Programming Fundamentals Lab

Lab # 06

Iteration: While Loop

Break and Continue

Instructor: Hurmat Hidayat

Course Code: CL1002

Semester Spring 2022

Department of Computer Science,
National University of Computer and Emerging Sciences FAST
Peshawar Campus

Contents

Iteration

Iteration means executing the same block of code over and over, potentially many times. A programming structure that implements iteration is called a **loop.** A loop statement allows us to execute a statement or group of statements multiple times.
In programming, there are two types of iteration, indefinite and definite:

- With **indefinite iteration**, the number of times the loop is executed isn't specified explicitly in advance. Rather, the designated block is executed repeatedly as long as some condition is met.
- With **definite iteration**, the number of times the designated block will be executed is specified explicitly at the time the loop starts.

For iteration, Python provides a standard for and while statement

While Loop

The while statement repeats a body of code as long as a condition is true. The while loop is the most simple of the loops in Python. The syntax for the loop is as follows:
Syntax

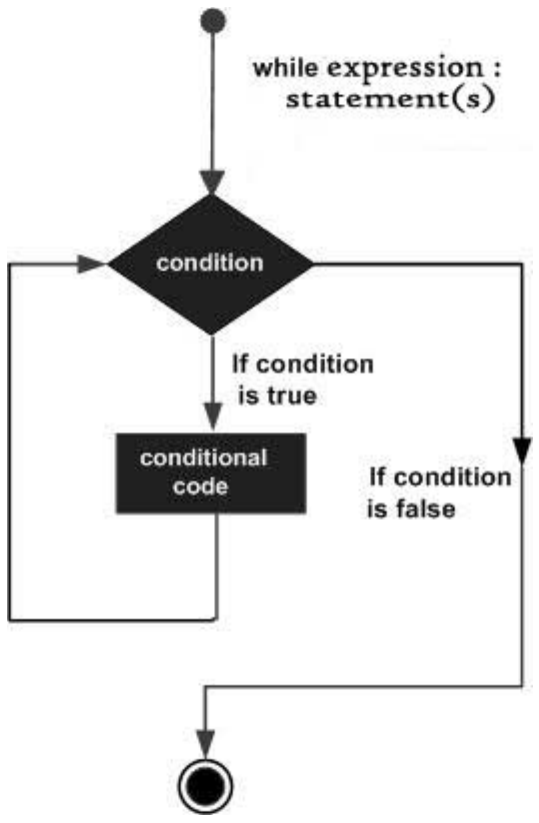
```
while <Boolean expression>:
       stmt1
       stmt2
       ...
       stmtn
stmtA
```

The manner in which this gets executed is as follows:

1. Evaluate the Boolean expression.
2. If it's true
    a. Go ahead and execute stm1 through stmtn, in order.
    b. Go back to step 1.
3. If the Boolean expression is false, skip over the loop and continue to stmtA.

The **Boolean expression** may be any expression, and true is any non-zero value. The loop iterates while the condition is true.
When the condition becomes false, program control passes to the line immediately following the loop.

Flow Diagram



Python program to print any statement multiple times.

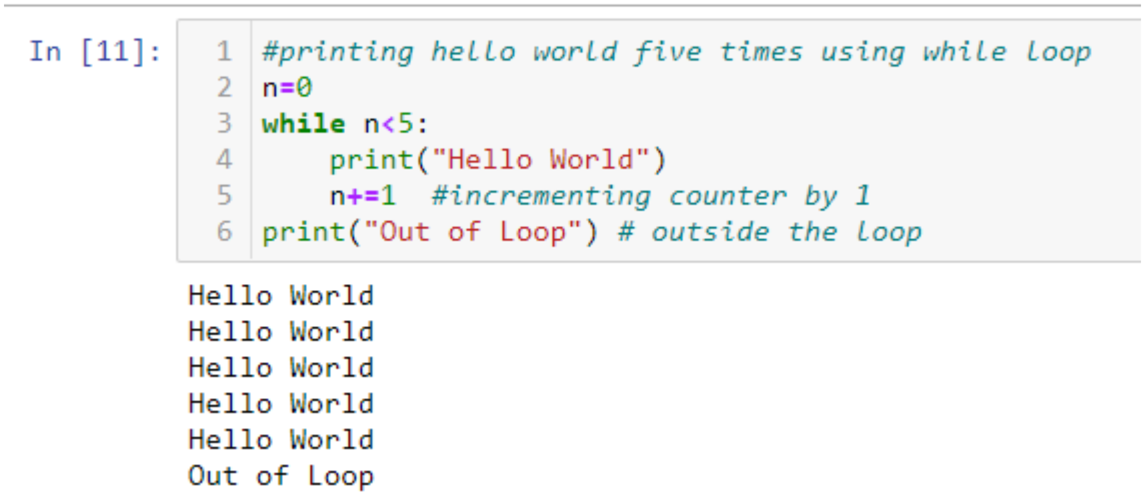
Code:

```
1. count = 1

2. while count <= 3:

3.       print(count, '. I love
        programming in Python!',
        sep='')

4.      count += 1

5. print("\n\n end of loop!!")
```

Trace table:

LINE NUMBER	START: count	CHECK: count <=3	ACTION: Print	INCREMENT: count++	Print: END OF LOOP
1.	1				
2.		true			
3.			print		
4.				2	
2.		true			
3.			print		
4.				3	
2.		true			
3.			print		
4.				4	
2.		false			
5.					End of loop

Example 1 (Printing Hello, world five times using while Loop)



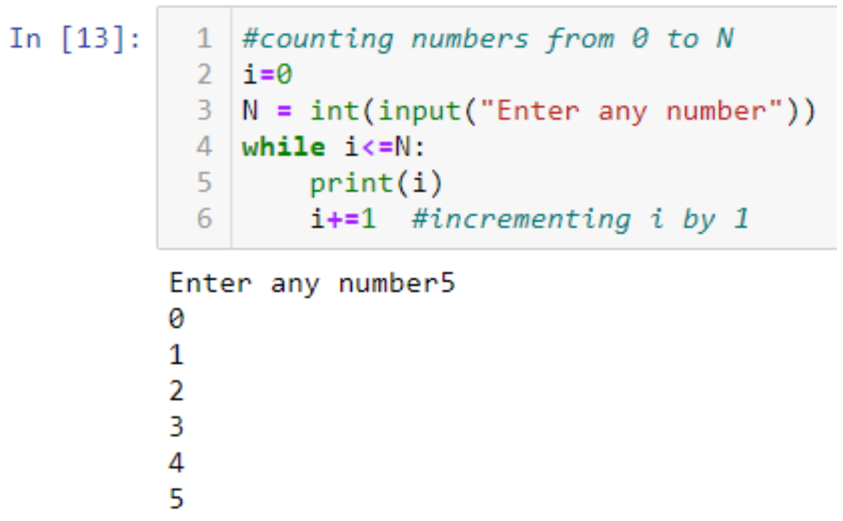
```python
#printing hello world five times using while loop
n=0
while n<5:
    print("Hello World")
    n+=1  #incrementing counter by 1
print("Out of Loop") # outside the loop
```

```
Hello World
Hello World
Hello World
Hello World
Hello World
Out of Loop
```

Prints out the phrase “Hello, world” five times. The condition on the while statement is evaluated at the start of each repetition. If the condition is True, the body of the statement will execute. It is easy to see the structure of a Python while statement due to the mandatory indentation pattern that the language enforces.

Here is another program that prints the number between 0 and N, where N is input:

Example 2 (Counting Numbers from 0 to N using while Loop)

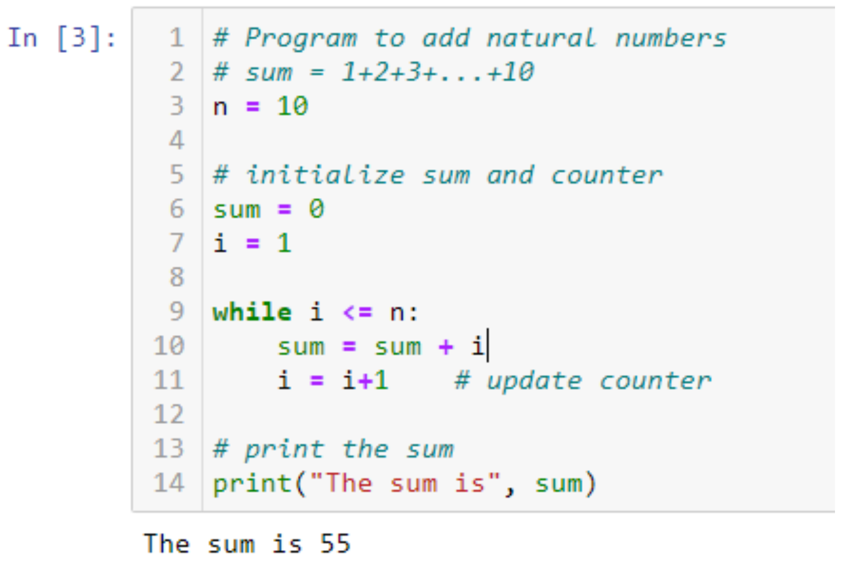


```python
#counting numbers from 0 to N
i=0
N = int(input("Enter any number"))
while i<=N:
    print(i)
    i+=1  #incrementing i by 1
```

```
Enter any number5
0
1
2
3
4
5
```

Note that it is important to make sure that the code block includes a modification of the test: if we had forgotten the line i+=1 in the example above, the while loop would have become an infinite loop. Note that any for loop can be written as a while loop. In practice however, it is better to use a for loop, as Python executes them faster

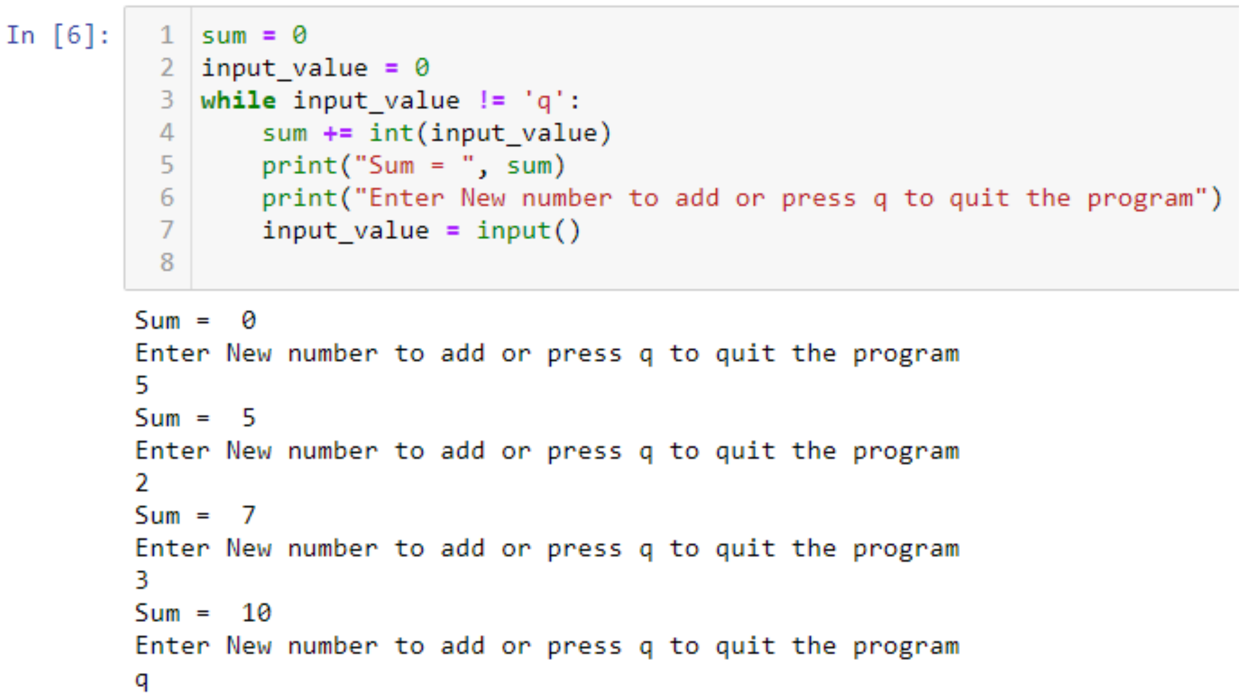
Example 3 (Program to add first 10 Natural Numbers)



```
In [3]:
# Program to add natural numbers
# sum = 1+2+3+...+10
n = 10

# initialize sum and counter
sum = 0
i = 1

while i <= n:
    sum = sum + i
    i = i+1     # update counter

# print the sum
print("The sum is", sum)
```

```
The sum is 55
```

Example 4 (Sum natural numbers until user press q to quit the program)



```
In [6]:
sum = 0
input_value = 0
while input_value != 'q':
    sum += int(input_value)
    print("Sum = ", sum)
    print("Enter New number to add or press q to quit the program")
    input_value = input()

```

```
Sum =  0
Enter New number to add or press q to quit the program
5
Sum =  5
Enter New number to add or press q to quit the program
2
Sum =  7
Enter New number to add or press q to quit the program
3
Sum =  10
Enter New number to add or press q to quit the program
q
```

Break points in control Structures

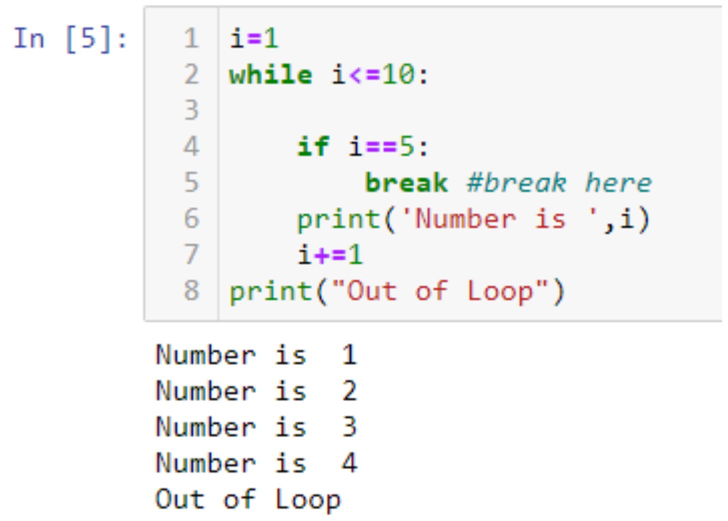
Using for loops and while loops in Python allow you to automate and repeat tasks in an efficient manner.

But sometimes, an external factor may influence the way your program runs. When this occurs, you may want your program to exit a loop completely, skip part of a loop before continuing, or ignore that external factor. You can perform these actions with break, continue, and pass statements.

Break Statement

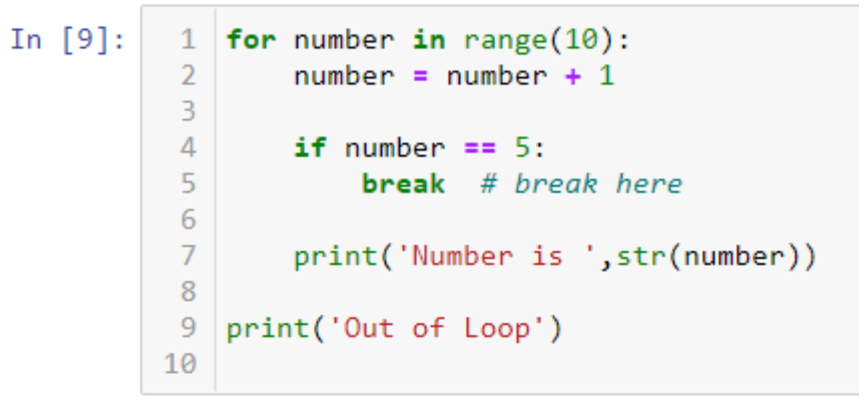
In Python, the **break** statement provides you with the opportunity to exit out of a loop when an external condition is triggered. You'll put the break statement within the block of code under your loop statement, usually after a **conditional if statement**.
Let's look at an example that uses the **break** statement in a **while loop**:

Example 5 (Using break Statement in Loops for and while)



```
In [5]:
i=1
while i<=10:

    if i==5:
        break #break here
    print('Number is ',i)
    i+=1
print("Out of Loop")
```

```
Number is  1
Number is  2
Number is  3
Number is  4
Out of Loop
```



```
In [9]:
for number in range(10):
    number = number + 1

    if number == 5:
        break   # break here

    print('Number is ',str(number))

print('Out of Loop')

```

```
Number is  1
Number is  2
Number is  3
Number is  4
Out of Loop
```

Continue Statement

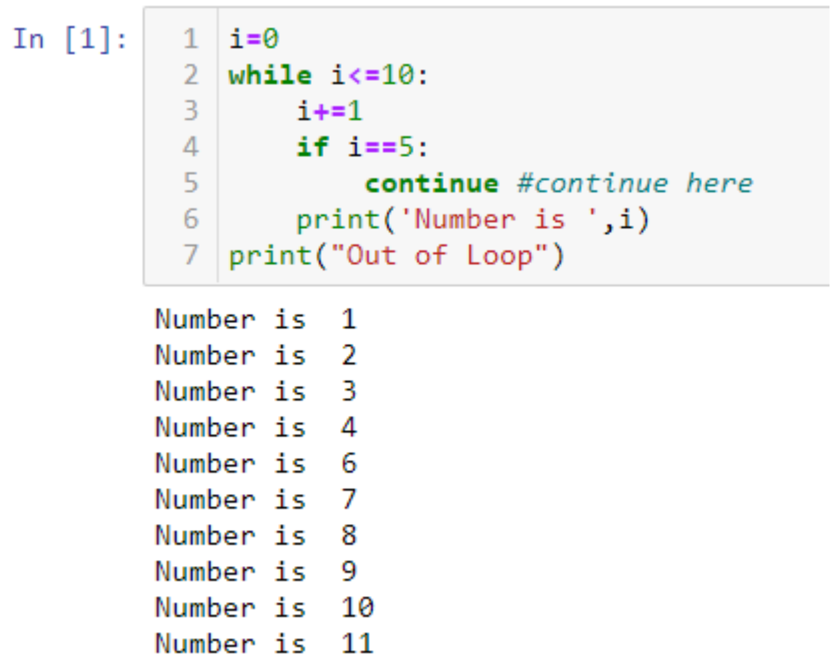
The **continue** statement gives you the option to skip over the part of a loop where an external condition is triggered, but to go on to complete the rest of the loop. That is, the current iteration of the loop will be disrupted, but the program will return to the top of the loop.

The **continue** statement will be within the block of code under the loop statement, usually after a conditional **if** statement.

Using the same **for** loop program as in the Break Statement section above, we'll use a **continue** statement rather than a **break** statement:

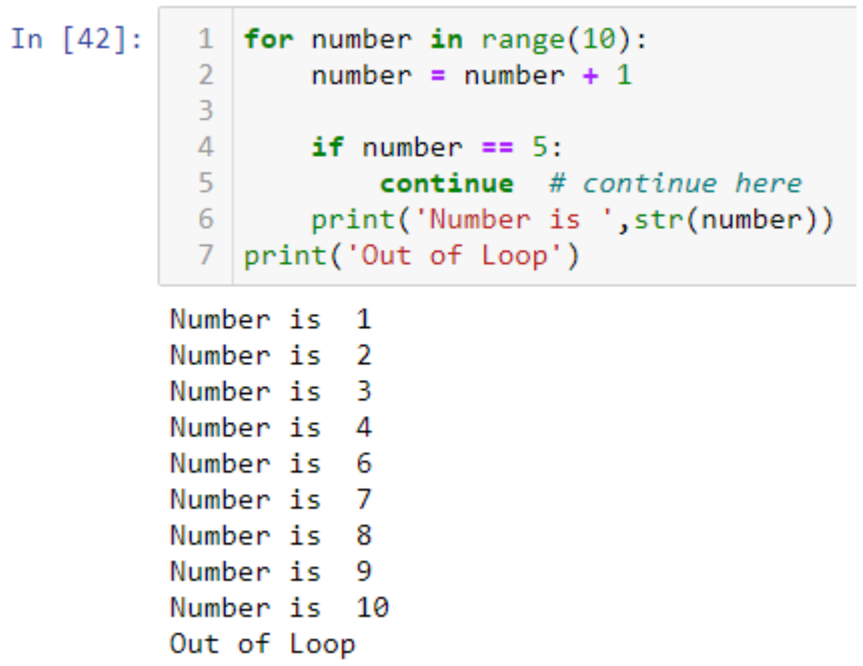
The **break & continue** statements in Python will allow you to use **for loops and while loops** more **effectively** in your code.

Example 6 (Using continue Statement in Loops for and while)



```
In [1]:
i=0
while i<=10:
    i+=1
    if i==5:
        continue #continue here
    print('Number is ',i)
print("Out of Loop")
```

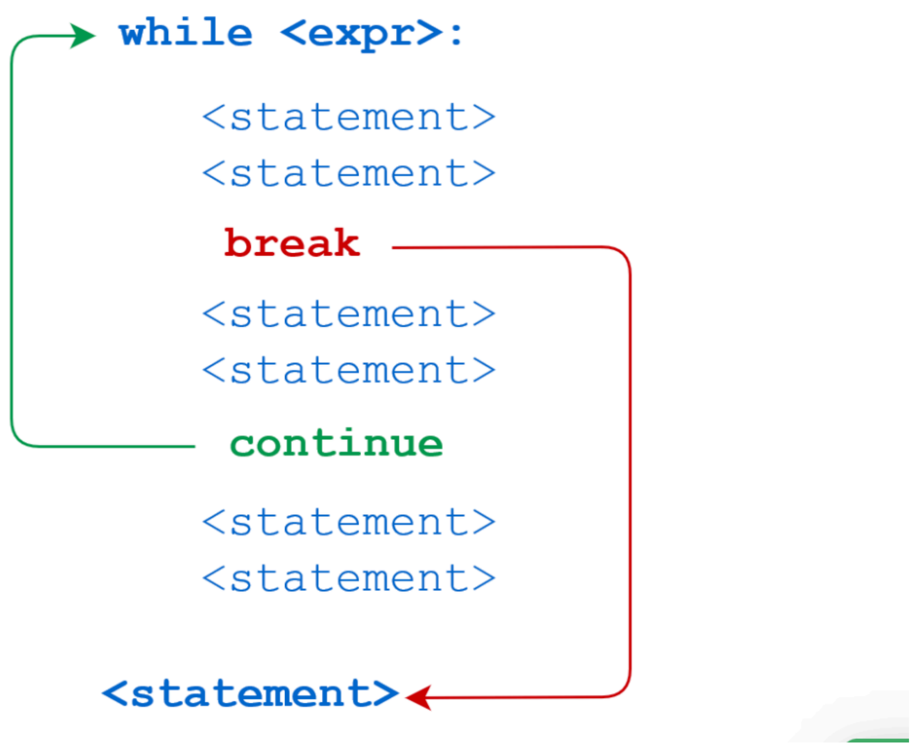
```
Number is  1
Number is  2
Number is  3
Number is  4
Number is  6
Number is  7
Number is  8
Number is  9
Number is  10
Number is  11
```



```
In [42]:
for number in range(10):
    number = number + 1

    if number == 5:
        continue   # continue here
    print('Number is ',str(number))
print('Out of Loop')
```

```
Number is  1
Number is  2
Number is  3
Number is  4
Number is  6
Number is  7
Number is  8
Number is  9
Number is  10
Out of Loop
```

The distinction between break and continue is demonstrated in the following diagram:



References

1. *https://www.tutorialspoint.com/python/python_loops.htm*
2. *https://www.programiz.com/python-programming/for-loop*
3. *https://realpython.com/python-while-loop/*