Assignment 02 Solutions

Q1. What is the relationship between classes and modules?

Ans: Modules are collections of methods and constants. They cannot generate instances. Classes may generate instances (objects), and have per-instance state (instance variables).

Q2. How do you make instances and classes?

Ans: Classes can be cretaed with the help of keyword class. To create instances of a class, we can call the class using class name and pass in whatever arguments its `__init__` method accepts.

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: Instance attributes are attributes or properties attached to an instance of a class. Instance attributes are defined in the constructor.Usually instance attributes are defined within the `__init__` method of class.

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 [3]:

```python
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('s red', 2100000, 'electric')
nexon_zx = Car('Pearl White',1800000, 'petrol')

print(nexon_ev.__dict__)
print(nexon_zx.__dict__)
```

```
{'color': 's red', 'price': 2100000, 'engine': 'electric'}
{'color': 'Pearl White', 'price': 1800000, 'engine': 'petrol'}
```

Q6. How does a Python class handle operator overloading?

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 [4]:

```python
class Book:
    def __init__(self,pages):
        self.pages = pages
    def __add__(self,other):
        return self.pages + other.pages
b1 = Book(800)
b2 = Book(945)
print(f'The total number of pages in 2 books is {b1+b2}')
```

```
The total number of pages in 2 books is 1745
```

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

Ans: We consider allowing operator overloading when we want to have different meaning for the same operator. For example operator + is used to add two integers as well as join two strings and merge two lists. It is achievable because '+' operator is overloaded by int class and str class.

In [5]:

```python
# Python program to show use of
# + operator for different purposes.

print(1 + 2)

# concatenate two strings
print("ineuron"+"Pw")

# Product two numbers
print(3 * 4)

# Repeat the String
print("Sudhanshu"*4)
```

```
3
ineuronPw
12
SudhanshuSudhanshuSudhanshuSudhanshu
```

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` .A very popular and convenient example is the Addition `(+)` operator. the '+' operator operates on two numbers and the same operator operates on two strings. It performs "Addition" on numbers whereas it performs `"Concatenation"` on strings.

In [12]:

```python
class name:
    def __init__(self,a):
        self.a = a
    def __add__(self,o):
        return self.a+o.a
obj1 = name(1)
obj2 = name(2)
obj3 = name('abhishek')
obj4 = name(' kale')
print(f'Sum -> {obj1+obj2}')
print(f'String Concatenation -> {obj3+obj4}')
```

```
Sum -> 3
String Concatenation -> abhishek kale
```

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

Ans: Classes and **objects** are the two most important concepts to grasp in order 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:

1. Inheritence
2. Abstraction
3. Polymorphism
4. Encapsulation

In [13]:

```
#polymorphism example :
class ineuron:
    def msg(self):
        print("this is a msg to ineruon")

class xyz:
    def msg(self):
        print("this is a msg to xyz")

def test(notes):
    notes.msg()

i = ineuron()
x = xyz()

test(i) # function giving some result
test(x) # same function giving some other result
```

```
this is a msg to ineruon
this is a msg to xyz
```

In [14]:

```
#inheritence example:
class Ineuron:
    company_website = 'https://ineuron.ai/'
    name = 'iNeuron'

    def contact_details(self):
        print('Contact us at ', self.company_website)

class Datascience(Ineuron):
    def __init__(self):
        self.year_of_establishment= 2018

    def est_details(self):
        print('{0} Company was established in {1}'
              .format(self.name,self.year_of_establishment))

ds = Datascience()
ds.est_details()
```

```
iNeuron Company was established in 2018
```