# PYTHON - Mémo PEP 8

https://peps.python.org/pep-0008/

Cheat sheet :
https://www.cs.utexas.edu/~mitra/csFall2022/cs313/lectures/pep8.pdf

*PEP 8 est le guide de style officiel pour écrire du code Python lisible et cohérent. Il fournit des recommandations sur la mise en forme, la structuration du code, et les bonnes pratiques à suivre pour favoriser la collaboration et la maintenabilité du code.*

---

- **Indentation :** Utilisez 4 espaces par niveau d'indentation.
- **Longueur des lignes :** Limitez la longueur des lignes à 79 caractères pour le code, les commentaires et les docstrings.
- **Imports :**
  - Placez tous les imports en haut du fichier
  - Organisez les imports en trois sections : imports de bibliothèques standard (natives python), imports de bibliothèques tierces (installées), puis imports locaux.
- **Lignes vides :**
  - Insérez deux lignes vides entre les imports et le code
  - Utilisez une ligne vide entre les méthodes d'une classe.
  - Utilisez deux lignes entre chaque fonctions créées
- **Espaces autour des opérateurs :**
  - Évitez d'ajouter des espaces immédiatement à l'intérieur des parenthèses, crochets ou accolades : ( `func(a, b)` ).
  - Ajoutez un espace avant et après les opérateurs binaires ( `=, +=, -, *, etc.` ).
- **Nommage :**

- Utilisez le snake_case : noms de variables en minuscules avec des underscores (`variable_name`)
- Définir le nom des variables, fonctions et classes en anglais.
- Utilisez le snake_case des pour les variables et les fonctions.
- Utilisez le CamelCase (`VariableName`) pour les noms de classes.
- Les constantes doivent être en majuscules avec des underscores (`CONSTANTE`).
- Mettre un espace entre le nom de la variable, le `=` et la valeur assignée

```
x = 2 # good
x=2 # bad
```

- **Typage :**
  - Définir au maximum le type des variables lors de la déclaration, assignation, etc.

```
x: int = 2
prices: list[int] = get_prices()
```

- **Commentaires et docstrings :**
  - Les commentaires doivent être complets et pertinents, pas de commentaires superflus.
  - Utilisez des docstrings en début de fonction ou de classe pour décrire leur utilité et comportement.

```
def add(a: int, b: int) → int:
    """Additionne deux nombres entiers.

        Args:
            a (int): Le premier nombre.
            b (int): Le second nombre.

        Returns: int: La somme de `a` et `b`.
```

```
    Exemple: >>> additionner(2, 3) 5
    """
    return a + b


def complex_computation(a: int, b: list[int]) → int:
    """Calcul compliqué

    Args:
        a (int): Le premier nombre.
        b (int): Le second nombre.

    Returns: int: La somme de `a` et `b`.
    """
    pass
```

- Spécifier au maximum les arguments des fonctions utilisés quand c'est util.

```
complex_computation(a=2, b=[4,3,2]) # good
complex_computation(2, 4) # bad
```