Python_advance_assignment_2

Q1. What is the relationship between classes and modules?

Ans: A Python class is like an outline/blueprint/mold for creating a new object. An object is anything that you wish to manipulate or change while working through the code Every time a class object is instantiated, which is when we declare a variable,a new object is initiated from scratch

Whereas in Python, Modules are simply files with the py extension containing Python code that can be imported inside another Python Program. In simple terms, we can consider a module to be the same as a code library or a file that contains a set of functions/Classes that you want to include in your application.

Q2. How do you make instances and classes?

Ans: For creating a class instance. we call a class by its name and pass the arguments which its **init** method accepts. Example: Ram = employee('Male',20000), Here Ram is an instance of class employee with attriubutes 'Male' and 20000.

Whereas for creating a class, we use the Class keyword. Class keyword is followed by classname and semicolon. Example: Here Employee is a class created with class keyword with arguments gender and salary. class Employee: def **init**(self, gender,salary): self.gender = gender self.salary = salary

Q3. Where and how should be class attributes created?

Ans: Class attributes or Class level Attributes belong to the class itself. These attributes will be shared by all the instances of the class. Hence these attributes are usually created/defined in the top of class definiation outside all methods.

Example: In the below code we are defining a class attribute called no_of_wheels which will be shared by all the instances of the class Car.

class Car: no_of_wheels = 4; # this is a class attribute def__init__(self,color,price,engine): self.color = color #All this are instance attributes self.price = price self.engine = engine

Q4. Where and how are instance attributes created?

Ans: Instances attributes are passed to the class when an object of the class is created. Unlike class attributes, instance attributes are not shared by all objects of the classs. instead each object maintains its own copy of instance attributes at object level. whereas incase of class attributes all instances of class refer to a single copy. Usually instance attributes are defined within the **init** method of class

Example: In the below sample code we are creating a class Car with instance varaibles color, price, engine, which will be provided when an instance of class Car is created.

In [ ]:

```
class Car: def __init__(self,color,price,engine): self.color = color
#All this are instance attributes self.price = price self.engine = engine
nexon_ev = Car('Indigo Blue', 1400000, 'electric')
safari = Car('Pearl White',2100000, 'petrol')
nexon_ev, safari are both the instances of class Car with different instance variables.
```

Q5. What does the term “self” in a Python class mean?

Ans: self represents the instance of the class (it represents the object itself). By using the “self” keyword we can access the attributes and methods of the class with in the class in python. It binds the attributes with the given arguments.

In [14]:

```
class Car:
    def __init__(self,color,price,engine):
        self.color = color # All this are instance attributes
        self.price = price
        self.engine = engine

nexon_ev = Car('Indigo Blue', 1400000, 'electric')
safari = Car('Pearl White',2100000, 'petrol')

print(nexon_ev.__dict__)
print(safari.__dict__)
```

```
{'color': 'Indigo Blue', 'price': 1400000, 'engine': 'electric'}
{'color': 'Pearl White', 'price': 2100000, 'engine': 'petrol'}
```

Q6. How does a Python class handle operator overloading?

In [ ]:

```
Ans: Python Classes handle operator overloading by using special methods
called Magic methods.
These special methods usually begin and end with __ (double underscore)

Example: Magic methods for basic arithmetic operators are:
+ -> __add__()
- -> __sub__()
* -> __mul__()
/ -> __div__()
```

In [13]:

```
class Book:
    def __init__(self,pages):
        self.pages = pages
    def __add__(self,other):
        return self.pages + other.pages
b1 = Book(100)
b2 = Book(200)

print(f'The total number of pages in 2 books is {b1+b2}')
```

```
The total number of pages in 2 books is 300
```

Q7. When do you consider allowing operator overloading of your classes?

Ans: When we want to have different meaning for the same operator accroding to the context we use operator overloading.

Q8.What is the most popular form of operator overloading?

Ans: The most popular form of operator overloading in python is by special methods called Magic methods. Which usually beign and end with double underscore .

In [4]:

```
class A:
   def __init__(self,a):
       self.a = a
   def __add__(self,o):
       return self.a+o.a
obj1 = A(1)
obj2 = A(2)
obj3 = A('Sus')
obj4 = A('mita')

print(f'Sum -> {obj1+obj2}')
print(f'String Concatenation -> {obj3+obj4}')
```

```
Sum -> 3
String Concatenation -> Susmita
```

Q9. What are the two most important concepts to grasp in order to comprehend Python OOP code?

Ans: Classes and objects are the two concepts to comprehend python OOP code as more formally objects are entities that represent instances of general abstract concept called class. Along with classes and objects the important concepts to grasp are:

- Inheritence
- Abstraction
- Polymorphism
- Encapsulation