Condition and Looping

Statements

Statements are the instructions given to the computer to perform any kind of action.
There are three types of statements

1. Empty Statement
2. Simple Statement(Single Statement)
3. Compound Statement

Empty Statement

The simplest statement is the empty statement i.e. a statement which does nothing. In Python empty statement is pass statement. It take the following form:

```
pass
```

A pass statement is useful in those instance where the syntax of the language require the presence of a statement but where the logic of the program does not. We will see it in loop.

Simple Statement

Any executable statement is a simple statement in Python. for e.g.

```
    name = intput("Enter any number :")

    print("Hello")
```

Compound Statement

A compound statement represents group of statement execute as unit. The compound statements of Python are written in a specific pattern as show below:

```
<compound statement header>:
    <intended body containing multiple simple and/or
compound statements>
```

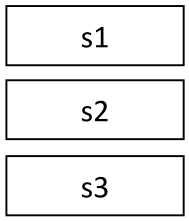
That is a compound statement has:

- a header line : which begins with a keyword and ends with a colon.
- a body : consisting of one or more statement, each intended inside the header line.

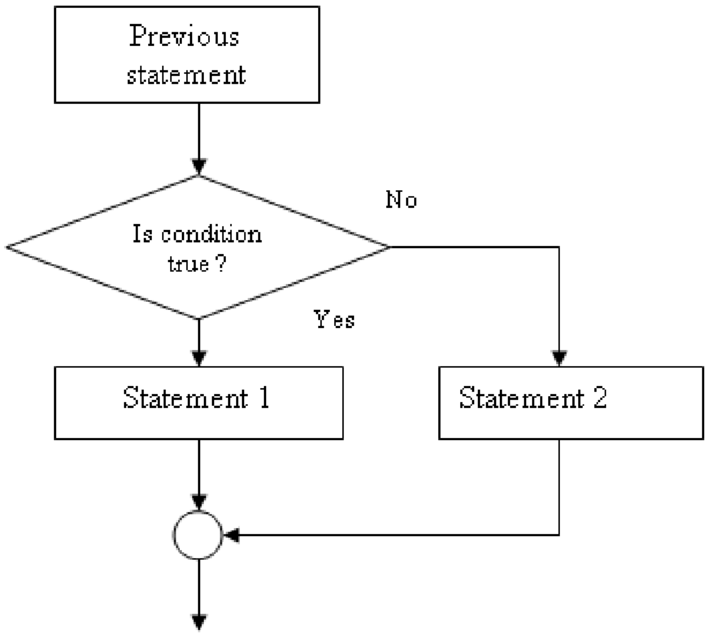
Statement Flow Control

In a program, statements may be executed sequentially, selectively or iteratively

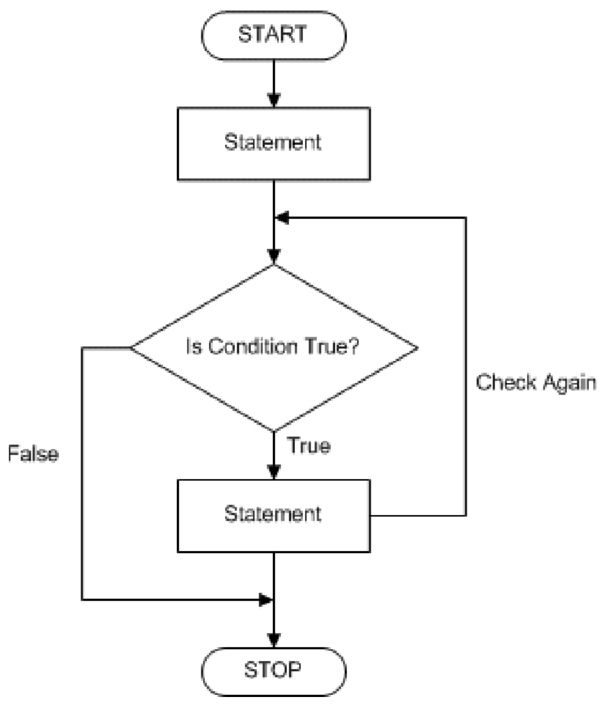
Sequence
The sequence construct means the statements are being executed sequentially. This represents the default flow of statement.



Selection
The selection construct means the execution of statement(s) depending upon a condition-test. If a condition evaluates to True, a course-of-action is followed otherwise another course of action is followed.



Iteration
The iteration construct mean repetition of a set-of-statements depending upon a condition-test. Till the time a condition is True, a set of statements are repeated again and again. As soon as the condition become False, the repetition stop. The iteration construct is also called looping construct.



if statement support selection and for and while supports iteration.

The if Statement of Python

The simplest form of if statement tests a condition and if the condition evaluates to True, it carries out some instructions and does nothing in case condition evaluates to False.

Syntax

```
    if <condition expression>:
        s1
        [s2]
```

for e.g.

```
    if ch== ‘ ’:
        space += 1
        chars += 1
```

The if-else statement

This form of if statement test a condition and if the condition evaluates to True, it carries out statement intended below if and in case condition evaluates to False, it carries out statement intended below else.

Syntax
```
     if <conditional expression>:
          s1
          [s2]
     else:
          s3
          [s4]
```
Practical Assignment:
WAP that takes a number and check whether the given number is odd or even.
WAP to display a menu for calculating area of circle or perimeter of a circle.

The if – elif statement

Sometimes, you need to check a condition in case the test-condition of if evaluates to False. That is, you want to check a condition when control reaches else, i.e. condition test in the form else if.
Syntax
```
     if <condition expression>:
          s1
          [s2]
     elif <condition expression>:
          s3
          [s4]
```

and
```
     if <condition expression>:
          s1
          [s2]
     elif <condition expression>:
          s3
          [s4]
     else :
          s5
          [s6]
```
We can use it in the situation like this
if runs are more than 100 then it is a century otherwise if it is more than 50 then it is fifty or if it is less than 50 then it is normal score

To serve conditions as above i.e. in else if form we will use if-elif and if-elif-else statement.

```
if runs >= 100:
    print("Batsman scored a century")
elif runs >= 50:
    print("Batsman scored a fifty")
else:
    print("Batsman has neither scored a century  nor fifty")
```

Practical Assignment:
WAP that reads two numbers and an arithmetic operator and displays the computed result.

The nested if statement

Sometime above discussed forms of if are not enough. You may need to test additional conditions. For such situations, Python also supports nested-if form of it.

A nested if is an if that has another if in its if's body or in elif's body or in its else's body.
Syntax

Form1 (if inside if's body)

```
if <conditional expression>:
    if<conditional expression>:
        s1
    else:
        s2
elif <condition expression>:
    s3
else:
    s4
```

Form 2(if inside the elif's body)

```
if <conditional expression>:
    s3
elif <condition expression>:
    if<conditional expression>:
        s1
else:
    s2
else:
    s4
```

Form3 (if inside else's body)

```
if <conditional expression>:
    s4
elif <condition expression>:
    s3
else:
    if<conditional expression>:
        s1
else:
    s2
```

Form4 (if inside ifs as well as else's or elif's body i.e. multiple ifs inside)

```
if <conditional expression>:
    if<conditional expression>:
        s1
else:
    s2
elif <condition expression>:
    if<conditional expression>:
        s3
else:
    s4
else:
    if<conditional expression>:
        s5
else:
    s6
```

Practical Assignment:

WAP to calculate and print roots of quadratic equation : $ax^2 + bx + c=0(a\neq0)$.

WAP to print whether a given character is an uppercase or a lowercase character or a digits or any other character.

Storing conditions

Sometimes the conditions being used in code are complex and repetitive. In such cases to make your program more readable, you can use named conditions i.e., you can store conditions in a name and then use that named conditional in the if statement.
For example
if a deposit is less than Rs 2000 and for 2 or more years, the interest rate is 5 percent.

```
eligible_for_5_percent  =  deposit < 2000 and time >= 2

if eligible_for_5_percent:
    rate = 0.05
```

The range() function

The range() function of Python generates a list which is a special sequence type. A sequence type in Python is a succession of values bound together by a single name. Some Python sequence types are strings, lists, tuples etc.

How it is work

The common use of range() is in the form given below:
`range(<lower limit>, <upper limit>)` # both limit should be integers
The function in the form range(l, u) will produce a list having values starting from `l` , `l+1, l+2, l+3,` ………`u-1`. Here lower limit is included in the list but upper limit is not included.
therefore range(0, 5) will produce `[0, 1, 2, 3, 4]`
`range(5, 0)` will return empty list [] as no number falls in the range beginning with 5 and ending at 0.
You can use following form of range() function:

```
range(<lower limit>, <upper limit>, <step value>) # all
values should be integers
range(l, u, s)
```

will produce `l, l + s, l + 2s, l + 3s + ….. u-1`

```
range(0, 10, 2)
```

will produce `[0, 2, 4, 6, 8]`

```
range(5, 0, -1)
```

will produce `[5, 4, 3, 2, 1]`

Operator in and not in

To check whether a value is contained inside a list you can use in operator

`3 in [1, 2, 3,  4]` will return True

`5 in [1, 2, 3, 4]` will return False

`5 not in [1, 2, 3, 4]` return True

Looping statements

Python provides two kind of loops

i) Counting loops
ii) Conditional loops

Counting loops : The loops that repeat a certain number of times; Python's for loop is a counting loop.

Conditional loops : the loops that repeat until a certain things happens i.e. they keep repeating as long as some condition is true; Python's while loop is conditional loop.

The for loop

The for loop of Python is designed to process the items of any sequence, such as a list or string, one by one.

Syntax

```
for <variable> in <sequence>:
    statement_to_repeat
for a in [1, 4, 5]:
    print(a)
```

A for loop in Python is processed as:

i) The loop-variable is assigned the first value in the sequence.
ii) All the statements in the body of for loop are executed with assigned value of loop variable.
iii) Once step2 is over, the loop-variable is assigned the next value in the sequence and the loop-body is executed with the new value of loop-variable.
iv) This continues until all values in the sequences are processed.

Therefore output of above loop will be

1

```
4
5
```
We can specify range in the for loop with range() function, list, string or tuple.

```
for val in range(3, 18):
    print (val)
```
The output will be

```
3
4
5
.
.
17
```

```
for ch in 'calm':
    print(ch)
```

```
c
a
l
m
```

Practical Assignment

WAP to print table of an input number.

WAP to print sum of natural numbers between 1 to 7. Print the sum progressively, i.e. after adding each natural number, print sum so far.

The while loop

A while loop is a conditional loop that will repeat the instructions within itself as long as conditional remain True.

Syntax

```
    while <logicalexpression>:
        loop-body
 for e.g.
a = 5
while a > 0:
    print("Hello", a)
    a = a - 3
print( "Loop Over")
```

This condition is tested, if it is True, the loop-body is executed. After the loop-body's execution, the condition is tested again, loop-body is executed as long as condition is True. The loop ends when the condition evaluates to False

Jump Statement – break and continue

Python offers two jump statements to be used within loops to jump out of loop-iteration. These are break and continue statements.

The break statement

The break statement enables a program to skip over part of the code. A break statement terminates the vary loop it lies within. Execution resume at the statement immediately following the body of the terminated statement.

```
while <test condition>:
      s1
      if <condition>:
           break
            s2
            s3
     s4

for <var>  in  <sequence>:
      s1
      if <condition>:
           break
            s2
            s3
      s4
#Program to demonstrate the concept of break
a = b = c =0
for i in range(0, 5):
    a = int(input("Enter any number"))
    b = int(input("Enter any number"))
    if b == 0:
        print("Divide by zero error !!! ")
        break
    else:
        c = a /b
        print("Q = ", c)

print("loop over")
```

The continue statement

The continue statement is another jump statement like the break statement as both the statement skip over a part of the code. But the continue statement is somewhat different from break. Instead of forcing termination, the continue statement forces the next iteration of the loop to take place, skipping any code in between.

```
while <test condition>:
      s1
      if <condition>:
           continue
            s2
            s3
     s4
```

```
for <var> in <sequence>:
      s1
      if <condition>:
           continue
            s2
            s3
     s4
```

```
#Program to demonstrate the concept of continue statement
a = b = c =0
for i in range(0, 5):
    a = int(input("Enter any number"))
    b = int(input("Enter any number"))
    if b == 0:
        print("Divide by zero is not define, Try another
number !!! ")
        continue
    c = a /b
    print("Q = ", c)

print("loop over")
```

Loop else statement

The else clause of a Python loop executes when the loop terminates normally, i.e. when test-condition results into false for a while loop or for loop has executed for the last value in the sequence; not when the break statement terminates the loop.

Before we proceed, have a look at the syntax of Python loops along with else clause. Notice that the else clause of a loop appears at the same indentation as that of loop keyword keyword while or for.

```
for  <variable> in <sequence>:
           s1
           s2
           .
else:
             s3

while  <test condition>:
           s1
           s2
           .
else:
             s3
```

for e.g. consider the following loop

```
for a in range(1, 4):
    if a % 2 == 0:
         break
    print(a)
else:
    print("Ending  loop  after  printing  all  elements  of
sequence")
```

Now the output is

Element is 1

```
# Program to check whether the number is prime or not
n = int(input("Enter any number : "))
f = 0
'''for i in range(2, n):
    if n % i ==0:
        f += 1
        break
if f==0:
    print("Given Number is prime")
elif f > 0:
    print("Given Number is not prime")'''

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

Nested Loops

A loop may contain another loop in its body. This form of a loop is called nested loop. But in a nested loop, the inner loop must terminate before the outer loop. The following is an example of a nested loop:

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

The output of above code will be

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