Introduction au Cours Python Avancé

Bienvenue dans ce cours avancé sur Python. Vous allez approfondir vos compétences en programmation Python tout en étant guidé par une approche structurée. Le cours est à la fois théorique et pratique, avec des exercices étapes par étapes et un projet fil rouge pour appliquer vos acquis.

Chapitre 1 : Introduction et Bonnes Pratiques

Objectifs :

- Revoir les bases essentielles de Python.
- Apprendre les conventions et bonnes pratiques pour écrire du code propre et maintenable.
- Configurer un environnement de développement efficace.

1.1. Rappel des Bases

Types de base :

- **Numériques** : `int`, `float`, `complex`
- **Chaînes de caractères** : `str`
- **Booléens** : `True`, `False`
- **Collections** : `list`, `tuple`, `dict`, `set`

Syntaxe :

- Indentation : Utilisez 4 espaces (pas de tabulations).
- Variables : Utilisez des noms explicites, comme `age`, `nom_utilisateur`.
- Exemple :

```
age = 25
nom = "Alice"
print(f"{nom} a {age} ans.")
```

1.2. Conventions de Style (PEP 8)

- **Noms de variables** : snake_case (ex. : `nombre_utilisateurs`)
- **Noms de classes** : PascalCase (ex. : `UtilisateurPremium`)
- **Commentaires** :
- Utilisez `#` pour expliquer le code.
- Les docstrings (""""""") sont pour décrire des fonctions et classes.

Exemple :

```python
def additionner(a, b):
    """
    Retourne la somme de deux nombres.
    :param a: Premier nombre.
    :param b: Deuxième nombre.
    :return: Somme de a et b.
    """
    return a + b
```

1.3. Configurer votre Environnement

1. **Choix de l'IDE/éditeur** :
2. **VS Code** : Extensions recommandées : Python, Pylance.
3. **PyCharm** : Idéal pour des projets complexes.
4. **Installation des outils** :
5. Python 3.10 ou version ultérieure.
6. Gestionnaire de paquets : `pip` ou `poetry`.
7. **Configuration des virtualenv** :
8. Créer un environnement virtuel :

```
python -m venv env
source env/bin/activate  # Sous Linux/Mac
env\Scripts\activate   # Sous Windows
```

9. Installer des bibliothèques :

```
pip install numpy pandas
```

10. **Utilisation de Git** :
11. Initialisez un dépôt local :

```
git init
```

12. Ajoutez vos fichiers :

```
git add .
git commit -m "Initial commit"
```

1.4. Exercices

Exercice 1 : Variables et types

- **Objectif** : Créer des variables représentant un utilisateur.
- **Instructions** :
- Déclarez une variable `nom` avec une chaîne.
- Déclarez une variable `age` avec un entier.
- Affichez une phrase comme : "Alice a 30 ans."

Indice : Utilisez `print` avec une f-string.

Solution :

```
nom = "Alice"
age = 30
print(f"{nom} a {age} ans.")
```

Exercice 2 : Conventions

- **Objectif** : Renommer une fonction pour suivre les conventions PEP 8.
- **Instructions** : Voici une fonction mal nommée :

```
def AdditionDeuxNombres(a, b):
    return a + b
```

Renommez-la correctement.

Solution :

```
def addition_deux_nombres(a, b):
    return a + b
```

Projet de fond : Gestionnaire de Tâches (Partie 1)

Objectif :

Créer une application pour gérer des tâches personnelles. Aujourd'hui, nous allons définir le squelette du projet.

Instructions :

1. Créez un répertoire `gestionnaire_taches`.
2. Ajoutez un fichier `main.py`.
3. Définissez une structure minimale :

```
"""
Application de gestion de tâches.
"""

def main():
    print("Bienvenue dans le gestionnaire de tâches !")

if __name__ == "__main__":
    main()
```

Prochaine étape :

Dans le prochain chapitre, nous allons approfondir les structures de données et commencer à ajouter des fonctionnalités à notre projet.

Chapitre 2 : Structures de Données Avancées

Objectifs :

- Comprendre et maîtriser les structures de données de Python.
- Utiliser les compréhensions de liste et les générateurs.
- Explorer les modules `collections` et `itertools`.

2.1. Listes, Tuples, Dictionnaires, Ensembles

Listes

```
fruits = ["pomme", "banane", "cerise"]
fruits.append("orange")
print(fruits[1])  # banane
```

Tuples (immutables)

```
coordonnees = (10, 20)
x, y = coordonnees
```

Dictionnaires

```
utilisateur = {"nom": "Alice", "age": 30}
print(utilisateur["nom"])
```

Ensembles (non ordonnés, uniques)

```
nombres = {1, 2, 3, 2}
print(nombres)  # {1, 2, 3}
```

2.2. Compréhensions

Liste des carrés de 0 à 9 :

```
carres = [x**2 for x in range(10)]
```

Dictionnaire de longueur des mots :

```
mots = ["chat", "chien", "cheval"]
longueurs = {mot: len(mot) for mot in mots}
```

Ensemble des premières lettres :

```
premieres_lettres = {mot[0] for mot in mots}
```

2.3. Générateurs

Syntaxe avec yield :

```
def compte_jusqua(n):
    for i in range(n):
        yield i
```

Utilisation :

```
for nombre in compte_jusqua(5):
    print(nombre)
```

2.4. Module `collections`

Counter

```
from collections import Counter
compteur = Counter("abracadabra")
print(compteur)
```

defaultdict

```
from collections import defaultdict
liste_notes = ["Alice", "Bob", "Alice"]
notes = defaultdict(list)
for eleve in liste_notes:
    notes[eleve].append(20)
```

namedtuple

```
from collections import namedtuple
Point = namedtuple("Point", ["x", "y"])
p = Point(1, 2)
print(p.x, p.y)
```

2.5. Module `itertools`

Produit cartésien

```
from itertools import product
for combinaison in product([1, 2], ["a", "b"]):
    print(combinaison)
```

Accumulateur

```
from itertools import accumulate
import operator
print(list(accumulate([1, 2, 3, 4], operator.mul)))  # [1, 2, 6, 24]
```

2.6. Exercices

Exercice 1 : Filtrer une liste

- **Instructions** : Créez une liste contenant uniquement les carrés pairs de 0 à 20.

- **Indice** : combinez une compréhension de liste et un test `x % 2 == 0`.

Solution :

```
carres_pairs = [x**2 for x in range(21) if x**2 % 2 == 0]
```

Exercice 2 : Utiliser Counter

- **Instructions** : Comptez les lettres dans "programmation".

Solution :

```
from collections import Counter
print(Counter("programmation"))
```

Projet de fond : Gestionnaire de Tâches (Partie 2)

Objectif :

Créer une structure pour stocker des tâches avec leurs statuts.

Instructions :

1. Ajoutez une structure `taches` dans `main.py` :

```
taches = [
    {"titre": "Faire les courses", "terminee": False},
    {"titre": "Apprendre Python", "terminee": True}
]
```

1. Affichez les tâches :

```
for t in taches:
    statut = "✔️" if t["terminee"] else "✘"
    print(f"{statut} {t['titre']}")
```

Prochaine étape :

Dans le chapitre suivant, nous explorerons la programmation orientée objet pour structurer notre application avec des classes.