(6.3) Function Decorators and Generators

1. Function Decorators

Decorator is a function which can take a function as argument and extend its functionality and returns modified function with extended functionality

The main objective of decorator functions is we can extend the functionality of existing functions without modifies that function

In [129…

```
def wish(name):
    print("Hello",name,"Good Morning")

wish("Pankaj")
wish("Guest")
```

```
Hello Pankaj Good Morning
Hello Guest Good Morning
```

This function can always print same output for any name

But we want to modify this function to provide different message if name is Guest.We can do this without touching wish() function by using decorator.

In [133…

```
def decor(func):
    def inner(name):
        if name=='Guest':
            print("Hello Guest How are You")
        else:
            func(name)
    return inner

@decor
def wish(name):
    print("Hello",name,"Good Morning")


wish('Pankaj')
wish('Guest')
wish('Mukesh')
```

```
Hello Pankaj Good Morning
Hello Guest How are You
Hello Mukesh Good Morning
```

In the above program whenever we call wish() function automatically decor function will be executed

How to call same function with decorator and without decorator

In [134…

```
def decor(func):
    def inner(name):
        if name == 'Guest':
            print("Hello Guest How are You")
        else:
```

```
            func(name)
    return inner


def wish(name):
    print("Hello",name,"Good Morning")

decorfunction=decor(wish)

wish("Pankaj") #decorator wont be executed
wish("Guest") #decorator wont be executed

decorfunction("Pankaj") #decorator will be executed
decorfunction("Guest") #decorator will be executed
```

```
Hello Pankaj Good Morning
Hello Guest Good Morning
Hello Pankaj Good Morning
Hello Guest How are You
```

Problem : Smart Division

In [135…

```
def smart_division(func):
    def inner(a,b):
        print("We are dividing",a,"with",b)
        if b==0:
            print('OOPS...cannot divide')
            return
        else:
            return func(a,b)
    return inner

@smart_division
def division(a,b):
    return a/b

print(division(20,2))
print(division(20,0))
```

```
We are dividing 20 with 2
10.0
We are dividing 20 with 0
OOPS...cannot divide
None
```

Decorator Chaining

We can define multiple decorators for the same function and all these decorators will form Decorator Chaining

In [136…

```
def decor1(func):
    def inner(x):
        if x>0 and x<=10:
            return x**3
        else:
            return func(x)
    return inner


def decor(func):
    def inner(x):
        if x>10 and x<=20:
```

```
            return x**2
        else:
            return func(x)
    return inner


@decor1
@decor
def num(x):
    return x

print(num(5))
print(num(15))
print(num(25))
```

```
125
225
25
```

Generators

Generator is a function which is responsible to generate a sequence of values

We can write generator functions just like ordinary functions, but it uses yield keyword to return values.

In [157…

```
def mygen():
    yield 'A'
    yield 'B'
    yield 'C'

g = mygen() #g have sequence of A,B,C
print(type(g))

print(next(g))
print(next(g))
print(next(g))
print(next(g))
```

```
<class 'generator'>
A
B
C
---------------------------------------------------------------------------
StopIteration                             Traceback (most recent call last)
C:\Users\PANKAJ~1\AppData\Local\Temp/ipykernel_10332/3045530034.py in <module>
     10 print(next(g))
     11 print(next(g))
---> 12 print(next(g))

StopIteration:
```

i) Countdown Program

In [161…

```
def countdown(num):
    print("start Countdown")
    while(num > 0):
        yield num
        num = num-1

values = countdown(5) # value have sequence off 1 2 3 4 5
```

```
for x in values:
    print(x)
```

```
start Countdown
5
4
3
2
1
```

ii) To generate first n number

In [163…

```
def firstn(num):
    n = 1
    while n<=num:
        yield n
        n=n+1

values = firstn(5)
for x in values:
    print(x)
```

```
1
2
3
4
5
```

We can convert generator into list as follows

```
values=firstn(10)
l1=list(values)
```

iii) to generate fibonacci numbers

In [164…

```
def fib():
    a,b=0,1
    while True:
        yield a
        a,b = b,a+b

for f in fib():
    if f>100:
        break
    print(f)
```

```
0
1
1
2
3
5
8
13
21
34
55
89
```

Advantages of Generator Functions:

1. when compared with class level iterators, generators are very easy to use
2. Improves memory utilization and performance.
3. Generators are best suitable for reading data from large number of large files
4. Generators work great for web scraping and crawling.

Generators vs Normal Collections w.r.t performance

In [177…

```
import random
import time

names = ['Ram','Shayam','Mohan','Sohan']
subjects = ['Python','C++','Blockchain']


def people_list(num):
    result = []
    for i in range(num):
        person={
            'id':i,
            'name':random.choice(names),
            'subject':random.choice(subjects)
        }
        result.append(person)
    return result


def people_generators(num):
    for i in range(num):
        person = {
            'id':i,
            'name':random.choice(names),
            'major':random.choice(subjects)
        }
        yield person


t1 = time.time()
people = people_list(1000000)
t2 = time.time()
print('Took {} sec'.format(t2-t1))


t1 = time.time()
people = people_generators(1000000)
t2 = time.time()
print('Took {} sec'.format(t2-t1))
```

```
Took 2.7579288482666016 sec
Took 0.17548847198486328 sec
```

Generators vs Normal Collections w.r.t Memory Utilization

Normal Collection: We will get MemoryError in this case because all these values are required to store in the memory

In [178…

```
l=[x*x for x in range(10000000000000000)]
print(l[0])
```

```
---------------------------------------------------------------------------
MemoryError                               Traceback (most recent call last)
C:\Users\PANKAJ~1\AppData\Local\Temp/ipykernel_10332/4273471219.py in <module>
----> 1 l=[x*x for x in range(10000000000000000)]
```

```
      2 print(l[0])
```

```
C:\Users\PANKAJ~1\AppData\Local\Temp/ipykernel_10332/4273471219.py in <listcomp>(.0)
----> 1 l=[x*x for x in range(10000000000000000)]
      2 print(l[0])
```

```
MemoryError:
```

Generators:We won't get any MemoryError because the values won't be stored at the beginning

In [181…

```
g=(x*x for x in range(10000000000000000))
print(next(g))
print(next(g))
print(next(g))
```

```
0
1
4
```

In [ ]: