Python Programming

PhD Students

Department of Physics

Python loops and loop control statements

Dr.Isam Khalil Abdullah

Python loops: This is our first control structure. Ordinarily the computer starts with the first line and then goes down from there. Control structures change the order that statements are executed or decide if a certain statement will be run. A loop statement allows us to execute a statement or group of statements multiple times. Generally, Python programming language provides following types of loops to handle looping requirements.

- ***While loop*** ***:*** Repeats a statement or group of statements while a given condition is TRUE. It tests the condition before executing the loop body

Syntax

The syntax of while loop in python programing is:

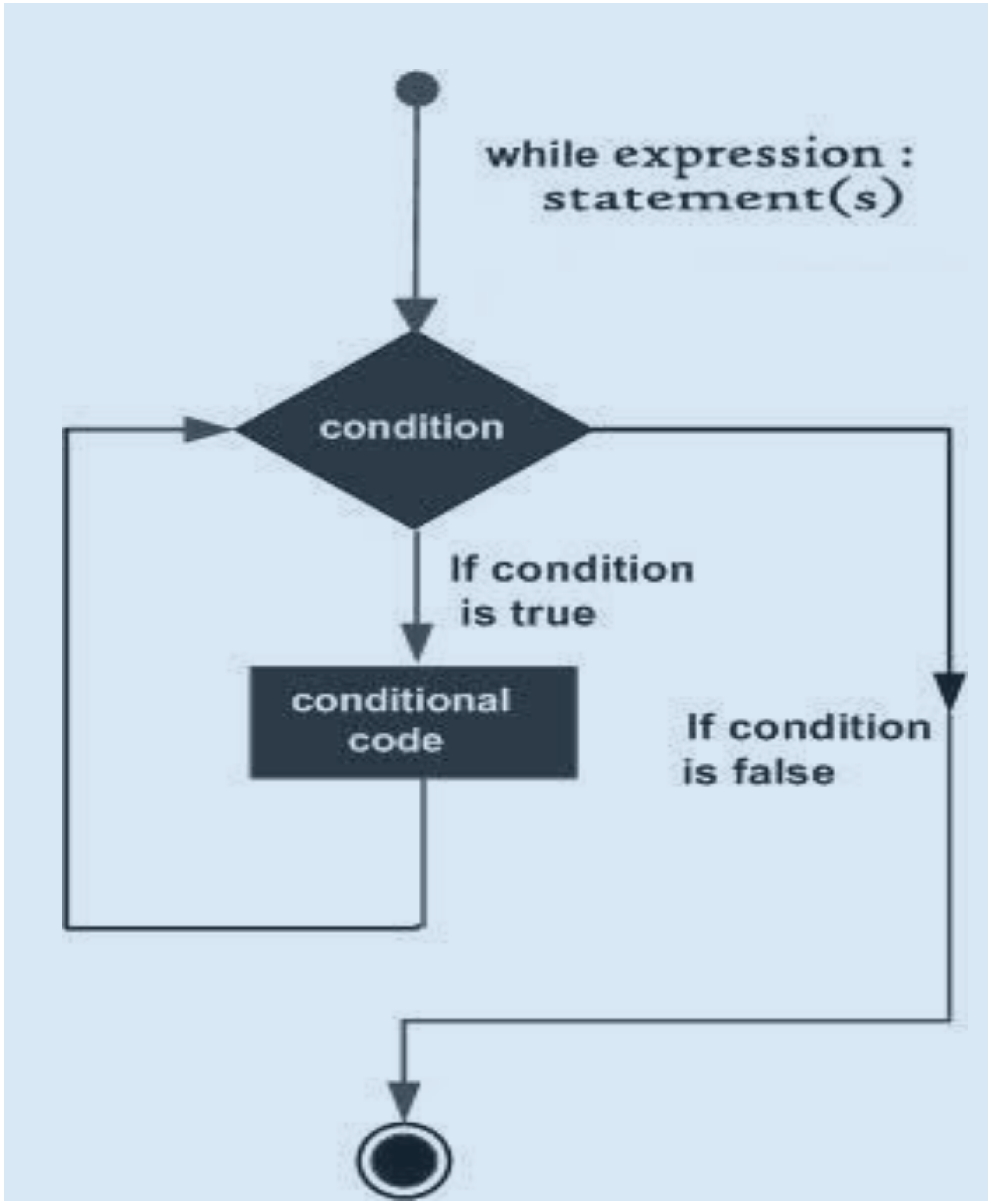
```
while expression:
    statement(s)
```

Here, ***statement(s)*** may be a single or a block of statements. The ***condition*** 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

When the condition is tested and the result is false, the loop body will be skipped and the first statement after the while loop will be executed.



Example:

```
a = 1
while a < 6:
    a += 1          # Same as a = a + 1
    print (a)
```

The output will be

```
2
3
4
5
6
```

Example

```
count = 0
while (count < 9):
    print 'The count is:', count
    count = count + 1
print "Good bye!“
```

The output is:

```
The count is: 0
The count is: 1
The count is: 2
The count is: 3
The count is: 4
The count is: 5
The count is: 6
The count is: 7
The count is: 8
Good bye!
```

Now that we have while loops, it is possible to have programs that run forever. An easy way to do this is to write a program like this:

Example:

```
while 1 ==1:
    print ("Help, I'm stuck in a loop.")
```

The output here is:

```
Help, I'm stuck in a loop

Help, I'm stuck in a loop

…

….

….
```

The way to stop it is to hit the Control (or Ctrl) button and `c' (the letter) at the same time. That will kill the program

Conditional Expressions in Python

Meaning	Math Symbol	Python Symbols
Less than	<	<
Greater than	>	>
Less than or equal	≤	<=
Greater than or equal	≥	>=
Equals	=	==
Not equal	≠	!=

Note:

- There should not be space between the two-symbol.
- Single equal sign is not used for check of equality because single equality is already used for assignments in Python.

- Using ***else*** statements with ***while***:

When we use the ***else*** statement with a ***while*** loop, the ***else*** statement is executed when the condition becomes False.

The following example illustrates the combination of an else statement with a while statement that prints a number as long as it is less than 6, otherwise else statement gets executed.

Example:

```
 count = 0
while count < 6:
   print count, " is less than 6"
   count = count + 1
else:
   print count, " is not less than 6“
```

```
1  is  less than 6
2  is  less than 6
3  is  less than 6
4  is  less than 6
5  is  less than 6
6  is  less than 6
7  is not less than 6
```

for loops in Python

A *for* loop is used to repeat a piece of code *n* number of times. The *for* loop is usually used with a list of things. The basic syntax for the *for* loop looks like this

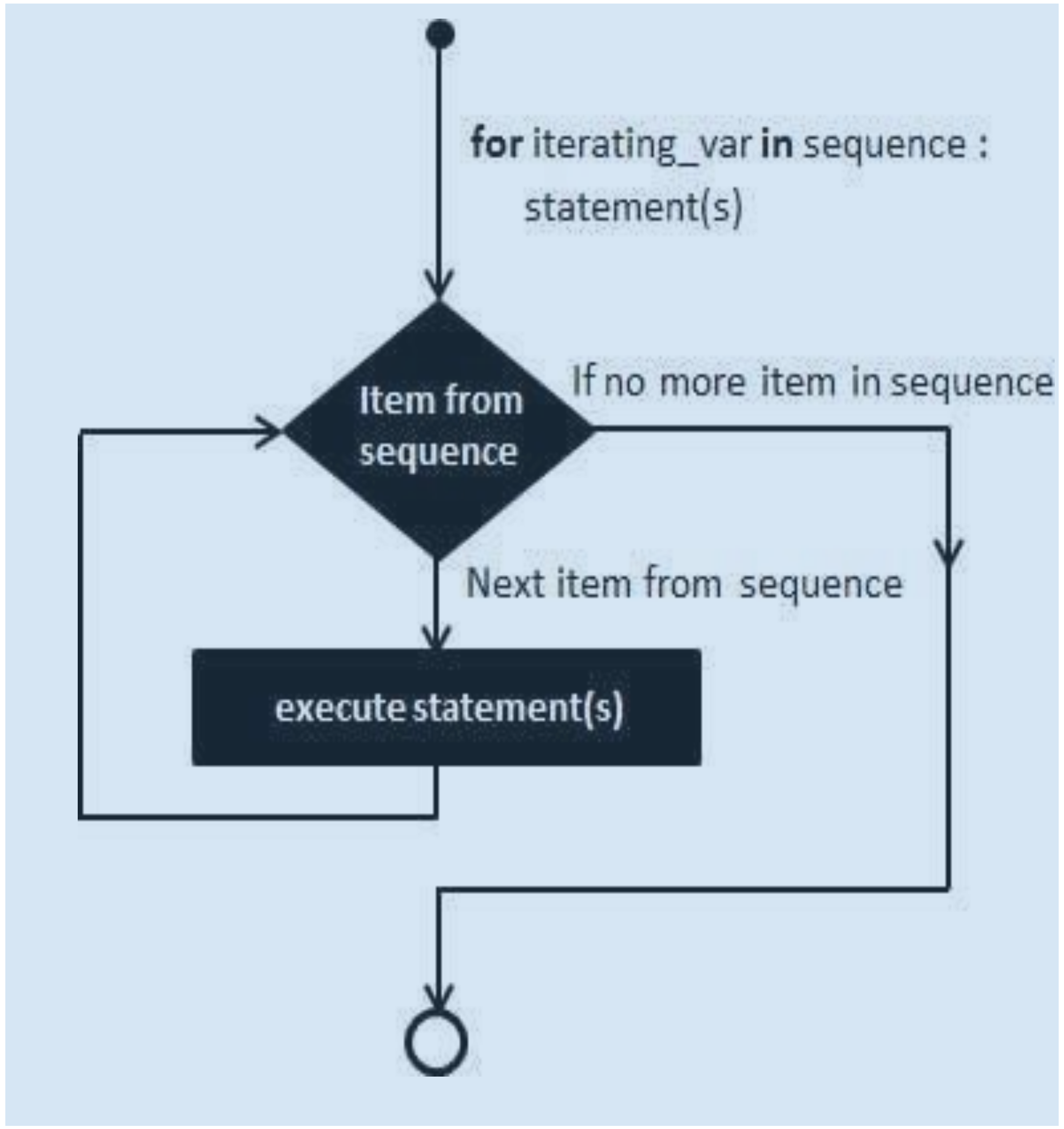
```
for item in list:
        print item
```

Example

```
x = [2, 8, 512]     #Creating a list
for i  in x:        #Here, i is the variable used to refer to individual items in the list
        print i
```

- Every *for* loop must reference a list or a range.
- Every *for* loop must close with a colon.
- Code to be executed as part of the *for* loop ***must*** be indented by four spaces (or one press of the Tab key).
- To use individual items in the list or range, we have to create a variable (item in syntax above or *iterating_var in the syntax below*). There's no need to declare this variable before creating the *for* loop. You can also reuse the same variable for other *for* loops.

Flow Diagram



- ***for*** loop in Python:

Executes a sequence of statements multiple times and abbreviates the code that manages the loop variable.
It has the ability to iterate over the items of any sequence, such as a list or a string

Syntax

```
for iterating_var in sequence:
   statements(s)
```

Here, each item in the list is assigned to *iterating_var*, and the statement(s) block is executed until the entire sequence is exhausted.

Example

```
for letter in 'Python':
    print 'Current Letter :', letter
```

```
Current Letter : P
Current Letter : y
Current Letter : t
Current Letter : h
Current Letter : o
Current Letter : n
```

Example

```
f = ['banana', 'apple',  'mango']
for fruit in f:
    print 'Current fruit :', fruit
print "Good bye!”
```

```
Current fruit : banana
Current fruit : apple
Current fruit : mango
Good bye!
```

Example

```
string = "Hello World"
for x in string:
    print x
```

Example

```
c = ['hey', 5, 'd']
for x in c:
    print x
```

Using else Statement with Loops

When we use the ***else*** statement with a ***for*** loop, the ***else*** statement is executed when the loop has exhausted iterating the list

Example

```
for x in xrange(5):
    print x
else:
    print 'Final x =' ,  x
0
1
2
3
4
Final x = 4
```

Python nested loop

Python programming language allows to use one loop inside another loop. In Python, these are heavily used whenever someone has a list of lists - an iterable object within an iterable object.

Syntax

```
for iterating_var in sequence:
   for iterating_var in sequence:              #   Python nested for loop
      statements(s)
   statements(s)
```

The syntax for a ***nested while loop*** statement in Python programming language is as follows

```
while expression:
   while expression:
      statement(s)                             #  Python nested while loop
   statement(s)
```

Note: you can put any type of loop inside of any other type of loop. For example a for loop can be inside a while loop or vice versa.

Python loop control statements

Loop control statements change execution from its normal sequence. When execution leaves a scope, all automatic objects that were created in that scope are destroyed.

- ***break statement***

The ***break*** statement in Python terminates the current loop and resumes execution at the next statement. In other words, the break statement is used to exit a ***for*** or a ***while*** loop. The purpose of this statement is to end the execution of the loop (for or while) immediately and the program control goes to the statement after the last statement of the loop.

Syntax

```
while (expression1) :
  statement_1
  statement_2
  ......
  if expression2 :
  break
```

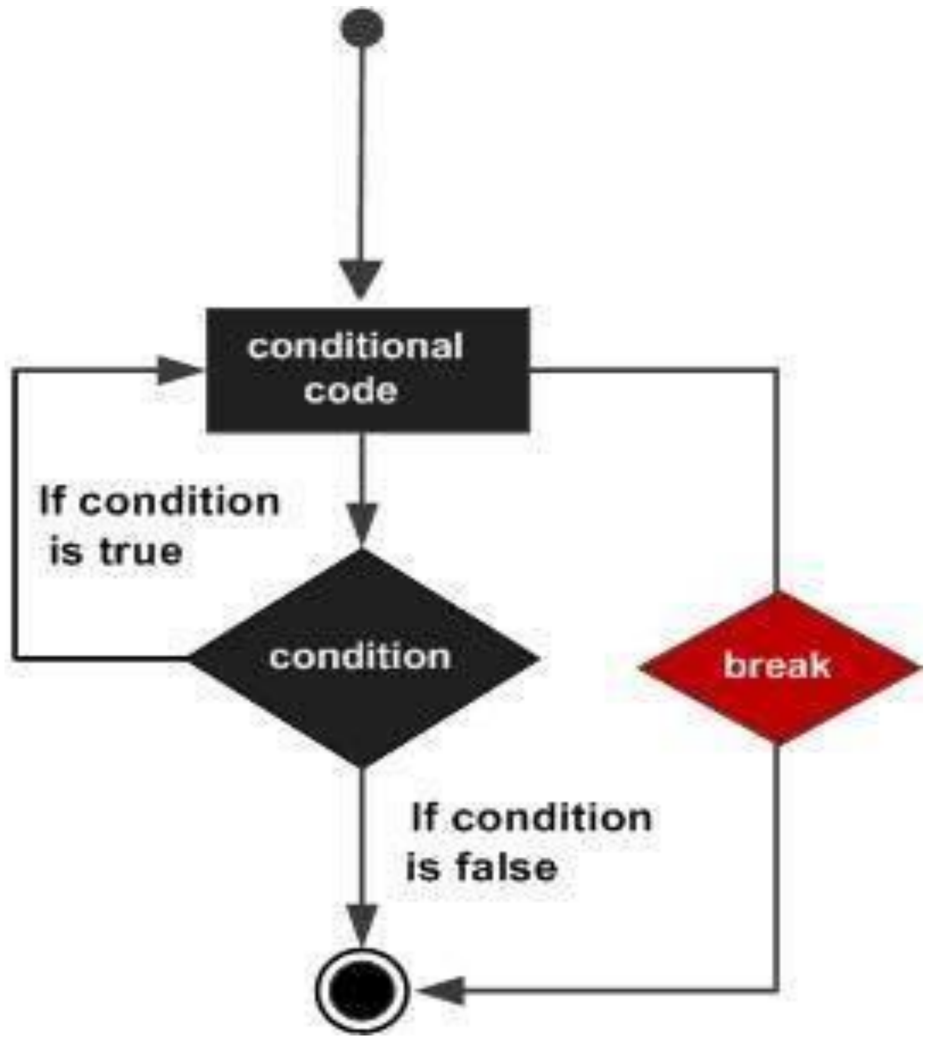
```
for iterating_variable sequence :
   statement_1
   statement_2

   .............
   if expression3 :
     break
```

Example

```
for letter in 'Python':
   if letter == 'h':
      break
   print 'Current Letter :', letter
```

The output will be:

```
Current Letter : P
Current Letter : y
Current Letter : t
```



Example

```
numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9]
sum = 0
count = 0
for x in numbers:
    sum = sum + x
    count = count + 1
    if count == 5:
        break
Print "Sum of first ",count,"integers is : ", sum
```

Here the print statement display the sum of first five elements.

```
Sum of first 5 integers is :  15
```

Example

```
sum = 0
count = 0
while(count<10):
    sum = sum + count
    count = count + 1
    if count== 5:
        break
print "Sum of first ", count, "integers is : ", sum
```

The output is:

```
Sum of first 5 integers is :  10
```

Example

```
num = 10
while num > 0:
   print 'Current number value :', num
   num = num -1
   if num == 5:
      break
print "Good bye!“
```

The output is:

```
Current number value : 10
Current number value : 9
Current number value : 8
Current number value : 7
Current number value : 6
Good bye!
```

- ***continue statement***

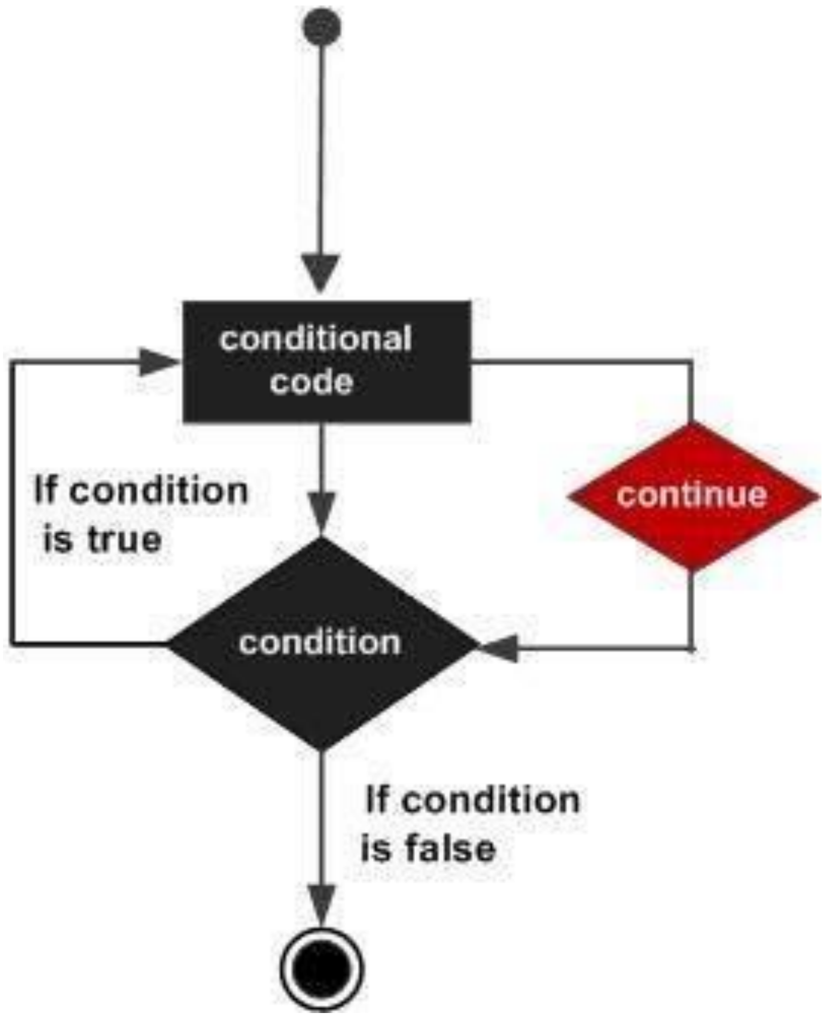
The continue statement is used in a while or for loop to take the control to the top of the loop without executing the rest statements inside the loop. In the other wards, causes the loop to skip the remainder of its body and immediately retest its condition prior to reiterating.

Example

```
for letter in 'Python':
   if letter == 'h':
      continue
   print 'Current Letter :', letter
```

The output here is:

```
Current Letter : P
Current Letter : y
Current Letter : t
Current Letter : o
Current Letter : n
```



Example

```
num = 10
while num > 0:
   num = num -1
   if num == 5:
      continue
   print 'Current number value :', num
print "Good bye!“
```

The computer display this output:

```
Current number value : 9
Current number value : 8
Current number value : 7
Current number value : 6
Current number value : 4
Current number value : 3
Current number value : 2
Current number value : 1
Current number value : 0
Good bye!
```

Example

```
for letter in 'geeksforgeeks':
    if letter == 'e' or letter == 's':
         continue
    print 'Current Letter :', letter                    # prints all letters except ‘e ‘ and  ‘s’
```

```
Current Letter : g
Current Letter : k
Current Letter : f
Current Letter : o
Current Letter : r
Current Letter : g
Current Letter : k
```

Example

```
for letter in 'geeks for geeks':
    if letter == 'e' or letter == 's':
         continue
    print 'Current Letter :', letter
```

Look the change in output

```
Current Letter : g
Current Letter : k
Current Letter :
Current Letter : f
Current Letter : o
Current Letter : r
Current Letter :
Current Letter : g
Current Letter : k
```

- ***pass*** statement

Pass is a null statement. The difference between a comment and pass statement is that, while the interpreter ignores a comment entirely, pass is not ignored. That means nothing happens when it executes.

Example

```
for letter in 'Python':
   if letter == 'h':
      pass
      print 'This is pass block'
   print 'Current Letter :', letter
print "Good bye!“
```

```
Current Letter : P
Current Letter : y
Current Letter : t
This is pass block
Current Letter : h
Current Letter : o
Current Letter : n
Good bye!
```

Example

```
number = 0
for number in range(10):
   number = number + 1
   if number == 5:
      pass    # pass here
   print('Number is ' + str(number))
print('Out of loop')
```

```
Number is 1
Number is 2
Number is 3
Number is 4
Number is 5
Number is 6
Number is 7
Number is 8
Number is 9
Number is 10
Out of loop
```

```
count = 0
while (count < 9):
    print 'The count is:', count
    count = count + 1
print "Good bye!“
```

The output is:

```
The count is: 0
The count is: 1
The count is: 2
The count is: 3
The count is: 4
The count is: 5
The count is: 6
The count is: 7
The count is: 8
Good bye!
```

Questions in python loop:

1- Change the following **Python code** from using a **while loop** to **for loop**:

```
x=1
while x<10:
print x,
x+=1
```

2-
```
x=10
while x>5:
    print x
    x-=1
```

3 Change the following Python code from a **for loop** to a **while loop**:

```
for i in range(1,50):
    print i,
```

4 What would be printed from the following Python code segment?

```
for i in range(20,0,-2):
    print I
```

What would be printed from the following Python code segment?

```
for x in range(1,6):
    for y in range(1,x+1):
        print x,' ',y
```