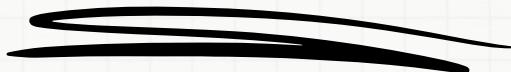




MÉNYSSA CHERIFA-LURON

POO x Hip-Hop



Programmation Orientée Objet en
Python : Application au rap

Niveau 1





MÉNYSSA CHERIFA-LURON

Programmation *impérative* vs *orientée objets*

PROGRAMMATION IMPÉRATIVE :

- Axée sur les instructions et la modification des données
- Code structuré en séquences d'instructions
- Utilise des variables, des boucles et des conditions

PROGRAMMATION ORIENTÉE OBJET :

- Axée sur les objets et leurs interactions
- Code structuré en classes et objets
- Utilise les concepts d'encapsulation, d'héritage et de polymorphisme





MÉNYSSA CHERIFA-LURON

Exemple

```
● ● ●  
1 # Programmation impérative  
2 rappeurs = ["Kendrick Lamar", "J. Cole", "Tyler, The Creator"]  
3  
4 for rappeur in rappeurs:  
5     print(f"Je suis le rappeur {rappeur}")
```

```
● ● ●  
1 # Programmation orientée objet :  
2 # Construction d'une classe "Rappeur"  
3 class Rappeur:  
4     def __init__(self, nom):  
5         self.nom = nom  
6  
7     def se_presente(self):  
8         print(f"Je suis le rappeur {self.nom}")  
9  
10 kendrick = Rappeur("Kendrick Lamar")  
11 kendrick.se_presente()
```





MÉNYSSA CHERIFA-LURON

Une approche **mérialiste**

La Programmation Orientée Objet (POO)
est un **paradigme de programmation**.

Elle est basée sur le concept **d'objets**,
qui contiennent des données (**attributs**)
et des fonctionnalités (**méthodes**)





MÉNYSSA CHERIFA-LURON

Concepts et **éléments essentiels**

- **Classe** : Structure pour créer des objets.
- **Instance** : Occurrence d'une classe.
- **Attributs** : Variables d'une classe.
- **Méthodes** : Fonctions dans une classe.
- **Constructeur (“`__init__`”)** : Initialise les instances.
- “**self**” : Référence à l'instance actuelle.
- **Héritage** : une classe hérite des caractéristiques d'une autre classe.





MÉNYSSA CHERIFA-LURON

1

Classe simple

```
1 class Rappeur:  
2     pass  
3  
4 menyssa = Rappeur()  
5 type(menyssa)
```

Ce code créé une **classe vide** “Rappeur” puis une **instance** “menyssa”





MÉNYSSA CHERIFA-LURON

2

Constructeur, méthode et instance

```
 1 # attribut "nom" et méthode "se_presente()".
 2 class Rappeur:
 3     def __init__(self, nom):
 4         self.nom = nom
 5
 6     def se_presente(self):
 7         return self.nom
 8
 9 menyssa = Rappeur("MC Meny")
10 lucas = Rappeur("Obstacle")
11 f"Je suis {menyssa.se_presente()} ! Et moi je suis {lucas.se_presente()} !"
```

“`__init__`” est le **constructeur** qui initialise chaque **instance** de “Rappeur” avec un **attribut** “nom”. La **méthode** “`se_presente`” retourne nom. Enfin, Ményssa et Lucas se présentent.





MÉNYSSA CHERIFA-LURON

3

Ajout d'une méthode

```
● ● ●  
1 class Rappeur:  
2     def __init__(self, nom, lieu_concert):  
3         self.nom = nom  
4         self.lieu_concert = lieu_concert  
5  
6     def se_presente(self):  
7         return f"Je m'appelle {self.nom}."  
8  
9     def se_produit(self):  
10        return f"Je me produirai bientôt à {self.lieu_concert}."  
11  
12 menyssa = Rappeur("MC Meny", "Toulouse")  
13 print(menyssa.se_presente())  
14 print(menyssa.se_produit())  
15  
16 lucas = Rappeur("Obstacle", "Brive-la-Gaillarde")  
17 print("\n" + lucas.se_presente()) # Ajout d'une ligne vide pour séparer les histoires  
18 print(lucas.se_produit())  
19  
20 kanye_west = Rappeur("Kanye West", "New York")  
21 print("\n" + kanye_west.se_presente())  
22 print(kanye_west.se_produit())
```

La classe “Rappeur” initie avec “nom” et “lieu_concert”, et possède des méthodes pour présenter et indiquer le lieu du prochain concert.





MÉNYSSA CHERIFA-LURON

4

Equivalent en Programmation Impérative



```
1 # Exemple en programmation impérative
2 def creer_rappeur(nom, lieu_concert):
3     return {"nom": nom, "lieu_concert": lieu_concert}
4
5 def se_presente(rappeur):
6     return f"Je m'appelle {rappeur['nom']}."
7
8 def se_produit(rappeur):
9     return f"Je me produirai bientôt à {rappeur['lieu_concert']}."
10
11 # Création des rappeurs
12 menyssa = creer_rappeur("MC Meny", "Toulouse")
13 print(se_presente(menyssa))
14 print(se_produit(menyssa))
15
16 lucas = creer_rappeur("Obstacle", "Brive-la-Gaillarde")
17 print("\n" + se_presente(lucas)) # Ajout d'une ligne vide pour séparer les histoires
18 print(se_produit(lucas))
19
20 kanye_west = creer_rappeur("Kanye West", "New York")
21 print("\n" + se_presente(kanye_west))
22 print(se_produit(kanye_west))
```





MÉNYSSA CHERIFA-LURON

La POO est une approche puissante,
adaptée aux
projets complexes
et aux développements à
grande échelle.





MÉNYSSA CHERIFA-LURON

Pas convaincu ? Voyons maintenant

L'héritage !





MÉNYSSA CHERIFA-LURON

5

L'héritage

```
● ● ●  
1 class RappeurDeCompton(Rappeur):  
2     def __init__(self, nom, lieu_concert, quartier):  
3         super().__init__(nom, lieu_concert)  
4         self.quartier = quartier  
5  
6     def rapper_from(self):  
7         print(f"et je suis originaire de {self.quartier}")  
8  
9  
10    dr_dre = RappeurDeCompton("Dr. Dre", "The Forum", "Compton")  
11    ice_cube = RappeurDeCompton("Ice Cube", "Hollywood Palladium", "Compton")  
12    kendrick_lamar = RappeurDeCompton("Kendrick Lamar", "Staples Center", "Compton")  
13  
14    # Test des méthodes  
15    print(dr_dre.se_presente())  
16    dr_dre.rapper_from()  
17  
18    print("\n" + ice_cube.se_presente())  
19    ice_cube.rapper_from()  
20  
21    print("\n" + kendrick_lamar.se_presente())  
22    kendrick_lamar.rapper_from()  
23
```

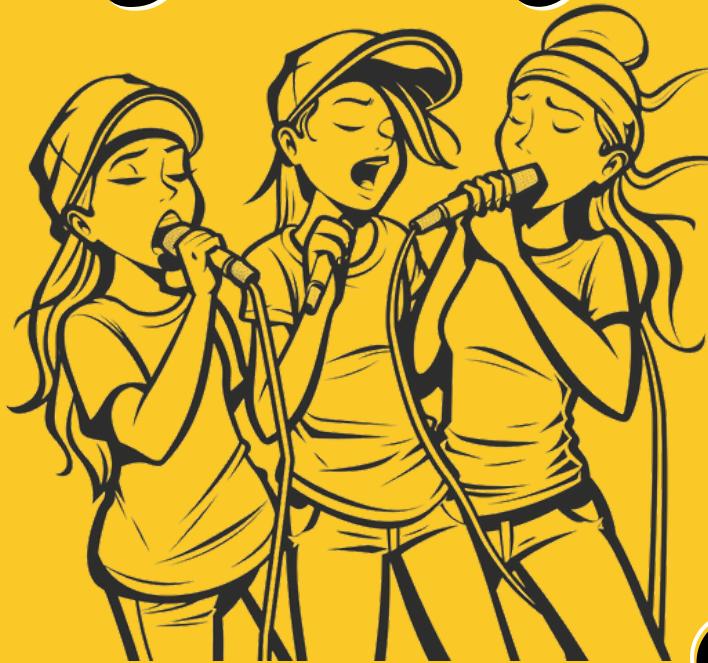
Le “super()” appelle le constructeur de la **classe parente** “Rappeur” et initialise l'**attribut** “quartier”. “RappeurDeCompton” affiche l’origine du rappeur





MÉNYSSA CHERIFA-LURON

NOW,LET'S GET
Groovy





MÉNYSSA CHERIFA-LURON

Exercice : Création d'une classe "Chanteur"

1 - Créez une classe Chanteur avec les attributs “nom”, “genre_musical”, et “nombre_de_fans”.

2 - Ajoutez une méthode “ajouter_fans” qui accepte un nombre spécifique de fans à ajouter au total existant de fans.

3 - Ajoutez une méthode “collaborer” qui simule une collaboration entre deux chanteurs. Cette méthode suppose que la collaboration augmente le nombre de fans de 10 % pour chaque artiste.

4 - Gestion des erreurs (Bonus)

- Le nombre de fans ajouté est “positif” et de type “int”. Si invalide, lever une “ValueError” avec un message approprié.
- L'objet passé à la méthode collaborée est une instance de classe Chanteur. Si ce n'est pas le cas, lever une “TypeError” avec un message approprié



Tu As Aimé ?



Tu veux plus d'astuces personnalisées ?

PRENDS RDV MAINTENANT !

Call Boost ❤️

@MÉNYSSA CHERIFA-LURON, PHD

Data Scientist et
Formatrice Freelance