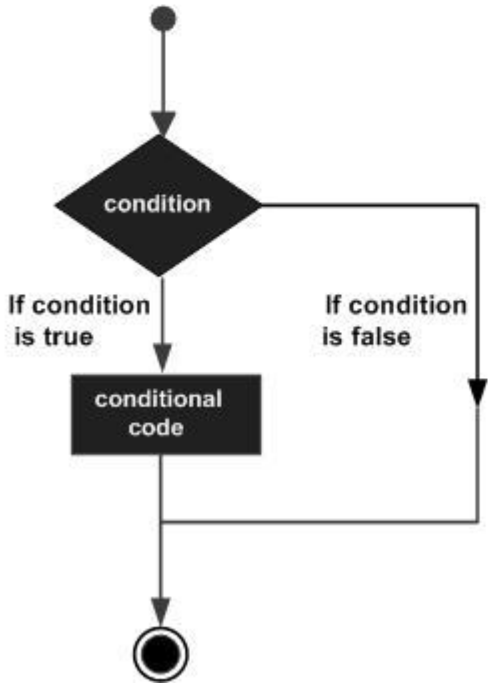
If, else, elif

```
if condition :
        indentedStatementBlockForTrueCondition
```



If, else, elif

```
if condition :
        indentedStatementBlockForTrueCondition
else:
        indentedStatementBlockForFalseCondition
```

```
if condition1 :
        indentedStatementBlockForTrueCondition1
elif condition2 :
        indentedStatementBlockForFirstTrueCondition2
elif condition3 :
        indentedStatementBlockForFirstTrueCondition3
elif condition4 :
        indentedStatementBlockForFirstTrueCondition4
else:
        indentedStatementBlockForEachConditionFalse
```

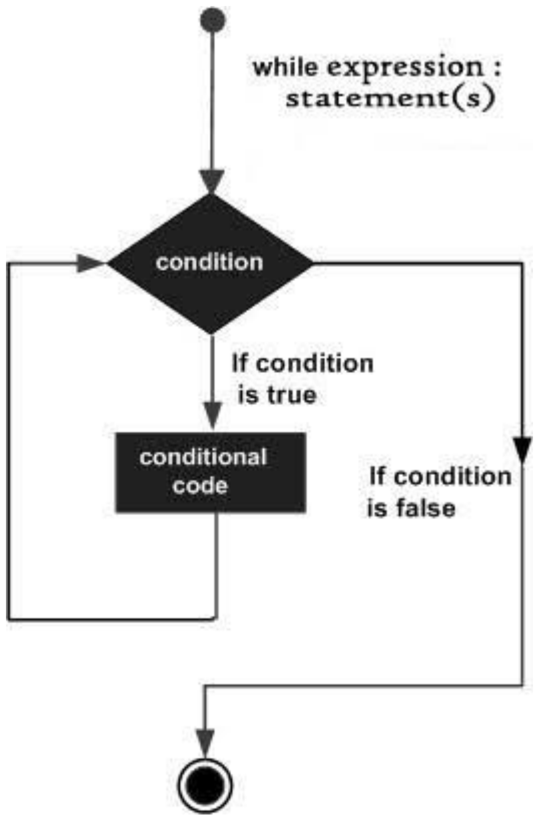
Conditions

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

Operator	Description	Example
and	Called Logical AND operator. If both the operands are true then then condition becomes true.	(a and b) is true.
or	Called Logical OR Operator. If any of the two operands are non zero then then condition becomes true.	(a or b) is true.
not	Called Logical NOT Operator. Use to reverses the logical state of its operand. If a condition is true then Logical NOT operator will make false.	not(a and b) is false.

While loops

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



While Loop

```
count = 0
while (count < 9):
    print 'The count is:', count
    count = count + 1

print "Good bye!"
```

While Loops and files

```
file = open("sample.txt")
while True:
 line = file.readline()
 print "Line is ",line
 if not line:
       break
file.close()
```

But this loops forever!

OK if we eventually come to a break statement.

But what if we don’t?

While Loops to read from a file

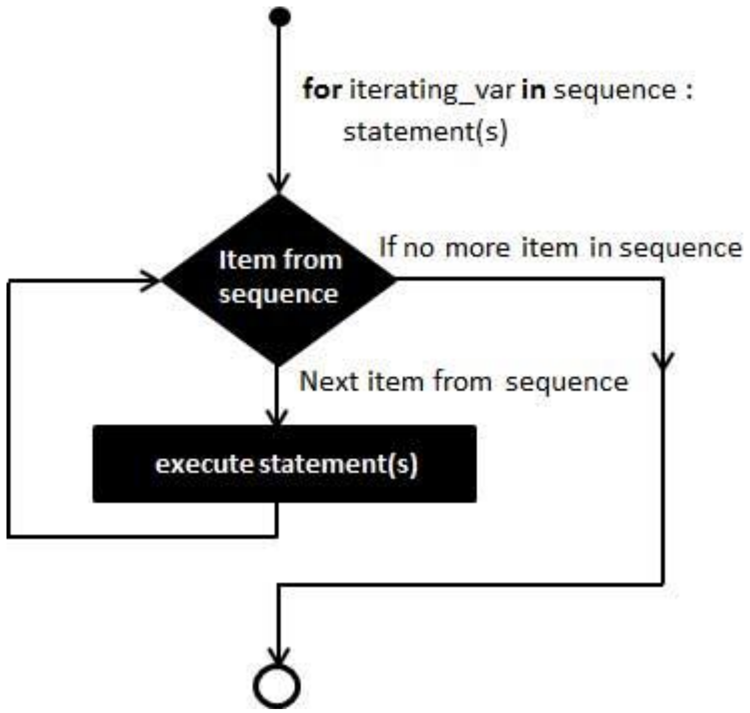
```
file = open("sample.txt")
while True:
        line = file.readline()
        if not line:
                break
        # do something
    file.close
```

But this loops forever!

OK if we eventually come to a break statement.

But what if we don't?

For loops



```
words = ['cat', 'window', 'defenestrate']
for w in words:
        print w, len(w)
```

For loops

For loops are traditionally used when you have a piece of code which you want to repeat *n* number of times.

Python's for statement iterates over the items of any sequence (a list or a string)

The range() Function is used to iterate over a sequence of numbers

```
>>> range(10)
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

>>> range(5, 10)
 [5, 6, 7, 8, 9]
>>> range(0, 10, 3)
 [0, 3, 6, 9]
 >>> range(-10, -100, -30)
[-10, -40, -70]
```

For loops

```
>>> a = ['Mary', 'had', 'a', 'little', 'lamb']
 >>> for i in range(len(a)):
                print i, a[i]
```

break and continue

Break breaks out of a loop.
Continue jumps to the beginning of the loop

For loops and files

```
for line in open("file"):
                print line
```

```
input_file = open('mytext.txt', 'r')
count_lines = 0
for line in input_file:
      print line
```