Encapsulation:(Data Binding/Grouping)
We can restrict the access to the variables and methods is called as Encapsulation.
OR
The process of wrapping up variables and methods into a single entity is known as Encapsulation
OR
It can prevent the data from being modified by accidently.
OR
It describes the idea of wrapping data and the methods that work on data within one unit.

Example:RealTimeUseCase
Employees==>Diff Deparments
Admin Department ==> These Employee records can access that Department Head only
HR Deparment ==> These Employee records can access that Department Head only
Admin Department Employee Details Unable to access HR Department is called Encapsulation.

Private & Protected members in Python
1 When the attributes of an object can only be accessed inside the class
2 Python use two underscores for private, single underscore for protect
3. We can not access private, protect attributes or methods outside the class

Summary of Encapsulation:
1 Encapsulation provide security by hiding the data from outside world
2 In Python we achieve encapsulation through Private & Protected access members
3 Private by leading two underscores, Protected by leading single underscore
4 Encapsulation ensures data protection & avoids the access of data accidentally

Example:
```
class Employee():
    def __init__(self,name,age):
        self.name=name
        self.age=age
    def DisplayDetails(self):
        print(self.name)
        print(self.age)

EE=Employee("SARA",30)
print(EE.name)
print(EE.age)
EE.DisplayDetails()
```

Example:
```
class DataBinding():
    def __init__(self):
        self.x="It is Public Access"
        print(self.x)

DD=DataBinding()
print(DD.x)

Example:
class DataBinding():
    def __init__(self):
        self._x="It is Protected Access"
        print(self._x)

DD=DataBinding()
print(DD._x)

Example:
class DataBinding():
    def __init__(self):
        self.__x="It is Private Access"
        print(self.__x)

DD=DataBinding()
print(DD.__x)#AttributeError:

Example:
class Banking():
#Public Method
    def PublicMethod(self):
        print("Public Method")
#Protected Method
    def _ProtectMethod(self):
        print("Protect Method")
#Private Method
    def __PrivateMethod(self):
        print("Private Method")

#CreatingInstance
BB=Banking()
BB.PublicMethod()
BB._ProtectMethod()
BB.__PrivateMethod()
```

NOTE:
AttributeError: 'Banking' object has no attribute '__PrivateMethod'

NOTE:
But we can access private methods & Private attributes using One UnderScore with Class Name (Protected Class Name)

```
Example:
class Banking():
    def PublicMethod(self):
        print("Public Method")
    def _ProtectMethod(self):
        print("Protect Method")
    def __PrivateMethod(self):
        print("Private Method")
```

```
#CreatingInstance
BB=Banking()
BB.PublicMethod()
BB._ProtectMethod()
BB._Banking__PrivateMethod()
```

Example:
```
class DataBinding():
    def __init__(self):
        self.__x="It is Private Access"
        print(self.__x)

DD=DataBinding()
print(DD._DataBinding__x)
```

Example:
```
class car():
    def __init__(self):
        self.__updatesoftware()
    def drive(self):
        print("I am Driving a Car")
    def __updatesoftware(self):
        print("Car Software Updated Successfully")

#Creating Instance  or Object
blackcar=car()
blackcar.drive()
```

O/P:
```
Car Software Updated Successfully
I am Driving a Car
```

Example:
```
class car:
    #def __init__(self):
        #self.__updatesoftware()
    def drive(self):
        print("Driving")
    def __updatesoftware(self):
        print("Update Software")

blackcar=car()
blackcar.drive()
blackcar.__updatesoftware()
```

```
AttributeError: 'car' object has no attribute '__updatesoftware'
```

Private Variables in PYTHON:
Private variables can be modified inside the class methods. We can not modify private variables out side the class.

Example:
```
class car:
    __maxspeed=0
    __name=""
    def __init__(self):
        self.__maxspeed=300
        self.name="SuperCar"
    def drive(self):
        print("Driving")
        print(self.__maxspeed)
    def setspeed(self,speed):
        self.__maxspeed=speed
        print(self.__maxspeed)

bluecar=car()
bluecar.drive()
bluecar.setspeed(100)
```

Protected Variables
Example:

```
class Shape():
    _length=10
    _width=20

class Circle(Shape):
    def __init__(self):
        print(self._length)
        print(self._width)

CC=Circle()
```

Example:

```
class Shape():
    _length=10
    _width=20

class Circle(Shape):
    def __init__(self):
        print(self._length)
        print(self._width)

CC=Circle()
print(CC.length)
print(CC.width)
```

NOTE:
1 AttributeError: 'Circle' object has no attribute 'length'
2 Python doesn't have real protected methods
3 A leading single underline always indicates protected it is just convention, still we can access

Example:

```
class Employee():
    Company="NareshIT"#Public variable
    __Group="IT-Training"#Private Variable

    def getDetails(self):
        self.Name=input("Name: ")
        self.__Salary=input(self.Name+" 's "+ "Salary: ")
        self.Programming=input("Programming: ")

    def Display(self):
        print("Name: ",self.Name)
        print("Salary: ",self.Salary)
        print("Programming Name: ",self.Programming)

    def RevisedSalary(self):
        print("Existing Salary: ",self.__Salary)
        self.__Salary=60000
        print("Revised Salary is: ",self.__Salary)

EE=Employee()
EE.getDetails()
EE.RevisedSalary()
```

Example:

```
class Employee():
    Company="NareshIT"#Public variable
    __Group="IT-Training"#Private Variable

    def getDetails(self):
        self.Name=input("Name: ")
        self.__Salary=input(self.Name+" 's "+ "Salary: ")
        self.Programming=input("Programming: ")

    def Display(self):
        print("Name: ",self.Name)
        print("Salary: ",self.Salary)
        print("Programming Name: ",self.Programming)

    def RevisedSalary(self):
        print("Existing Salary: ",self.__Salary)
        self.__Salary=60000
        print("Revised Salary is: ",self.__Salary)

EE=Employee()
EE.getDetails()
EE.RevisedSalary()
EE.Display()
```

NOTE:
O/P: AttributeError: 'Employee' object has no attribute 'Salary'
We can't access private resources outside the class
We can access in the existing class with help of two underscore at the leading

Example:

```
class Edpresso:
   def __init__(self, name, project):
      # public variable
      self.name = name
      # private variable
      self.__project = project

# creating an instance of the class Sample
edp = Edpresso('TeamRajuSir', 3)

# accessing public variable name
print("Name:",edp.name)
```

```
# accessing private variable __project using
# _Edpresso__project name
print("Project:",edp._Edpresso__project)
```

Encapsulation:
Suppose there is a tree. Now a tree can have its components like root, stem, branches, leaves, flowers and fruits. It has some functionalities like Photosynthesis. But in a single unit we call it a tree. In same way encapsulation is a characteristic to bind data members and functions in single unit.

PYTHON ABSTRACTION OR DATA HIDING
Abstraction is used to hide internal details & show only functionalities. It is blue print for other classes.
OR
Abstraction is used to hide the internal functionality of the function from the users.
OR
Hiding all implemenations, show only essential parts. Here, using inheritance concept.

RealTimeUseCase:
Suppose you are going to an ATM to withdraw money. You simply insert your card and click some buttons and get the money. You don't know what is happening internally on press of these buttons. Basically you are hiding unnecessary information from user.

Abstrac Class
A class that consists of one or more abstract methods is called the abstract class. An abstract class can be useful when we are designing large functions. An abstract class is also helpful to provide the standard interface for different implementations of components.

Remember Points:
1 We are unable to create instance for abstract class.
2. We want to create instance, then we must convert into concrete class.
3. Concrete class means no abstract methods.
4. Abstract methods we can override through inheritance.
5. Abstract class is base class, concrete class is child class, we can override methods

Syntax:
```
from abc import ABC
abc :Abstract Base Class
abc : Module Name, ABC ==>Class Name
```

Abstract Method:
Only function call with empty definition.Abstract class can be inherited by the subclass and abstract method gets its definition in the subclass.

Syntax:
```
@abstractmethod
```

Use of Abstraction:
Abstraction classes are meant to be the blueprint of the other class.

An abstract class can be useful when we are designing large functions.
An abstract class is also helpful to provide the standard interface
for different implementations of components.