Assignment - 23 Iterator, Generator and Decorator

1. Use iter and next method to access all the elements of a given set using while loop.

```
set1={2,3,4,5,6,7,8}
it=iter(set1)
i=1
while i<=len(set1):
    print(next(it))
    i+=1
```

2. Create a generator to produce first n odd natural numbers.

```
def odd_num(n):
    x=1
    while n:
        yield x
        x+=2
        n-=1

for x in odd_num(int(input("enter the value  "))):
    print(x,end="\n")
```

3. Create a generator to produce first n even natural numbers.

```
def odd_num(n):
    x=2
    while n:
        yield x
        x+=2
        n-=1

for x in odd_num(int(input("enter the value  "))):
    print(x,end="\n")
```

4. Create a generator to produce squares of first N natural numbers.

```
def odd_num(n):
    x=1
    while n:
        yield x**2
        x+=1
        n-=1

for x in odd_num(int(input("enter the value  "))):
    print(x,end="\n")
```

5. Create a generator to produce first n terms of Fibonacci series.

```
def fib_series(n):
    x,y=0,1
    while n:
        yield x
        x,y=y,x+y
        n-=1

for x in fib_series(int(input("enter the value  "))):
    print(x,end="\n")
```

6. Create a generator to produce first n prime numbers.

```
def n_prime(n):
    c=2
    while n:
        for x in range(2,c):
            if c%x==0:
                break
        else:
            yield c
            n-=1
        c+=1
for x in n_prime(int(input("enter the value  "))):
    print(x,end="\n")
```

7. Create an endless iterator using generator method to produce terms of Fibonacci series.

```
def fib_series():
    x,y=0,1
    while True:
        yield x
        x,y=y,x+y

it=fib_series()
l1=[]
while True:
    reply=(input("do you want to generate new element[ha/na]  "))
    if reply=='ha':
        l1.append(next(it))
    else:
        break
print(l1)
```

8. Create an endless iterator using generator method to produce Prime numbers.

```
def fib_series():
    c=2
    while True:
        for x in range(2,c):
            if c%x==0:
                break
```

```
        else:
            yield c
        c+=1
it=fib_series()
l1=[]
while True:
    reply=(input("do you want to generate new element[ha/na]  "))
    if reply=='ha':
        l1.append(next(it))
    else:
        break
print(l1)
```

9. Define a function which takes lengths of three sides of a triangle as arguments and display the perimeter or triangle. Define and apply a decorator which checks for the validity of the triangle on the basis of lengths of the side. This makes the perimeter to be displayed when the triangle can exist with the given lengths of the sides.

```
def checkValidity(a, b, c):

  if (a + b <= c) or (a + c <= b) or (b + c <= a) :
    return False
  else:
    return True
print("enter the three number")
a,b,c=int(input()),int(input()),int(input())
if checkValidity(a, b, c):
  print("Valid")
else:
  print("Invalid")
```

10. Define a function which calculates HCF of two numbers. Define and apply a decorator to check whether two given numbers are co-prime or not.

```
idef fun(n1,n2):
   z=(n1 if n1<n2 else n2)
   for x in range(z,0,-1):
        if n1%x==0 and n2%x==0:
            break
   return x
def __gcd(a, b):

  if (a == 0 or b == 0):  return 0

  if (a == b): return a

  if (a > b):
    return __gcd(a - b, b)
  return __gcd(a, b - a)

def coprime(a, b):

  if ( __gcd(a, b) == 1):
```

```
    print("Co-Prime")
  else:
    print("Not Co-Prime")

print("enter the two number ")
a,b=int(input()),int(input())
coprime(a, b)
res=fun(a,b)
print("hcf is ",res)
```