# Cours Python EPSI

---

# Cours 1 : Introduction à Python

## Avant de débuter

### Installation de l'environnement Python

Python est par défaut fourni avec la plupart des distributions Linux. Cependant, la version installée peut varier en fonction de la distribution choisie.

Pour utiliser la version la plus récente de Python (3.11 pour ce module), il est recommandé d'ajouter le dépôt **deadsnakes** à vos sources d'apt.

Pour installer Python 3.11, exécutez les commandes suivantes :

```
sudo apt install software-properties-common
sudo add-apt-repository ppa:deadsnakes/ppa
sudo apt update
sudo apt-get install python3.11
```

### Éditeur de Code

Bien que Python soit facile à écrire avec des éditeurs de texte simples comme emacs, vim ou nano, il est conseillé d'utiliser un IDE pour une meilleure expérience de développement. Je recommande PyCharm (Community Edition) ou VSCode pour leurs fonctionnalités avancées et leur support étendu.

## Python : Les Bases

Contrairement à C/C++, Python est un langage interprété et à typage dynamique. Cela implique une manière différente de penser et de coder.

### Variables, Types et Fonctions

Voyons comment les concepts de base en Python se comparent à C/C++ :

```c
int sum(int a, int b) {
    return a + b;
}

int strlen(char *str) {
    int i = 0;

    while (str[i] != 0) {
        i = i + 1;
    }
    return i;
}

int main() {
  int a = 3;
  float b = 3.;
  char *str = "Hello World";

  // printf("Sum of %d + %f = %d, len of str %d", a, b, sum(a, b),
  ↪  strlen(str)); // Ne compile pas
  printf("Sum of %d + %f = %d, len of str %d", a, b, sum(a, (int)b),
↪  strlen(str));
}
```

### Exemple en C/C++

```python
def sum(a, b):
    return a + b


def strlen(an_str):  # Équivalent à len(an_str)
    i = 0
    for _ in an_str:  # Itération sur chaque caractère de la chaîne
        i += 1
    return i


if __name__ == '__main__':
    ex_a = 3
    ex_b = 3.
    ex_str = "Hello World"
    print(f"Sum of {ex_a} + {ex_b} = {sum(ex_a, ex_b)}, len of str is
    ↪  {strlen(ex_str)}")
```

**Équivalent en Python**

### Structures de Contrôle

**Conditions** En Python, les structures conditionnelles sont définies avec les mots-clés `if`, `elif`, et `else`. Voici comment cela se compare à C/C++ :

**Exemple en Python :**

```python
age = 20

if age < 18:
    print("Vous êtes mineur.")
elif 18 <= age < 65:
    print("Vous êtes adulte.")
else:
    print("Vous êtes senior.")
```

**Comparaison avec C/C++ :**

```c
int age = 20;

if (age < 18) {
    printf("Vous êtes mineur.\n");
} else if (age >= 18 && age < 65) {
    printf("Vous êtes adulte.\n");
} else {
    printf("Vous êtes senior.\n");
}
```

### Boucles

- **Boucle `while`**

En Python, la boucle `while` fonctionne de manière similaire à C/C++. Elle continue tant que la condition est vraie.

**Exemple en Python :**

```python
count = 0
while count < 5:
    print(count)
    count += 1
```

**Comparaison avec C/C++ :**

```c
int count = 0;
while (count < 5) {
```

```
    printf("%d\n", count);
    count++;
}
```

- **Boucle `for` avec `in`**

En Python, la boucle `for` est souvent utilisée pour itérer sur des éléments d'une séquence comme une liste, un tuple ou une chaîne de caractères.

**Exemple en Python :**

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

**Comparaison avec C/C++ :**

En C/C++, une boucle `for` pour itérer sur une séquence ressemble généralement à ceci :

```
#include <stdio.h>

int main() {
    const char *fruits[] = {"pomme", "banane", "cerise"};
    int n = sizeof(fruits) / sizeof(fruits[0]);
    for (int i = 0; i < n; i++) {
        printf("%s\n", fruits[i]);
    }
    return 0;
}
```

**Différences Clés :**

- **Typage Dynamique** : Pas de type explicite pour chaque variable ; Python déduit les types automatiquement.
- **Syntaxe Simplifiée** : Pas de point-virgule `;` à la fin de chaque ligne et pas d'accolades `{}` pour définir les blocs de code. Les blocs sont délimités par l'indentation.
- **Conversion Automatique** : Pas de cast explicite lors de l'addition d'entiers et de flottants.
- **Syntaxe de Condition** : Les parenthèses `()` sont facultatives pour les conditions en Python.

## Python : Les structures de données

### Types primitifs en python

- bool
- int
- float
- str

### A propos de la taille des types

Concernant les entiers, Python étant un langage haut-niveau, il n'y a pas de notion de taille comme dans d'autre langage (int vs long / float vs double). Ce qu'il faut retenir => pas de limite de taille (si ce n'est la mémoire de la machine)

```
2 ** 128 - 2 ** 127  # Equivalent to 2^128 - 2^127
# 170141183460469231731687303715884105728
```

### Les collections

Python offre de base plusieurs conteneurs, dont les principaux sont:

- list
- dict
- set
- tuple
- iterable
- collections

Les implémentations sont précisées sur cette page

**list** Comparable à un array, mais sans qu'il soit nécessaire de spécifier une taille lors de la création.

*En interne la liste utilise un tableau qu'elle aggrandie lorsque celui-ci devient trop petit*

```
# List
list_regular = []
for i in range(0, 10):  # range return an iterable
    list_regular.append(i)
# Or
list_one_line = [i for i in range(0, 10)]

# Or
list_from_iterable = list(range(0, 10))
```

**dict** Généralement appelé Map dans la plupart des langages de programmation, il permet d'associer une clé unique à une valeur.

```
# dict
dict_regular = {}  # or dict_a = dict()
for i in range(0, 10):
    dict_regular[i] = i ** 2
# Or
dict_one_liner = {i: i ** 2 for i in range(0, 10)}
# Or
dict_from_iterable = dict([(i, i ** 2) for i in range(0, 10)])
```

**set** Un set est une collection non ordonnée d'éléments uniques. Il est utile pour les opérations mathématiques comme l'union, l'intersection, et la différence.

```

# set
set_a = {1, 2, 3, 4, 5}
set_b = set([3, 4, 5, 6, 7])

# Opérations de base
union_set = set_a | set_b  # Union
intersection_set = set_a & set_b  # Intersection
difference_set = set_a - set_b  # Différence

print("Set A:", set_a)
print("Set B:", set_b)
print("Union:", union_set)  # 1 2 3 4 5 6 7
print("Intersection:", intersection_set)  # 3 4 5
print("Difference:", difference_set)  # 1 2 6 7
```

**Tuples** Grossièrement, une liste figée

```
a_tuple = ('a', 2)
```

**Iterable** En Python, un objet itérable est tout objet capable de retourner ses membres un par un. Cela inclut des structures comme les listes, les tuples, les ensembles et les dictionnaires.

```
# Exemple d'itération sur une liste
iterable_list = [1, 2, 3, 4, 5]
for item in iterable_list:
    print(item)
```

```python
# Exemple d'itération sur un dictionnaire
iterable_dict = {'a': 1, 'b': 2, 'c': 3}
for key, value in iterable_dict.items():
    print(f"Key: {key}, Value: {value}")
```

### IO

**stdin/stdout/stderr** En Python, les flux d'entrée/sortie standard sont disponibles via les objets sys.stdin, sys.stdout, et sys.stderr. Ils permettent de lire et d'écrire des données en utilisant ces flux.

Exemple de lecture de stdin :

```python
import sys

print("Entrez votre nom : ", end="")
name = sys.stdin.readline().strip()
print(f"Bonjour, {name} !")
```

Exemple d'écriture dans stdout et stderr :

```python
import sys

print("Ceci est une sortie standard", file=sys.stdout)
print("Ceci est une erreur", file=sys.stderr)
```

**Manipulation de fichiers** Python propose plusieurs méthodes pour lire et écrire des fichiers. Voici quelques exemples de base :

Lire un fichier :

```python
with open('example.txt', 'r') as file:
    content = file.read()  # Charge l'intégralité du fichier en
↪   mémoire
    print(content)
```

Écrire dans un fichier :

```python
with open('example.txt', 'w') as file:
```

```
    file.write("Hello, World!\n")
    file.write("Python is great!")
```

Ajouter à un fichier :

```
with open('example.txt', 'a') as file:
    file.write("\nAppended text")
```

### Les Classes

En Python, les classes sont définies avec le mot-clé class. La syntaxe est simple et les concepts de base incluent les attributs, les méthodes, et les constructeurs.

Exemple de classe :

```
class Person:
    def __init__(self, name, age):
        self.name = name  # Attribut public
        self.age = age

    def greet(self):  # Méthode
        return f"Hello, my name is {self.name} and I am {self.age}
        ↪  years old."


# Création d'une instance
person = Person("Alice", 30)
print(person.greet())
```

En Python, les attributs sont publics par défaut, et pour créer des attributs privés, on utilise une convention de nommage avec un préfixe double underscore ___.

Exemple avec attributs privés :

```
class Person:
    def __init__(self, name, age):
        self.__name = name
        self.__age = age

    def greet(self):
        return f"Hello, my name is {self.__name} and I am {self.__age}
        ↪  years old."


# Création d'une instance
person = Person("Alice", 30)
print(person.greet())
# Accéder à un attribut privé directement n'est pas possible :
# print(person.__name)  # Cela générera une erreur
```

## TP

Lire un fichier csv et le stocker en mémoire. Puis ensuite proposer un prompt à l'utilisateur permettant de :

- Chercher une ville ("get city $city") –> Retourne la ville en entier
- Lister les villes présente dans un département ("get department $department")
- Ajouter une nouvelle ville (add *city*,department,*country*,population)
- Ecrire le contenu de la mémoire dans le fichier db.csv (flush)
- Supprimer une ville ("delete $city")
- Supprimer un departement ("delete $department")

L'affichage se fait au format csv.

## Ressources Complémentaires

- Documentation officielle de Python : https://docs.python.org/fr/3/tutorial/index.html
- Tutoriel POO Python : https://www.w3schools.com/python/python_classes.asp

---

**Fin du Cours 1**