

## Task-7 Implementation python generator and decoders.

8/9/25

Aim: write a python program to implement python generator and decorators.

- produce a sequence of number when provided with start end and step values.
- produce a default sequence of number starting from 0, ending 10, and with a step 1 if (no values are provided)

### Algorithm:-

1. Define generator function:

Define the function number sequence (start, end, step-1).

2. Initialize current value:

• set current to value of start

3. generate sequence:

- while current is less than or equal to end:
  - yield the current value of current.
  - increment current by step.

4. get user input:

- Read the starting number (start) from user input.
- Read the ending number (end) from user input.

5. create generator object:

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

6. print generated sequence:

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

### Program:-

```
def number-sequence (start, end, step=1):  
    current = start  
    while current <= end:  
        yield current  
        current += step
```



Output:

Enter the starting number: 1  
Enter the ending number: 50  
Enter the step value: 5

6  
11  
16  
21  
26  
31  
36  
41  
46

Results  
Thus the Python program, using Function's concept was successfully executed and the output was verified.

VEL TECH - CSE	
NO.	
PERFORMANCE (%)	
QUALITY ANALYSIS (%)	
INNOVATION (%)	
LEADERSHIP (%)	
ADAPTABILITY (%)	
TEAMWORK (%)	
PROBLEM SOLVING (%)	
COMMUNICATION (%)	
CRITICAL THINKING (%)	
CREATIVITY (%)	
EMOTIONAL INTELLIGENCE (%)	
RESILIENCE (%)	
SELF-MANAGEMENT (%)	
WORK ETHIC (%)	
LEADERSHIP SKILLS (%)	
TEAMWORK SKILLS (%)	
PROBLEM SOLVING SKILLS (%)	
COMMUNICATION SKILLS (%)	
CRITICAL THINKING SKILLS (%)	
CREATIVITY SKILLS (%)	
EMOTIONAL INTELLIGENCE SKILLS (%)	
RESILIENCE SKILLS (%)	
SELF-MANAGEMENT SKILLS (%)	
WORK ETHIC SKILLS (%)	



```
start = int ("Enter the starting number")  
end = int (input ("Enter the ending number:"))  
# create the generator
```

```
sequence-generator = number-sequence (start, end, step)
```

```
# print the generator sequence of number.
```

```
for number in sequence-generator:
```

```
    print (number)
```

Ex. 1(b) program:- produce a default sequence of number starting from 0, ending at 10, and with a step of 1 value provided.

Algorithm:-

1. Start function:-

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

2. Initialize counter:-

Set value to 0.

3. generate values:-

while value is less than n;

4. create generator object:-

call my-generator (11) to create generator object.

5. Iterate and print values:-

For each value produced by generator object:  
print value.

Program:-

```
def my-generator (n):
```

```
# initialize counter
```

```
    value = 0
```

```
# loop until counter less than n.
```

```
    while value < n:
```

```
        # produce the current value of counter.
```

```
        yield value
```

```
        # increment the counter
```

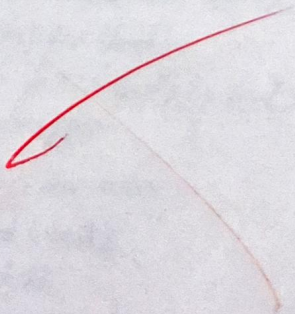
```
        value += 1
```



# Iterate over generator object produced by my-generator.  
for value in my-generator (3):

# Print each value produced by generator.  
print(value).

Hi, I am created by a function based on an argument  
to, from created by a function based on an argument





## Task (7.2) Algorithm 1.

8/9/25

### Algorithm 1

1. create decorators:-
  - Define uppercase-decorator to convert the result of a function to uppercase.
  - Define lowercase-decorator to convert result of a function to lowercase.
2. Define functions:
  - Define shout function to return the input text. Apply @upper-decorator to this function.
3. Define greet function:
  - Define greet function that:
    - Accepts a function (func) as input:
    - calls this function with text "Hi, I am created by function passed as an argument.
    - Prints the result.
4. Execute program:
  - call greet(shout) to print greeting in uppercase.
  - call greet(whisper) to print greeting in lower case.

### Program

```
def uppercase-decorator(func):  
    def wrapper(text):  
        return func(text).upper()  
    return wrapper  
  
def lowercase-decorator(func):  
    def wrapper(text):  
        return func(text).lower()  
    return wrapper  
  
@uppercase-decorator  
def shout(text):  
    return text  
  
@lowercase-decorator  
def whisper(text):  
    return text
```



greet (shout)

greet (whisper)

Algorithm

1. Write to file

- Define write\_file (filename) function

- open a file named "log.txt" in write mode

- write the following text to file

Error objects are thrown when something goes wrong. The object can also be used as a way to handle errors.

- close the file

2. Read from file

- Define read\_file (filename) function

- open the file specified by filename in read mode using "r" with statement

- Read the entire content of file

- print content

3. Execute program

- call write\_file("write") to write

log.txt

- call read\_file("read") to read

log.txt and print content

and execute

VEL TECH - CSE	
EX NO.	
PERFORMANCE (5)	7
RESULT AND ANALYSIS (5)	5
VIVA VOCE (5)	5
RECORD (5)	5
TOTAL (20)	15
SIGN WITH DATE	

Result:

~~Hi, I am created by a function passed as an argument.~~

~~hi, I am created by function passed as an argument.~~

Thus, the python program to implement python generator - generator and decorator was successfully executed and output was verified.