POO sous Python séance num 2

Enseignant : GHNEM

- **Qu'est-ce qu'une classe en Python ?**
- Une classe est un modèle ou un plan pour créer des objets. Elle définit les attributs (données) et les méthodes (actions) que les objets auront.
- **Exemple simple :** La classe "Voiture" définit que toutes les voitures auront une marque, une couleur, et pourront démarrer.

- Allons étape par étape, en créant tout d'abord une classe vide (pas d’ attributs, pas de méthodes)

```
class Produit:
    pass
```

- **pass : l'instruction "ne rien faire"**
- pass est un **placeholder** (remplaçant) qui dit à Python : *"Continue, il n'y a rien à exécuter ici"*

- Félicitation vous venez de créer la classe Produit

- **Qu'est-ce qu'un objet en Python ?**
- Un objet est une instance d'une classe. C'est une variable créée à partir du modèle défini par la classe.
- **Exemple :** Si "Voiture" est une classe, alors "ma_voiture = Voiture()" crée un objet de type Voiture.

Allons donc instancier un produit (créer un produit nommé p1)

- p1=Produit()

- Maintenant on va mentionner à python qu’on souhaite ajouter un attribut ‘nom’ pour la classe Produit (et on profite de cette occasion pour affecter une valeur pour le nom de p1), et dorénavant n’ importe quel nouvel objet aura son attribut nom :

p1.nom="Ordinateur DeLL"

- Maintenant voila le nouveau modèle de la classe produit, chaque nouveau produit aura un nom

- **Qu'est-ce qu’un constructeur en Python ?**
- **__init__ : le constructeur**
- C'est une **méthode spéciale** appelée automatiquement quand on crée un nouvel objet
- Son nom signifie "initialize" (initialiser)
- Elle sert à **initialiser les attributs** de l'objet quand il est créé

- **Qu'est-ce que signigie ‘self’ en Python ?**
- **self : référence à l'instance**
- self représente **l'instance courante** de la classe
- C'est comme dire "moi-même" pour l'objet
- Permet d'accéder aux attributs et méthodes de CET objet précis

- Donc
- Si la classe "Voiture" est une classe ayant un constructeur, on peut **au moment** de l'instanciation affecter des valeurs aux attributs.
- **Exemple :** Si "Voiture" est une classe ayant un constructeur, alors :

```
ma_voiture = Voiture("Ferrari")
```

crée un objet de type Voiture ayant "Ferrari" comme valeur de son attribut marque.

- Maintenant on va fermer l’ancienne console d’exécution et on ouvre un nouveau code, on va créer notre classe , mais en utilisant un constructeur cette fois :

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

Et pour instancier :

```
p1 = Produit("Ordinateur DeLL")
```

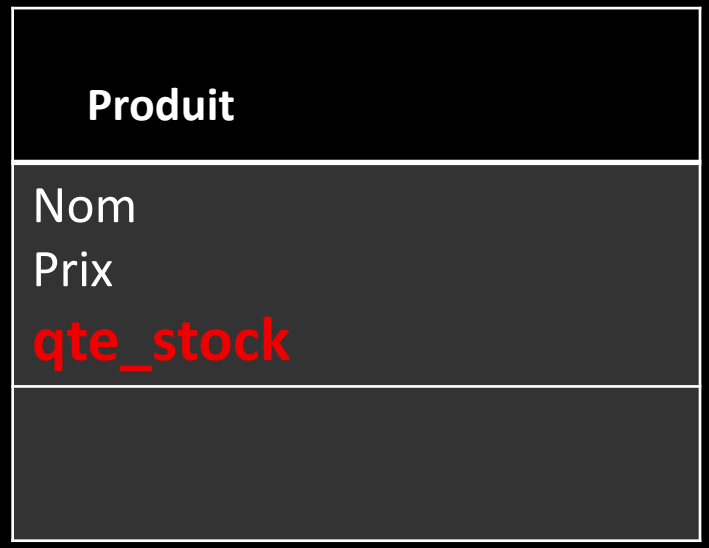
Que va afficher print ?

print(p1.nom)

- Ajoutons un autre attribut ‘prix’

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

- Ajoutons un autre attribut pour savoir la quantité en stock de ce produit



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

- Et si voulait définir une fonction ‘afficher’ , membre de cette classe et qui affiche ‘bonjour’ , juste pour tester , même si ça n’a pas de sens !
- Si elle affiche seulement ‘bonjour’ pas la peine au’elle soit membre d’ une classe spécifique !


Produit
Nom
Prix
qte_stock
Afficher()

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

    def afficher(self):
        print("bonjour")
```

Modifions le comportement de la fonction afficher pour qu’elle mérite la nomination : « Méthode »! Une méthode de la classe Produit , spécifique pour cette classe , elle sait comment afficher des infos specifiques à cette classe

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

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

- Ajoutons maintenant une fonction qui incrémente la quantité stocké dans le magasin d’un produit

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

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

    def ajouter_stock(self, m):
        self.qte_stock += m
        print(f"{m} unités ajoutées. Nouveau stock: {self.qte_stock}")
```

- Une autre méthode pour décrémenter le stock en cas de vente

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

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

    def ajouter_stock(self, m):
        self.qte_stock += m
        print(f"{m} unités ajoutées. Nouveau stock: {self.qte_stock}")

    def vendre(self, k):
        if k <= self.qte_stock:
            self.qte_stock = self.qte_stock  -  k
            print(f"{k} unités vendues. Stock restant: {self.qte_stock}")
        else:
            print("Stock insuffisant!")
```

Création d'objets Produit

```
# Création de deux objets Produit
produit1 = Produit("Laptop", 800, 10)
produit2 = Produit("Souris", 25, 50)
```

Utilisation des méthodes

```
# Appel des méthodes sur produit1
produit1.afficher()        # Affiche les infos du produit
produit1.ajouter_stock(5)  # Ajoute 5 unités au stock
produit1.vendre(3)          # Vend 3 unités
produit1.afficher()        # Affiche les infos mises à jour
```

Classe Rectangle

Classe avec attributs hauteur et largeur, et méthode pour calculer la surface.

```
class Rectangle:
    def __init__(self, hauteur, largeur):
        self.hauteur = hauteur
        self.largeur = largeur

    def surface(self):
        return self.hauteur * self.largeur
```

Classe CompteBancaire

Classe avec numéro de compte et solde, méthodes pour déposer et retirer.

```
class CompteBancaire:
    def __init__(self, numero_compte, solde):
        self.numero_compte = numero_compte
        self.solde = solde

    def deposer(self, montant):
        self.solde += montant

    def retirer(self, montant):
        if montant <= self.solde:
            self.solde -= montant
        else:
            print("Solde insuffisant")
```

- On déclare un attribut privé en utilisant deux underscores __ au début du nom de l'attribut.

Classe Note :

Classe avec nom_etudiant public et note privé, avec getter et setter.

```
class Note:
    def __init__(self, nom_etudiant, note):
        self.nom_etudiant = nom_etudiant
        self.__note = note  # Attribut privé

    def setNote(self, nouvelle_note):
        if 0 <= nouvelle_note <= 20:
            self.__note = nouvelle_note
        else:
            print("Erreur: La note doit être entre 0 et 20")

    def getNote(self):
        return self.__note
```

```
n1 = Note("kamal", 9)      # Instanciation
n1.nom_etudiant = "Ahmed"  # Modification du nom
n1.setNote(10)             # Modification de la note via setter
print(n1.nom_etudiant)     # Affichage direct du nom
print(n1.getNote())        # Affichage de la note via getter
```

Et si on essaye de tricher ?

- Si quelqu’un veut modifer la note sans passer par setNote ?

- On va instancier un objet n1 et on exécutera cette instruction :

n1.__note = 18

Quand vous faites :

n1.__note = 18

Python ne modifie pas l'attribut privé __note de la classe, mais crée un NOUVEL attribut public appelé __note !

```
class Note:
    def __init__(self, nom_etudiant, note):
        self.nom_etudiant = nom_etudiant
        self.__note = note  # Attribut privé → devient _Note__note

    def getNote(self):
        return self.__note  # Retourne _Note__note

# Test
n1 = Note("ahmed", 9)

print("Avant modification:")
print(f"getNote(): {n1.getNote()}")  # 9
print(f"Attributs: {n1.__dict__}")   # {'nom_etudiant': 'ahmed', '_Note__note': 9}

# Modification "externe"
n1.__note = 18

print("\nAprès n1.__note = 18:")
print(f"getNote(): {n1.getNote()}")  # 9 ← toujours l'ancienne valeur !
print(f"n1.__note: {n1.__note}")     # 18 ← nouveau attribut public
print(f"Attributs: {n1.__dict__}")   # {'nom_etudiant': 'kamal', '_Note__note': 9, '__note': 18}
```

Classe Etudiant

Classe représentant un étudiant avec nom, âge et méthodes étudier et dormir.

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

    def etudier(self):
        print(f"{self.nom} est en train d'étudier")

    def dormir(self):
        print(f"{self.nom} est en train de dormir")
```

Classe Voiture

Classe Voiture avec méthode démarrer qui cache la complexité interne.

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

    def demarrer(self):
        self.verifier_carburant()
        self.allumer_moteur()
        self.verifier_ceintures()
        print("Voiture démarrée!")

    def verifier_carburant(self):
        print("Vérification du carburant...")

    def allumer_moteur(self):
        print("Allumage du moteur...")

    def verifier_ceintures(self):
        print("Vérification des ceintures...")
```

```
v1=Voiture("Fiat","2025")
v1.demarrer()
```

Classe Livre (bibliothèque)

Classe Livre avec titre, auteur et méthodes pour emprunter et retourner.

```
class Livre:
    def __init__(self, titre, auteur):
        self.titre = titre
        self.auteur = auteur
        self.disponible = True

    def emprunter(self):
        if self.disponible:
            self.disponible = False
            print(f"Livre '{self.titre}' emprunté")
        else:
            print("Livre déjà emprunté")

    def retourner(self):
        self.disponible = True
        print(f"Livre '{self.titre}' retourné")
```

Classe Animal (zoo)

Classe Animal de base avec nom, âge et méthodes manger et dormir.

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

    def manger(self):
        print(f"{self.nom} mange")

    def dormir(self):
        print(f"{self.nom} dort")
```

Classe Employé (zoo)

Classe Employé avec nom, poste et méthodes pour nourrir et soigner les animaux.

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

    def nourrir_animaux(self):
        print(f"{self.nom} nourrit les animaux")

    def soigner_animaux(self):
        print(f"{self.nom} soigne les animaux")
```

Classe Enclos (zoo)

Classe Enclos avec nom, taille et liste d'animaux, méthodes pour ajouter animal et nettoyer.

```
class Enclos:
    def __init__(self, nom, taille):
        self.nom = nom
        self.taille = taille
        self.liste_animaux = []

    def ajouter_animal(self, animal):
        self.liste_animaux.append(animal)
        print(f"Animal {animal.nom} ajouté à l'enclos {self.nom}")

    def nettoyer(self):
        print(f"Enclos {self.nom} nettoyé")
```