Decorators

Decorator is a special function in python.

Decorator is a nested function or inner function, which is used to decorate a function.

Decorators are a very powerful and useful tool in **Python** since it allows programmers to modify the behaviour of a function or class.

A **decorator** is a design pattern in **Python** that allows a user to add new functionality to an existing object without modifying its structure.

Decorators in Python are functions that takes another function as an argument and extends its behavior without explicitly modifying it.

These decorators are two types

1. Predefined decorators
2. User defined decorators

Predefined decorators
The decorators provided by python are called predefined decorators.
Example: @staticmethod, @abstractmethod, @property,…

User defined decorators
The decorators provided by programmer are called user defined decorators or application specific decorators.

Basic steps to with decorators

1. Define a function, which receives input as another function
2. Inside this function define another function which modify the function which it received
3. Return inner function/modified function/updated function

Syntax:

```
def <decorator-function-name>(function):
        def <inner function>([parameters]):
                statement-1
                statement-2
        return inner-function
```

After developing decorator it is applied to a function using @decorator syntax

Example:

```
def draw(function):
    def display_new():
        print("*"*30)
        function()
        print("*"*30)
    return display_new


@draw
def display():
    print("Hello Python")

@draw
def print_data():

dict1={'naresh':50,'suresh':60,'kishore':45
}
    for name,age in dict1.items():
        print(f'{name}---->{age}')



# internals
#function=draw(display)
#function()

display()
```

Output

```
*********************
*********
Hello Python
*********************
*********
*********************
*********
naresh---->50
suresh---->60
kishore---->45
*********************
*********
```

```
print_data()
```

Example

```
def smart_div(function):
    def new_div(n1,n2):
        if n2==0:
            return 0
        else:
            n3=function(n1,n2)
            return n3

    return new_div


@smart_div
def div(n1,n2):
    n3=n1/n2
    return n3



num1=int(input("Enter First Number "))
num2=int(input("Enter Second Number "))
num3=div(num1,num2)
print(f'The division of
{num1}/{num2}={num3:.2f}')
```

Output

```
Enter First Number 5
Enter Second
Number 0
The division of
5/0=0.00

Enter First Number 5
Enter Second
Number 2
The division of
5/2=2.50
```

Example

```
def upper(function):
    def print_upper_strings(strings):
        for s in strings:
            print(s.upper())
    return print_upper_strings


@upper
def print_strings(strings):
    for s in strings:
```

Output

```
ABC
XYZ
PQR
MNO
NARESH
SURESH
KISHORE
```

```
        print(s)

@upper
def print_dict_names(keys):
    for k in keys:
        print(k)

list1=["abc","xyz","pqr","mno"]
print_strings(list1)

emp_dict={'naresh':50000,'suresh':65000,'
kishore':45000}
print_dict_names(emp_dict.keys())
```

Decorator chaining

Chaining decorators involves applying multiple decorators to a single function. Python allows you to chain decorators by stacking them on top of each other, and they are executed from the innermost to the outermost decorator.

In Python, a decorator is a special construct that allows us to add extra functionality to an existing function or class without modifying its source code. A decorator is a callable that takes another function or class as input and returns a modified version of it.