Python

Prof Gautam sngh

What is Python?

Python is a **high-level, interpreted, object-oriented** programming language.
It focuses on **code readability** and **developer productivity**.

Example

```
print("Hello, Python")
```

What are the features of Python?

Easy to learn

Interpreted

Platform independent

Object-oriented

Large library support

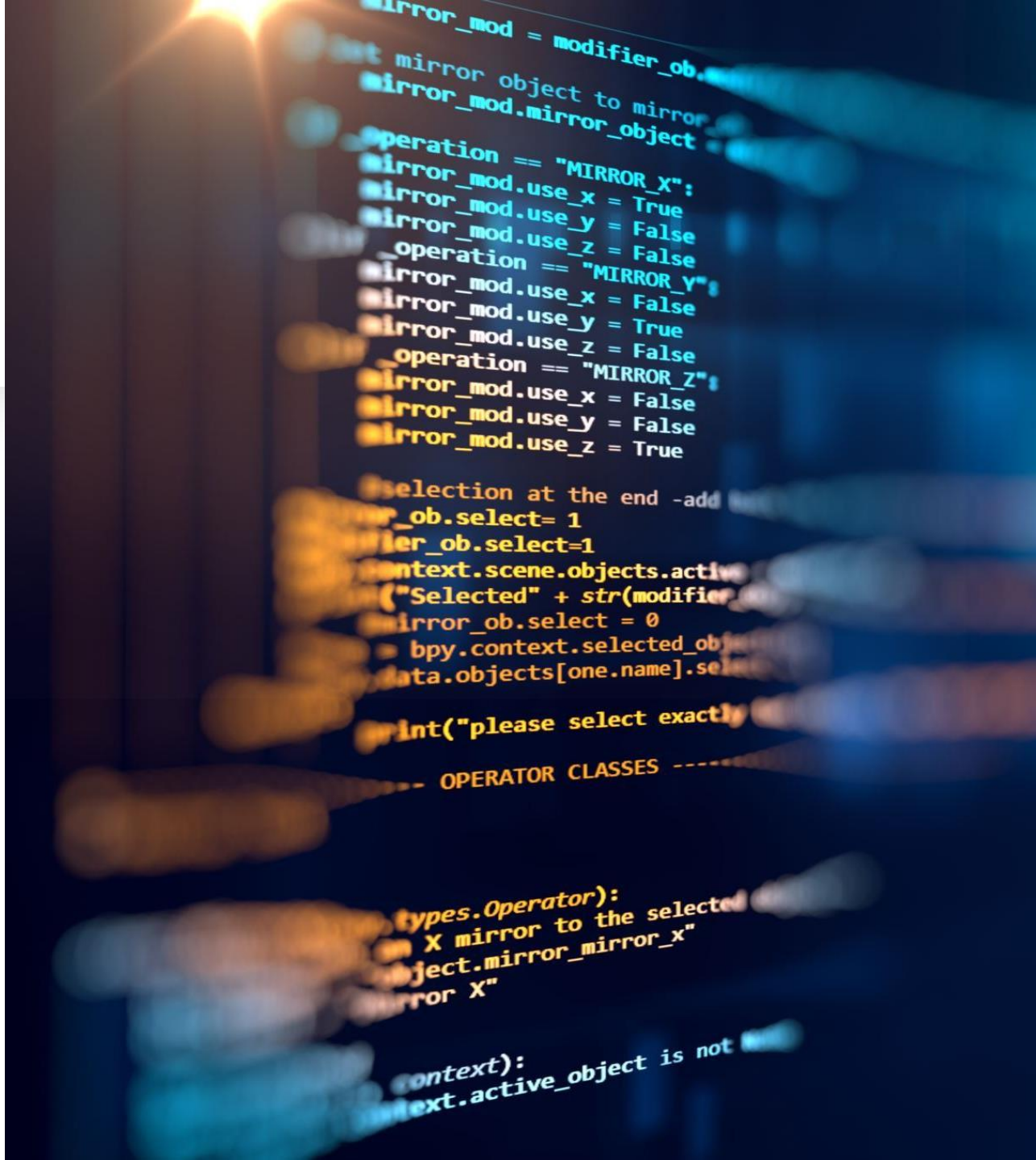
Example

import math

print(math.sqrt(16))

What is an interpreted language?

- Python executes code **line by line**, not compiled all at once.
- Example
- `x = 10`
- `print(x)`



What are Python keywords?

Reserved words that **cannot be used as variable names.**

Example

```
if True:
    print("Hello")
```

What is dynamic typing?

Variable type is decided **at runtime**, not declared explicitly.

Example

```
x = 10
x = "Python"
print(x)
```

What are mutable and immutable objects?

- Explanation
- Mutable → Can be changed
- Immutable → Cannot be changed
- Example
- `# Mutable`
- `list1 = [1, 2, 3]`
- `list1[0] = 10`
-
- `# Immutable`
- `x = 5`
- `# x[0] = 3` ❌

Difference between List and Tuple?

Explanation

List is mutable, tuple is immutable.

Example

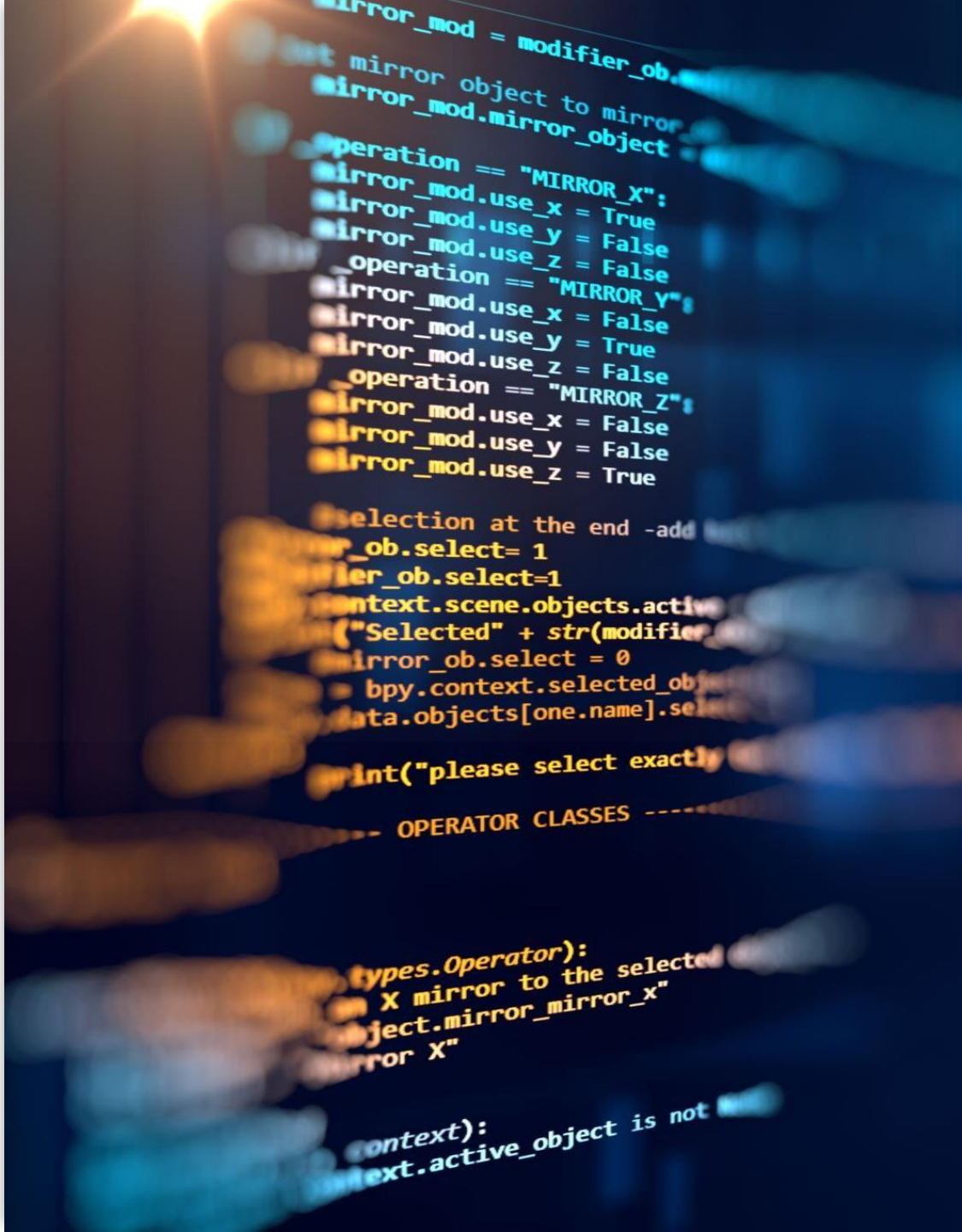
```
list1 = [1, 2]
tuple1 = (1, 2)
list1[0] = 10   # allowed
```

What is a Dictionary?

- Explanation
- Stores data in **key-value pairs**.
- Example
- `student = {`
- `    "name": "Rahul",`
- `    "age": 21`
- `}`
- `print(student["name"])`

What is a function?

- Explanation
- Reusable block of code that performs a task.
- Example
- `def add(a, b):`
- `    return a + b`
-
- `print(add(2, 3))`



What is *args?

Explanation

Used to pass **multiple positional arguments**.

Example

```
def total(*args):
    return sum(args)

print(total(1, 2, 3))
```

What is **kwargs?

Explanation

Used to pass **multiple keyword arguments**.

Example

```
def info(**kwargs):
    print(kwargs)

info(name="Amit", age=22)
```

What is a Lambda Function?

Explanation

Anonymous function written in **one line**.

Example

```
square = lambda x: x * x
print(square(5))
```

What is List Comprehension?

Explanation

Short syntax to create lists.

Example

```
squares = [x*x for x in range(1, 6)]
print(squares)
```

Difference between `break`, `continue`, and `pass`?

- Explanation
- `break`: exits loop
- `continue`: skips iteration
- `pass`: placeholder
- Example
- `for i in range(5):`
- `    if i == 2:`
- `        continue`
- `    print(i)`

`break` Statement

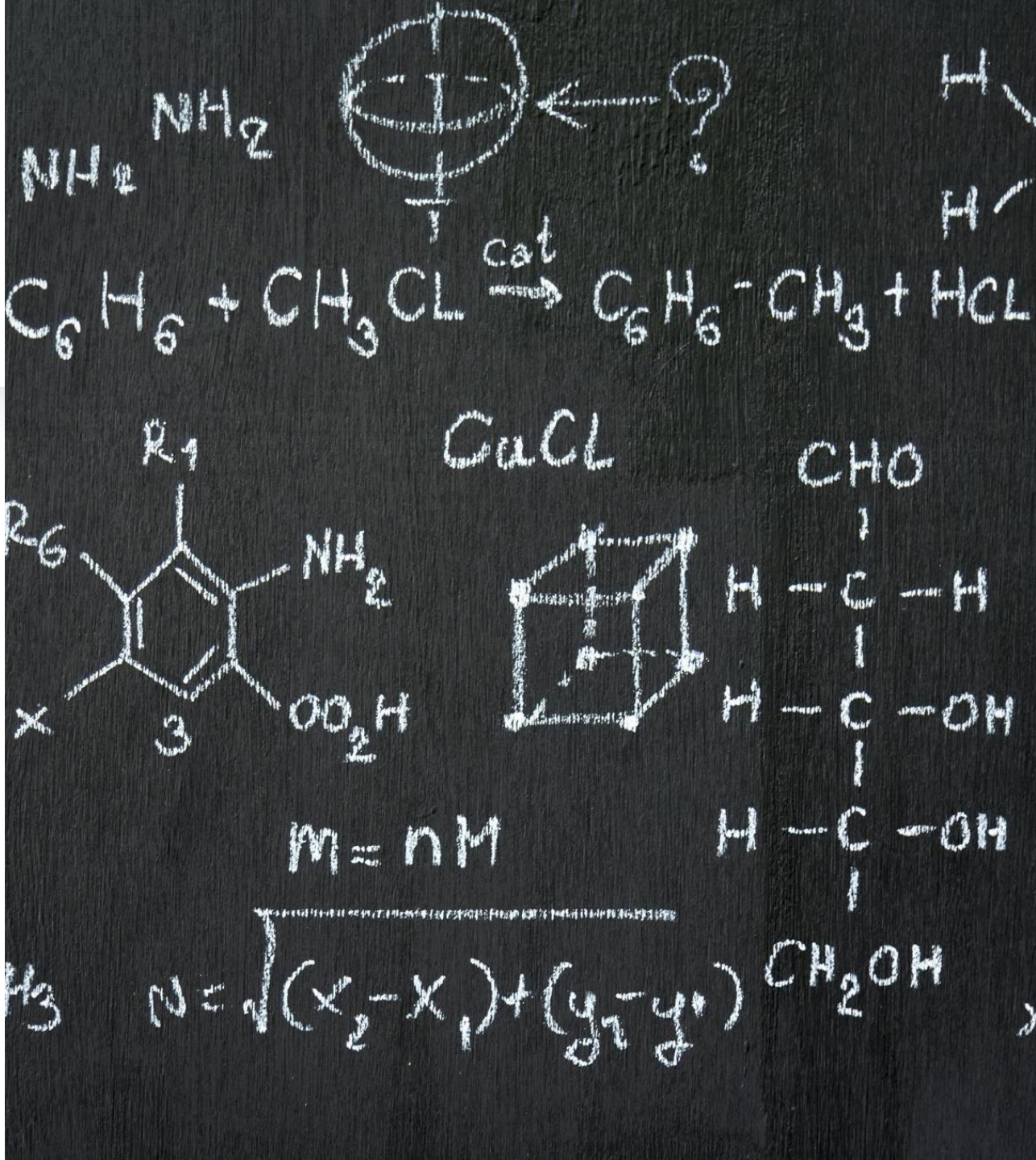
- The **`break`** statement is used to **terminate (exit) a loop completely**, even if the loop condition is still true.
- for i in range(1, 6):

```
 if i == 3:
     break
print(i)
```

pass Statement

- Explanation
- The **pass** statement does **nothing**.
  It is used as a **placeholder** where a statement is syntactically required but no action is needed.
- 👉 Prevents **syntax error**.

- Example 1: Inside Loop
- `for i in range(1, 4):`
- `    if i == 2:`
- `        pass`
- `    print(i)`



Key Differences (Important for Interview)

Statement	Purpose	Effect on Loop
`break`	Exit loop	Loop stops completely
`continue`	Skip iteration	Loop continues
`pass`	Do nothing	No effect

What is Exception Handling?

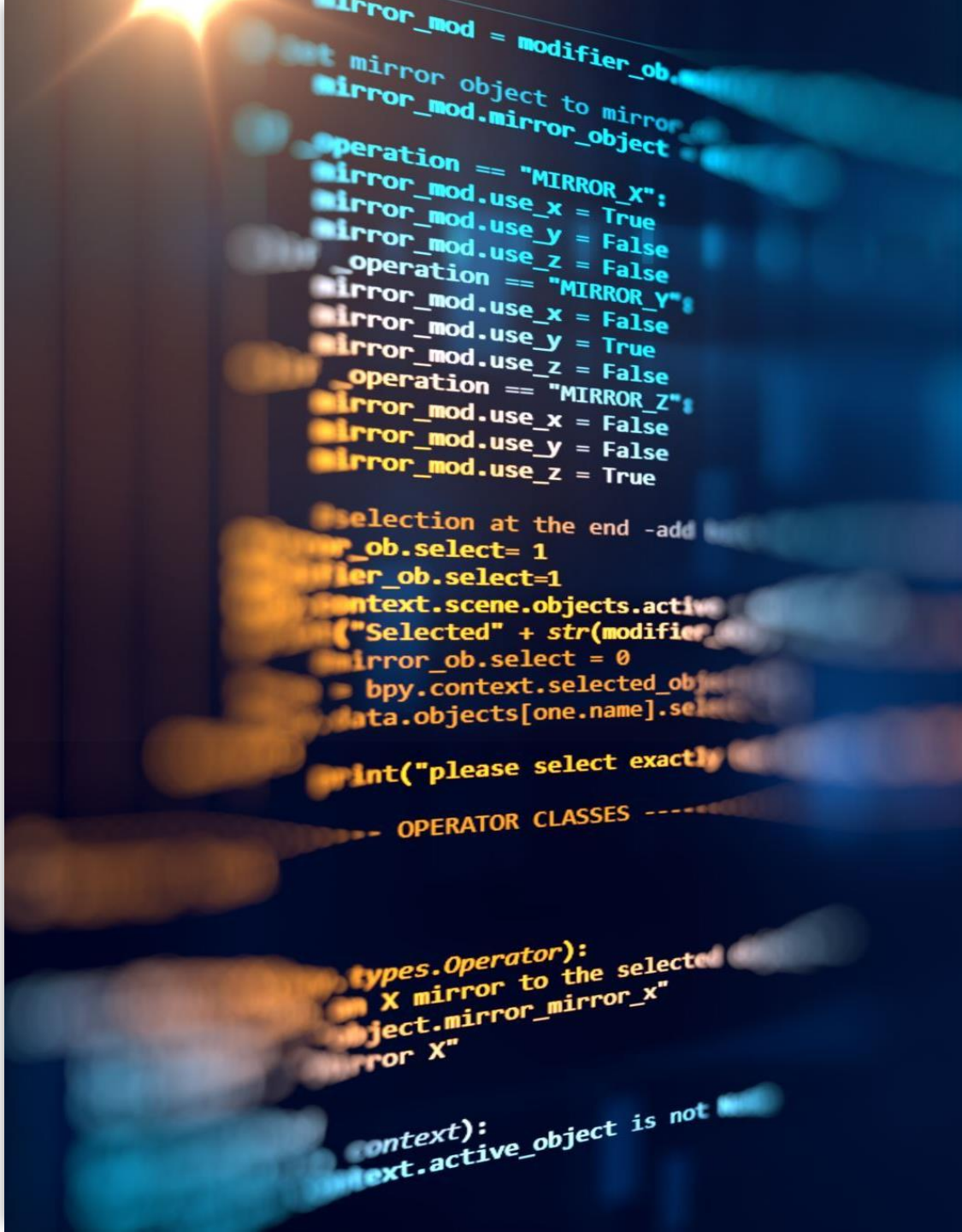
Handles runtime errors safely.

Example

```
try:
    x = 10 / 0
except:
    print("Error occurred")
```

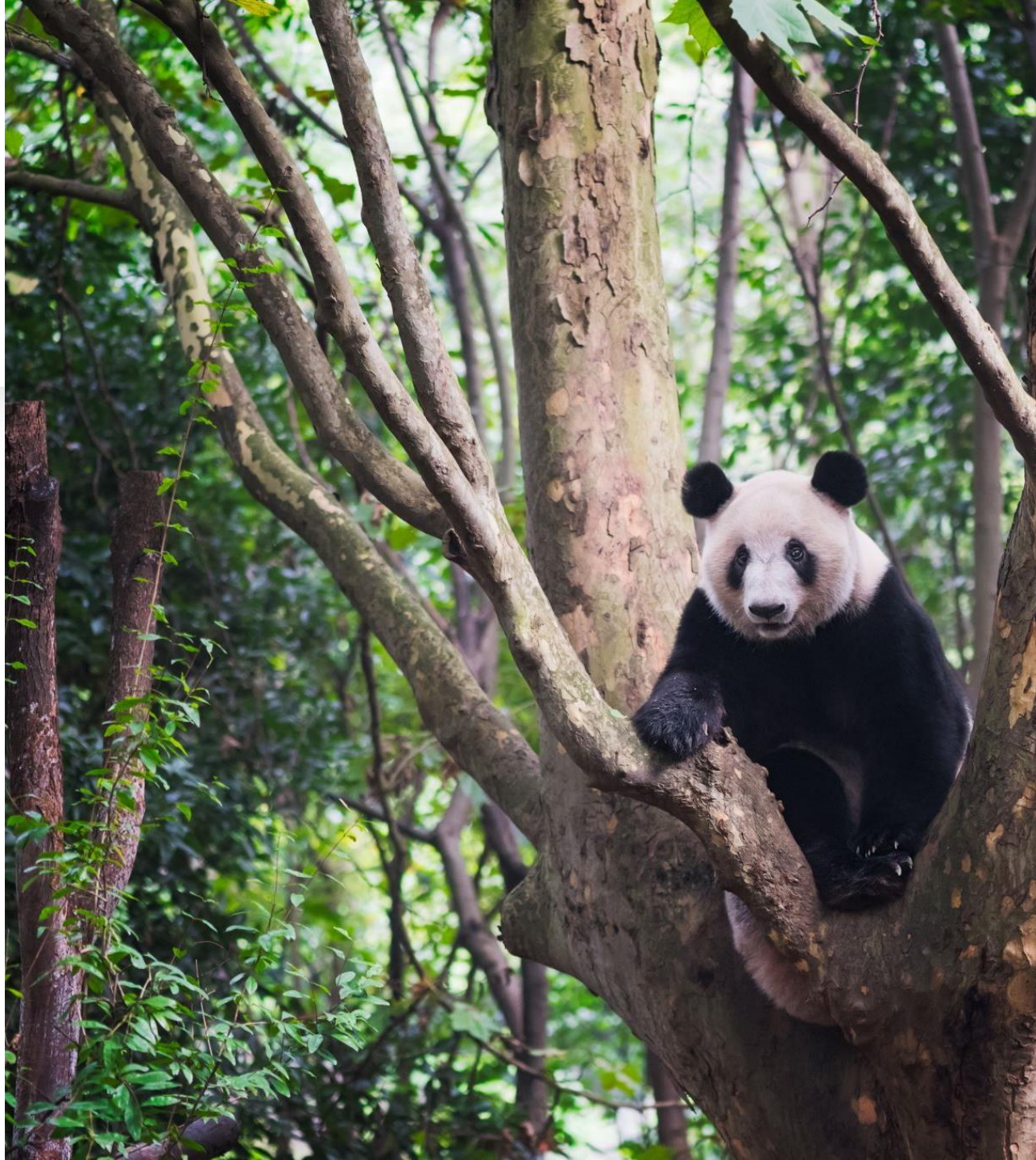
What is a Module?

- Explanation
- A file containing Python code.
- Example
- `import math`
- `print(math.pi)`



What is a Package?

- Explanation
- A collection of modules.
- 📌 Example: `numpy`, `pandas`



What is a Package in Python?

- Explanation
- A **package** in Python is a **collection of related modules** grouped together in a directory.
- 👉 It helps in:
- Organizing code
- Avoiding name conflicts
- Improving readability and maintenance
- 📌 A package usually contains:
- Multiple `.py` files (modules)
- `__init__.py` file (marks directory as a package)

NumPy

Description

NumPy is a package used for **numerical and mathematical operations**.

Example

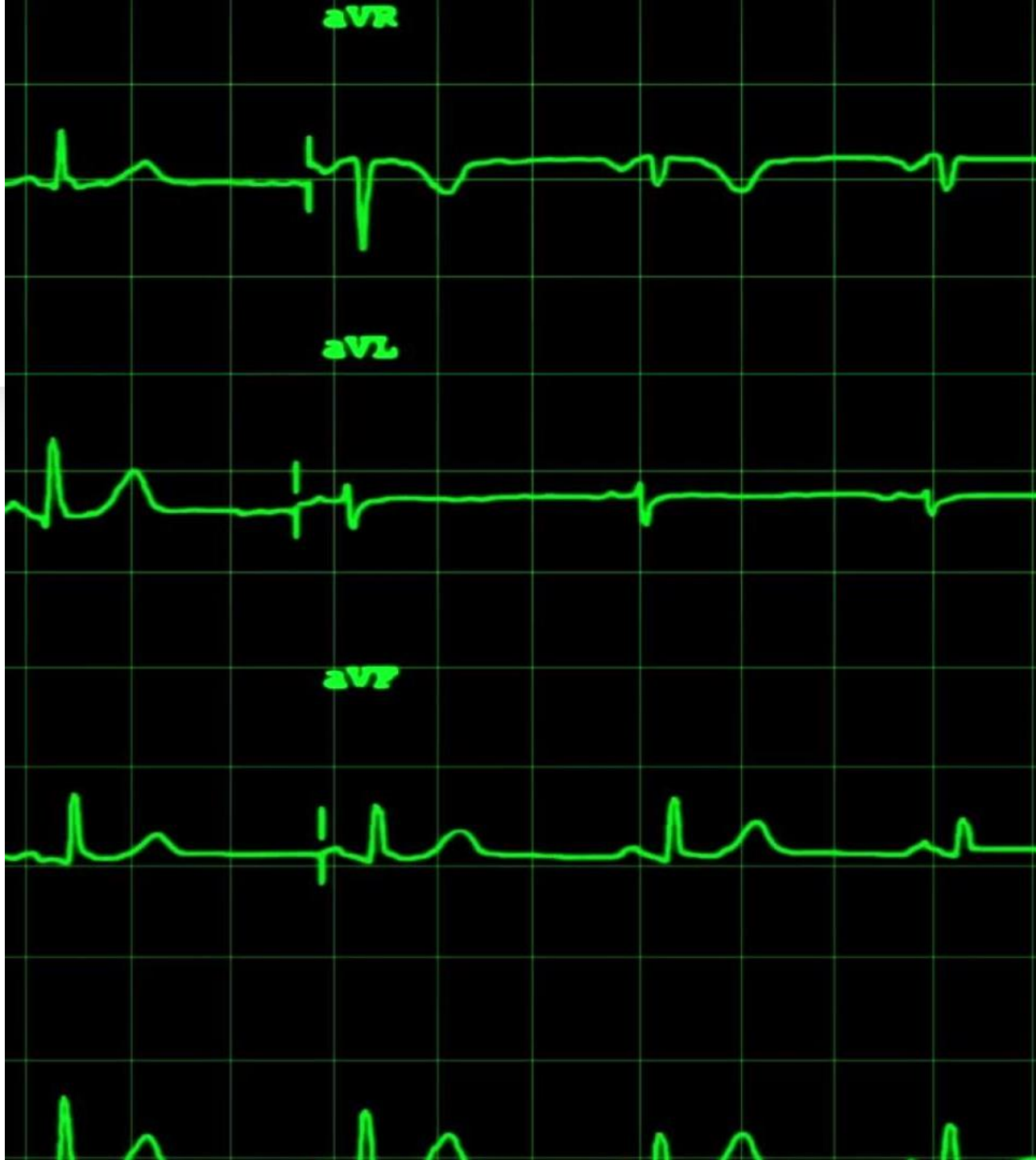
```
import numpy as np
arr = np.array([1, 2, 3])
print(arr)
```

Pandas

- Description
- Pandas is used for **data analysis and data manipulation**.
- Example
- `import pandas as pd`
-
- `data = {"name": ["Amit", "Rahul"], "age": [22, 23]}`
- `df = pd.DataFrame(data)`
- `print(df)`

Matplotlib

- Description
- Matplotlib is used for **data visualization and plotting graphs**.
- Example
- `import matplotlib.pyplot as plt`
- `x = [1, 2, 3]`
- `y = [2, 4, 6]`
-
- `plt.plot(x, y)`
- `plt.show()`



Requests

- Description
- Requests is used to **send HTTP requests** (GET, POST).
- Example
- `import requests`
-
- `response = requests.get("https://example.com")`
- `print(response.status_code)`

Math (Built-in Package)

Description

Math package provides **mathematical functions.**

Example

```
import math

print(math.sqrt(25))
print(math.factorial(5))
```

What is File Handling?

Explanation

Used to read/write files.

Example

```
file = open("test.txt", "w")
file.write("Hello")
file.close()
```

What is OOP?

Explanation

Programming based on **objects and classes**.

What is a Class?

Explanation

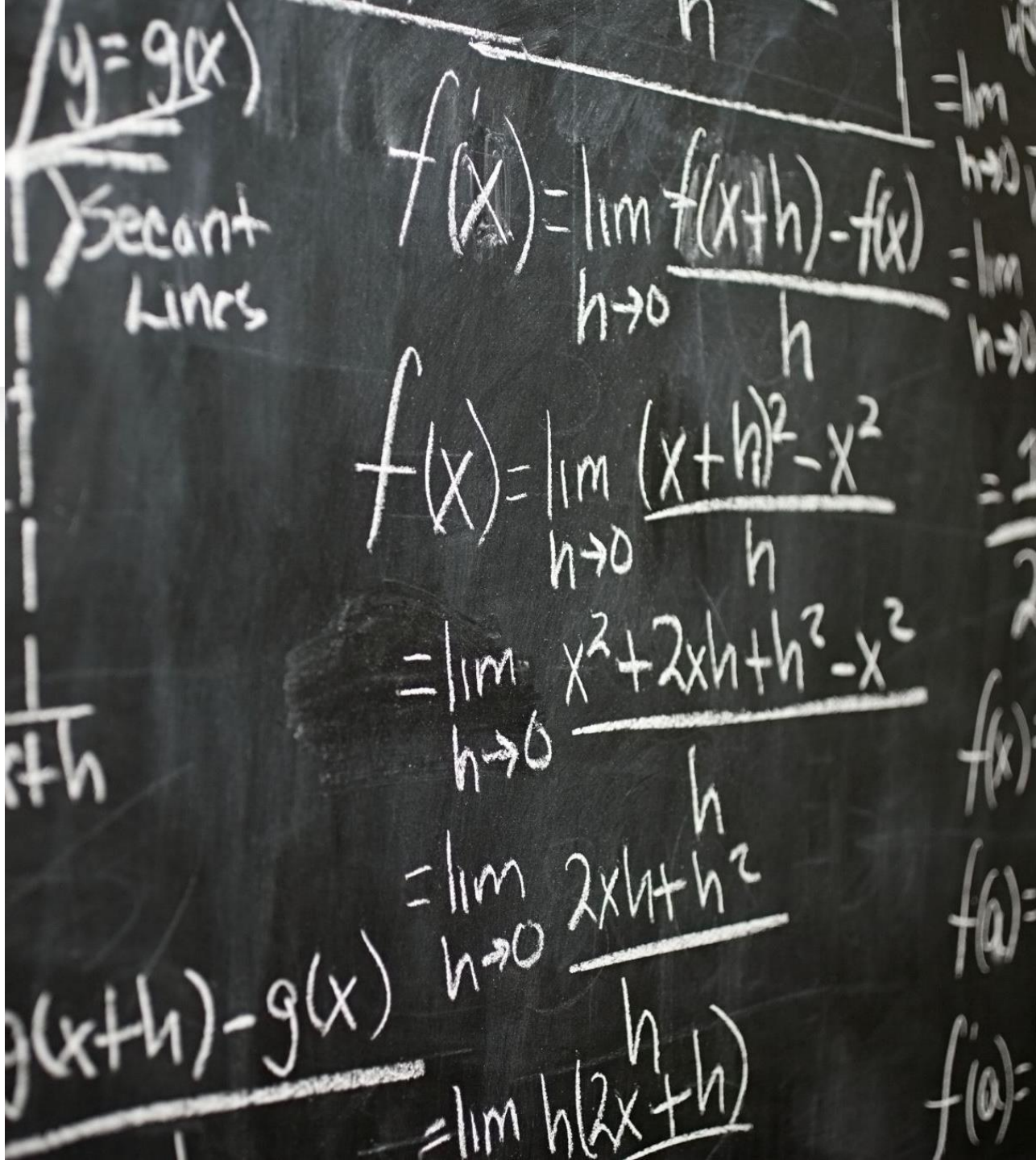
Blueprint for objects.

Example

```
class Student:
    name = "Amit"
```

What is an Object?

- Explanation
- Instance of a class.
- Example
- `s1 = Student()`
- `print(s1.name)`



What is a Constructor?

- Explanation
- A **constructor** is a special method in Python named `__init__`.
  It is **automatically called** when an object is created.
  Its main purpose is to **initialize object data (variables)**.
- class Student:
- def __init__(self, name, age):
- self.name = name
- self.age = age

- # Creating object
- s1 = Student("Rahul", 21)

- print(s1.name)
- print(s1.age)

What is `self`?

- Explanation
- `self` refers to the **current object of the class**.
  It is used to **access instance variables and methods** inside a class.
- 👉 Without `self`, Python cannot differentiate between variables.
- class Student:
- def __init__(self, name):
- self.name = name # self refers to current object

- def show(self):
- print("Name:", self.name)

- s1 = Student("Amit")
- s1.show()

What is Inheritance?

- Explanation
- **Inheritance** allows one class (**child class**) to **reuse properties and methods** of another class (**parent class**).

✓ Improves code reusability
✓ Supports hierarchical relationship

```
class Parent:
    def show_parent(self):
        print("This is Parent class")


class Child(Parent):
    def show_child(self):
        print("This is Child class")


obj = Child()
obj.show_parent()
obj.show_child()
```

What is Polymorphism?

Explanation

Polymorphism means **same function name, different behavior**, depending on the object or data type.

📌 Example: len() works on different data types.

✅ Complete Example

```
print(len("Python"))    # String
print(len([1, 2, 3]))   # List
print(len((10, 20)))    # Tuple
```

Types of Polymorphism in Python

Python supports polymorphism mainly in **4 ways**:

1. **Polymorphism using Built-in Functions**
2. **Polymorphism using Method Overriding**
3. **Polymorphism using Operator Overloading**
4. **Polymorphism using Duck Typing**

1 Polymorphism using Built-in Functions

Explanation

The same function works differently for different data types.

✅ Example: len()

```
print(len("Python"))    # String
print(len([1, 2, 3]))   # List
print(len((10, 20)))    # Tuple
```

2

Polymorphism using Method Overriding

- Explanation
- Child class **overrides** the method of the parent class with the same name.

- ✅ Example

```
class Parent:
    def show(self):
        print("This is Parent class")

class Child(Parent):
    def show(self):
        print("This is Child class")

obj = Child()
obj.show()
```

3 Polymorphism using Operator Overloading

Explanation

The same operator performs **different operations** for different data types.

✅ Example: + Operator

```
print(10 + 20)          # Addition
print("Hello " + "World")   # String concatenation
print([1, 2] + [3, 4])      # List merging
```

4 Polymorphism using Duck Typing (Important 🔥)

Explanation

"If it **looks like a duck and quacks like a duck**, it is a duck."

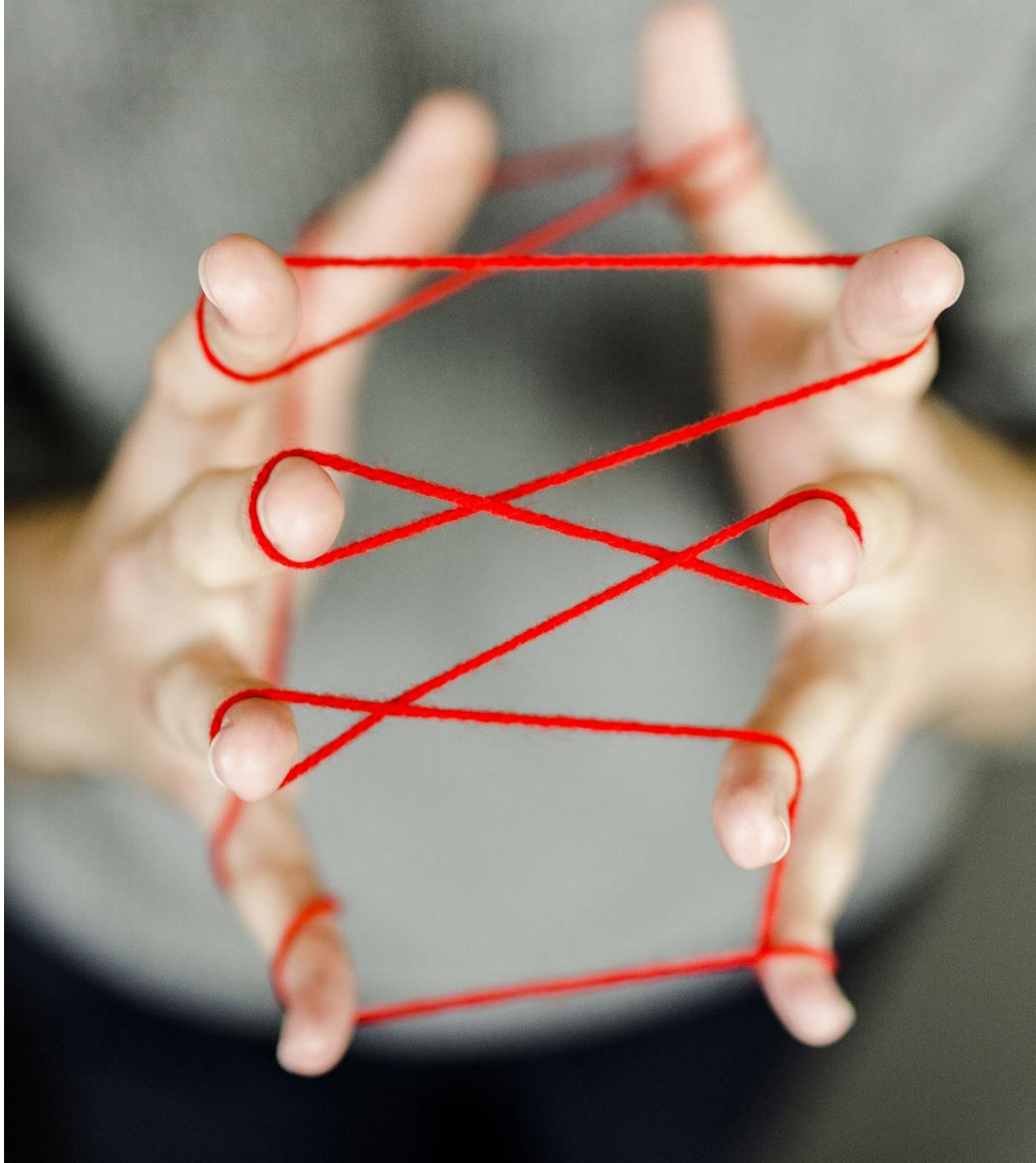
👉 Object type does not matter, **method name matters**.

- Example

```
class Dog:
    def sound(self):
        print("Dog barks")

class Cat:
    def sound(self):
        print("Cat meows")

def animal_sound(animal):
    animal.sound()

dog = Dog()
cat = Cat()

animal_sound(dog)
animal_sound(cat)
```

5 What is Encapsulation?

- Explanation
- **Encapsulation** means **data hiding** by restricting access to variables.
  It is achieved using **private variables** (`__variable`).
- 👉 Protects data from direct modification.
-



✅ Complete Example

```
class Bank:
 def __init__(self):
 self.__balance = 1000 # private
variable

 def show_balance(self):
 print("Balance:", self.__balance)

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

b = Bank()
b.show_balance()
b.deposit(500)
b.show_balance()
```