Decorators

Function Decorators: Decorator is a function which can take a function as argument and extend its functionality and returns modified function with extended functionality.

The main objective of decorator functions is we can extend the functionality of existing functions without modifying that function.

```
def wish(name):
        print("Hello",name,"Good Morning")
```

This function can always print same output for any name

Hello Navin Good Morning
Hello Haider Good Morning
Hello Nitin Good Morning

But we want to modify this function to provide a different message if the name is Navin. We can do this without touching the wish() function by using a decorator.

```
def decor(func):
        def inner(name):
                if name=="Navin":
                        print("Hello Navin Bad Morning")
                else:
                        func(name)
                                return inner
        @decor  def wish(name):
                print("Hello",name,"Good Morning")

    wish("haider")
    wish("Navin")
    wish("Nitin")
```

Output

```
        Hello haider Good Morning
        Hello Nitin Good Morning
        Hello Navin Bad Morning
```

Python's Object Oriented Programming (OOPs)

What is Class:

In Python everything's an object. To create objects we required some Model or Plan or Blue print, which is nothing but class.

We can write a class to represent properties (attributes) and actions (behaviour) of object.

Properties can be represented by variables

Actions can be represented by Methods.

Hence class contains both variables and methods.

How to Define a class?

We can define a class by using the class keyword.

Syntax:

```
class className:
        ''' documentation string '''
          variables:instance variables,static and local variables
          methods: instance methods,static methods,class methods
```

Documentation string represents the description of the class. Within the class doc string is always optional. We can get doc strings by using the following 2 ways.

1. print(classname.__doc__)
2. help(classname)

Within the Python class we can represent data by using variables.

There are 3 types of variables allowed.

1. Instance Variables (Object Level Variables)
2. Static Variables (Class Level Variables)
3. Local variables (Method Level Variables)

Within the Python class, we can represent operations by using methods. The following are various types of allowed methods

1. Instance Methods
2. Class Methods
3. Static Methods

Example for class:

```
class Student:
        """Developed by Navin Reddy for python demo'''
        def __init__(self):
                self.name='NavinReddy'
                self.age=40
                self.marks=80
        def talk(self):
                print("Hello I am :",self.name)
                print("My Age is:",self.age)
                print("My Marks are:",self.marks)
```

What is Object:

Physical existence of a class is nothing but an object.

We can create any number of objects for a class.

Syntax to create object:

```
reference variable = classname()
```

Example: s = Student()

What is Reference Variable:

The variable which can be used to refer to an object is called a reference variable.

By using reference variables, we can access properties and methods of objects

Program:
Write a Python program to create a Student class and create an object to it. Call the method talk() to display student details

```
class Student:
        """Developed by Navin Reddy for python demo"'
        def __init__(self):
            self.name='NavinReddy'
            self.age=40
            self.marks=80
        def talk(self):
            print("Hello I am :",self.name)
            print("My Age is:",self.age)
            print("My Marks are:",self.marks)
```

Output:

```
D:\python classes>py test.py
Hello My Name is: NavinReddy
My Rollno is: 101
My Marks are: 8
```

Self variable:

self is the default variable which is always pointing to the current object
(like this keyword in Java)

By using self we can access instance variables and instance methods of objects.

Note:

1. self should be the first parameter inside constructor def __init__(self):
2. self should be first parameter inside instance methods def talk(self):

Constructor Concept:

- Constructor is a special method in python.
- The name of the constructor should be __init__(self)
- Constructor will be executed automatically at the time of object creation.
- The main purpose of constructor is to declare and initialise instance variables.
- Per object constructor will be executed only once.
- Constructor can take at least one argument(at least self)
- Constructor is optional and if we are not providing any constructor then python will provide a default constructor.

Example:

```
def __init__(self,name,rollno,marks):
        self.name=name
        self.rollno=rollno
        self.marks=marks
```

Program to demonstrate constructor will execute only once per object:

```
class Test:
    def __init__(self):
        print("Constructor execution...")

    def m1(self):
        print("Method execution...")

t1=Test()
t2=Test()
t3=Test()
t1.m1()
```

Output
Constructor execution...
Constructor execution...
Constructor execution...
Method execution…

Differences between Methods and Constructors:

Method	Constructor
1. Name of method can be any name	1. Constructor name should be always __init__
2. Method will be executed if we call that method	2. Constructor will be executed automatically at the time of object creation.
3. Per object, method can be called any number of times.	3. Per object, Constructor will be executed only once
4. Inside method we can write business logic	4. Inside Constructor we have to declare and initialize instance variables

.

Types of Variables:
Inside Python class 3 types of variables are allowed.
1. Instance Variables (Object Level Variables)
2. Static Variables (Class Level Variables)
3. Local variables (Method Level Variables)

1. Instance Variables:
If the value of a variable is varied from object to object, then such types of variables are called instance variables. For every object a separate copy of instance variables will be created.

Where we can declare Instance variables:
1. Inside Constructor by using self variable
2. Inside Instance Method by using self variable
3. Outside of the class by using object reference variable

1. **Inside Constructor by using self variable:**
We can declare instance variables inside a constructor by using self keyword. Once we create an object, automatically these variables will be added to the object.

2. **Inside Instance Method by using self variable**:
We can also declare instance variables inside the instance method by using self variables. If any instance variable is declared inside the instance method, that instance variable will be added once we call that method.

3. **Outside of the class by using object reference variable:**
We can also add instance variables outside of a class to a particular object

2. Static variables:
If the value of a variable is not varied from object to object, such type of variables we have to declare within the class directly but outside of methods.
Such types of variables are called Static variables.
For the total class only one copy of the static variable will be created and shared by all objects of that class.
We can access static variables either by class name or by object reference. But I recommend using class name.

Various places to declare static variables:
1. In general we can declare within the class directly but from outside of any method
2. Inside constructor by using class name
3. Inside instance method by using class name
4. Inside classmethod by using either class name or cls variable
5. Inside static method by using class name

Local variables:
Sometimes to meet temporary requirements of a programmer,we can declare variables inside a method directly,such types of variables are called local variables or temporary variables.

Local variables will be created at the time of method execution and destroyed once method completes.

Local variables of a method cannot be accessed from outside of the method.

Types of Methods:

Inside Python class 3 types of methods are allowed

1. Instance Methods
2. Class Methods
3. Static Methods

1. Instance Methods:

Inside method implementation if we are using instance variables then such types of methods are called instance methods. Inside the instance method declaration,we have to pass a self variable.

def m1(self):

By using the self variable inside method we can able to access instance variables.

Within the class we can call the instance method by using a self variable and from outside of the class we can call by using object reference.

Setter and Getter Methods:

We can set and get the values of instance variables by using getter and setter methods.

Setter Method: setter methods can be used to set values to the instance variables. setter methods also known as mutator methods.

```
syntax: def setVariable(self,variable):
            self.variable=variable

Example: def setName(self,name):
            self.name=name
```

Getter Method: Getter methods can be used to get values of the instance variables. Getter methods also known as accessor methods.

```
syntax: def getVariable(self):
            return self.variable

Example: def getName(self):
            return self.name
```

2. Class Methods:

Inside method implementation if we are using only class variables (static variables), then such type of methods we should declare as class method.

We can declare class method explicitly by using @classmethod decorator.
For class method we should provide cls variable at the time of declaration
We can call class method by using classname or object reference variable.

3. Static Methods:
In general these methods are general utility methods.
Inside these methods we won't use any instance or class variables.
Here we won't provide self or cls arguments at the time of declaration.

We can declare static method explicitly by using @staticmethod decorator We can access static methods by using classname or object reference.

Polymorphism
Poly means many. Morphs means forms. Polymorphism means 'Many Forms'.

Eg1: Yourself is the best example of polymorphism.
In front of Your parents You will have one type of behaviour and with friends another type of behaviour.Same person but different behaviours at different places,which is nothing but polymorphism.

Eg2: + operator acts as concatenation and arithmetic addition
Eg3: * operator acts as multiplication and repetition operator
Eg4: The Same method with different implementations in Parent class and child classes.(overriding)

Related to polymorphism the following 4 topics are important

1. Duck Typing Philosophy of Python
2. Overloading
    1. Operator Overloading
    2. Method Overloading
    3. Constructor Overloading
3. Overriding
    1. Method overriding
    2. constructor overriding

1. Duck Typing Philosophy of Python:
In Python we cannot specify the type explicitly. Based on the provided value at runtime the type will be considered automatically. Hence Python is considered as a Dynamically Typed Programming Language.

```
def f1(obj):
        obj.talk()
```

What is the type of obj? We cannot decide at the beginning. At runtime we can pass any type.Then how can we decide the type?

At runtime if 'it walks like a duck and talks like a duck,it must be duck'. Python follows this principle. This is called Duck Typing Philosophy of Python.

Overloading:
We can use the same operator or methods for different purposes.

Eg1: + operator can be used for Arithmetic addition and String concatenation
print(10+20)#30 print('Navin'+'reddy')#Navinreddy

Eg2: * operator can be used for multiplication and string repetition purposes.
print(10*20)#200
print('Navin'*3)#NavinNavinNavin

Eg3: We can use deposit() method to deposit cash or cheque or dd deposit(cash) deposit(cheque) deposit(dd)

There are 3 types of overloading
1. Operator Overloading
2. Method Overloading
3. Constructor Overloading

1. Operator Overloading:
We can use the same operator for multiple purposes, which is nothing but operator overloading.
Python supports operator overloading.
Eg1: + operator can be used for Arithmetic addition and String concatenation
print(10+20)#30
print('Navin'+'reddy')#Navinreddy

Eg2: * operator can be used for multiplication and string repetition purposes.
print(10*20)#200
print('Navin'*3)#Navin

2. **Method Overloading:**
If 2 methods have the same name but different types of arguments then those methods are said to be overloaded methods.
Eg: m1(int a)
m1(double d)

But in Python Method overloading is not possible. If we are trying to declare multiple methods with same name and different number of arguments then Python will always consider only last method

3. Constructor Overloading:
Constructor overloading is not possible in Python. If we define multiple constructors then the last constructor will be considered.

Python Inheritance

Inheritance is an important aspect of the object-oriented paradigm. Inheritance provides code reusability to the program because we can use an existing class to create a new class instead of creating it from scratch.

In inheritance, the child class acquires the properties and can access all the data members and functions defined in the parent class. A child class can also provide its specific implementation to the functions of the parent class. In this section of the tutorial, we will discuss inheritance in detail.

In python, a derived class can inherit base class by just mentioning the base in the bracket after the derived class name. Consider the following syntax to inherit a base class into the derived class.

```
class derived-class(base class):

    <class-suite>
```

A class can inherit multiple classes by mentioning all of them inside the bracket. Consider the following syntax.

```
class derived-class(<base class 1>, <base class 2>, ..... <base class n>):

        <class - suite>
```

Python Multi-Level inheritance

Multilevel inheritance is possible in python like other object-oriented languages. Multi-level inheritance is achieved when a derived class inherits another derived class. There is no limit on the number of levels up to which the multilevel inheritance is archived in python.

The syntax of multilevel inheritance is given below.

Syntax

```
class class1:
    <class-suite>
class class2(class1):
    <class suite>
class class3(class2):
    <class suite>
```

Python Multiple inheritance

Python provides us the flexibility to inherit multiple base classes in the child class.

The syntax to perform multiple inheritance is given below.

Syntax

```
class Base1:
    <class-suite>

class Base2:
    <class-suite>
.
.
.
class BaseN:
    <class-suite>

class Derived(Base1, Base2, ..... BaseN):
```

13. <class-suite>

The issubclass(sub,sup) method

The issubclass(sub, sup) method is used to check the relationships between the specified classes. It returns true if the first class is the subclass of the second class, and false otherwise.

The isinstance (obj, class) method

The isinstance() method is used to check the relationship between the objects and classes. It returns true if the first parameter, i.e., obj is the instance of the second parameter, i.e., class.