

Introduction à la programmation orientée objet

Caroline Koudoro-Parfait

Crédits : Hugo Chevrotton

Doctorant ObTIC, STIH, SCAI

caroline.parfait@sorbonne-universite.fr

28 février 2022

Source

- James RUMBAUGH, (1997), *Modélisation et conception orientées objet*, Édition MASSON
- Laurent Audibert, (2014), *UML 2 de l'Apprentissage à la Pratique*, Édition Ellipses
- <https://openclassrooms.com>, *Découvrez la programmation orientée objet avec Python*
- Wikipédia

A quoi sert ce cours ?

Pourquoi êtes vous ici ?

Cette matière forme aux métiers du **développement d'applications** numériques pour le traitement automatique des Langues Orientales

Développement d'application

- Une application est un programme (ou un ensemble logiciel) directement utilisé pour **réaliser une tâche** (Wikipédia).
- Le développement de logiciel consiste à **étudier, Concevoir, construire, transformer, mettre au point, maintenir** et **améliorer** des logiciels (Wikipédia).

Le développement d'application pour réaliser une tâche (ou répondre à un cahier des charges) demande de la méthode.

Méthode de développement dite "orientée objet" - 4 étapes

Analyse : Vise à comprendre le **but** d'une application.

Conception du système : Définitions des éléments composants le logiciel et leurs rôles.

Conception de objet : Description précise de la structure des éléments et de l'ensemble de l'application.

Implémentation :

L'ensemble est implémenté dans un langage.

Les décisions des 3 premières étapes sont indépendantes du langage de programmation choisi.

Exemple - Une famille veut faire construire une maison

Analyse : Définissions des besoins.

- Quelles sont les envies de chacun pour cette maison ?
- Pour combien de personnes ?
- A quelle distance des commerces, écoles, bureaux ?
- Avec ou sans jardin ?

Conception du système : Ébauche une solution et fixe les points important.

- Choix du terrain
- Choix du nombre de pièce
- Choix de l'agencement général

Conception des objets : Rentrer dans le détail

- Conception du réseau électrique, du réseau eau
- Choix des matériaux
- Réalisation de plans d'architecte

Implémentation : Construction

- Choix du maître d'œuvre
- Travaux

A quoisert ce cours ?

Ce cours est une initiation aux outils et concept qui vous permettrons d'utiliser la méthode de développement orienté objet :

Outils et concepts

- POO : Programmation Orientée Objet
- UML : Langage de Modélisation Unifié
- Python

POO : Programmation Orientée Objet

Concept de programmation organisant un logiciel en une collection d'**objet** interagissant entre eux.

Chaque objet possède :

- Des données propres (attributs\valeurs)
- Un comportement (méthode\opération)

Exemple

John est un homme. Comme tous les hommes, il a un nom et une adresse et il fume la pipe. Il possède deux chiens Castor et Polux qui mangent des croquettes.

UML : Langage de Modélisation Unifié

Langage de modélisation graphique utilisé pour visualiser la conception d'un système orienté objet.

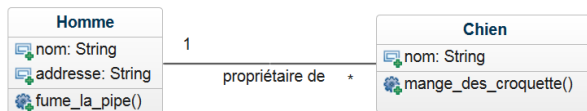


Figure – Diagramme de classe en UML

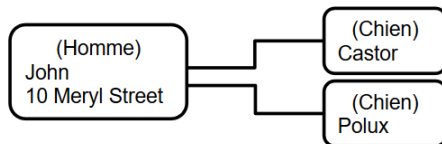


Figure – Diagramme d'instance

Python

Outils pour implémenter le logiciel une fois celui-ci conçu.

```
10 class Homme:
11     def __init__(self,nom, adresse):
12         self.nom = nom
13         self.adresse= adresse
14
15     def fume_la_pipe(self):
16         print("Peuf peuf")
17
18 class Chien:
19     def __init__(self,nom):
20         self.nom = nom
21     def mange_des_croquettes(self):
22         print("Cronch cronch")
23
24 def main():
25     john = Homme("John", "10 Meryl Street")
26     castor = Chien("Castor")
27     polux = Chien("Polux")
28
29     john.fume_la_pipe()
30     castor.mange_des_croquettes();
31     polux.mange_des_croquettes();
32
33 main()
```

Sortie de la console :

Peuf peuf
Cronch cronch
Cronch cronch

Les objets, c'est la base

Un objet sert :

- A représenter précisément un élément du monde réel dans le contexte de notre application
- A proposer une base pratique pour l'implémentation



(Homme)
John
10 Meryl Street

Les classes d'objet

Les objets ayant les mêmes caractéristiques sont regroupés et décrits par une même **classe**.

Une classe d'objet définit

- **Les attributs** (ces caractéristiques)
- **Les méthodes** (ces actions possibles)

... des objets qu'elle représente.

Une classe n'est pas quelque chose de "physique", mais elle décrit des élément physique.

Les instances d'objet




Les **instances d'objet** représentent des objets réels de l'application. Elles **instancient** la classe qui les décrit.

Pour les instances, on parle :

- **De valeur (d'attribut)**
- **D'opération**

Exemple



Homme	
	nom: String
	adresse: String
	fume_la_pipe()

(Homme)
John
10 Meryl Street

(Homme)
Bobby
15 Jackson Avenue

(Homme)
Wilfried
2 Monty Hole

Identité

Du seul fait de son existence,
chaque objet a une **identité propre**.

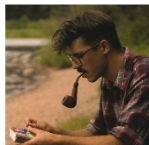
Il a conscience de :

- Lui-même.
- La classe à laquelle il appartient.
- Des liens qu'il entretient avec les autres objets.

Exemple



(Homme)
Bobby
10 Meryl Street



(Homme)
Bobby
10 Meryl Street

- Bobby (1) et Bobby (2) sont tous les deux des hommes, et ils le savent.
- Bobby (1) et Bobby (2) ont les mêmes valeurs d'attribut.
- Mais Bobby (1) et Bobby (2) ne sont pas la même personne.

Syntaxe en Python

```
10 class Homme:
11     def __init__(self, nom):
12         self.nom = nom
13         self.nombre_de_chien = 0
14
15     def fume_la_pipe(self):
16         print(self.nom + "fume sa pipe.")
17
18 def main():
19     john = Homme("John")
20     john.fume_la_pipe()
21
22 main()
```

Appel des attributs et méthode d'un objet

Dans le **Main** :

```
mon_objet.attribut  
mon_objet.méthode()
```

Utilisation d'un . .

Syntaxe

En Python, respectez rigoureusement les tabulations ! ! !

`class` Ma_classe :

Cette ligne permet de déclarer ma **classe**, sa syntaxe est proche de celle pour déclarer une fonction.

Le constructeur de classe

```
def __init__(self, un_paramètre ) :
```

Cette fonction s'appelle un **constructeur**.

Elle permet de créer une instance d'objet d'une classe donnée.

L'objet en question est retourné par le constructeur.

Son nom particulier `__init__` lui permet d'être reconnu par le compilateur.

Attention, mettez bien 2 `_` de chaque côté de `init`.

`self` est un mot clef du langage python. Il fait référence à l'objet lui même.

Il est systématiquement passé en premier paramètre d'une méthode de l'objet.

Appel du constructeur

Dans le main :

```
variable = Ma_classe(un_paramètre)
```

La fonction portant le nom de la classe appelle directement le constructeur `__init__` de la classe.
L'instance d'objet qu'elle retourne est stocké dans la variable.

Définition des attributs d'une classe

`self.attributs = valeur`

Comme dit plus haut, `self` permet d'accéder à l'objet lui même.
Cette ligne permet :

- de définir les attributs de l'objet
- de leur attribuer une valeurs

Le fait de définir les attributs d'une classe dans le constructeur est une particularité du Python.
Il est recommandé de bien définir **tous** les attributs d'une classe dans son constructeur.

Définition des méthodes d'une classe

```
def ma_methode(self) :
```

Une méthode de classe se définit comme n'importe quelle fonction.

Comme particularité, elle prend en premier paramètre l'objet **self** qui l'appelle.

Remarque sur les appels de méthode

```
mon_objet.méthode()
```

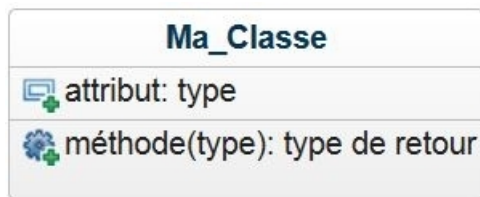
Dans le **main**, lors de l'appel d'une méthode d'un objet, l'objet est automatiquement passé en premier paramètre de la méthode.

C'est pourquoi ce paramètre (**self**) doit être déclaré dans le prototype de la fonction.

Méthode de classe

```
def Ma_classe :  
    def fonction_de_classe() :  
        print("Something")  
  
Ma_Classe.fonction_de_classe()
```

Syntaxe UML



Se créer un compte sur Google, Puis utiliser **Draw io** :
https://app.diagrams.net/#G13E_20KcTie3mHAP0LTo3Vz1puDcqER_Y

Protection des méthodes

Les méthodes implémentées dans classe peuvent avoir différents niveaux de visibilité en fonction de leur utilisation.

- Si elles sont utilisées en dehors de la classe : **public**
- Si elles ne sont utilisées qu'à l'intérieur de la classe (et que l'utilisateur ne doit pas y avoir accès) : **private**