Building Blocks of Python Programs
Comments

We want people to be able to read and understand our programs. The # symbol introduces a comment, which is a note for human readers of the code. Comments are ignored by computers. Anything to the right of a # symbol is part of the comment and ignored.
You should get in the habit of putting a comment at the top of every program saying at least
  a) Your name
  b) What the program does

Here is a nice format for this
    # gradebook.py
    # This simulates a digital gradebook
    # author: Bob Geitz
    # Last modified January 29, 2015
Variables

A variable is a name that represents something in your program.

Variables start with a letter and consist of letters, digits, and underscores. No spaces, periods, hyphens, etc.

Here are some good variable names

averageScore
letterCount
letter_count
Most programming languages require variables to be *declared*, which requires saying what kind of data the variable can hold. There are no variable declarations in Python. You create a variable by giving it a value, as in

\[ x = 5 \]
Assignment statements give values to variables. We use = for this. We can say

\[
\begin{align*}
x &= 5 \\
x &= 6
\end{align*}
\]

The first use of a variable creates it, so the line \( x=5 \) creates variable \( x \) and puts the value 5 into it. The line \( x=6 \) changes the value stored in \( x \) to 6.

Don't confuse \( = \) (for assignments) with \( == \) (for comparisons)
Here are 4 simple types of data:

- **Integers**: 2, -3, 0
- **Floats**: 3.14, -6.2, 5.0
- **Strings**: "Bob", "Oberlin College", ""
- **Booleans**: True, False
Integer data

• Read with `eval(input( <prompt> ))` as in
  
x = eval( input( "Enter a number: " ))

• Arithmetic operations +, *, -, /, //, %, **
• / is for floating point division: 7/2 is 3.5
• // is for integer division: 7/2 is 3
• ** is for exponentiation: 3**4 is 81
• % is the modulus (or remainder) operation
  7 % 5 is 2
Note that % (the modulus or remainder operator) is more useful than you might think:

- I usually pronounced $a\%b$ as "a mod b"
  Some people say "a remainder b"
- $b$ divides evenly into $a$ if $a\%b$ is 0
- $x$ is even if $x\%2$ is 0; $x$ is odd if $x\%2$ is 1
- days $d_1$ and $d_2$ of a given month fall on the same day of the week if $d_1\%7$ is the same as $d_2\%7$. 
The Arithmetic Rule for operators +, -, *
If a and b are both integers, then a op b is an int.

If either a or b or both are floats, then a op b is a float.
There isn't a lot to say about floats except that they are there. Internally the integer 3 is stored in a completely different way than the float 3.0. This makes comparing floats and integers for equality problematic.

You can convert an int x to a float with

```python
float(x)
```

as in

```python
float(3)
```

which gives you 3.0.
Strings

• Strings are delimited with either single quotes: 'bob'
or double quotes: "bob"
• read with input( )
• if blah is a string that represents a valid Python expression, then eval(blah) gets the value of that expression:
  • eval("4") is 4.
  • eval("2+3") is 5.
• The + operator between 2 strings *concatenates* or pushes the strings together.
  "Marvin " + "Krislov" is "Marvin Krislov"
• The comparison operators <, <=, ==, >=, >, != compare strings in dictionary order, but all of the capital letters come before all of the lower-case ones.
You can use indexes to get at the individual characters (letters) of a string. We always start indexing at 0.

Suppose s is the string "abcd". Then s[0] is "a", s[1] is "b", and so forth. The number of characters in string s is len(s). So the valid indexes of string s are any integers between 0 and len(s)-1.
s[a: b] is the portion of string s starting at index a, going up to but not including index b. So if s is "Bob the Great", s[4:7] is "the". Similarly s[a:] is all of s starting with index a, and s[:b] is the portion of s up to but not including index b.

You can even use negative indexes: s[-1] is the last character of string s. But I find it easy to get confused with negative indexes so I tend to avoid them.
Finally, if s is a string then s.upper() is s with its lower-case letters converted to upper-case. "23 skidoo".upper() is "23 SKIDOO".

There is a similar .lower() method that converts upper-case letters to lower-case.
**Booleans** (named after George Boole, a British logician)

There are two Boolean values: **True** and **False**. Note the capitalization: `true` has no meaning in Python, **True** does. If you are feeling demented, `true=False` is a valid expression in Python.

You can connect two Boolean expression with **and**, **or**, **not**.
Here is an expression that says variable $x$ has a value between 1 and 10:

$$\text{if } (x \geq 1) \text{ and } (x \leq 10):$$
$$\text{ blah blah blah}$$

It is possible in Python to write this as

$$1 \leq x \leq 10$$

but I have seen so many people do that incorrectly that I much prefer to write compound expressions with explicit operators like \texttt{and, or}. 