[Day-10]

Polymorphism

Polymorphism - Allows obj of different classes to be treated as object of a common superclass. It provides a way to perform a single action in different forms

↳ Polymorphism is typically achieved through method overriding and interfaces

↳ Method Overriding - allows a child class to provide a specific implementation of a method that is already defined in its parent class.

```
#Base class
class Animal:
    def speak (self):
        return "Sound of animal"

## Desired class1
class Dog(Animal):
    def speak (self):
        return "Woof!"

# Derived class
class Cat (Animal):
    def speak (self):
        return "Meow"
```

```
# Function that demonstrates
Polymorphism
def animal_speak(animal):
    print (animal.speak())
animal_speak(dog) ——— # Woof!
```

```
dog = Dog()
cat = Cat()
dog.speak() ——— # Woof!
cat.speak() ——— # Meow!
```

Polymorphism with abstract base class

ABC's are used to define common methods for a group of related objects. They can enforce that derived classes implement particular methods, promoting consistency across different implementations.

```
from abc import ABC, abstractmethod

# Define an abstract class
class Vehicle(ABC):
    @abstractmethod
    def start_engine(self):
        pass

# Derived class 1
class Car (Vehicle):
    def start_engine (self)
        return "Car engine Started"

# Derived class 2
class Motorcycle (Vehicle)
    def start_engine (self):
        return "Motorcycle engine Started
```

```
# Create objects
Car = Car( )
motorcycle = Motorcycle( )

Start_vehicle(car)

O/p: Car engine Started
```

Encapsulation and Abstraction

- → These are two fundamental Principles of OOP that help in designing robust, maintainable and reusable code.
- → Encapsulation involves bundling data & methods that operate on the data within a single unit, while abstraction involves hiding complex implementat^n details & exposing only the necessary features.