o9 Feb Ass

February 20, 2023

[ ]:
```
Q1, Create a vehicle class with an init method having instance variables as␣
 ↪name_of_vehicle, max_speed
and average_of_vehicle.
```

ANS -

[1]:
```
class vehicle:
    def __init__(self , name , max_speed , avg_of_vehicle):

        self.name = name
        self.max_speed = max_speed
        self.avg_of_vehicle = avg_of_vehicle

        def vehicle_details(self):
            print(self.name , self.max_speed , self.avg_of_vehicle)
```

[2]:
```
car = vehicle("mercedes" , 318 , 15)
```

[3]:
```
car.name
```

[3]:
```
'mercedes'
```

[4]:
```
car.max_speed
```

[4]:
```
318
```

[5]:
```
car.avg_of_vehicle
```

[5]:
```
15
```

[ ]:
```

```

[ ]:
```
Q2. Create a child class car from the vehicle class created in Que 1, which␣
 ↪will inherit the vehicle class.
Create a method named seating_capacity which takes capacity as an argument and␣
 ↪returns the name of
the vehicle and its seating capacity.
```

ANS -

[ ]:
```
Q3. What is multiple inheritance? Write a python code to demonstrate multiple␣
↪inheritance.
```

ANS -

When a class is derived from more than one base class it is called multiple Inheritance. The derived class inherits all the features of the base case.

[10]:
```
class class1:
    def test_class1(self):
        print("this is my class1")
```

[11]:
```
class class2:
    def test_class2(self):
        print("this is my class 2")
```

[12]:
```
class class3 (class1 , class2) :
    pass
```

[13]:
```
obj_class3 = class3()
```

[14]:
```
obj_class3.test_class1()
```

```
this is my class1
```

[15]:
```
obj_class3.test_class2()
```

```
this is my class 2
```

[ ]:
```

```

[ ]:
```
Q4. What are getter and setter in python? Create a class and create a getter␣
↪and a setter method in this
class.
```

ANS -

Getters :- These are the methods used in Object-Oriented Programming(OOPS) which helps to access the private attributes from a class. Setters :- These are the methods used in OOPS feature which helps to set the value to private attributes in a class.

[16]:
```
class pwskills:

    def __init__(self , course_price , course_name):
        self.__course_price = course_price
        self.course_name = course_name
```

[17]:
```
pw = pwskills(3500 , "Data Science Master")
```

[18]:
```
pw.course_name
```

[18]:
```
'Data Science Master'
```

[19]:
```
class pwskills:

    def __init__(self , course_price , course_name):
        self.__course_price = course_price
        self.course_name = course_name

    @property
    def course_price_access(self):
        return self.__course_price
```

[20]:
```
pw = pwskills(3500 , "Data Science Master")
```

[21]:
```
pw.course_price_access
```

[21]:
```
3500
```

[ ]:
```

```

[22]:
```
class pwskills:

    def __init__(self , course_price , course_name):
        self.__course_price = course_price
        self.course_name = course_name

    @property
    def course_price_access(self):
        return self.__course_price

    @course_price_access.setter
    def course_price_set(self , price):
        if price <= 3500:
            pass
        else:
            self.__course_price = price
```

[23]:
```
pw = pwskills(3500 , "Data Science Master")
```

[24]:
```
pw.course_price_access
```

[24]:
```
3500
```

[25]:
```
pw.course_price_set = 2500
```

[26]:
```
pw.course_price_access
```

[26]:
```
3500
```

[27]:
```
pw.course_price_set = 5000
```

[28]:
```
pw.course_price_access
```

[28]:
```
5000
```

[ ]:

[ ]:

[ ]:
```
Q5.What is method overriding in python? Write a python code to demonstrate␣
↪method overriding.
```

ANS -

Method overriding is a feature of Object-Oriented programming languages where the subclass or child class can provide the program with specific characteristics or a specific implementation process of data provide that are already defined in the parent class or superclass.

[48]:
```
class  parent ():
    def __init__(self):
        self.value = "inside parent"
    def show(self):
        print(self.value)
```

[49]:
```
class child(parent):
    def __init_(self):
        self.value = "inside child"
    def show(self):
            print(self.value)
```

[50]:
```
obj1 = parent()
```

[51]:
```
obj2 = child()
```

[52]:
```
obj1.show()
```

```
inside parent
```

[53]:
```
obj2.show()
```

```
inside parent
```

[ ]:

[1]:
```
class parent1 ():
    def display(self):
        print ("this is my parent1")
```

[3]:
```
class parent2():
    def display(self):
        print("this is my parent2")
```

[4]:
```
class child(parent1 , parent2):
    def show(self):
        print("inside child")
```

[5]:
```
obj = child()
```

[6]:
```
obj.show()
```

```
inside child
```

[7]:
```
obj.display()
```

```
this is my parent1
```

[ ]: