

Task 8: Implement python generator and decorators

Aim:-

write a python program to implement python generators and decorators

1st write a python program that includes a generator function to produce a sequence of numbers.

- Produce a sequence of numbers when provided with start, end, and step values
- Produce a default sequence of numbers starting from 0, ending at 10, and with a step of 1 if no values are provided

Produce a sequence of numbers when provided with start, end, and step values.

Algorithm:

1. Define Generator Function:

- Define the function `number` -
Sequence (start, end, step)

2. Initialize Current Value:

- set current to the value ~~to~~ start

3. Generate Sequence

- while current is less than or equal to end:

- yield the current value of current

- increment current by step.

4. create Generator object:

- Create a generator object by calling number - sequence with user-provided values

5. Print Generated Sequence:

- Iterate over the values provided by the generator object
- print each value

8.1 program

```
def number - sequence (start, end, step=1):
```

```
    current = start
```

```
    while current <= end
```

```
        yield current
```

```
        current += step
```

```
start = int (input ("Enter the starting  
number: "))
```

```
end = int (input ("Enter the ending  
number: "))
```

```
step = int (input ("Enter the step  
value: "))
```

```
# Create the generator
```

```
Sequence - generator = number - sequence
```

```
# print the generator Sequence of  
numbers
```

```
for number in sequence - generator  
    print (number)
```

out put!

Enter the starting number: 1

Enter the ending number: 50

Enter the step value: 5

1

6

11

16

21

26

31

36

41

46

produce a default sequence of numbers starting from 0, ending at 10 and with a step of 1 if no values are provided

Algorithm:-

1. Start Function:

- Define the function my-generator(n) that takes a parameter n

2. Initialize Counter:

- Set value to 0.

3. Generate Values

- While value is less than n:
 - Yield the current value
 - Increment value by 1

4. Create Generator object

- Call my-generator(12) to create a generator object.

5. Iterate and print values

- For each value produced by the generator object:
 - Print value.

8.2 (b) Program:

```
def my-generator():  
    # Initialize counter  
    value = 0  
  
    # loop until counter is less than  
    while value < 3:  
        # produce the current value of  
        the counter  
        yield value  
        # increment the counter  
        value += 1  
  
# Iterate over the generator object  
# produced by my-generator for  
value in my-generator():  
    # Print each value produced by  
    generator  
    print (value)
```

Output:

0

1

2

Imagine you are working on a messaging application that needs to format messages differently based on the user's preferences. Users can choose to have their messages automatically converted to uppercase or to uppercase - decorator and lower case - decorator. Write a program to implement it.

Algorithm:

1. Create Decorators:

- Define uppercase - decorator to convert the result of a function to upper case
- Define lowercase - decorator to convert the result of a function to lower case.

2. Define Functions:

- Define shout function to return the input text. Apply @uppercase decorator to this function
- Define whisper function to return the input text. Apply @lowercase decorator to this function.

3. Define greet Function:

- Define greet function that:
 - Accepts a function as input.
 - calls this function with the text "Hi, I am created by a function passed as an argument."
 - prints the result.

4. Execute the program

- call greet to print the greeting in upper case
- call greet to print the greeting in lower case.

Program:-

```
def upper case - decorator (func):  
    def wrapper (text):  
        return func(text). upper()  
    return wrapper
```

```
def lower case - decorator (func):  
    def wrapper (text):  
        return func(text) - lower()  
    return wrapper
```

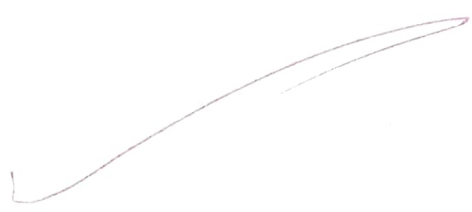
@ upper case - decorator
def shout (text):
 return text

@ lower case - decorator
def whisper (text):
 return text

```
def greet (func):  
    greeting = func ("Hi, I am created  
        by a function passed as an  
        argument.")  
    print (greeting)
```

greet (shout)

greet (whisper)



VELTECH	
EX No.	8
PERFORMANCE (3)	5
RES. TEST AND ANALYSIS (5)	5
VIVA VOCE (5)	5
PROJECT (5)	5
TOTAL (20)	25
SIGN WITH DATE	17/9

Result:- Thus the python programs implementing
python generator and decorators was
successfully executed and the output
was verified.

Output:-

Hi, I AM CREATED BY A FUNCTION
PASSED AS AN ARGUMENT

hi, i am created by a function passed
as an argument