04 Python OOP

Object-oriented Programming (OOP)	Procedure-oriented Programming (POP)
Encapsulates data and functions into objects	Focuses on functions or procedures to operate on data
Provide Data hiding	Doesn't provide, proper way for Data binding
C++, C#, Java, .Net, Python	C, Fortran, Pascal, VB

Overview of OOP (Object Oriented Programming)

- Python is a multi-paradigm programming language (POP & OOP)

```
def add(a, b):
    return a + b
result = add(5, 3)


class Calculator:
    def add(self, a, b):
        return a + b
calc = Calculator()
result = calc.add(5, 3)
```

01 class & object

- Class is a collection of data (variables) & methods

```
        class <class name>(<parent class name>):
```

- Object is an instance of a class

```
<object name> = <class name>
```

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

    def display(self):
        return f"Car: {self.brand} {self.model}"

my_car1 = Car("Toyota", "Corolla")
my_car2 = Car("Mahindra", "Thar")
print(my_car1.display())
print(my_car2.display())
```

Constructor

- Special method `__init__` to initialize objects
- **Non-parameterized Constructor**
  - Uses when we do not want to manipulate the value
  - The constructor that has only self as an argument
- **Parameterized Constructor**
  - Has multiple parameters along with the self
  - `self.parameter` required

Accessor & Mutator

- Methods that access object attributes - Accessor
- Methods that modify object attributes - Mutator

```
class Circle:
    def __init__(self, radius):    # constructor
        self.__radius = radius     # private attribute

    def get_radius(self):          # accessor
        return self.__radius

    def set_radius(self, radius):  # mutator
        self.__radius = radius

circle = Circle(5)
print(circle.get_radius())  # 5
circle.set_radius(10)
print(circle.get_radius())  # 10
```

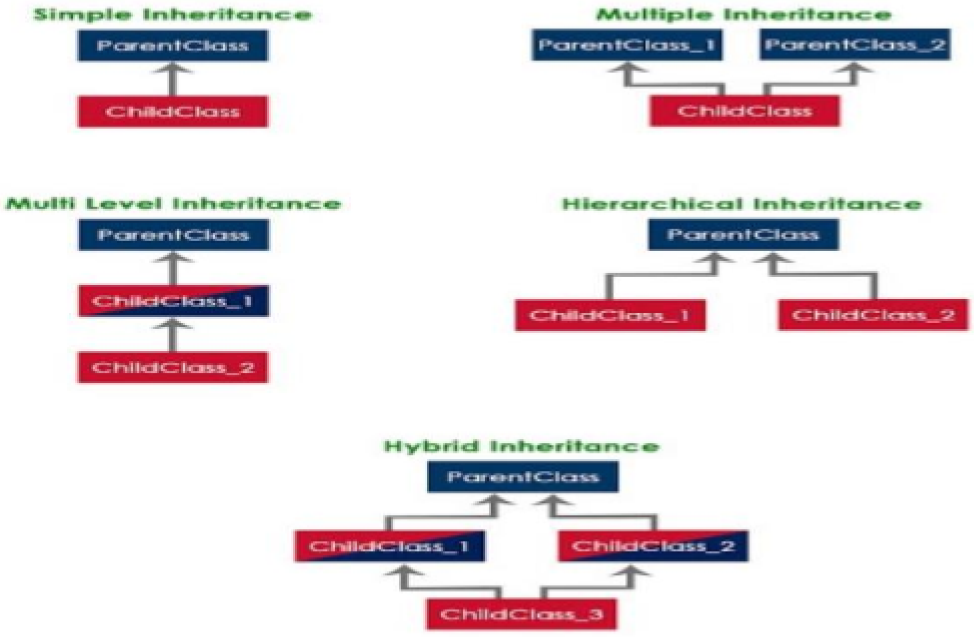
02 Data Encapsulation

- Restrictions imposed on the access to methods and variables
- The attribute `__balance` is defined with double underscores. This makes it a private attribute
- When you define a private attribute `__balance`, Python internally changes its name to `_ClassName__balance`. This is done to avoid attribute name conflicts in subclasses

```
class BankAccount:
        def __init__(self, balance): # constructor
                self.__balance = balance

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

        def getBalance(self):        # accessor
                return self.__balance

account = BankAccount(1000)
# print(account._BankAccount__balance) : Output: 1000
account.deposit(2000)
print(account.getBalance()) # 3000
```

03 Inheritance

- Inheritance allows a class (child class) to inherit attributes and methods from another class (parent class)



```
class Animal:
    def __init__(self, name):
        self.name = name
    def speak(self):
        pass

class Dog(Animal):
    def speak(self):
        return f"{self.name} says Woof!"

dog = Dog("Buddy")
print(dog.speak())  # Buddy says Woof
```

Method Resolution Order (MRO)

- Order in which Python looks for a method in a hierarchy of classes
- MRO works in a depth first left to right way
- eg - Multiple-inheritance

Super() function

- Make the child class inherit all the methods and properties from its parent
- By using the super() function, you do not have to use the name of the parent element, it will automatically inherit the methods and properties from its parent

04 Polymorphism

- object of a class can have many different forms to respond in different ways to any message or action
  - Operator Overloading
  - Method Overloading
  - Method Overriding

```
class Bird:
    def speak(self):
        return "Bird is making a sound."

class Parrot(Bird):
    def speak(self):
        return "Parrot is talking."

class Sparrow(Bird):
    def speak(self):
        return "Sparrow is chirping."

def make_bird_speak(bird):
    print(bird.speak())

parrot = Parrot()
sparrow = Sparrow()

make_bird_speak(parrot)  # Parrot is talking.
make_bird_speak(sparrow)  # Sparrow is chirping.
```

05 Abstraction

- Abstraction is the concept of hiding the complex implementation details and showing only the essential features of an object
- It is achieved using abstract classes and interfaces

```
from abc import ABC, abstractmethod

class Shape(ABC):
    @abstractmethod
    def area(self):
        pass

    @abstractmethod
    def perimeter(self):
        pass

class Rectangle(Shape):
    def __init__(self, width, height):
        self.width = width
        self.height = height

    def area(self):
        return self.width * self.height

    def perimeter(self):
        return 2 * (self.width + self.height)

class Circle(Shape):
    def __init__(self, radius):
        self.radius = radius

    def area(self):
        return 3.14 * self.radius * self.radius

    def perimeter(self):
        return 2 * 3.14 * self.radius

# Usage
```

```
shapes = [Rectangle(10, 20), Circle(15)]
for shape in shapes:
    print(f"Area: {shape.area()}, Perimeter: {shape.perimeter()}")
```

Exception Handling

- `try` block contains the code that might raise an exception
- `except` blocks handle specific exceptions
- `else` block executes if no exceptions are raised
- `finally` block executes regardless of whether an exception was raised
- `raise` An exception can be raised forcefully

```
def divide(a, b):
    try:
        result = a / b
    except ZeroDivisionError:
        print("Error: Division by zero is not allowed.")
    except TypeError:
        print("Error: Invalid input type. Please provide numbers.")
    else:
        print(f"Result: {result}")
    finally:
        print("Execution completed.")

# Usage
divide(10, 2)
divide(10, 0)
divide(10, 'a')
```

- ZeroDivisionError
- FileNotFoundError
- NameError
- TypeError
- ValueError
- NotImplementedError : When an object is supposed to support an operation but it has not been implemented yet
- User-Defined Exception

```
class invalidAge(Exception):
        def display(self):
                print("Age cannot be below 18 ")

try:
        age = int(input("Enter the Age :"))
        if age < 18:
                raise invalidAge()
except invalidAge as a:
        a.display()
else:
        print(name, "Congratulations !!! You can enter the CLUB")
```