Python Fundamentals day 12

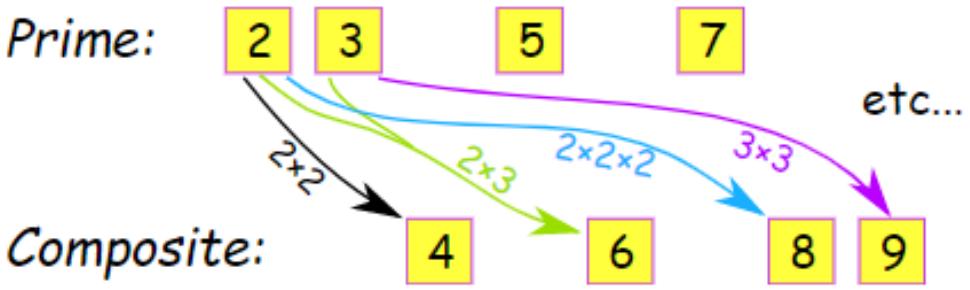
Today's Agenda

- Break statement
- Continue statement
- Lambda functions

Break statement

Before we know what the break statement does, let us see where and why we should use it. Let us consider an example of prime number

A prime number is a number which is divisible by 1 and the number itself. If a number is divisible by any other number 1 and number itself then the number is not a prime number or also called as composite numbers.



```
n=int(input("Enter a number\n"))

for i in range(2,n+1):
    if n%i==0:
        pass
if i==n:
    print(n,"is prime")
else:
    print(n,"is not prime")
```

Output:

```
In [10]: runfile('C:/Users/rooman/OneDrive/
Desktop/python/test.py', wdir='C:/Users/rooman/
OneDrive/Desktop/python')

Enter a number
5
5 is prime
```

pass is a null statement. The interpreter does not ignore a **pass** statement, but nothing happens and the statement results into no operation. The **pass** statement is useful when you don't write the implementation of a function but you want to implement it in the future.

Let us see if the same logic works when we enter a non-prime number

```
In [11]: runfile('C:/Users/rooman/OneDrive/
Desktop/python/test.py', wdir='C:/Users/rooman/
OneDrive/Desktop/python')

Enter a number
6
6 is prime
```

We can see that there is a mistake in logic because 6 is divisible by 2 and 3 before being divisible by number itself therefore it is not a prime number.

So when we check for divisibility of 2, 3 and so on, once we encounter perfect divisibility the current loop should break and resume the next statements. This is done by break statement.

Let us see the above example using break statement

```
n=int(input("Enter a number\n"))

for i in range(2,n+1):
    if n%i==0:
        break
if i==n:
    print(n,"is prime")
else:
    print(n,"is not prime")
```

Output:

```
In [12]: runfile('C:/Users/rooman/OneDrive/
Desktop/python/test.py', wdir='C:/Users/rooman/
OneDrive/Desktop/python')

Enter a number
5
5 is prime

In [13]: runfile('C:/Users/rooman/OneDrive/
Desktop/python/test.py', wdir='C:/Users/rooman/
OneDrive/Desktop/python')

Enter a number
6
6 is not prime
```

We see that now the logic works perfect for both prime and non-prime numbers.

- Let us see some short cut operators with the following example

```
sum=5
print(sum)
sum = sum+5
print(sum)
```

Output:

```
In [14]: runfile('C:/Users/rooman/OneDrive/
Desktop/python/test.py', wdir='C:/Users/rooman/
OneDrive/Desktop/python')
5
10
```

Instead of writing sum = sum + 5 the shorter version of the same expression is sum += 5. Let us verify this

```
sum=5
print(sum)
sum +=5
print(sum)
```

Output:

```
In [15]: runfile('C:/Users/rooman/OneDrive/
Desktop/python/test.py', wdir='C:/Users/rooman/
OneDrive/Desktop/python')
5
10
```

Same applies for multiplication, division, subtraction etc

Continue statement

Let us first consider an example of calculating even and odd sum

```
even_sum,odd_sum = 0,0
n=int(input("Enter the value of n:\n"))
for i in range(1,n+1):
    if i%2==0:
        even_sum += i

    odd_sum += i

print("Sum of all even numbers is:",even_sum)
print("Sum of all odd numbers is:",odd_sum)
```

Output:

```
In [1]: runfile('C:/Users/rooman/OneDrive/
Desktop/python/test.py', wdir='C:/Users/rooman/
OneDrive/Desktop/python')

Enter the value of n:
5
Sum of all even numbers is: 6
Sum of all odd numbers is: 15
```

There is some problem with the odd sum logic. That is, once we know the number is even the odd sum statement should not execute, instead i value should increment and if statement should execute again. This is where continue statement would fit in properly.

The continue statement rejects all the remaining statements in the current iteration of the loop and moves the control back to the top of the loop. The continue statement can be used in both while and for loops.

```
even_sum,odd_sum = 0,0
n=int(input("Enter the value of n:\n"))
for i in range(1,n+1):
    if i%2==0:
        even_sum += i
        continue

    odd_sum += i

print("Sum of all even numbers is:",even_sum)
print("Sum of all odd numbers is:",odd_sum)
```

Output:

```
In [2]: runfile('C:/Users/rooman/OneDrive/
Desktop/python/test.py', wdir='C:/Users/rooman/
OneDrive/Desktop/python')

Enter the value of n:
5
Sum of all even numbers is: 6
Sum of all odd numbers is: 9
```

Lambda functions

In python whenever we want to create a function we can use two keywords def and lambda.

We have seen earlier that when we create a function using def we have to name the function. Whereas in lambda functions there is no name for the function or we can say that it is anonymous function.

Syntax:

lambda arguments : expression

lambda functions are single line, single use or one time use functions.

Let us see how to create one and how to call lambda functions.

```
res = (lambda num,p : num**p) (2,5)
print(res)
```

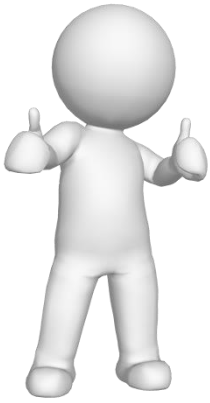
Output:

```
In [1]: runfile('C:/Users/rooman/OneDrive/
Desktop/python/test.py', wdir='C:/Users/rooman/
OneDrive/Desktop/python')
32
```

What if we want to call the lambda function more than once? Is it possible? Because without name how can a function be called? If these are the questions in back of your mind, let us see the next example and get the doubts cleared

```
fun = lambda num,den : num/den
res = fun(100,2)
print(res)

res1 = fun(10,2)
print(res1)
```



Output:

```
In [3]: runfile('C:/Users/rooman/OneDrive/
Desktop/python/test.py', wdir='C:/Users/rooman/
OneDrive/Desktop/python')
50.0
5.0
```

We can see that, to use lambda function more than once we have to assign it a reference and pass the inputs using that reference as many times as we want.