

Syntaxe de Base de la POO en Python

Anis SAIED

12 octobre 2023

Plan de la séance

- 1 Introduction aux classes et aux objets
- 2 Création de classes et d'objets
- 3 Attributs et méthodes
- 4 Constructeurs et destructeurs
- 5 Exercices
- 6 A Retenir

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

- En programmation, une classe est une structure de données fondamentale qui définit un modèle ou un plan pour créer des objets.
- Une classe encapsule des données (appelées attributs) et des comportements (appelés méthodes) liés à ces données.
- Elle permet de regrouper des fonctionnalités similaires, 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 ?

- En programmation, un objet est une instance concrète créée à partir d'une classe.
- Un objet est une entité qui encapsule des données (attributs) et des comportements (méthodes) définis par sa classe parente.

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 "Toyota Camry 2022".
- Cet objet aurait des valeurs spécifiques pour les attributs "marque" et "année".

Comment créer une classe en Python ?

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

```
1 class MaClasse:  
2     # Attributs et méthodes de la classe
```

- Vous pouvez définir vos attributs et méthodes à l'intérieur de la classe.

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()
```

- 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 Personne:
2     def __init__(self):
3         self.nom = "Ahmed Ben Mahmoud"
4
5 # Création d'un objet de la classe Personne
6 personne1 = Personne()
```

- Dans cet exemple, nous avons créé une classe "Personne" avec un attribut "nom" (celui associé au mot clé 'self').
- Nous avons également créé un objet (instance) "personne1" de cette classe.
- En Python, 'self' est un paramètre spécial utilisé pour faire référence à l'instance actuelle de la classe.

Attributs

- Les attributs sont des variables associées à une classe.
- Ils permettent de stocker des données spécifiques à une classe ou à ses objets.
- Les attributs peuvent être de deux types principaux :
 - Les attributs de classe : Ils sont partagés par toutes les instances de la classe. Définis à l'intérieur de la classe, mais en dehors des méthodes.
 - Les attributs d'instance : Chaque objet (instance) de la classe possède sa propre copie de ces attributs. Définis dans le constructeur de la classe.

Attributs de Classe

- Les attributs de classe sont partagés par toutes les instances de la classe.
- Ils sont définis à l'intérieur de la classe, mais en dehors des méthodes.
- Et accessibles via le nom de la classe depuis n'importe où.

```
1 class Voiture:
2     compteur = 0
3     # Constructeur
4     def __init__(self):
5         Voiture.compteur += 1
6
7 # Exemple d'utilisation
8 voiture1 = Voiture()
9 voiture2 = Voiture()
10 print(Voiture.compteur) # Affiche le nombre total de voitures
```

Attributs d'Instance

- Les attributs d'instance sont spécifiques à chaque objet (instance) de la classe.
- Ils sont définis dans le constructeur de la classe.
- Les attributs d'instance sont accessibles via *self* à l'intérieur de la classe et via le nom de l'instance en dehors de la classe.

```
1 class Personne:
2     def __init__(self, nom):
3         self.nom = nom # attribut d'instance
4
5 # Exemple d'utilisation
6 personne1 = Personne("John Doe")
7 personne2 = Personne("Jane Smith")
8 print(personne1.nom) # Affiche le nom de la première personne
9 print(personne2.nom) # Affiche le nom de la deuxième personne
```

L'utilisation de self

- En Python, 'self' est un paramètre spécial utilisé pour faire référence à l'instance actuelle de la classe.
- Il est utilisé pour accéder aux attributs d'instance et aux méthodes de l'objet en cours.
- Lorsque vous définissez des méthodes dans une classe, 'self' est généralement le premier paramètre dans la liste des paramètres de méthode.
- Il est important d'utiliser 'self' pour différencier les attributs d'instance des variables locales.
- 'self' n'est pas un mot réservé en Python, mais il est largement utilisé par convention.

Attributs d'Instance Additionnels

- Parfois, il est nécessaire d'ajouter des attributs spécifiques à certaines instances 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"]
12
13 # Les attributs personnalisés ne sont pas définis pour toutes les voitures
```

Méthodes en Python - Syntaxe

- Les méthodes sont des fonctions spécifiques à une classe.
- Elles sont définies dans une classe en utilisant la syntaxe suivante :

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

- 'self' est le premier paramètre de méthode, faisant référence à l'instance actuelle de la classe.
- Vous pouvez définir des méthodes avec des paramètres comme toute autre fonction.

Exemple de Méthode en Python

```
1 class Personne:
2     def __init__(self, nom):
3         self.nom = nom
4
5     def afficher_nom(self):
6         print("Nom de la personne :", self.nom)
7
8 # Création d'un objet de la classe Personne
9 personne1 = Personne("John Doe")
10 personne1.afficher_nom() # Appel de la méthode
```

- Dans cet exemple, nous avons défini une méthode 'afficher_nom()' dans la classe 'Personne'.
- La méthode est appelée sur un objet de la classe avec la syntaxe **objet.méthode(param)** (sauf le paramètre 'self') pour afficher le nom de la personne.

Exercice d'Application

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

La Méthode `__init__()` (Constructeur)

- La méthode `__init__()` est le constructeur d'une classe en Python.
- Elle est automatiquement appelée lors de la création d'un objet à partir de la classe.
- Le constructeur permet d'initialiser les attributs de l'objet.
- Voici la syntaxe de base pour définir le constructeur :

```

1 class MaClasse:
2     def __init__(self, parametres):
3         # Initialisation des attributs ici

```

• Exemple :

```

1 class Personne:
2     def __init__(self, nom, age):
3         self.nom = nom
4         self.age = age
5
6 # Création d'un objet de la classe Personne
7 personnel = Personne("Alice", 25)

```


La Méthode `__del__()` (Destructeur)

- La méthode `__del__()` est le destructeur d'une classe en Python.
- Elle est automatiquement appelée lorsque l'objet est détruit ou n'est plus référencé.
- Le destructeur permet de nettoyer les ressources associées à l'objet.
- Voici la syntaxe de base pour définir le destructeur :

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

Exemple de Destructeur en Python

- Exemple :

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

Exercice 1 : Créez une classe Voiture

- Créez une classe Python appelée Voiture.
- Ajoutez un attribut marque à la classe.
- Ajoutez une méthode demarrer() qui affiche "La voiture démarre."
- Instanciez un objet de la classe et appelez la méthode demarrer().
- Ajoutez une méthode arreter() à la classe Voiture.
- La méthode doit afficher "La voiture s'arrête."
- Appelez la méthode pour arrêter la voiture.

Exercice 2 : Mathématique avec la POO en Python

- Exercice :

- ❶ Créez une classe 'Point' avec trois attributs x, y et nom pour représenter des coordonnées et le nom d'un point.
- ❷ Instanciez plusieurs objets de type "Point" avec différentes coordonnées et stockez-les dans une liste.
- ❸ Calculez la distance entre chaque paire de points dans la liste.
- ❹ Affichez les distances calculées.

- Exemple :

- Point A(1, 2)
- Point B(3, 4)
- Point C(0, 0)
- Calcul des distances :
 - Distance entre A et B : 2.828
 - Distance entre A et C : 2.236
 - Distance entre B et C : 4.472

Solution Exercise 2 I

```
1 import math
2
3 class Point:
4     def __init__(self, x, y):
5         self.x = x
6         self.y = y
7
8 # Création des objets Point
9 pointA = Point(1, 2)
10 pointB = Point(3, 4)
11 pointC = Point(0, 0)
12
13 # Calcul des distances
14 def distance_entre_points(point1, point2):
15     return math.sqrt((point1.x - point2.x)**2 \
16                     + (point1.y - point2.y)**2)
17
18 distance_AB = distance_entre_points(pointA, pointB)
19 distance_AC = distance_entre_points(pointA, pointC)
20 distance_BC = distance_entre_points(pointB, pointC)
```

Solution Exercice 2 II

```
21
22 # Affichage des distances
23 print("Distance entre A et B :", distance_AB)
24 print("Distance entre A et C :", distance_AC)
25 print("Distance entre B et C :", distance_BC)
```

Exercice3 : Utilisation de Dictionnaires

- Créez une classe "Etudiant" avec des attributs "nom", "note_math" et "note_physique" pour représenter les étudiants et leurs notes.
- Instanciez plusieurs objets "Etudiant" avec différents noms et notes, et stockez-les dans un dictionnaire où la clé est le nom de l'étudiant.
- Calculez la moyenne des notes en mathématiques et en physique pour tous les étudiants.
- Affichez les noms des étudiants avec leurs notes et la moyenne.

Solution Exercise 3 I

```
1 class Etudiant:
2     def __init__(self, nom, note_math, note_physique):
3         self.nom = nom
4         self.note_math = note_math
5         self.note_physique = note_physique
6
7 # Création des objets Etudiant
8 etudiant1 = Etudiant("Alice", 90, 85)
9 etudiant2 = Etudiant("Bob", 78, 92)
10 etudiant3 = Etudiant("Charlie", 88, 76)
11 # Stockage des étudiants dans un dictionnaire
12 etudiants = {
13     etudiant1.nom: etudiant1,
14     etudiant2.nom: etudiant2,
15     etudiant3.nom: etudiant3
16 }
17
18 # Calcul de la moyenne des notes
19 def calculer_moyenne(etudiants):
20     total_math = sum(e.note_math for e in etudiants.values())
```


Solution Exercise 3 II

```
21     total_physique = sum(e.note_physique for e in etudiants.values())
22     nombre_etudiants = len(etudiants)
23     moyenne_math = total_math / nombre_etudiants
24     moyenne_physique = total_physique / nombre_etudiants
25     return moyenne_math, moyenne_physique
26
27 moyenne_math, moyenne_physique = calculer_moyenne(etudiants)
28
29 # Affichage des résultats
30 print("Moyenne en Mathématiques :", moyenne_math)
31 print("Moyenne en Physique :", moyenne_physique)
32 for nom, e in etudiants.items():
33     print("{} - Math : {}, Phys : {}".format(e.nom, e.note_math, e.note_physique))
```

C'est qu'on doit retenir

- La Programmation Orientée Objet (POO) est un paradigme de programmation basé sur le concept d'objets.
- Une classe est un modèle pour créer des objets, définissant à la fois des attributs (données) et des méthodes (comportements).
- Les objets sont des instances de classes, créés à partir de la classe.
- Les attributs peuvent être de trois types : de classe, d'instance, ou associés à l'objet après son instantiation.
- 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.