PYTHON

From Simple to Complex
With
Examples

AYESHA NOREEN

Bachelor's in Software Engineering,

Master's in Computer Science

from COMSATS University, Islamabad

NOTE!!!

In these notes Screenshots of practice examples and coding are added. The code files are also available in code folder that contain .ipynb files that are created on Jupyter notebook.

Chapter16
Decorators

Decorators are used to enhance the functionality of other functions.

Syntax:

```
Def decotaor_funct(any_func):
        wrapper_func():
                any_func()
                return ‘here extra functionality’
        return wrapper_fun
```

- # Decorator function

```
def func1():
    print("This is function1")
def func2():
    print("This is function2")
func1()
func2()
# If we want to add extra functionality with both functions e.g
# add a line 'This is awsome function' with both functionsthan we use decorator function as
def decorator_func(any_func):
    def wrapper_func():
        any_func()
        return 'This is awsome function'
    return wrapper_func
var=decorator_func(func1)
print(var())
var=decorator_func(func2)
print(var())
```

```
This is function1
This is function2
This is function1
This is awsome function
This is function2
This is awsome function
```

- **Syntactic sugar@**

We can also define decorator function first than place @than decorator function name at above of any function than simply call function this is a shortcut called syntactic sugar.

```python
#syntactic sugar
def decorator_func(any_func):
    def wrapper_func():
        any_func()
        return 'This is awsome function'
    return wrapper_func
@decorator_func
def func1():
    print("This is function1")
@decorator_func
def func2():
    print("This is function2")

print(func1())
print(func2())
```

```
This is function1
This is awsome function
This is function2
This is awsome function
```

- # Decorator with args and kwargs

```python
#decorator with args and kwargs
def decorator_func(any_func):
    def wrapper_func(*args,**kwarg):
        any_func(*args,**kwarg)
        print('this is awsome function')
    return wrapper_func
@decorator_func
def func1(num):
    print(f"This is function 1 with argument:{num}")
def func2(a,b):
    return a+b
print(func1(5))
print(func2(14,6))
```

```
This is function 1 with argument:5
this is awsome function
None
20
```

- # @wraps() function

```python
from functools import wraps
def decorator_func(any_func):
    @wraps(any_func)
    def wrapper_func(*args,**kwarg):
        print('This is wrapper function')
        return any_func(*args,**kwarg)
    print('this is awsome function')
    return wrapper_func
@decorator_func
def func1():
    """This is add function"""
print(func1.__doc__)  #it prints This is add function
print(func1.__name__) #it print func1
```

```
this is awsome function
This is add function
func1
```

TODO Task

Define a square function that prints execution time of program by using decorator.

```
from functools import wraps
import time
t2=time.time()
def calculate_time(any_func):
    @wraps(any_func)
    def wrapper(*args,**kwarg):
        print(f"Executing ........{any_func.__name__}")
        t1=time.time()
        returned_value=any_func(*args,**kwarg)

        total_time=t2-t1
        print(f"This function takes {total_time }seconds to execute")
        return returned_value
    return wrapper
@calculate_time
def square(num):
    return num**2
n=int(input("Enter any number whose square you want to calculate:"))
print(square(n))
```

```
Enter any number whose square you want to calculate:67
Executing ........square
This function takes -2.6086087226867676seconds to execute
4489
```

TODO Task

Define a function that take many inputs and return total if any character is entered than print wrong input otherwise return sum also use decorator function.

```
from functools import wraps
def only_int_allow(function):
    @wraps(function)
    def wrapper(*args,**kwarg):
        datatype=[]
        for arg in args:
            datatype.append(type(arg)==int)
        if all(datatype):
            return function(*args,**kwarg)
        else:
            return "invalid input"
    return wrapper
@only_int_allow
def add_all(*args):
    total=0
    for i in args:
        total+=i
    return total
print(add_all(1,2,3,4,5,10,3,2,'ayesha'))
print(add_all(1,2,3,4,5,10,3,2))
```

```
invalid input
30
```

- **Decorator inside decorator**

```
from functools import wraps
def only_datatype_allow(datatype):
    def decorator(function):
        @wraps(function)
        def wrapper(*args,**kwarg):
            if all([type(arg)==datatype for arg in args]):
                return function(*args,**kwarg)
            print("invalid input")
            return wrapper
        return decorator
@only_datatype_allow(str)
def concatestr(*args):
    string=''
    for i in args:
        string+=i
        return string
print(concatestr('ayesha',' noree','n')) #print ayesha noreen
print(concatestr('sana',2))              #print invalid input
```