Programmation orienté objet avec Python

Les bases

Plan

- Comparaison entre la programmation procédurale et la programmation orientée objet
- Définition d'un objet
- Définition d'une classe
- Les attributs
- Les méthodes

Paradigmes de programmation

- Le mot paradigme = une représentation du monde, une manière de voir les choses, un modèle cohérent du monde qui repose sur un fondement défini

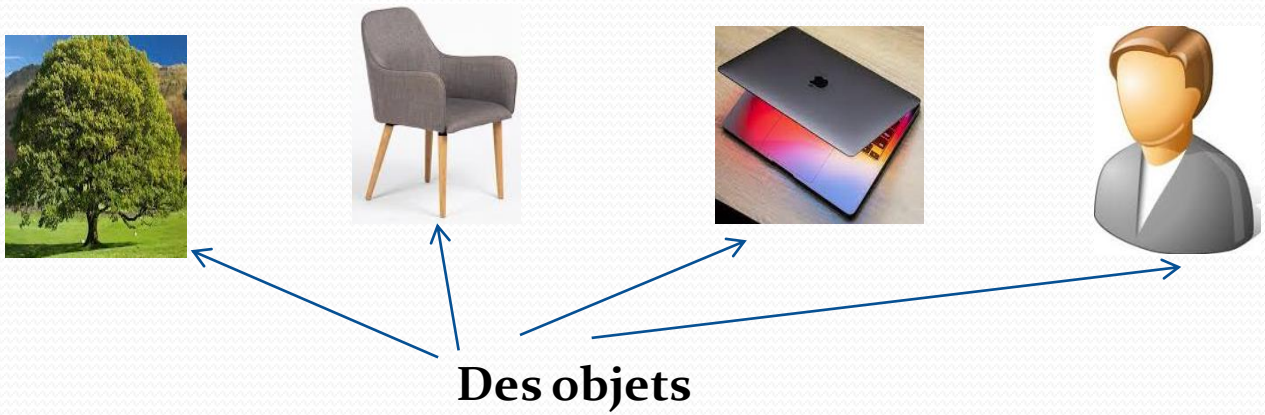
Source : The Guardian, Chronicle/Alamy

Source : Apollo 17 (1972)

- Les paradigmes en programmation est une façon de concevoir un code informatique. Ou un moyen de formuler un code informatique
- Par exemple , on parle de paradigme de la programmation procédurale, le paradigme de la programmation orientée objet et d'autres paradigmes

La programmation orientée objet

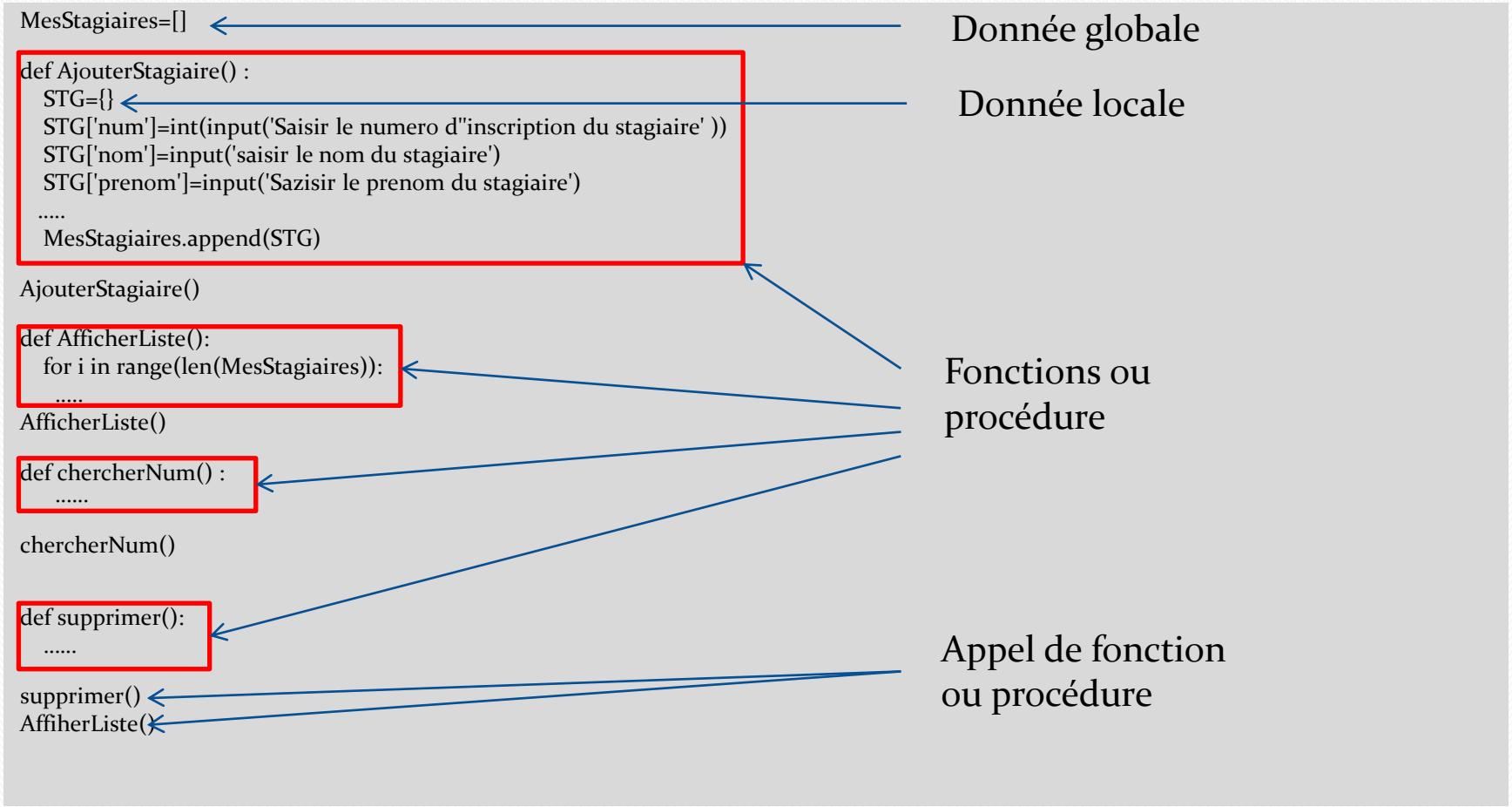
- Les premières personnes qui ont pensé à la Programmation orientée objet, ont remarqué que tous ce qui nous entoure sont des objets: un **arbre**, une **chaise**, un **ordinateur**, une **maison**, …. même une **personne** est considérée comme un objet. Ce qui a donné naissance à un nouveau paradigme de programmation basé sur les objets.
- La programmation orientée objet est donc un type de programmation basée sur la notion **d'objet.**
- Donc tout ce qu'on manipule dans la POO sont sous forme d'objets.



La programmation procédurale

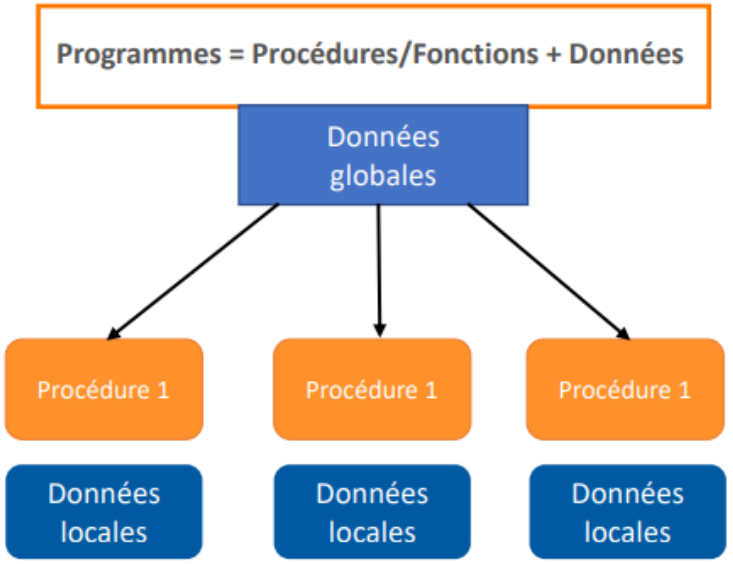
- Un programme procédural est un programme basé sur la notion de procédure c'est-à-dire les fonction. Dans ce type de programmation on met l'accent sur les **fonctions** puis les **données**.
- Par la suite une comparaison rapide entre les deux types de programmation

La programmation procédurale



La programmation procédurale

- Dans la programmation procédurale, le programme est divisé en des blocs d'instructions, appelés procédures ou fonctions.
- Ensuite, pour résoudre chaque problématique, une ou plusieurs procédures/fonctions sont utilisées
- Dans la programmation procédurale, **les données et le traitement de ces données sont séparées**



La programmation procédurale

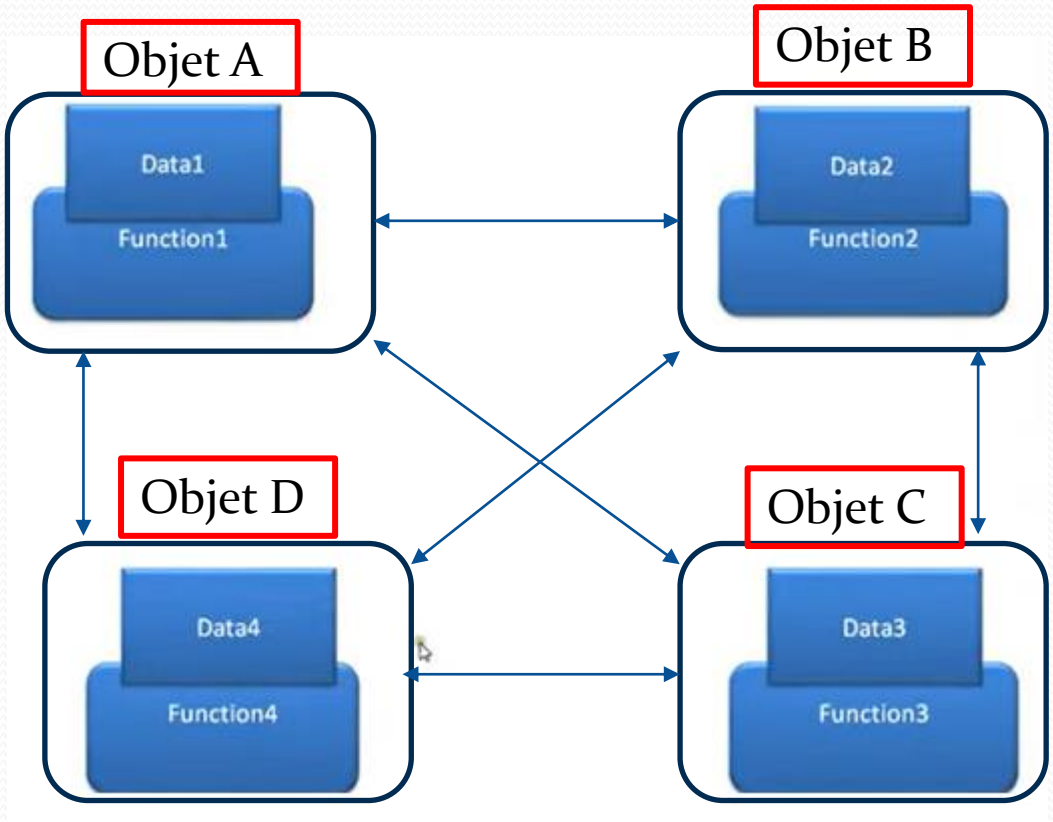
Inconvénient de la programmation procédurale :

lorsqu'il s'agit de résoudre des problèmes plus complexes, le code prend très rapidement de l'ampleur. Bien qu'il reste lisible, il perd en **clarté du fait de son volume**. Ce qui en produit,

- Difficulté de la **maintenance** de grandes applications
- Difficulté de **réutilisation** du code

Avantages de la POO

1. **Regroupement des données et les fonctions dans une même entité**

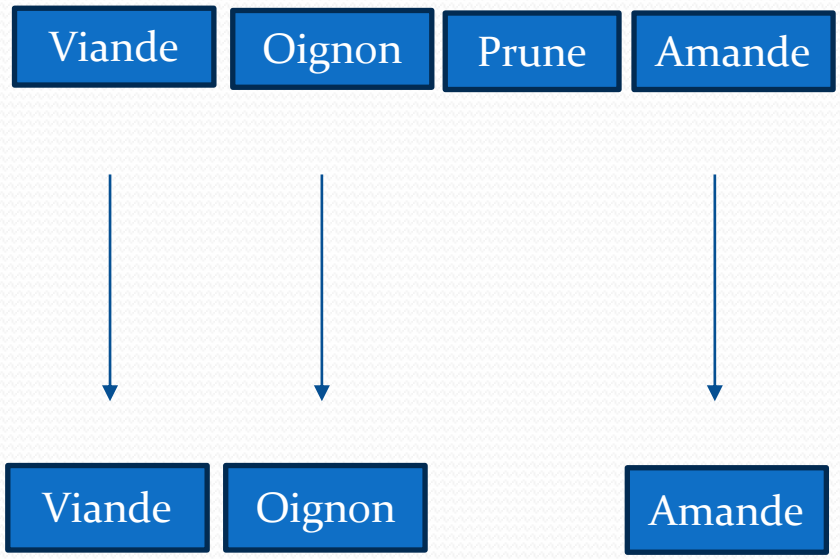


Avantages de la POO

2) La **réutilisabilité** du code:

Recette plat 1:
Viande
Oignon
Prunes
Amandes

Recette plat 2:
Viande
Oignon
Abricots
Amandes



Avantages de la POO

3) Extensibilité :

Recette plat 1:
Viande
Oignon
Prune
Amandes

Recette plat 2:
Viande
Oignon
Prune
Amandes
Œufs

La programmation orientée objet Historique

- Les concepts de la POO ont vu le jour vers la fin de années 1960.
- Les premiers langages de programmation véritablement orientés objet ont été .
  - **Simula** (1966) : a été le premier langage à **regrouper données et procédures.**
  - **Simula I** (1972) : a formalisé les **notions d'objet et de classe.**
- Un peu plus d'une décennie plus tard (80'), de nombreux langages basés sur la POO ont été publiés, tel que :
  - Eiffel créé par Bertrand Meyer
  - **C++** créé par Bjarne Stroustru
- Le 23 mai **1995** a marqué la naissance du célèbre langage **Java** par la société Sun Microsystems.
- De nos jour la majorité des langages sont orientés objet: PHP, c#, ruby, python …

Réflexion

- Il faut bien distinguer entre un **objet concret** et la **définition d'un objet.**
- Un objet concret c'est celui qu'on **utilise** (et peut être matériel ou non matériel; une chaise est matérielle mais une commande n'est pas matériel)
- La définition d'un objet c'est générale, par exemple lorsqu'on dit qu'une voiture est une machine qui possède 4 roues, un volant, 5 chaises, qui peut rouler à une grande vitesse, qui peut s'arrêter,... dans ce cas on est entrain de **définir ce que c'est une voiture** (comme une définition dans un dictionnaire), cette définition c'est ce qu'on appelle **une classe.**

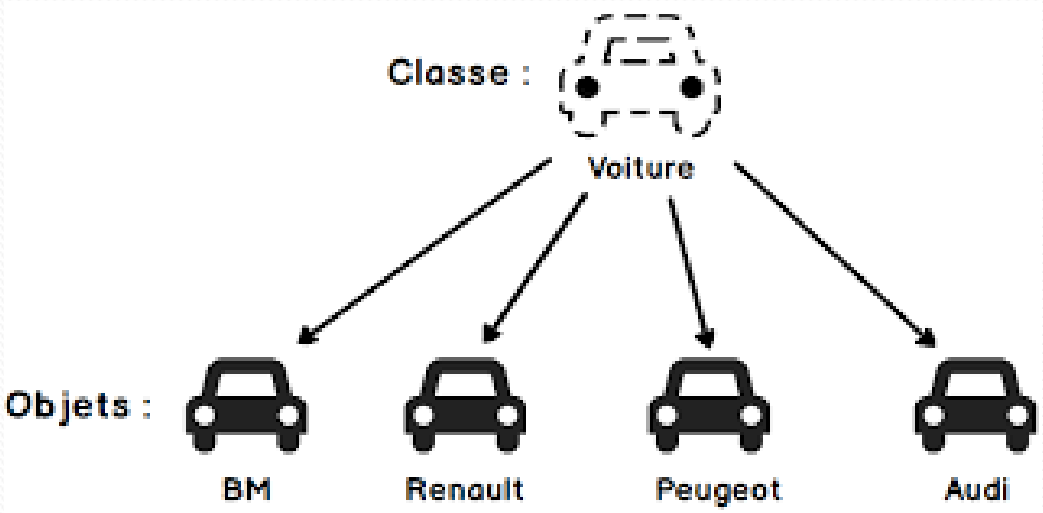
Définition d'une classe

- Une classe est considéré comme un **modèle** (ou un **moule**) à partir duquel vont être créés un ensemble d'objets.

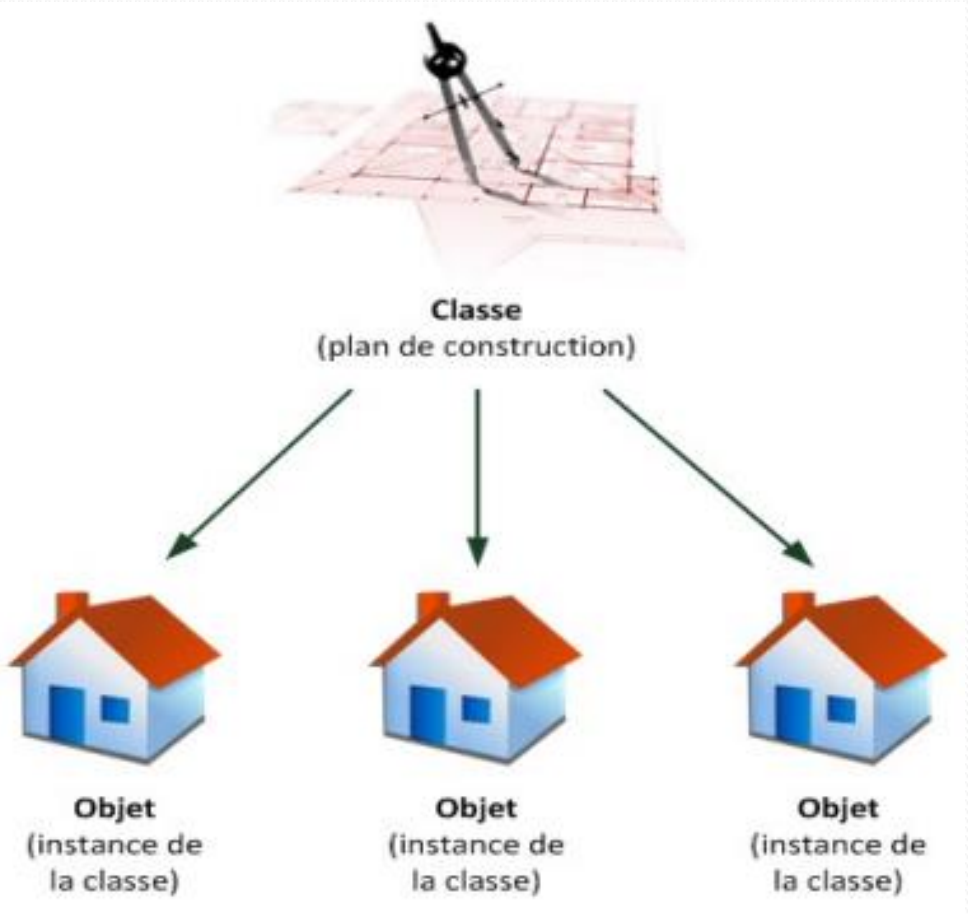
- Une classe peut être assimilée également à la notion de **type** que l'on voit dans les langages de programmation procédurales. (une classe est une sorte de type personnalisé)

Exemple de classe et d'objets

- Exemple :

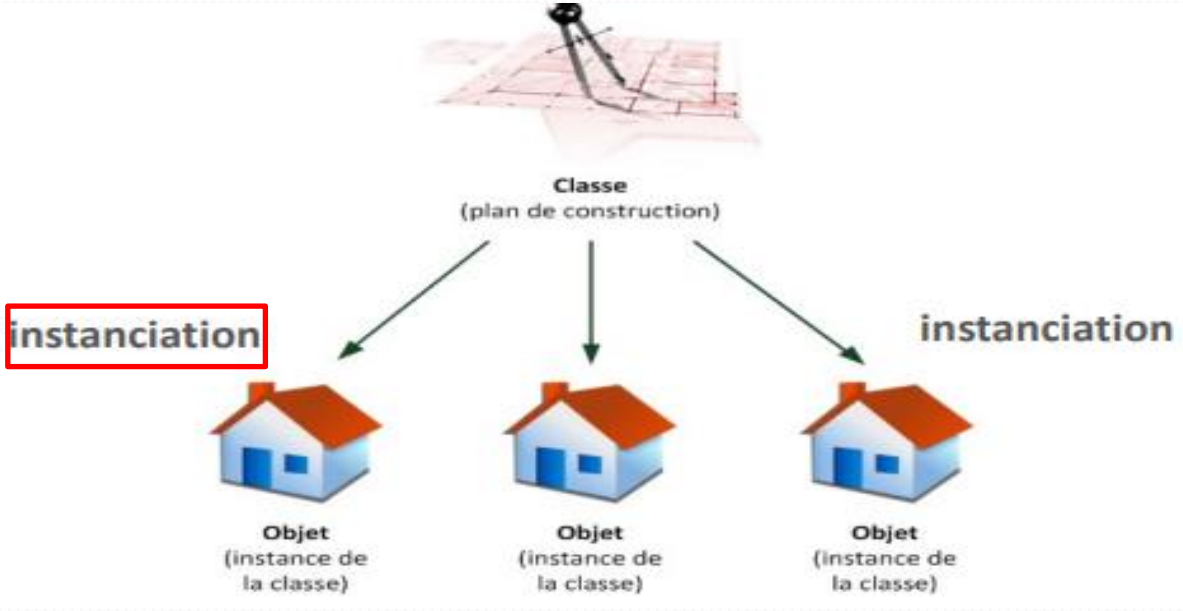


Exemples de classe et d'objets



Création d'un Objet

- Un **objet** est appelé aussi une **instance de classe**.
- La création d'un objet à partir d'une classe s'appelle **l'instanciation**



Qu'est ce que c'est un objet?

1. Un objet possède des **propriétés (ou attributs)** qui le **caractérisent**:
   - Par exemple une **chaise** est caractérisé par : 4 pieds, une **couleur**, une **forme**,...
   - Un **chat** est caractérisé par une **couleur**, un **poids**, un **âge**,...

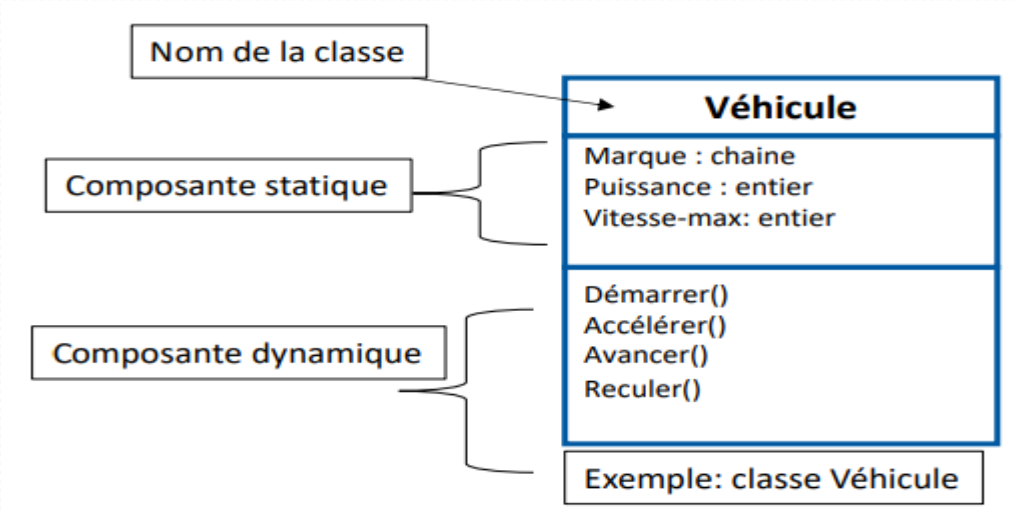
   Rq: deux objets de la même classe, ont les **mêmes propriétés mais des valeurs différentes pour chaque objet.**

2. Les objets peuvent faire des **actions** ou avoir des **comportements**:
   - Un **chat** peut **miauler**, **manger**, **sauter**,....
   - Une **voiture** peut **rouler**, **s'arrêter**, **klaxonner**

3. Les objets peuvent aussi **interagir entre eux:**
   - Un **conducteur** démarre la **voiture**
   - La **voiture** fait tourner l'objet **volant**

Modélisation d'une classe

- Une classe est caractérisée par :
  - **Un nom**
  - Une composante statique : des **champs** (ou **attributs**). Ce sont **les propriétés** qui **caractérisent l'état des objets** pendant l'exécution du programme
  - Une composante dynamique : des **méthodes** représentant le **comportement (ou les actions)** des objets de cette classe. Elles manipulent les champs des objets et caractérisent les actions pouvant être effectuées par les objets



Création d'une classe en python

- Voici la syntaxe pour définir une classe :

```
class NomDeLaClasse :

    // Donnees et operations ici
```

```
class Voiture :
    pass
```

- **Remarque**: le mot pass de python est un mot spécial qui **ne fait rien,** il est utilisé lorsque le contenu du bloc n'est pas encore définit **.** Son rôle est de ne pas générer une erreur, car on ne peut pas créer une classe qui ne contient rien.

Comment instancier un nouvel objet

- Dans l'exemple précédent on a créé une classe voiture encore vide. Mais on peut créer un objet **MaVoiture comme** suit:

```
NomObjet=NomClasse()

Exemple:
MaVoiture=Voiture()
```

- MaVoiture est une **instance** de classe Voiture()
- MaVoiture est un **objet**.

Le contenue d'une classe

- Une classe python contient généralement:

1. Des méthodes
2. Des constructeur

Les attributs d'instance

- Un **attribut** ou **champ** est une variable spéciale qui a comme objectif de **stocker l'état d'un objet**.
- Par exemple, les attributs d'une voiture peuvent être:
  - Une matricule, une marque, une couleur, une puissance,…
- Un attribut peut être de type :
  - simple: entier, réel, chaine de caractères, caractère, etc
  - complexe : comme une liste
  - Objet de type classe: Etudiant, Voiture, etc

Accès aux attributs

- Pour accéder à un attribut d'un objet on indique le nom de la référence de l'objet suivi par le nom de l'attribut dans l'objet de la manière suivante :

nomObjet . nomAttribut

  - **nomObjet** : nom de de la référence à l'objet
  - **nomAttribut** = nom de l'attribut

Définir les attributs d'un objet

- Comment définir ces attribut dans une classe?

```
class Voiture :
    pass

#instance
MaVoiture=Voiture()

# on affecte des valeurs à chaque attribut de l'objet MaVoiture
MaVoiture.matricule="AB1254"
MaVoiture.marque="Nissan"
MaVoiture.couleur="Noir"

print (MaVoiture.matricule)          # afficher la matricule de MaVoiture
```

Supprimer un attribut

- On peut supprimer un attribut grâce au mot clé **del** de la manière suivante:

```
del  MaVoiture . couleur
```

Les méthodes d'instance

- Une méthode d'instance représente un **comportement** d'un objet ou une **action** qui peut être réalisée par un objet:
  - Une voiture peut rouler, klaxonner, s'arrêter..
- Une méthode (d'instance), en pratique est une fonction définie à l'intérieur d'une classe et qui possède un paramètre obligatoire (**self**)

```
class Voiture :

   def  Infos(self) :
        print("Les informations de mon objet :", self.matricule," ,self.marque," ",self.couleur )
```

Accéder à une méthode d'instance

- Pour accéder à une méthode d'instance, on utilise la syntaxe suivante:
- Exemple:

NomObjet . NomMethode()

```
class Voiture :

   def  Infos(self) :
      print("Les informations de mon objets :", self.matricule," ",self.marque," ",self.couleur )
#instance
MaVoiture=Voiture()
# on affecte des valeurs à chaque attribut de l'objet MaVoiture
MaVoiture.matricule="AB1254"
MaVoiture.marque="Nissan"
MaVoiture.couleur="Noir"

MaVoiture.Infos()
```

Le mot self

- Le mot **self** est le nom du paramètre qui correspond à **l'instance depuis laquelle la méthode est appelée** dans la programmation orientée objet.

```
MaVoiture1=Voiture()
# on affecte des valeurs à chaque attribut de l'objet MaVoiture
MaVoiture1.matricule="AB1254"
MaVoiture1.marque="Nissan"
MaVoiture1.couleur="Noir"

MaVoiture1.Infos()          #self fait référence à MaVoiture1

MaVoiture2=Voiture()
# on affecte des valeurs à chaque attribut de l'objet MaVoiture
MaVoiture2.matricule="NF547"
MaVoiture2.marque="Ford"
MaVoiture2.couleur="Rouge"

MaVoiture2.Infos()          #self fait référence à MaVoiture2
```

Appel d'une méthode dans une autre

- pour appeler une méthode dans une méthode il suffit de faire :

```
Self . method(arguments)
```

sans avoir à passer *self* puisque l'instance *self* va le faire elle même.

Exemple:

```
class Rectangle():
    ...
    def surfBase(self):
        return self.largeur * self.longueur

    def volume(self):
        return self.hauteur * self.surfBase()
```