**TASK SIX**

**GENERATORS, LIST COMPREHENSION AND DECORATORS**

1. Write a program in Python to find out the character in a string which is uppercase using list

comprehension.

test\_str = "SangeetaBalchandani"  
  
print("The original string is : " + str(test\_str))  
  
res = [char for char in test\_str if char.isupper()]  
print("The uppercase characters in string are : " + str(res))

2. Write a program to construct a dictionary from the two lists containing the names of students and their corresponding subjects. The dictionary should map the students with their respective subjects.

Let’s see how to do this using for loops and dictionary comprehension.

HINT - Use Zip function also

Sample input: students = ['Smit', 'Jaya', 'Rayyan'] subjects = ['CSE', 'Networking', 'Operating System']

Expected output: {‘Smit’ : ’CSE’ , ’Jaya’ : ’Networking’ , ’Rayyan’ : ’Operating System’}

students = ['Smit', 'Jaya', 'Rayyan']  
subjects = ['CSE', 'Networking', 'Operating System']  
  
print("Original student list is : " + str(students))  
print("Original subject list is : " + str(subjects))  
  
res = {students[i]: subjects[i] for i in range(len(students))}  
print("Resultant dictionary is : " + str(res))

3. Learn More about Yield, next and Generators

Yield: **Yield**is a keyword in Python that is used to return from a function without destroying the states of its local variable and when the function is called, the execution starts from the last yield statement. Any function that contains a yield keyword is termed as generator.

Example showing yield and printing even number from the provided test list.

def print\_even(test\_list):  
 for i in test\_list:  
 if i % 2 == 0:  
 yield i  
  
test\_list = [1, 4, 5, 6, 7]  
  
print("The original list is : " + str(test\_list))  
print("The even numbers in list are : ", end=" ")  
for j in print\_even(test\_list):  
 print(j, end=" ")

Generators:

**Generator-Function :**A generator-function is defined like a normal function, but whenever it needs to generate a value, it does so with the [yield keyword](https://www.geeksforgeeks.org/use-yield-keyword-instead-return-keyword-python/)rather than return. If the body of a def contains yield, the function automatically becomes a generator function.

**Generator-Object :**Generator functions return a generator object. Generator objects are used either by calling the next method on the generator object or using the generator object in a “for in” loop.

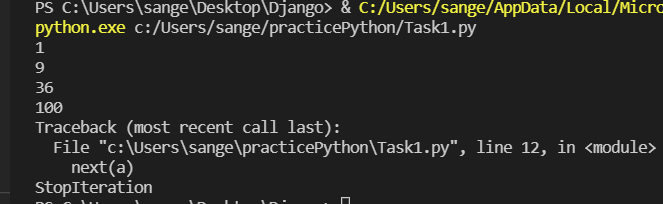
There is a lot of work in building an [iterator](https://www.programiz.com/python-programming/iterator) in Python. We have to implement a class with \_\_iter\_\_() and \_\_next\_\_() method, keep track of internal states, and raise StopIteration when there are no values to be returned.

This is both lengthy and counterintuitive. Generator comes to the rescue in such situations.

Next Example:

my\_list = [1, 3, 6, 10]  
  
a = (x\*\*2 for x in my\_list)  
print(next(a))  
  
print(next(a))  
  
print(next(a))  
  
print(next(a))  
  
next(a)

Output:



4. Write a program in Python using generators to reverse the string.

Input String = “Consultadd Training”

def rev\_str(my\_str):  
 length = len(my\_str)  
 for i in range(length - 1, -1, -1):  
 yield my\_str[i]  
  
  
# For loop to reverse the string  
for char in rev\_str("Consultadd Training"):  
 print(char)

5. Write an example on decorators.

def hello\_decorator(func):  
 def inner1(\*args, \*\*kwargs):  
 print("before Execution")  
  
 returned\_value = func(\*args, \*\*kwargs)  
 print("after Execution")  
 return returned\_value  
 return inner1  
  
@hello\_decorator  
def sum\_two\_numbers(a, b):  
 print("Inside the function")  
 return a + b  
  
  
a, b = 1, 2  
print("Sum =", sum\_two\_numbers(a, b))