Introduction to Python and the Working Environment

par :

Dr. Bilal Dendani

جامعة باجي مختار - عنابة
BADJI MOKHTAR - ANNABA UNIVERSITY

Outline

- **Getting Started with Jupyter Notebook**
  - Installing Jupyter Notebook
  - Overview of the Jupyter Notebook Environment
- **First Steps with Python**
  - Introduction
  - Basic Syntax
  - Control Structures in Python
    - Conditional Statements *(if / else)*
    - Loops *(for, while)*
  - Data Structures in Python: **Lists, Tuples, Dictionaries**

Objectives

- Get familiar with the **Jupyter Notebook environment**.
- Learn the **basics of Python programming** (variables, data types, conditions, loops).
- Discover the **basic data structures** (lists, tuples, dictionaries).
- Introduction to **NumPy** and **Pandas**.

Part 1: Getting Started with Jupyter Notebook

Objectifs

- Understand the **role of Jupyter Notebook** in data science.
- Install and launch Jupyter Notebook.
- Get familiar with the **Jupyter Notebook interface**.
- Learn how to run code cells and use keyboard shortcuts.
- Create and save your first notebook.

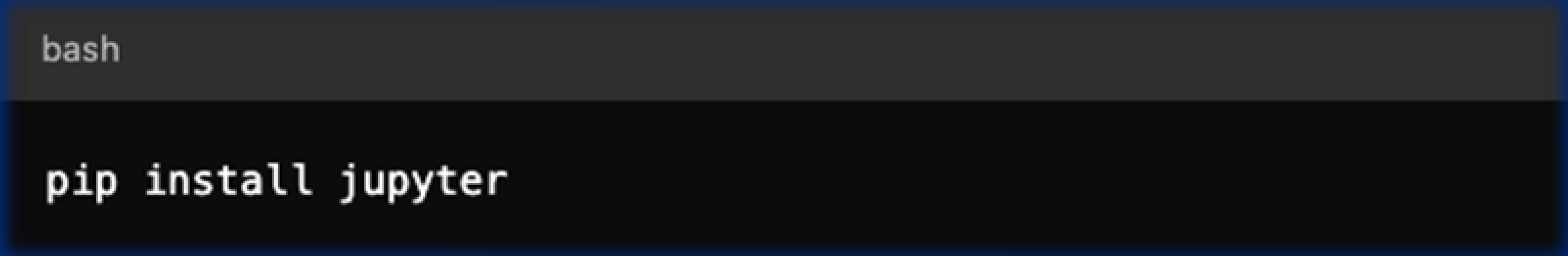
What is Jupyter Notebook?

- Jupyter Notebook is an interactive environment for writing and executing Python code.
- It is the preferred tool for data science because it allows combining code, explanations (text), and visualizations in a single document.
- It supports multiple programming languages (Python, R, Julia, Scala, Java, JavaScript, PHP, etc.),
  but it is mainly used with Python.

Installing Jupyter Notebook

- **Option 1: Install via Anaconda (recommended for beginners)**
  - Download and install Anaconda.
  - Jupyter comes preinstalled with Anaconda, along with other useful tools such as Spyder, Pandas, and NumPy.
- **Option 2: Install via pip (if VS Code is already installed on your machine)**

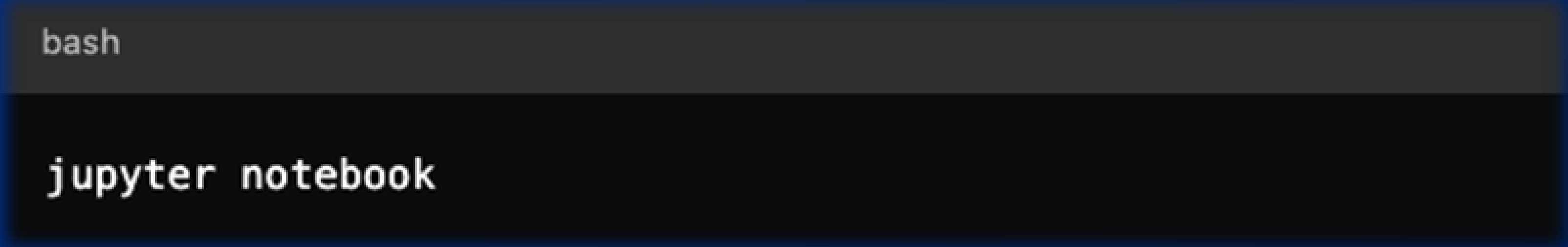
Open a terminal and type:



bash

```
pip install jupyter
```

To launch Jupyter Notebook:

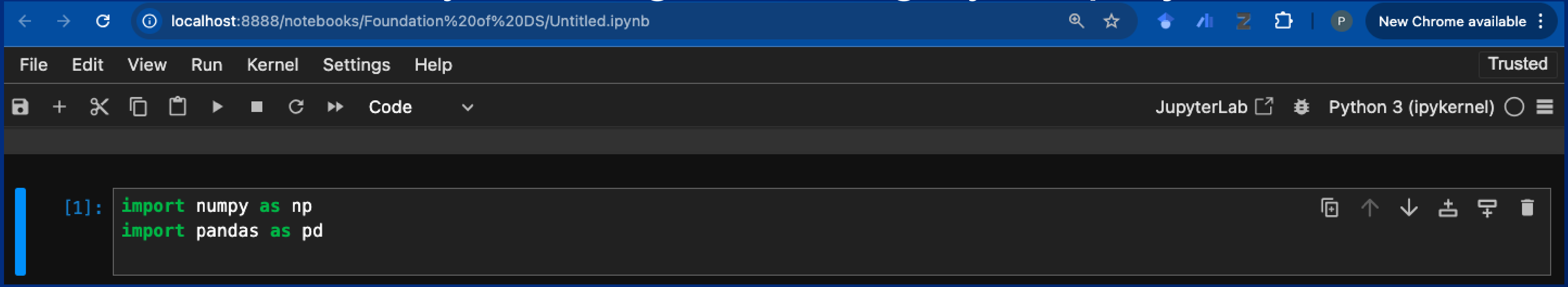


bash

```
jupyter notebook
```

Jupyter Notebook Interface

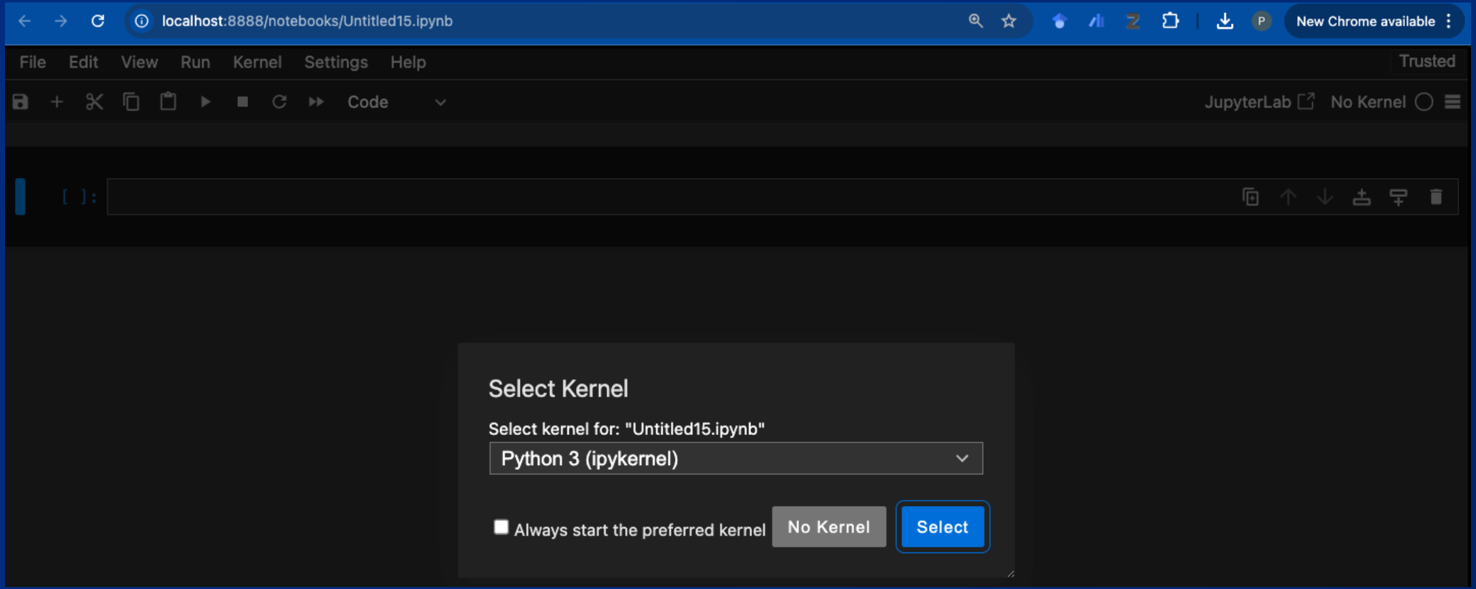
- **Toolbar**: Contains options such as saving, inserting cells, and running code.
- **Cells**: The blocks where you write your code or text.
- "**Kernel**" **Tab:** Allows you to restart or reset the execution environment.
- "**Files**" **Tab:** Lets you navigate through your project files.



```
[1]: import numpy as np
     import pandas as pd
```

Create a New Notebook

- Click on "**New**" in the interface and select "**Python 3**."
- A new notebook will open with a code cell ready to use.
- Save your notebook with a clear and descriptive name using the **.ipynb** extension.



Select Kernel

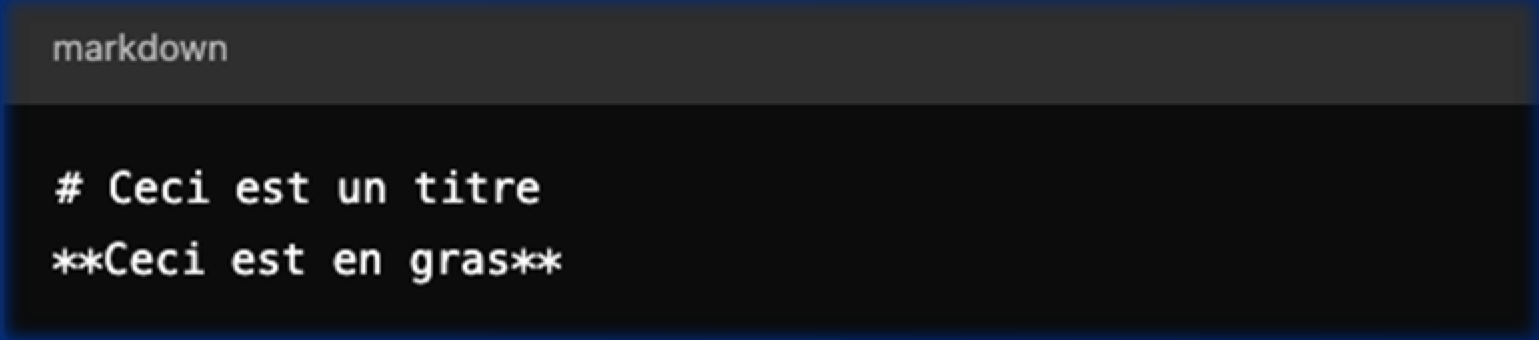
Select kernel for: "Untitled15.ipynb"

Python 3 (ipykernel)

☐ Always start the preferred kernel

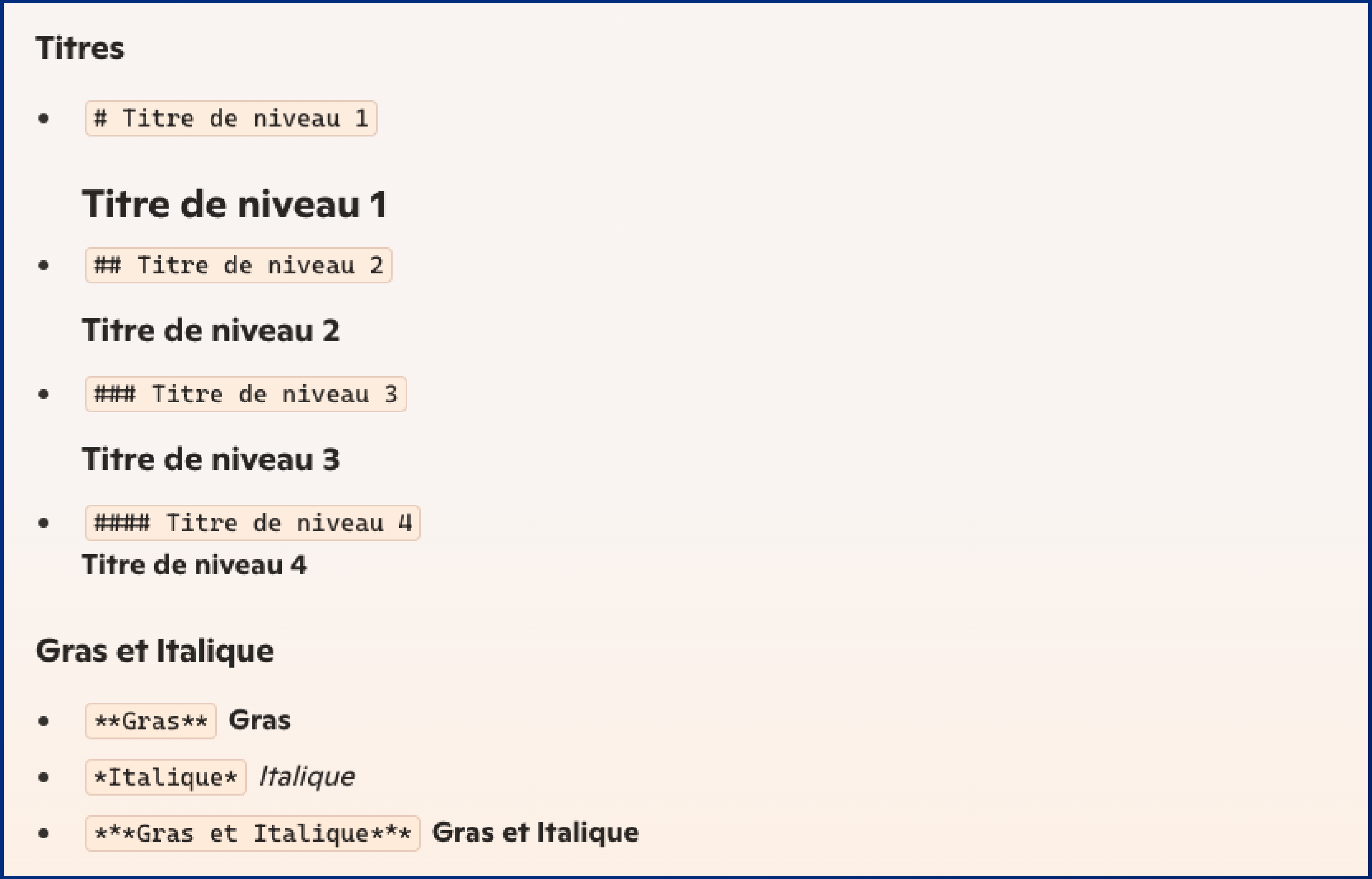
Types of Cells

- There are two main types of cells:
- **Code Cell:** Used to write and execute Python code.
- **Text (Markdown) Cell:** Used to write formatted text, formulas, or explanations.
  - **Markdown** allows you to add titles, bold or italic text, links, and more.
  - **Example of Markdown syntax: # Title, **bold**, *italic***



```markdown
# Ceci est un titre
**Ceci est en gras**
```

Markdown Cell



Titres

- `# Titre de niveau 1`

  Titre de niveau 1

- `## Titre de niveau 2`

  Titre de niveau 2

- `### Titre de niveau 3`

  Titre de niveau 3

- `#### Titre de niveau 4`

  Titre de niveau 4

Gras et Italique

- `**Gras**` **Gras**
- `*Italique*` *Italique*
- `***Gras et Italique***` ***Gras et Italique***

Running a Cell

- Click on the cell and press **Shift + Enter** to execute it.
- The results appear immediately below the cell.
- The **execution order** is shown on the left of the cell (e.g., **In [1]**).

Saving and Exporting Notebooks

- **Automatic saving:** Jupyter automatically saves your work at regular intervals.
- **Exporting:** You can export your notebook in different formats such as **PDF**, **HTML**, or **Python script (.py)**.
- Go to **“File” > “Download as”** to choose the desired export format.

Practical Exercise: Create and Run a Notebook

1. Create a new notebook.
2. In a code cell, display the message: **Welcome to Jupyter**.
3. Add a **Markdown cell** as a title to explain the purpose of the notebook.
4. Save and rename the file.

Part 2: Getting Started with Python

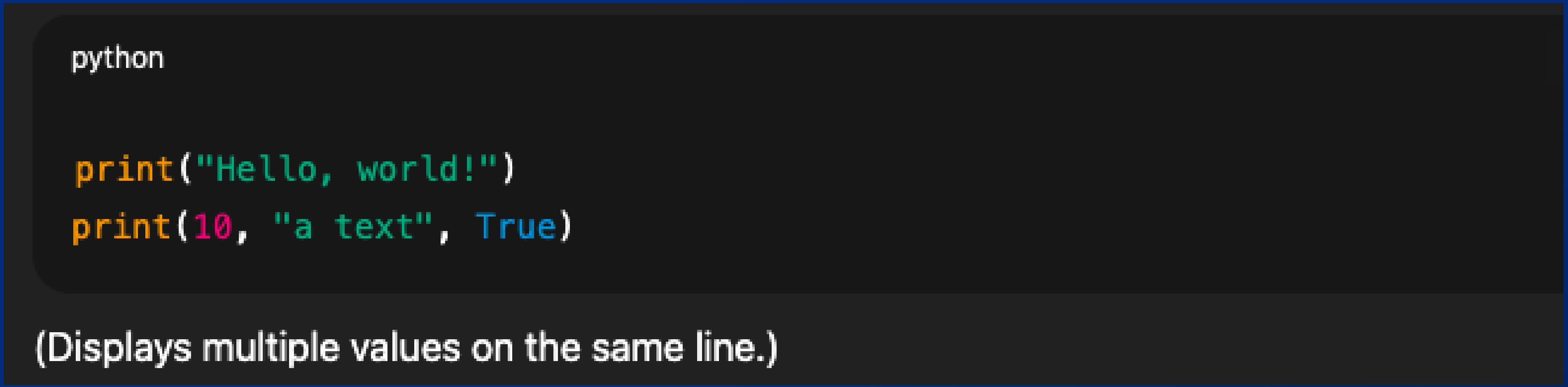
- **Objectives**
  - Understand the basic syntax of Python.
  - Get familiar with variables, data types, conditions, and loops.
  - Discover basic data structures (lists, tuples, dictionaries).
  - Use Jupyter Notebook to write and execute Python code.

Why Python?

- **Open-source language**: Python is popular in data science for its simplicity and readability.
- Used for **data processing, artificial intelligence, and web development**.
- Works in multiple environments: **Jupyter Notebook, VS Code, PyCharm, Spyder**, etc.
- One of the **most widely used languages** in industry.
- Offers a **large library ecosystem** for data manipulation, analysis, and visualization (e.g., *Pandas, NumPy, Matplotlib, Seaborn, Scikit-learn*).
- Backed by a **strong community** with abundant learning resources.
- Applied in **various domains** such as finance, healthcare, and marketing.

Python Basic Syntax

- **Comments:** Use # to add comments.
  Example: # This is a comment
- **Display text:** Use the print() function to display text.
  Example:



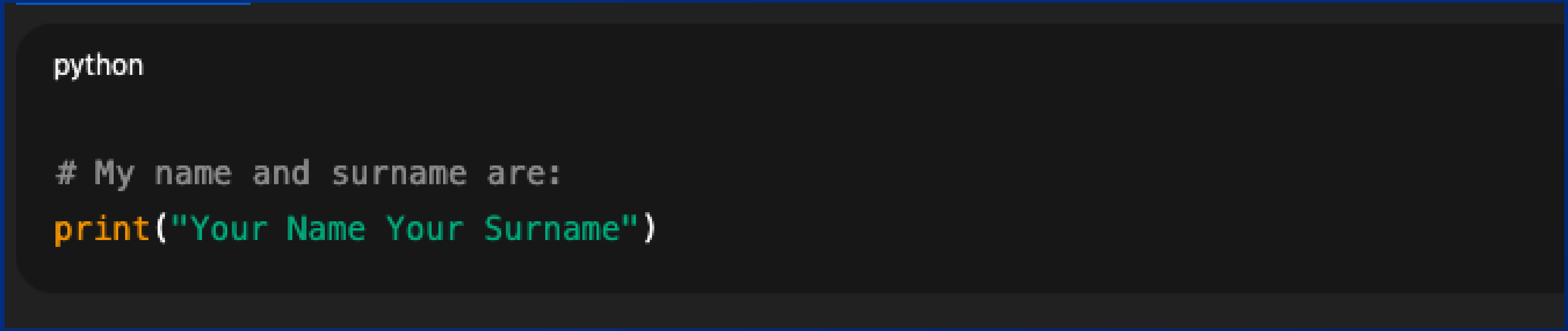
python

```python
print("Hello, world!")
print(10, "a text", True)
```

(Displays multiple values on the same line.)

Practical Exercise

- Display your name and surname and add a comment using Jupyter Notebook.
  Example task:



python

```python
# My name and surname are:
print("Your Name Your Surname")
```

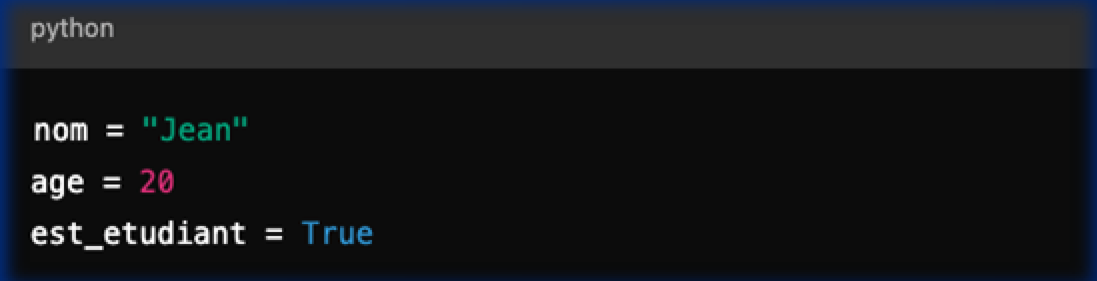
Variables and data types

- Variables: Store values (no need to declare type).
- Data types:
  - Numbers (integers, floats)
  - Strings
  - Booleans (True or False)

Exercise:

Declare variables for your name, age, and student status, then print them.

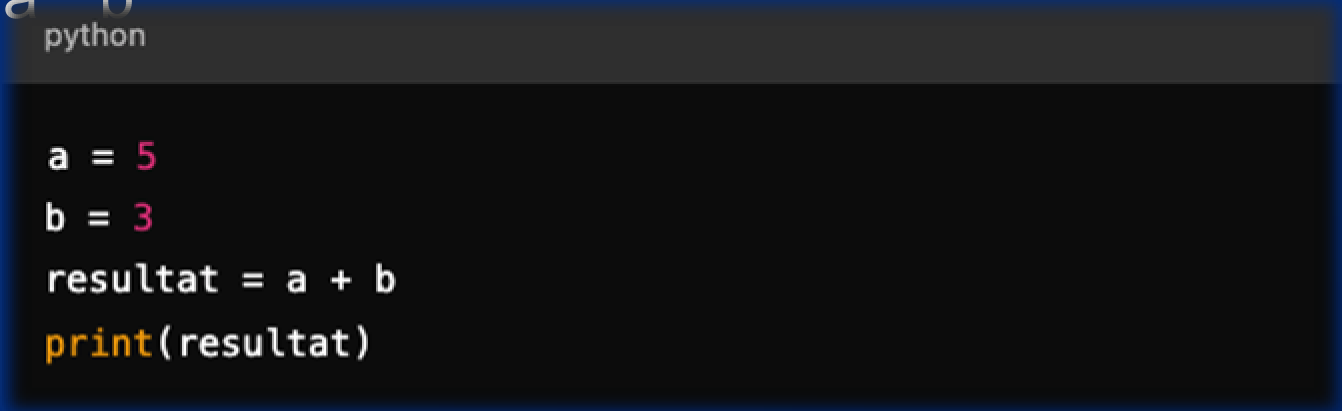
- Affichez ces variables avec la fonction print().
- Ajouter un titre niveau 1 pour cette cellule



```python
nom = "Jean"
age = 20
est_etudiant = True
```

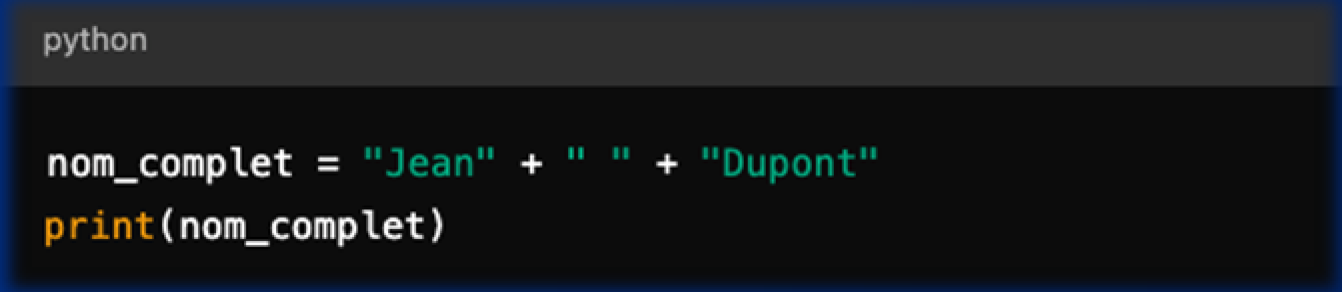
Operations on Variables

- Mathematical operations:
- Addition: a + b
- Multiplication: a * b



```
a = 5
b = 3
resultat = a + b
print(resultat)
```

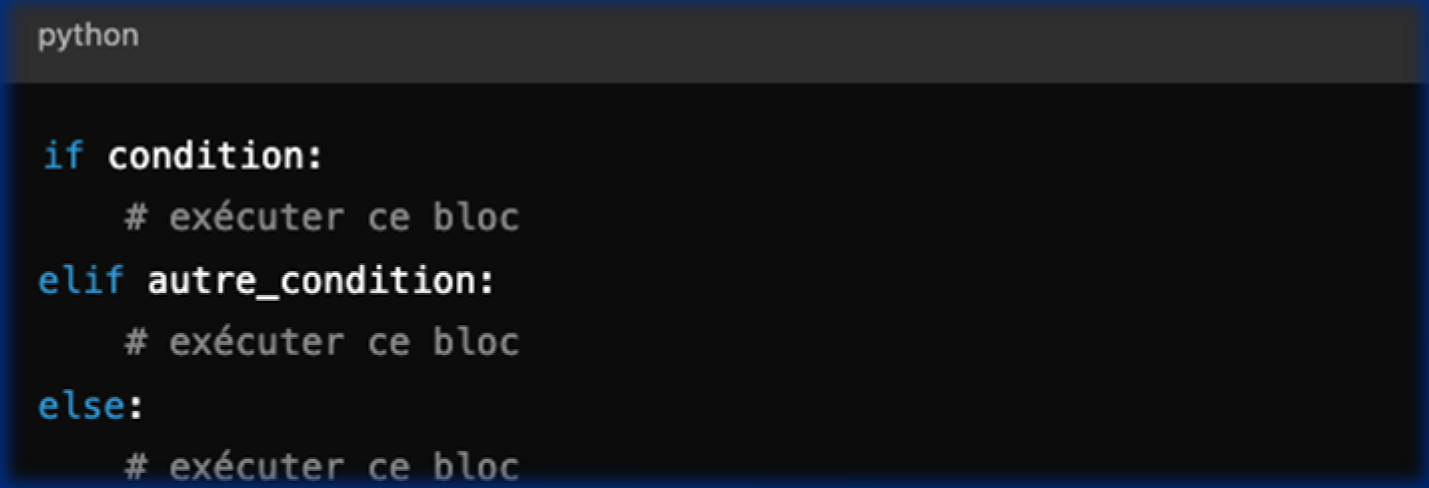
- String concatenation



```
nom_complet = "Jean" + " " + "Dupont"
print(nom_complet)
```

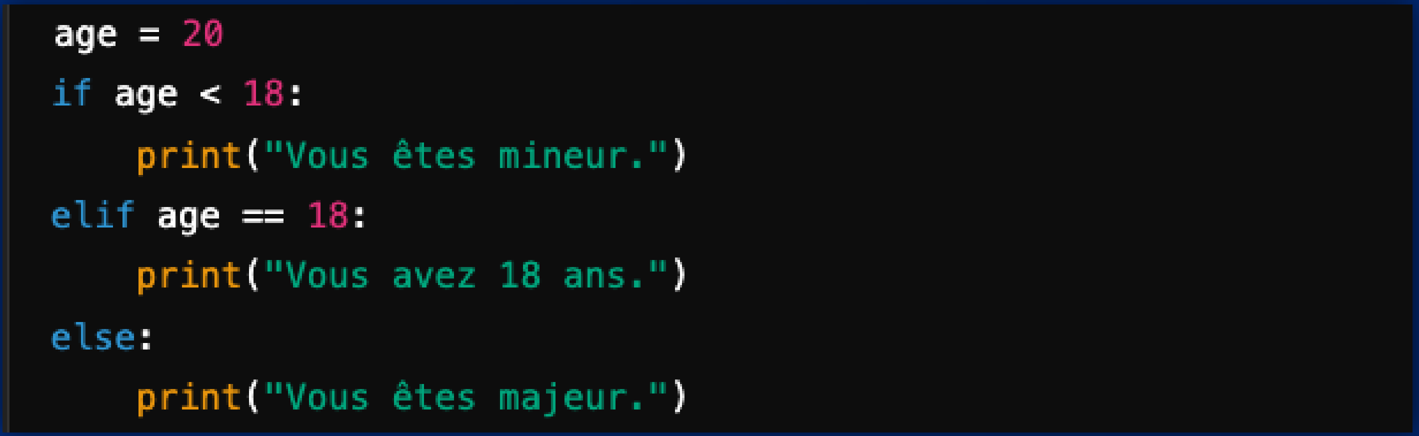
Conditions (if, elif, else)

- Used to test logical conditions.
  Example:



python

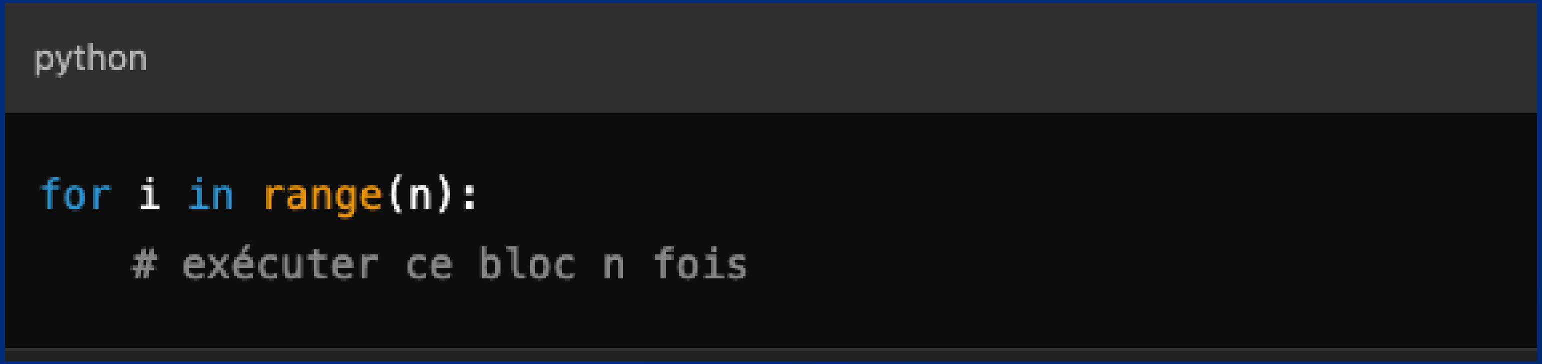
```python
if condition:
    # exécuter ce bloc
elif autre_condition:
    # exécuter ce bloc
else:
    # exécuter ce bloc
```



```python
age = 20
if age < 18:
    print("Vous êtes mineur.")
elif age == 18:
    print("Vous avez 18 ans.")
else:
    print("Vous êtes majeur.")
```

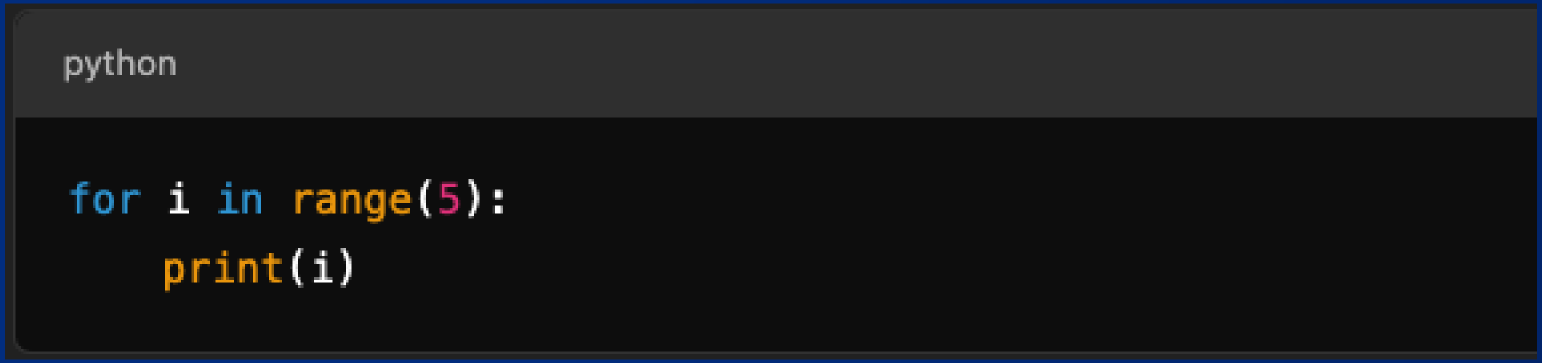
Loops

- For loop: repeats a block a defined number of times.
  While loop: repeats as long as a condition is true.



```python
for i in range(n):
    # exécuter ce bloc n fois
```

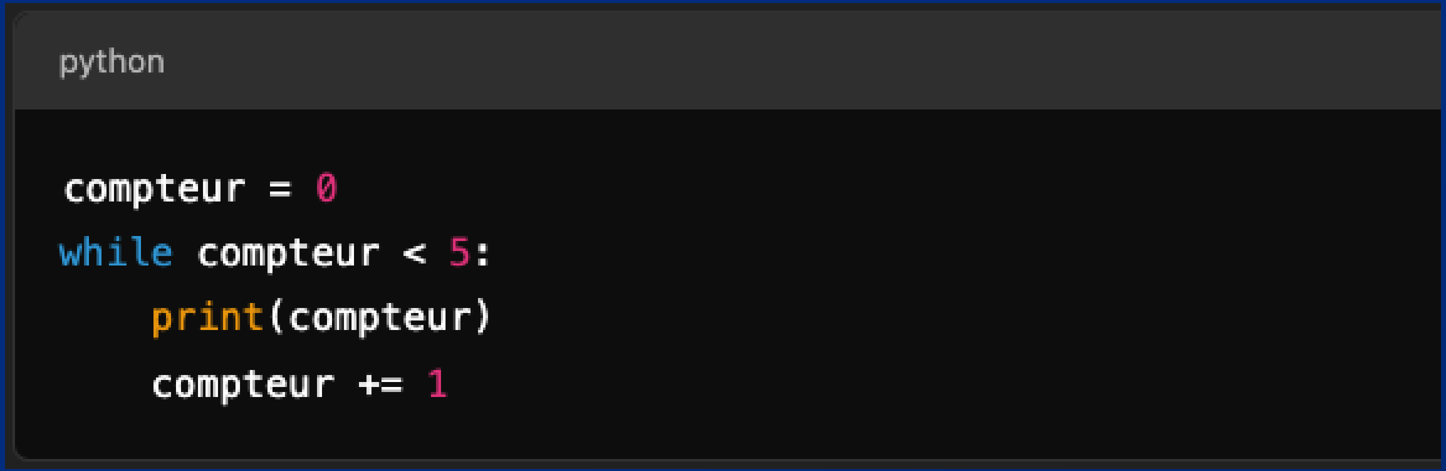
- Example



```python
for i in range(5):
    print(i)
```

While loop

- The while loop repeats a block of code as long as a condition is true. Syntax:



python

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

- Practical exercice:

  - Display even numbers.
    - Write a **for loop** that displays all even numbers between 1 and 10.
    - Write a **while loop** that displays the numbers from 0 to 5.

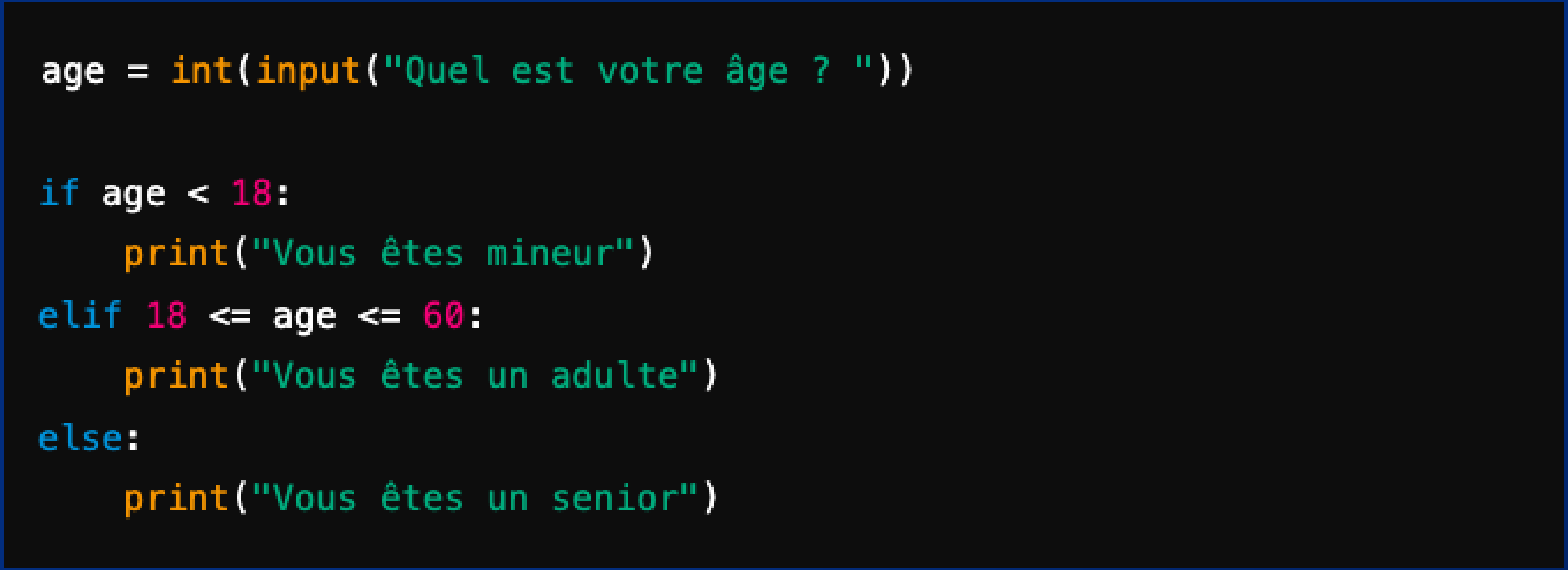
Exercice 01:

- **Age Verification Program**

Write a Python program that prompts the user for their age and displays a message based on the age they enter.

1. Ask the user to enter their age.
2. If the age is less than 18, display: **”Vous etes mineur."**
3. If the age is between 18 and 60, display: **”Vous etes un adulte."**
4. If the age is greater than 60, display: **”Vous etes un senior."**

Python code for exercice 01:



```python
age = int(input("Quel est votre âge ? "))

if age < 18:
    print("Vous êtes mineur")
elif 18 <= age <= 60:
    print("Vous êtes un adulte")
else:
    print("Vous êtes un senior")
```

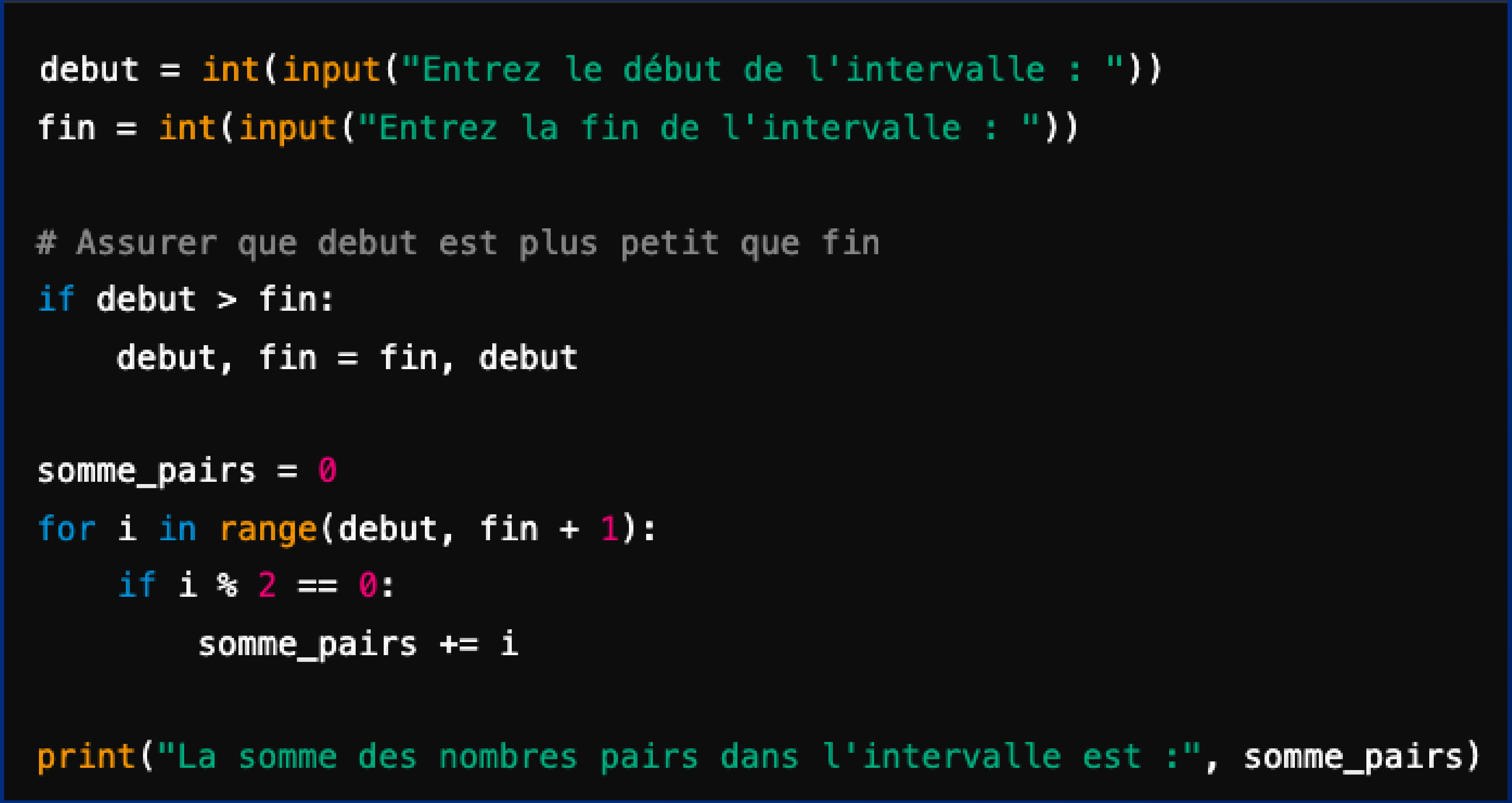
Exercice 02:

- **Use a loop to calculate the sum of even numbers within a range provided by the user.**

Instructions:

1. Ask the user to enter two numbers: a start and an end of the range.
2. Use a **for loop** to calculate and display the sum of all even numbers within that range.
3. Make sure the program works regardless of the order in which the numbers are entered.

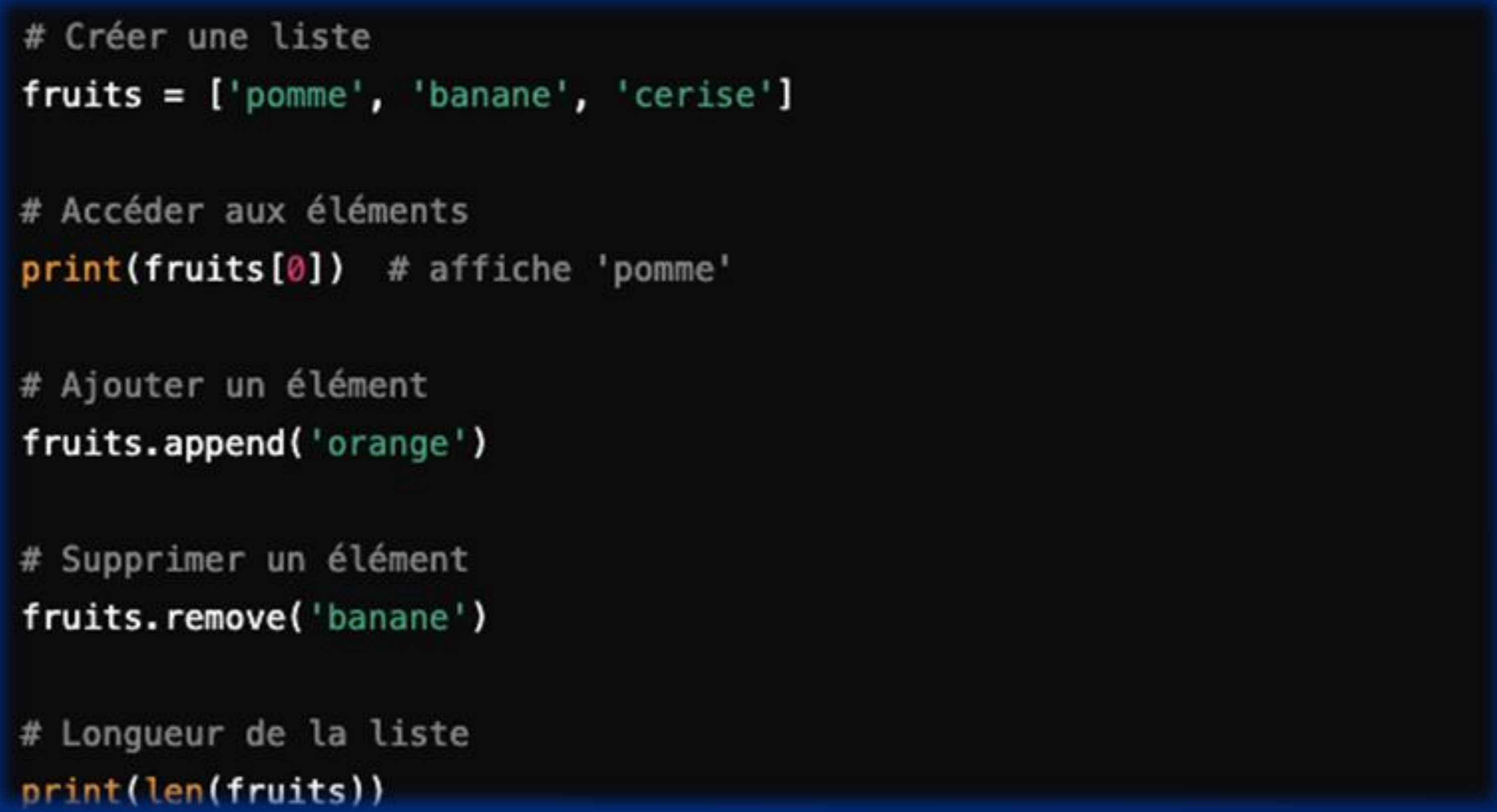
Code python for exercice 2



```python
debut = int(input("Entrez le début de l'intervalle : "))
fin = int(input("Entrez la fin de l'intervalle : "))

# Assurer que debut est plus petit que fin
if debut > fin:
    debut, fin = fin, debut

somme_pairs = 0
for i in range(debut, fin + 1):
    if i % 2 == 0:
        somme_pairs += i

print("La somme des nombres pairs dans l'intervalle est :", somme_pairs)
```

Data Structures in Python: LISTS

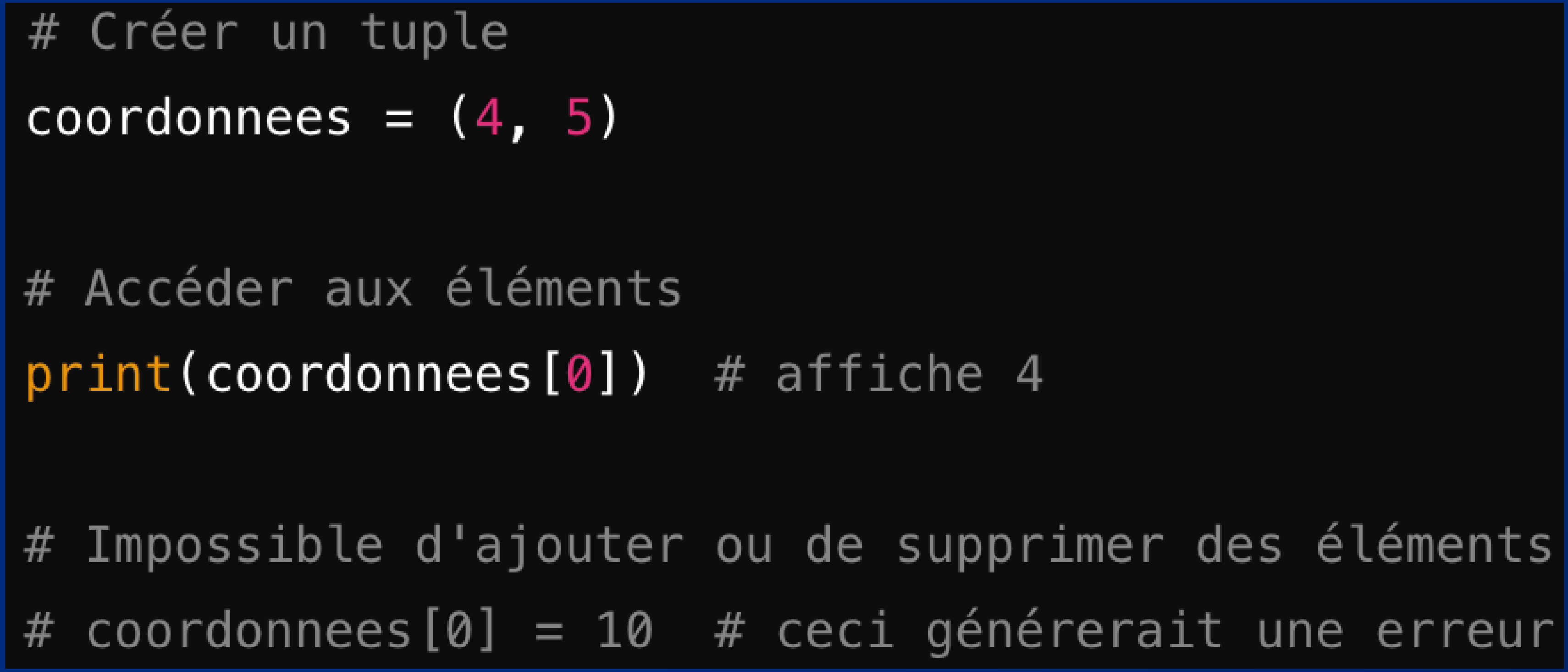
- A **list** is a data structure that allows you to store multiple values within a single object. It is **mutable**, meaning its elements can be modified after creation.



```
# Créer une liste
fruits = ['pomme', 'banane', 'cerise']

# Accéder aux éléments
print(fruits[0])  # affiche 'pomme'

# Ajouter un élément
fruits.append('orange')

# Supprimer un élément
fruits.remove('banane')

# Longueur de la liste
print(len(fruits))
```

Data Structures in Python: TUPLES

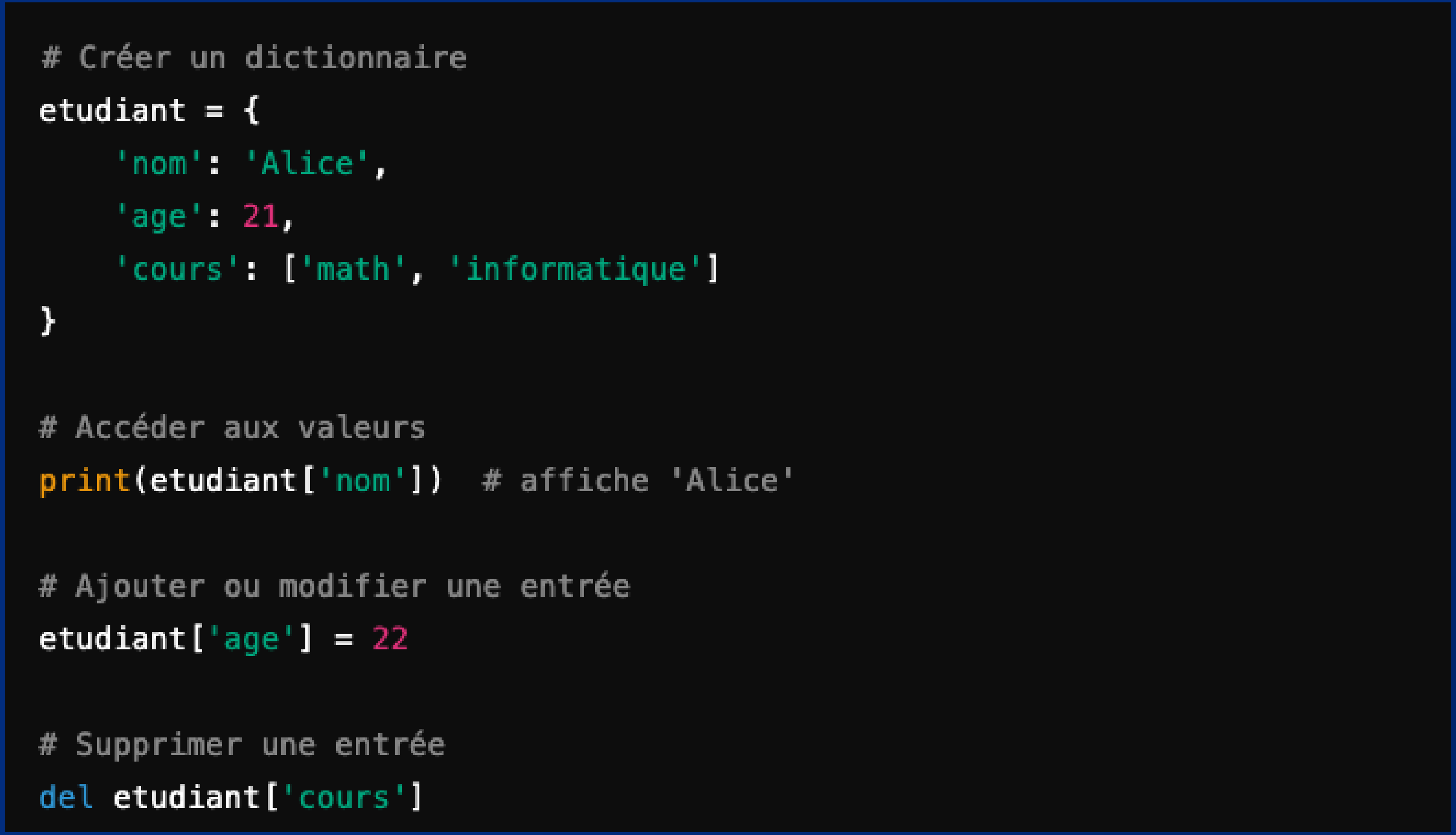
- Similar to a list, but it is **immutable**. This means that once created, the elements of a tuple cannot be modified.



```python
# Créer un tuple
coordonnees = (4, 5)

# Accéder aux éléments
print(coordonnees[0])  # affiche 4

# Impossible d'ajouter ou de supprimer des éléments
# coordonnees[0] = 10  # ceci générerait une erreur
```

Data Structures in Python: Dictionaries

- A **dictionary** is a collection of key-value pairs used to associate unique keys with specific values. It is very useful for storing structured data, such as a record or a database entry.



```python
# Créer un dictionnaire
etudiant = {
    'nom': 'Alice',
    'age': 21,
    'cours': ['math', 'informatique']
}

# Accéder aux valeurs
print(etudiant['nom'])  # affiche 'Alice'

# Ajouter ou modifier une entrée
etudiant['age'] = 22

# Supprimer une entrée
del etudiant['cours']
```

Exercice 03:

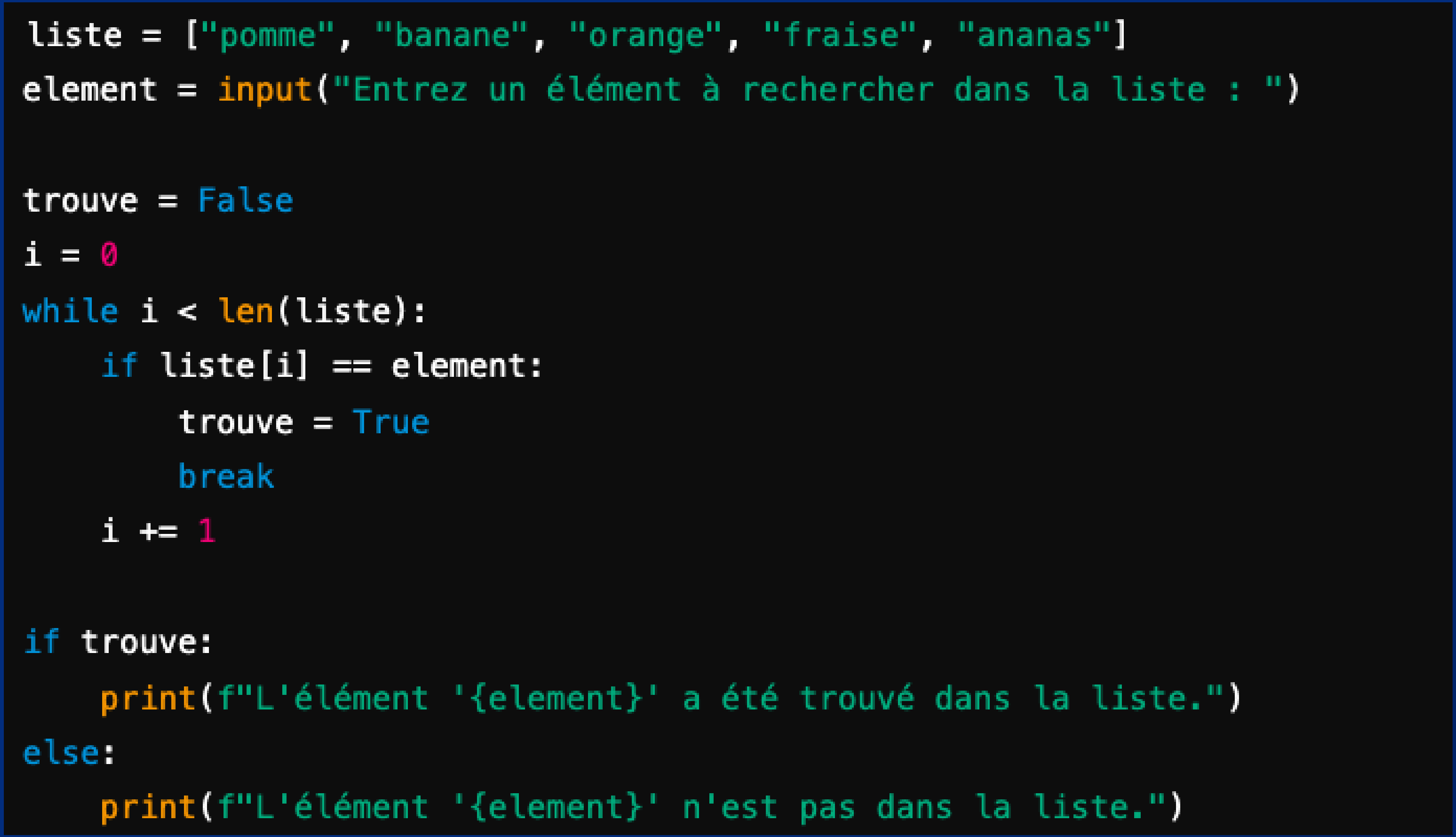
- **Searching in a List**

Use a **while loop** to search for an element in a list provided by the user.

- **Instructions:**

1. Create a list with at least 5 elements (numbers or strings).
2. Ask the user to enter an element to search for in the list.
3. Use a **while loop** to go through the list and display whether the element was found or not.

Python for Exercice 03:



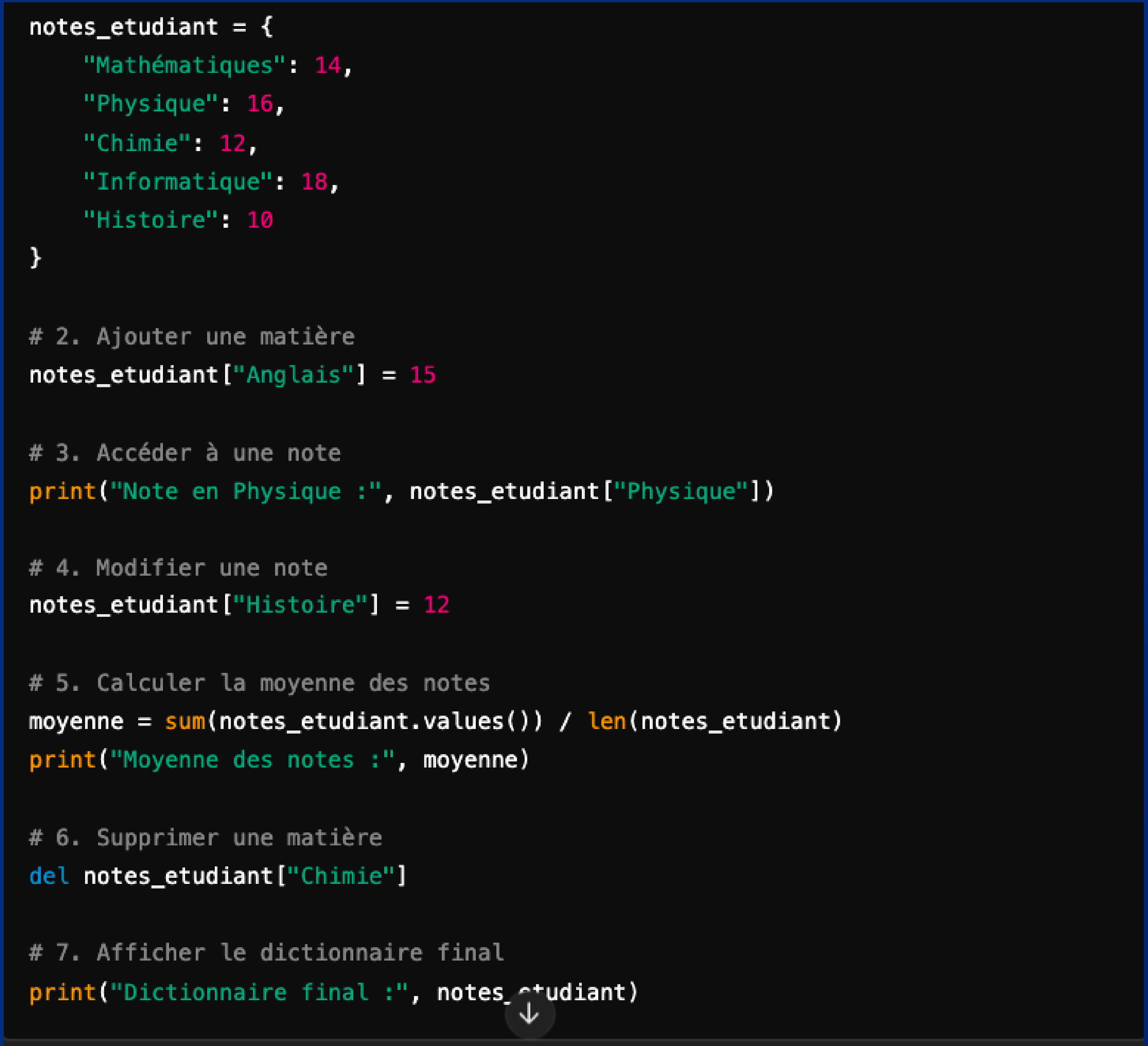
```python
liste = ["pomme", "banane", "orange", "fraise", "ananas"]
element = input("Entrez un élément à rechercher dans la liste : ")

trouve = False
i = 0
while i < len(liste):
    if liste[i] == element:
        trouve = True
        break
    i += 1

if trouve:
    print(f"L'élément '{element}' a été trouvé dans la liste.")
else:
    print(f"L'élément '{element}' n'est pas dans la liste.")
```

Exercice 04

1. Create a dictionary named **student_grades** with the following subjects as keys and their corresponding grades as values:

- Mathematics: 14
- Physics: 16
- Chemistry: 12
- Computer Science: 18
- History: 10

2. Add a new subject **English** with the grade 15 to the dictionary.
3. Write code to display the grade obtained in **Physics**.
4. Correct the grade for **History** and change it to 12.
5. Calculate and display the **average grade** of the student using the values in the dictionary.
6. Remove the subject **Chemistry** from the dictionary (assuming it is no longer considered for this year).
7. Display the final dictionary to verify all the modifications.



```python
notes_etudiant = {
    "Mathématiques": 14,
    "Physique": 16,
    "Chimie": 12,
    "Informatique": 18,
    "Histoire": 10
}

# 2. Ajouter une matière
notes_etudiant["Anglais"] = 15

# 3. Accéder à une note
print("Note en Physique :", notes_etudiant["Physique"])

# 4. Modifier une note
notes_etudiant["Histoire"] = 12

# 5. Calculer la moyenne des notes
moyenne = sum(notes_etudiant.values()) / len(notes_etudiant)
print("Moyenne des notes :", moyenne)

# 6. Supprimer une matière
del notes_etudiant["Chimie"]

# 7. Afficher le dictionnaire final
print("Dictionnaire final :", notes_etudiant)
```

Partie 3 : **Libraries for Data Manipulation**

pandas

NumPy

matplotlib

NumPy Library

- Created by **Travis Oliphant** in the early 2000s.
- Short for *Numerical Python*.
- Provides **multidimensional arrays**, **matrices**, and **mathematical functions** for efficient computation.
- Enables **high-performance** data manipulation, surpassing built-in Python structures for complex operations.

Pandas Library

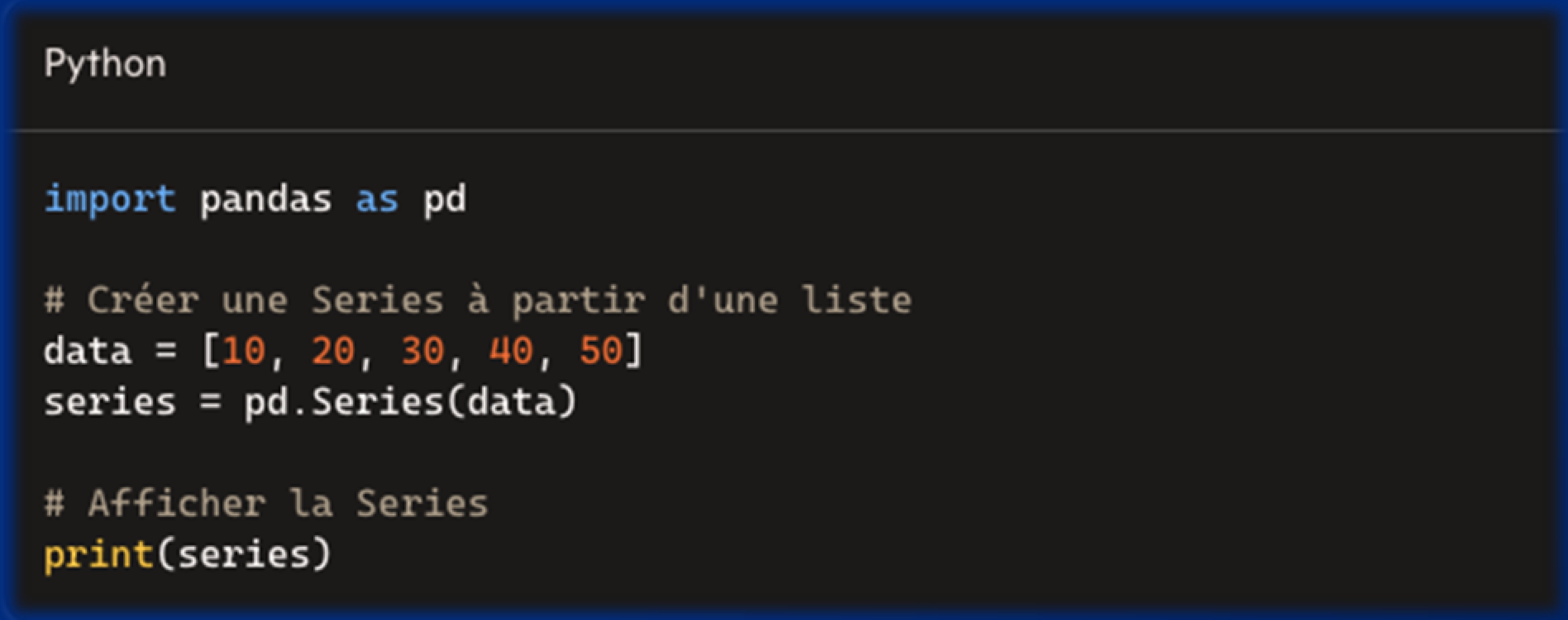
- Developed by **Wes McKinney** in 2008.
- Built on **NumPy**, offering flexible structures for handling structured data.
- Provides two main structures:
  - **Series** (1D labeled array)
  - **DataFrame** (2D labeled data structure)
- Highly efficient for large-scale data analysis.

Feature	NumPy	Pandas
Data type	Homogeneous	Heterogeneous
Dimensions	Multi-dimensional	Up to 2D (DataFrame)
Memory	Lower usage (faster)	Higher usage (slower)
Ease of use	Requires practice	Intuitive and user-friendly

Basic Pandas Usage

- Declare a **Series**
- Declare a **DataFrame**
- Import NumPy and Pandas with version display
- Create a DataFrame from a .csv file

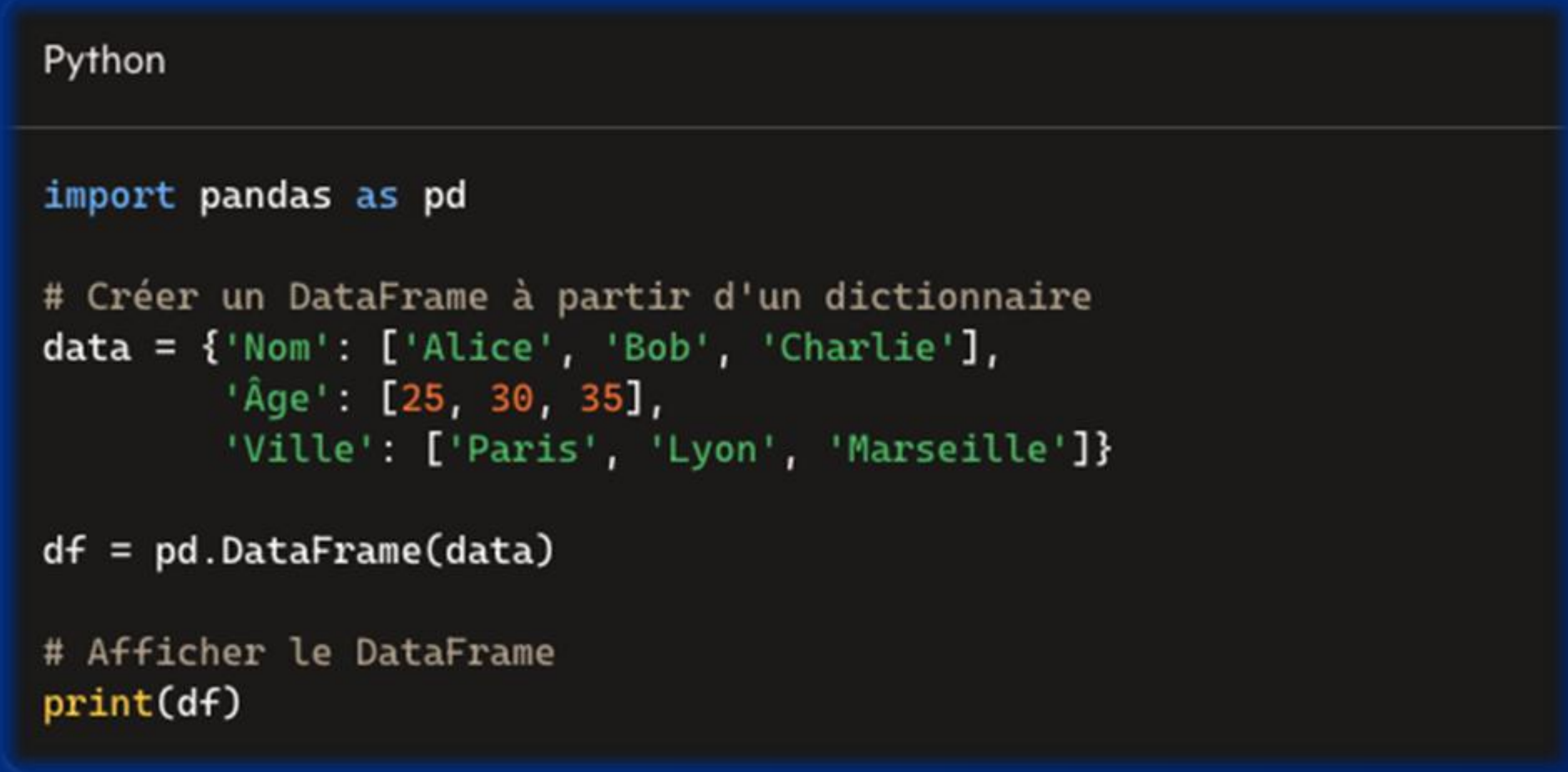
Declare a **Series with pandas**



Python

```python
import pandas as pd

# Créer une Series à partir d'une liste
data = [10, 20, 30, 40, 50]
series = pd.Series(data)

# Afficher la Series
print(series)
```

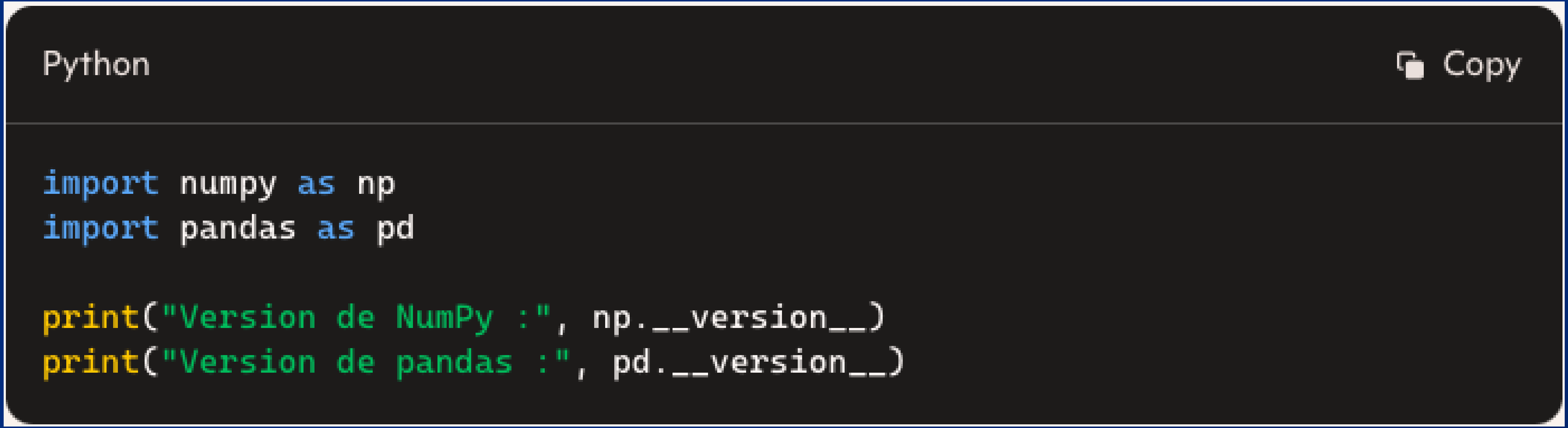
Declare Dataframe with pandas



Python

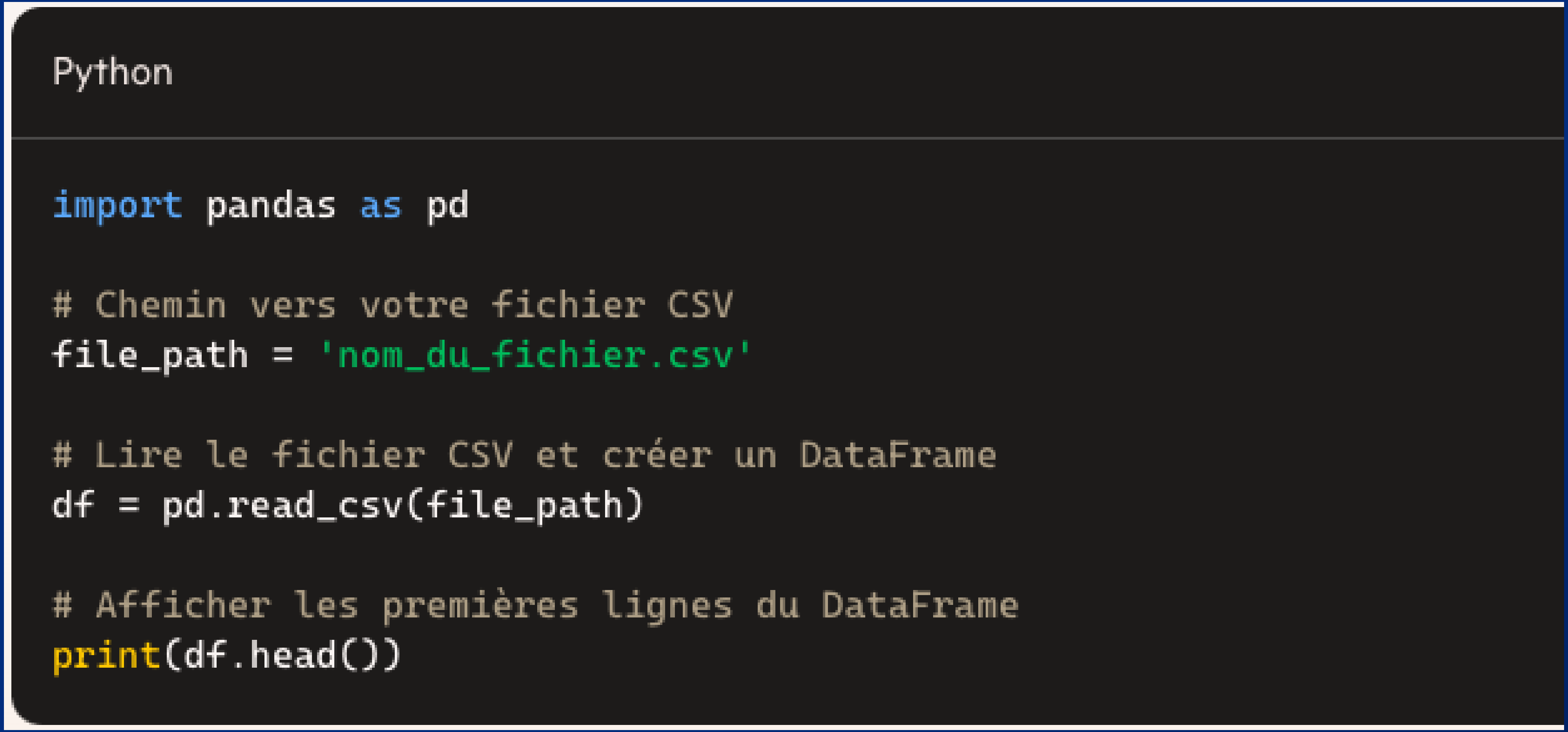
```python
import pandas as pd

# Créer un DataFrame à partir d'un dictionnaire
data = {'Nom': ['Alice', 'Bob', 'Charlie'],
        'Âge': [25, 30, 35],
        'Ville': ['Paris', 'Lyon', 'Marseille']}

df = pd.DataFrame(data)

# Afficher le DataFrame
print(df)
```

•Import NumPy and Pandas with version display



```python
import numpy as np
import pandas as pd

print("Version de NumPy :", np.__version__)
print("Version de pandas :", pd.__version__)
```

Create a DataFrame from a .csv file


Python

```
import pandas as pd

# Chemin vers votre fichier CSV
file_path = 'nom_du_fichier.csv'

# Lire le fichier CSV et créer un DataFrame
df = pd.read_csv(file_path)

# Afficher les premières lignes du DataFrame
print(df.head())
```