A1 Functions is created using keyford ‘def’

##create a function to return a list of odd numbers from 1 to 25

def myFunc(n):

for i in range(1,n):

if i%2!=0:

print(i)

i =i+1

return n

A2 In order to pass variable(unsure) number of arguments to our function we use these 2 types

1 \*args

2 \*\*kwargs

Example

def myadder(\*num):

sum = 0

for n in num:

sum = sum + n

print("Sum:",sum)

myadder(3,5)

myadder(4,5,6,7)

myadder(1,2,3,5,6)

Results:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
Sum: 8

Sum: 22

Sum: 17

A3 Iterator is an object that allows us to iterate/traverse through list, tuple, dictionary, tuple, sets.

\_Iter()\_method initialize iterator

Example method to print the first five elements of the given list [2, 4, 6, 8, 10, 12, 14, 16,

18, 20].

lst= [2,4,6,8,10,12,14,16,18,20]

i =iter(lst)

while (True):

elmnt = next(i)

print(elmnt)

Result

2

4

6

8

10

12

14

16

18

20

A4. A generator function helps us to create an iterator for faster access of data.

Its like abstract container eg range() function useful in very large data sets and can only be iterated only once.

‘yield’ keyword is used to control the flow of generator function, similar to return statement.

It signifies the end of execution and return result to caller function

Example generator function

def test1(a):

n=[]

for i in a:

if type(i) == int:

n.append(i)

return n

Calling my generator function

for i in test\_fab(10):

print(i)

Result

0

1

1

2

3

5

8

13

21

34

A5.

##create a generator function for prime numbers less than 1000.

##Use the next() method to print the first 20 prime numbers

def genprime(a):

for i in range(1,a):

for j in range(2,(i+1)):

if i%j==0:

if i==j:

yield(i)

break

Calling

for i in genprime(1000):

print (i)

Results

2,3,5,7,11,13,……………………..977,983,991,997