Cours Complet Python

De l'initiation aux concepts avancés

Abdeljalil BOUZINE
Code Crafters

21 Octobre 2024

Table des Matières

Introduction à Python

Qu’est-ce que Python ?

- Python est un langage de programmation interprété, polyvalent et de haut niveau.
- Créé par Guido van Rossum et sorti en 1991.
- Connu pour sa syntaxe simple et lisible.
- Favorise la productivité et la facilité de maintenance.
- Supporte plusieurs paradigmes de programmation : procédural, orienté objet, fonctionnel.
- Dispose d’une grande bibliothèque standard et de nombreux modules tiers.

Exemple 1 : Afficher ”Hello, Crafters !” en Python

```
1 print("Hello, Crafters !")
```

Exemple 2 : Calculer la somme de deux nombres

```
1  a = 10
2  b = 20
3  somme = a + b
4  print("La somme est :", somme)  # Affiche "La somme est : 30"
```

Installation et Configuration

Installation de Python

1. Rendez-vous sur le site officiel : `https://www.python.org/downloads/`
2. Téléchargez la version adaptée à votre système (Windows, macOS, Linux).
3. Suivez les instructions d'installation en veillant à ajouter Python au PATH.
4. Vérifiez l'installation en exécutant `python --version` dans le terminal.
5. **Conseil :** Utilisez des environnements virtuels (`venv`) pour gérer vos dépendances.

Exemple : Créer et activer un environnement virtuel

```
# Création d'un environnement virtuel
python -m venv mon_env

# Activation sur Windows
mon_env\Scripts\activate

# Activation sur Unix/MacOS
source mon_env/bin/activate
```

Choix de l'Environnement de Développement (IDE)

- **PyCharm :** IDE complet avec débogueur intégré, auto-complétion, gestion des virtualenv, etc.
- **Visual Studio Code :** Éditeur léger avec extensions pour Python (linting, debugging, etc.).
- **Jupyter Notebook :** Idéal pour la data science, permet d'exécuter du code par cellules avec visualisation intégrée.
- **Spyder :** Orienté vers la science des données, avec des outils de visualisation intégrés.
- **Sublime Text, Atom :** Éditeurs de texte polyvalents avec support Python via des plugins.

Astuce : Choisissez un environnement qui correspond à vos besoins et à votre niveau de confort.

Variables et Types de Données

Variables et Affectation

- Une variable est un conteneur pour stocker des valeurs.
- Nommer les variables de manière descriptive améliore la lisibilité.
- Types dynamiques : le type est déterminé au moment de l'exécution.
- Convention de nommage : `snake_case` pour les variables et fonctions.
- Les variables peuvent changer de type en cours de programme.

Exemple : Variables et Affectation (Partie 1)

```
1  # Affectation simple
2  nom_utilisateur = "JL"
3  age_utilisateur = 25
4
5  # Affectations multiples
6  a, b, c = 5, 10, 15
7
8  # Échange de variables
9  a, b = b, a
```

Exemple : Variables et Affectation (Partie 2)

```python
# Changement de type
pi = 3.14
pi = "Nombre Pi"

# Utilisation de variables dans des opérations
longueur = 5
largeur = 3
aire = longueur * largeur
print("L'aire est :", aire)  # Affiche "L'aire est : 15"

# Typage dynamique
variable = 10
print(variable)  # 10
variable = "Dix"
print(variable)  # "Dix"
```

Les Types de Données Principaux

- **Numériques :** int, float, complex.
- **Booléens :** True, False.
- **Chaînes de caractères :** str.
- **Listes :** list (mutables, ordonnées).
- **Tuples :** tuple (immuables, ordonnés).
- **Dictionnaires :** dict (paires clé-valeur).
- **Ensembles :** set (non ordonnés, éléments uniques).

Exemple : Types de Données (Partie 1)

```python
pi = 3.1416                # float
est_actif = True           # bool
message = "Hello!"         # str
nombres = [1, 2, 3]        # list
coordonnees = (10, 20)     # tuple
personne = {"nom": "JL", "âge": 21}  # dict
lettres = {'a', 'b', 'c'}  # set
nombre_complexe = 2 + 3j   # complex
```

Exemple : Types de Données (Partie 2)

```
# Accès aux éléments
print(nombres[0])          # 1
print(coordonnees[1])      # 20
print(personne["nom"])     # JL
```

Exemple : Types de Données (Partie 3)

```python
# Opérations sur chaînes de caractères
prenom = "Code"
nom = "Crafters"
nom_complet = prenom + " " + nom
print(nom_complet)          # Code Crafters

# Conversion de types
age_str = "30"
age_int = int(age_str)
print(age_int)              # 30
```

Opérations et Contrôle de Flux

Opérateurs Arithmétiques

- Addition : `+`
- Soustraction : `-`
- Multiplication : `*`
- Division : `/`
- Division entière : `//`
- Modulo : `%`
- Puissance : `**`

Opérateurs d'affectation : `+=`, `-=`, `*=`, `/=`, etc.

Exemple : Opérateurs Arithmétiques (Partie 1)

```
1  a = 10
2  b = 3
3
4  print(a + b)   # 13
5  print(a - b)   # 7
6  print(a * b)   # 30
7  print(a / b)   # 3.333...
8  print(a // b)  # 3
9  print(a % b)   # 1
10 print(a ** b)  # 1000
```

Exemple : Opérateurs Arithmétiques (Partie 2)

```
a += 5  # a = a + 5
print(a)  # 15

# Opérations avec des flottants
c = 2.5
d = 4.0
print(c * d)   # 10.0
```

Exemple : Opérateurs Arithmétiques (Partie 3)

```
1  # Opérations avec des complexes
2  e = 1 + 2j
3  f = 3 + 4j
4  print(e + f)   # (4+6j)
```

Opérateurs de Comparaison et Logiques

- Égal à : `==`
- Différent de : `!=`
- Inférieur à : `<`
- Supérieur à : `>`
- Inférieur ou égal à : `<=`
- Supérieur ou égal à : `>=`
- Opérateurs logiques : `and`, `or`, `not`

Exemple : Opérateurs de Comparaison et Logiques

```
x = 5
y = 10

print(x == y)          # False
print(x != y)          # True
print(x < y and x > 0) # True
print(not x > y)       # True
print(x < y or x > 100) # True

# Opérations sur des booléens
a = True
b = False
print(a and b)         # False
print(a or b)          # True
print(not a)           # False
```

Structures Conditionnelles

- Permettent d'exécuter du code en fonction d'une condition.
- Syntaxe :
  - `if` condition:
  - `elif` condition:
  - `else`:
- Indentation obligatoire pour délimiter les blocs de code.
- Les conditions peuvent être combinées avec des opérateurs logiques.

Exemple : Structures Conditionnelles (Partie 1)

```
1  note = 85
2
3  if note >= 90:
4      print("Excellent")
5  elif note >= 75:
6      print("Très bien")
7  elif note >= 60:
8      print("Bien")
9  else:
10     print("Peut mieux faire")
```

Exemple : Structures Conditionnelles (Partie 2)

```python
# Condition imbriquée
age = 20
if age >= 18:
    if age >= 21:
        print("Accès autorisé aux boissons alcoolisées.")
    else:
        print("Accès autorisé, mais boissons alcoolisées interdites.")
else:
    print("Accès refusé.")
```

Boucles

- **Boucle `for` :** Parcourt une séquence d'éléments.
- **Boucle `while` :** S'exécute tant qu'une condition est vraie.
- Utilisation de `break` pour sortir de la boucle.
- Utilisation de `continue` pour passer à l'itération suivante.
- Les boucles peuvent être imbriquées.

Exemple de boucle `for`

```python
fruits = ["pomme", "banane", "cerise"]
for fruit in fruits:
    print(fruit)

# Boucle avec range()
for i in range(5):
    print(i)

# Boucle imbriquée
for i in range(3):
    for j in range(2):
        print(f"i = {i}, j = {j}")
```

Exemple de boucle `while` - Partie 1

```
1 compteur = 0
2 while compteur < 5:
3     print(compteur)
4     compteur += 1
```

Exemple de boucle `while` - Partie 2

```
1  compteur = 0
2  while compteur < 5:
3      if compteur == 3:
4          compteur += 1
5          continue  # Passe à l'itération suivante
6      print(compteur)
7      compteur += 1
```

Fonctions

Définition de Fonctions

- Une fonction est un bloc de code réutilisable.
- Utilise le mot-clé `def`.
- Peut accepter des paramètres et retourner des valeurs.
- Permet de structurer le code et d'éviter les répétitions.
- Peut contenir une documentation (docstring) pour décrire son fonctionnement.

Exemple : Définition de Fonctions - Partie 1

```python
def saluer(nom):
    """Cette fonction salue la personne passée en paramètre."""
    print(f"Hello, {nom}!")

saluer("JL")  # Affiche "Hello, JL!"

# Fonction sans paramètre
def afficher_date():
    from datetime import datetime
    print("Nous sommes le", datetime.now().date())

afficher_date()
```

Exemple : Définition de Fonctions (Partie 2.1)

```python
# Fonction avec paramètres par défaut et retour de valeur
def addition(a, b=0):
    """Retourne la somme de a et b."""
    return a + b

resultat = addition(5)       # 5 + 0 = 5
resultat2 = addition(5, 3)   # 5 + 3 = 8
print(f"Addition(5) = {resultat}")             # 5
print(f"Addition(5, 3) = {resultat2}")         # 8
```

Exemple : Définition de Fonctions (Partie 2.2)

```python
# Fonction retournant plusieurs valeurs
def calculs(a, b):
    """Retourne la somme et le produit de a et b."""
    somme = a + b
    produit = a * b
    return somme, produit

s, p = calculs(4, 5)
print(f"Somme: {s}")       # Somme: 9
print(f"Produit: {p}")     # Produit: 20
```

Exemple : Définition de Fonctions (Partie 2.3)

```python
# Paramètres nommés lors de l'appel de fonction
def afficher_info(nom, age):
    """Affiche les informations d'une personne."""
    print(f"Nom: {nom}, Âge: {age}")

afficher_info(age=22, nom="simo")  # Affiche "Nom: simo, Âge: 30"
```

Structures de Données

Listes Avancées

- **Slicing :** Accéder à une partie de la liste.
- **Compréhensions de listes :** Créer des listes avec des boucles intégrées.
- Méthodes utiles : `append()`, `remove()`, `sort()`, `reverse()`.
- Les listes peuvent contenir des éléments de types différents.

Exemple : Slicing

```python
nombres = [0, 1, 2, 3, 4, 5]
sous_liste = nombres[2:5]  # [2, 3, 4]
sous_liste2 = nombres[:3]  # [0, 1, 2]
sous_liste3 = nombres[::2] # [0, 2, 4]
print(f"Sous-liste [2:5] : {sous_liste}")
print(f"Sous-liste [:3] : {sous_liste2}")
print(f"Sous-liste [::2] : {sous_liste3}")
```

Exemple : Compréhensions de liste

```python
carres = [x**2 for x in range(6)]            # [0, 1, 4, 9, 16, 25]
pairs = [x for x in range(10) if x % 2 == 0] # [0, 2, 4, 6, 8]
print(f"Carrés : {carres}")
print(f"Nombres pairs : {pairs}")
```

Exemple : Méthodes utiles

```python
fruits = ["pomme", "banane", "cerise"]
fruits.append("orange")
print(f"Après append : {fruits}")  # ['pomme', 'banane', 'cerise', 'orange']

fruits.remove("banane")
print(f"Après remove : {fruits}")  # ['pomme', 'cerise', 'orange']

fruits.sort()
print(f"Après sort : {fruits}")    # ['cerise', 'orange', 'pomme']

fruits.reverse()
print(f"Après reverse : {fruits}") # ['pomme', 'orange', 'cerise']
```

Dictionnaires Avancés

- Stockent des paires clé-valeur.
- Clés uniques et immuables.
- Méthodes utiles : `items()`, `keys()`, `values()`, `get()`.
- Permettent un accès rapide aux données via les clés.

Exemple : Dictionnaires Avancés - Partie 1

```python
etudiant = {"nom": "JL", "âge": 22, "matière": "Informatique"}

# Parcourir les clés et les valeurs
for clé, valeur in etudiant.items():
    print(f"{clé} : {valeur}")

# Accéder à une valeur avec get()
age = etudiant.get("âge", "Non spécifié")
print(f"Âge de l'étudiant : {age}")  # 22
```

Exemple : Dictionnaires Avancés - Partie 2

```python
# Ajouter ou mettre à jour une clé
etudiant["âge"] = 23
print(f"Après mise à jour de l'âge : {etudiant['âge']}")   # 23

# Supprimer une clé
del etudiant["matière"]
print(f"Dictionnaire après suppression : {etudiant}")  # {'nom': 'JL', 'âge': 2

# Vérifier si une clé existe
if "nom" in etudiant:
    print("Le nom est présent.")  # Le nom est présent.
```

Programmation Orientée Objet (POO)

Introduction à la Programmation Orientée Objet

- La Programmation Orientée Objet (POO) est une façon d'organiser le code.
- Elle repose sur la création de classes et d'objets.
- Les classes définissent des modèles ou des "plans" pour les objets.
- Les objets sont des instances de classes avec des attributs (données) et des méthodes (fonctions).

Création d'une classe simple

```python
# Définition d'une classe "Personne"
class Personne:
    def __init__(self, nom, age):
        self.nom = nom
        self.age = age
```

- `__init__` : Méthode spéciale appelée lors de la création d'une instance.
- `self` : Représente l'instance actuelle. Utilisé pour accéder aux attributs.

Utilisation d'une classe

```python
# Création d'un objet "Personne"
personne1 = Personne("Alice", 25)

# Accès aux attributs
print(personne1.nom)  # Alice
print(personne1.age)  # 25
```

- Un objet est créé à l'aide du nom de la classe suivi de parenthèses.
- Les valeurs passées dans les parenthèses sont transmises au constructeur (__init__).

Ajout de méthodes

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

    def se_presenter(self):
        return f"Je m'appelle {self.nom} et j'ai {self.age} ans."
```

- Les méthodes sont des fonctions définies à l'intérieur des classes.
- Elles permettent aux objets d'effectuer des actions.
- `self` est toujours le premier paramètre d'une méthode.

Utilisation des méthodes

```python
# Création d'une instance de Personne
personne2 = Personne("Bob", 30)

# Appel d'une méthode
message = personne2.se_presenter()
print(message)  # "Je m'appelle Bob et j'ai 30 ans."
```

- Les méthodes sont appelées à l'aide du point (`.`).
- Elles permettent d'interagir avec les données de l'objet.

Mise à jour des attributs

```python
# Modifier les attributs directement
personne2.nom = "Robert"
print(personne2.se_presenter())  # "Je m'appelle Robert et j'ai 30 ans."

# Modifier l'âge
personne2.age = 31
print(personne2.se_presenter())  # "Je m'appelle Robert et j'ai 31 ans."
```

- Les attributs peuvent être modifiés directement via l'objet.
- Cette approche simple est idéale pour débuter.

Concepts clés de la POO

- **Classe :** Modèle ou plan pour créer des objets.
- **Objet :** Instance d'une classe, avec des données et des comportements.
- **Attributs :** Données associées à un objet.
- **Méthodes :** Fonctions qui permettent aux objets d'interagir avec leurs données.

Gestion des Exceptions

Exceptions Personnalisées

- Création de classes d'exceptions personnalisées en héritant de `Exception`.
- Permet de gérer des erreurs spécifiques à votre application.
- Utilisation de `try`, `except`, `finally` pour gérer les exceptions.
- Améliore la robustesse et la fiabilité du code.

Exemple : Exceptions Personnalisées (Partie 1.1)

```python
class ErreurPersonnalisee(Exception):
    """Exception personnalisée pour les erreurs spécifiques."""
    pass

def division(a, b):
    """Effectue une division et lève une exception personnalisée si b est zéro.
    if b == 0:
        raise ErreurPersonnalisee("Division par zéro impossible.")
    return a / b
```

Exemple : Exceptions Personnalisées (Partie 1.2)

```
# Gestion des exceptions
try:
    result = division(10, 0)
except ErreurPersonnalisee as e:
    print(f"Caught an error: {e}")  # Affiche "Division par zéro impossible."
finally:
    print("Opération terminée.")  # Toujours exécuté
```

Exemple : Exceptions Personnalisées - Partie 2

```python
# Gestion de plusieurs exceptions
try:
    fichier = open("inexistant.txt", "r", encoding='utf-8')
except FileNotFoundError:
    print("Le fichier 'inexistant.txt' n'existe pas.")
except ErreurPersonnalisee as e:
    print(f"Erreur personnalisée : {e}")
except Exception as e:
    print(f"Une erreur est survenue : {e}")
else:
    print("Fichier ouvert avec succès.")
    fichier.close()
```

Opérations sur les Fichiers

Lecture et Écriture Avancées

- **Lecture ligne par ligne :** Utilisation de `readline()` ou itération directe.
- **Fichiers CSV :** Utilisation du module `csv` pour manipuler des données tabulaires.
- Gestion des exceptions lors de l'ouverture des fichiers.
- Modes d'ouverture : `'r'` (lecture), `'w'` (écriture), `'a'` (ajout).

Exemple : Lecture et Écriture (Partie 1.1)

```
# Lecture d'un fichier ligne par ligne
# Assurez-vous que 'fichier.txt' existe ou créez-le pour tester
try:
    with open('fichier.txt', 'r', encoding='utf-8') as f:
        print("Contenu de 'fichier.txt' :")
        for ligne in f:
            print(ligne.strip())  # strip() supprime les espaces blancs en débu
except FileNotFoundError:
    print("Le fichier 'fichier.txt' n'existe pas. Créons-le avec des exemples."
```

Exemple : Lecture et Écriture (Partie 1.2)

```
    # Création du fichier avec des exemples
    with open('fichier.txt', 'w', encoding='utf-8') as f:
        f.write("Première ligne\nDeuxième ligne\nTroisième ligne")
    # Lecture du fichier nouvellement créé
    with open('fichier.txt', 'r', encoding='utf-8') as f:
        print("Contenu de 'fichier.txt' :")
        for ligne in f:
            print(ligne.strip())
```

Exemple : Lecture ligne par ligne et Écriture - Partie 2

```python
# Écriture dans un fichier
with open('sortie.txt', 'w', encoding='utf-8') as f:
    f.write("Ceci est une ligne de texte.\n")
    f.write("Ceci est une autre ligne.")
print("\nÉcriture dans 'sortie.txt' terminée.")

# Lecture et écriture binaire
# Assurez-vous que 'image.jpg' existe ou remplacez par un fichier binaire dispo
print("\n--- Lecture et écriture binaire ---")
try:
    with open('image.jpg', 'rb') as f:
        contenu = f.read()
        # Traitement du contenu binaire (exemple : affichage des premiers octet
        print(f"Contenu de l'image (premiers 10 octets) : {contenu[:10]}...")
except FileNotFoundError:
```

Exemple : Fichiers CSV (Partie 1.1)

```python
import csv

# Lecture d'un fichier CSV
try:
    with open('donnees.csv', 'r', newline='', encoding='utf-8') as csvfile:
        lecteur = csv.reader(csvfile)
        print("Contenu de 'donnees.csv' :")
        for ligne in lecteur:
            print(ligne)
except FileNotFoundError:
    print("Le fichier 'donnees.csv' n'existe pas. Créons-le avec des exemples.'
```

Exemple : Fichiers CSV (Partie 1.2)

```
    # Création du fichier CSV avec des exemples
    with open('donnees.csv', 'w', newline='', encoding='utf-8') as csvfile:
        ecrivain = csv.writer(csvfile)
        ecrivain.writerow(['Nom', 'Âge'])
        ecrivain.writerow(['JL', 21])
        ecrivain.writerow(['Soulayman', 22])
        ecrivain.writerow(['Salma', 23])
```

Exemple : Fichiers CSV (Partie 1.3)

```
    # Lecture du fichier nouvellement créé
    with open('donnees.csv', 'r', newline='', encoding='utf-8') as csvfile:
        lecteur = csv.reader(csvfile)
        print("Contenu de 'donnees.csv' :")
        for ligne in lecteur:
            print(ligne)
```

Exemple : Manipulation de fichiers CSV - Partie 2

```python
# Écriture dans un fichier CSV
with open('export.csv', 'w', newline='', encoding='utf-8') as csvfile:
    ecrivain = csv.writer(csvfile)
    ecrivain.writerow(['Sanae', 'Âge'])
    ecrivain.writerow(['BAKA', 28])
    ecrivain.writerow(['L2atrache', 22])
print("\nÉcriture dans 'export.csv' terminée.")
```

Interfaces Graphiques (GUI)

Widgets de Base avec Tkinter

- Bouton : `Button`
- Entrée : `Entry`
- Label : `Label`
- Dispositions : `pack()`, `grid()`, `place()`
- Création d'interfaces interactives et responsives.

Exemple : Widgets de Base avec Tkinter - Partie 1

```
1  import tkinter as tk
2
3  def saluer():
4      """Fonction pour saluer l'utilisateur."""
5      nom = entree.get()
6      if nom.strip():
7          label_resultat.config(text=f"Hello, {nom}!")
8      else:
9          label_resultat.config(text="Veuillez entrer un nom.")
10
11 # Création de la fenêtre principale
12 fenetre = tk.Tk()
13 fenetre.title("Application de Salutation")
```

Exemple : Widgets de Base avec Tkinter - Partie 2

```
# Label pour l'invite
label_invite = tk.Label(fenetre, text="Entrez votre nom :")
label_invite.pack(pady=10)

# Champ de saisie
entree = tk.Entry(fenetre)
entree.pack(pady=5)

# Bouton pour déclencher l'action
bouton = tk.Button(fenetre, text="Saluer", command=saluer)
bouton.pack(pady=10)

# Label pour afficher le résultat
label_resultat = tk.Label(fenetre, text="")
label_resultat.pack(pady=5)
```

Exemple : Widgets de Base avec Tkinter - Partie 3

```
# Bouton pour quitter
bouton_quitter = tk.Button(fenetre, text="Quitter", command=fenetre.quit)
bouton_quitter.pack(pady=10)

# Démarrage de la boucle principale
fenetre.mainloop()
```

Programmation Avancée

Générateurs et Itérateurs

- **Générateurs :** Fonctions qui utilisent `yield` pour produire une séquence de valeurs.
- **Itérateurs :** Objets qui implémentent les méthodes
- Permettent de gérer efficacement les séquences de grande taille.
- Utiles pour économiser de la mémoire en générant les valeurs à la demande.

Exemple : Générateurs et Itérateurs - Partie 1

```python
def compteur(n):
    """Générateur qui compte de 0 à n-1."""
    i = 0
    while i < n:
        yield i
        i += 1

# Utilisation du générateur
print("\n--- Utilisation du générateur compteur ---")
for nombre in compteur(5):
    print(nombre)  # Affiche 0, 1, 2, 3, 4
```

Exemple : Générateurs et Itérateurs - Partie 2

```python
# Générateur infini
def nombres_pairs():
    """Générateur infini de nombres pairs."""
    i = 0
    while True:
        yield i
        i += 2

pairs = nombres_pairs()
print("\n--- Utilisation du générateur infini nombres_pairs ---")
print(next(pairs))  # 0
print(next(pairs))  # 2
print(next(pairs))  # 4
```

Programmation Asynchrone avec `asyncio`

- Permet d'exécuter des tâches de manière asynchrone.
- Utilise les mots-clés `async` et `await`.
- Utile pour les opérations d'entrée/sortie non bloquantes.
- Améliore les performances des applications réseau et serveurs web.

Exemple : Programmation Asynchrone avec `asyncio` - Partie 1

```python
import asyncio

async def dire(texte, delai):
    """Affiche un texte après un délai spécifié."""
    await asyncio.sleep(delai)
    print(texte)
```

Exemple : Programmation Asynchrone avec `asyncio` - Partie 2

```python
async def main_async():
    """Fonction principale asynchrone."""
    # Création de tâches asynchrones
    tache1 = asyncio.create_task(dire("HALA", 2))
    tache2 = asyncio.create_task(dire("WALLAH", 1))

    # Attente de la complétion des tâches
    await tache1
    await tache2

# Exécution de la boucle d'événements
print("\n--- Exécution de la boucle d'événements asyncio ---")
asyncio.run(main_async())
```

Conclusion

Conclusion

- Python offre une multitude de fonctionnalités pour tous les niveaux.
- Continuer à pratiquer est la clé pour maîtriser le langage.
- N'hésitez pas à explorer les bibliothèques tierces pour étendre vos capacités.
- Rejoignez la communauté Python pour partager et apprendre davantage.
- Restez curieux et expérimentez avec de nouveaux projets.

Ressources Utiles :

- Documentation officielle : `https://docs.python.org/3/`
- Tutoriels en ligne : `https://www.w3schools.com/python/`
- Communauté Stack Overflow : `https://stackoverflow.com/questions/tagged/python`
- Livres recommandés : "Apprendre à programmer avec Python" de Gérard Swinnen.

Merci pour votre attention