Encapsulation

- Action of enclosing something
  Example: Music (Encapsulation of variety of sounds)

Why:

- While dealing with classes and sensitive data, providing global access to data and variables will result in modification of data unknowingly.

Definition:

Encapsulation is wrapping data (methods and variables) under one unit, hiding data, and preventing it from accessing using another class than the class it is declared.

- In order to restrict the data (variables and methods) from getting overridden accidentally and protect data/partial of data from accessing outside the declared class, there are some ways which Encapsulation provides us to access data members.

Access Modifiers

Public data Members

Data variables can be accessed from both inside and outside of class.

```
class Employee:
    def __init__(self, name, salary, project):
        self.name = name
        self.salary = salary
        self.project = project
    def show(self):
        print("Name: ", self.name, 'Salary:', self.salary)
    def work(self):
        print(self.name, 'is working on', self.project)
emp = Employee('Jessica', 8000, 'Data Science')
emp.show()
emp.work()
emp.salary = 6000
emp.project = "Machine Learning"
emp.show()
emp.work()
```

Protected data Members

Data variables can be accessed from only inside the base class and its derived classes. We declare protected numbers by prefixing with single underscore _.

```
class Company:
   def __init__(self):
      self._project = "NLP"
class Employee(Company):
   def __init__(self, name):
      self.name = name
      Company.__init__(self)
   def show(self):
      print("Employee name :", self.name)
      print("Working on project :", self._project)
c = Employee("Kaushik")
c.show()
print('Project:', c._project)
```

Private data Members

Data variables which can be accessed from only inside the base class but not from and its derived classes and outside of the class. We declare protected numbers by prefixing with double underscore __.

```
class Employee:
   def __init__(self, name, salary, project):
      self.__name = name
      self.salary = salary
      self.project = project
   def show(self):
      print("Name: ", self.__name, 'Salary:', self.salary)
e1 = Employee("Eyk", 9000, "Project-Z")
e1.show()
e1.__name()
```

Abstraction

- Process of hiding the unnecessary information and providing the required info to the user. Example: Hotstar App, Car, Remote
- Achieved by using Abstract classes and methods

Abstract Classes: Class of the Classes (Blueprint of the other classes)

A class can be referred to as abstract class if it has one or more abstract method.

Abstract Methods: Function that has declaration but not any implementation.

- Abstract classes are used when we want to provide common interface for all implementations of component.

Python does not provide abstract class by default; we create by importing **abc** module.

abc module – gives infrastructure for defining abstract base class

@abstractmethod - defines an abstract class method

- Methods whose definition would change as per different class/subclass known as Abstract Class.

```
from abc import ABC, abstractmethod
class AbstractClassName(ABC):
    @abstractmethod
    def abstract_method_name(self):
     #Empty body
pass
```

Concrete methods

- Methods whose definition/implementation remains same for all subclasses and those should be declared in abstract class itself.

```
from abc import ABC, abstractmethod
class Parent(ABC):
   @abstractmethod
  def abs_fn(self): #is supposed to have different implementation in child
classes
     pass
```

Practice Example:

```
from abc import ABC,abstractmethod
class OTT(ABC):
    def movies(self):
        print("Can watch movies")
    def webshows(self):
        print("Can watch webshows")
    @abstractmethod
    def rest_content(self):
        pass

class Netflix(OTT):
    def rest_content(self):
        print("Only movies & webshows acn be watched in Netflix")

class Hotstar(OTT):
    def movies(self):
        print("Sports along with Movies & Webshows can be watched in
Hotstar")

class SonyLiv(OTT):
        pass

n = Netflix()
n.movies()
n.webshows()
n.rest_content()
s = SonyLiv()
s.movies()
s.webshows()
s.rest_content()
```

- A subclass must implement all abstract methods defined in the parent class otherwise it results in an error.
- Always provide an implementation of the abstract method in the child class even when implementation is given in the abstract class.