POO sous Python
séance num 5

Enseignant : GHANEM

Sommaire

1) La signification de ‘self’ et l’ utilisations de méthodes statiques
2) Pourquoi pas de mot-clé static pour les attributs ?
3) Exercices

1) La signification de ‘self’ et l’ utilisations de méthodes statiques

Comprendre le self en Python

Qu'est-ce que self ?

- self est une convention en Python pour représenter **l'instance courante** de la classe
- C'est le **premier paramètre** de toutes les méthodes d'instance
- Il permet d'accéder aux **attributs et méthodes** de l'objet

À quoi sert self ?

- Accéder aux attributs de l'objet : self.nom
- Appeler d'autres méthodes : self.ma_methode()
- Différencier les attributs d'instance des variables locales

- **Exemple 1 - Classe avec constructeur et self**

```
class Etudiant:
    def __init__(self, nom, age):
        self.nom = nom      # Attribut public
        self.age = age      # Attribut public

    def afficher(self):     # Méthode avec self
        print(f"Étudiant: {self.nom}, Âge: {self.age} ans")
```

- **Utilisation de la classe Etudiant**

```
# Création d'un objet
etudiant1 = Etudiant("Ahmed", 20)

# Appel de la méthode afficher
etudiant1.afficher()
```

- **Résultat :**

```
Étudiant: Ahmed, Âge: 20 ans
```

Explication :

- self.nom et self.age accèdent aux attributs de CET objet précis
- Sans self, Python ne saurait pas de quel objet on parle

- **Exemple 2 - Méthode sans self (méthode statique)**

```
class Ecole_marocaine:
    @staticmethod
    def afficher():
        print("Maroc")
```

- **Utilisation :**

```
# Appel direct via la classe
Ecole_marocaine.afficher()  # Output: Maroc

# Ou via un objet
ecole = Ecole_marocaine()
ecole.afficher()          # Output: Maroc
```

Caractéristiques :

C’est à peu pres comme la différence entre : statique VS dynamique, mais ici : statique signifie statique pour la classe (intacte , non modifiable) et n’ appartient pas à un objet , mais appartient à tous les objets

- Pas besoin de self car la méthode n'accède à aucun attribut d'instance
- Décorateur @staticmethod indique que c'est une méthode statique

- **Exemple 3 - Attribut de classe et méthode statique**

```
class Ecole_Nationale:
    pays = "Maroc"  # Attribut de classe (statique)

    @staticmethod
    def afficher():
        print(f"Pays: {Ecole_Nationale.pays}")
```

- **Utilisation flexible :**

```
# Appel via la classe
Ecole_Nationale.afficher()  # Output: Pays: Maroc

# Appel via un objet
ecole1 = Ecole_Nationale()
ecole1.afficher()          # Output: Pays: Maroc
```

- **Avantage de l'approche statique**
- **Facilité de maintenance :**

```
# Pour adapter au Sénégal, on change UNE seule ligne :
class Ecole_Nationale:
    pays = "Sénégal"  #  Modification unique

    @staticmethod
    def afficher():
        print(f"Pays: {Ecole_Nationale.pays}")
```

- **Résultat après modification :**

```
Ecole_Nationale.afficher()  # Output: Pays: Sénégal
```

Avantages :

- **Pas besoin de modifier le code des méthodes**
- **Tous les appels continuent de fonctionner**
- **Changement centralisé et contrôlé**

Attention aux méthodes sans décorateur @staticmethod

```
class Ecole_marocaine:
    def afficher():          #  Méthode sans self, sans @staticmethod !!
        print("Maroc")

# Appel via la classe -  Ça marche
Ecole_marocaine.afficher()   # Output: Maroc

# Appel via un objet -  ERREUR !
ecole = Ecole_marocaine()
ecole.afficher()             # ERREUR !
```

- **Explication du problème :**

Quand on appelle via un objet :

ecole.afficher()

Python transforme automatiquement en :

Ecole_marocaine.afficher(ecole) # Il passe l'objet en paramètre !

Mais la méthode n'attend aucun paramètre → ERREUR !

- Par contre, si on **avait déclaré** avec le décorateur @staticmethod, ça **aurait marché**.

Pourquoi utiliser @staticmethod ?

- Compatible avec l'appel via classe ET via objet
- Intention claire : cette méthode n'a pas besoin de self

2) Pourquoi pas de mot-clé static pour les attributs ?

Python distingue naturellement les attributs :

- **Attributs de classe** : déclarés directement dans la classe
- **Attributs d'instance** : déclarés dans __init__ avec self.

- **Comment Python fait la différence**

```
class Ecole_Nationale:
    pays = "Maroc"  # Attribut de classe (partagé)

    def __init__(self, nom):
        self.nom = nom  # Attribut d'instance (spécifique à chaque objet)
```

- **Utilisation :**

```
# Attribut de classe - partagé par toutes les instances
print(Ecole_Nationale.pays)  # "Maroc"

# Attribut d'instance - spécifique à chaque objet
ecole1 = Ecole_Nationale("SupMti")
print(ecole1.nom)  # "SupMti"
```

- **Démonstration de la différence**

```
class Test:
    attribut_classe = "Je suis partagé"  # Attribut de classe

    def __init__(self, valeur):
        self.attribut_instance = valeur  # Attribut d'instance

# Test
obj1 = Test("Objet 1")
obj2 = Test("Objet 2")

print(obj1.attribut_classe)     # "Je suis partagé"
print(obj2.attribut_classe)     # "Je suis partagé"

print(obj1.attribut_instance)   # "Objet 1"
print(obj2.attribut_instance)   # "Objet 2"  (différent!)
```

- **Modification centralisée :**

```
class Ecole_Nationale:
    pays = "Maroc"  #  Un seul endroit à modifier

    @staticmethod
    def afficher():
        print(f"Pays: {Ecole_Nationale.pays}")

# Pour changer pour le Sénégal :
Ecole_Nationale.pays = "Sénégal"  #  Toutes les instances voient le changement

Ecole_Nationale.afficher()  # Output: Pays: Sénégal
```

Récapitulatif

Type	Syntaxe Python	Comment reconnaître
Attribut de classe	pays = "Maroc"	Déclaré directement dans la classe
Attribut d'instance	self.nom = nom	Déclaré dans __init__ avec self.
Méthode statique	**@staticmethod**	**Avec décorateur**
Méthode d'instance	def methode(self):	Avec self comme premier paramètre

3) Exercice

Exercice 1 : Domaine du Commerce

- Créez une classe Produit avec :
- Deux attributs : nom (chaîne) et prix (nombre)
- Une méthode afficher() qui affiche les informations du produit
- Créez un objet et testez la méthode afficher()

- **Solution :**

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

    def afficher(self):
        print(f"Produit: {self.nom}, Prix: {self.prix} DH")

# Test
p1 = Produit("Ordinateur Portable", 4500)
p1.afficher()
```

Exercice 2 : Domaine de l'Éducation

- Créez une classe Universite avec :
- Un attribut de classe pays = "Maroc"
- Un attribut d'instance nom
- Une méthode afficher_info() qui utilise l'attribut de classe
- Créez deux objets et testez la méthode

- **Solution :**

```
class Universite:
    pays = "Maroc"  # Attribut de classe

    def __init__(self, nom):
        self.nom = nom  # Attribut d'instance

    def afficher_info(self):
        print(f"Université: {self.nom}, Pays: {Universite.pays}")

# Test
u1 = Universite("Université Mohammed V")
u2 = Universite("Université Al Akhawayn")

u1.afficher_info()
u2.afficher_info()
```

Exercice 3 : Domaine des Transports Publics

- Créez une classe TransportPublic avec :
- Un attribut de classe tarif_base = 6 (DH)
- Une méthode statique afficher_tarif() qui affiche le tarif actuel
- Une méthode statique modifier_tarif(nouveau_tarif) pour ajuster le tarif
- Testez en appelant les méthodes via la classe et via un objet

```
class TransportPublic:
    tarif_base = 6  # Attribut de classe (tarif en DH)

    @staticmethod
    def afficher_tarif():
        print(f"Tarif actuel du transport: {TransportPublic.tarif_base} DH")

    @staticmethod
    def modifier_tarif(nouveau_tarif):
        TransportPublic.tarif_base = nouveau_tarif
        print(f"Tarif modifié à: {TransportPublic.tarif_base} DH")

TransportPublic.afficher_tarif() # Appel via la classe
TransportPublic.modifier_tarif(7) # Modification du tarif

bus1 = TransportPublic()
bus1.afficher_tarif()     # Appel via un objet

tramway = TransportPublic()
tramway.afficher_tarif() # Vérification avec un autre objet
```