Introduction à Python - Les Bases

H. Abouabid

14 Décembre 2023

Qu'est-ce que Python ?

- **Introduction à Python** :
  - Python est un *langage de programmation interprété*, ce qui signifie que les instructions sont exécutées directement sans compilation préalable. Cette caractéristique le rend particulièrement adapté pour le prototypage rapide et les itérations de développement.
  - En tant que *langage de haut niveau*, Python permet une abstraction significative par rapport au langage machine, rendant le code plus lisible, compréhensible et maintenable.

Qu'est-ce que Python ?

- **Caractéristiques Principales de Python** :
  - *Facilité d'apprentissage* : Sa syntaxe claire et sa communauté active en font un choix excellent pour les débutants en programmation.
  - *Interprétable* : Python est exécuté dans un environnement d'exécution, ce qui facilite le test et le débogage de programmes complexes.
  - *Polyvalence* : Utilisé dans une vaste gamme d'applications, de l'automatisation de scripts simples au développement de systèmes complexes, Python est extrêmement adaptable.
  - *Applications diverses* : Python est omniprésent dans le développement web (via des frameworks comme Django et Flask), l'analyse de données (avec des outils tels que Pandas et NumPy), l'intelligence artificielle (grâce à des bibliothèques comme TensorFlow et PyTorch), et dans le calcul scientifique.

Qu'est-ce que Python ?

- **Python dans la Science des Données** :
  - Python est largement utilisé dans la science des données pour des tâches telles que le nettoyage de données, l'analyse statistique, la visualisation de données, et le machine learning.
  - Des bibliothèques comme Pandas pour la manipulation de données, Matplotlib et Seaborn pour la visualisation, et Scikit-learn pour le machine learning, rendent Python particulièrement puissant pour l'analyse de données.
  - Sa capacité à intégrer avec d'autres langages et outils, comme SQL pour les bases de données, et son support pour le calcul parallèle et distribué, le rendent idéal pour travailler avec de grandes ensembles de données.

Installation de Python

- **Guide d'Installation Étape par Étape** :
  - *Téléchargement* : Accédez au site officiel de Python, python.org, et téléchargez la dernière version stable pour votre système d'exploitation.
  - *Installation* : Lancez l'installateur téléchargé. Sur Windows, assurez-vous de cocher l'option "Add Python to PATH" avant de commencer l'installation.
  - *Configuration Post-Installation* : Il peut être nécessaire de configurer certaines variables d'environnement, surtout sous Linux et macOS, pour faciliter l'accès aux commandes Python et Pip (gestionnaire de paquets Python) depuis le terminal.
- **Vérification de l'Installation de Python** :
  - Après l'installation, ouvrez un terminal ou une invite de commande et tapez 'python –version' (ou 'python3 –version' sur certains systèmes Linux/macOS). Si Python est correctement installé, cette commande affichera la version installée.
  - Il est également recommandé de vérifier l'installation de Pip, le gestionnaire de paquets de Python, en exécutant 'pip –version'.

Installation de Python

- **Installation de Python sur Différentes Plateformes** :
  - *Windows* : L'installateur inclut l'option d'ajouter Python au PATH, facilitant l'accès depuis l'invite de commande.
  - *macOS* : Python peut être installé via l'installateur téléchargé ou en utilisant des gestionnaires de paquets comme Homebrew.
  - *Linux* : Python est souvent pré-installé sur de nombreuses distributions Linux. Des versions spécifiques peuvent être installées via le gestionnaire de paquets de la distribution, comme apt pour Ubuntu ou yum pour Fedora.

Votre premier programme Python

```
print("Bonjour le monde!")
```

- **Structure du Programme** :
  - Ce programme est un exemple classique de "Hello World" dans le monde de la programmation.
  - En une seule ligne de code, il démontre la capacité de Python à exécuter une tâche simple : afficher un message à l'utilisateur.
- **Explication de la Syntaxe** :
  - *La fonction print* : 'print()' est une fonction intégrée en Python, utilisée pour afficher le texte ou la valeur de variable que l'utilisateur souhaite voir.
  - *Guillemets pour les chaînes de caractères* : Les guillemets (" ") sont utilisés pour délimiter des chaînes de caractères en Python.
  - *Parenthèses* : Les parenthèses après 'print' indiquent qu'il s'agit d'une fonction.
  - *Importance de la simplicité* : Ce programme illustre également la nature concise et lisible de Python, qui rend le langage accessible aux débutants tout en étant puissant pour les développeurs avancés.

Exercice 1

Tâche

Écrire un programme pour imprimer votre nom et la date du jour.

Exercice 1

Tâche

Écrire un programme pour imprimer votre nom et la date du jour.

- **Objectifs de l'Exercice** :
  - Cet exercice vise à familiariser les apprenants avec les opérations de base en Python, notamment l'affichage de texte et l'utilisation de fonctions intégrées.
  - Il introduit également la notion de manipulation de dates et de chaînes de caractères en Python.
- **Étapes Suggérées** :
  - *Affichage du Nom*
  - *Affichage de la Date* :
  - *Combiner les Deux Éléments*

Variables et Types de Données

- Définition des variables.
- Types de données courants : int, float, string, boolean.
- Exemple : Déclaration et utilisation de différentes variables en Python.

Exercice 2

Tâche

Créer des variables de différents types et les imprimer.

Opérateurs en Python

- Opérateurs arithmétiques.
- Opérateurs de comparaison.
- Exemple : Utilisation des opérateurs pour effectuer des calculs et des comparaisons simples.

Exercice 3

Tâche

Écrire un programme pour calculer la surface d'un rectangle.

Structures de Contrôle : If-else

- Explication des instructions conditionnelles.
- Syntaxe et exemples.
- Exemple : Programme utilisant if-else pour prendre des décisions en fonction de conditions.

Exercice 4

Tâche

Écrire un programme qui détermine si un nombre est pair ou impair.

Structures de Contrôle : Boucles

- Introduction aux boucles (for, while).
- Instructions de contrôle des boucles (break, continue).
- Exemple : Utilisation des boucles pour répéter des actions.

Exercice 5

Tâche

Écrire un programme pour imprimer les 10 premiers nombres naturels en utilisant une boucle.

Agenda de la Semaine 3

- Introduction aux Fonctions
- Définir et Appeler des Fonctions
- Arguments et Valeurs de Retour des Fonctions
- Introduction aux Modules
- Importation et Utilisation des Modules
- Exercice : Construire un Calculateur Simple

Introduction aux Fonctions

- Définition : Une fonction est un bloc de code organisé et réutilisable utilisé pour effectuer une action unique et liée.
- Avantages : Réutilisation du code, modularité, et masquage de l'information.
- Exemple : Fonctions pour calculer des opérations mathématiques simples comme l'addition, la soustraction, etc.

Définir et Appeler des Fonctions

Syntaxe

```
def nom_fonction(paramètres):
    """docstring"""
    instruction(s)
```

Exemple

```
def saluer(nom):
    """Saluer quelqu'un par son nom."""
    print("Bonjour, " + nom + " !")
saluer("Alice")
```

- Explication : La fonction ‘saluer‘ prend un paramètre ‘nom‘ et affiche un message de salutation.
- Utilisation : Peut être utilisée pour personnaliser les messages dans les applications.

Exercice 1 : Créer une Fonction

Créez une fonction qui prend un nombre en argument et retourne son carré.

Exemple

```
def carre(nombre):
    return nombre * nombre
```

- Tâche : Écrivez et testez cette fonction dans un environnement Python.
- Application : Cette fonction peut être utilisée dans des calculs mathématiques ou des applications scientifiques.

Arguments dans les Fonctions

- Les fonctions peuvent prendre des arguments pour le traitement.
- Syntaxe : `nom_de_fonction(arg1, arg2, ...)`
- Types d'arguments :
  - Arguments requis : Ceux sans lesquels la fonction ne peut pas s'exécuter. Exemple : `def addition(a, b)` où `a` et `b` sont requis.
  - Arguments par mot-clé : Permettent de passer des arguments dans un ordre quelconque en utilisant le nom de l'argument. Exemple : `def saluer(nom=''Mundo'')` appelé avec `saluer(nom=''Alice'')`.
  - Arguments par défaut : Ils ont une valeur par défaut. Si l'argument n'est pas fourni, la valeur par défaut est utilisée. Exemple : `def saluer(nom=''Mundo'')`.
  - Arguments de longueur variable : Ils permettent à une fonction de recevoir un nombre quelconque d'arguments. Exemple : `def somme(*nombres)` et appelé par `somme(1, 2, 3)`.

Valeurs de Retour

- Les fonctions peuvent renvoyer des valeurs en utilisant l'instruction `return`.
- Syntaxe : `return [expression]`
- La fonction se termine dès qu'elle atteint l'instruction `return`.

Exemple

```
def addition(a, b):
    return a + b
resultat = addition(5, 3)
print(resultat) % Sortie: 8
```

- Une fonction peut retourner plusieurs valeurs sous forme de tuple. Exemple : `return a, b`.

Portée des Variables

- La portée détermine l'accessibilité des variables.
- Types de portée :
  - Portée locale : Les variables locales sont accessibles uniquement dans la fonction. Exemple : `a` dans `def fonction() { int a = 5 }`.
  - Portée globale : Les variables globales sont accessibles dans tout le programme. Déclarées en dehors de toute fonction.
  - Portée non locale : Dans les fonctions imbriquées, une variable non locale est accessible dans la fonction enfant, mais pas dans la fonction parente.
- Les variables globales et locales peuvent avoir le même nom mais sont différentes. La variable locale "cache" la variable globale dans la fonction.

Introduction aux Modules et Bibliothèques Python

- Module : Un fichier Python contenant un ensemble de fonctions et de variables.
- Bibliothèque : Une collection de modules.
- Avantages : Réutilisation de code, partitionnement des espaces de noms, organisation.
- Exemple de Module : Le module `datetime` pour la manipulation des dates et des heures.
- Exemple de Bibliothèque : `Pandas`, largement utilisée en science des données pour la manipulation de données.

Importer et Utiliser des Modules

- Utilisez l'instruction `import` pour utiliser des modules.
- Syntaxe : `import nom_du_module`
- Accédez aux fonctions avec `nom_du_module.nom_de_fonction()`
- Importation sélective : `from nom_du_module import nom_de_fonction`
- Renommage lors de l'importation : `import nom_du_module as alias`

Importer et Utiliser des Modules

Exemple

```
import math
print(math.sqrt(16)) # Sortie : 4.0

from datetime import datetime
print(datetime.now()) # Affiche la date et l'heure actuell

import numpy as np
print(np.array([1, 2, 3])) # Crée un tableau numpy
```

Pratique : Création d'un Calculateur Simple

- Objectif : Créer un calculateur simple en utilisant des fonctions.
- Implémenter les opérations de base : Addition, Soustraction, Multiplication, Division.
- Utiliser des fonctions pour encapsuler chaque opération.
- Explication sur l'importance de l'encapsulation pour maintenir le code organisé et facile à maintenir.

Calculateur - Exemples de Fonctions

Fonction d'Addition

```
def add(x, y):
    return x + y
```

Fonction de Soustraction

```
def subtract(x, y):
    return x - y
```

Calculateur - Programme Principal

Programme Principal

- Programme principal pour utiliser les fonctions du calculateur
- Ajouter la logique de saisie utilisateur et d'appel de fonction ici
- Exemple d'interface utilisateur pour choisir l'opération

Exercice : Améliorer le Calculateur

- Ajouter la gestion des erreurs pour la division par zéro.
- Implémenter une fonction racine carrée.
- Permettre une opération continue jusqu'à ce que l'utilisateur décide de quitter.
- Ajouter des exemples sur la manière de gérer les entrées utilisateur et les boucles pour des opérations continues.

Lists

- Definition and Creation of Lists
- Dynamic Nature and Flexibility
- Accessing List Elements
- List Operations : Append, Remove, Sort, Reverse, Index
- Slicing Lists

List Example

```
# Creating a list
my_list = [1, 2, 3, 4, 5]

# Accessing elements
print(my_list[0])  # Output: 1
print(my_list[-1]) # Output: 5

# Appending an element
my_list.append(6)  # Adds 6 to the end

# Removing an element
my_list.remove(3)  # Removes the first occurrence of 3
# Sorting a list
my_list.sort()     # Sorts the list in ascending order
# Reversing a list
my_list.reverse()  # Reverses the order of elements
```

List Operations Detailed Explanation

Detailed Explanation

- **Append :** Adds a new element at the end of the list.
- **Remove :** Removes the first occurrence of the specified value.
- **Sort :** Sorts the elements in ascending order by default. Use ‘sort(reverse=True)‘ for descending order.
- **Reverse :** Reverses the current ordering of elements in the list.
- **Index :** Returns the index of the first occurrence of the specified element.
- **Slicing :** Allows accessing a subset of the list, specifying a start and an end index.

List Exercise

Exercise 1

Create a list of your favorite fruits and write a function to add a new fruit to the list. Then, demonstrate how to sort and reverse the list.

Tuples

- Definition and Creation of Tuples
- Immutability of Tuples
- Use Cases for Tuples
- Tuple Operations : Indexing, Slicing, Concatenation

Tuple Example

```
# Creating a tuple
my_tuple = (1, 2, 3, 4, 5)
# Accessing elements
print(my_tuple[1])  # Output: 2
# Slicing a tuple
print(my_tuple[1:4])  # Output: (2, 3, 4)
```

Tuple Exercise

Exercise 2

Create a tuple containing different types of data (integer, string, etc.) and demonstrate how to access and slice these elements.

Sets

- Definition and Creation of Sets
- Uniqueness of Elements in Sets
- Set Operations : Add, Remove, Union, Intersection

Set Example

```
# Creating a set
my_set = {1, 2, 3, 4, 5}
# Adding an element
my_set.add(6)
# Removing an element
my_set.remove(2)
# Set Operations
another_set = {4, 5, 6, 7, 8}
print(my_set.union(another_set))
# Output: {1, 3, 4, 5, 6, 7, 8}
```

Set Exercise

Exercise 3

Create two sets of numbers and demonstrate how to perform union, intersection, and difference operations.

Dictionaries

- Definition and Creation of Dictionaries
- Key-Value Pairs in Dictionaries
- Dictionary Operations : Add, Update, Delete
- Iterating over Dictionaries

Dictionary Example

```
# Creating a dictionary
my_dict = {'apple': 10, 'banana': 5, 'orange': 8}
# Accessing value by key
print(my_dict['apple'])  # Output: 10
# Adding a new key-value pair
my_dict['grape'] = 15
# Updating an existing key
my_dict['banana'] = 6
```

Dictionary Exercise

Exercise 4

Create a dictionary representing a phone book, with names as keys and phone numbers as values. Write a function to add a new contact to the phone book.

Iterating over Data Structures

- Techniques for Iterating : For Loops, While Loops
- Using Iterators and Generators
- List Comprehensions and Dictionary Comprehensions
- Practical Examples of Iteration in Data Structures

Iteration Example

```
# Iterating over a list
for item in [1, 2, 3, 4, 5]:
    print(item)

# List Comprehension
squared_list = [x**2 for x in range(10)]

# Iterating over a dictionary
for key, value in my_dict.items():
    print(f"Key: {key}, Value: {value}")
```

Iteration Exercise

Exercise 5

Given a list of numbers, use a list comprehension to create a new list containing only the even numbers from the original list.

Conclusion

- Recap of Data Structures
- Importance in Python Programming
- Upcoming : Working with Files and Exception Handling

Lecture d'un Fichier

- Opérations de base sur les fichiers : ouvrir, lire, fermer.
- Lire un fichier en Python :
  - `open('nomdefichier', 'r')`
  - Utilisation de l'instruction `with` pour une meilleure gestion.
- Exemple :

Exemple

```
with open('exemple.txt', 'r') as fichier:
    contenu = fichier.read()
    print(contenu)
```

- Exercice : Lire un fichier texte et afficher son contenu.

Écriture dans un Fichier

- Écrire dans des fichiers : ouvrir, écrire, fermer.
- Modes : 'w' pour écrire, 'a' pour ajouter.
- Exemple :

Exemple

```
with open('sortie.txt', 'w') as fichier:
    fichier.write("Bonjour, le monde !")
```

- Exercice : Écrire un programme qui prend une entrée utilisateur et l'écrit dans un fichier.

Comprendre les Chemins de Fichier et les Répertoires

- Chemins de fichier : Chemins absolus vs. Chemins relatifs.
- Travailler avec des répertoires : module `os`.
- Exemple : Lister les fichiers dans un répertoire.

Exemple

```
import os
for nomdefichier in os.listdir('.'):
   print(nomdefichier)
```

- Exercice : Écrire un script pour compter le nombre de fichiers dans un répertoire.

Gestion Basique des Exceptions

- Comprendre les exceptions.
- Bloc try-except.
- Gestion d'exceptions spécifiques.
- Exemple : Gestion d'une FileNotFoundError.

Exemple

```
try:
    with open('inexistant.txt', 'r') as fichier:
        contenu = fichier.read()
except FileNotFoundError:
    print("Fichier non trouvé.")
```

- Exercice : Modifier le script de lecture de fichier pour gérer les exceptions.

Techniques de Débogage

- Stratégies de débogage courantes.
- Utilisation de déclarations print.
- Introduction aux outils de débogage (par exemple, pdb).
- Exercice : Déboguer un script fourni avec des erreurs logiques.

Pratique : Création d'un Analyseur de Fichier Journal

- Objectif : Créer un script Python qui analyse un fichier journal et extrait des informations spécifiques.
- Étapes :
  1. Lecture du contenu du fichier journal.
  2. Identification des modèles (par exemple, messages d'erreur).
  3. Extraction et affichage des informations requises.
- Exemple : Extraire des messages d'erreur d'un fichier journal.
- Exercice : Les étudiants créeront un analyseur de fichier journal basique pour un fichier journal d'exemple fourni.