***Python Basics***

This block of code is at the beginning to make sure that when we print we see only the output and not the useless stuffs:

import os

os.system('cls' if os.name == 'nt' else 'clear')

**This is an f string, this allows us to have placeholders for a variable inside the quotations itself:**

# first = "Bro"  
# print(f"Hello, {first} !")

**TYPECASTING:**

# gpa = 3.98  
# print(f"Your GPA is {int(gpa)}") # 3   
# print(f"Your GPA is {gpa}") # 3.98

**TYPECASTING + USER INPUT:**

# name = input("Enter your name: ")  
# age = int(input("Enter your age: "))  
# print(f"Hello, {name} !")   
# print (f"You are {age} years old.")  
# print(f"Next year, you will be {age+1} years old.")

**MATHS:**x = 4  
y = 3  
z = 15   
# power1 = x \*\* 3 # This is to the power of 3  
# power2 = pow(x, 3) # This is also to the power of 3  
# print(power2)

# print(max(x, y, z)) # THIS finds the max number

# print(min(x, y, z)) # This finds the minimum number

# We have various other math functions like abs, round, floor, ceil, etc.

# We can use the math module to access these functions.

# import math

# print(math.sqrt(16))

# print(math.floor(3.9))

# print(math.ceil(3.1))

# print(math.pow(2, 3))

# print(math.pi)

# print(math.e)

# print(math.inf)

# print(math.nan)

# print(round(3.512121, 2)) # 3.51

# We also have % operator which gives the remainder of the division.

# print(10 % 3) # 1

# print(10 % 2) # 0

**Recursive Function:**

The order of the conditions is important # if we put this condition at the end, it will never be executed # because the previous condition will be executed first

# def Suggest\_Clothes(temp):

# if temp >= 50:

# return "INVALID, TEMPERATURE TOO HIGH"

# elif temp >= 30:

# return "Wear light clothes"

# elif 20 <= temp < 30:

# return "Wear a T-shirt"

# elif temp > 10:

# return "Wear a jacket"

# else:

# return "Wear a coat"

# temp = int(input("Enter the temperature (celcius): "))

# print(Suggest\_Clothes(temp))

**Logical operators:**

# and, or, not

# They are used to combine multiple conditions.

# and: returns True if all conditions are True

# or: returns True if at least one condition is True

# not: returns the opposite of the condition like True -> False, False -> True

**Conditional expressions:**

It is a shorthand for if else statement, **X if condition else Y**

# x = 4 if 3 > 2 else 5. HERE X: x = 4 and condition = 3>2 and Y = 5

# print(x) # 4

**Here are some string methods (use them like name.method())**

# len() # these are used to get the length of the string. This counts the spaces as well, and from 1 not 0

# \_sizeof\_() # these are used to get the size of the string in bytes

# capitalize() # these are used to capitalize the first letter of the string

# upper() # these are used to convert the string to uppercase or lowercase

# lower() # these are used to convert the string to uppercase or lowercase

# title() # these are used to capitalize the first letter of each word

# swapcase() # these are used to swap the case of the string

# strip() # these are used to remove the spaces

# lstrip() # these are used to remove the spaces

# rstrip() # these are used to remove the spaces

# count() # these are used to count the number of times a substring appears in the string

# find() # these are used to find the index of the string, this counts from 0

# rfind() # these are used to find the index of the string, This counts from the end

# index() # these are used to find the index of the string

# rindex() # these are used to find the index of the string

# startswith() # these are used to check the string

# endswith() # these are used to check the string

# isalnum() # these are used to check the string

# isalpha() # these are used to check the string

# isdigit() # these are used to check the string

# islower() # these are used to check the string

# isupper() # these are used to check the string

# isspace() # these are used to check the string

# split() # these are used to split the string, example split(" ") will split the string at the spaces, split(",") will split the string at the commas.

# rsplit() # these are used to split the string

# replace() # these are used to replace the string

# format() # these are used to format the string

# format\_map() # these are used to format the string

# encode() # these are used to encode the string

# zfill() # these are used to fill the string with zeros

# partition() # these are used to partition the string

# rpartition() # these are used to partition the string

# expandtabs() # these are used to expand the tabs

# translate() # these are used to translate the string

# maketrans() # these are used to make a translation table

# splitlines() # these are used to split the string at the line breaks

# join() # these are used to join the strings

# rjust() # these are used to justify the string to the right

# center() # these are used to center the string

# ljust() # these are used to justify the string to the left

**Format Specifiers in Python are used to format the output. {value:flags}**

# price1 = 3.14159

# price2 = -987.65

# price3= 12.34

# print(f"Price 1 is ${price1:.2f}") # 2 decimal places

# print(f"Price 2 is ${price2:10}") # 10 spaces

# print(f"Price 3 is ${price3:<10}") # left justified

# print(f"Price 3 is ${price3:>10}") # right justified

# print(f"Price 3 is ${price3:^10}") # center justified

# print(f"Price 3 is ${price3:010}") # fill with zeros

# price4 = 3000.14159

# print(f"Price 4 is ${price4:+,.2f}") # comma separated, 2 decimal places, + sign

**While Loop:**

# name = input("Enter your name: ")

# while name == "":

# print("Please enter your name")

# name = input("Enter your name: ")

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

# name = input("Enter your name (or type 'quit' to exit): ")

# while name != "quit": # Type "quit" to exit the loop

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

# name = input("Enter your name: ")

# print("Goodbye!")

**For Loop:**

# for i in range(5): # 0 to 4 # range(5) is equivalent to range(0, 5)

# print(i)

# for i in range(1, 5): # 1 to 4

# print(i)

# for i in range(1, 10, 2): # 1 to 9, step 2

# print(i)

# for i in range(10, 1, -1): # 10 to 2, step -1

# print(i)

# for i in reversed(range(5)): # 4 to 0

# print(i)

# for i in reversed(range(1, 5)): # 4 to 1

# print(i)

# num = 123456789

# for x in str(num):

# print(x)

# for x in range(1,21):

# if x == 13:

# continue/break

# else:

# print(x)

# for x in range(1,21):

# print(x, end="") # 1234567891011121314151617181920

**# # Positional arguments: These are the most common type of arguments and are passed to the function in the correct positional order.**

# def greet(name, age):

# print(f"Hello, {name}! You are {age} years old.")

# greet("Alice", 30) # Positional arguments

**# # Default arguments: These arguments take a default value if no value is provided during the function call.**

# def greet(name, age=25):

# print(f"Hello, {name}! You are {age} years old.")

# greet("Bob") # Default argument for age

# greet("Charlie", 35) # Overriding the default argument

**# # Keyword arguments: These arguments are passed to the function by explicitly specifying the parameter name.**

# def greet(name, age):

# print(f"Hello, {name}! You are {age} years old.")

# greet(name="David", age=40) # Keyword arguments

# greet(age=45, name="Eve") # Order doesn't matter with keyword arguments

**# # Arbitrary arguments: These arguments allow you to pass a variable number of arguments to a function. They are defined using \*args and \*\*kwargs.**

# def greet(\*names):

# for name in names:

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

# greet("Frank", "Grace", "Hank") # Arbitrary positional arguments

# def greet(\*\*person):

# print(f"Hello, {person['name']}! You are {person['age']} years old.")

# greet(name="Ivy", age=50) # Arbitrary keyword arguments

**MATCH CASE STATEMENT:**

# def day\_of\_week(day):

# match day:

# case "Monday":

# print("It's Monday!")

# case "Tuesday":

# print("It's Tuesday!")

# case "Wednesday":

# print("It's Wednesday!")

# case "Thursday":

# print("It's Thursday!")

# case "Friday":

# print("It's Friday!")

# case "Saturday":

# print("It's Saturday!")

# case "Sunday":

# print("It's Sunday!")

# case \_:

# print("Invalid day!")

**# CLASSES:**

# class Student:

# class\_year = 2025

# def \_\_init\_\_(self, name, age):

# self.name = name

# self.age = age

# def greet(self):

# print(f"Hello, {self.name}! You are {self.age} years old.")

# student1 = Student("Alice", 20)

# student2 = Student("Bob", 25)

# student1.greet()

# student2.greet()

**# Magic methods (also called dunder methods because they have double underscores \_\_ before and after their names) are special methods in Python that allow objects to define their behavior in built-in operations like arithmetic, comparisons, string representation, and more.**

**# Common Magic Methods:**

**# 1. Object Initialization & Representation**

# \_\_init\_\_(self, ...) → Constructor, initializes an instance.

# \_\_str\_\_(self) → String representation for str(obj).

# \_\_repr\_\_(self) → String representation for debugging (repr(obj)).

**# 2. Arithmetic Operations**

# \_\_add\_\_(self, other) → Defines self + other.

# \_\_sub\_\_(self, other) → Defines self - other.

# \_\_mul\_\_(self, other) → Defines self \* other.

# \_\_truediv\_\_(self, other) → Defines self / other.

# \_\_floordiv\_\_(self, other) → Defines self // other.

# \_\_mod\_\_(self, other) → Defines self % other.

**# 3. Comparison Operators**

# \_\_eq\_\_(self, other) → Defines self == other.

# \_\_ne\_\_(self, other) → Defines self != other.

# \_\_lt\_\_(self, other) → Defines self < other.

# \_\_le\_\_(self, other) → Defines self <= other.

# \_\_gt\_\_(self, other) → Defines self > other.

# \_\_ge\_\_(self, other) → Defines self >= other.

**# 4. Attribute Access & Management**

# \_\_getattr\_\_(self, name) → Defines behavior for accessing non-existent attributes.

# \_\_setattr\_\_(self, name, value) → Defines behavior for setting attributes.

# \_\_delattr\_\_(self, name) → Defines behavior for deleting attributes.

**# 5. Container & Sequence Methods**

# \_\_len\_\_(self) → Defines len(self).

# \_\_getitem\_\_(self, key) → Defines self[key].

# \_\_setitem\_\_(self, key, value) → Defines self[key] = value.

# \_\_delitem\_\_(self, key) → Defines del self[key].

**# 6. Object Call & Iteration**

# \_\_call\_\_(self, \*args, \*\*kwargs) → Allows an instance to be called like a function.

# \_\_iter\_\_(self) → Defines iteration behavior (for x in obj).

# \_\_next\_\_(self) → Defines next(obj).

# Example Usage:

# class Point:

# def \_\_init\_\_(self, x, y):

# self.x = x

# self.y = y

# def \_\_add\_\_(self, other):

# return Point(self.x + other.x, self.y + other.y)

# def \_\_str\_\_(self):

# return f"Point({self.x}, {self.y})"

# p1 = Point(2, 3)

# p2 = Point(4, 5)

# p3 = p1 + p2 # Calls \_\_add\_\_

# print(p3) # Calls \_\_str\_\_: Output -> Point(6, 8)

**# Getter and Setter Methods**

# class Student:

# def \_\_init\_\_(self, name, age):

# self.\_name = name

# self.\_age = age

# @property

# def get\_name(self):

# return self.\_name

# @name.setter

# def set\_name(self, name):

# self.\_name = name

# @property

# def get\_age(self):

# return self.\_age

# @age.setter

# def set\_age(self, age):

# self.\_age = age

# student = Student("Alice", 20)

# print(student.get\_name()) # Alice

# student.set\_name("Bob")

# print(student.get\_name()) # Bob

# print(student.get\_age()) # 20

# student.set\_age(25)

# print(student.get\_age()) # 25

**# Exception handling in Python allows you to gracefully handle errors that may occur during program execution, preventing crashes and providing meaningful error messages.**

**# 1. What is an Exception?**

# An exception is an error that occurs during program execution, stopping normal flow. Examples include:

# ZeroDivisionError: Division by zero.

# TypeError: Invalid operations between data types.

# FileNotFoundError: Accessing a missing file.

**# 2. Handling Exceptions Using try-except**

# Python provides the try-except block to catch and handle exceptions.

# try:

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

# result = 10 / num

# print("Result:", result)

# except ZeroDivisionError:

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

# except ValueError:

# print("Error: Please enter a valid number.")

# except Exception as e:

# print(f"Unexpected error: {e}")

**# How it works:**

**# Code inside try runs normally.**

**# If an exception occurs, it jumps to except, handling the error.**

**# except Exception as e catches any unexpected errors.**

**# 3. Using finally for Cleanup**

# The finally block always executes, even if an exception occurs. It is used for cleanup tasks like closing files or network connections.

# try:

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

# content = file.read()

# except FileNotFoundError:

# print("File not found.")

# finally:

# print("Closing the file.")

# file.close()

**# 4. Raising Custom Exceptions (raise)**

# You can manually raise exceptions using raise.

# def withdraw(amount):

# if amount > 1000:

# raise ValueError("Withdrawal limit exceeded!")

# print(f"Withdrew {amount}")

# try:

# withdraw(1500)

# except ValueError as e:

# print(f"Transaction failed: {e}")

**# 5. Custom Exception Classes**

# You can define your own exception types by inheriting from Exception.

# class InsufficientFundsError(Exception):

# pass

# def withdraw(balance, amount):

# if amount > balance:

# raise InsufficientFundsError("Not enough funds.")

# print(f"Withdrew {amount}")

# try:

# withdraw(500, 1000)

# except InsufficientFundsError as e:

# print(f"Error: {e}")

**File Handling in Python**

File handling in Python allows reading, writing, and manipulating files (text or binary). Python provides built-in functions to work with files efficiently.

**1. Opening a File (open())**

Python uses the open() function to access files. Syntax:

file = open("filename.txt", mode)

**File Modes:**

| **Mode** | **Meaning** |
| --- | --- |
| 'r' | Read mode (default). File must exist. |
| 'w' | Write mode. Creates file if not exists, overwrites if exists. |
| 'a' | Append mode. Adds data to file without deleting existing content. |
| 'x' | Exclusive creation. Fails if file exists. |
| 'b' | Binary mode (e.g., images, videos). |
| 't' | Text mode (default). |

Example:

file = open("example.txt", "r") # Open in read mode

print(file.read()) # Read and print file content

file.close() # Always close the file

**2. Reading a File**

Python provides multiple ways to read a file.

**a) Read the entire file**

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

content = file.read()

print(content)

file.close()

**b) Read line-by-line (readline())**

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

print(file.readline()) # Reads one line

file.close()

**c) Read all lines as a list (readlines())**

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

lines = file.readlines()

print(lines) # Returns a list of lines

file.close()

**3. Writing to a File ('w' or 'a' mode)**

* 'w' mode **overwrites** existing content.
* 'a' mode **appends** to the file.

**a) Overwriting a File ('w' mode)**

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

file.write("Hello, World!\n")

file.write("Writing to a file in Python.")

file.close()

**b) Appending to a File ('a' mode)**

file = open("example.txt", "a")

file.write("\nAppending this line.")

file.close()

**4. Using with Statement (Best Practice)**

Using with automatically closes the file after use.

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

content = file.read()

print(content)

**5. Working with Binary Files ('b' mode)**

Binary mode is used for images, audio, video, etc.

**Reading a binary file**

with open("image.jpg", "rb") as file:

data = file.read()

print(data[:10]) # Print first 10 bytes

**Writing to a binary file**

with open("copy.jpg", "wb") as file:

file.write(data)

**6. Checking if a File Exists (os and pathlib)**

Before reading a file, check if it exists.

import os

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

print("File exists")

else:

print("File does not exist")

Using pathlib (recommended in modern Python):

from pathlib import Path

file\_path = Path("example.txt")

if file\_path.exists():

print("File exists")

**1. Writing to a .txt File**

You can use Python's built-in open() function.

**Example: Writing to a .txt File**

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

file.write("Hello, World!\n")

file.write("This is a text file.\n")

* **Mode 'w'**: Overwrites the file if it exists.
* **Mode 'a'**: Appends to the existing file.

**2. Writing to a .csv File**

Python's csv module makes it easy to write structured CSV data.

**Example: Writing to a CSV File**

import csv

data = [

["Name", "Age", "City"],

["Alice", 25, "New York"],

["Bob", 30, "San Francisco"],

]

with open("example.csv", "w", newline="") as file:

writer = csv.writer(file)

writer.writerows(data) # Writes multiple rows at once

* **newline=""** prevents extra newlines on Windows.
* **writerow()** writes a single row.
* **writerows()** writes multiple rows.

**3. Writing to a .json File**

Python’s json module is used for writing JSON files.

**Example: Writing a JSON File**

import json

data = {

"name": "Alice",

"age": 25,

"city": "New York",

"skills": ["Python", "Machine Learning"]

}

with open("example.json", "w") as file:

json.dump(data, file, indent=4) # Pretty-printed JSON

* **json.dump()** writes a dictionary to a file.
* **indent=4** formats the JSON for readability.

**Multithreading in Python**

Multithreading in Python allows running multiple threads concurrently to improve performance in I/O-bound tasks (e.g., file handling, network requests). Python's threading module provides tools to work with threads.

**1. Why Use Multithreading?**

✅ **Improves performance** for tasks that involve waiting (e.g., file I/O, web scraping).  
✅ **Runs tasks in parallel**, making programs more efficient.  
✅ **Better resource utilization**, especially on multi-core CPUs.

🚨 **Note:** Due to Python's **Global Interpreter Lock (GIL)**, threads **don’t execute Python bytecode in true parallelism**. Use **multiprocessing** for CPU-bound tasks.

**2. Creating a Thread in Python**

Python’s threading.Thread is used to create and start threads.

**Example: Creating and Running Threads**

import threading

def print\_numbers():

for i in range(5):

print(f"Number: {i}")

# Create a thread

thread = threading.Thread(target=print\_numbers)

# Start the thread

thread.start()

# Main thread continues execution

print("Main thread running...")

# Wait for the thread to complete

thread.join()

print("Thread finished execution.")

**Output Example (May Vary)**

Main thread running...

Number: 0

Number: 1

Number: 2

Number: 3

Number: 4

Thread finished execution.

**3. Using Multiple Threads**

You can run multiple threads in parallel.

import threading

import time

def task(name):

for i in range(3):

print(f"Task {name}: {i}")

time.sleep(1)

# Creating multiple threads

thread1 = threading.Thread(target=task, args=("A",))

thread2 = threading.Thread(target=task, args=("B",))

# Starting threads

thread1.start()

thread2.start()

# Wait for both threads to finish

thread1.join()

thread2.join()

print("All threads finished.")

**How it works:**

* Creates two threads that run independently.
* Uses start() to run them.
* Uses join() to ensure the main program waits for them.

**4. Thread Synchronization (Avoid Race Conditions)**

When multiple threads access shared data, **race conditions** may occur. Use threading.Lock to prevent data corruption.

**Example: Using a Lock**

import threading

counter = 0

lock = threading.Lock()

def increment():

global counter

for \_ in range(100000):

with lock: # Locking critical section

counter += 1

# Create multiple threads

thread1 = threading.Thread(target=increment)

thread2 = threading.Thread(target=increment)

thread1.start()

thread2.start()

thread1.join()

thread2.join()

print("Final Counter:", counter) # Should be 200000

* **with lock** ensures that only one thread modifies counter at a time.

**5. When to Use Multithreading vs. Multiprocessing**

| **Aspect** | **Multithreading** | **Multiprocessing** |
| --- | --- | --- |
| Best for | I/O-bound tasks (e.g., web scraping, file I/O) | CPU-bound tasks (e.g., number crunching) |
| Uses | threading module | multiprocessing module |
| True parallelism? | ❌ No (due to GIL) | ✅ Yes (separate processes) |
| Memory usage | Low (shares memory) | High (separate memory) |

### ****Connecting to an API Using Python****

Python provides the requests module to send HTTP requests and interact with APIs.

## ****1. Installing**** requests

Ensure you have the requests library installed:

pip install requests

## ****2. Making a GET Request****

A **GET request** retrieves data from an API.

import requests

url = "https://jsonplaceholder.typicode.com/posts/1" # Sample API

response = requests.get(url) # Send GET request

if response.status\_code == 200: # Check if successful

data = response.json() # Convert response to JSON

print(data)

else:

print("Error:", response.status\_code)

## ****3. Sending Data with a POST Request****

A **POST request** sends data to an API.

url = "https://jsonplaceholder.typicode.com/posts"

data = {

"title": "Hello API",

"body": "This is a test post.",

"userId": 1

}

response = requests.post(url, json=data)

if response.status\_code == 201: # Created successfully

print("Response:", response.json())

else:

print("Error:", response.status\_code)

## ****4. Adding Headers (Authentication)****

Some APIs require authentication using **API keys**.

headers = {

"Authorization": "Bearer YOUR\_API\_KEY",

"Content-Type": "application/json"

}

response = requests.get("https://api.example.com/data", headers=headers)

print(response.json())

## ****5. Handling API Errors Gracefully****

try:

response = requests.get("https://api.example.com/data", timeout=5)

response.raise\_for\_status() # Raises HTTPError for 4xx/5xx

print(response.json())

except requests.exceptions.RequestException as e:

print("API request failed:", e)

## ****6. Handling Query Parameters****

Some APIs require query parameters (e.g., filtering).

params = {"userId": 1}

response = requests.get("https://jsonplaceholder.typicode.com/posts", params=params)

print(response.json()) # Returns posts from userId=1