POO en Python

Syntaxe de Base

Partie 1/3

Anis SAIED

IPEIN

2025/26 v1.0

Plan de la séance

Introduction

Objectif d'étudier la programmation orientée objets (POO) en Python :

- Comprendre comment les types en Python fonctionnent
- Créer de nouveaux types (**classes**)
- Reutiliser le code grâce à l'**héritage** et la **composition**
- Maîtriser les concepts d'**encapsulation** et le **polymorphisme**

Introduction

Exemple : Comprendre les types en Python

```
x = 2
print(type(x)) # <class 'int'>

y = 3.14
print(type(y)) # <class 'float'>

z = "Bonjour"
print(type(z)) # <class 'str'>

a = [1, 2, 3]
print(type(a)) # <class 'list'>
```

Introduction aux objets en Python

En Python, toutes les variables (entiers, chaînes, listes, etc.) sont des instances (objets) d'une classe spécifique.

Exemple :

```
x = 2
print(isinstance(x, int)); type(x)==int  # True

y = "Bonjour"
print(isinstance(y, str))  # True

a = [1, 2, 3]
print(isinstance(a, list))  # True
```

Grâce aux **classes**, nous pouvons créer des types de données plus complexes.

Qu'est-ce qu'une classe en programmation ?

- Une **classe** est une structure de données qui définit un *modèle* pour créer des objets.
- Une classe encapsule des *données* (appelées **attributs**) et des *comportements* (appelés **méthodes**) liés à ces données.
- Elle permet de créer des **objets** réutilisables et d'organiser le code de manière modulaire.

Exemple :

- Une classe "Voiture" peut avoir des attributs tels que "marque" et "année".
- Elle peut également avoir des méthodes pour démarrer, arrêter et accélérer la voiture.

Qu'est-ce qu'un objet en programmation ?

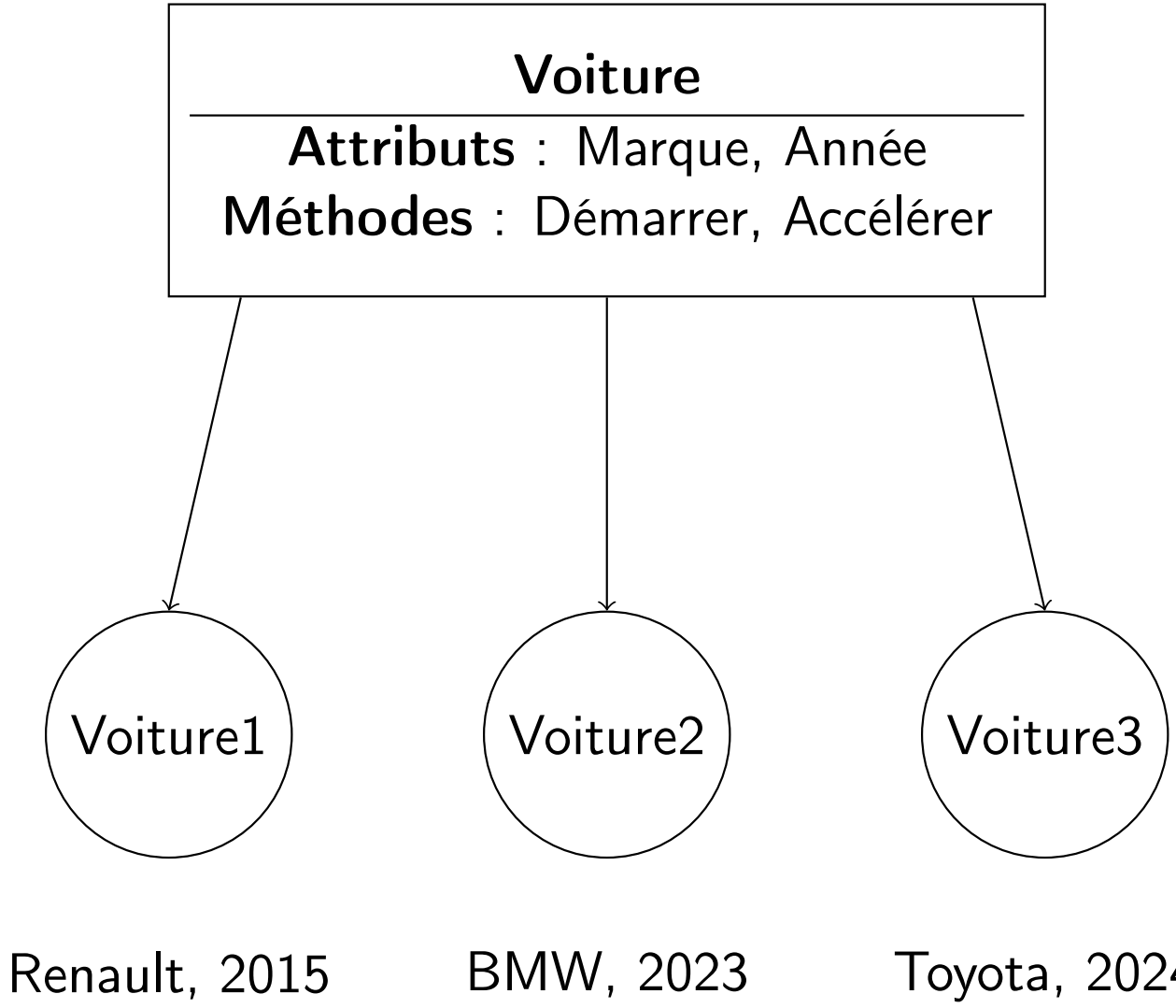
Un **objet** est une instance concrète créée à partir d'une classe.

Exemple :
Supposez que nous ayons une classe "Voiture" avec des attributs comme "marque" et "année".

- Un objet spécifique créé à partir de cette classe pourrait être une "Peugeot 1980".
- Cet objet aurait des valeurs spécifiques pour les attributs "marque" et "année".

Classe & Objtes

Exemples



Comment créer une classe en Python ?

- Pour créer une classe en Python, utilisez le mot-clé **class**.
- Voici la syntaxe de base :

```
1 class Voiture:
2     # Attributs : Marque, Année
3     # Méthodes: Démarrer, Accélérer
```

Comment créer un objet ?

- Pour créer un objet à partir d'une classe, utilisez la syntaxe suivante :

```
1 # Création d'un objet à partir d'une classe
2 nom_objet = NomDeLaClasse()
3 v1 = Voiture()
```

- Remplacez 'NomDeLaClasse' par le nom de la classe à partir de laquelle vous souhaitez créer un objet.

Exemple de création de classe et d'objet

```
1 class Voiture:
2     def __init__(self):
3         self.marque = ""
4         self.annee = 0
5
6 # Création d'un objet de la classe Voiture
7 v1 = Voiture()
```

- Dans cet exemple, nous avons créé une **classe** *Voiture* avec deux attributs *marque* et *année* (associés au mot clé **self**).
- Nous avons également créé un **objet** (instance) *v1* de cette classe.
- En Python, **self** est un paramètre spécial utilisé pour faire référence à l'instance actuelle de la classe.

L'utilisation de `self`

En Python, **self**

- fait référence à l'instance actuelle de la classe.
- utilisé pour accéder aux attributs d'instance et aux méthodes de l'objet en cours.
- est le premier paramètre dans la liste des paramètres de méthode.
- n'est pas un mot réservé en Python, mais il est largement utilisé par convention.

Note : Dans une méthode (fonction définie dans une classe)

- une variable associée à self est un **attribut d'instance**.
- une variable non associée à self est une **variable locale**

Attributs

Les attributs peuvent être de deux types principaux :

Les attributs d'instance

- Chaque objet (instance) de la classe possède sa propre copie de ces attributs.
- Définis dans le **constructeur** (`__init__`) de la classe.

Les attributs de classe

- Partagés par toutes les instances de la classe.
- Définis à l'intérieur de la classe, mais en dehors des méthodes.

Attributs d'Instance

- Spécifiques à chaque objet (instance) de la classe.
- Définis dans le **constructeur** (__init__) de la classe.
- Accessibles via ***self*** à l'intérieur de la classe et via le nom de l'instance en dehors de la classe.

```
1  class Voiture:
2      def __init__(self, marque, annee):
3          self.marque = marque
4          self.annee = annee
5
6  # Exemple d'utilisation
7  v1 = Voiture("Toyota", 2024)
8  v2 = Voiture("Renault", 2015)
9  print(v1.marque) # Affiche Toyota
10 print(v2.marque) # Affiche Renault
```

Attributs de Classe

- Partagés par toutes les instances de la classe.
- Définis à l'intérieur de la classe, mais en dehors des méthodes.
- Accessibles via le nom de la classe depuis n'importe où.

```
1  class Voiture:
2      compteur = 0 # Attribut de classe
3      def __init__(self, marque, annee):
4          self.marque = marque
5          self.annee = annee
6          Voiture.compteur += 1
7
8  # Exemple d'utilisation
9  v1 = Voiture("Toyota", 2024)
10 v2 = Voiture("Renault", 2015)
11
12 #Affiche le nombre total de voitures créés
13 print(Voiture.compteur)
```

Attributs d'Instance Additionnels

- Parfois, il est nécessaire d'ajouter à certaines instances des attributs spécifiques qui ne sont pas définis dans le constructeur (`__init__`).
- Exemple : Vous avez une classe "Voiture" avec des attributs de base, mais pour certaines voitures spécifiques, vous devez ajouter des attributs personnalisés tels que "couleur" ou "options".

```
1  class Voiture:
2      def __init__(self, marque, annee):
3          self.marque = marque
4          self.annee = annee
5
6  # Création d'une voiture standard
7  voiture1 = Voiture("Toyota", 2020)
8  voiture2 = Voiture("Honda", 2022)
9  # Attributs supplémentaires
10 voiture2.couleur = "Rouge"
11 voiture2.options = ["Toit ouvrant", "Sièges en cuir"]
```

Méthodes

Syntax

Une **méthode** est une fonction définie dans une classe, dont sa signature commence *self*.

```
1 class MaClasse:
2     def ma_methode(self, parametres):
3         # Corps de la méthode
```

- 'self' : premier paramètre de méthode, faisant référence à l'instance actuelle de la classe sur laquelle la méthode est appliquée .
  Exemple : `v1.__init__("Toyota", 2020)`
- Vous pouvez définir des méthodes avec des paramètres comme toute autre fonction.

Méthodes

Exemple

```python
class Voiture:
    def __init__(self, marque, annee):
        self.marque = marque
        self.annee = annee
    def afficher(self):
        print(f"Marque:{self.marque}, année :{self.annee}")

# Création d'un objet de la classe Voiture
voiture1 = Voiture("Toyota", 2020)
voiture1.afficher()  # Appel de la méthode
```

Appeler une méthode sur un objet :

- **objet.méthode(paramètres)** → `v1.afficher()`
- **classe.méthode(objet,paramètres)** → `voiture.afficher(v1)`

`__init__()`

Constructeur

La méthode `__init__()` est le **constructeur** d'une classe en Python.

- Automatiquement appelée lors de la création d'un objet à partir de la classe.
- Permet d'initialiser les attributs de l'objet.

Syntaxe :

```
class MaClasse:
    def __init__(self, parametres):
        # Initialiser les attributs par les paramètres
```

Exemple :

```
class Voiture():
    def __init__(self, marque, annee):
        self.marque = marque
        self.annee = annee

# Création et initialisation d'un objet de la classe Voiture
voiture1 = Voiture("Toyota", 2020)
```

`__del__()`

Destructeur

La méthode `__del__()` est le **destructeur** d'une classe en Python.

- Automatiquement appelée lorsque l'objet est détruit ou n'est plus référencé.
- Permet de nettoyer les ressources associées à l'objet.

Syntaxe :

```
1 class MaClasse:
2     def __del__(self):
3         # Code de nettoyage ici
```

__del__()

Destructeur

Exemple :

```
1  class Voiture:
2      def __init__(self, marque):
3          self.marque = marque
4      def __del__(self):
5          print(f"La voiture {self} a été détruite.")
6
7  # Création d'un objet de la classe Voiture
8  voiture1 = Voiture("Toyota")
9
10 # Destruction de l'objet
11 del voiture1
12 # La voiture <__main__.Voiture object at 0x7f085e8d8560>
13 # a été détruite.
```

Exercices

Exercice 1

1. Créez une classe **Rectangle** avec des attributs **longueur** et **largeur** (attributs d'instance).
2. Ajoutez une méthode **calculer_surface** qui calcule et retourne la surface du rectangle.
3. Créez ensuite deux objets de cette classe :
   - Rectangle 1 : longueur = 5, largeur = 3
   - Rectangle 2 : longueur = 4, largeur = 6
4. Calculez et affichez la surface pour chaque objet.

Solution Exercice 1

```
1  class Rectangle:
2      def __init__(self,larg=0, long=0):
3          self.long = long
4          self.larg = larg
5      def calcluler_surface(self):
6          return self.long * self.larg
7
8  r1 = Rectangle(4,6)
9  r2 = Rectangle(3,5)
10 r3 = Rectangle()
11
12 #meth 1 : nom_classe.nom_methode(obj, parametres)
13 Rectangle.calcluler_surface(r1) # list.pop(l)
14
15 #meth 2 : nom_objet.nom_methode(parametres)
16 r1.calcluler_surface() # l.pop()
```

Exercices

Exercice 2

1. Créez une classe **Point** avec trois attributs **x**, **y** et **nom** pour représenter des coordonnées et le nom d'un point.
2. Instanciez plusieurs objets de type *Point* avec différentes coordonnées et stockez-les dans une liste.
   Exemple:
   - A = Point(1, 2,"A")
   - B = Point(3, 4,"B")
   - C = Point(0, 0,"C")
   - L = [A, B, C]
3. Créer une fonction **calculer_distance(p1, p2)** qui permet de calculer la distance entre deux points **p1** et **p2**.
4. Calculer et afficher la distance entre chaque paire de points dans la liste.
   Exemple:
   - Distance entre A et B : 2.828
   - Distance entre A et C : 2.236
   - Distance entre B et C : 4.472

Solution Exercice 2

```
1  from math import *
2  from random import *
3  class Point:
4   def __init__(self, x, y):
5       self.x, self.y = x, y
6  # Création des objets Point
7  n = randint(5,10); L=[]
8  for i in range(n):
9      x, y, nom=uniform(100), uniform(100), 'P'+str(i)
10     L.append(Point(x, y, nom))
11 def calculer_distance(p1, p2):
12     return sqrt((p1.x - p2.x)**2+ (p1.y - p2.y)**2)
13 # Affichage des distances
14 ch = "Distance entre {} et {} : {}"
15 for i in range(n-1):
16     for j in range(i+1, n):
17         d = calculer_distance(L[i], L[j])
18         print(ch.format(L[i].nom, L[j].nom, d))
```

C'est qu'on doit retenir

- Définir un nouveau type c'est définir une nouvelle **classe**.
- Une **classe** est un modèle pour créer des objets, définissant à la fois des attributs (données) et des méthodes (comportements).
- On peut créer des **instances** (**objets**) à partir d'une classe.
- Les **attributs** peuvent être de trois types : de classe, d'instance (**self.attribut**), ou associés à l'objet après son instanciation.
- Les **méthodes** sont des fonctions définies dans une classe pour effectuer des opérations sur les objets.
- Le **constructeur (__init__)** est une méthode spéciale pour initialiser les attributs lors de la création d'un objet.
- Le **destructeur (__del__)** est une méthode spéciale pour nettoyer les ressources associées à un objet lors de sa destruction.

Travail à faire

TD1 POO :

- Exercice 1
- Exercice 4