Iteration statements or loop statements allow us to execute a block of statements as long as the condition is true.

Loops statements are used when we need to run same code again and again, each time with a different value.

In Python Iteration (Loops) statements are of three type :-

1. While Loop
2. For Loop
3. Nested For Loops

1. While Loop In Python

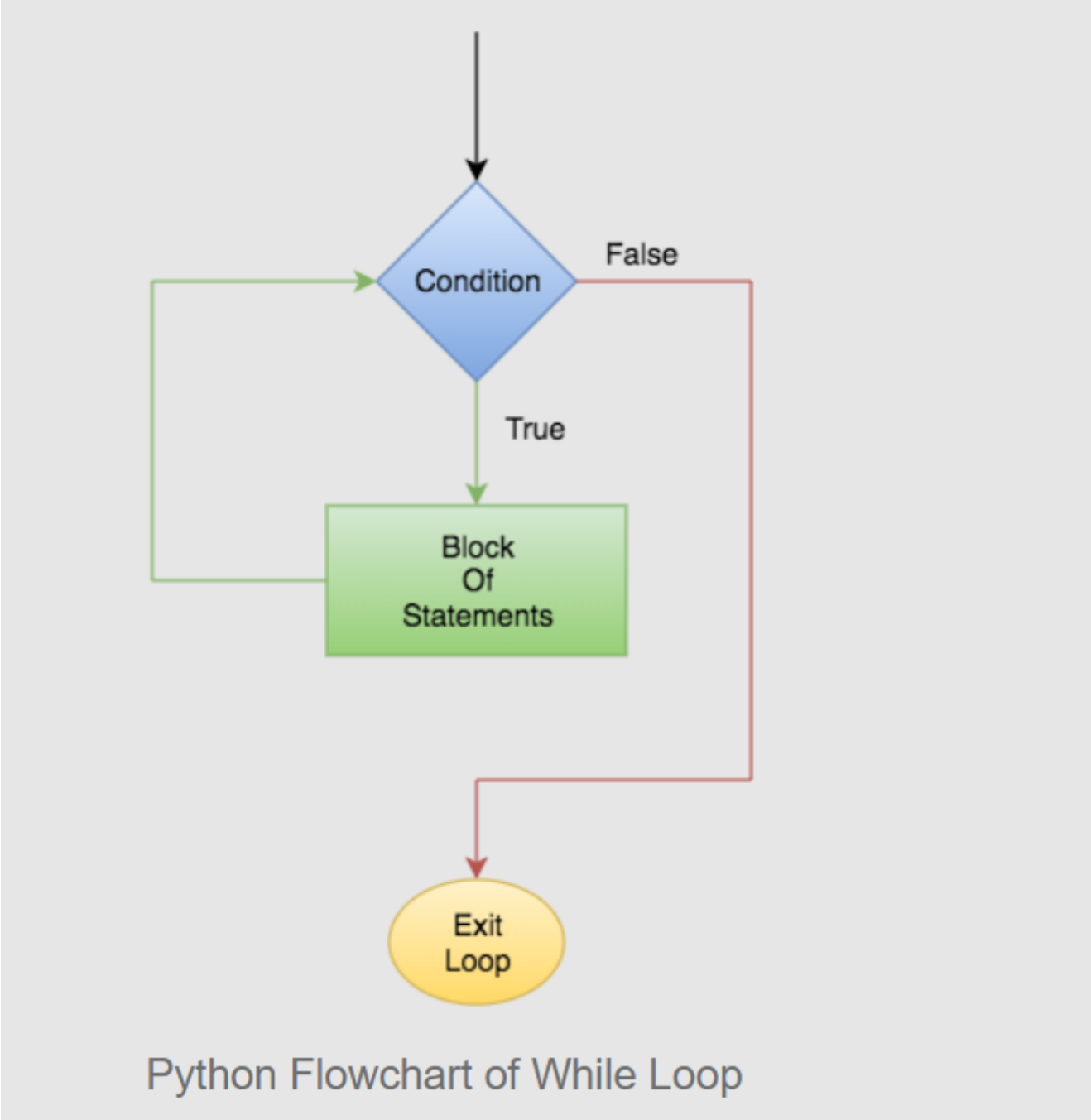
While Loop In Python is used to execute a block of statement as long as a given condition is true. And when the condition is false, the control will come out of the loop.

The condition is checked every time at the beginning of the loop.

```
##### While Loop Syntax

while (condition):
statements

```



In [4]:

```
x = 0
while(x<5):
    print(x)
    x=x+1
```

```
0
1
2
3
4
```

While Loop With Else In Python

The else part is executed if the condition in the while loop becomes False.

```
# syntax
while (condition):
```

```
    loop statements
else:
    else statements
```

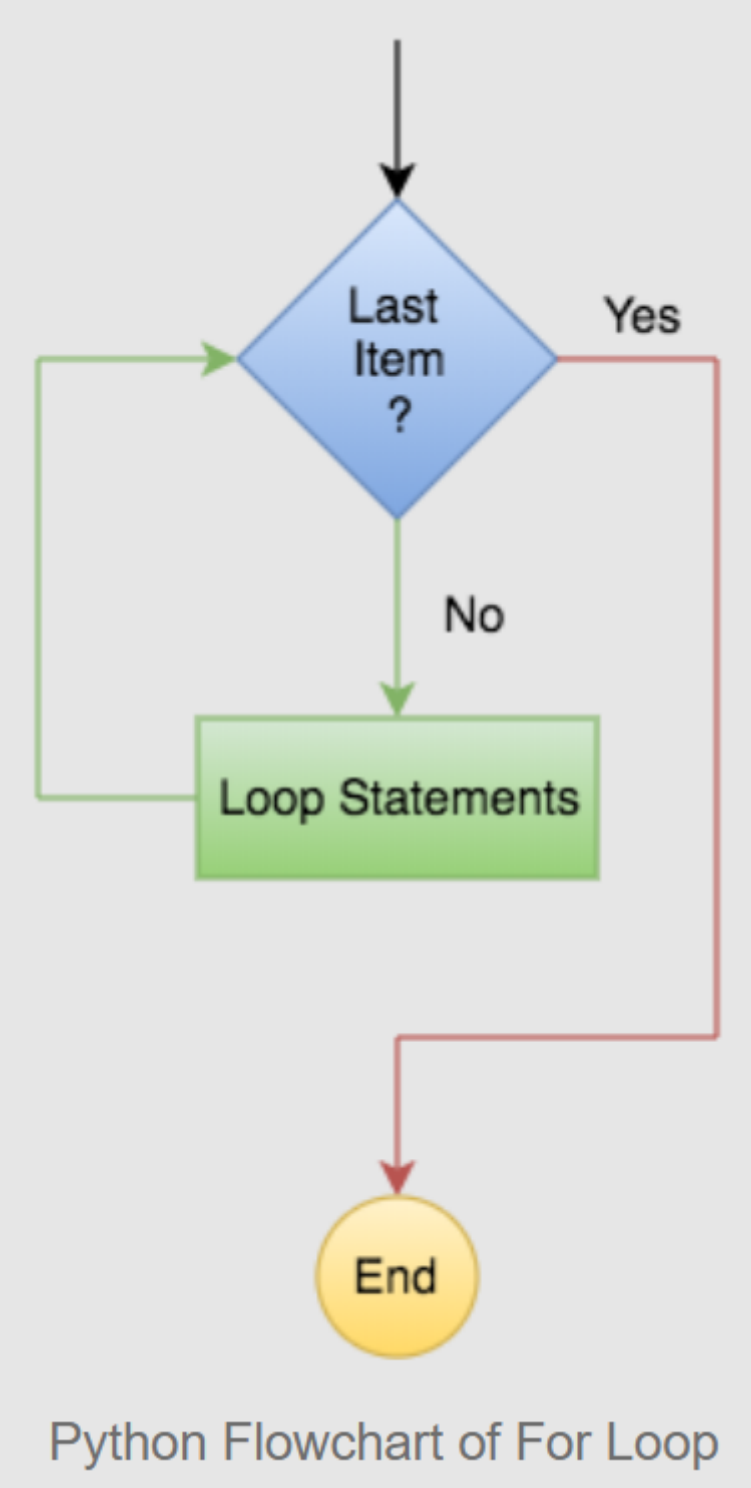
In [6]:

```
x = 1
while (x < 5):
    print('inside while loop value of x is ',x)
    x = x + 1
else:
    print('inside else value of x is ', x)
```

```
inside while loop value of x is  1
inside while loop value of x is  2
inside while loop value of x is  3
inside while loop value of x is  4
inside else value of x is  5
```

2. For Loop In Python

```
# syntax
for val in sequence:
    statements
```



In [7]:

```
for i in range(1,5):
    print(i)
```

```
1
2
3
4
```

The range() Function In Python

The range() function is a built-in that is used to iterate over a sequence of numbers.

- Syntax Of range() Function:

```
range(start, stop[, step])
```

The range() Function Parameters

- start: Starting number of the sequence.
- stop: Generate numbers up to, but not including this number.
- step(Optional): Determines the increment between each numbers in the sequence.

In [8]:

```
# Example 1 of range() function
for i in range(10):
    print(i)
```

```
0
1
2
3
4
5
6
7
8
9
```

In [9]:

```
# Example 2 of range() function
for i in range(2,9):
    print(i)
```

```
2
3
4
5
6
7
8
```

In [10]:

```
# Example 3 of range() function using step parameter
for i in range(2,9,2):
    print(i)
```

```
2
4
6
8
```

In [12]:

```
# Example 4 of range() function
for i in range(0,-10,-2):
    print(i)
```

```
0
-2
-4
-6
-8
```

In [14]:

```
# Example to iterate a list using for loop
my_list = [42, 61, 47, 46, 91, 18]
for i in my_list:
    print(i)
```

```
42
61
47
46
91
18
```

In [27]:

```
# Find Even Number In Python Using For Loop
for i in range(1,11):
    if(i%2 == 0):
        print(i)
```

```
2
4
6
8
10
```

In [31]:

```
# Find Prime Number In Python
StartNo =int(input('enter Start number\n'))
EndNo=int(input('enter End number\n'))
for i in range(StartNo,EndNo+1):
    if i > 1:
        for j in range(2, i):
            if (i % j) == 0:
                print(i, "is not a prime number")
                break
            else:
                    print(i, "is a prime number")
        else:
            print(i, "is not a prime number")
```

```
enter Start number
1
enter End number
10
2 is not a prime number
3 is a prime number
3 is not a prime number
4 is not a prime number
5 is a prime number
5 is a prime number
5 is a prime number
5 is not a prime number
6 is not a prime number
7 is a prime number
7 is a prime number
7 is a prime number
7 is a prime number
7 is a prime number
7 is not a prime number
8 is not a prime number
9 is a prime number
9 is not a prime number
10 is not a prime number
```

3. Nested For loops

Nested for is nothing but a for loop inside another for a loop.

In [15]:

```
rows = 5
for i in range(1, rows + 1):
    for j in range(1, i + 1):
        print("*", end=" ")
    print('')
```

```
*
* *
* * *
* * * *
* * * * *
```

In [4]:

```
for i in range(1,6):
    for j in range(0,i):
        print(i,end=" ")
    print('')
```

```
1
2 2
3 3 3
4 4 4 4
5 5 5 5 5
```

In [23]:

```
for i in range(1,6):
    for j in range(5,i-1,-1):
        print(i,end=" ")
    print('')
```

```
1 1 1 1 1
2 2 2 2
3 3 3
4 4
5
```

In [4]:

```
n = 5
for i in range(1,n+1):
    print ((" ") *(n-i))+((str(i)+ " ")*i))
```

```
    1
   2 2
  3 3 3
 4 4 4 4
5 5 5 5 5
```

In [35]:

```
data = [ ' Verma' ,  ' raj ' ,  ' Gupta ' ,  ' SaNdeeP ' ]
capitalized = [ a.capitalize()  for a in data ]
print (capitalized)
```

```
[' verma', ' raj ', ' gupta ', ' sandeep ']
```

In [33]:

```
data = ['VARMA', 'raj', 'Gupta', 'SaNdeeP']

capitalized = [a.capitalize() for a in data]

print(capitalized)
```

```
['Varma', 'Raj', 'Gupta', 'Sandeep']
```

In [36]:

```
a = ['VARMA', 'raj', 'Gupta', 'SaNdeeP']
res = [x.title() for x in a]
print(res)
```

```
['Varma', 'Raj', 'Gupta', 'Sandeep']
```

In [42]:

```
l = []
for i in range(-100,0):
    if(i%2==0):
        l.append(i)
print(l)
```

```
[-100, -98, -96, -94, -92, -90, -88, -86, -84, -82, -80, -78, -76, -74, -72,
-70, -68, -66, -64, -62, -60, -58, -56, -54, -52, -50, -48, -46, -44, -42, -
40, -38, -36, -34, -32, -30, -28, -26, -24, -22, -20, -18, -16, -14, -12, -1
0, -8, -6, -4, -2]
```

list(range(-100,-1,2))

Feedback: The range function here creates elements from -100 with an increase in value by 2, as the step count of 2 is passed as a third argument.

✓ Correct You missed this!

sorted(set(range(-2,-101,-2)))

Feedback: The range function, range(-2,-101) would create an empty list, as the start value is greater than the end value. Instead, here the inclusion of the third parameter step plays an important role. Since the value specified is -2 it means to consider starting value as -2 and decrease value by 2 each time. This would return elements as -2, -4,-6….-100. Applying a set function would create a set of the range objects. But applying an in-built sorted function on this set would return a sorted list, i.e., -100, -98,-96…..