Syntaxe de Base de la POO en Python

Anis SAIED

9 octobre 2023

Plan de la séance

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 :

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

- Vous pouvez définir vos attributs et méthodes à l'intérieur de la classe.
- Par exemple, créons une classe simple appelée "Personne" avec un attribut "nom".

Comment créer un objet ?

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

```
# Création d'un objet à partir d'une classe
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

```
class Personne:
    def __init__(self):
        self.nom = "Ahmed Ben Mahmoud"

# Création d'un objet de la classe Personne
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.

```
class Voiture:
    compteur = 0
    # Constructeur
    def __init__(self):
        Voiture.compteur += 1

# Exemple d'utilisation
voiture1 = Voiture("Toyota")
voiture2 = Voiture("Honda")
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.

```
class Personne:
    def __init__(self, nom):
        self.nom = nom # attribut d'instance

# Exemple d'utilisation
personne1 = Personne("John Doe")
personne2 = Personne("Jane Smith")
print(personne1.nom)  # Affiche le nom de la première personne
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.

Méthodes en Python - Syntaxe

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

```
class MaClasse:
    def ma_methode(self, parametres):
        # 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

```
class Personne:
    def __init__(self, nom):
        self.nom = nom

    def afficher_nom(self):
        print("Nom de la personne :", self.nom)

# Création d'un objet de la classe Personne
personne1 = Personne("John Doe")
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

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 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 :

```
class MaClasse:
    def __init__(self, parametres):
        # Initialisation des attributs ici
```

- Exemple :

```
class Personne:
    def __init__(self, nom, age):
        self.nom = nom
        self.age = age

# Création d'un objet de la classe Personne
personne1 = 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 :

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

Exemple de Destructeur en Python

- Exemple :

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

    def __del__(self):
        print("La voiture de marque "+ self.marque + " a été dé

# Création d'un objet de la classe Voiture
voiture1 = Voiture("Toyota")

# Destruction de l'objet
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 :
  1. Créez une classe "Point" avec des attributs x et y pour représenter des coordonnées.
  2. Instanciez plusieurs objets "Point" avec différentes coordonnées et stockez-les dans une liste.
  3. Calculez la distance entre chaque paire de points dans la liste.
  4. 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 (Partie 1)

```python
import math

class Point:
    def __init__(self, x, y):
        self.x = x
        self.y = y

# Création des objets Point
pointA = Point(1, 2)
pointB = Point(3, 4)
pointC = Point(0, 0)
```

Solution (Partie 2)

```
# Calcul des distances
def distance_entre_points(point1, point2):
    return math.sqrt((point1.x - point2.x)**2 \
           + (point1.y - point2.y)**2)

distance_AB = distance_entre_points(pointA, pointB)
distance_AC = distance_entre_points(pointA, pointC)
distance_BC = distance_entre_points(pointB, pointC)

# Affichage des distances
print("Distance entre A et B :", distance_AB)
print("Distance entre A et C :", distance_AC)
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 (Partie 1)

```
class Etudiant:
    def __init__(self, nom, note_math, note_physique):
        self.nom = nom
        self.note_math = note_math
        self.note_physique = note_physique

# Création des objets Etudiant
etudiant1 = Etudiant("Alice", 90, 85)
etudiant2 = Etudiant("Bob", 78, 92)
etudiant3 = Etudiant("Charlie", 88, 76)
# Stockage des étudiants dans un dictionnaire
etudiants = {
    etudiant1.nom: etudiant1,
    etudiant2.nom: etudiant2,
    etudiant3.nom: etudiant3
}
```

Solution (Partie 2)

```
# Calcul de la moyenne des notes
def calculer_moyenne(etudiants):
    total_math = sum(etudiant.note_math for etudiant in \
                     etudiants.values())
    total_physique =sum(etudiant.note_physique for etudiant \
                        in etudiants.values())
    nombre_etudiants = len(etudiants)
    moyenne_math = total_math / nombre_etudiants
    moyenne_physique = total_physique / nombre_etudiants
    return moyenne_math, moyenne_physique

moyenne_math, moyenne_physique = calculer_moyenne(etudiants)
```

Solution (Partie 3)

```
# Affichage des résultats
print("Moyenne en Mathématiques :", moyenne_math)
print("Moyenne en Physique :", moyenne_physique)
for nom, e in etudiants.items():
    print("{} - Math : {}, Phys :{}"\
    .format(e.nom,e.note_math,e.note_physique))
```

Résumons

- 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 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.