Polymorphism

presented by

Alam Zaib

Table of content

- Introduction Polymorphism
- Class polymorphism and inheritance class polymorphism
- Example of polymorphism with code
- Encapsulation with example
- Abstraction with example

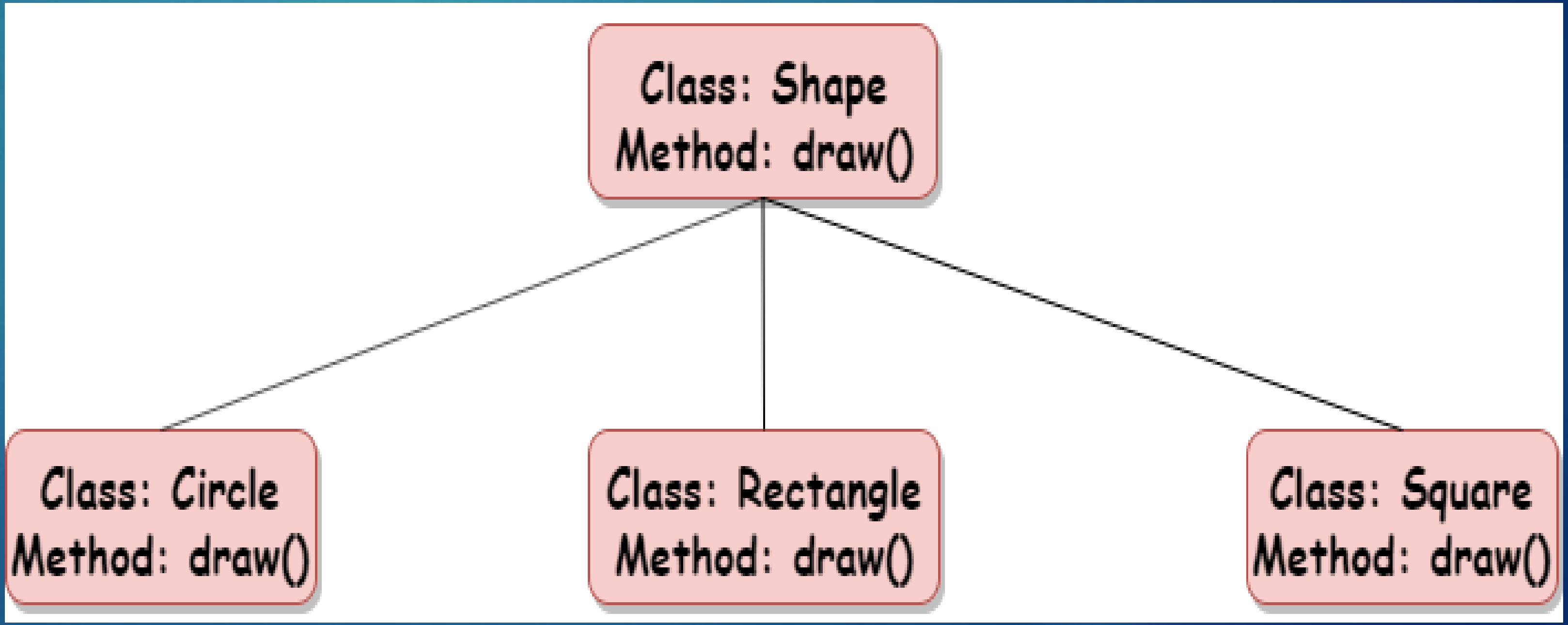
Polymorphism

1.The word “polymorphism” means "many forms", and in programming it refers to methods/functions/operators with the same name that can be executed on many objects or classes.

2. In simple words, we can define polymorphism as the ability of a message to be displayed in more than one form.

Class Polymorphism

Polymorphism is often used in Class methods, where we can have multiple classes with the same method name.
For example, say we have three classes: Circle, Rectangle, and Square, and they all have a method called draw():

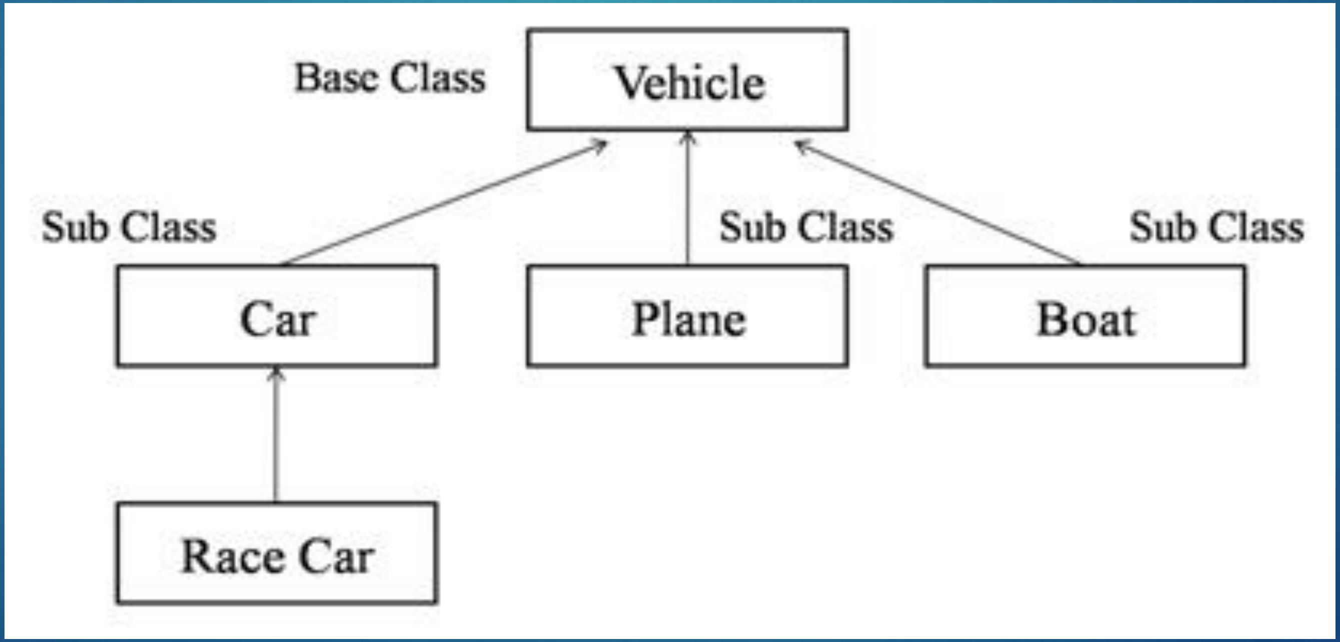


Example
Different classes with the same method:

```
class Car:
  def __init__(self, brand, model):
    self.brand = brand
    self.model = model

  def move(self):
    print("Drive!")

class Boat:
  def __init__(self, brand, model):
    self.brand = brand
    self.model = model

  def move(self):
    print("Sail!")

car1 = Car("Ford", "Mustang")       #Create a Car object
boat1 = Boat("Ibiza", "Touring 20") #Create a Boat object
for x in (car1, boat1):
  x.move()
```

Inheritance Class Polymorphism

What about classes with child classes with the same name? Can we use polymorphism there?
Yes. If we use the example and make a parent class called `Vehicle`, and make `Car`, `Boat`, `Plane` child classes of `Vehicle`, the child classes inherits the `Vehicle` methods, but can override them:



Example
Create a class called Vehicle and make Car, Boat, Plane child classes of Vehicle:

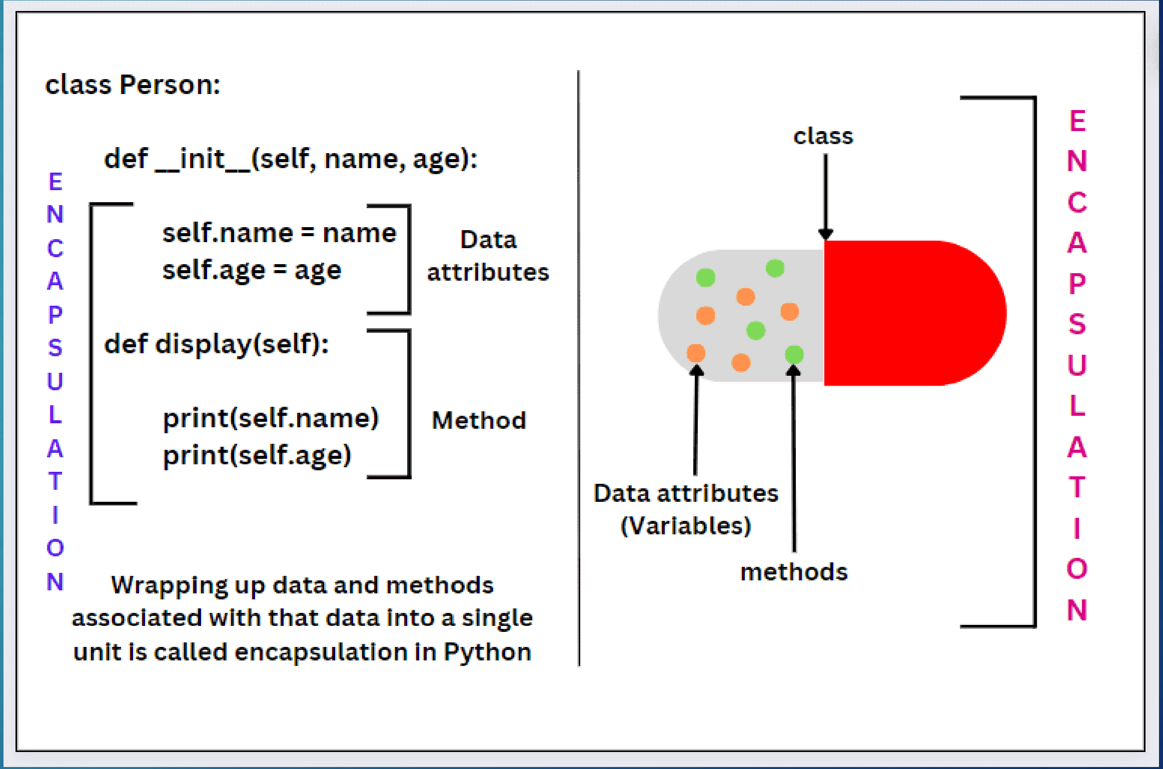
```
class Vehicle:
  def __init__(self, brand, model):
    self.brand = brand
    self.model = model

  def move(self):
    print("Move!")

class Car(Vehicle):
  pass

class Boat(Vehicle):
  def move(self):
    print("Sail!")


car1 = Car("Ford", "Mustang")       #Create a Car object
boat1 = Boat("Ibiza", "Touring 20") #Create a Boat object

for x in (car1, boat1):
  print(x.brand)
  print(x.model)
  x.move()
```

Encapsulation

Encapsulation is the process of hiding the internal state of an object and requiring all interactions to be performed through an object's methods. This approach:

- Provides better control over data.
- Prevents accidental modification of data.
- Promotes modular programming.

Python achieves encapsulation through **public**, **protected** and **private** attributes.



```
class Person:

    def __init__(self, name, age):
        self.name = name
        self.age = age
    def display(self):
        print(self.name)
        print(self.age)
```

E N C A P S U L A T I O N
Data attributes
Method
class
Data attributes (Variables)
methods
E N C A P S U L A T I O N

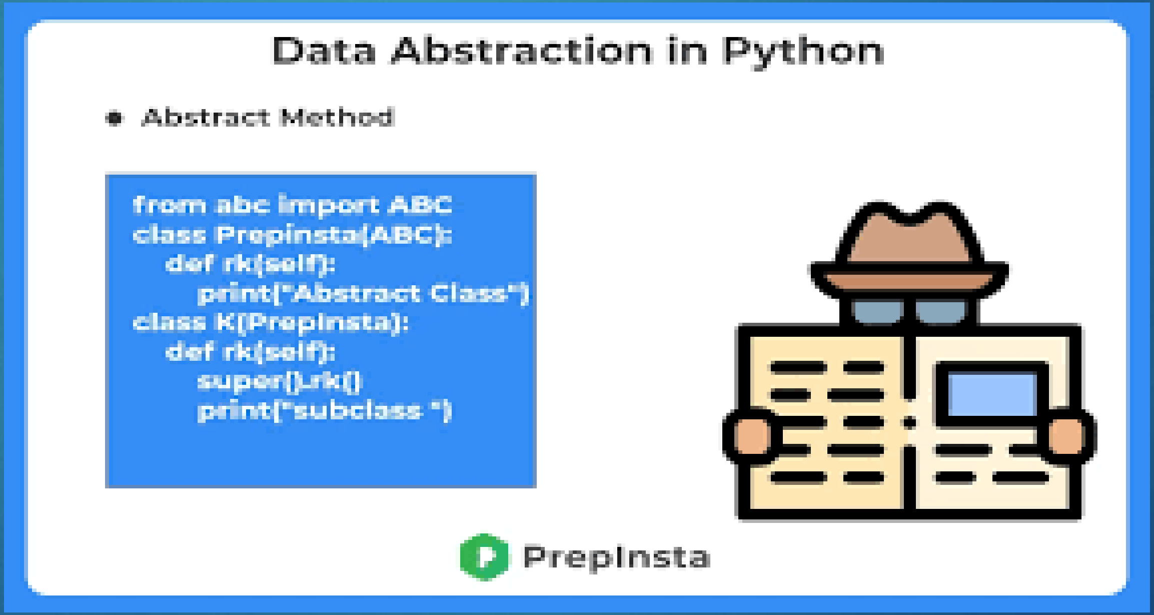
Wrapping up data and methods associated with that data into a single unit is called encapsulation in Python

Example

```
class Car:
    def __init__(self, brand, speed):
        self.brand = brand  # Public attribute
        self.__speed = speed  # Private attribute

    def get_speed(self):
        return self.__speed  # Controlled access

car = Car("Toyota", 120)
print(car.get_speed())  # Accessing private data via a method
```

Data abstraction

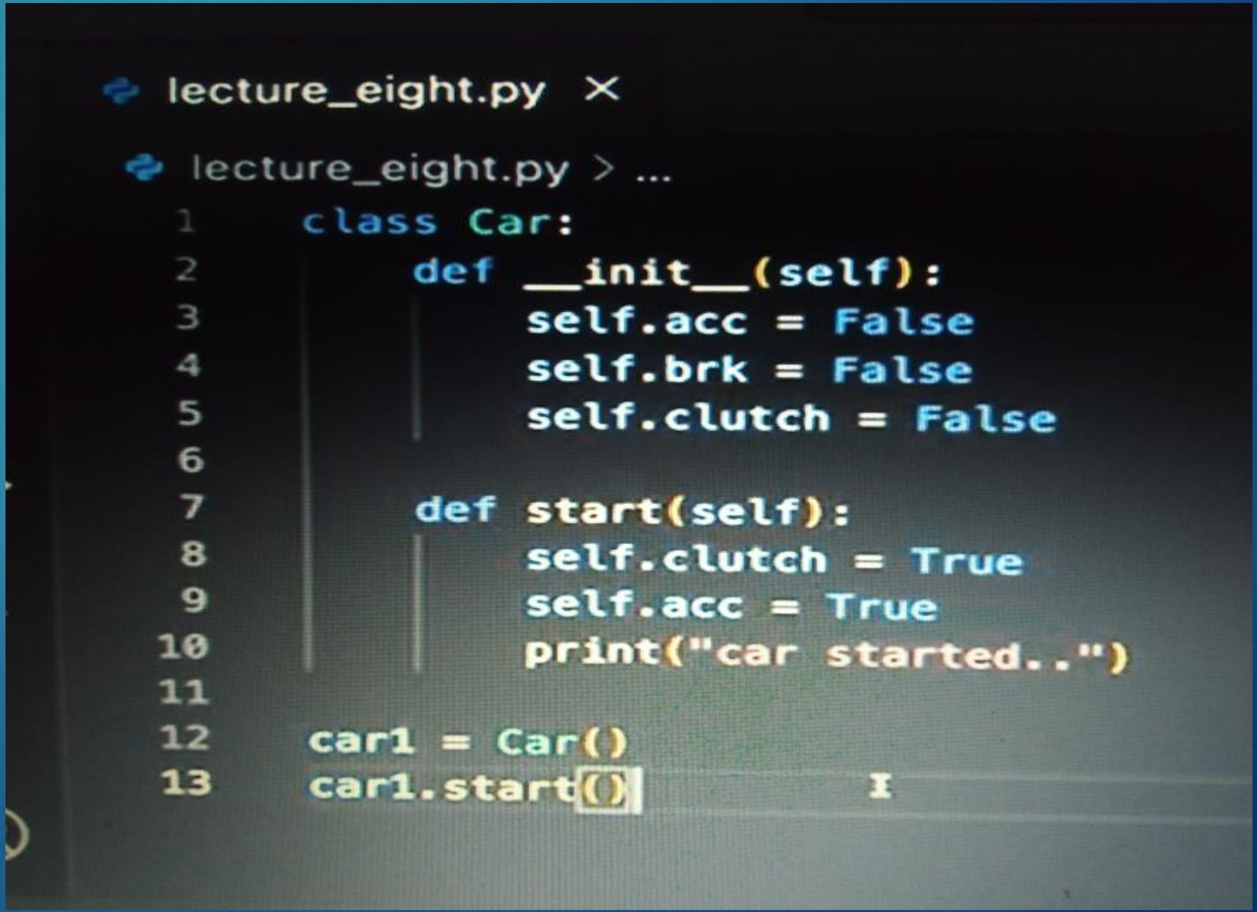
Data abstraction is one of the most essential concepts of Python OOPs which is used to hide irrelevant details from the user and show the details that are relevant to the users.



```
from abc import ABC
class Prepinsta(ABC):
    def rk(self):
        print("Abstract Class")
class K(Prepinsta):
    def rk(self):
        super().rk()
        print("subclass ")
```

Example

A simple example of this can be a car. A car has an accelerator, clutch, and break and we all know that pressing an accelerator will increase the speed of the car and applying the brake can stop the car but we don't know the internal mechanism of the car and how these functionalities can work this detail hiding is known as data abstraction.



```python
class Car:
    def __init__(self):
        self.acc = False
        self.brk = False
        self.clutch = False

    def start(self):
        self.clutch = True
        self.acc = True
        print("car started..")

car1 = Car()
car1.start()
```

THANK
YOU