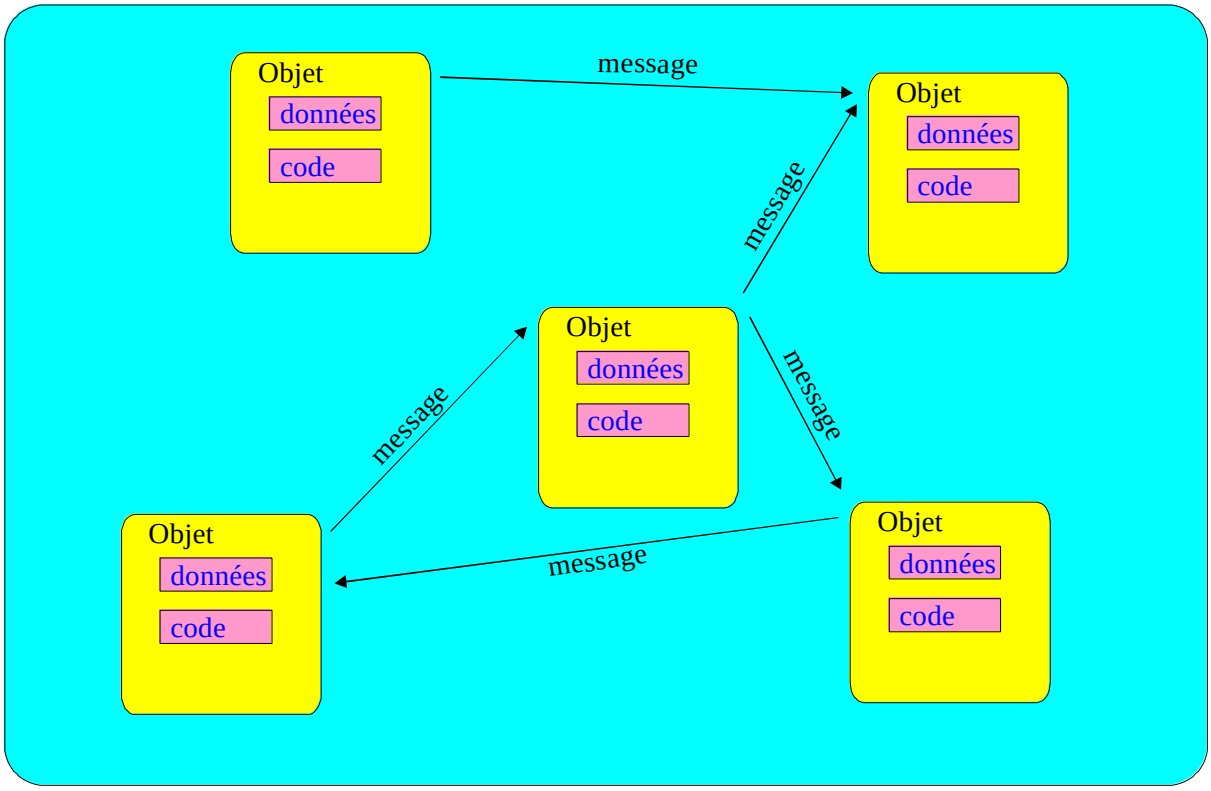
Algorithmique et Programmation
La Programmation Orientée Objet - POO

I- La programmation orientée objet

La programmation orientée objets (POO) est une technique d'organisation du code d'un programme en le groupant en objets, les objets étant ici des éléments individuels comportant des informations (valeurs de données) et des fonctionnalités. L'approche orientée objet permet de regrouper des éléments particuliers d'informations (par exemple, les informations d'un enregistrement musical : titre de l'album, titre de la piste ou nom de l'artiste) avec des fonctionnalités ou des actions communes associées à ces informations (comme l'ajout de la piste à une liste de lecture ou bien la lecture de tous les enregistrements de cet artiste). Ces éléments sont rassemblés en un seul, l'objet (par exemple, un « Album », ou une « piste »). La possibilité d'intégrer ainsi toutes ces valeurs et ces fonctions offre divers avantages : par exemple, il est possible de ne suivre qu'une seule variable plutôt que plusieurs d'entre elles, de regrouper des fonctionnalités liées entre elles et de structurer les programmes pour qu'ils se rapprochent davantage du fonctionnement humain.



STRUCTURE D'UN PROGRAMME ORIENTE OBJET

II- Notion d'objet

La **programmation orientée objet** consiste à modéliser informatiquement un ensemble d'éléments d'une partie du monde réel (que l'on appelle *domaine*) en un ensemble d'entités informatiques. Ces entités informatiques sont appelées **objets**. Il s'agit de données informatiques regroupant les principales caractéristiques des éléments du monde réel (taille, couleur, ...).

La difficulté de cette modélisation consiste à créer une représentation abstraite, sous forme d'objets, d'entités ayant une existence matérielle (chien, voiture, ampoule, ...) ou bien virtuelle (sécurité sociale, temps, idée...).

Un objet est caractérisé par plusieurs notions :

- **Les attributs** : Il s'agit des données caractérisant l'objet. Ce sont des variables stockant des informations d'état de l'objet
- **Les méthodes** (appelées parfois *fonctions membres*) : Les méthodes d'un objet caractérisent son comportement, c'est-à-dire l'ensemble des actions (appelées *opérations*) que l'objet est à même de réaliser. Ces opérations permettent de faire réagir l'objet aux sollicitations extérieures (ou d'agir sur les autres objets). De plus, les opérations sont étroitement liées aux attributs, car leurs actions peuvent dépendre des valeurs des attributs, ou bien les modifier
- **L'identité :** L'objet possède une identité, qui permet de le distinguer des autres objets, indépendamment de son état. On construit généralement cette identité grâce à un identifiant découlant naturellement du problème (par exemple un produit pourra être repéré par un code, une voiture par un numéro de série, etc.)

III- Notion de classe

Les classes sont les principaux outils de la programmation orientée objet ou POO (*Object Oriented Programming ou OOP* en anglais).

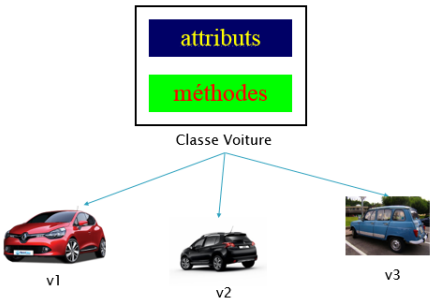
1. Définition

On appelle **classe** la structure d'un objet, c'est-à-dire la déclaration de l'ensemble des entités qui composeront un objet. Un objet est donc « issu » d'une classe, c'est le produit qui sort d'un moule. En réalité on dit qu'un objet est une **instanciation** d'une classe, c'est la raison pour laquelle on pourra parler indifféremment d'**objet** ou d'**instance** (éventuellement d'*occurrence*).

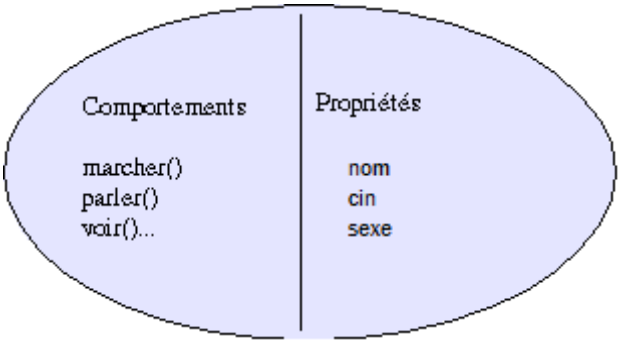
Une classe est composée de deux parties :

- **Les attributs** (parfois appelés *données membres*) : il s'agit des données représentant l'état de l'objet
- **Les méthodes** (parfois appelées *fonctions membres*): il s'agit des opérations applicables aux objets

Si on définit la classe *voiture*, les objets *Peugeot 406*, *Renault 18* seront des instanciations de cette classe. Il pourra éventuellement exister plusieurs objets *Peugeot 406*, différenciés par leur numéro de série. Mieux: deux instanciations de classes pourront avoir tous leurs attributs égaux sans pour autant être un seul et même objet. C'est le cas dans le monde réél, deux T-shirts peuvent être strictement identiques et pourtant ils sont distincts. D'ailleurs, en les mélangeant, il serait impossible de les distinguer...



Exemple, la classe Humain *définit* des propriétés (nom, cin, sexe, …) et des comportements (marcher, parler, voir...).



2. Déclaration d'une classe

Pour créer une nouvelle classe d'objets Python, on utilise l'instruction `class`. Nous allons donc apprendre à utiliser cette instruction, en commençant par définir un type d'objet, lequel sera simplement un nouveau type de donnée. Nous avons déjà utilisé différents types de données jusqu'à présent, mais c'étaient à chaque fois des types intégrés dans le langage lui-même. Nous allons maintenant créer un nouveau type composite : le type `Point`.

```
class Point:
    """ Classe représentant un Point"""

    def __init__(self, px , py):
        #  variables, uniques pour chaque objet (les attributs)
        self.x = px
        self.y = py
```

La méthode spéciale **__init__()** est appelée automatiquement lorsqu'un objet est créé.
Il est possible de lui passer des arguments qui seront utilisés pour initialiser les attributs de l'objet créé avec certaines valeurs.

Vous êtes tenu de déclarer self, le premier paramètre, qui sera l'instance en cours.

Un Autre exemple : Class CompteBancaire

```
class CompteBancaire:
    """ Classe représentant un compte bancaire"""

     def __init__(self, nom):
        #  variables par défaut, uniques pour chaque objet :
        self.nom = nom
        self.solde = 0.0

    # Méthodes de l'objet :

    def afficherSolde(self):
        print self.solde


    def effectuerRetrait(self, montant):
        self.solde -= montant


    def effectuerDépôt(self, montant):
        self.solde += montant
```

3. Instanciation des objets :

Nous venons de définir une classe **Point()**. Nous pouvons dès à présent nous en servir pour créer des objets de ce type, par ***instanciation***. Créons par exemple un nouvel objet **p1** :

```
>>> p1 = Point(5, 3)
```

Après cette instruction, la variable p1 contient la référence d'un nouvel objet **Point**. Nous pouvons dire également que **p1** est ***une nouvelle instance*** de la classe **Point**.

4. Accès aux attributs des objets :

Les attributs d'un objet sont accessibles et modifiables après la construction de **l'objet.**
La syntaxe à utiliser est : **objet.attribut**

Exemple :

```
p = Point(2, 4)

print(p.x)   # affichera 2

p.x =  10

print(p.x) # affichera 10
```

5. Les méthodes d'une classe :

- Les méthodes ou les opérations d'une classe décrivent le comportement des objets générées à partir de cette classe.
- Les méthodes assurent l'interaction entre les différents objets à travers l'échange de messages.
- Une méthode représente une fonction qui permet de réaliser une ou plusieurs tâches.

Pour définir une méthode, il faut :

- Indiquer son nom
- Indiquer les arguments entre des parenthèses.
- Le premier argument d'une méthode doit être self.

Exemple :

```
class Point:
    """ Classe représentant un Point"""

    def __init__(self, px , py):
        #  variables, uniques pour chaque objet (les attributs)
        self.x = px
        self.y = py

    def affichePoint(self):
        print(self.x , self.y)

    def distance(self, point):
        return sqrt((self.x-point.x)**2+(self.y-point.y)**2)
```

Pour accéder aux méthodes d'un objet, on indique :

- le nom de la variable qui fait référence à cet objet
- un point
- le nom de la méthode

Exemple :

P1 = Point(2, 4)

P2 = Point(4, 7)

P1.affichePoint( )

P2.affichePoint( )

D = P1.distance(P2)

IV- Surcharge des méthodes

La surcharge (redéfinition) d'une méthode est le fait de réécrire dans une classe B une méthode d'une des classes dont B hérite.

Lorsque cette méthode sera appelée sur un objet de type B, alors c'est la méthode redéfinie qui sera utilisée et non la méthode de la classe mère.

Pour simplifier cette notion, on considère que la surcharge des méthodes consiste à modifier le comportement par défaut de certaines méthodes.

V- Surcharge des méthodes

- Les méthodes spéciales commencent et finissent par _ (deux underscores '_'). Elles sont héritées de la classe object.
- Les principales méthodes que l'on peut surcharger pour une classe donnée sont :

Méthode de la classe à surcharger	utilisation
object.__add__(self, other)	self+other
object.__sub__(self, other)	self-other
object.__mul__(self, other)	self*other
object.__div__(self, other)	self/other
object.__floordiv__(self, other)	self//other
object.__ne__(self, other)	self!=other
object.__le__(self, other)	self<=other
object.__lt__(self, other)	self < other
object.__ge__(self, other)	self>=other
object.__gt__(self, other)	self > other
object.__and__(self, other)	self and other
object.__or__(self, other)	self or other
object.__str__()	print(self) ;str(z)
object.__repr__()	repr(self)
object.__len__(self)	len(self)

Exemple :

Soit un point P :

P = Point (5, 6)

S = str(P) # On convertir notre objet en chaîne de caractères

print(S)

>> __main__.class Point object at 0x23A61356

Pour changer le résultat de conversion des objet de classe point il faut redéfinir la méthode str :

```
class Point:
    """ Classe représentant un Point"""

    def __init__(self, px , py):
        #  variables, uniques pour chaque objet (les attributs)
        self.x = px
        self.y = py

    def affichePoint(self):
        print(self.x , self.y)

    def distance(self, point):
        return sqrt((self.x-point.x)**2+(self.y-point.y)**2)

    def __str__(self):
        return "(" + str(self.x) +"," + str(self.y) +")"
```

Maintenant : print(P) donne >> (5, 6)