La syntaxe en Python

Introduction

Python est un langage de programmation apprécié pour sa simplicité et sa lisibilité. L'une des raisons principales de cette lisibilité réside dans son usage strict de l'indentation. Contrairement à d'autres langages, Python utilise l'indentation pour délimiter les blocs de code (par exemple, les blocs d'instructions dans une condition ou une boucle).

Dans ce cours, nous allons apprendre :

1. **La structure d'un fichier Python**
2. **La syntaxe de base de Python.**
3. **Les règles d'indentation.**
4. **Quelques erreurs courantes que vous pourriez rencontrer et comment les éviter.**

Partie 1. Structure d'un fichier Python : Ordre et organisation

Lorsque vous écrivez un programme Python, il est important d'organiser correctement votre code pour qu'il soit **lisible**, **maintenable** et **clair**.

Voici l'ordre typique d'un fichier Python bien organisé :

1. **Les imports des modules** : Les bibliothèques et modules externes utilisés dans le programme.
2. **Les constantes globales (optionnel)** : Les variables globales définies en majuscules (par exemple, les valeurs fixes comme des mots de passe ou des paramètres).
3. **Les déclarations des variables principales** : Les variables nécessaires au programme.
4. **Les définitions des fonctions (si utilisées)**.
5. **Le code principal du programme** : C'est ici que vous exécutez les instructions principales.

1.1 Les imports de modules

Les imports permettent d'utiliser des bibliothèques standard ou des modules externes dans votre programme (par exemple : **random**, **math**). Ils doivent être placés au tout début du fichier.

Exemple :

```
import random  # Pour générer des nombres aléatoires
import math    # Pour effectuer des calculs mathématiques
```

Règles à respecter :

- Les imports doivent être regroupés au début du fichier.
- Si vous utilisez plusieurs types de modules, suivez cet ordre :
    1. **Modules intégrés à Python** (comme math, random, etc.).
    2. **Modules installés** via des bibliothèques externes (comme numpy, pandas, etc.).

Exemple d'ordre correct :

```
import math
import random
import datetime
import numpy
```

1.2. Les constantes globales

Les constantes sont des **variables** dont la **valeur ne change** pas pendant l'exécution du programme. Par convention, elles sont écrites en **MAJUSCULES** avec des **underscores** _ pour séparer les mots. Elles permettent de définir des paramètres globaux comme un token, une limite ou une configuration.

Exemple :

```
LIMITE_AGE = 18
TOKEN = "1234567890"
```

1.3. Les déclarations de variables principales

Ensuite, vous pouvez déclarer les variables principales nécessaires pour votre programme. Ce sont des valeurs qui sont utilisées tout au long de l'exécution.

Exemple :

```
nom_utilisateur = "Alice"
age_utilisateur = 25
```

1.4. Les fonctions

Les **définitions des fonctions** doivent être regroupées après les variables et avant le code principal. Cela rend le programme plus lisible et facile à comprendre.

```
def afficher_message_bienvenue():
    print("Bienvenue dans le programme !")
```

1.5. Le code principal du programme

Le code principal correspond aux **instructions** qui exécutent directement le programme. Par exemple, vous pouvez **afficher** des messages, **interagir** avec l'utilisateur ou effectuer des **calculs**.

Ce code principal est souvent écrit après les fonctions et les déclarations globales.

Exemple complet d'un fichier Python structuré :

```python
# 1. Imports
import random  # Pour générer des nombres aléatoires

# 2. Constantes globales
LIMITE_AGE = 18  # Âge minimum pour être majeur
MOT_DE_PASSE = "Python123"  # Mot de passe pour accéder au programme

# 3. Déclarations de variables principales
nom_utilisateur = "Alice"
age_utilisateur = 20

# 4. Code principal
print("Bienvenue dans le programme.")

# Vérification de l'âge
if age_utilisateur >= LIMITE_AGE:
    print(f"{nom_utilisateur} est majeur(e).")
else:
    print(f"{nom_utilisateur} est mineur(e).")

# Jeu de devinette simple
nombre_a_trouver = random.randint(1, 10)
print(f"Un nombre entre 1 et 10 a été choisi. Essayez de le deviner.")
```

1.6 Les bonnes pratiques

1. **Regroupez vos imports :** Ne mélangez pas les imports avec d'autres parties du code. Ils doivent toujours être au tout début.
2. **Utilisez des noms clairs pour les variables :** Par exemple, préférez age_utilisateur à x.
3. **Ajoutez des commentaires :** Expliquez ce que fait chaque partie du code pour faciliter la compréhension, surtout si le programme devient plus long.
4. **Respectez l'ordre logique :** Même si vous ne comprenez pas encore tout (par exemple les fonctions), respectez l'ordre présenté ici pour vous habituer à une bonne structure.

Partie 2. La syntaxe de base de Python

1.1 Instructions simples

En Python, une **instruction** est généralement une ligne de code qui effectue une opération. Par exemple :
Chaque instruction se **termine directement à la fin de la ligne** (pas de point-virgule nécessaire comme dans certains langages).

```
x = 10  # Affectation d'une valeur à une variable
print(x)  # Affichage de la variable
```

1.2 Les commentaires

Les **commentaires** sont des lignes de texte dans votre code qui ne sont pas exécutées. Ils sont utilisés pour expliquer ce que fait le programme. En Python, un **commentaire** commence par le caractère **#**.

Exemples :

```
# Ceci est un commentaire
x = 10  # On stocke la valeur 10 dans la variable x
print(x)  # On affiche la valeur de x
```

1.3 Les blocs de code

Un bloc de code est un ensemble d'instructions qui sont exécutées ensemble. Par exemple :

- Les instructions dans un **if**.
- Les instructions dans une **boucle for** ou **while**.

En Python, les blocs de code sont définis par **l'indentation** (voir la partie suivante).

Partie 3. L'indentation en Python

L'indentation consiste à ajouter des espaces ou des tabulations au début d'une ligne de code. Python utilise l'indentation pour indiquer qu'une ligne appartient à un **bloc de code spécifique**. Contrairement à d'autres langages, **Python impose l'indentation**. Si vous oubliez d'indenter correctement, vous obtiendrez une erreur.

2.1 Règles d'indentation

1. **Début d'un bloc :** Chaque fois que vous commencez une structure comme un if, une boucle for ou while, le code qui doit être exécuté dans ce bloc doit être indenté.
2. **Indentation uniforme :** Les lignes d'un même bloc doivent avoir le même niveau d'indentation.
3. **Fin d'un bloc :** Pour indiquer la fin d'un bloc, il suffit de revenir à l'indentation précédente.

Avec une condition if :

```
x = 10

if x > 5:  # Début du bloc
    print("x est supérieur à 5")  # Indentation nécessaire
    print("Ceci est dans le bloc if")  # Même niveau d'indentation

print("Ceci est hors du bloc if")  # Pas d'indentation : hors du bloc
```

Sortie :

```
x est supérieur à 5
Ceci est dans le bloc if
Ceci est hors du bloc if
```

Avec une boucle while :

```
x = 0

while x < 3:  # Début de la boucle
    print(x)  # Indentation : fait partie de la boucle
    x += 1  # Indentation : également dans la boucle

print("Fin de la boucle")  # Pas d'indentation : hors de la boucle
```

Sortie :

```
0
1
2
Fin de la boucle
```

2.2 Espaces ou tabulations ?

Python permet d'utiliser des **espaces (4)** ou des **tabulations** pour indenter le code.

Attention : Ne mélangez pas les **espaces (4)** et les **tabulations** dans un même fichier, cela provoquera une erreur.

Partie 4. Erreurs courantes liées à l'indentation

3.1 Pas d'indentation après un bloc qui en exige une

```
x = 10

if x > 5:
print("x est supérieur à 5")  # Erreur : pas d'indentation
```

Erreur levée :

```
IndentationError: expected an indented block
```

3.2 Indentation incorrecte ou inégale

```
x = 10

if x > 5:
    print("x est supérieur à 5")
   print("Ceci est dans le bloc if")  # Erreur : indentation inégale
```

Erreur levée :

```
IndentationError: unindent does not match any outer indentation level
```

3.3 Retour à la ligne sans bloc

Si une ligne indentée n'a pas de lien logique avec un bloc précédent, cela causera une erreur.

```
x = 10
if x > 5:
    print("x est supérieur à 5")
    print("x est grand")
print("Fin")
```