Control Flow

Control Flow

The **control flow** is *the order in which individual statements, instructions or function calls of a program are executed or evaluated*.

- First of all, the code is executed **sequentially** (the first instruction is executed before the second instruction and so on).
- The control flow is generally regulated by **conditional statements** (`if`, `elif`, `else`), **loops** (`for`, `while`, `else`)...

If statement

If is a conditional statement, and it **is used to check whether a given condition is true.** Its syntax is:

```
if condition1:
    block1
elif condition2:
    block2
…
else:
    blockN
```

elif and else are optional blocks

```
a = -4
if a > 0 :
    print("Positive number")
else:
    print("Negative number")
```

```
Negative number
```

```
a = -4
if a > 0 :
    print("Positive number")
elif a == 0 :
    print("Zero")
else:
    print("Negative number")
```

```
Negative number
```

```
a = -4
if a > 0 :
    print("Positive number")
```

It is a **conditional statement**: the first condition is checked, if it is satisfied, the first block is executed and the following blocks are ignored. If a condition is not satisfied, the elif conditions are checked in order; if none is satisfied, the else block is executed.

Comparison operators

Symbol	Name	Usage
==	Equal	x == y
!=	Not equal	x != y
>	Greater than	x > y
<	Less than	x < y
>=	Greater than or equal to	x >= y
<=	Less than or equal to	x <= y

Logical operators

Symbol	Operation	Usage
and	Returns True if both statements are true	x < 5 and x < 10
or	Returns True if one of the statements is true	x < 5 or x < 4
not	Reverse the result, returns False if the result is true	not(x < 5 and x < 10)

Identity operators

Symbol	Operation	Usage
is	Returns true if both variables are the same object	x is y
is not	Returns true if both variables are not the same object	x is not y

Membership operators

Symbol	Operation	Usage
in	Returns True if a sequence with the specified value is present in the object	x in y
not in	Returns True if a sequence with the specified value is not present in the object	x not in y

Those operations return **boolean values** (True or False) that determine whether a given condition is fulfilled.

If statement with operators

```
variable="ciao"
if type(variable) != str:
    print("This is NOT a string")
elif "x" in variable or "X" in variable:
    print("This is a string containing the letter X")
elif  len(variable)==0:
  print("This is an empty string")
else:
  print("This is a string")
```

```
This is a string
```

Note: A scope is delimited by indentation

For statement

The for statement defines an iterative loop that repeats a given instruction for a defined number of times. It has the syntax:

```
for variable in sequence:
    block
```

Python can loop on many different variables: strings, lists, tuples…

```
>> my_string = "dog"
>> for s in my_string:
>>   print(s)
   d
   o
   g
```

For statement

In order to iterate on integer values, in python you can use the function `range([start=0,] stop[, step=1])`.

Range returns a list of integers between start and stop, with a given step.

```
>> for n in range(3):
>>   print(n)
   0
   1
   2
```

```
>> for n in range(5, 190, 74):
>>   print(n)
   5
   79
   153
```

While statement

The while statement defines an iterative loop that repeats a given instruction as long as a given condition is **True**. It has the syntax:

```
while condition:
    block
```

Note: while/for/if statements can be nested multiple times.

```
>> i = 0
>> while i<3:
>>   i+=1
>>   if (i%2) == 0 :
>>     print(i)
  2
```

Continue – Break statements

A (for/while) loop can be interrupted in specific circumstances.

The **break** statement blocks the execution of a loop. Remember that in a nested loop, the **break** statement only interrupts the innermost loop in which it is located.

The **continue** statement forces the loop to jump to the next iteration. Again, if there is a nested loop, **continue** only acts in the innermost loop in which it is found.

```
for variable1 in iterable:
  statement1
  if condition1:
    continue
```

```
for variable1 in iterable:
  statement1
  for variable2  in iterable:
    if condition1:
      continue
    elif condition2:
      break
  statement2
```

Pass

The pass statement does not do anything. It can be used in any part of your code.

It is useful when there is the need of a statement but nothing should be done.

```
>> li = "abcd"
>> for c in li:
>>   if(c =='a'):
>>     pass
>>   else:
>>     print(c)
  b
  c
  d
```