Advanced Python

OOP's concepts:

the OOP's concepts are certain rules to developing an applications.

OOP's means Object Oriented Programming Principles/Concepts.

Any high-level programming langauge which support OOP's concepts that type of programming languages are called Object Oriented Programming langauges.

the OOP's concepts are,

- Class
- Object
- Encapsulation
- Inheritance
- PolyMorphism
- Data Abstraction

we can develope any one application according to these OOP's concepts,we will get some benefits in that application.

the benefits of OOP's concepts are

- Security
- Reusability
- Flexibility

what is Encapsulation?

the concept of grouping or binding related data members along with it's related functionalities,is known as a Encapsulation.

we can implement the Encapsulation practically to required Class.

what is Class?

In Generally,a class is a Blueprint or structure.

in Programming terminalogy,a class is a syntaxer structure,which is used to represent the related data members along with its related functionalities.

a class is a Logical entity,it is a collection of Objects.

syntax

```
class calssname:
        """doc string"""
        Data
        Operations
```

we can represent the Data in class by using variables like static,non-static and local variables.

we can represent the Operations in class by using methods.

what is method?
----
        a method is a syntaxer structure,which is used to represent the business logics to perform a particular operation within the class.

                (or)

        a method is a block of organized,related code to perform a single action inside the class.

```
        syntax
        ------
        class classname:
                -------
                -------
                def methodname(self):
                        -------
                        -------
```

ex1:

```
class test:
    print("hello world")
```

output:

```
hello world
```

ex2:

```
class wishes:
    print("hai")
    def msg(self):
        print("siva krishna")
        print("good morning")
    print("how are you")
print("bye")
```

output:

```
hai
how are you
bye
```

note:
-----
in the above example our class is executed automatically but method is not executed automatically.

whenever we are calling a method then only method logic will be executed.

we can call the methods with the help of object.

what is object?
----
in generally object is a physical existance of a class.

in programming terminalogy,object is a instance of a class,which is used to allocate the memory space for non-static variables of a class.

syntax

```
reference_variable=classname()
```

what is reference variable?
----
the reference variable is a variable,which holds the hashcode of that particular object,i.e., whenever we are creating an object then immediately our python virtual machine(PVM) takes that address and to generate the hashcode and that hashcode is passing to reference_variable.

throught that reference variable we can access the static,non-static and methods of that class from out side that class.

ex3:

```
class wishes: #class definition
    print("hai")
    def msg(self): #method definition
        print("siva krishna")
        print("good morning")
    print("how are you")
w=wishes() #object creation
print(w)
w.msg() #method calling
print("bye")
```

output:

```
hai
how are you
<__main__.wishes object at 0x00000186DDACEAA0>
siva krishna
good morning
```

```
bye
```

we can create N-of objects to a single class,but each and every object haveing unique addressing.

ex4:

```
class test:
    def m1(self):
        pass
t1=test()
print(t1)
t2=test()
print(t2)
t3=test()
print(t3)
```

output:

```
<__main__.test object at 0x000001E88B2CEA70>
<__main__.test object at 0x000001E88B2CEB30>
<__main__.test object at 0x000001E88B2CEAA0>
```

with respect to one object we can call one method N-no.of times.

ex5:

```
class test:
    def m1(self):
        print("hai")
t1=test()
t1.m1()
t1.m1()
t1.m1()
t2=test()
t2.m1()
t3=test()
t3.m1()
t3.m1()
```

output:

```
hai
hai
hai
hai
hai
hai
```

```
what is self?
-------------
        a self is a reference variable,which is pointing to current class object.

        whenever we are instantiating an object then immediately internally it-self
that address is passing to default parameter(self).

ex:
---
class test:
    def m1(self):
        print(self)
t1=test()
print(t1)
t1.m1()
print('*'*30)
t2=test()
print(t2)
t2.m1()

output:
-------
<__main__.test object at 0x000002155AB5EA70>
<__main__.test object at 0x000002155AB5EA70>
******************************
<__main__.test object at 0x000002155AB5EB30>
<__main__.test object at 0x000002155AB5EB30>

ex2:
---
class test:
    def m1(siva):
        print(siva)
t1=test()
print(t1)
t1.m1()
print('*'*30)
t2=test()
print(t2)
t2.m1()

output:
-------
<__main__.test object at 0x00000172491EEA70>
<__main__.test object at 0x00000172491EEA70>
******************************
<__main__.test object at 0x00000172491EEB30>
<__main__.test object at 0x00000172491EEB30>
```