Decorators

Decorator is nested/inner function, which receives a function as input and returns a function as output.
In python every function is called one object.
Decorator is a special function, which is used to modify functionality of existing function without modifying existing function.
Decorators are used to transform one function to another function.
In Python, a decorator is a design pattern that allows you to modify the functionality of a function by wrapping it in another function.
By definition, a decorator is a function that takes another function and extends the behavior of the latter function without explicitly modifying it.

In application development decorators are used for function transformation (changing behavior of function or class).

Once decorator is developed it can be applied to one function or more than one function.

Python provides predefined decorators
Example: staticmethod, classmethod,…

Syntax of writing decorator

```
def decorator-name(function):
        def transformed-function(parameters):
                statement-1
                statement-2
        return transformed-function
```

Syntax of applying decorator

```
@decorator-name
def function-name(parameter):
        statement-1
        statement-2
```

Example:

```
def print_upper(f):
    def print_message_upper(msg):
        print(msg.upper())
```

```
    return print_message_upper


@print_upper
def print_message(msg):
    print(msg)


print_message("Hello Python")
```

Output:
HELLO PYTHON

Example:

```
# Working functionality of decorator
def print_upper(f):
    def print_message_upper(msg):
        print(msg.upper())
    return print_message_upper
@print_upper
def print_message(msg):
    print(msg)


# interal functionality of PVM
# pru=print_upper(print_message)
# pru("hello python")

print_message("hello python")
```

Output:
HELLO PYTHON

Example:

```
def draw(f):
    def print_info_with_headerfooter(msg):
        print("*"*40)
        f(msg)
```

```
        print("*"*40)
    return print_info_with_headerfooter

@draw
def print_info(msg):
    print(msg)

def header(f):
    def student_report_header(**kwargs):
        print("Naresh I Technologies")
        f(**kwargs)
        print()
        print("OPP: Satyam,Ameerpet,Hyd")
    return student_report_header

@header
def student_report(**iterable):
    for rollno,name in iterable.items():
        print(f'{rollno}\t{name}')

@header
def marks_report(**iterable):
    for name,marks in iterable.items():
        print(f'{name}-->{marks}')

print_info("Python Decorators")
stud={'S1':"naresh",'S2':'suresh','S3':'kishore'}
student_report(**stud)
stud_marks={'naresh':[50,60],'kishore':[60,70],'suresh':[90,80]}
marks_report(**stud_marks)
```

Output:

```
****************************************
Python Decorators
****************************************
Naresh I Technologies
S1      naresh
```

S2 suresh
S3 kishore

OPP: Satyam,Ameerpet,Hyd
Naresh I Technologies
naresh-->[50, 60]
kishore-->[60, 70]
suresh-->[90, 80]

OPP: Satyam,Ameerpet,Hyd

Example:

```
def smart_div(f):
    def div_with_zero(num,d):
        if d==0:
            return 0
        else:
            r=f(num,d)
            return r
    return div_with_zero

@smart_div
def div(num,d):
    res=num/d
    return res

result1=div(5,2)
print(result1)
result2=div(4,2)
print(result2)
result3=div(7,0)
print(result3)
```

Output:

```
2.5
2.0
0
```