Class: blueprint of objects

Object is called instance of a class

Class emp:  
 empid = 0

emp1 = emp()  
emp1.empid = 1001  
print(f”empid: {emp1.empid}”)

* The . notation is used to access the attributes of the class
* A function inside a class is called a method
* A constructor is a special method in class used to create and initialize an obj in class. It is invoked automatically when an obj of class is created.
* Syntax: def \_\_init\_\_(self) [init is a reserved function in py. In oop, it is known as a constructor]
* When u do no twrite the constructor in the class created, py itself creates a constructor during the compilation of the program. It generates an empty constructor with no code in it.
* When a constructor accepts arguments along with self, it is called parameterized constructor

class Student:

# Constructor

def \_\_init\_\_(self, name, roll\_no):

self.name = name

self.roll\_no = roll\_no

def display(self):

print("Name:", self.name)

print("Roll Number:", self.roll\_no)

# Creating object → constructor runs automatically

s1 = Student("Alice", 101)

s1.display()

Encapsulation:

* binds attributes and methods together, restricts direct data access using private/protected members, and promotes data hiding and abstraction.
* class Employee:
* def \_\_init\_\_(self, name, salary):
* self.name = name # Public attribute
* self.\_department = "IT" # Protected attribute
* self.\_\_salary = salary # Private attribute
* def display\_info(self):
* print("Name:", self.name)
* print("Department:", self.\_department)
* # Public method to access private data
* def get\_salary(self):
* return self.\_\_salary
* # Object creation
* emp1 = Employee("Alice", 50000)
* # Accessing public member
* print(emp1.name) # ✅ Allowed
* # Accessing protected member
* print(emp1.\_department) # ⚠ Allowed, but not recommended
* # Accessing private member directly → Error
* # print(emp1.\_\_salary) # ❌ AttributeError
* # Accessing private member through method
* print(emp1.get\_salary()) # ✅ Correct way
* public data members are accessible within and outside of a class. All member variables of a class are by default public
* protected members are accessible with the class and subclass.
* Private members are accessible only within the class and cannot be accessed directly from class object.

Inheritance

* It is a concept where a new class (child/derived class) can use the properties and methods of an existing class (parent/base class).
* It helps in code reusability and creating a hierarchical relationship between classes.

1. Single Inheritance

* A child class inherits from one parent class.

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() # Inherited method

obj.show\_child()

2. Multiple Inheritance

* A child class inherits from more than one parent class.

class Father:

def skill1(self):

print("Father: Knows driving")

class Mother:

def skill2(self):

print("Mother: Knows cooking")

class Child(Father, Mother):

def skill3(self):

print("Child: Knows painting")

obj = Child()

obj.skill1()

obj.skill2()

obj.skill3()

3. Multilevel Inheritance

* A chain of inheritance where a class inherits from a child class, forming a hierarchy.

class Grandparent:

def property1(self):

print("Grandparent: Owns land")

class Parent(Grandparent):

def property2(self):

print("Parent: Owns house")

class Child(Parent):

def property3(self):

print("Child: Owns car")

obj = Child()

obj.property1() # Inherited from Grandparent

obj.property2() # Inherited from Parent

obj.property3()

polymorphism

* allows the same function/method/operator to behave differently depending on the object or context.
* It increases flexibility and reusability of code.
* 1. Operator Overloading

The same operator works differently for different data types.

print(5 + 10) # Integer addition → 15

print("Hello " + "World") # String concatenation → Hello World

print([1, 2] + [3, 4]) # List concatenation → [1, 2, 3, 4]

* 2. Method Overriding (Runtime Polymorphism)
* A child class can redefine a method of the parent class.
* class Animal:
* def speak(self):
* print("Animal makes a sound")
* class Dog(Animal):
* def speak(self): # Overriding parent method
* print("Dog barks")
* obj = Dog()
* obj.speak() # Output: Dog barks
* 3. Duck Typing
* Polymorphism is based on behavior, not inheritance.  
  “If it walks like a duck and quacks like a duck, it’s a duck.”
* class Bird:
* def fly(self):
* print("Bird can fly")
* class Airplane:
* def fly(self):
* print("Airplane can fly")
* # Function that accepts any object with fly()
* def lift\_off(obj):
* obj.fly()
* lift\_off(Bird()) # Bird can fly
* lift\_off(Airplane()) # Airplane can fly

How Polymorphism Supports OOP Principles

* Code Reusability: Same method name can work for different objects.
* Extensibility: New classes can be added without changing existing code.
* Abstraction: Focuses on what an object does, not how it does it.
* Flexibility: Reduces complexity by handling different types of objects through a common interface.

File handling

* key function for working with files is open()
* open() function takes 2 parameters, filename and mode
* syntax: f= open(filename,mode) OR whith open(filename,mode) as f:
* modes: r, w , a, r+, w+, a+

Errors: problems in program due to which program will stop execution.

Exceptions are raised when some internal events occur which changes the flow of the program.

Statements that raise the exception are kept inside the try clause and the statements that handle the exception are placed inside the except clause.

* Finally: The code inside the finally block is always executed, whether an exception occurs or not.
* It is generally used for cleanup activities, such as closing a file or a database connection

break Statement

Used to exit the loop prematurely when a condition is met.

continue Statement

Skips the current iteration and continues with the next one.

pass Statement

Used as a placeholder where a statement is syntactically required but no action is needed. pass does nothing but helps avoid an error.

UNIT1

Characteristics of python programming

* Interpreted language
* Weakly typed i.e. doesn’t declare a variable to be of specific type
* Free and open source
* Object oriented programming
* Easy syntax(user friendly)
* Cross platform
* Large standard library

Arbitrary Arguments (\*args)

Used when you don’t know how many arguments will be passed to the function.

Inside the function, they are received as a tuple.

def add\_numbers(\*args):

total = sum(args)

print("Sum:", total)

add\_numbers(10, 20, 30) # Output: Sum: 60

Arbitrary Keyword Arguments (\*\*kwargs)

Used when you want to accept named arguments that you don’t know in advance.

Inside the function, they are received as a dictionary.

def print\_info(\*\*kwargs):

for key, value in kwargs.items():

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

print\_info(name="Alice", age=22)

# Output:

# name: Alice

# age: 22

A function in Python is a block of reusable code that performs a specific task. Functions help organize code, reduce repetition, and improve readability.

Need & Benefits of Functions:

Modularity: Breaks code into smaller reusable blocks.

Reusability: Write once, use multiple times.

Readability: Makes code easier to understand.

Maintainability: Easier to debug and update.

Reduces Redundancy: Avoids writing same code repeatedly.

Global Variables:

A global variable is declared outside all functions.

x = 5

def change():

global x

x = 20

change()

print(x)  # Output: 20

Constants:

A constant is a variable whose value should not change.

Python does not have built-in constant types, so by convention, constants are written in all uppercase letters.

PI = 3.14159 # Constant

GRAVITY = 9.8

print("Value of PI is:", PI)

🔸 Local Variables:

Declared inside a function.

Accessible only within that function.

Not visible outside the function.

🔸 Global Variables:

Declared outside all functions.

Accessible throughout the program, inside or outside functions (when declared as global in a function).

It can be accessed and modified from anywhere in the program.

break Statement

Used to exit the loop prematurely when a condition is met.

continue Statement

Skips the current iteration and continues with the next one.

pass Statement

Used as a placeholder where a statement is syntactically required but no action is needed. pass does nothing but helps avoid an error.

A tuple is an immutable (unchangeable) ordered collection of items.

It is defined using parentheses ().

Tuples can store elements of different data types.

A black screen with white text

AI-generated content may be incorrect.