**Decorators in python**

Decorators are used in python to add functionality to an existing code. Before learn decorator we must know some basic things in python.

In python, everything considered as objects. Names that define for these objects are simply identifiers. Functions are also objects with attributes.

We are free to assaign various names to a single funcion.

For example :

**def funct1(m):**

**print(m)**

execute this code

**funct1(‘Python’)**

It displays, the word **Python**

execute this code

**funct2=funct1**

**unct2(‘Python’)**

It displays, the word **Python**

Here it is clear that the **funct1() and funct2()** are refer to the same function.

A fuction in python is capable of having **another function as it’s parameter**.

For example:

**def addone(x):**

**return x + 1**

**def minusone(x):**

**return x - 1**

**def execute(func, x):**

**result = func(x)**

**return result**

Here as you see, one of the parameter of function execute is an another funtion. **execute(addone,5)** gives **6** as result and **execute (minusone,5)** gives the result **5**.

A function can r**eturn another function too**.

For example:

**def called():**

**def returned():**

**print("Xanthron")**

**return returned**

**new = called()**

Here, the **returned()** function is under the **called()** function and the **called()** function returns **returned()** function.

Now we learn what is **decoraters in python**

As you know, all functions and methods are callable. A decorator is a callable and it returns a callable.

In general, a decorator takes a function as input and add some functionality to it and return it.

I shall explain it with an example.

**def decorated(func):**

**def inner():**

**print("Decorated with new function")**

**func()**

**return inner**

**def myfunct():**

**print("Before Decoration")**

Here there are three functions,

1. myfunct()

2. decorated()

3. inner()

The **decorated()** function retruns another function called **inner()**. If we execute the function **myfunct()**,

we will get the following result.

myfunct()

Before Decoration.

Now try the following.

afterdecor = decorated(myfunct)

afterdecor()

The result will be the following:

Decorated with new function

Before Decoration

From this output it is very clear that we decorated the myfunct() with another function decorated() and add a new functionality inner() to it.

We can use the @ symbol along with the name of the decorator to refer the decorator before the original function. For example

**def decorated(func):**

**def inner():**

**print("Decorated with new function")**

**func()**

**return inner**

**@decorated**

**def myfunct():**

**print("Before Decoration")**

**Assaignment**

1. Write a program using decorators with multiple parameters.

2. Write a program using multiple decorator to add functionalities to and original function