Iterators, Generators, Lambda

Iterators

- iter() or **iter**()
- next() or **next**()

In [1]:

```
# print 1 to 5 numbers
for i in range(1,6):
    print(i,end=" ")
```

```
1 2 3 4 5
```

In [5]:

```
# print 1 to 5 numbers using iterators
mylist =[1,2,3,4,5]
temp=iter(mylist)
print(temp)
```

```
<list_iterator object at 0x00000226CB3F4B80>
```

In [6]:

```
print(next(temp))
print(next(temp))
print(next(temp))
print(next(temp))
print(next(temp))
```

```
1
2
3
4
5
```

In [7]:

```
print(next(temp))
```

```
---------------------------------------------------------------------------
StopIteration                             Traceback (most recent call last)
<ipython-input-7-872946c4992d> in <module>
----> 1 print(next(temp))

StopIteration:
```

In [10]:

```
# print tuple values by using __iter__() & __next__()
a= ("hello",'hi',"How r u?","when?","where?")
t = a.__iter__()
print(t)
print(t.__next__())
```

```
<tuple_iterator object at 0x00000226CB3DA430>
```

Out[10]:

```
'hello'
```

In [11]:

```
print(t.__next__())
```

```
hi
```

In [12]:

```
print(t.__next__())
```

```
How r u?
```

Create Own Iterator

In [14]:

```
class Nums:  # intialize , current value store, incresing value upto "stopIteration"
    MAX = 4
    def __init__(self):
        self.current = 0
    def __iter__(self):
        return self
    def __next__(self):
        next_value = self.current
        if next_value >= self.MAX:
            raise StopIteration
        self.current+=1
        return next_value

nums = Nums()
print(nums)
```

```
<__main__.Nums object at 0x00000226CB3F4C40>
```

In [15]:

```
for num in nums:
    print(num)
```

```
0
1
2
3
```

In [20]:

```
# Infinate Iterator
class Example:
    def __iter__(self):
        self.a = 1
        return self
    def __next__(self):
        x = self.a
        self.a+=5
        return x

# Create Iterator/Class Object
temp = Example()
print(temp)
```

```
<__main__.Example object at 0x00000226CB4D6FD0>
```

In [19]:

```
for t in temp:
    print(t,end= " ")
```

...

In [21]:

```
t1 = Example()
print(t1)
my_temp = iter(t1)
print(my_temp)
```

```
<__main__.Example object at 0x00000226CB4D6B50>
<__main__.Example object at 0x00000226CB4D6B50>
```

In [22]:

```
print(next(my_temp))
```

```
1
```

In [23]:

```
print(next(my_temp))
```

```
6
```

In [24]:

```
print(next(my_temp))
```

```
11
```

Advantages of Iterators

- Cleaner Code
- Iterators can work with Infinate Sequences

- Iterators save Resources

Generators

- Generator is a function that returns an Object which can iterate over as one value at a time.

Return Statement will terminate the entire function

Yield Pauses the function & saving all its sates

In [30]:

```
# print numbers from 0 to 3 using Generator
def gen_nums():
    n = 0
    while n < 4:
        yield n
        n += 1

gen_temp = gen_nums()
print(gen_temp)

```

```
<generator object gen_nums at 0x00000226CB51C120>
```

In [26]:

```
print(list(gen_temp))
```

```
[0, 1, 2, 3]
```

In [29]:

```
for m in gen_temp:
    print(m,end= ' ')
```

```
0 1 2 3
```

In [31]:

```
print(next(gen_temp))
```

```
0
```

In [32]:

```
print(next(gen_temp))
```

```
1
```

In [33]:

```
print(next(gen_temp))
```

```
2
```

In [34]:

```
print(next(gen_temp))
```

```
3
```

Benfits of Generators

- Simplified Code
- Better Performance

In [42]:

```
# Example -2
def mygen1(n):
    for  i in range(n):
        yield i

def mygen2(n,m):
    for j in range(n,m):
        yield j

def mygen3(n,m):
    yield from mygen1(n)
    yield from mygen2(n,m)
    yield from mygen2(n,m)

print(list(mygen1(5)))
print(list(mygen2(5,10)))
print(list(mygen3(0,10)))
```

```
[0, 1, 2, 3, 4]
[5, 6, 7, 8, 9]
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Lambda

- Lambda is a small anonymous Function
- Lambda function can take any number of arguments, but can only have one Expression

In [43]:

```
# Syntax of Lambda:-
#  lambda arguments : expression
```

In [44]:

```
x= lambda a: a+10
print(x(5))
```

15

In [45]:

```
# Multiplication of a,b
mul = lambda a,b : a*b
mul(8,9)
```

Out[45]:

72

In [46]:

```
add =lambda x,y,z : x+y+z
add(1,2,3)
```

Out[46]:

6