Generators

What is generator?

A function which returns a generator iterator.
It looks like a normal function except that it contains yield expressions for producing a series of values usable in a for-loop or that can be retrieved one at a time with the next() function.

yield keyword

yield is a keyword
yield return value to caller pause execution of function
when iterator next() method is called it resumes back execute function

generator iterator

An object created by a generator function.
Each yield temporarily suspends processing, remembering the location execution state (including local variables and pending try-statements).
When the *generator iterator* resumes, it picks up where it left off

Example

```
def fun1():
    yield 1
    yield 10
    yield 40
    yield 20
    yield 90

def main():
    f=fun1() # return generator iterator object
    print(f)
    value1=next(f)
    print(value1)
    value2=next(f)
    print(value2)
    value3=next(f)
    print(value3)
    value4=next(f)
    print(value4)
    value5=next(f)
    print(value5)
```

```
    f=fun1()
    for value in f:
        print(value,end=' ')
main()
```
Output:
```
<generator object fun1 at 0x0000000353842DD0>
1
10
40
20
90
1 10 40 20 90
```

Example:
```
# create generator function which generates factorials within given range
def factorial_gen(start,stop):
    for num in range(start,stop+1): # 1 2 3 4 5
        fact=1
        for i in range(1,num+1): # 1 2 3 4
            fact=fact*i
        yield fact

def main():
    fact=factorial_gen(1,5) # create generator iterator object
    for value in fact:
        print(value)

    fact=factorial_gen(1,10)
    s=0
    for value in fact:
        print(value,end=' ')
        s=s+value
    print('--->',s)

main()
```
Output:
```
1
2
6
```

```
24
120
1 2 6 24 120 720 5040 40320 362880 3628800 ---> 4037913
>>>
```

Example:

```
import random
def otp_generator():
    while True:
        otp=random.randint(100000,999999)
        yield otp

def main():
    otpg=otp_generator() # genreator iterator object
    otp1=next(otpg)
    print(otp1)
    otp2=next(otpg)
    print(otp2)
main()
```

Output:

```
859532
395573
>>>
```

Generator Expression

An expression that returns an iterator. It looks like a normal expression followed by a for clause defining a loop variable, range, and an optional if clause.

```
>>> odd=(num for num in range(1,10,2))
>>> next(odd)
1
>>> next(odd)
3
>>> next(odd)
5
>>> sqr=(num**2 for num in range(1,6))
>>> next(sqr)
1
>>> print(sqr)
<generator object <genexpr> at 0x00000032E498BC10>
```

```
>>> next(sqr)
4
>>> next(sqr)
9
>>>
```

Lambda Functions or lambda expressions
Lambda function is anonymous function
A function which does not have any name is called anonymous function
Lambda function is having only one statement
Lambda functions are used as a higher order functions
A function which is defined as argument to another function is called higher order function

Syntax:
lambda [arg1,arg2,..]:expression

Example:
```
def main():
    a=lambda:print("lambda function")
    print(a)
    a()
    a()
    a()
    b=lambda:print("lambda function")
    b()

main()
```
Output:
```
<function main.<locals>.<lambda> at 0x0000004B42DFAE50>
lambda function
lambda function
lambda function
lambda function
>>>
```

Q: What is difference between a function and lambda function?

Function	**Lambda function**
Function is with name	Lambda function is without name
Function is written using def	Lambda function written using

keyword	lambda keyword
Function can have multiple statements	Lambda expression is having only one statement
Function cannot be defined as an argument	Lambda function is defined as an argument/can be used as higher order function
return statement is used to return value	return statement is not allowed

Example:

```
def main():
    add=lambda a,b:a+b
    sub=lambda a,b:a-b
    res1=add(10,20)
    res2=sub(10,5)
    print(res1)
    print(res2)
    find_max=lambda a,b:a if a>b else b
    res3=find_max(100,10)
    res4=find_max(20,50)
    print(res3,res4)

main()
```

Output:

```
30
5
100 50
>>>
```