Scientific Programming in Python
Notes
22 September 2009

Comparisons and conditions

Python allows us to compare values and evaluate truth statements. There are two truth values, `True` and `False`. Comparison operators are used to check whether a value or variable satisfies a condition. The comparison operators are

- equality (`==`);
- inequality (`!=`);
- less than (`<`);
- greater than (`>`);
- less than or equal to (`<=`);
- greater than or equal to (`>=`).

For numbers these operators behave in the standard mathematical manner.

```
>>> 3+4 == 7
True
>>> 200 <= 5
False
>>> 0 < 67 < 100
True
```

We can also combine or modify comparisons using the Boolean[1] operators `and`, `or` and `not`.

- `x and y` returns `True` when both `x` and `y` are `True`.
- `x or y` returns `True` when at least one of `x` and `y` are `True`.
- `not x` inverts the truth value of `x`.

x	y	x and y	x or y	not x
False	False	False	False	True
False	True	False	True	
True	False	False	True	False
True	True	True	True	

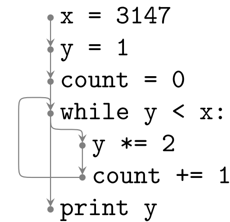
The following statements are all `True`.

```
(True and False) == False
(22//3 > 4) or (5 == 3)
not (x and y) == (not x or not y)
```

The last statement above is true for any Boolean values of `x` and `y` and is known as one of *De Morgan's laws.*

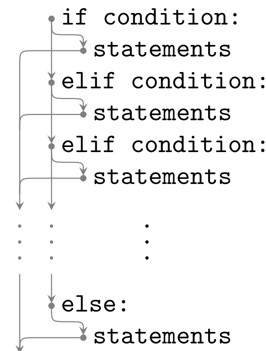
[1] A value that can be either `True` or `False` is often referred to as a *Boolean* value after George Boole, a nineteenth century mathematician and philosopher.

Flow control



Very simple programs follow a linear path. The first line of code is executed, then the second, then the third, and so on until the final line has been reached and the program ends. In more interesting programs the flow is non-linear. Some of the statements are executed multiple times, while some statements might never be executed. Here we will look at two types of programming structures that control the flow of the program—conditional statements and loops. The figure to the right shows an example of a program with a loop. The arrows indicate the possible ways in which the point of execution can change.

The `if` conditional



The general form of the `if` statement is shown to the right. The first condition is evaluated and the first statements are executed if and only if this condition is `True`. If it is `False` instead, the second condition is evaluated. The second statements are executed if and only if this condition is `True`, and so on. If *all* of the conditions are `False` the statements after the `else` keyword are executed. The `elif` and `else` parts of the construction are optional.

The `if` construction allows us to execute code once or not at all depending on whether a condition is `True` or `False`. Here is a simple example.

```
if x < y:
    print x, "is less than", y
```

If the condition `x < y` is `True` the `print` statement will be executed, otherwise it will be skipped and the first line after the `if` block will be executed. What will the following code display?

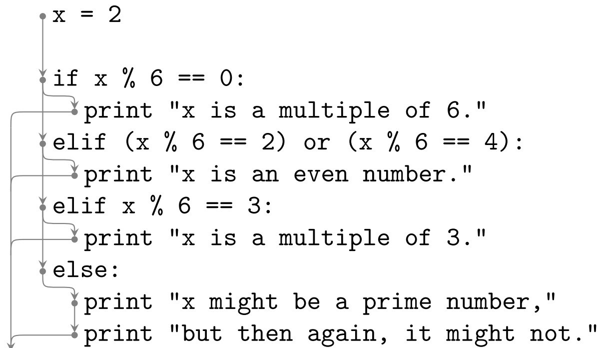
```
x = 3
if x < 10:
    x *= 2
    x += 1
print x
```

This code results in the value 7 being printed since the condition is `True` and the statements following the `if` are executed.

We can extend an `if` statement with an `else` part. The `else` keyword follows directly after the `if` statement and the code in the `else` statement is executed if the result from the condition is `False`. The following code multiplies a variable by 2 if it is odd and divides it by 2 if it is even.

```
if x % 2 == 1:
    x = x * 2
else:
    x = x // 2
```

The `if` statement can be extended even further using `elif`, which is short for *else if*.

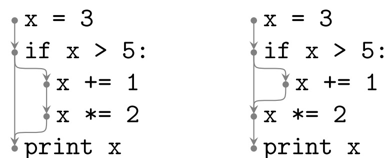


In this example the first condition, `x % 6 == 0`, is `False` so Python moves on to test the second condition. The second condition is `True` and the output displayed is

```
x is an even number.
```

In general you can have any number of `elif` statements and the `else` statement at the end is optional—it gets executed if all of the conditions are `False`.

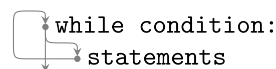
Finally note that you can have multiple statements inside an `if`, `elif` or `else` block. It is required that all of the statements in such a block are indented. Python uses indentation to determine which statements are inside and which ones are after a block. For example, the code on the left will display the result 3, but the code on the right will display the result 6.



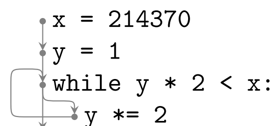
Since the `x *= 2` line is not indented in the code on the right, it will be executed regardless of whether the `if` condition is `True` or `False`. *Indentation matters!*

The `while` loop

The general form of a `while` loop is shown on the right. The condition is evaluated and the statements executed if it is `True`. If the statements were executed, the condition is evaluated *again* and the statements executed a second time if it is still `True`. The condition will be checked and the statements executed until it becomes `False`. In the end the statements might be executed 0, 1, 2, or more times.



The following code finds the largest power of 2 less than `x`. Note that y starts off at the value 1 and is then doubled as long as it is less than x. As soon as the next power of 2 (i.e. `y * 2`) is going to be greater than or equal to `x`, the `while` loop terminates.



You can also use a block of code rather than a single statement inside the `while` loop. What follows is a more complex example, which calculates the greatest common divisor of two integers using Euclid's algorithm. The greatest common divisor of two integers is the largest positive integer that leaves no remainder when divided into each one of the two inputs. You can experiment with the code by modifying the values of the inputs, `a` and `b`.

```
a = 44551
b = 455791

print "This program calculates the greatest common divisor of a and b."
print "  a =", a
print "  b =", b

# Make sure that b is smaller than a
if a < b:
  # Swap a and b
  temp = a
  a = b
  b = temp

while b > 0:
  a = a % b

  # Swap a and b
  temp = a
  a = b
  b = temp

print "  gcd(a,b) =", a
```