Attributes, properties, fields state, variables, characteristics are all same thing

Object is a real world thing.

It has

1. Identity : Coffee mug
2. Behaviour: fill(), Empty(), clean()
3. Attributes : color, size, fullness

Nouns: Object

Verbs: Behaviours

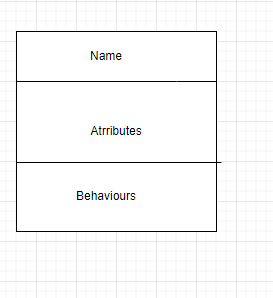
Class : it is a code template for creating objects

Class has:

1. Name
2. Attributes
3. Behaviours

Method: A block of code that can do some action and return some value

Methods are part of a class but functions can exist independently.



Object is an instance of a class.

Pillar of OOPS: APIE

1. Abstraction:  It’s the process of hiding the internal details of an application from the outer world and showing only the necessary features of an object.

**Real-life examples**: Your car and microwave are great examples of abstraction. You don’t need to know how the engine starts or how the microwave heats your food. The internal implementation and complex logic are completely hidden from you

from abc import ABC, abstractmethod

class Absclass(ABC):

def print(self,x):

print("Passed value: ", x)

@abstractmethod

def task(self):

print("We are inside Absclass task")

class test\_class(Absclass):

def task(self):

print("We are inside test\_class task")

test = test\_class()

test.task()

test.print(10)

1. Polymorphism :  [In OOP, polymorphism describes situations where you can access objects of different types through the same interface1](https://stackify.com/oop-concept-polymorphism/)[2](https://www.geeksforgeeks.org/polymorphism-in-java/)[3](https://www.codewithc.com/exploring-the-power-of-polymorphism-in-object-oriented-programming/). [Each type can provide its own independent implementation of this interface](https://stackify.com/oop-concept-polymorphism/)

* **Real-life examples**: A person can have different characteristics at the same time. For example, a man can be a father, a husband, and an employee. So the same person possesses different behaviors in different situations. [This is called polymorphis](https://www.geeksforgeeks.org/polymorphism-in-java/)m

Here are some key points about polymorphism in Python:

* [**Function Polymorphism**: Python supports function polymorphism where the same function can be used for different types1](https://www.geeksforgeeks.org/polymorphism-in-python/). [For example, the len() function can be used with many data types such as string, list, tuple, set, and dictionary1](https://www.geeksforgeeks.org/polymorphism-in-python/).

print(len("geeks")) # Outputs: 5

print(len([10, 20, 30])) # Outputs: 3

* **Operator Polymorphism**: Operators in Python can also exhibit polymorphism. [For example, the + operator performs arithmetic addition on numbers, concatenation on strings, and merge operation on lists1](https://www.geeksforgeeks.org/polymorphism-in-python/).

# For integer data types, \* operator is used to perform arithmetic multiplication operation.

num1 = 3

num2 = 4

print(num1 \* num2) # Outputs: 12

# For string and integer data types, \* operator is used to repeat the string.

str1 = "Python"

num1 = 3

print(str1 \* num1) # Outputs: PythonPythonPython

* [**Class Polymorphism**: Python allows different classes to have methods with the same name2](https://www.programiz.com/python-programming/polymorphism). [This allows us to use objects of different classes in the same way2](https://www.programiz.com/python-programming/polymorphism). [For example, if there is a make\_sound method in both Dog and Cat classes, we can call this method on an object without worrying about the class of the object2](https://www.programiz.com/python-programming/polymorphism).

class India():

def capital(self):

print("New Delhi is the capital of India.")

def language(self):

print("Hindi is the most widely spoken language of India.")

def type(self):

print("India is a developing country.")

class USA():

def capital(self):

print("Washington, D.C. is the capital of USA.")

def language(self):

print("English is the primary language of USA.")

def type(self):

print("USA is a developed country.")

obj\_ind = India()

obj\_usa = USA()

for country in (obj\_ind, obj\_usa):

country.capital()

country.language()

country.type()

* [**Polymorphism with Inheritance**: In Python, polymorphism lets us define methods in the child class that have the same name as the methods in the parent class1](https://www.geeksforgeeks.org/polymorphism-in-python/). [This is particularly useful when the method inherited from the parent class doesn’t quite fit the child class1](https://www.geeksforgeeks.org/polymorphism-in-python/). [In such cases, we re-implement the method in the child class1](https://www.geeksforgeeks.org/polymorphism-in-python/). [This process of re-implementing a method in the child class is known as Method Overriding](https://www.geeksforgeeks.org/polymorphism-in-python/)

class Bird:

def intro(self):

print("There are many types of birds.")

def flight(self):

print("Most of the birds can fly but some cannot.")

class Sparrow(Bird):

def flight(self):

print("Sparrows can fly.")

class Ostrich(Bird):

def flight(self):

print("Ostriches cannot fly.")

obj\_bird = Bird()

obj\_spr = Sparrow()

obj\_ost = Ostrich()

obj\_bird.intro()

obj\_bird.flight()

obj\_spr.intro()

obj\_spr.flight()

obj\_ost.intro()

obj\_ost.flight()

1. Inheritance

Inheritance is a key concept in Object-Oriented Programming (OOP). [It allows you to create new classes that reuse, extend, and modify the behavior defined in another class1](https://www.codecademy.com/resources/blog/what-is-inheritance/)[2](https://www.geeksforgeeks.org/inheritance-in-java/)[3](https://www.geeksforgeeks.org/inheritance-in-c/). [The class whose members are inherited is called the **base class**, and the class that inherits those members is called the **derived class**1](https://www.codecademy.com/resources/blog/what-is-inheritance/)[2](https://www.geeksforgeeks.org/inheritance-in-java/)[3](https://www.geeksforgeeks.org/inheritance-in-c/).

Here are some key points about inheritance:

* [**Code Reusability**: Inheritance allows us to reuse code from the base class, reducing redundancy and making the code more manageable1](https://www.codecademy.com/resources/blog/what-is-inheritance/)[2](https://www.geeksforgeeks.org/inheritance-in-java/)[3](https://www.geeksforgeeks.org/inheritance-in-c/).
* [**Types of Inheritance**: There are several types of inheritance, including single, multiple, multilevel, and hierarchical1](https://www.codecademy.com/resources/blog/what-is-inheritance/)[2](https://www.geeksforgeeks.org/inheritance-in-java/)[3](https://www.geeksforgeeks.org/inheritance-in-c/).
* **Access Specifiers**: Access specifiers like public, private, and protected play a crucial role in inheritance. [They define the accessibility of the base class members in the derived class](https://www.codecademy.com/resources/blog/what-is-inheritance/)

class Animal: # Base class

def eat(self):

print("I can eat")

class Dog(Animal): # Derived class

def eat(self):

print("I like to eat bones")

# Create an object of the subclass

labrador = Dog()

# Call the eat() method on the labrador object

labrador.eat()

class Mammal:

def mammal\_info(self):

print("Mammals can give direct birth.")

class WingedAnimal:

def winged\_animal\_info(self):

print("Winged animals can flap.")

class Bat(Mammal, WingedAnimal): # Multiple Inheritance

pass

# Create an object of Bat class

b1 = Bat()

# Call methods from both parent classes

b1.mammal\_info() # Outputs: "Mammals can give direct birth."

b1.winged\_animal\_info() # Outputs: "Winged animals can flap."

1. Encapsulation

Encapsulation is a fundamental concept in Object-Oriented Programming (OOP), including C#. [It is the process of wrapping up data and methods into a single unit known as a class1](https://www.geeksforgeeks.org/c-sharp-encapsulation/)[2](https://dotnettutorials.net/lesson/encapsulation-csharp/)[3](https://www.scaler.com/topics/csharp/encapsulation-in-c-sharp/)[4](https://www.shekhali.com/csharp-encapsulation/)[5](https://www.c-sharpcorner.com/article/encapsulation-in-C-Sharp/). [This mechanism binds together the data and the functions that manipulate them1](https://www.geeksforgeeks.org/c-sharp-encapsulation/).

Here are some key points about encapsulation:

* [**Data Hiding**: Encapsulation hides the internal state and functionality of an object and only allows access through a public set of functions1](https://www.geeksforgeeks.org/c-sharp-encapsulation/)[2](https://dotnettutorials.net/lesson/encapsulation-csharp/)[3](https://www.scaler.com/topics/csharp/encapsulation-in-c-sharp/)[4](https://www.shekhali.com/csharp-encapsulation/)[5](https://www.c-sharpcorner.com/article/encapsulation-in-C-Sharp/). [The variables or data of a class are hidden from any other class and can be accessed only through any member function of its own class in which they are declared1](https://www.geeksforgeeks.org/c-sharp-encapsulation/)[2](https://dotnettutorials.net/lesson/encapsulation-csharp/)[3](https://www.scaler.com/topics/csharp/encapsulation-in-c-sharp/)[4](https://www.shekhali.com/csharp-encapsulation/)[5](https://www.c-sharpcorner.com/article/encapsulation-in-C-Sharp/).

[**Abstract Classes and Methods**: In Python, we can achieve data abstraction by using abstract classes and abstract methods1](https://www.geeksforgeeks.org/encapsulation-in-python/)[2](https://pynative.com/python-encapsulation/). [An abstract class is a class in which one or more methods are declared but not implemented1](https://www.geeksforgeeks.org/encapsulation-in-python/)[2](https://pynative.com/python-encapsulation/). [These methods are known as abstract methods and are declared using the @abstractmethod decorator](https://www.geeksforgeeks.org/encapsulation-in-python/)

class Employee:

def \_\_init\_\_(self, name, salary):

self.\_name = name # private attribute

self.\_salary = salary # private attribute

# getter method

def get\_salary(self):

return self.\_salary

# setter method

def set\_salary(self, value):

self.\_salary = value

# method to calculate yearly salary

def display\_yearly\_salary(self):

print("Yearly Salary: ", self.\_salary \* 12)

emp = Employee("John", 5000)

emp.display\_yearly\_salary() # Outputs: "Yearly Salary: 60000"