PYTHON GENERATORS

Python provides a generator to create your own iterator function. A generator is a special type of function which does not return a single value, instead, it returns an iterator object with a sequence of values. In a generator function, a yield statement is used rather than a return statement.

Differences between Generator function and a Normal function
1. Generator function contains one or more yield statement.
2.Compared with class level iterators, generators are very easy to use
3.Improves memory utilization and performance.
4.Generators are best suitable for reading data from large number of large files
5.Generators work great for web scraping and crawling.

Yield vs. Return
The Yield statement is responsible for controlling the flow of the generator function.
The Return statement returns a value and terminates the whole function.

Example:
```
def MultipleYield():
    str1 = "First String"
    yield str1
    str2 = "Second string"
    yield str2
    str3 = "Third String"
    yield str3
Obj= MultipleYield()
print(next(Obj))
print(next(Obj))
print(next(Obj))
```

Example:
```
def MultipleYield():
    str1 = "First String"
    return str1
    str2 = "Second string"
    yield str2
    str3 = "Third String"
    yield str3
Obj= MultipleYield()
print(next(Obj))
print(next(Obj))
print(next(Obj))
```

Example:
```
def Cube():
    n=1
    while n<=10:
        result=n**3
        n+=1
        return(result)
x=Cube()
print(x)
```

```
Example:
def Cube():
    n=1
    while n<=10:
        result=n**3
        n+=1
        yield result
x=Cube()
for i in x:
    print(i)

Example: Generating Genetrator Object
def Remote_Control():
    yield "CNN"
    yield "ABC"
    yield "ESPN"
itr=Remote_Control()
print(itr) #<generator object Remote_Control at 0x000000F7D12EFC50>

Example:Generator as Interator
def Remote_Control():
    yield "CNN"
    yield "ABC"
    yield "ESPN"
itr=Remote_Control()
print(next(itr))
print(next(itr))
print(next(itr))

Example:
def table(n):
    for i in range(1,11):
        yield n*i
        i = i+1
for i in table(15):
    print(i)

Example:
def table(n):
    for i in range(1,11):
        return n*i
        i = i+1
print(table(5))

Example:
def NumGen(n):
    x=1
    while x<=n:
        yield x
        x=x+2
Nums=NumGen(4)
for i in Nums:
    print(i)

Example:
def FebGen():
    x,y=0,1
    while True:
        yield x
        x,y=y,x+y
for i in FebGen():
    if i<100:
        print(i)
    else:
        break
```

Example:

```
from random import *
def NumGen():
    Nums="0123456789"
    while True:
        patt=''
        for i in range(4):
            patt+=choice(Nums)
        yield patt
for i in NumGen():
    print(i)
```

List Comprehension vs Generators Expression in Python

List Comprehension

It is one of the best ways of creating the list in one line of Python code. It is used to save a lot of time in creating the list.

Example:

```
print([j**2 for j in range(1,11)])
```

Generator Expression

It is one of the best ways to use less memory for solving the same problem that takes more memory in the list compression.

Example:

```
PyGe=(j**2 for j in range(1,11))
for j in PyGe:
    print(j, end=' ')
```

Example:

```
def infinite_sequence():
    num = 0
    while True:
        yield num
        num += 1
for i in infinite_sequence():
    print(i)
```

Example:

```
def infinite_sequence():
    num = 0
    while True:
        return num
        num += 1
print(infinite_sequence())
```

PYTHON CLOSURES:

A Closure is a function object that remembers values in enclosing scopes even if they are not present in memory.

Example:
```
def OuterFun():
    x=10
    def InnerFun():
        print(x)
    InnerFun()
OuterFun()
```

Example:
```
def OuterFun():
    x=10
    def InnerFun():
        print(x)
InnerFun()
OuterFun()
```

Example:
```
def OuterFun():
    x=10
    def InnerFun():
        print(x)
    return InnerFun()

OO=OuterFun()
print(OO)
```

Example:
```
def OuterFun():
    x=10
    def InnerFun():
        y=20
        Total=x+y
        return Total
    return InnerFun()#Executing function definition

OO=OuterFun()
print(OO)
```

Example:
```
def OuterFun():
    x=10
    def InnerFun():
        y=20
        Total=x+y
        return Total
    return InnerFun #Returing function reference

OO=OuterFun()
print(OO)
```

Example:
```
def OuterFun():
    x=10
    def InnerFun():
        y=20
        Total=x+y
        return Total
    return InnerFun #Returing function reference

OO=OuterFun()
print(OO.__name__)
```

Example:

```
def OuterFun():
    x=10
    def InnerFun():
        y=20
        Total=x+y
        return Total
    return InnerFun #Returing function reference

OO=OuterFun()
print(OO())
```

NOTE:
We executed inner function body outside its scope, this technique is called Closure.

Example:

```
 def Outer_Func():
    Msg="PYTHON"
    def Inner_Func():
        print(Msg)
    return Inner_Func
My_Func=Outer_Func()
My_Func()
```

Example:

```
def Outer_Func(msg):
    message=msg
    def Inner_Func():
        print(message)
    return Inner_Func
hi_func=Outer_Func("Hi")
hello_func=Outer_Func("Hello")
hi_func()
hello_func()
```

When we create closures:
1. Nested Functions
2. Nested Function must refer values in enclosing scope
3. Enclosing function must return nested function

Why to use Closures?
1 We can avoid gloable values
2 They provide some sort of data hiding.
3 This helps us to reduce the use of global variables.
4. We can implement in Decorators

Functions are Objects
Example:

```
def Fun():
    print("Hello")
print(Fun)
Gun=Fun
print(Gun)
Gun()
```