cours_initiation_programmation_orientee_objet

December 9, 2020

1 Programmation orientée objet

1.1 Introduction

La programmation orientée objet (notée **POO** dans ce cours) a vu le jour dans les années 1980/1990. Ce n'est pas un nouveau langage de programmation mais un nouveau **paradigme de programmation**. Ce paradigme a connu un grand succès à tel point que la plupart des langages informatiques actuellement utilisés sont des langages orientés objets comme python (Voir classement).

1.1.1 Un nouveau paradigme

Un **paradigme de programmation** est une façon d'aborder la programmation informatique. C'est "la vue" qu'ont les développeurs informatiques de leur programme

Pour l'instant vous connaissez la : * La programmation impérative : On considère un programme comme une séquence d'instructions. Les instructions élémentaires sont l'affectation, les conditions, les boucles. C'est le paradigme "élémentaire" et "naturel" et aussi celui utilisé par les processeurs.

- La programmation procédurale : programmation impérative + notions de fonctions. On considère un programme, appelé programme principal, comme un programme qui appelle des "sous-programmes" (les fonctions)

En programmation orientée objet, on considère le programme comme une interaction entre ce qu'on va appeler des **objets**

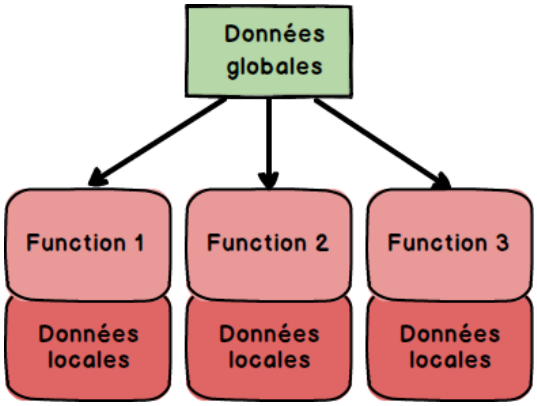
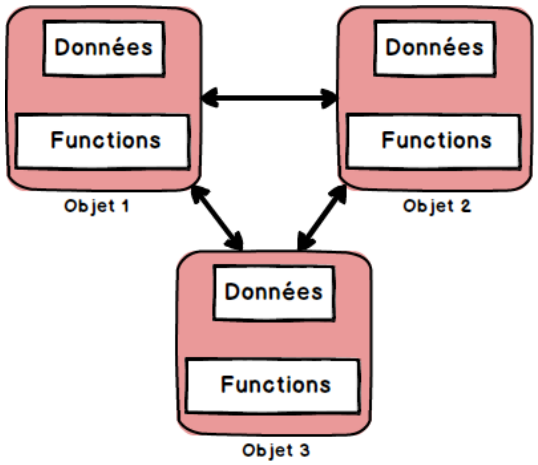
A noter que souvent les langages informatiques modernes sont multi-paradigmes (comme python)

Pour l'instant vous avez "entrevu" la notion d'orienté objet, notamment au travers de la **notation pointée** ou de la fonction `type`. En python, tout est objet…

[1]:
```
ma_liste = [1,7,5,9]
print(ma_liste)
print(type(ma_liste)) # Vous apprendrez que ma_liste est un OBJET INSTANCE de␣
 ↪la CLASSE list


ma_liste.append(3) # Vous apprendrez que append est une METHODE de la classe␣
 ↪list
print(ma_liste)
```

```
[1, 7, 5, 9]
<class 'list'>
[1, 7, 5, 9, 3]
```

PROGRAMMATION PROCEDURALE	PROGRAMMATION ORIENTEE OBJET
	
En programmation procédurale, on se focalise sur la démarche pour résoudre le problème (et donc les instructions). Les données du problème sont des variables globales au programme	En POO, on se focalise sur les objets : qui sont-ils et comment interagissent-ils entre eux ? Les données du problème sont réparties dans les différents objets. De même les fonctions traitant ces données sont réparties dans les différents objets

Sources des images : https://waytolearnx.com/2018/09/difference-entre-programmation-procedurale-et-orientee-objet.html

La programmation procédurale a quelques inconvénients : * Les données sont comme éparpillées dans le programme. L'utilisation de variables globales sont souvent source de problèmes lors du développement du code…

- La modification du code est difficile lorsque le programme commence à être long

La POO répond à ces inconvénients : le programme est mieux organisé et donc plus facilement modifiable. Dans ce paradigme, on pourrait dire que le programme est d'abord centré sur les données manipulées, regroupées au sein d'objets…

1.2 Notion d'objet

Bonne nouvelle : le concept d'objet en informatique correspond bien au concept d'objet dans la vie courante…

Cependant, les objets ne sont pas uniquement des choses matérielles (une date, un animal, un individu peuvent être des objets au sens informatique du terme)

Activité : à partir d'objets du monde réel, introduire les notions d'identité, de caractéristiques propres à l'objet et de comportement de l'objet

Prenons par exemple : une voiture, une bougie mais aussi un compte bancaire, un personnage de jeu vidéo, un joueur etc…

Un objet possède : * une **identité** propre : un espace mémoire spécifique est réservé à chaque objet. Chaque objet est donc différent même s'il a des caractéristiques identiques à un autre objet.

- des **attributs** : les caractéristiques de l'objet. Les attributs sont des données du programme
- des **méthodes** : le comportement de l'objet. Les méthodes sont des fonctions définiées **dans** l'objet et pouvant manipuler les attributs

1.3 Notion de classe

Activité : à partir d'exemples du monde réel, comprendre que les classes sont l'"archétype" ou le modèle des objets.

Une classe est une "usine" à construire des objets selon un modèle bien défini.

La classe permet de définir : * Les caractéristiques (attributs) que possèdent **TOUS** les objets de cette classe

- Le comportement (méthode) que peuvent avoir **TOUS** les objets

Tous les objets créés à partir de cette classe auront les mêmes attributs et les mêmes comportements (mais pas les mêmes valeurs sur leurs attributs). Dit autrement, tous les objets sont construits sur le même modèle mais ne sont pas tous identiques. On dit que les objets sont **instance** de la classe dont ils dépendent.

1.4 Syntaxe

- Le mot-clé `class` permet de créer un bloc (indenté !) contenant le code définissant une nouvelle classe
- La méthode `__init__` est appelée (injustement) **constructeur de classe** :
  - Cette méthode permet de définir les attributs de l'objet et leurs valeurs à la création de l'objet
  - On passe en paramètres eventuels à la méthode `__init__` des valeurs nécessaires à la création de l'objet
- Le paramètre `self` représente un objet instance de la classe. Il doit obligatoirement être le **PREMIER paramètre** dans les parenthèses !

Par convention, le nom de la classe doit toujours commencer par une majuscule !

1.4.1 Code python pour créer une classe

```
class Nom_classe :
    def __init__(self, parametres_eventuels):
        self.attribut1 = valeur1
        self.attribut2 = valeur2

    def nom_methode1(self, parametres_eventuels) :
        # code python de la methode1
```

```
    def nom_methode2(self) :
        # code python de la methode2

    # On peut ajouter autant de méthodes que nécessaire...
```

1.4.2 Code python pour manipuler un objet

```
# CREATION DES OBJETS. Ici on a créé 2 instances de notre classe

mon_objet = Nom_classe(parametres_eventuels)
un_autre_objet = Nom_classe(parametres_eventuels)

# MANIPULATION DES OBJETS

# Accès aux attributs
mon_objet.attribut1

# Utilisations des méthodes
mon_objet.nom_methode1(parametres_eventuels)
mon_objet.nom_methode2()
```

Remarque : Les méthodes ont toujours des parenthèses (normal : ce sont des fonctions !)

1.5 Exemple

1.5.1 Activité

On veut manipuler des objets représentants des **comptes bancaires**. On suppose que, pour les besoins de notre programme, les comptes bancaires doivent avoir posséder : * un solde * un titulaire * un numero de compte.

D'autre part, il doit être possible d'y déposer et de retirer de l'argent.

1.5.2 solution : code python

création de la classe

[2]:
```
class CompteBancaire :
    def __init__(self, numero, titulaire):
        self.numero = numero
        self.titulaire = titulaire
        self.solde = 0

    def deposer_argent(self, montant) :
        self.solde = self.solde + montant

    def retirer_argent(self, montant) :
        self.solde = self.solde - montant
```

instanciation d'un objet

[3]:
```
un_compte = CompteBancaire (12345, "Alice")
```

manipulation de l'objet

- Accès aux attributs

[4]:
```
un_compte.numero
```

[4]: 12345

[5]:
```
un_compte.titulaire
```

[5]: 'Alice'

[6]:
```
un_compte.solde
```

[6]: 0

- Utilisation des méthodes

[7]:
```
un_compte.deposer_argent(100)
un_compte.retirer_argent(20)

print(un_compte.solde)
```

80

1.6 A retenir

Conformément au programme de terminale, voilà selon moi, les notions à **maîtriser parfaitement**

- objet
- classe
- attribut
- methode
- instance
- syntaxe de base pour créer de nouvelles classe et manipuler des objets

1.7 Limitations de la POO dans le programme de terminale

Conformément au programme, ce cours est une introduction à la POO. Il vous est seulement demandé de comprendre les notions d'objet, de classe, de méthode et d'attribut. Malheureusement, cette limitation ne permet pas de bien comprendre tout l'intérêt et la puissance de la POO. En effet, au delà de ces notions au programme, la POO s'appuie sur 4 piliers :

- L'**encapsulation** : La structure de l'objet est cachée et propose plutôt des méhodes pour manipuler les propriétés de cet objet. De cette manière on peut s'assurer de la manière dont seront traitées ces propriétés.

- L'**aggrégat** : Le fait qu'un objet peut contenir d'autres objets
- L'**héritage** : Mécanisme qui permet aux classes "filles" d'utiliser les caractéristiques d'une classe "parent"
- Le **polymorphisme** : Possibilité d'utiliser le même code informatique pour des types informatiques différents

Parmi ces 4 piliers, on abordera l'encapsulation et un peu l'aggrégat (d'après la "philosophie" du programme). Pour le reste, cela sera abordé en projet (…peut-être, c'est pas sûr, éventuellement, pourquoi pas, si le projet s'y prête…)