonction prend un seul argument : `self`
- Il n'est pas possible de changer le nom de cette méthode.
- Remarque : Il existe en fait une autre méthode spéciale pour l'affichage, `__str__`, mais si elle n'est pas définie, Python utilise la méthode `__repr__` à la place

LA MÉTHODE __REPR__ : EXEMPLE

- Exemple

```
class Personne :  
  
    def __init__(self, p, n):  
        self.prenom = p  
        self.nom = n  
  
    def __repr__(self):  
        s = "Personne : " + self.prenom + " " + self.nom  
        return s
```

```
x = Personne("Jack", "Smith")  
print(x)
```

- Résultat

```
Personne : Jack Smith
```

METHODES "SPÉCIALES"

- Il existe ainsi plusieurs méthodes "spéciales" que l'on peut définir pour une classe
- Toutes ces méthodes ont un nom de la forme `__nomDeLaMethode__` (deux underscore de chaque côté)
- Ces méthodes "spéciales" permettent de définir certaines opérations classiques pour cette classe
 - Test de comparaison : égalité (`__eq__`), plus grand que (`__gt__`), ...
 - Arithmétique : addition (`__add__`), soustraction (`__sub__`), multiplication (`__mul__`), ...
 - Indexation : accès au $i^{\text{ème}}$ élément (`__getitem__`) (symbole `[]`), ...

MINI-TD

PROCHAINE SÉANCE

Vendredi 19 décembre

[TD] TROLLS ET CHÂTEAUX

