All the IPython Notebooks in this lecture series are available at https://github.com/rajathkumarmp/Python-Lectures

Control Flow Statements

If

if some_condition:

    algorithm

In [3]:
```
x = 12
if x >10:
    print("Hello")
```

```
Hello
```

If-else

if some_condition:

    algorithm

else:

    algorithm

In [2]:
```
x = 12
if x > 10:
    print "hello"
else:
    print "world"
```

```
hello
```

if-elif

if some_condition:

    algorithm

elif some_condition:

    algorithm

else:

```
    algorithm
```

In [3]:
```
x = 10
y = 12
if x > y:
    print "x>y"
elif x < y:
    print "x<y"
else:
    print "x=y"
```

```
x<y
```

if statement inside a if statement or if-elif or if-else are called as nested if statements.

In [4]:
```
x = 10
y = 12
if x > y:
    print "x>y"
elif x < y:
    print "x<y"
    if x==10:
        print "x=10"
    else:
        print "invalid"
else:
    print "x=y"
```

```
x<y
x=10
```

Loops

For

for variable in something:

```
    algorithm
```

In [5]:
```
for i in range(5):
    print i
```

```
0
1
2
3
4
```

In the above example, i iterates over the 0,1,2,3,4. Every time it takes each value and executes the algorithm inside the loop. It is also possible to iterate over a nested list illustrated below.

In [6]:
```
list_of_lists = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
for list1 in list_of_lists:
        print list1
```

```
[1, 2, 3]
[4, 5, 6]
[7, 8, 9]
```

A use case of a nested for loop in this case would be,

In [7]:
```
list_of_lists = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
for list1 in list_of_lists:
    for x in list1:
        print x
```

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

While

while some_condition:

```
algorithm
```

In [8]:
```
i = 1
while i < 3:
    print(i ** 2)
    i = i+1
print('Bye')
```

```
1
4
Bye
```

Break

As the name says. It is used to break out of a loop when a condition becomes true when executing the loop.

In [9]:
```
for i in range(100):
    print i
    if i>=7:
        break
```

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

Continue

This continues the rest of the loop. Sometimes when a condition is satisfied there are chances of the loop getting terminated. This can be avoided using continue statement.

In [10]:
```
for i in range(10):
    if i>4:
        print "The end."
        continue
    elif i<7:
        print i
```

```
0
1
2
3
4
The end.
The end.
The end.
The end.
The end.
```

List Comprehensions

Python makes it simple to generate a required list with a single line of code using list comprehensions. For example If i need to generate multiples of say 27 I write the code using for loop as,

In [11]:
```
res = []
for i in range(1,11):
    x = 27*i
    res.append(x)
print res
```

```
[27, 54, 81, 108, 135, 162, 189, 216, 243, 270]
```

Since you are generating another list altogether and that is what is required, List comprehensions is a more efficient way to solve this problem.

In [12]:
```
[27*x for x in range(1,11)]
```

```
Out[12]: [27, 54, 81, 108, 135, 162, 189, 216, 243, 270]
```

That's it!. Only remember to enclose it in square brackets

Understanding the code, The first bit of the code is always the algorithm and then leave a space and then write the necessary loop. But you might be wondering can nested loops be extended to list comprehensions? Yes you can.

```
In [13]: [27*x for x in range(1,20) if x<=10]
```

```
Out[13]: [27, 54, 81, 108, 135, 162, 189, 216, 243, 270]
```

Let me add one more loop to make you understand better,

```
In [14]: [27*z for i in range(50) if i==27 for z in range(1,11)]
```

```
Out[14]: [27, 54, 81, 108, 135, 162, 189, 216, 243, 270]
```