PYTHON ULTIMATE NOTES

Contents

7. File Handling

- Reading & Writing Files
- Working with CSV & JSON

8. Exception Handling

Try, Except, Finally
Raising Custom Errors

9. Modules & Libraries

- Importing Modules
- Built-in vs. External Libraries
- Using math, random, datetime

10. Database Connectivity

- Connecting Python with MySQL or SQLite
- CRUD Operations (Create, Read, Update, Delete)

11. Web Scraping

- Using requests and BeautifulSoup
- Extracting Data from a Website

12. Data Science & Machine Learning (Intro)

- Using pandas, numpy, matplotlib
- Basic Data Visualization
- Introduction to scikit-learn

13. Two Mini Projects

- Project 1: To-Do List App (Console-Based)
- Users can add, remove, and mark tasks as completed
- Data stored in a text file or JSON

- Project 2: Weather App Using API
- Fetch weather data from an API (like OpenWeatherMap)
- Display temperature, humidity, and weather conditions
- Simple GUI using tkinter (optional)

1. INTRODUCTION

1. What is Python?

Python is a high-level, interpreted programming language known for its simplicity and readability. It is widely used for web development, data science, automation, AI/ML, game development, cybersecurity, and more.

2. Who Created Python & When?

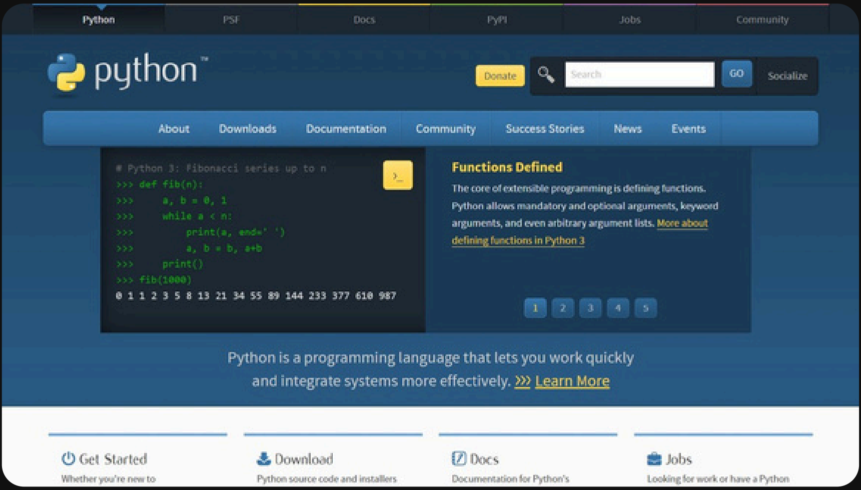
Python was created by Guido van Rossum in 1989 and officially released in 1991.

3. Why Guido van Rossum Create a Python?

Guido wanted to create a simple, easy-to-learn programming language as an alternative to ABC language (which was complex). He named it "Python" after the British comedy show "Monty Python's Flying Circus", not the snake! 🐍

Installation & Setup

First we are going to download python



```
# Python 3: Fibonacci series up to n
>>> def fib(n):
>>>     a, b = 0, 1
>>>     while a < n:
>>>         print(a, end=' ')
>>>         a, b = b, a+b
>>>     print()
>>> fib(1000)
0 1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987
```

Functions Defined

The core of extensible programming is defining functions. Python allows mandatory and optional arguments, keyword arguments, and even arbitrary argument lists. More about defining functions in Python 3

Python is a programming language that lets you work quickly and integrate systems more effectively. >>> Learn More

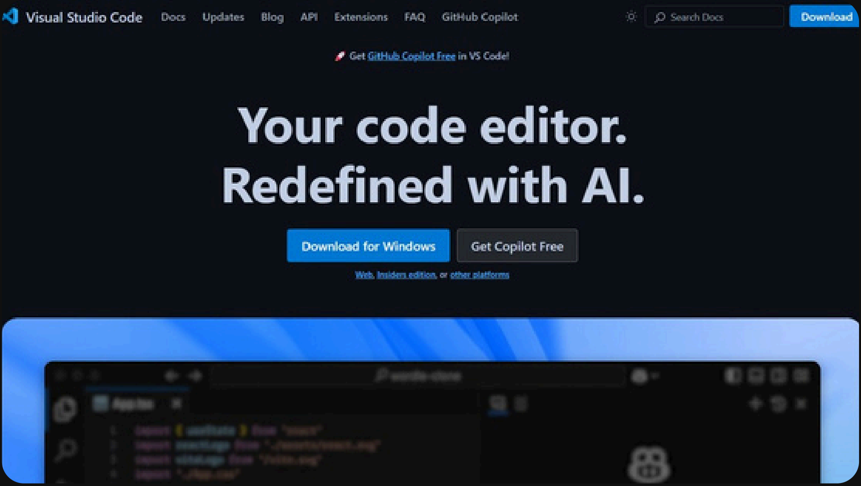
Get Started
Whether you're new to

Download
Python source code and installers

Docs
Documentation for Python's

Jobs
Looking for work or have a Python

Second we are going to download VS Code



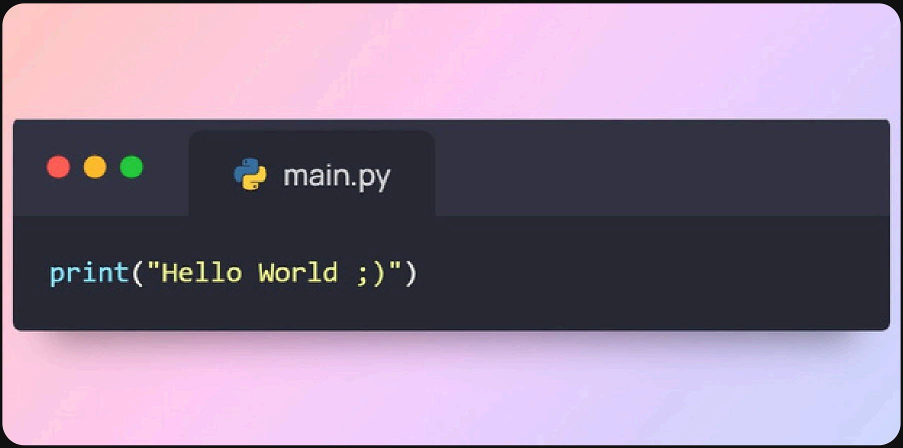
Get GitHub Copilot Free in VS Code!

Your code editor. Redefined with AI.

Web, Insiders edition, or other platforms

Let's write your first program

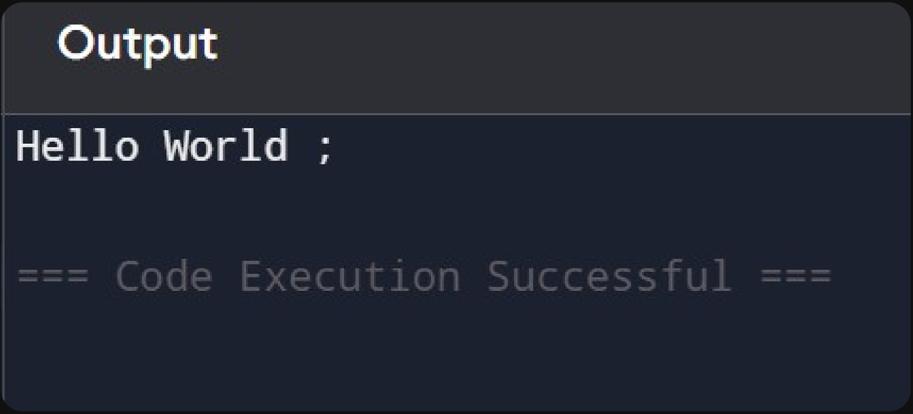
Input:-



```python
print("Hello World ;)")
```

In this program we are just simply printing this hello world program using python

Output:-



```
Hello World ;


=== Code Execution Successful ===
```

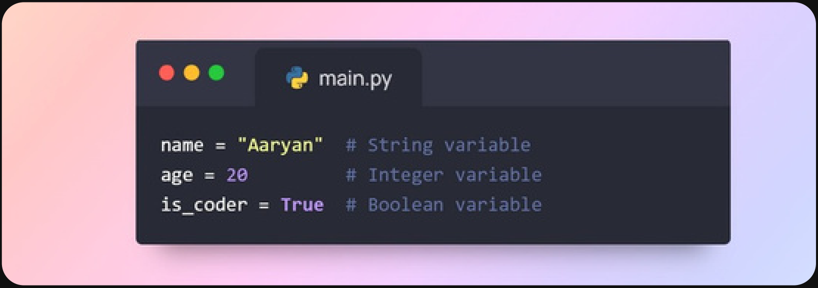
So here you can see Hello World ; is printed successfully

2. PYTHON BASIC

Variables & Data Types

1. Variables

A variable is a named storage location in memory that holds a value, which can change during program execution. Variables allow programmers to store, retrieve, and manipulate data efficiently.



main.py

```python
name = "Aaryan"  # String variable
age = 20         # Integer variable
is_coder = True  # Boolean variable
```

Here, name, age, and is_coder are variables storing different types of data.

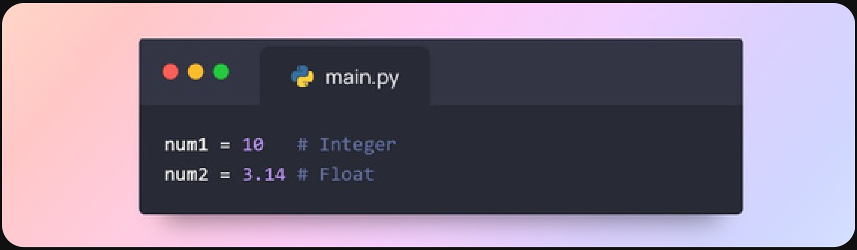
2. Data Types

Data types define the type of data a variable can hold. Different programming languages have different data types, but the common ones include:

- **Numeric Data Type**
- **String**
- **Boolean**
- **List**
- **Tuple**
- **Dictionary**
- **Set**

Numeric Data Type

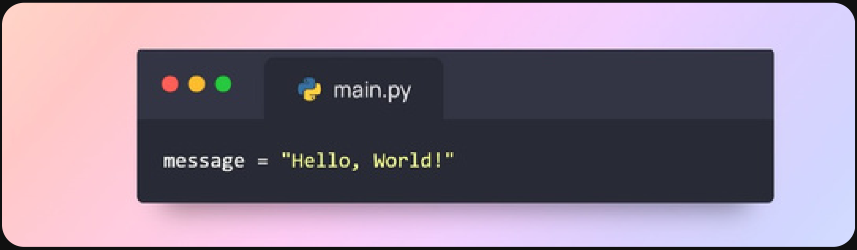
- Integer (int): Whole numbers (e.g., 5, -10, 1000)
- Floating Point (float): Numbers with decimal points (e.g., 3.14, -0.99, 2.0)
- Complex (complex): Numbers with real and imaginary parts (e.g., 3 + 5j in Python)



```python
num1 = 10   # Integer
num2 = 3.14 # Float
```

String

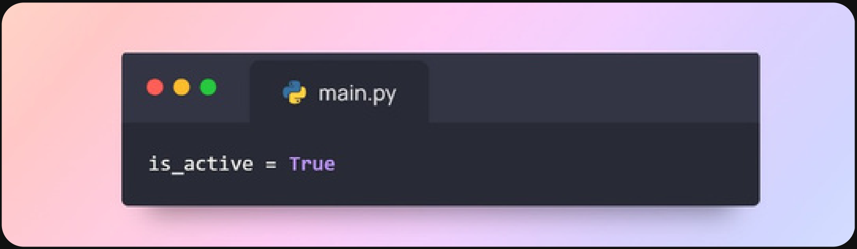
- A sequence of characters enclosed in quotes.



```python
message = "Hello, World!"
```

Boolean (bool)

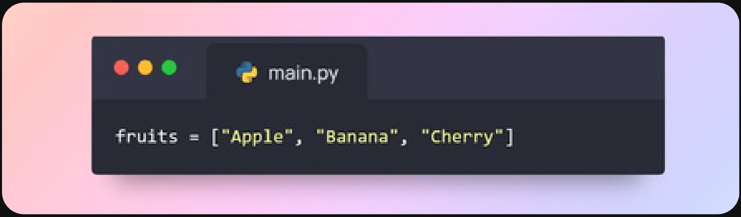
- Represents True or False values.



```python
is_active = True
```

List (Array in JavaScript)

- Ordered, mutable collection of elements.

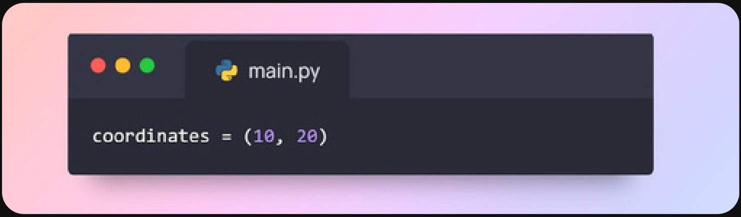


main.py

```python
fruits = ["Apple", "Banana", "Cherry"]
```

Tuple

- Similar to a list but immutable (cannot be changed).

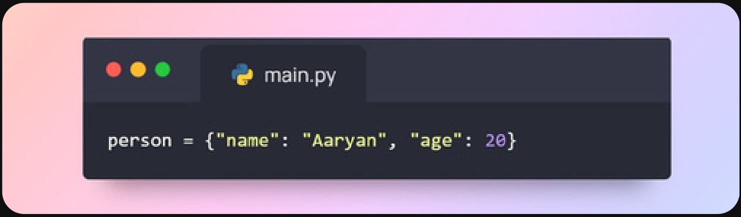


main.py

```python
coordinates = (10, 20)
```

Dictionary (dict)

- Stores key-value pairs.

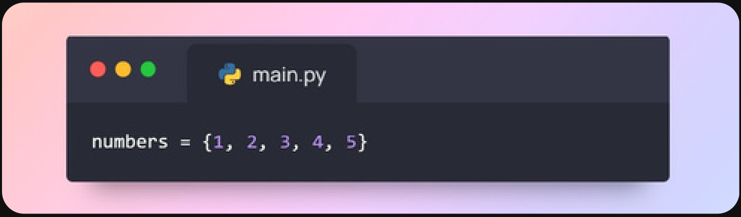


main.py

```python
person = {"name": "Aaryan", "age": 20}
```

Set

- Unordered collection of unique elements.



main.py

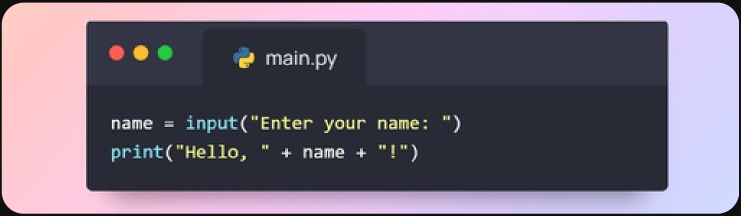
```python
numbers = {1, 2, 3, 4, 5}
```

Input & Output

In Python, input and output operations are performed using built-in functions like input(), print(), and file handling methods.

1. Input in Python

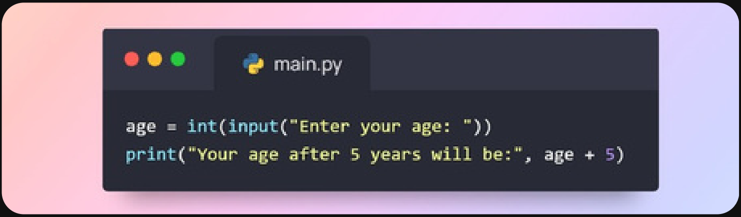
Python provides the input() function to take input from the user.



```python
name = input("Enter your name: ")
print("Hello, " + name + "!")
```

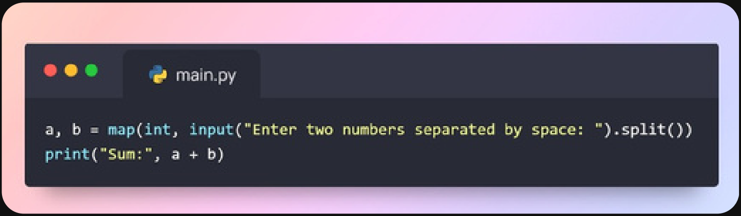
By default, input() takes input as a string.

Example: Taking integer input



```python
age = int(input("Enter your age: "))
print("Your age after 5 years will be:", age + 5)
```

Example: Taking multiple inputs

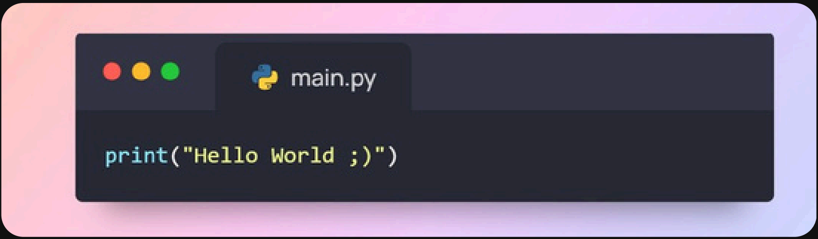


```python
a, b = map(int, input("Enter two numbers separated by space: ").split())
print("Sum:", a + b)
```

2. Output in Python

Python uses the print() function to display output.

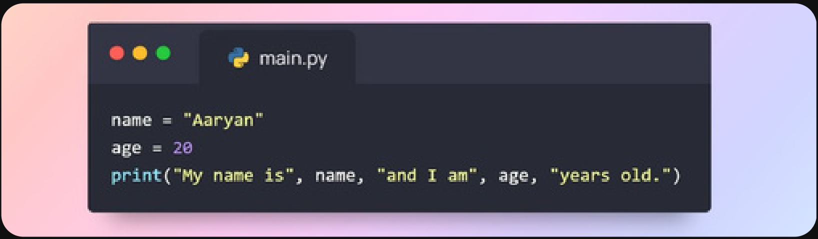
Example: Simple output



main.py

```python
print("Hello World ;)")
```

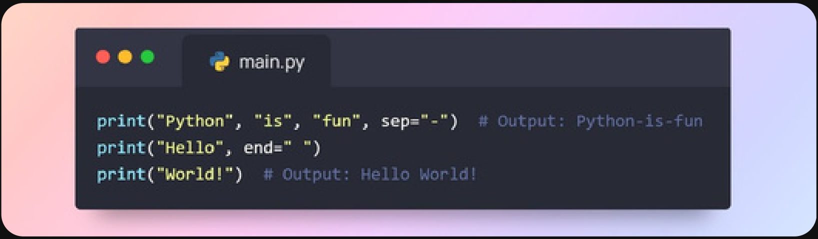
Example: Printing multiple values pytho



main.py

```python
name = "Aaryan"
age = 20
print("My name is", name, "and I am", age, "years old.")
```

Using sep and end parameters



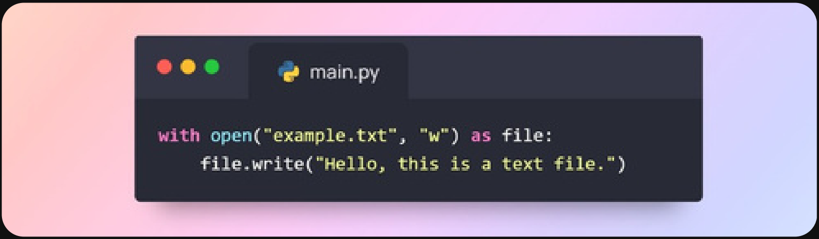
main.py

```python
print("Python", "is", "fun", sep="-")  # Output: Python-is-fun
print("Hello", end=" ")
print("World!")  # Output: Hello World!
```

3. File Input & Output in Python

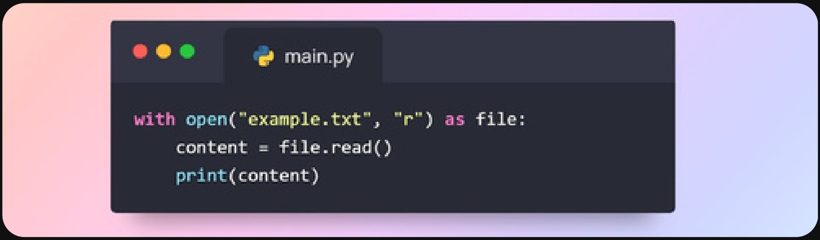
Python allows reading and writing files using open().

Writing to a file



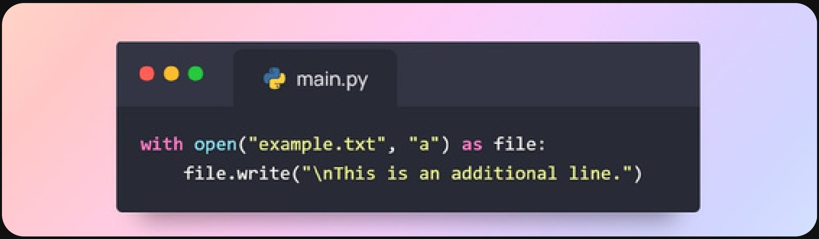
```python
with open("example.txt", "w") as file:
    file.write("Hello, this is a text file.")
```

Reading from a file



```python
with open("example.txt", "r") as file:
    content = file.read()
    print(content)
```

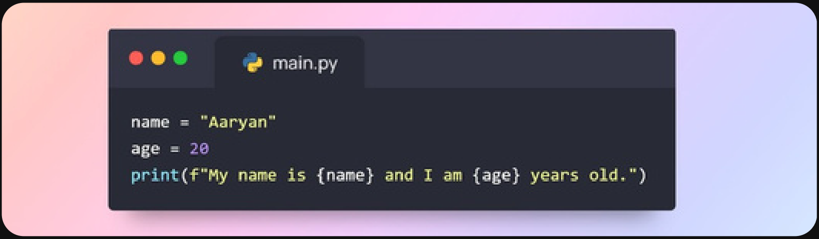
Appending to a file python



```python
with open("example.txt", "a") as file:
    file.write("\nThis is an additional line.")
```

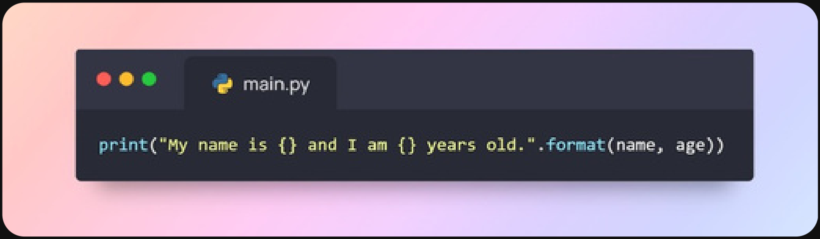
4. Formatted Output

Python provides f-strings for formatted output.



```python
name = "Aaryan"
age = 20
print(f"My name is {name} and I am {age} years old.")
```

Using format() method



```python
print("My name is {} and I am {} years old.".format(name, age))
```

Conclusion

✅ Use input() to get user input.

✅ Use print() to display output.

✅ Use open() for file handling.

✅ Use f-strings for formatted output.

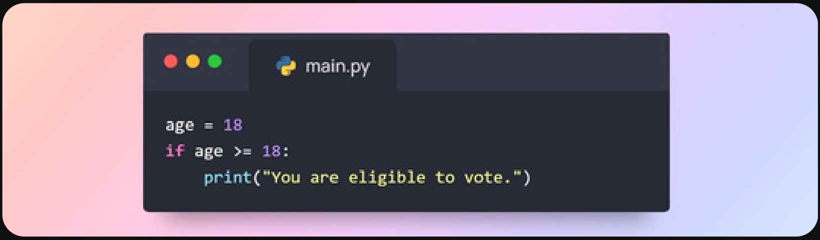
3. Control Flow

1.Conditional Statements (if, elif, else)

Conditional statements allow a program to make decisions based on conditions.

if Statement

Executes a block of code if a condition is True.



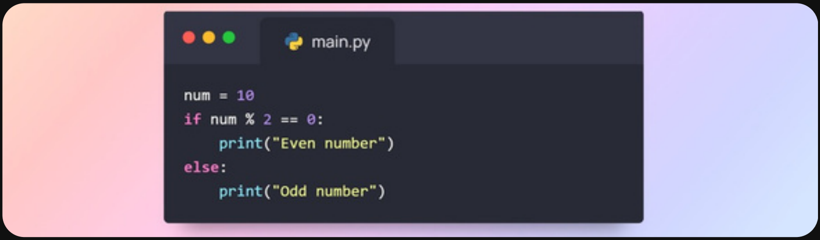
main.py

```python
age = 18
if age >= 18:
    print("You are eligible to vote.")
```

➡ If age is 18 or more, the message is printed.

if-else Statement

Executes one block of code if True, another if False.



main.py

```python
num = 10
if num % 2 == 0:
    print("Even number")
else:
    print("Odd number")
```

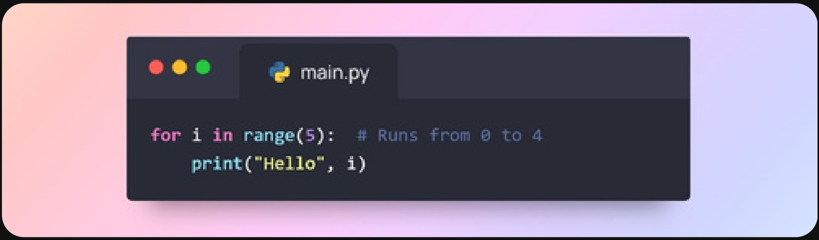
➡ If num is divisible by 2, it prints "Even number"; otherwise, "Odd number".

Loops in Python

Loops help execute a block of code multiple times.

👉 for Loop

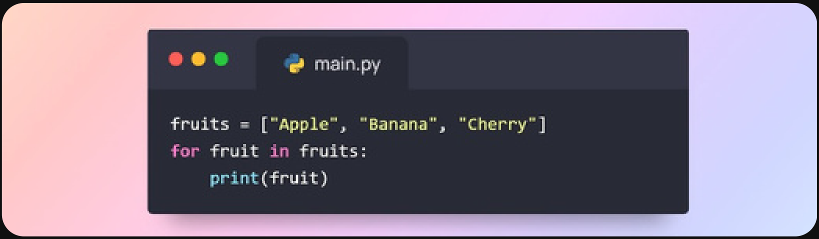
Used to iterate over a sequence (list, string, range, etc.)



```python
for i in range(5):  # Runs from 0 to 4
    print("Hello", i)
```

➡ **range(5) generates numbers from 0 to 4.**

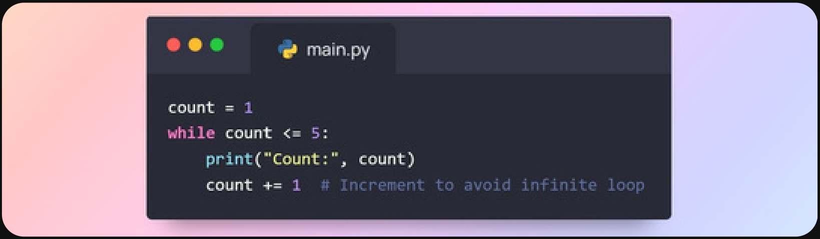
Looping through a list:



```python
fruits = ["Apple", "Banana", "Cherry"]
for fruit in fruits:
    print(fruit)
```

👉 while Loop

Repeats as long as the condition is True.



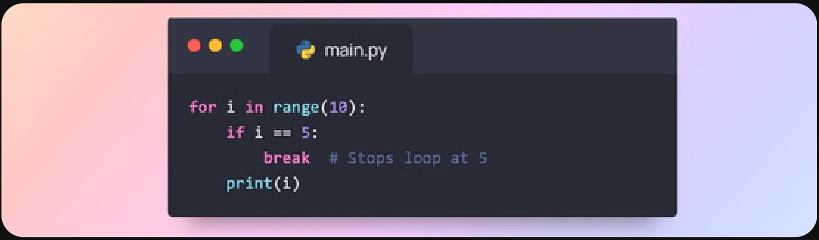
```python
count = 1
while count <= 5:
    print("Count:", count)
    count += 1  # Increment to avoid infinite loop
```

➡ **Runs while count is ≤ 5.**

Break, Continue, Pass

👉 break Statement

Exits the loop immediately.



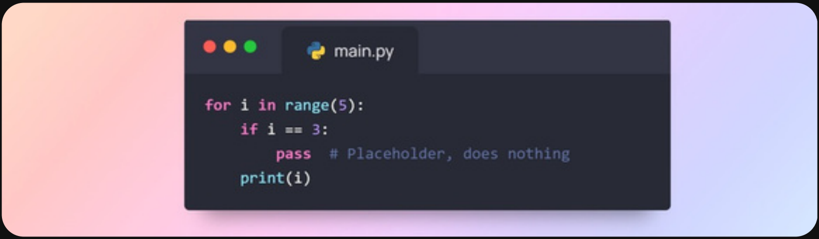
main.py

```
for i in range(10):
    if i == 5:
        break  # Stops loop at 5
    print(i)
```

➡ The loop stops when i reaches 5.

👉 continue Statement

Skips the current iteration and moves to the next.



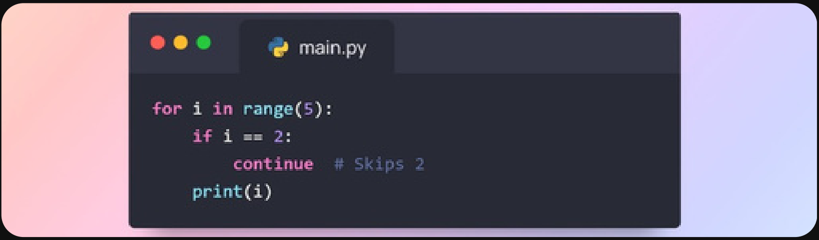
main.py

```
for i in range(5):
    if i == 3:
        pass  # Placeholder, does nothing
    print(i)
```

➡ The number 2 is skipped in the output.

👉 pass Statement

A placeholder that does nothing (used when a block is required but not written yet).



main.py

```
for i in range(5):
    if i == 2:
        continue  # Skips 2
    print(i)
```

➡ pass is used when a statement is required but not implemented.

Conclusion

✅ if-elif-else → Decision making
✅ for loop → Iterates over a sequence
✅ while loop → Runs while the condition is True
✅ break → Exits loop
✅ continue → Skips current iteration
✅ pass → Does nothing (placeholder)

4. Functions

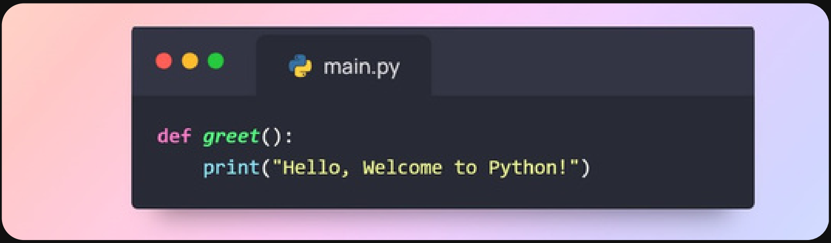
Functions in Python

Functions are reusable blocks of code that perform a specific task. They help make programs modular and reduce repetition.

Defining & Calling Functions

A function is defined using the def keyword.

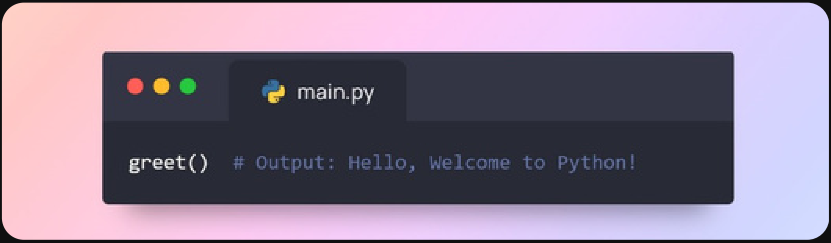
👉 Defining a Function



main.py

```python
def greet():
    print("Hello, Welcome to Python!")
```

👉 Calling a Function



main.py

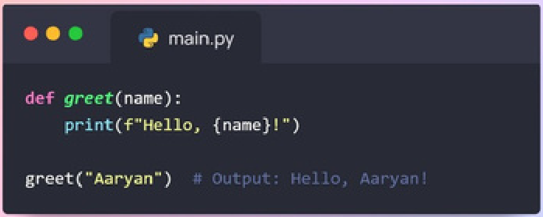
```python
greet()  # Output: Hello, Welcome to Python!
```

➡ The function is executed when called.

Arguments & Return Values

👉 Function with Parameters

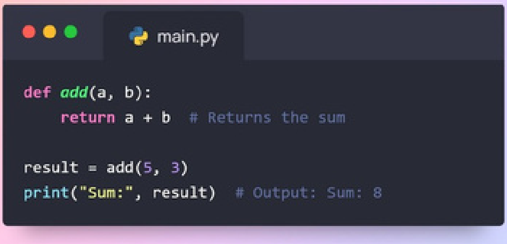
A function can take inputs called parameters.



main.py

```python
def greet(name):
    print(f"Hello, {name}!")

greet("Aaryan")  # Output: Hello, Aaryan!
```

👉 Function with Multiple Parameters



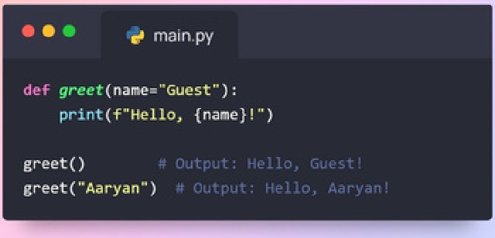
main.py

```python
def add(a, b):
    return a + b  # Returns the sum

result = add(5, 3)
print("Sum:", result)  # Output: Sum: 8
```

➡ return sends a value back to the caller.

👉 Default Parameter Value

If no argument is passed, it uses the default value python

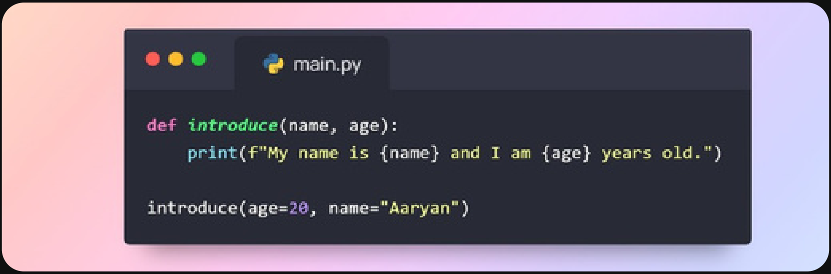


main.py

```python
def greet(name="Guest"):
    print(f"Hello, {name}!")

greet()         # Output: Hello, Guest!
greet("Aaryan")  # Output: Hello, Aaryan!
```

👉 Keyword Arguments (Named Parameters)

You can pass arguments by name, making them more readable.

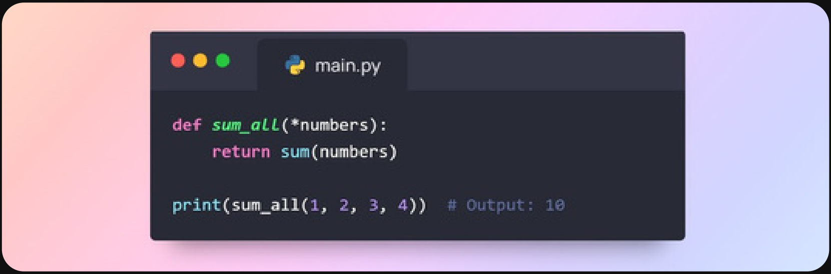


```python
def introduce(name, age):
    print(f"My name is {name} and I am {age} years old.")

introduce(age=20, name="Aaryan")
```

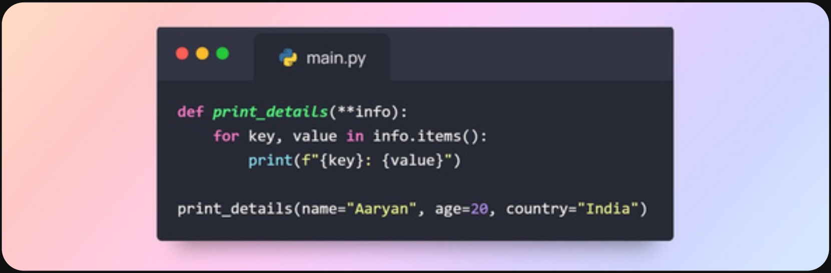
➡ Order doesn't matter when using keyword arguments.

👉 Variable-Length Arguments (*args and **kwargs)

*args → Allows multiple positional arguments.
**kwargs → Allows multiple keyword arguments.



```python
def sum_all(*numbers):
    return sum(numbers)

print(sum_all(1, 2, 3, 4))  # Output: 10
```



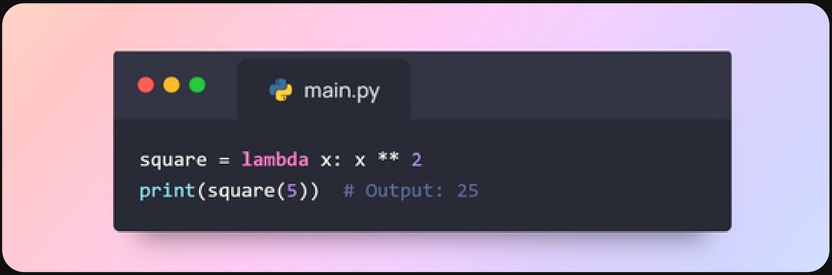
```python
def print_details(**info):
    for key, value in info.items():
        print(f"{key}: {value}")

print_details(name="Aaryan", age=20, country="India")
```

➡ *args collects multiple values as a tuple.
➡ **kwargs collects key-value pairs as a dictionary.

Lambda Functions (Anonymous Functions)

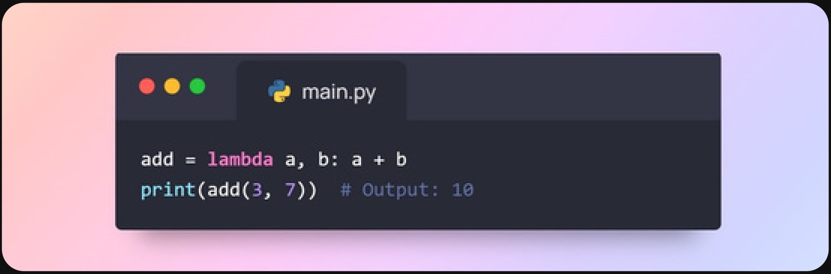
A lambda function is a small anonymous function with a single expression.

👉 Example: Lambda Function



```python
square = lambda x: x ** 2
print(square(5))  # Output: 25
```

👉 Example: Lambda with Multiple Arguments



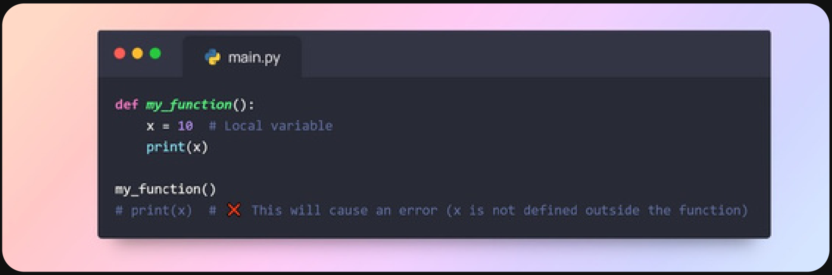
```python
add = lambda a, b: a + b
print(add(3, 7))  # Output: 10
```

➡ **Lambda functions are useful for short, single-use operations.**

Scope of Variables (Local & Global)

Scope determines where a variable can be accessed

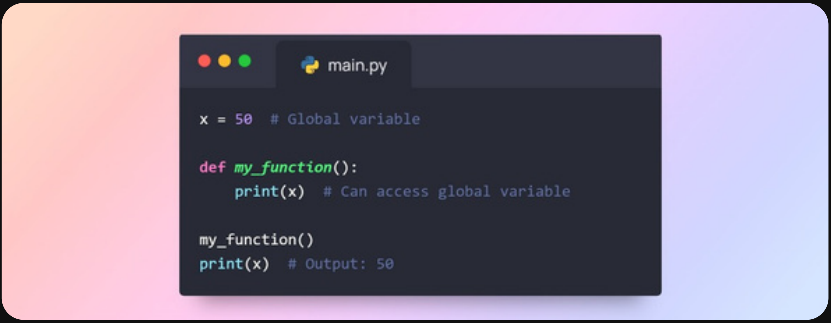
👉 Local Variable (Exists only inside a function)



```python
def my_function():
    x = 10  # Local variable
    print(x)

my_function()
# print(x)  # ❌ This will cause an error (x is not defined outside the function)
```

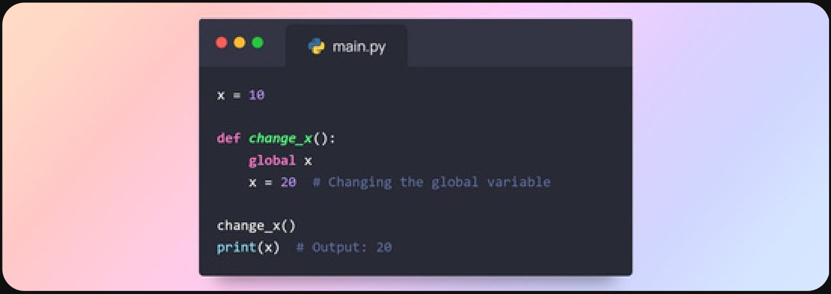
➡ **x is inside the function and can't be accessed outside.**

👉 Global Variable (Accessible everywhere)



```python
x = 50  # Global variable

def my_function():
    print(x)  # Can access global variable

my_function()
print(x)  # Output: 50
```

👉 Modifying Global Variables inside Functions
Use the global keyword if you need to modify a global variable inside a function.



```python
x = 10

def change_x():
    global x
    x = 20  # Changing the global variable

change_x()
print(x)  # Output: 20
```

➡ Without global, Python will create a new local variable instead of modifying the global one.

Conclusion

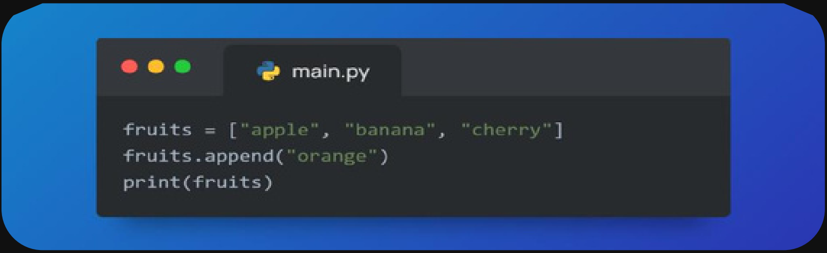
✅ Functions → Reusable code blocks
✅ Arguments → Inputs to functions
✅ return → Sends a value back
✅ Lambda → Short anonymous functions
✅ Scope → Defines where a variable is accessible

5. Data Structures

Lists

Ordered, mutable collection of items.

Code Example:

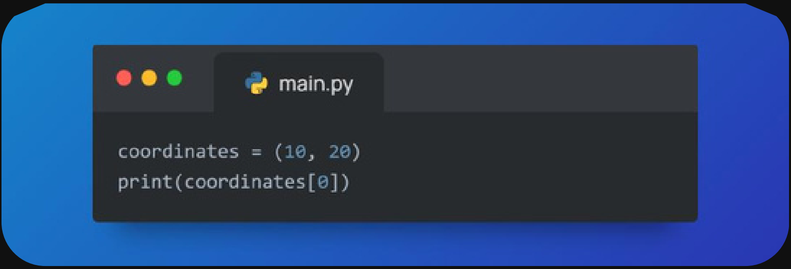


```
fruits = ["apple", "banana", "cherry"]
fruits.append("orange")
print(fruits)
```

Tuples

Ordered, immutable collection of items.

Code Example:

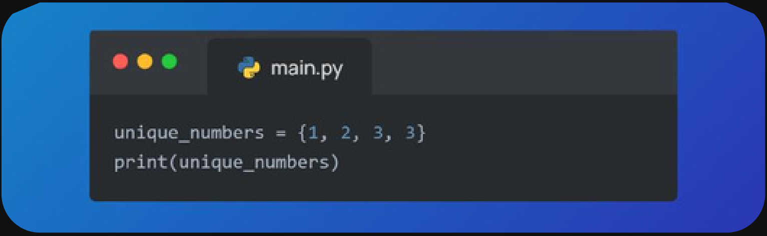


```
coordinates = (10, 20)
print(coordinates[0])
```

Sets

Unordered collection of unique items.

Code Example:

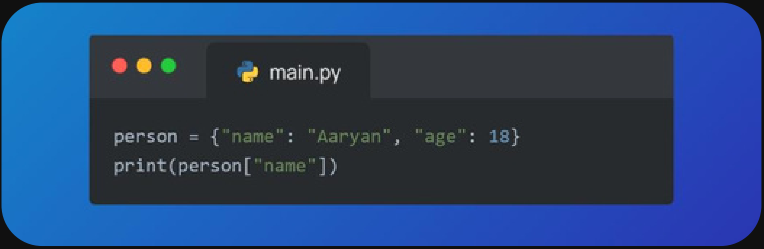


```
unique_numbers = {1, 2, 3, 3}
print(unique_numbers)
```

Dictionaries

Key-value pairs.

Code Example:



```
person = {"name": "Aaryan", "age": 18}
print(person["name"])
```

Conclusion

Data structures like lists, tuples, sets, and dictionaries are essential for storing and manipulating data.
Practice Questions

Create a list of numbers and find the sum of all elements.

Write a program to count the frequency of each character in a string using a dictionary.

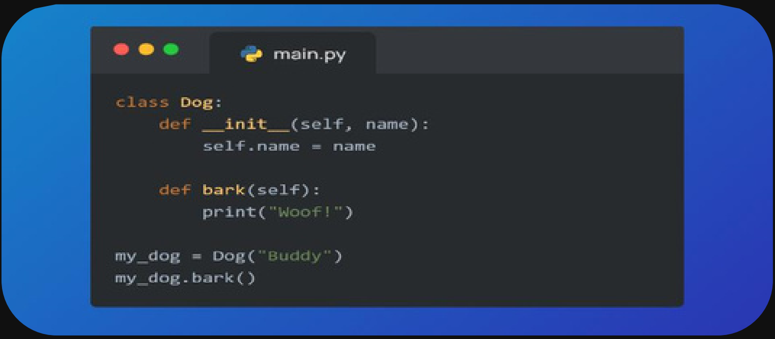
6. Object-Oriented Programming (OOP)

Classes & Objects

Class: Blueprint for creating objects.

Object: Instance of a class.

Code Example:

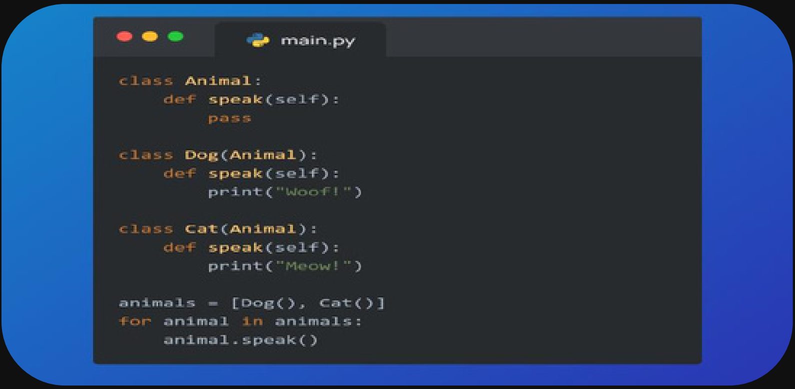


```python
class Dog:
    def __init__(self, name):
        self.name = name

    def bark(self):
        print("Woof!")

my_dog = Dog("Buddy")
my_dog.bark()
```

Inheritance & Polymorphism

Inheritance: A class can inherit attributes and methods from another class.
Polymorphism: Methods can behave differently based on the object.

Code Example:



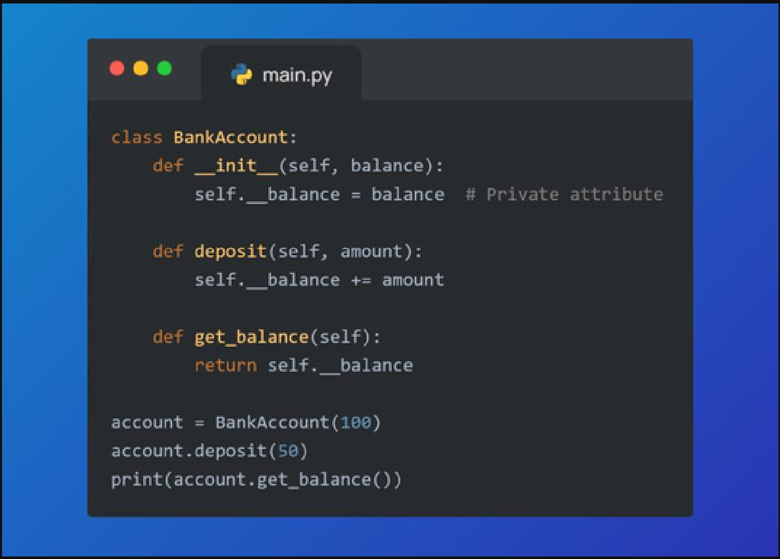
```python
class Animal:
    def speak(self):
        pass

class Dog(Animal):
    def speak(self):
        print("Woof!")

class Cat(Animal):
    def speak(self):
        print("Meow!")

animals = [Dog(), Cat()]
for animal in animals:
    animal.speak()
```

Encapsulation & Abstraction

Encapsulation: Restricting access to certain components.

Abstraction: Hiding complex implementation details.

Code Example:



main.py

```python
class BankAccount:
    def __init__(self, balance):
        self.__balance = balance  # Private attribute

    def deposit(self, amount):
        self.__balance += amount

    def get_balance(self):
        return self.__balance

account = BankAccount(100)
account.deposit(50)
print(account.get_balance())
```

Conclusion

OOP helps in organizing code by modeling real-world entities.

Practice Questions

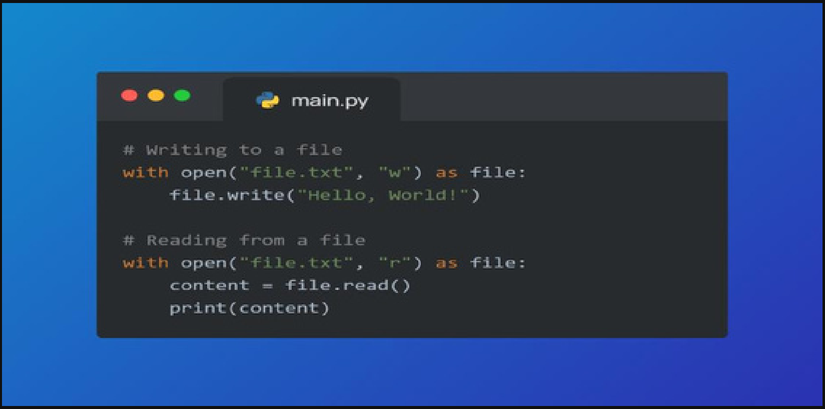
Create a class Car with attributes brand and model. Add a method to display the car details.
Implement inheritance with a base class Shape and derived classes Circle and Rectangle.

7. File Handling

Reading & Writing Files

Use open() to read or write files.

Code Example:

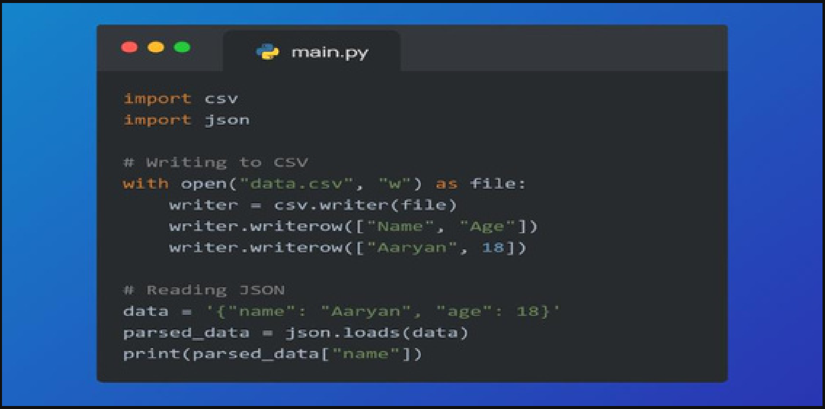


```python
# Writing to a file
with open("file.txt", "w") as file:
    file.write("Hello, World!")

# Reading from a file
with open("file.txt", "r") as file:
    content = file.read()
    print(content)
```

Working with CSV & JSON

Use csv and json modules.

Code Example:



```python
import csv
import json

# Writing to CSV
with open("data.csv", "w") as file:
    writer = csv.writer(file)
    writer.writerow(["Name", "Age"])
    writer.writerow(["Aaryan", 18])

# Reading JSON
data = '{"name": "Aaryan", "age": 18}'
parsed_data = json.loads(data)
print(parsed_data["name"])
```

File handling is essential for reading and writing data to files.

Practice Questions

Write a program to read a text file and count the number of words.

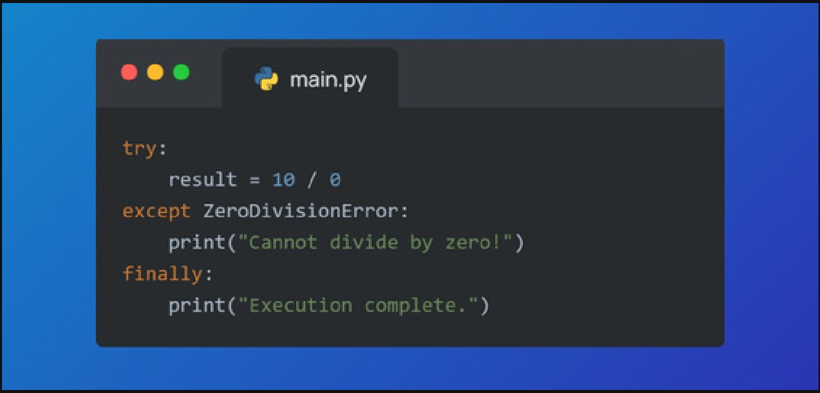
Create a CSV file with student data and read it using Python.

8. Exception Handling

Try, Except, Finally

Handle errors using try, except, and finally.

Code Example:

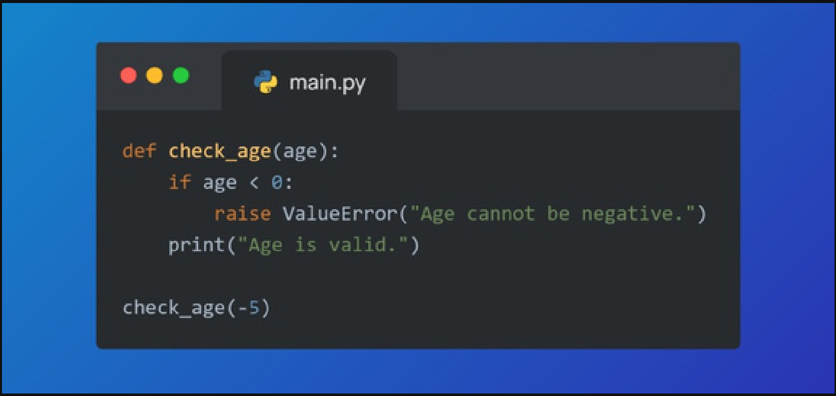


```python
try:
    result = 10 / 0
except ZeroDivisionError:
    print("Cannot divide by zero!")
finally:
    print("Execution complete.")
```

Raising Custom Errors

Use raise to throw exceptions.

Code Example:



```python
def check_age(age):
    if age < 0:
        raise ValueError("Age cannot be negative.")
    print("Age is valid.")

check_age(-5)
```

Conclusion

Exception handling ensures that your program can handle errors gracefully.

Practice Questions

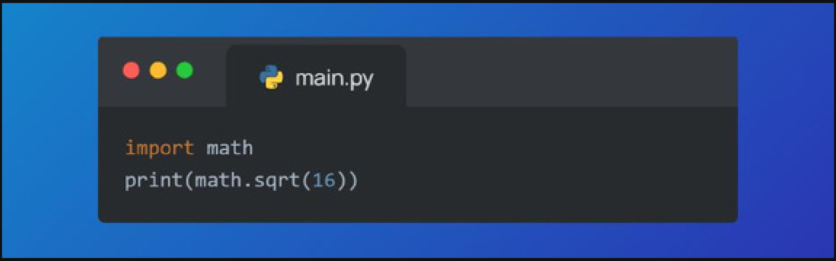
Write a program to handle a FileNotFoundError.

Create a custom exception for invalid email formats.

9. Modules & Libraries

Importing Modules

Use import to include modules.



```python
import math
print(math.sqrt(16))
```

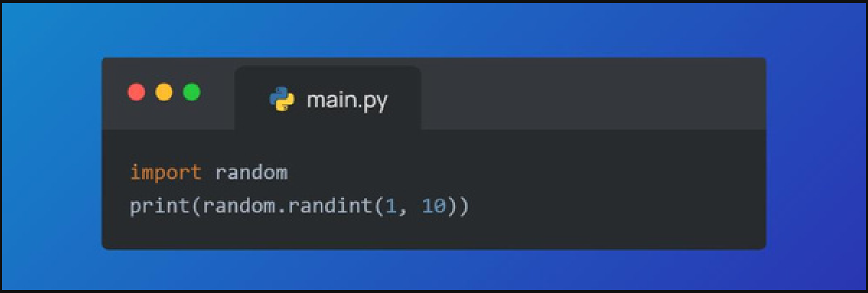
Code Example:

Built-in vs. External Libraries

Built-in: math, random, datetime.

External: Install using pip (e.g., numpy, pandas).

Code Example:



```python
import random
print(random.randint(1, 10))
```

Conclusion

Modules and libraries extend Python's functionality.

Practice Questions

Write a program to generate a random password using the random module.

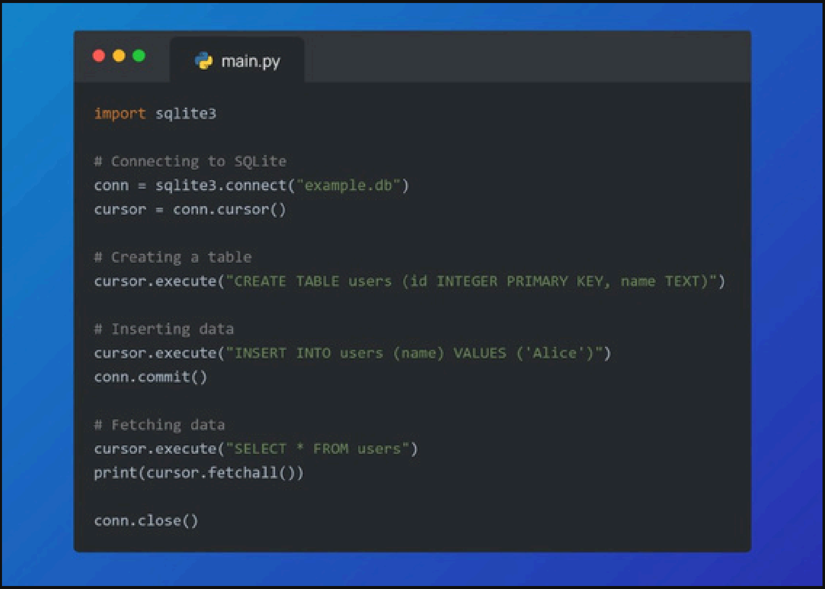
Use the datetime module to display the current date and time.

10. Database Connectivity

Connecting Python with MySQL or SQLite

Use mysql-connector or sqlite3.

Code Example:



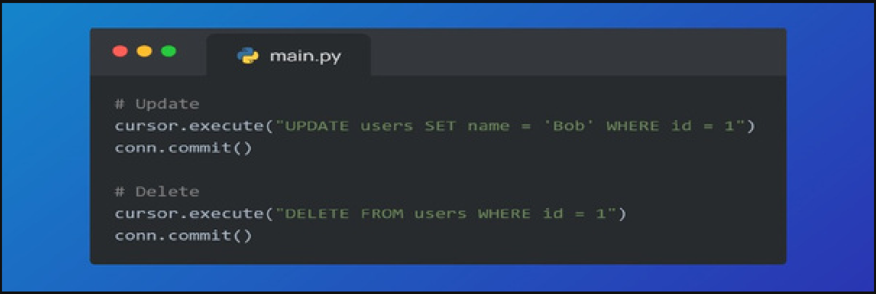
main.py

```python
import sqlite3

# Connecting to SQLite
conn = sqlite3.connect("example.db")
cursor = conn.cursor()

# Creating a table
cursor.execute("CREATE TABLE users (id INTEGER PRIMARY KEY, name TEXT)")

# Inserting data
cursor.execute("INSERT INTO users (name) VALUES ('Alice')")
conn.commit()

# Fetching data
cursor.execute("SELECT * FROM users")
print(cursor.fetchall())

conn.close()
```

CRUD Operations

Create, Read, Update, Delete.

Code Example:



main.py

```python
# Update
cursor.execute("UPDATE users SET name = 'Bob' WHERE id = 1")
conn.commit()

# Delete
cursor.execute("DELETE FROM users WHERE id = 1")
conn.commit()
```

Conclusion

Python can interact with databases to perform CRUD operations.

Practice Questions

Create a SQLite database and perform CRUD operations on a students table.

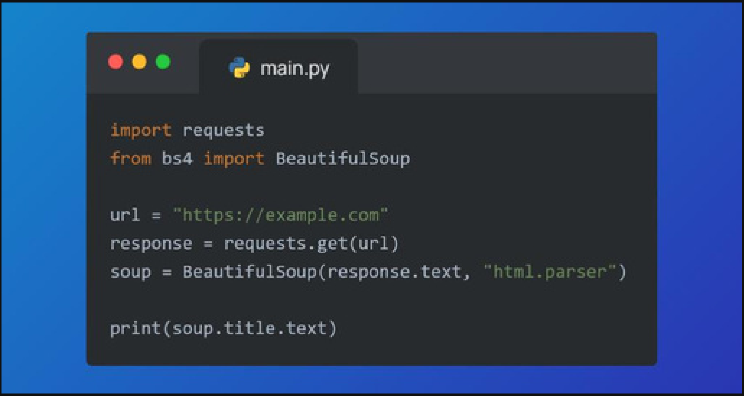
Connect Python to MySQL and fetch data from a table.

11. Web Scraping

Using requests and BeautifulSoup

Extract data from websites.

Code Example:



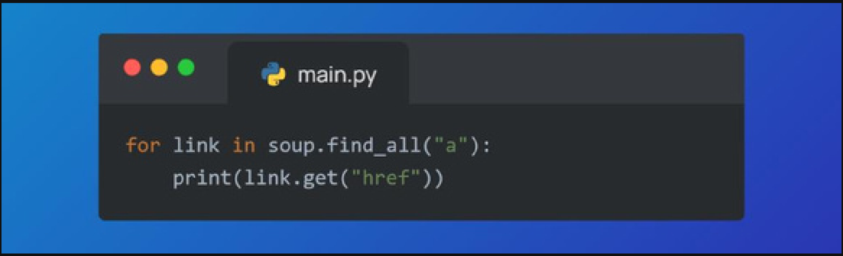
main.py

```python
import requests
from bs4 import BeautifulSoup

url = "https://example.com"
response = requests.get(url)
soup = BeautifulSoup(response.text, "html.parser")

print(soup.title.text)
```

Extracting Data from a Website

Use tags and attributes to extract specific data.

Code Example:



main.py

```python
for link in soup.find_all("a"):
    print(link.get("href"))
```

Conclusion

Web scraping allows you to extract data from websites for analysis.

Practice Questions

Scrape the titles of all articles from a news website.

Extract all image URLs from a webpage.

12. Data Science & Machine Learning (Intro)

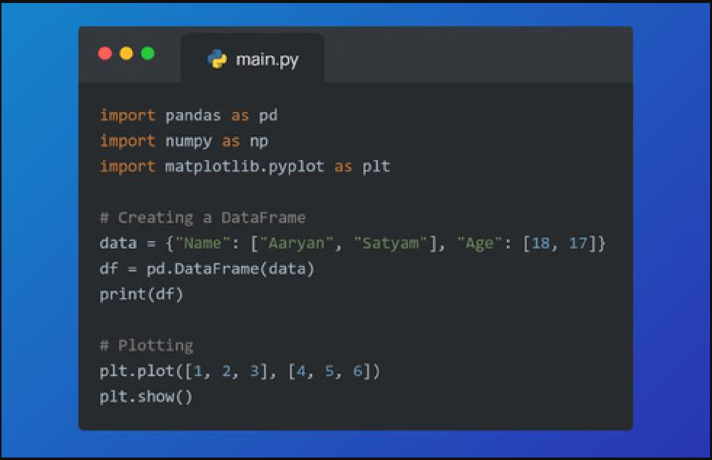
Using pandas, numpy, matplotlib

pandas: Data manipulation.

numpy: Numerical computations.

matplotlib: Data visualization.

Code Example:

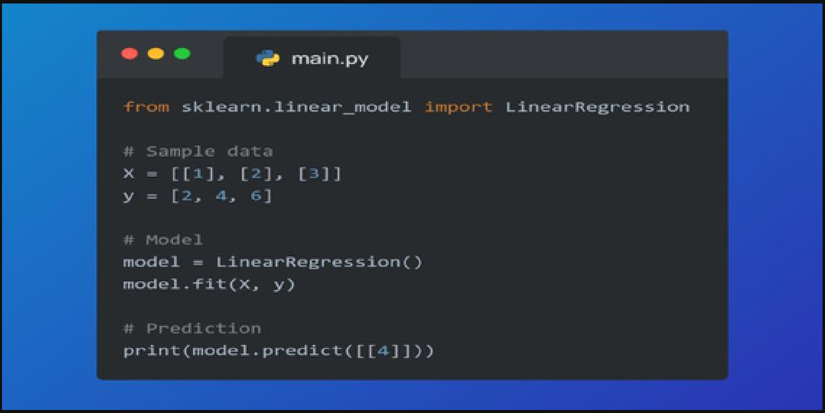


```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

# Creating a DataFrame
data = {"Name": ["Aaryan", "Satyam"], "Age": [18, 17]}
df = pd.DataFrame(data)
print(df)

# Plotting
plt.plot([1, 2, 3], [4, 5, 6])
plt.show()
```

Introduction to scikit-learn

Machine learning library.

Code Example:



```python
from sklearn.linear_model import LinearRegression

# Sample data
X = [[1], [2], [3]]
y = [2, 4, 6]

# Model
model = LinearRegression()
model.fit(X, y)

# Prediction
print(model.predict([[4]]))
```

Conclusion

Python is a powerful tool for data science and machine learning.

Practice Questions

Create a DataFrame and perform basic operations like filtering and sorting.
Train a simple linear regression model using scikit-learn.

Final Conclusion

By completing this course, you have gained a solid foundation in Python programming. You can now build applications, analyze data, and even dive into machine learning. Keep practicing and exploring Python's vast ecosystem!

13 Let’s Mini Projects

Mini Project 1: To-Do List Application

Description

Create a simple **To-Do List Application** using Python. This project will help you practice working with **lists, functions, file handling, and user input**.

Features

1. Add a new task to the to-do list.
2. View all tasks in the list.
3. Mark a task as completed.
4. Delete a task from the list.
5. Save the to-do list to a file and load it when the program starts.

Mini Project 2: Weather App Using Web Scraping

Description

Create a Weather App that fetches the current weather information for a city using web scraping. This project will help you practice web scraping, working with libraries, and handling user input.

Features

1. **Fetch the current temperature and weather condition for a city.**
2. **Display the weather information in a user-friendly format.**

3. **Handle errors (e.g., invalid city name).**

Thank You