**🐍 Introduction to Python**

**🔹 What is Python?**

**Python** is a high-level, interpreted, general-purpose programming language. It is known for its **simple syntax**, **readability**, and **versatility**, making it ideal for beginners as well as professionals.

**🔹 Key Features of Python:**

| **Feature** | **Description** |
| --- | --- |
| **Easy to Learn** | Python has a clean and readable syntax. |
| **Interpreted Language** | No need to compile; code runs directly line by line. |
| **Dynamically Typed** | No need to declare variable types. |
| **High-Level Language** | Closer to human language, not machine language. |
| **Portable** | Python programs can run on different platforms with little or no change. |
| **Extensive Libraries** | Comes with powerful built-in and third-party libraries (NumPy, Pandas, etc.) |
| **Object-Oriented** | Supports classes, objects, and inheritance. |
| **Open Source** | Freely available to use and distribute. |

**🔹 What Can Python Be Used For?**

✅ Web Development – using frameworks like **Django**, **Flask**  
✅ Data Science & Analytics – with **Pandas**, **NumPy**, **Matplotlib**  
✅ Machine Learning – using **scikit-learn**, **TensorFlow**, **PyTorch**  
✅ Automation / Scripting – automate repetitive tasks  
✅ Game Development – with **Pygame**  
✅ App Development, Networking, Cybersecurity, and more!

**🔹 Python Versions**

* **Python 2.x** – Legacy version, no longer maintained
* **Python 3.x** – Current version (Python 3.12 as of 2024)

**🔹 Hello World Program in Python**

print("Hello, World!")

**🔹 How Python Code is Executed**

Python code is executed **line by line** by the **Python interpreter**.

**🔹 How to Run Python**

* **Online**: Use platforms like [Replit](https://replit.com/" \t "_new), Google Colab
* **Offline**: Install Python from [python.org](https://www.python.org/downloads/" \t "_new), then use:
  + **IDLE** (Python’s built-in editor)
  + **VS Code**, **PyCharm**, or other IDEs

**🔹 Python File Extension**

Python files have the .py extension.

Example:

python myscript.py

**DATA TYPES:**

In Python, **data types** specify the type of value a variable holds. Python is **dynamically typed**, which means you don’t need to declare the data type explicitly — it's inferred at runtime.

### 🔹 ****1. Numeric Types****

#### ✅ a. int – Integer

Whole numbers (positive or negative, no decimals).

x = 10

y = -50

#### ✅ b. float – Floating Point Number

Numbers with decimal points.

pi = 3.14

height = -5.6

#### ✅ c. complex – Complex Number

Used in scientific applications. Written as a + bj.

z = 3 + 4j

### 🔹 ****2. String (****str****)****

A sequence of characters enclosed in quotes.

name = "Alice"

message = 'Hello, Python!'

#### ✅ String Operations:

s = "hello"

print(s.upper()) # HELLO

print(s[1]) # e

print(s[0:3]) # hel

print(len(s)) # 5

### 🔹 ****3. Boolean (****bool****)****

Represents one of two values: True or False.  
Used mainly in conditions and comparisons.

is\_active = True

print(5 > 3) # True

### 🔹 ****4. Sequence Types****

#### ✅ a. list – Ordered, Mutable Collection

fruits = ["apple", "banana", "cherry"]

fruits.append("mango")

#### ✅ b. tuple – Ordered, Immutable Collection

colors = ("red", "green", "blue")

#### ✅ c. range – Sequence of numbers

for i in range(5):

print(i) # prints 0 to 4

### 🔹 ****5. Set Types****

#### ✅ a. set – Unordered, No duplicate elements

numbers = {1, 2, 3, 2}

print(numbers) # {1, 2, 3}

#### ✅ b. frozenset – Immutable set

fs = frozenset([1, 2, 3])

### 🔹 ****6. Mapping Type****

#### ✅ dict – Dictionary (Key-Value Pairs)

student = {"name": "Alice", "age": 21}

print(student["name"]) # Alice

### 🔹 ****7. None Type****

Represents a null or no value.

x = None

## 🔸 Summary Table:

| **Type** | **Example** | **Description** |
| --- | --- | --- |
| int | 5, -100 | Integer numbers |
| float | 3.14, -0.99 | Decimal numbers |
| complex | 2+3j | Complex numbers |
| bool | True, False | Boolean values |
| str | "hello" | Text data |
|  |  |  |
| list | [1, 2, 3] | Mutable ordered collection |
| tuple | (1, 2, 3) | Immutable ordered collection |
| dict | {"a": 1} | Key-value pairs |
| set | {1, 2, 3} | Unique unordered collection |
| None | None | Absence of a value |

## Python Programming Statements

In Python, a **statement** is an instruction that the Python interpreter can execute. Python statements control how your program behaves and executes tasks.

### 🔹 1. ****Assignment Statement****

Used to assign values to variables.

age = 25

name = "John"

🧠 **Real-time Example:**

In a student registration system:

student\_name = "Aarav"

student\_id = 1023

### 🔹 2. ****Input and Output Statements****

input() is used to take input from the user.

print() is used to display output.

name = input("Enter your name: ")print("Welcome", name)

🧠 **Real-time Example:**

In an ATM interface:

pin = input("Enter your PIN: ")print("Processing your request...")

### 🔹 3. ****Conditional Statements****

Used to perform actions based on conditions.

age = int(input("Enter your age: "))if age >= 18:

print("You are eligible to vote.")else:

print("You are not eligible to vote.")

🧠 **Real-time Example:**

In an online shopping app:

cart\_value = 700if cart\_value >= 500:

print("You get free delivery!")else:

print("Delivery charge will be applied.")

### 🔹 4. ****Looping Statements****

**loop** is a programming construct used to repeat a block of code multiple times. It is especially useful when you want to perform an action several times without writing the same code again and again.Used to repeat a block of code.

#### for loop:

Used when you want to iterate over a **sequence** (like a list, tuple, string, or range).

for i in range(1, 6):

print(i)

🧠 **Real-time Example:**

Displaying product IDs:

product\_ids = [101, 102, 103]for pid in product\_ids:

print("Product ID:", pid)

#### while loop:

Used when you want to execute a block of code **as long as a condition is true**.

count = 1while count <= 3:

print("Try number", count)

count += 1

🧠 **Real-time Example:**

Retry login attempts:

attempt = 1while attempt <= 3:

print("Login attempt", attempt)

attempt += 1

### 🔹 5. ****Jump Statements****

Used to control loop flow.

#### a. break

for i in range(1, 6):

if i == 3:

break

print(i)

🧠 **Real-time Example:**

Stop processing when stock is not available:

items = ["Shoes", "Shirt", "Out of Stock", "Watch"]for item in items:

if item == "Out of Stock":

break

print("Processing:", item)

#### b. continue

for i in range(1, 6):

if i == 3:

continue

print(i)

🧠 **Real-time Example:**

Skip unavailable items:

items = ["Shoes", "Unavailable", "Watch"]for item in items:

if item == "Unavailable":

continue

print("Shipping:", item)

### 🔹 6. ****Function Call Statement****

Used to reuse code blocks.

def greet():

print("Hello!")

greet()

🧠 **Real-time Example:**

Call login and logout functions in an app:

def login():

print("User logged in")

def logout():

print("User logged out")

login()

logout()

### Summary Table

| **Statement Type** | **Syntax Example** | **Real-Time Use Case** |
| --- | --- | --- |
| Assignment | amount = 1000 | Wallet balance, salary |
| Input/Output | input(), print() | ATM PIN, welcome message |
| Conditional | if, elif, else | Offer eligibility, age check |
| Looping | for, while | Product listings, login attempts |
| Jump | break, continue | Exit on error, skip unavailable items |
| Function Call | greet() | Login/logout, reusable code blocks |

**🟡 1. VARIABLES in Python**

**🔸 What is a Variable?**

A **variable** is a container used to store data values. In Python, you don't need to declare the data type explicitly — it is inferred automatically.

**🔸 Variable Naming Rules:**

* Can contain letters, digits, and underscores (\_)
* Must start with a letter or underscore
* Cannot start with a digit
* Cannot be a Python keyword (like if, while, class, etc.)

**🔸 Example:**

name = "Alice" # String

age = 25 # Integer

height = 5.7 # Float

is\_student = True # Boolean

**🔸 Reassigning Variables:**

x = 10

x = "Now I am a string!" # x can change type in Python

**🟢 2. OPERATORS in Python**

Operators are special symbols used to perform operations on variables and values.

**🔹 A. Arithmetic Operators**

| **Operator** | **Description** | **Example** | **Output** |
| --- | --- | --- | --- |
| + | Addition | 5 + 3 | 8 |
| - | Subtraction | 5 - 3 | 2 |
| \* | Multiplication | 5 \* 3 | 15 |
| / | Division (float) | 5 / 2 | 2.5 |
| // | Floor Division | 5 // 2 | 2 |
| % | Modulus (remainder) | 5 % 2 | 1 |
| \*\* | Exponentiation | 2 \*\* 3 | 8 |

**✅ Example:**

a = 10

b = 3

print(a + b) # 13

print(a % b) # 1

print(a \*\* b) # 1000

**🔹 B. Assignment Operators**

| **Operator** | **Example** | **Equivalent to** |
| --- | --- | --- |
| = | a = 5 | Assign value |
| += | a += 2 | a = a + 2 |
| -= | a -= 2 | a = a - 2 |
| \*= | a \*= 2 | a = a \* 2 |
| /= | a /= 2 | a = a / 2 |

**✅ Example:**

x = 5

x += 3 # x = x + 3 => 8

**🔹 C. Comparison Operators**

Used to compare two values. Returns True or False.

| **Operator** | **Description** | **Example** |
| --- | --- | --- |
| == | Equal to | 5 == 5 → True |
| != | Not equal to | 5 != 3 → True |
| > | Greater than | 5 > 3 → True |
| < | Less than | 5 < 3 → False |
| >= | Greater or equal | 5 >= 5 → True |
| <= | Less or equal | 3 <= 5 → True |

**🔹 D. Logical Operators**

Used to combine conditional statements.

| **Operator** | **Description** | **Example** |
| --- | --- | --- |
| and | True if both are True | True and True → True |
| or | True if at least one is True | True or False → True |
| not | Reverses result | not True → False |

**✅ Example:**

a = 5

b = 10

print(a < 10 and b > 5) # True

print(not(a == b)) # True

**🔹 E. Identity Operators**

| **Operator** | **Description** |
| --- | --- |
| is | True if two variables point to the same object |
| is not | True if they point to different objects |

x = [1, 2]

y = x

z = [1, 2]

print(x is y) # True

print(x is z) # False

**🔹 F. Membership Operators**

| **Operator** | **Description** |
| --- | --- |
| in | True if value is found |
| not in | True if value is **not** found |

fruits = ["apple", "banana"]

print("apple" in fruits) # True

print("mango" not in fruits) # True

## 🧠 What is a Function?

A **function** is a **block of reusable code** that performs a specific task.

You **define** a function once and **call** it whenever you need it.

## 🔧 Why Use Functions?

Avoid repetition

Make code modular and easier to debug

Increase reusability

Improve readability

## 📌 Function Syntax in Python:

def function\_name(parameters):

# code block

return value

## 1️⃣ Example: Basic Function Without Parameters

def greet():

print("Hello! Welcome to the session.")

greet() # Calling the function

## 2️⃣ Function With Parameters

def greet\_user(name):

print(f"Hello, {name}! How can I help you?")

greet\_user("Shamsheera")

## 3️⃣ Function With Return Value

def add(a, b):

return a + b

result = add(10, 20)print("Sum is:", result)

## 4️⃣ Default Parameters

def greet(name="Student"):

print(f"Hello, {name}!")

greet() # Hello, Student!

greet("Ayesha") # Hello, Ayesha!

## 5️⃣ Keyword vs Positional Arguments

def profile(name, age):

print(f"{name} is {age} years old")

profile("Rahul", 25) # Positional

profile(age=30, name="Sana") # Keyword

## 6️⃣ Arbitrary Arguments (\*args, \*\*kwargs)

### ✅ \*args = multiple positional arguments (as tuple)

def total\_marks(\*marks):

return sum(marks)

print(total\_marks(85, 90, 88)) # 263

### ✅ \*\*kwargs = multiple keyword arguments (as dictionary)

### To pass Multople argument as Dictionary

def student\_details(\*\*details):

for key, value in details.items():

print(f"{key}: {value}")

student\_details(name="Aisha", age=20, grade='A')

## 7️⃣ Real-Time Examples

### 📌 A. Billing System – Function to calculate total with tax

def calculate\_total(bill\_amount, tax=0.05):

return bill\_amount + (bill\_amount \* tax)

print("Total Amount:", calculate\_total(1000))

### 📌 B. Attendance System – Mark attendance

def mark\_attendance(name):

print(f"{name} marked present at 9:00 AM.")

mark\_attendance("Abdul")

### 📌 C. Student Grade Calculator

def calculate\_grade(marks):

avg = sum(marks) / len(marks)

if avg >= 90:

return "A"

elif avg >= 75:

return "B"

elif avg >= 50:

return "C"

else:

return "Fail"

print("Grade:", calculate\_grade([80, 70, 85]))

## 8️⃣ Recursion (Function calling itself)

### ✅ Real-Time Example: Factorial

def factorial(n):

if n == 1:

return 1

return n \* factorial(n - 1)

print("Factorial:", factorial(5)) # 120

## 🔟 Lambda Function (Anonymous Function)

Useful for one-line operations, especially in sorting, filtering.

square = lambda x: x \* xprint(square(5)) # 25

## 🧵 What is a String in Python?

## ****strings**** are sequences of characters, and they come with many ****built-in methods**** that help with manipulation and analysis.

## String are immutable

A **string** is a sequence of **characters** enclosed in **single (**'**)**, **double (**"**)**, or **triple quotes (**''' **or** """**)**.

name = "Shamsheera"

college = 'St. Francis College'

msg = """This is a multiline string"""

## 🔑 Key Properties of Strings:

| **Property** | **Description** |
| --- | --- |
| Immutable | Strings **cannot be changed** |
| Ordered | Characters have **positions** |
| Iterable | Can loop through it |
| Indexed | Access by index (0, 1, 2...)  Access by -ve index(-1,-2…-n) |

## 📌 String Creation

s1 = 'Hello'

s2 = "World"

s3 = '''This is

a multiline

string.'''

## 📍 Accessing Characters (Indexing)

text = "Python"print(text[0]) # Pprint(text[-1]) # n

## 🔁 Traversing a String (Looping)

for ch in "Hello":

print(ch)

## ✂️ String Slicing

text = "DataScience"print(text[0:4])

# Dataprint(text[:4])

# Dataprint(text[4:])

# Scienceprint(text[::-1]) # Reverse the string

## 🔤 String Functions & Methods

### 1. upper()

Converts all characters to uppercase.

text = "hello"print(text.upper())

**Output:**

HELLO

### 2. lower()

Converts all characters to lowercase.

text = "HELLO"print(text.lower())

**Output:**

hello

### 3. title()

Converts the first character of each word to uppercase.

text = "python programming"print(text.title())

**Output:**

Python Programming

### 4. capitalize()

Capitalizes the first character of the string.

text = "hello world"print(text.capitalize())

**Output:**

Hello world

### 5. strip()

Removes leading and trailing whitespaces.

text = " hello "print(text.strip())

**Output:**

hello

### 6. replace(old, new)

Replaces a substring with another.

text = "I like Python"print(text.replace("Python", "Java"))

**Output:**

I like Java

### 7. split(separator)

Splits a string into a list.

text = "apple,banana,orange"print(text.split(","))

**Output:**

['apple', 'banana', 'orange']

### 8. join()

Joins elements of a list into a single string.

fruits = ['apple', 'banana', 'orange']print(", ".join(fruits))

**Output:**

apple, banana, orange

### 9. startswith() ****/**** endswith()

Checks if a string starts or ends with a specific substring.

text = "hello world"print(text.startswith("hello")) # Trueprint(text.endswith("world")) # True

### 10. find()

Returns the index of the first occurrence of a substring. Returns -1 if not found.

text = "hello world"print(text.find("world"))

**Output:**

6

### 11. count()

Counts how many times a substring appears.

text = "apple apple banana"print(text.count("apple"))

**Output:**

2

### ✅ Real-time Example:

Check if a sentence contains the word "Python" (case insensitive):

sentence = "I am learning python"if "python" in sentence.lower():

print("Yes, Python is mentioned.")

**Output:**

Yes, Python is mentioned.

| **Method** | **Description** | **Example** |
| --- | --- | --- |
| len() | Get length of string | len("Hello") → 5 |
| .upper() | Converts to uppercase | "hello".upper() → HELLO |
| .lower() | Converts to lowercase | "HELLO".lower() → hello |
| .title() | Title case | "my name is".title() → My Name Is |
| .strip() | Removes spaces | " Hello ".strip() → Hello |
| .replace() | Replaces part of string | "data".replace("a", "o") → doto |
| .find() | Finds index of first match | "python".find("t") → 2 |
| .count() | Counts occurrences | "banana".count("a") → 3 |
| .split() | Splits string into list | "a,b,c".split(",") → ['a','b','c'] |
| .join() | Joins list into string | " ".join(['Hi','there']) → Hi there |
| .isdigit() | Checks if all chars are digits | "123".isdigit() → True |
| .isalpha() | Checks if all chars are letters | "abc".isalpha() → True |

## 🔁 Real-Time Examples

### ✅ 1. Validate Mobile Number

def validate\_mobile(mobile):

if mobile.isdigit() and len(mobile) == 10:

print("Valid Mobile Number")

else:

print("Invalid Mobile Number")

validate\_mobile("9876543210")

### ✅ 2. Count Vowels in a Name

def count\_vowels(name):

vowels = "aeiouAEIOU"

count = 0

for ch in name:

if ch in vowels:

count += 1

print(f"Vowels in '{name}': {count}")

count\_vowels("Shamsheera")

### ✅ 3. Email Validator (Basic)

def is\_valid\_email(email):

return "@" in email and "." in email

print(is\_valid\_email("abc@gmail.com")) # True

### ✅ 4. Reverse a String

name = "Python"print(name[::-1]) # nohtyP

### ✅ 5. Capitalize First Letter of Every Word

sentence = "python is fun"print(sentence.title()) # Python Is Fun

## 🧠 Interview Tips

Strings are **immutable** – s[0] = 'X' will cause an error

String slicing does **not include end index** – s[1:4] returns index 1 to 3.

Use .split() to break strings and .join() to combine them.

Use in keyword to check for substring: "apple" in "pineapple" → True

## ✅ Practice Questions

Write a program to check if a string is a palindrome.

Count number of words in a sentence.

Replace all spaces with - in a given sentence.

Check if a password has at least 8 characters, one uppercase, and one number.

Extract domain from an email (user@example.com → example.com)

## 🧾 What is a List in Python?

A **list** is a **collection of ordered, mutable (changeable), and heterogeneous** elements.  
You can store **integers, strings, floats, even other lists or objects** in a list.

### ✅ Syntax:

my\_list = [10, 20, 30]

## 📌 Key Features of Lists

| **Feature** | **Description** |
| --- | --- |
| Ordered | Maintains the insertion order |
| Mutable | You can change, add, or remove elements |
| Indexed | Access items by position (starting at 0) |
| Duplicates | Allows duplicate elements |
| Mixed types | Can store different data types |

## 🎯 List Creation

empty\_list = []

numbers = [1, 2, 3, 4]

names = ["Aisha", "Rahul", "Neha"]

mixed = [10, "hello", 5.5, True]

## 📥 Accessing List Elements (Indexing & Slicing)

data = [10, 20, 30, 40, 50]print(data[0]) # 10print(data[-1]) # 50print(data[1:4]) # [20, 30, 40]

## 🔁 Looping Through a List

fruits = ["apple", "banana", "cherry"]for fruit in fruits:

print(fruit)

## 🧰 Common List Operations

| **Operation** | **Example** | **Output** |
| --- | --- | --- |
| Append (add at end) | lst.append(5) | Adds 5 to end |
| Insert (at position) | lst.insert(2, "Python") | Inserts at index 2 |
| Remove (by value) | lst.remove("apple") | Removes first occurrence |
| Pop (by index) | lst.pop(1) | Removes item at index 1 |
| Index | lst.index("banana") | Returns index of "banana" |
| Count | lst.count(10) | Counts how many 10s |
| Sort | lst.sort() | Sorts the list |
| Reverse | lst.reverse() | Reverses the list |
| Length | len(lst) | Number of elements |

lst = [10, 20, 30, "apple", "banana", 10]

print("Original List:", lst)

Output:

Original List: [10, 20, 30, 'apple', 'banana', 10]

1. Append (add at end)

lst.append(5)

print("After append(5):", lst)

Output:

After append(5): [10, 20, 30, 'apple', 'banana', 10, 5]

2. Insert (at position)

lst.insert(2, "Python")

print("After insert(2, 'Python'):", lst)

Output:

After insert(2, 'Python'): [10, 20, 'Python', 30, 'apple', 'banana', 10, 5]

3. Remove (by value)

lst.remove("apple")

print("After remove('apple'):", lst)

Output:

After remove('apple'): [10, 20, 'Python', 30, 'banana', 10, 5]

4. Pop (by index)

removed\_item = lst.pop(1)

print("After pop(1):", lst)

print("Removed item:", removed\_item)

Output:After pop(1): [10, 'Python', 30, 'banana', 10, 5]

Removed item: 20

5. Index (find position of element)

position = lst.index("banana")

print("Index of 'banana':", position)

Output:

Index of 'banana': 3

6. Count (number of occurrences)

count\_10 = lst.count(10)

print("Count of 10:", count\_10)

Output:

Count of 10: 2

7. Sort (ascending order)

⚠️ Note: You can only sort if the list contains the same data type (e.g., all integers).

num\_list = [3, 1, 4, 2]

num\_list.sort()

print("Sorted num\_list:", num\_list)

Sorted num\_list: [1, 2, 3, 4]

8. Reverse (reverses the order)

num\_list.reverse()

print("Reversed num\_list:", num\_list)

Reversed num\_list: [4, 3, 2, 1]

9. Length (number of elements)

print("Length of lst:", len(lst))

Output:

Length of lst: 6

## ✅ 1. ****Concatenation (****+****)****

Joins two lists into one.

list1 = [1, 2, 3]

list2 = [4, 5]

result = list1 + list2print(result)

**Output:**

[1, 2, 3, 4, 5]

## ✅ 2. ****Repetition (****\*****)****

Repeats the list elements.

list1 = ["A", "B"]

result = list1 \* 3print(result)

**Output:**

['A', 'B', 'A', 'B', 'A', 'B']

## ✅ 3. ****Membership (****in****,**** not in****)****

Checks whether an element is in the list.

fruits = ['apple', 'banana', 'cherry']print('banana' in fruits) # Trueprint('mango' not in fruits) # True

**Output:**

TrueTrue

## ✅ 4. ****Indexing (****[]****)****

Accesses an element by its index.

my\_list = [10, 20, 30, 40]print(my\_list[2]) # 30

## ✅ 5. ****Slicing (****[:]****)****

Retrieves a sublist from the list.

numbers = [1, 2, 3, 4, 5, 6]print(numbers[1:4]) # Elements at index 1, 2, 3

**Output:**

[2, 3, 4]

## ✅ 6. ****Comparison Operators (****==****,**** !=****,**** <****,**** >****)****

Compare lists element-wise.

a = [1, 2, 3]

b = [1, 2, 3]

c = [1, 2, 4]print(a == b) # Trueprint(a != c) # Trueprint(a < c) # True (compares element-wise)

## ✅ 7. ****Length (****len()****)****

Returns the number of elements in the list.

my\_list = ['a', 'b', 'c']print(len(my\_list))

**Output:**

3

## ✅ 8. ****List Comprehension (special usage)****

Uses an expression to create a new list from an existing one.

nums = [1, 2, 3, 4]

squares = [x\*x for x in nums]print(squares)

**Output:**

[1, 4, 9, 16]

## 🧪 Real-Time Examples

### ✅ 1. ****Student Attendance Tracker****

present\_students = ["Rahul", "Neha", "Aisha"]

present\_students.append("Vikram")print(present\_students)

### ✅ 2. ****Online Shopping Cart****

cart = []

cart.append("Shirt")

cart.append("Jeans")

cart.remove("Shirt")print("Current Cart:", cart)

### ✅ 3. ****Classroom Marks Analyzer****

marks = [85, 90, 76, 88, 92]print("Highest:", max(marks))print("Lowest:", min(marks))print("Average:", sum(marks) / len(marks))

### ✅ 4. ****Reverse a List of Names****

names = ["Zara", "Abdul", "Kiran"]

names.reverse()print(names) # ['Kiran', 'Abdul', 'Zara']

### ✅ 5. ****Check if a Student is Present****

students = ["Amit", "Sara", "John"]

name = "Sara"if name in students:

print(f"{name} is present.")else:

print(f"{name} is absent.")

## 🧠 List Comprehension (Short Way to Create Lists)

squares = [x\*\*2 for x in range(1, 6)]print(squares) # [1, 4, 9, 16, 25]

## 🔗 Nested List (List inside List)

students = [["Amit", 85], ["Sara", 92], ["John", 78]]for student in students:

print(f"{student[0]} scored {student[1]}")

## 📝 Summary Table

| **Task** | **Syntax or Example** |
| --- | --- |
| Create list | lst = [1, 2, 3] |
| Add item | lst.append(4) |
| Insert item | lst.insert(1, 10) |
| Delete by value | lst.remove("apple") |
| Delete by index | lst.pop(2) |
| Access element | lst[0], lst[-1], lst[1:4] |
| Check existence | "apple" in lst |
| List length | len(lst) |
| Sort list | lst.sort() |
| Reverse list | lst.reverse() |

## ✅ Practice Questions

Add and remove student names from a list

Create a list of even numbers from 1 to 20.

Find the average of marks from a list.

Check if a product is available in a shopping cart.

Count how many times a name appears in a list.

## 🧾 ****1. Dictionary in Python****

### ✅ ****Definition:****

A **dictionary** is a collection of key-value pairs. It is **unordered**, **mutable**, and does **not allow duplicate keys**.

### 🔹 ****Syntax:****

my\_dict = {

"key1": "value1",

"key2": "value2"

}

### ✅ ****Key Features:****

Uses {} curly braces.

Keys must be **unique and immutable** (strings, numbers, tuples).

Values can be any data type.

Dictionary is **mutable** (can be updated).

### 🔧 ****Common Dictionary Methods:****

| **Method** | **Description** |
| --- | --- |
| dict.keys() | Returns all keys |
| dict.values() | Returns all values |
| dict.items() | Returns key-value pairs as tuples |
| dict.get(key) | Returns value for key, or None if not found |
| dict.pop(key) | Removes key-value pair |
| dict.update() | Updates with another dictionary |

### 🎯 ****Real-Time Example: Student Marks Record****

student = {

"name": "John",

"roll\_no": 101,

"marks": {"math": 90, "science": 85}

}

print("Student Name:", student["name"])print("Math Marks:", student["marks"]["math"])

**Output:**

Student Name: JohnMath Marks: 90

student = {

"name": "Alice",

"age": 20,

"course": "BCA",

"marks": 85

}

🔧 1. get() – Returns the value for the given key

print(student.get("name")) # Alice

print(student.get("gender", "N/A")) # N/A (default if key not found)

🔧 2. keys() – Returns a list of all keys

print(student.keys()) # dict\_keys(['name', 'age', 'course', 'marks'])

🔧 3. values() – Returns a list of all values

print(student.values()) # dict\_values(['Alice', 20, 'BCA', 85])

🔧 4. items() – Returns key-value pairs as tuples

for key, value in student.items():

print(key, ":", value)

Output:

name : Alice

age : 20

course : BCA

marks : 85

🔧 5. update() – Updates existing key or adds a new one

student.update({"marks": 90, "gender": "Female"})

print(student)

Output:

{'name': 'Alice', 'age': 20, 'course': 'BCA', 'marks': 90, 'gender': 'Female'}

🔧 6. pop() – Removes a key and returns its value

removed = student.pop("age")

print("Removed age:", removed)

print(student)

Output:

Removed age: 20

{'name': 'Alice', 'course': 'BCA', 'marks': 90, 'gender': 'Female'}

🔧 7. popitem() – Removes the last inserted item

student.popitem()

print(student)

Output:

{'name': 'Alice', 'course': 'BCA', 'marks': 90}

🔧 8. clear() – Removes all items

student.clear()

print(student) # {}

🔧 9. copy() – Returns a shallow copy of the dictionary

original = {"a": 1, "b": 2}

duplicate = original.copy()

duplicate["a"] = 100

print("Original:", original)

print("Duplicate:", duplicate)

Output:

Original: {'a': 1, 'b': 2}

Duplicate: {'a': 100, 'b': 2}

🔧 10. fromkeys() – Creates a new dictionary from keys with the same value

subjects = ["Math", "Science", "English"]

default\_marks = dict.fromkeys(subjects, 0)

print(default\_marks)

Output:

{'Math': 0, 'Science': 0, 'English': 0}

🎯 Real-Time Example: Attendance Tracking

attendance = {}

students = ["Ravi", "Neha", "Kiran"]

for name in students:

attendance[name] = "Present"

print(attendance)

Output:

{'Ravi': 'Present', 'Neha': 'Present', 'Kiran': 'Present'}

## 🧾 ****2. Tuple in Python****

### ✅ ****Definition:****

A **tuple** is an **ordered**, **immutable** collection of items. Tuples are often used to store data that **should not change**.

### 🔹 ****Syntax:****

my\_tuple = (1, 2, 3)

### ✅ ****Key Features:****

Uses () parentheses.

Can contain mixed data types.

Supports indexing and slicing.

**Immutable** – values cannot be changed once created.

Faster than lists due to immutability.

### 🔧 ****Common Tuple Methods:****

| **Method** | **Description** |
| --- | --- |
| tuple.count(x) | Counts occurrences of value x |
| tuple.index(x) | Returns index of first x |

### ✅ ****Key Characteristics of Tuples:****

| **Feature** | **Description** |
| --- | --- |
| Ordered | Elements maintain the order they were added. |
| Immutable | Cannot change, add, or remove elements. |
| Indexable | Access elements using indexing like my\_tuple[0]. |
| Heterogeneous | Can contain different data types (int, str, etc.). |
| Faster than Lists | Due to immutability, especially in large data. |

### 📘 ****Creating Tuples****

# Empty tuple

empty = ()

# Tuple with integers

nums = (1, 2, 3)

# Mixed data types

mixed = ("John", 25, 5.6, True)

# Tuple without parentheses (comma-separated values)

no\_parens = 10, 20, 30

# Tuple with one element (note the comma)

single = (5,)

### 🔍 ****Accessing Tuple Elements****

person = ("Alice", "HR", 30000)print(person[0]) # Aliceprint(person[-1]) # 30000

### 🔁 ****Iterating Through a Tuple****

colors = ("red", "green", "blue")for color in colors:

print(color)

### 🔧 ****Tuple Methods****

| **Method** | **Description** | **Example** |
| --- | --- | --- |
| count(x) | Counts occurrences of value x | (1, 2, 2, 3).count(2) → 2 |
| index(x) | Returns first index of value x | ("a", "b", "c").index("b") → 1 |

### 🛠️ ****Tuple Operations****

a = (1, 2)

b = (3, 4)

# Concatenationprint(a + b) # (1, 2, 3, 4)

# Repetitionprint(a \* 2) # (1, 2, 1, 2)

# Membershipprint(2 in a) # True

# Lengthprint(len(b)) # 2

### 🎯 ****Real-Time Example 1: Coordinates (Latitude, Longitude)****

location = (12.9716, 77.5946)print("Latitude:", location[0])print("Longitude:", location[1])

### 🎯 ****Real-Time Example 2: Storing Student Info (Immutable)****

student = ("Ravi", "BCA", "ANP-D1108")print(f"Name: {student[0]}, Course: {student[1]}, Batch: {student[2]}")

## ✅ When to Use a Tuple?

When the data should **not change**.

When you want to use a data structure as a **dictionary key**.

For **fixed collections** like coordinates, dates, or status codes.

### 🎯 ****Real-Time Example: Coordinates or Login Info****

# GPS Coordinates (Latitude, Longitude)

location = ("12.9716", "77.5946")print("Latitude:", location[0])print("Longitude:", location[1])

**Output:**

Latitude: 12.9716Longitude: 77.5946

### ✅ Another Example: Employee Info as Tuple

employee = ("Alice", "HR", 30000)print(f"Name: {employee[0]}, Department: {employee[1]}, Salary: {employee[2]}")

**Output:**

Name: Alice, Department: HR, Salary: 30000

## 📌 Summary Table:

| **Feature** | **Dictionary** | **Tuple** |
| --- | --- | --- |
| Type | Key-value pairs | Ordered values |
| Mutable | Yes | No |
| Syntax | {} | () |
| Access | By key | By index |
|  |  |  |
| Use case | When key-based access is needed | When data shouldn't change |

## 🔷 ****Set in Python****

### ✅ ****Definition:****

A **set** is an **unordered**, **unindexed**, and **mutable** collection of **unique** elements in Python. It is commonly used to store non-duplicate data and to perform mathematical set operations like union, intersection, and difference.

## 🔹 ****Syntax:****

my\_set = {1, 2, 3}

Or using the set() constructor:

my\_set = set([1, 2, 3])

❗ Note: Empty set must be created using set(), not {}, which creates an empty dictionary.

## ✅ ****Key Features of Sets:****

| **Feature** | **Description** |
| --- | --- |
| Unordered | No indexing or slicing |
| Unique Items | Duplicates are automatically removed |
| Mutable | Elements can be added or removed |
| Heterogeneous | Can store different types (int, str, float, etc.) |

## 🔧 ****Basic Set Operations****

### 📌 Creating a Set:

fruits = {"apple", "banana", "orange", "apple"}print(fruits) # Duplicates removed

**Output:**

{'banana', 'orange', 'apple'}

### 📌 Adding Elements

#### add() – Add a single element

fruits.add("mango")print(fruits)

### 📌 Removing Elements

#### remove() – Removes element; throws error if not found

fruits.remove("banana")

#### discard() – Removes element; **no error** if not found

fruits.discard("banana")

#### pop() – Removes a random element

fruits.pop()

#### clear() – Removes all elements

fruits.clear()

## 🔁 ****Set Iteration****

for fruit in fruits:

print(fruit)

## 🔍 ****Set Operations****

### 1. union() – Combines two sets

a = {1, 2, 3}

b = {3, 4, 5}print(a.union(b)) # {1, 2, 3, 4, 5}

### 2. intersection() – Common elements

print(a.intersection(b)) # {3}

### 3. difference() – Elements in a not in b

print(a.difference(b)) # {1, 2}

### 4. symmetric\_difference() – Elements in either set but not both

print(a.symmetric\_difference(b)) # {1, 2, 4, 5}

## 📘 ****Set Methods Summary Table****

| **Method** | **Description** |
| --- | --- |
| add(x) | Adds element x to the set |
| remove(x) | Removes x; error if not present |
| discard(x) | Removes x; no error if not present |
| pop() | Removes a random element |
| clear() | Clears all elements |
| union(set2) | Combines both sets |
| intersection(set2) | Common elements |
| difference(set2) | Elements in set1 but not in set2 |
| symmetric\_difference() | Unique elements from both sets |
| issubset(set2) | Checks if set1 is subset of set2 |
| issuperset(set2) | Checks if set1 is superset of set2 |
| isdisjoint(set2) | True if no common elements |

## 🎯 ****Real-Time Example: Finding Unique Visitors on a Website****

day1\_visitors = {"Alice", "Bob", "John", "Neha"}

day2\_visitors = {"Neha", "Bob", "David"}

# Unique visitors over 2 days

all\_visitors = day1\_visitors.union(day2\_visitors)print("All Visitors:", all\_visitors)

# Repeated visitors

repeat\_visitors = day1\_visitors.intersection(day2\_visitors)print("Repeated Visitors:", repeat\_visitors)

# Only on Day 1

day1\_only = day1\_visitors.difference(day2\_visitors)print("Visited only on Day 1:", day1\_only)

## ✅ When to Use Sets?

To remove duplicates from a list

To perform fast membership tests (x in set)

To apply mathematical set operations

For storing unique, unordered items (e.g., tags, categories, users)

## 📘 ****Dictionaries in Python****

### 🔹 ****Definition:****

A **dictionary** in Python is an unordered collection of **key-value** pairs. Each key is unique and maps to a value.

**Syntax:**

my\_dict = {

"key1": "value1",

"key2": "value2"

}

### 🔹 ****Key Features:****

Unordered (as of Python 3.6+, insertion order is preserved).

Mutable (can be changed).

Indexed by keys (not numbers like lists).

Keys must be **immutable** types (e.g., string, number, tuple).

Values can be of any type.

### 🔹 ****Creating Dictionaries:****

# Using curly braces

student = {

"name": "Alice",

"age": 22,

"course": "BCA"

}

# Using dict() constructor

employee = dict(id=101, name="John", salary=50000)

### 🔹 ****Accessing Values:****

print(student["name"]) # Alice

print(student.get("course")) # BCA

### 🔹 ****Modifying/Updating Values:****

student["age"] = 23

student["college"] = "St. Xavier's"

### 🔹 ****Deleting Items:****

del student["course"]

student.pop("age")

student.clear() # Clears entire dictionary

### 🔹 ****Iterating Through a Dictionary:****

for key in student:

print(key, student[key])

for key, value in student.items():

print(f"{key}: {value}")

### 🔹 ****Dictionary Methods:****

| **Method** | **Description** |
| --- | --- |
| dict.keys() | Returns all keys |
| dict.values() | Returns all values |
| dict.items() | Returns all key-value pairs as tuples |
| dict.get(key) | Returns value for key or None |
| dict.update(dict2) | Adds or updates from another dictionary |
| dict.pop(key) | Removes the item with given key |
| dict.clear() | Removes all items |
| key in dict | Checks if key exists |

### 🔹 ****Real-Time Examples:****

#### ✅ Example 1: Student Marks Record

marks = {

"Alice": 85,

"Bob": 78,

"Charlie": 92

}

# Update marks

marks["Bob"] = 82

# Add new student

marks["Diana"] = 88

# Print reportfor student, mark in marks.items():

print(f"{student} scored {mark} marks.")

#### ✅ Example 2: Inventory System

inventory = {

"pen": 120,

"notebook": 75,

"eraser": 40

}

# Check availability

item = input("Enter item to check: ")if item in inventory:

print(f"{item} is available: {inventory[item]} in stock.")else:

print(f"{item} is not available.")

### 🔹 ****Nested Dictionaries:****

students = {

"101": {"name": "Alice", "age": 20},

"102": {"name": "Bob", "age": 21}

}

print(students["101"]["name"]) # Alice

### ✅ Use Cases of Dictionary in Real Life:

**User profiles** (e.g., storing user data in a web app)

**Configuration settings** in applications

**JSON** data parsing and handling

**Word-frequency counter** in NLP

**Exception Handling**

An **exception** is an error that occurs **during the execution** of a program, which **interrupts** the normal flow of instructions.  
Python provides a way to handle exceptions using try-except blocks.

### 3. ****Why Use Exception Handling?****

Prevents program from crashing

Handles runtime errors gracefully

Improves user experience and debugging

### 🔹 ****Common Built-in Exceptions in Python:****

| **Exception Type** | **Description** |
| --- | --- |
| ZeroDivisionError | Raised when a number is divided by zero |
| ValueError | Raised when a function receives the wrong type |
| TypeError | Raised when an operation is applied to an inappropriate type |
| IndexError | Raised when accessing an invalid index in a list |
| KeyError | Raised when a dictionary key is not found |
| FileNotFoundError | Raised when trying to open a file that doesn't exist |

### 🔹 ****Syntax of Try-Except Block:****

try:

# Code that may raise an exception

except ExceptionType:

# Code to handle the exception

### Try:

Use try block to write risky code.

### Except:

Use except to handle specific or general exceptions.

### Finally:

Use finally to ensure cleanup code runs.

### Else:

Use else (optional) to run code only if no exception occurs.

### Raise:

Throw an exception manually

Ex:

try:

a = [1, 2, 3]

print(a[5])

except IndexError:

print("Index out of range!")

except Exception as e:

print("Some error occurred:", e)

2.

try:

num = int(input("Enter a number: "))

result = 100 / num

except ZeroDivisionError:

print("Divide by zero error!")

else:

print("Result is:", result)

### ****Division (Handling ZeroDivisionError)****

try:

num1 = int(input("Enter numerator: "))

num2 = int(input("Enter denominator: "))

result = num1 / num2

print("Result:", result)

except ZeroDivisionError:

print("Error: Cannot divide by zero.")

except ValueError:

print("Error: Please enter valid integers.")

****File Handling (Handling FileNotFoundError)****

try:

file = open("data.txt", "r")

content = file.read()

print(content)

file.close()except FileNotFoundError:

print("Error: File not found.")

### 🔹 ****Using**** finally ****Block (Optional)****

The finally block runs **no matter what** (whether an exception occurred or not).

try:

print("Trying to open file...")

f = open("demo.txt", "r")

print(f.read())

except FileNotFoundError:

print("File not found!")

finally:

print("Closing file (if opened).")

**Using Else:**

try:

num = int(input("Enter a number: "))

result = 100 / num

except ZeroDivisionError:

print("Divide by zero error!")

else:

print("Result is:", result)

### ✅ ****1. ATM PIN Validation (Handling**** ValueError****)****

try:

pin = int(input("Enter your 4-digit ATM PIN: "))

if len(str(pin)) != 4:

raise ValueError("PIN must be 4 digits.")

print("PIN accepted. Access granted.")except ValueError as ve:

print("Invalid input:", ve)

### ✅ ****2. Student Grade Calculator (Handling**** ZeroDivisionError****)****

try:

total\_marks = int(input("Enter total marks obtained: "))

total\_subjects = int(input("Enter number of subjects: "))

average = total\_marks / total\_subjects

print("Average Marks:", average)except ZeroDivisionError:

print("Subjects cannot be zero.")except ValueError:

print("Please enter numeric values only.")

### ✅ ****3. Dictionary Lookup (Handling**** KeyError****)****

student\_data = {

"101": "Alice",

"102": "Bob",

"103": "Charlie"

}

try:

roll\_no = input("Enter student roll number: ")

print("Student Name:", student\_data[roll\_no])

except KeyError:

print("Error: Roll number not found.")

### ✅ ****4. List Access (Handling**** IndexError****)****

fruits = ["apple", "banana", "mango"]

try:

index = int(input("Enter index (0-2) to get a fruit: "))

print("Selected fruit:", fruits[index])

except IndexError:

print("Error: Index out of range.")

except ValueError:

print("Please enter a valid number.")

### ✅ ****5. Age Validator (Handling Custom Exceptions with**** raise****)****

try:

age = int(input("Enter your age: "))

if age < 18:

raise Exception("You must be at least 18 years old to register.")

print("Registration successful!")

except Exception as e:

print("Error:", e)

### ✅ ****6. File Upload Simulation (Handling**** FileNotFoundError****)****

try:

filename = input("Enter filename to read: ")

with open(filename, "r") as f:

print(f.read())except FileNotFoundError:

print("Error: The file does not exist.")

### ✅ ****7. Email Format Checker (Handling**** ValueError****)****

### try:

### email = input("Enter your email address: ")

### if "@" not in email or "." not in email:

### raise ValueError("Invalid email format.")

### print("Email accepted.")

### except ValueError as ve:

### print("Error:", ve)

### ****9.Custom Exception Class****

You can define your own exception by extending the Exception class:

class NegativeNumberError(Exception):

pass # Correct lowercase 'pass'

num = int(input("Enter a positive number: "))

if num < 0:

raise NegativeNumberError("Negative numbers are not allowed.")

### 🔹 10. ****Real-Time Example****

**Banking Application – Withdrawal Scenario:**

class InsufficientFunds(Exception):

pass

balance = 5000

withdraw = int(input("Enter amount to withdraw: "))

try:

if withdraw > balance:

raise InsufficientFunds("Not enough balance!")

balance -= withdraw

print("Withdrawal successful. Remaining balance:", balance)

except InsufficientFunds as e:

print€

## What is File I/O?

File I/O (Input/Output) in Python refers to the process of reading from and writing to files. It helps in:

Storing data permanently

Reading data from files like .txt, .csv, etc.

Logging or writing output of a program

## 🔹 Opening a File

Python uses the built-in open() function:

open(file\_name, mode)

### Common Modes:

| **Mode** | **Description** |
| --- | --- |
| 'r' | Read (default). Error if file doesn’t exist |
| 'w' | Write. Creates a new file or overwrites existing |
| 'a' | Append. Adds data to the end of the file |
| 'x' | Create. Error if file already exists |
| 'b' | Binary mode |
| 't' | Text mode (default) |
| '+' | Read and write |

## 🔹 Reading from a File

### 1. read() – Reads entire content

with open("data.txt", "r") as file:

content = file.read()

print(content)

### 2. readline() – Reads one line at a time

with open("data.txt", "r") as file:

line = file.readline()

print(line)

### 3. readlines() – Reads all lines into a list

t

with open("data.txt", "r") as file:

lines = file.readlines()

for line in lines:

print(line.strip())

## 🔹 Writing to a File

### 1. write() – Writes a single string

with open("output.txt", "w") as file:

file.write("Hello, this is a file write demo!\n")

file.write("Writing another line.")

### 2. writelines() – Writes list of strings

lines = ["Line 1\n", "Line 2\n", "Line 3\n"]with open("output.txt", "w") as file:

file.writelines(lines)

## 🔹 Appending to a File

with open("output.txt", "a") as file:

file.write("\nAppending a new line at the end.")

## 🔹 File Handling Best Practices

Use with open(...) to automatically close the file.

Always handle exceptions while dealing with file I/O.

## 🔹 Checking File Exists

import os

if os.path.exists("data.txt"):

print("File exists!")else:

print("File not found!")

## 🔹 Real-time Example 1: Count Words in a File

with open("sample.txt", "r") as file:

text = file.read()

words = text.split()

print("Total words:", len(words))

## 🔹 Real-time Example 2: Log Student Attendance

students = ["Alice", "Bob", "Charlie"]with open("attendance.txt", "w") as file:

for student in students:

file.write(f"{student} - Present\n")

## 🔹 Real-time Example 3: Search a Word in File

word\_to\_find = "Python"

found = False

with open("notes.txt", "r") as file:

for line in file:

if word\_to\_find in line:

found = True

break

if found:

print(f"'{word\_to\_find}' found in the file!")else:

print(f"'{word\_to\_find}' not found.")

## 🔹 Deleting a File

import os

if os.path.exists("oldfile.txt"):

os.remove("oldfile.txt")else:

print("File does not exist.")

## 🔚 Summary

Use open() with correct mode ('r', 'w', 'a')

Use with open() for better resource handling

Use read(), readline(), or readlines() to read

Use write() or writelines() to write

Always validate file existence before operations

## ✅ ****Ways to Create a File in Python****

### 🔹 1. Using 'w' Mode – Write Mode

Creates a new file if it does not exist.

If file **already exists**, it **overwrites** the content.

# Creates 'example1.txt' or overwrites if it already exists

file = open("example1.txt", "w")

file.write("This is a new file created using write mode.\n")

file.close()

### 🔹 2. Using 'x' Mode – Create Mode

Creates a new file.

If file **already exists**, it raises a FileExistsError.

try:

file = open("example2.txt", "x")

file.write("This file was created using exclusive creation mode.")

file.close()except FileExistsError:

print("File already exists!")

### 🔹 3. Best Practice – Using with open(...)

Automatically closes the file:

with open("example3.txt", "w") as file:

file.write("Created using 'with open' context manager.")

## 📁 Result

After executing any of the above, the file (e.g., example1.txt) will be created in the **current working directory**.

## 🔍 Optional: Check File Created

import os

if os.path.exists("example1.txt"):

print("File created successfully!")else:

print("File not found.")

## 1. Handling ****CSV Data**** in Python

### 🔹 Step 1: Import csv module

import csv

### 🔹 Step 2: Read from a CSV file

with open('students.csv', mode='r') as file:

reader = csv.reader(file)

for row in reader:

print(row)

📌 **Assume** students.csv **file contains:**

Name,Age,Grade

Alice,20,A

Bob,21,B

### 🔹 Step 3: Write to a CSV file

data = [

["Name", "Age", "Grade"],

["John", 22, "A"],

["Jane", 23, "B"]

]

with open('new\_students.csv', mode='w', newline='') as file:

writer = csv.writer(file)

writer.writerows(data)

### 🔹 Step 4: Read CSV as a dictionary

with open('students.csv', mode='r') as file:

reader = csv.DictReader(file)

for row in reader:

print(f"Name: {row['Name']}, Age: {row['Age']}, Grade: {row['Grade']}")

### 🔹 Step 5: Write CSV using dictionary

students = [

{"Name": "Tom", "Age": 24, "Grade": "A"},

{"Name": "Jerry", "Age": 22, "Grade": "B"}

]

with open('dict\_students.csv', mode='w', newline='') as file:

fieldnames = ["Name", "Age", "Grade"]

writer = csv.DictWriter(file, fieldnames=fieldnames)

writer.writeheader()

writer.writerows(students)

## ✅ 2. Handling ****JSON Data**** in Python

### 🔹 Step 1: Import json module

import json

### 🔹 Step 2: Convert Python object to JSON (serialization)

student = {"name": "Alice", "age": 20, "grade": "A"}

json\_data = json.dumps(student)print(json\_data)

### 🔹 Step 3: Write JSON to a file

with open('student.json', mode='w') as file:

json.dump(student, file)

### 🔹 Step 4: Read JSON from a file

with open('student.json', mode='r') as file:

data = json.load(file)

print(data)

### 🔹 Step 5: Convert JSON string to Python object (deserialization)

json\_string = '{"name": "Bob", "age": 21, "grade": "B"}'

student = json.loads(json\_string)print(student['name'])

## 🔄 CSV vs JSON Summary

| **Feature** | **CSV** | **JSON** |
| --- | --- | --- |
| Format | Tabular (rows & columns) | Hierarchical (key-value) |
| Use case | Spreadsheets, flat data | Nested data, APIs, configs |
| Module | csv | json |
| File Extension | .csv | .json |