Date : 7th - 09 - 2020
Morning Session : 9am – 11.00 PM
By ~ Rohan Kumar

Topics: Python OOPs

OOP: Object Oriented Programing

We have 3 types of programing

1) **Procedure oriented programing (big complex problem divided into small functions, each function hold definition and calling that function)**

2) **Functional programing (defining functions, functions under functions)**

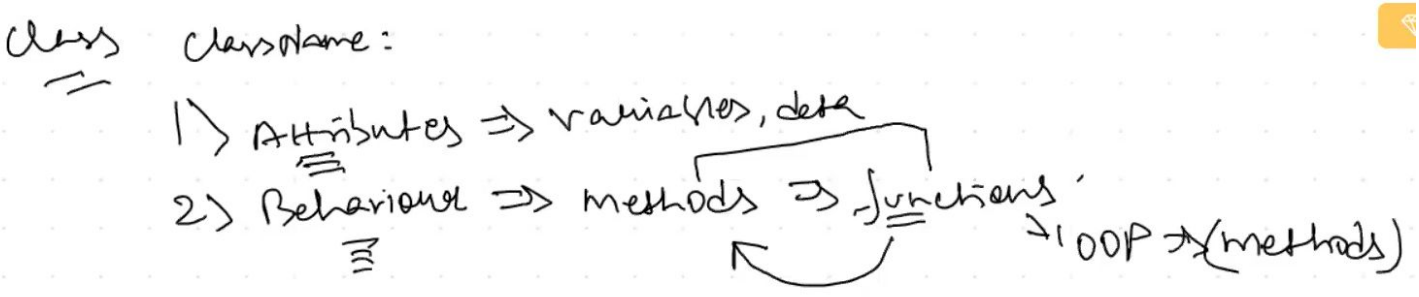
3) **Object oriented programing (everything is an object)**

Objects:

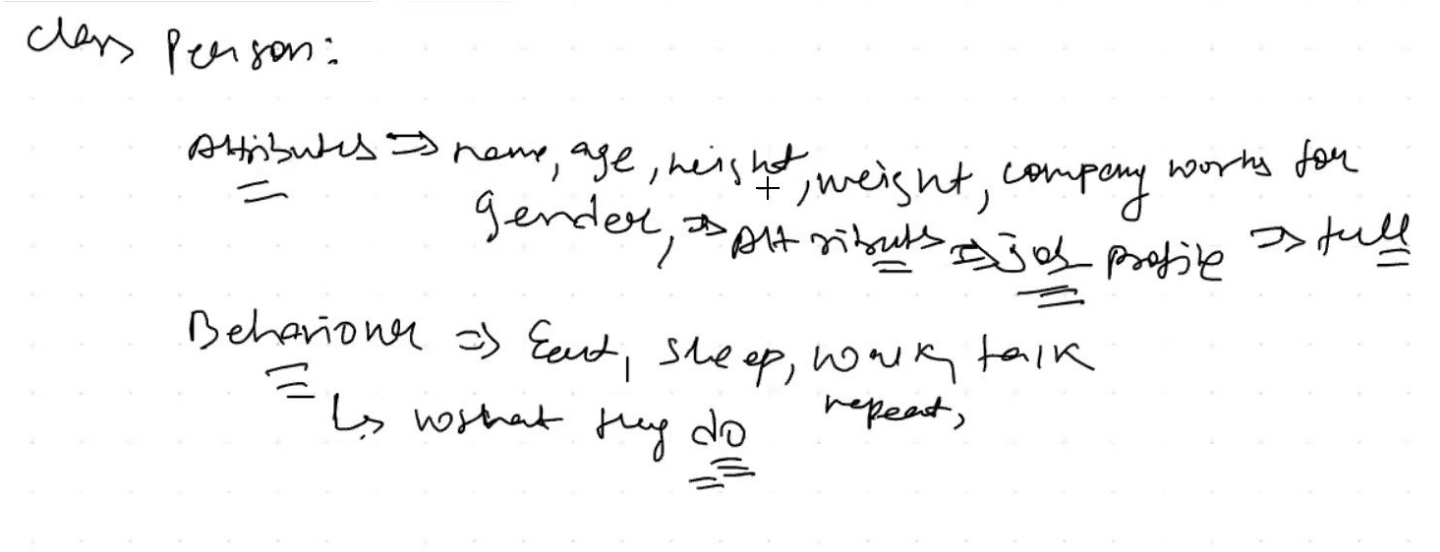
- **to do something we need objects,**
- **instance of class**

Class <int> , class <string> etc. these inbuilt classes

Creating Classes: Class creates a user-defined data structure, which holds its own data members and member functions, which can be accessed and used by creating an instance of that class. A class is like a blueprint for an object.



Example:



Storing the data: Attributes, => variables
Using the data : behaviour , methods , => functions

Example:

```
class Person:
    def getFullName(self):
        print("kumar rohan")

a = 5
string = "this is example"
person1 = Person()   #this is the way to create objetcs  person1 is an object

print(type(person1))
```

```
<class '__main__.Person'>
```

It is a class but with a '_main_.peron'

To excess the method under the class

```
class Person:
    def getFullName(self):
        print("kumar rohan")

a = 5
string = "this is example"
person1 = Person()   #this is the way to create objetcs  person1 is an object
person2 = Person() # object 2

print(type(person1))


Person.getFullName(person1)
```

```
<class '__main__.Person'>
kumar rohan
```

Self:

- Class methods must have an extra first parameter in method definition. We do not give a value for this parameter when we call the method, Python provides it.
- If we have a method which takes no arguments, then we still have to have one argument.
- This is similar to this pointer in C++ and this reference in Java.
- Example :

```
class Person:
    def getFullName(self,name):
        self.name = name
        print("my name is",self.name)

a = 5
string = "this is example"
person1 = Person()   #this is the way to create objetcs  person1 is an object
person2 = Person() # object 2

print(type(person1))


Person.getFullName(person1,"ramesh")
Person.getFullName(person2,"ram")
```

```
<class '__main__.Person'>
my name is ramesh
my name is ram
```

Example:

```
class Person:
    def info(self,name,age):
        print(name,age)

person1 = Person()   #object Created

person1.info('kumar',19)
```

```
kumar 19
```

__init__ () Method: The __init__ method is similar to constructors in C++ and Java. Constructors are used to initialize the object's state. Like methods, a constructor also contains a collection of statements(i.e. instructions) that are executed at the time of Object creation. It is run as soon as an object of a class is instantiated. The method is useful to do any initialization you want to do with your object.

Example:

```
class Person:
    def __init__(self,name,age): #get called automatically    #construtor
        self.name = name
        self.age = age

    def getDetails(self):
        print("name is",self.name)
        print("age is",self.age)

p1 = Person('abc',19) #create an object
p2 = Person('def',20)

p1.getDetails()
p2.getDetails()
```

```
name is abc
age is 19
name is def
age is 20
```

Explanation: 1st execution starts from p1, p1 =person object has been created now abc, 19 here passed as name and age, self passed as p1, we are storing as p1.name = abc, p1.age = 19, p1 got created and prind name = abe, age = 19.

P2 will called , p2 will pass instead of p1, p2.name = def, p2.age = 20, got created and print name = def, age = 20.

Example:

```
class Person:
    def __init__(self): #get called automatically    #construtor
        print("i am getting called")

p1 = Person() #create an object
p2 = Person()
```

```
i am getting called
i am getting called
```

Every class has a object (class is the blueprint, based on the blueprint we can create object (mobiles))

What is the size of an object??

Depends on number of variables or size of each variable

Who allocates size?

constructor allocate size.

```python
class Person:
    def __init__(self): #initializing,constructor
        self.name = "rohan"
        self.age = 18

    def update(self):
        self.age = 30

    def compare(self,other):
        if self.age == other.age:
            return True
        else:
            return False

p1 = Person()
p2 = Person()

p1.name = "mohit"

print(p1.age)
print(p2.age)

if p1.compare(p2):
    print("they are same")
else:
    print("different")
```

```
18
18
thet are same
```

Self: **when calling methods how pointer will now which object we are talking about**

Type of variable : 1) instance variable 2) class or static variable

From above Example :

Instance variable: name and age are instance variables bcoz for objects this variable is getting used. (DEPENDED ON THE OBJECT)

Class Or static variable : both are the same when variable is an independent object it is class/static variable.

Types of methods : 1) instance method 2) class method 3) static Method

Instance method : where ever we use self we can say it is instance method, bcoz its method will get change based on the object (update, compare)

Class method: when not using object variables.

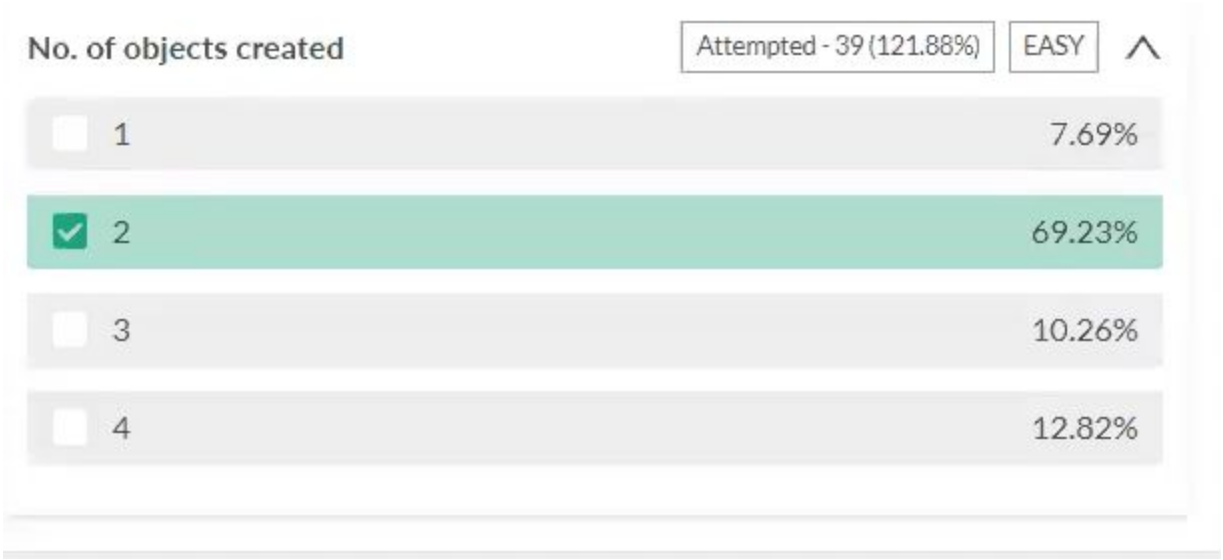
Static method: when we are not using class variable or instance variable then we can say it is static method.

MCQ's:

```
class Test:
    exam = "Unit test"

    def __init__(self,questions,total_marks,marks_got):
        self.questions = questions
        self.total_marks = total_marks
        self.marks_got = marks_got

    def final_marks(self):
        return self.marks_got

    @classmethod
    def exam_name(cls):
        return cls.exam

    @staticmethod
    def no_of_days():
        print("10 days")

t1 = Test()
t2 = Test()
```

From above Question

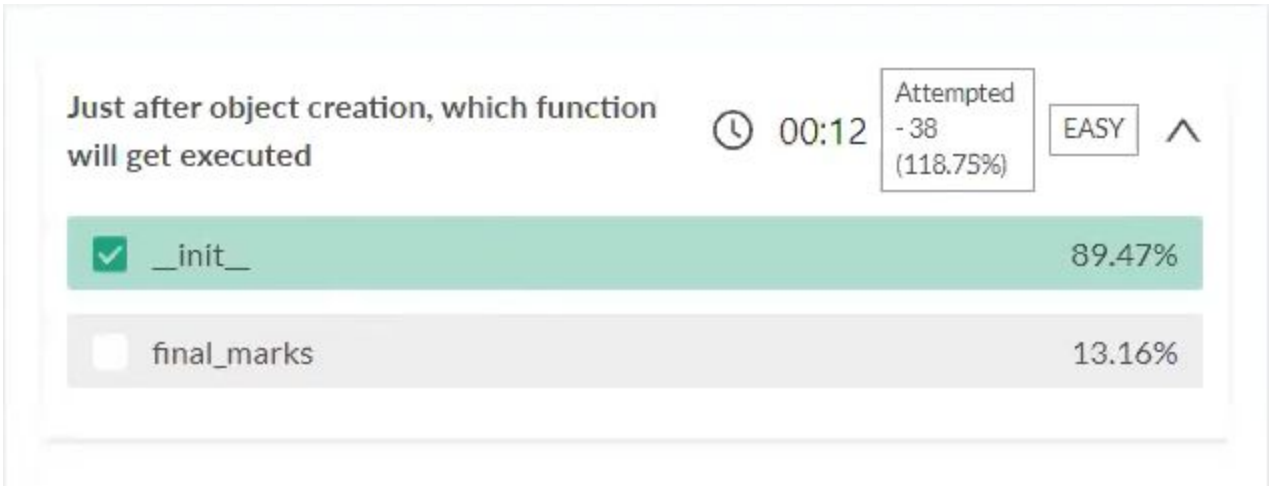
MCQ 1:



MCQ 2:

MCQ 3:

MCQ 4:


Just after object creation, which function will get executed
00:12
Attempted - 38 (118.75%)
EASY
__init__
89.47%
final_marks
13.16%