Generators

What is generator?

A generator special function, which returns generator iterator object. This iterator object is used to iterate/read values generated by generator function.
In Python, a generator is a function that returns an iterator that produces a sequence of values when iterated over.
Generators in Python are used to create iterators and return a traversal object.
Generator function returns value or generate values using “yield” keyword.

What is yield?

“yield” is a keyword.
This keyword returns value generated by iterator object. After returning value it pause execution of function. When iterator, iterate next value it resume back and continue execution of function.

Syntax:

```
def  <function-name>([parameters]):
        statement-1
        statement-2
        yield <value>/<expression>
```

Basic steps working generator function

1. Create generator function
2. Create generator iterator object
3. Iterator values using next function or for loop

Applications of generators

1. Customized iterator
    a. Create iteator on collections
    b. Create iterator on database
    c. Create iterator on files
    d. Create iteartor for generating series
        i. Month generator
        ii. Year generator
        iii. Days generator
        iv. Factorial generator
        v. Prime generator

vi. Sin generator
vii. Cos generator
viii. Log generator
ix. Ascii generate

Example:

```
def generate_numbers(): # creating generator function
    yield 7
    yield 10
    yield 4
    yield 50
    yield 70


a=generate_numbers() # creating generator iterator object
value1=next(a)
value2=next(a)
print(value1,value2,sep="\n")
value3=next(a)
value4=next(a)
print(value3,value4,sep="\n")
value5=next(a)
print(value5)
```

Output:

```
7
10
4
50
70
```

Example:

```
def reverse_iterator(iterable):
    for i in range(-1,-(len(iterable)+1),-1):
        yield iterable[i]
```

```
list1=[10,20,30,40,50]
a=iter(list1)
value1=next(a)
value2=next(a)
print(value1,value2)

b=reverse_iterator(list1)
v1=next(b)
v2=next(b)
print(v1,v2)
for value in b:
    print(value)
```

Output:

```
10 20
50 40
30
20
10
```

Example:

```
def month_generator():
month_list=['jan','feb','mar','apr','may','jun','jul','aug','sep','oct','nov','dec'
]
    for month_name in month_list:
        yield month_name



month_iterator=month_generator() # creating iterator object
for month in month_iterator:
    print(month)

month_iterator=month_generator()
sales_dict=dict.fromkeys(month_iterator)
```

```
print(sales_dict)
sales_dict['jan']=50000
print(sales_dict)
```

Output:

```
jan
feb
mar
apr
may
jun
jul
aug
sep
oct
nov
dec
{'jan': None, 'feb': None, 'mar': None, 'apr': None, 'may': None, 'jun': None,
'jul': None, 'aug': None, 'sep': None, 'oct': None, 'nov': None, 'dec': None}
{'jan': 50000, 'feb': None, 'mar': None, 'apr': None, 'may': None, 'jun': None,
'jul': None, 'aug': None, 'sep': None, 'oct': None, 'nov': None, 'dec': None}
```

Example:

```
def prime_generator(start,stop):
    for num in range(start,stop):
        c=0
        for i in range(1,num+1):
            if num%i==0:
                c=c+1
        if c==2:
            yield num


prime_iterator=prime_generator(3,20)
for value in prime_iterator:
    print(value,end=' ')
```

Output:
3 5 7 11 13 17 19

What is difference between return and yield?

return	**yield**
After returning value it terminates execution of function	After returning value it pause execution of function

Function Recursion or Recursive Function

What is function recursion?
Calling function itself is called recursive function call or function recursion. Recursive functions are functions that calls itself. It is always made up of 2 portions, the base case and the recursive case. The base case is the condition to stop the recursion. The recursive case is the part where the function calls on itself.

Applications of function recursion
Building Algo

- Sorting
- Searching
- Dynamic Programming
- Traversing
- Data Structures

Recursive function is developed using one concept called divide and conq.