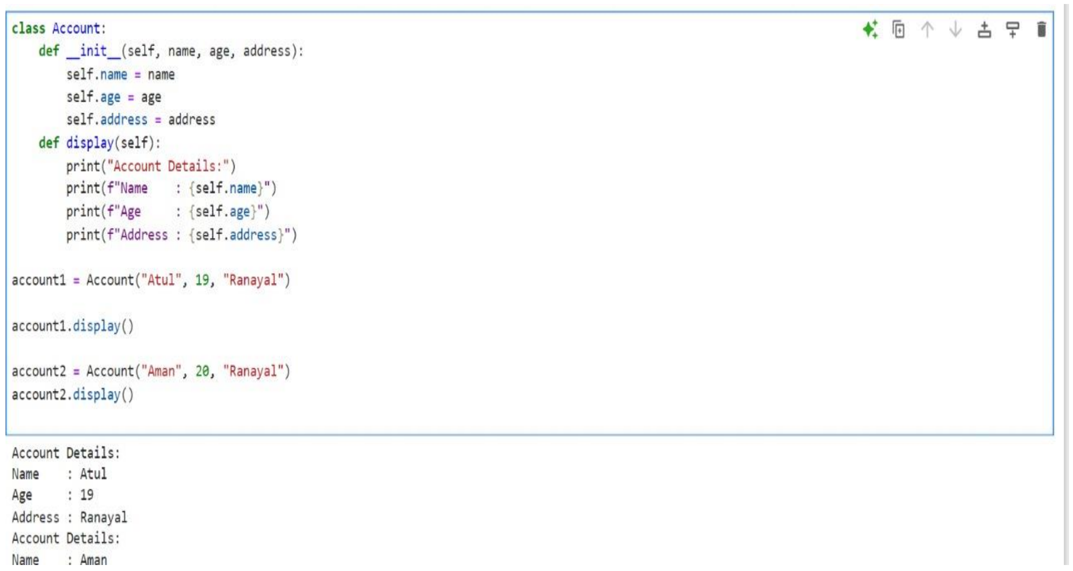
Question no.2] WAP to create a class 'account' which contains name ,age,address. Also define a display function?

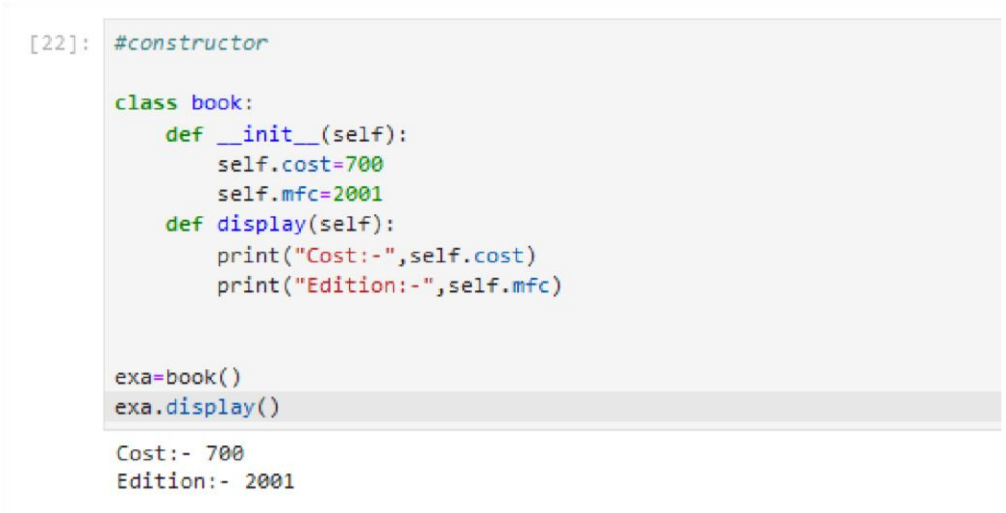


```
class Account:
    def __init__(self, name, age, address):
        self.name = name
        self.age = age
        self.address = address
    def display(self):
        print("Account Details:")
        print(f"Name    : {self.name}")
        print(f"Age     : {self.age}")
        print(f"Address : {self.address}")

account1 = Account("Atul", 19, "Ranayal")

account1.display()

account2 = Account("Aman", 20, "Ranayal")
account2.display()
```

```
Account Details:
Name    : Atul
Age     : 19
Address : Ranayal
Account Details:
Name    : Aman
```

Summury: This code demonstrates the use of a class to represent an account. The `Account` class defines attributes like `name`, `age`, and `address`, initialized through the constructor. The `display()` method allows controlled access to these attributes and prints them in a formatted way.

Queston.3] WAP to implement a constructor

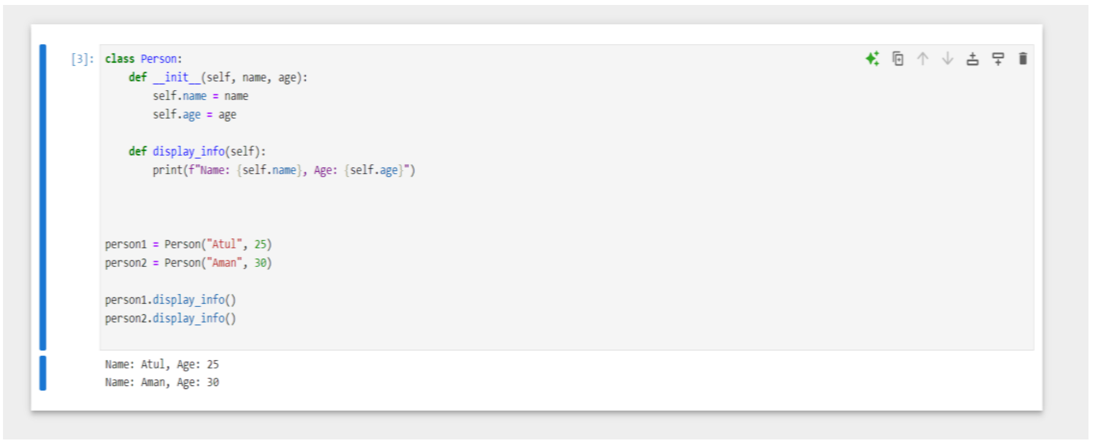


```
[22]: #constructor

      class book:
          def __init__(self):
              self.cost=700
              self.mfc=2001
          def display(self):
              print("Cost:-",self.cost)
              print("Edition:-",self.mfc)


      exa=book()
      exa.display()
```

```
Cost:- 700
Edition:- 2001
```

Summury:. The constructor `__init__` is used to initialize object attributes when a new object is created. This is an essential OOP concept for setting up an object with initial values. The code demonstrates the automatic calling of the constructor during object instantiation,

Queston.4] WAP to implement a constructor and initialize attributes

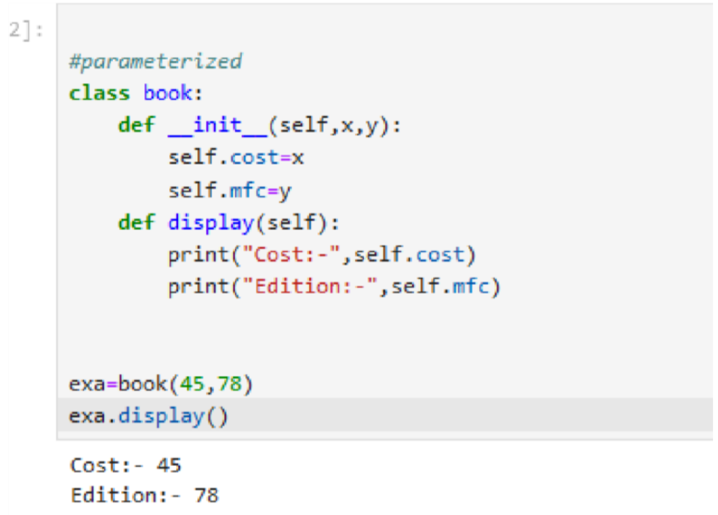


```
[3]: class Person:
         def __init__(self, name, age):
             self.name = name
             self.age = age

         def display_info(self):
             print(f"Name: {self.name}, Age: {self.age}")



     person1 = Person("Atul", 25)
     person2 = Person("Aman", 30)

     person1.display_info()
     person2.display_info()
```

```
Name: Atul, Age: 25
Name: Aman, Age: 30
```

<u>Summury:</u> The constructor `__init__` initializes the attributes of the class when a new object is created. This allows different objects to have unique attribute values. The code demonstrates how the constructor provides a convenient way to set initial values when creating instances of a class.

<u>Queston.5]</u> WAP a program to implement parameterized constructor

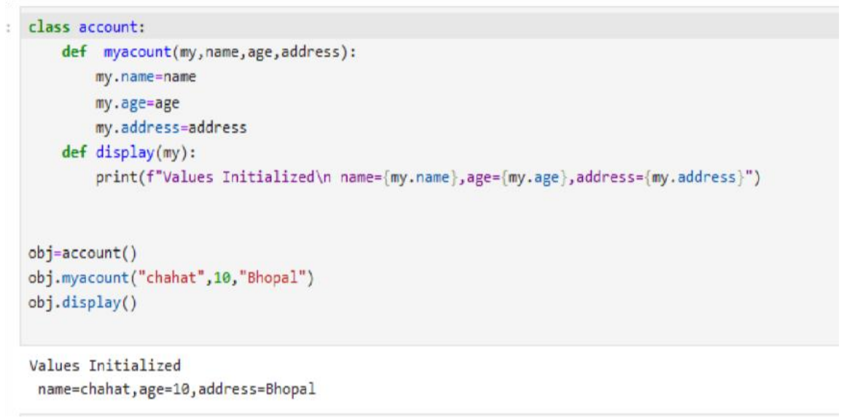


```
2]:
    #parameterized
    class book:
        def __init__(self,x,y):
            self.cost=x
            self.mfc=y
        def display(self):
            print("Cost:-",self.cost)
            print("Edition:-",self.mfc)


    exa=book(45,78)
    exa.display()
```

```
Cost:- 45
Edition:- 78
```

<u>Summury</u> : A parameterized constructor is used to initialize attributes with values passed during object creation. This allows for more flexible object creation, where each object can be initialized with unique values. The code demonstrates how to pass arguments to the constructor during object instantiation to initialize attributes.

<u>Queston.6]</u> WAP a program to implement constructor with using any reference variable?

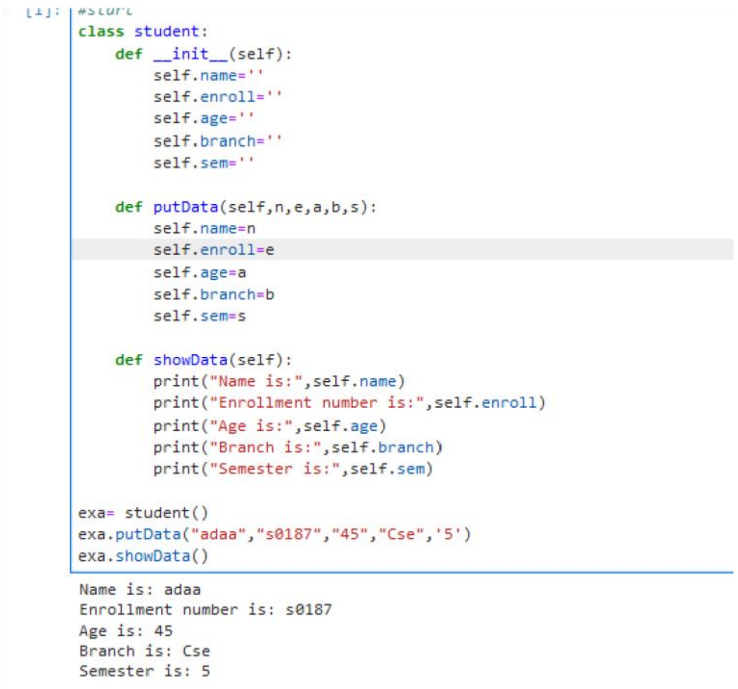


```
class account:
    def  myacount(my,name,age,address):
        my.name=name
        my.age=age
        my.address=address
    def display(my):
        print(f"Values Initialized\n name={my.name},age={my.age},address={my.address}")


obj=account()
obj.myacount("chahat",10,"Bhopal")
obj.display()
```

```
Values Initialized
 name=chahat,age=10,address=Bhopal
```

Summury : In this case, the constructor is executed even without storing the object in a variable. The code demonstrates that the constructor still performs its function and initializes the object's attributes when called, regardless of whether the object is referenced later.

Queston.7] WAP to create a class student with data member name,enroll num,age branch and sem, and define function put data and display the data inserted?



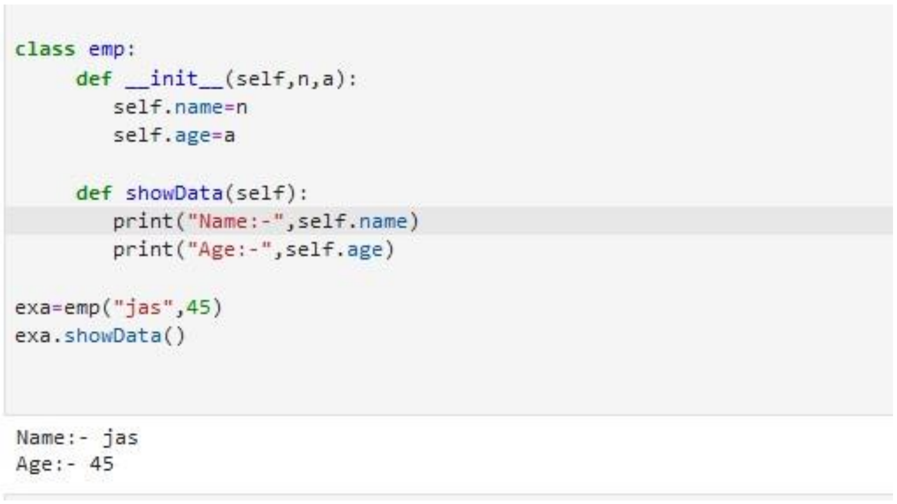
```
#start
class student:
    def __init__(self):
        self.name=''
        self.enroll=''
        self.age=''
        self.branch=''
        self.sem=''

    def putData(self,n,e,a,b,s):
        self.name=n
        self.enroll=e
        self.age=a
        self.branch=b
        self.sem=s

    def showData(self):
        print("Name is:",self.name)
        print("Enrollment number is:",self.enroll)
        print("Age is:",self.age)
        print("Branch is:",self.branch)
        print("Semester is:",self.sem)

exa= student()
exa.putData("adaa","s0187","45","Cse",'5')
exa.showData()
```

```
Name is: adaa
Enrollment number is: s0187
Age is: 45
Branch is: Cse
Semester is: 5
```

Summury : The `Student` class manages student data through attributes like `name`, `enroll_number`, `age`, `branch`, and `semester`. Methods `put_data()` and `display()` are defined to set and display these attributes. This demonstrates how OOP allows organizing student data and performing operations like data input and display within a structured class framework.

Queston.8] WAP to Define a class Emp with constructor initialize member variables and define function to show employee data.?

Name: Atul Chandravanshi   Roll no: 0187CS221066   Sec: CSE-1

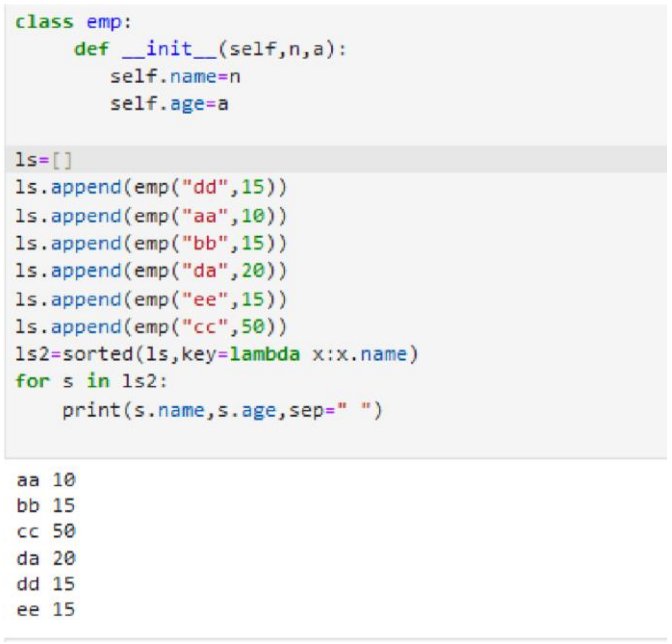


```
class emp:
     def __init__(self,n,a):
        self.name=n
        self.age=a

     def showData(self):
        print("Name:-",self.name)
        print("Age:-",self.age)

exa=emp("jas",45)
exa.showData()
```

```
Name:- jas
Age:- 45
```

Summury : The `Emp` class is used to represent an employee with attributes like `name` and `id`. The constructor initializes these attributes, and the `show()` method provides controlled access to display the employee's details. This illustrates how classes can encapsulate employee information and methods for data management.

Queston.9] Create a class employee and create a list of employee , display the list in sorted order acc to names?



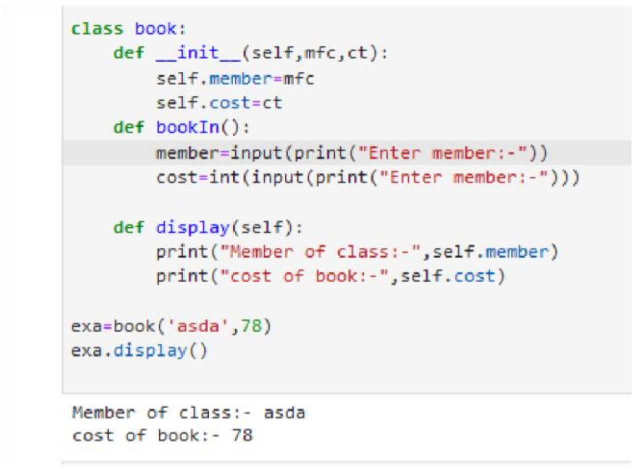
```
class emp:
     def __init__(self,n,a):
        self.name=n
        self.age=a

ls=[]
ls.append(emp("dd",15))
ls.append(emp("aa",10))
ls.append(emp("bb",15))
ls.append(emp("da",20))
ls.append(emp("ee",15))
ls.append(emp("cc",50))
ls2=sorted(ls,key=lambda x:x.name)
for s in ls2:
    print(s.name,s.age,sep=" ")
```

```
aa 10
bb 15
cc 50
da 20
dd 15
ee 15
```

Summury : This code demonstrates how to use classes to manage a collection of objects (employees in this case). The `Employee` class defines attributes like `name` and `age`, and a list of employee objects is created. The list is then sorted by the `name` attribute, showcasing how OOP can handle and manipulate collections of objects.

Queston.10] Define class book with members of class as book name, cost, Also define a input function and one to read a books details.

Name: Atul Chandravanshi Roll no: 0187CS221066 Sec: CSE-1

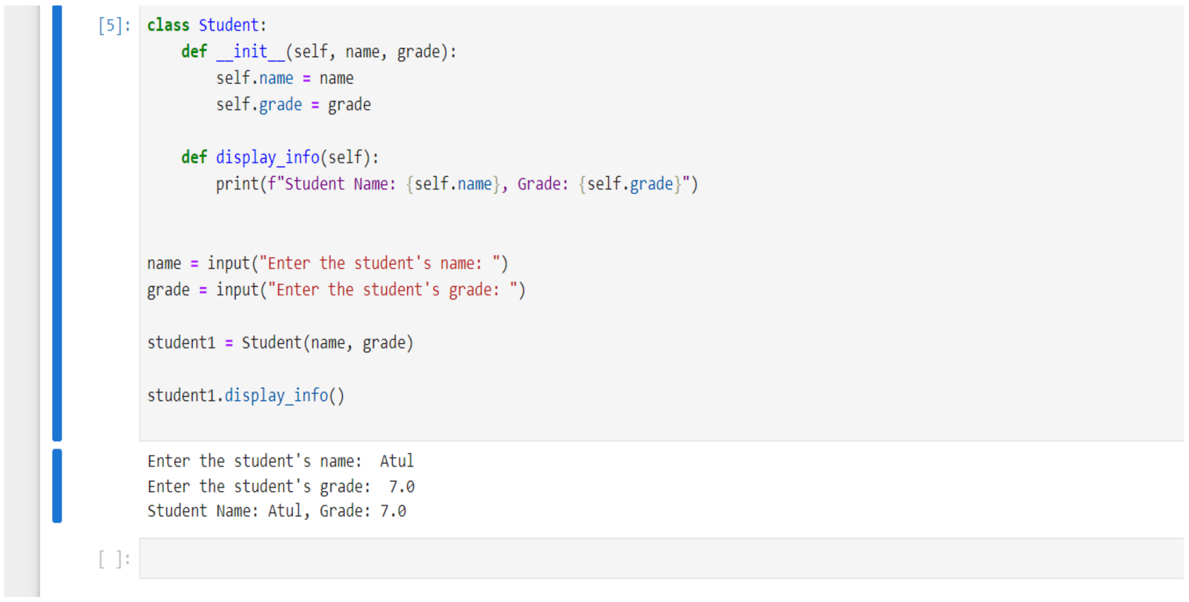


```
class book:
    def __init__(self,mfc,ct):
        self.member=mfc
        self.cost=ct
    def bookIn():
        member=input(print("Enter member:-"))
        cost=int(input(print("Enter member:-")))

    def display(self):
        print("Member of class:-",self.member)
        print("cost of book:-",self.cost)

exa=book('asda',78)
exa.display()
```

```
Member of class:- asda
cost of book:- 78
```

Summury : The `Book` class defines attributes like `name` and `cost`, and includes methods for inputting and displaying book details. This demonstrates encapsulation and the use of methods to handle class data. It shows how OOP structures can be used to manage and display book information in a controlled manner.

Queston.11] WAP to take input from the user for class attributes.



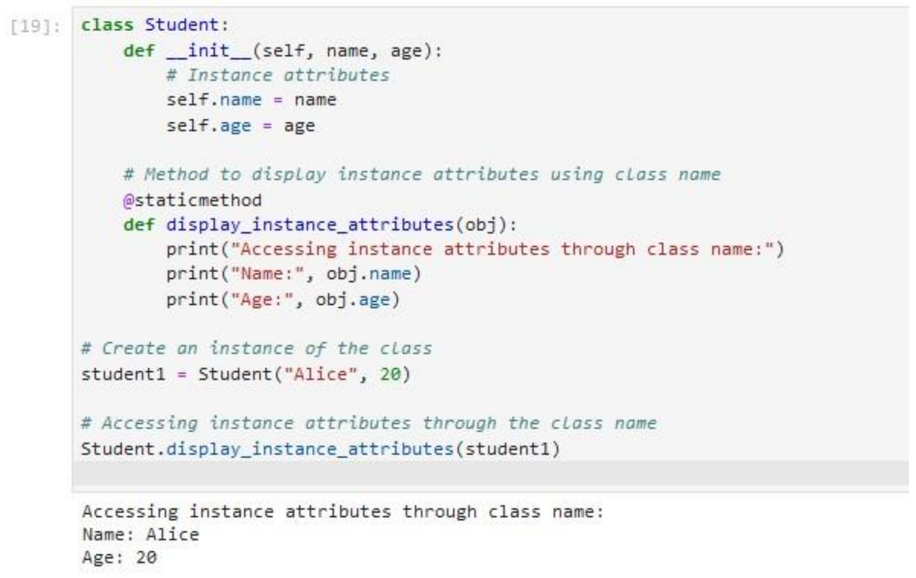
```
[5]: class Student:
         def __init__(self, name, grade):
             self.name = name
             self.grade = grade

         def display_info(self):
             print(f"Student Name: {self.name}, Grade: {self.grade}")


     name = input("Enter the student's name: ")
     grade = input("Enter the student's grade: ")

     student1 = Student(name, grade)

     student1.display_info()
```

```
Enter the student's name:  Atul
Enter the student's grade:  7.0
Student Name: Atul, Grade: 7.0
```

```
[ ]:
```

Summury : This code shows how user input can be used to set class attributes. The `input()` function allows for dynamic attribute initialization, demonstrating how OOP can interact with external inputs to initialize object data, providing flexibility in creating object instances.

Queston.12] WAP to access instance attributes through class name?

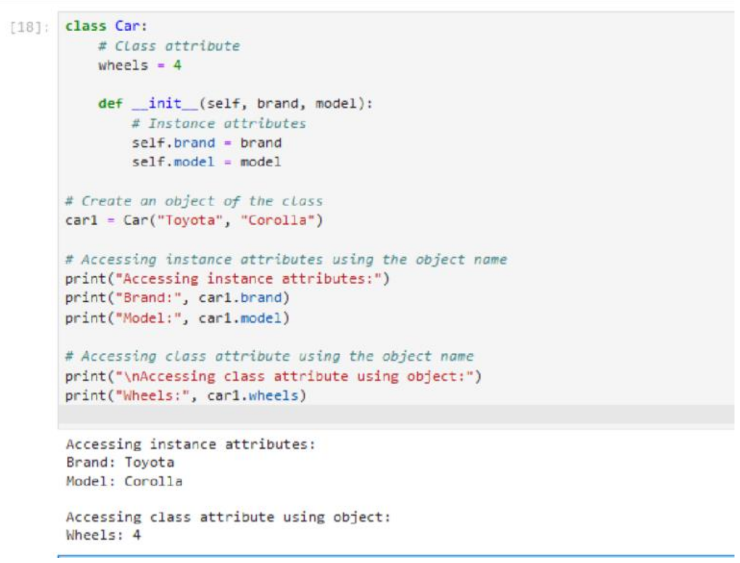


```
[19]: class Student:
          def __init__(self, name, age):
              # Instance attributes
              self.name = name
              self.age = age

          # Method to display instance attributes using class name
          @staticmethod
          def display_instance_attributes(obj):
              print("Accessing instance attributes through class name:")
              print("Name:", obj.name)
              print("Age:", obj.age)

      # Create an instance of the class
      student1 = Student("Alice", 20)

      # Accessing instance attributes through the class name
      Student.display_instance_attributes(student1)
```

```
Accessing instance attributes through class name:
Name: Alice
Age: 20
```

<u>Summury</u>: Attempting to access instance attributes through the class name demonstrates that instance attributes belong to objects, not the class itself. This code highlights the concept of objectoriented data encapsulation and how attributes should be accessed through objects instead of class names.

<u>Queston.13]</u> WAP to access instance attribute,class attribute using objectname?

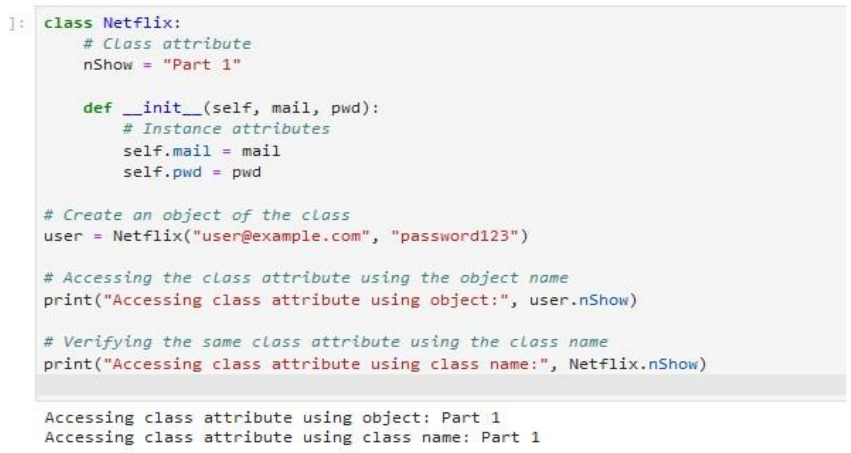


```
[18]: class Car:
          # Class attribute
          wheels = 4

          def __init__(self, brand, model):
              # Instance attributes
              self.brand = brand
              self.model = model

      # Create an object of the class
      car1 = Car("Toyota", "Corolla")

      # Accessing instance attributes using the object name
      print("Accessing instance attributes:")
      print("Brand:", car1.brand)
      print("Model:", car1.model)

      # Accessing class attribute using the object name
      print("\nAccessing class attribute using object:")
      print("Wheels:", car1.wheels)
```

```
Accessing instance attributes:
Brand: Toyota
Model: Corolla

Accessing class attribute using object:
Wheels: 4
```

<u>Summury</u> : In this case, both instance and class attributes can be accessed via the object. This demonstrates the difference between class-level and instance-level data in OOP, and how objects can interact with both types of attributes.

<u>Queston.14]</u> WAP to access class attribute accessed using an objectname?

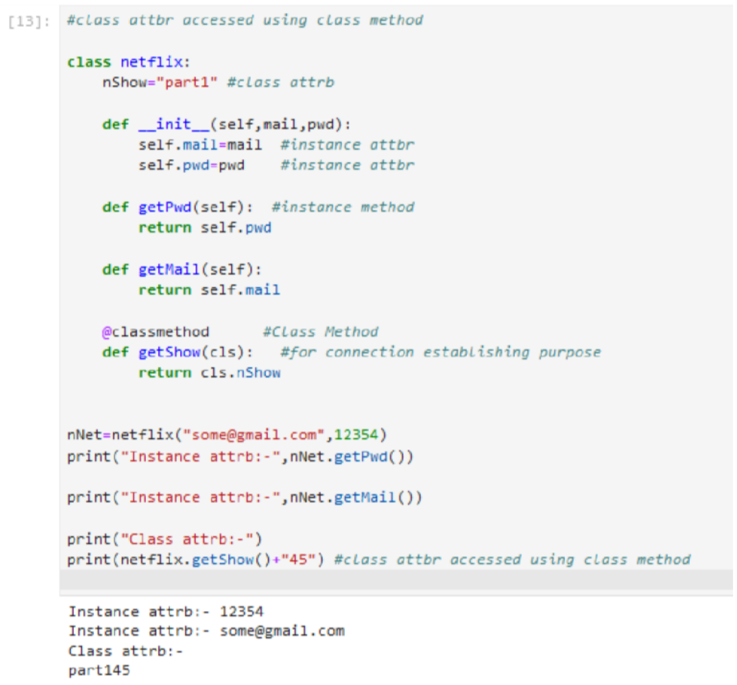


```
]: class Netflix:
       # Class attribute
       nShow = "Part 1"

       def __init__(self, mail, pwd):
           # Instance attributes
           self.mail = mail
           self.pwd = pwd

   # Create an object of the class
   user = Netflix("user@example.com", "password123")

   # Accessing the class attribute using the object name
   print("Accessing class attribute using object:", user.nShow)

   # Verifying the same class attribute using the class name
   print("Accessing class attribute using class name:", Netflix.nShow)
```

```
Accessing class attribute using object: Part 1
Accessing class attribute using class name: Part 1
```

Summury : The code demonstrates that class attributes can be accessed through an object, although it is more common to use the class name. This emphasizes that attributes in OOP can be accessed through both the class and its instances, showcasing the flexibility in how data can be retrieved.

Queston.15] WAP to define an instance method

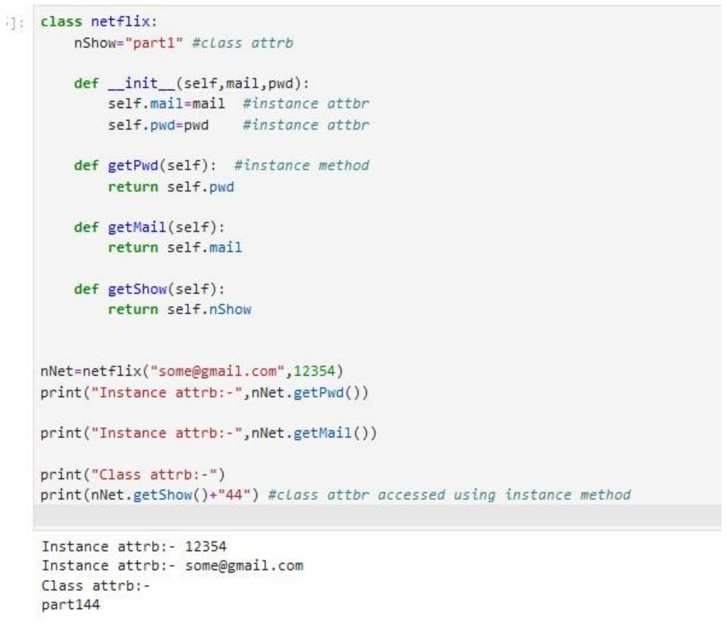


```
[13]: #class attbr accessed using class method

      class netflix:
          nShow="part1" #class attrb

          def __init__(self,mail,pwd):
              self.mail=mail  #instance attbr
              self.pwd=pwd    #instance attbr

          def getPwd(self):  #instance method
              return self.pwd

          def getMail(self):
              return self.mail

          @classmethod      #Class Method
          def getShow(cls):   #for connection establishing purpose
              return cls.nShow


      nNet=netflix("some@gmail.com",12354)
      print("Instance attrb:-",nNet.getPwd())

      print("Instance attrb:-",nNet.getMail())

      print("Class attrb:-")
      print(netflix.getShow()+"45") #class attbr accessed using class method
```

```
Instance attrb:- 12354
Instance attrb:- some@gmail.com
Class attrb:-
part145
```

Summury : An instance method operates on object-specific data and is accessed through an instance. The code illustrates how instance methods can manipulate or return object data, which is a fundamental concept in OOP for managing object behavior.

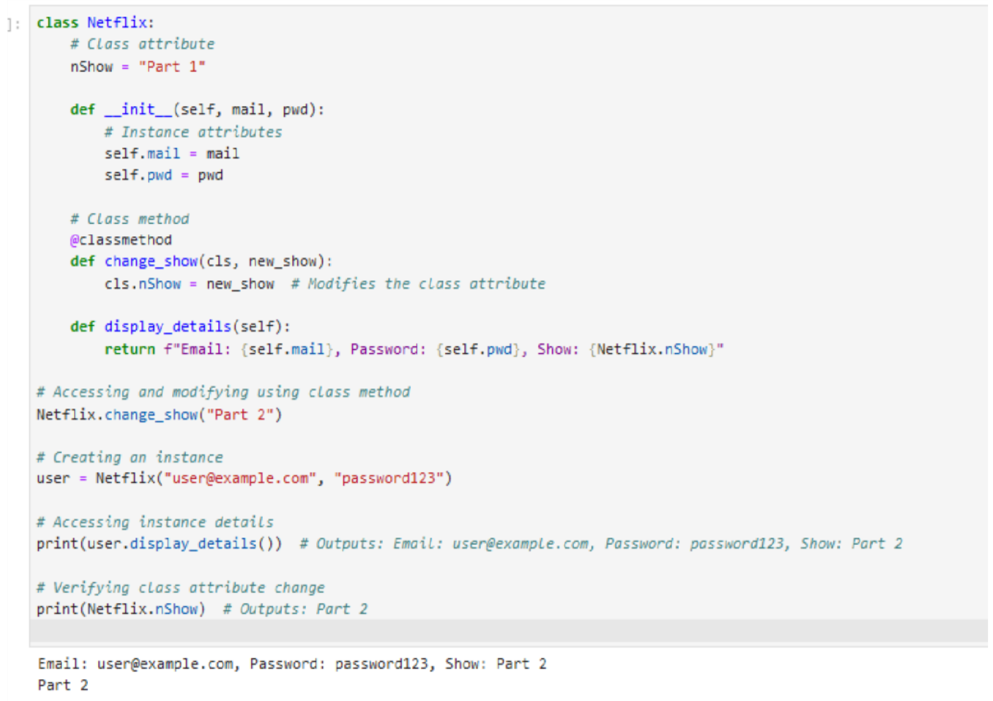
Q**ueston.16]** WAP to define a instance method and access instance and class attribute?


```
class netflix:
    nShow="part1" #class attrb

    def __init__(self,mail,pwd):
        self.mail=mail  #instance attbr
        self.pwd=pwd    #instance attbr

    def getPwd(self):  #instance method
        return self.pwd

    def getMail(self):
        return self.mail

    def getShow(self):
        return self.nShow


nNet=netflix("some@gmail.com",12354)
print("Instance attrb:-",nNet.getPwd())

print("Instance attrb:-",nNet.getMail())

print("Class attrb:-")
print(nNet.getShow()+"44") #class attbr accessed using instance method
```

```
Instance attrb:- 12354
Instance attrb:- some@gmail.com
Class attrb:-
part144
```

Summury : Instance methods can access both instance and class attributes, demonstrating how methods defined within a class can work with both object-specific and class-level data. This shows the flexibility of OOP in accessing and modifying various types of data within a class.

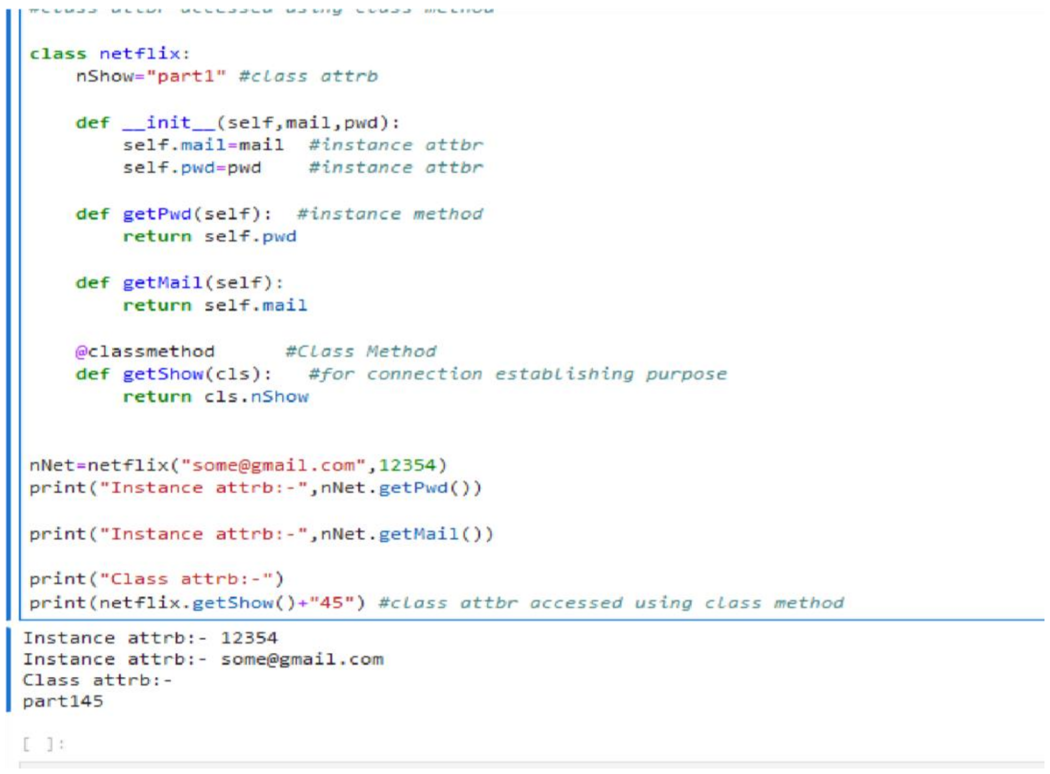
Queston.17] WAP to define a class method ?


```
class Netflix:
    # Class attribute
    nShow = "Part 1"

    def __init__(self, mail, pwd):
        # Instance attributes
        self.mail = mail
        self.pwd = pwd

    # Class method
    @classmethod
    def change_show(cls, new_show):
        cls.nShow = new_show  # Modifies the class attribute

    def display_details(self):
        return f"Email: {self.mail}, Password: {self.pwd}, Show: {Netflix.nShow}"

# Accessing and modifying using class method
Netflix.change_show("Part 2")

# Creating an instance
user = Netflix("user@example.com", "password123")

# Accessing instance details
print(user.display_details())  # Outputs: Email: user@example.com, Password: password123, Show: Part 2

# Verifying class attribute change
print(Netflix.nShow)  # Outputs: Part 2
```

```
Email: user@example.com, Password: password123, Show: Part 2
Part 2
```

Summury : A class method operates on the class itself rather than on instance-specific data. This code demonstrates how class methods are used to interact with class-level attributes and methods, emphasizing the concept of class-wide data manipulation in OOP.

Queston.18] WAP to define a class method to access class attribute.



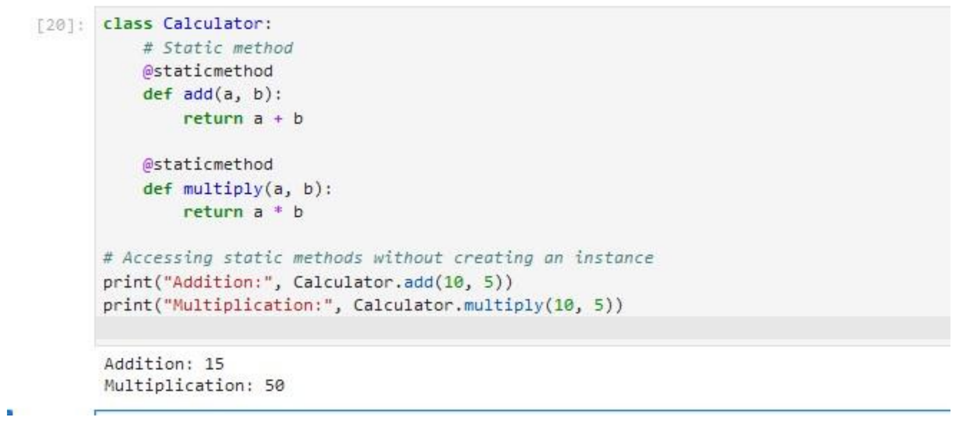
```
class netflix:
    nShow="part1" #class attrb

    def __init__(self,mail,pwd):
        self.mail=mail  #instance attbr
        self.pwd=pwd    #instance attbr

    def getPwd(self):  #instance method
        return self.pwd

    def getMail(self):
        return self.mail

    @classmethod      #Class Method
    def getShow(cls):   #for connection establishing purpose
        return cls.nShow


nNet=netflix("some@gmail.com",12354)
print("Instance attrb:-",nNet.getPwd())

print("Instance attrb:-",nNet.getMail())

print("Class attrb:-")
print(netflix.getShow()+"45") #class attbr accessed using class method
```

```
Instance attrb:- 12354
Instance attrb:- some@gmail.com
Class attrb:-
part145
```

```
[ ]:
```

Summury : This code defines a class method that accesses and displays class attributes. It highlights how class methods can interact with data shared across all instances of the class, which is a key feature of OOP for managing global data within a class.

Queston.19] WAP to implement static method?

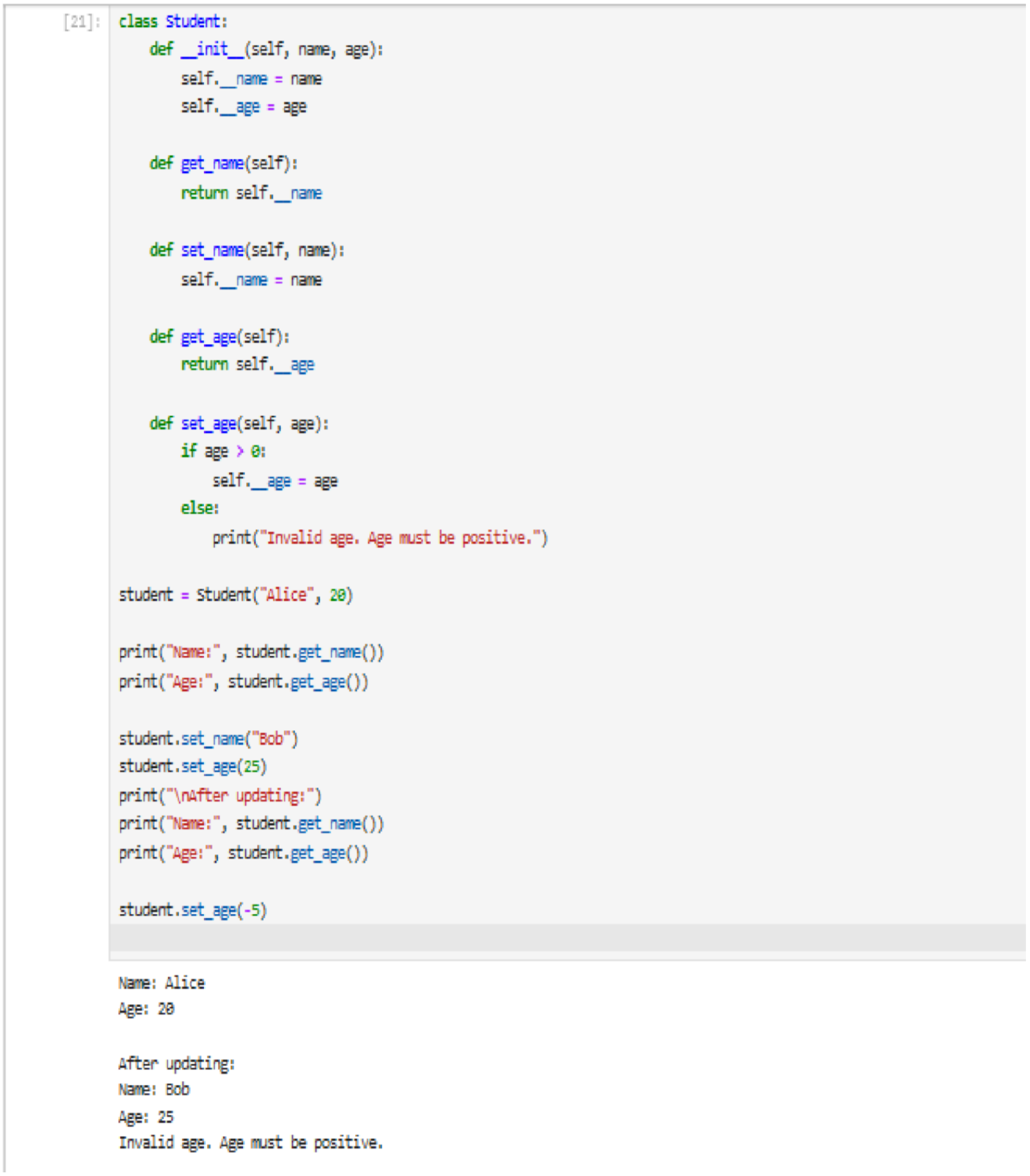


```
[20]: class Calculator:
          # Static method
          @staticmethod
          def add(a, b):
              return a + b

          @staticmethod
          def multiply(a, b):
              return a * b

      # Accessing static methods without creating an instance
      print("Addition:", Calculator.add(10, 5))
      print("Multiplication:", Calculator.multiply(10, 5))
```

```
Addition: 15
Multiplication: 50
```

Summury : Static methods are defined using the `@staticmethod` decorator and are independent of instance or class data. This code shows how static methods operate without needing a reference to the instance or class, making them useful for utility functions in OOP.

Queston.20] WAP to implement encapsulation?



```
[21]: class Student:
          def __init__(self, name, age):
              self.__name = name
              self.__age = age

          def get_name(self):
              return self.__name

          def set_name(self, name):
              self.__name = name

          def get_age(self):
              return self.__age

          def set_age(self, age):
              if age > 0:
                  self.__age = age
              else:
                  print("Invalid age. Age must be positive.")

      student = Student("Alice", 20)

      print("Name:", student.get_name())
      print("Age:", student.get_age())

      student.set_name("Bob")
      student.set_age(25)
      print("\nAfter updating:")
      print("Name:", student.get_name())
      print("Age:", student.get_age())

      student.set_age(-5)
```

```
Name: Alice
Age: 20

After updating:
Name: Bob
Age: 25
Invalid age. Age must be positive.
```

Summury : Encapsulation is demonstrated by making attributes private (using __) and providing getter and setter methods to access them. This code shows how OOP allows for data hiding, ensuring that object data is only accessible through controlled interfaces.

Name: _Atul **Roll no: 0187CS221066** **Sec: CSE-1**