Intro to Computing—CSCI-1310

Email David

Fri, Jan 16, 2015

Variables, operators, data types

Programming is about solving problems. Some problems are small, and can be represented with an algebraic formula, while other problems are extremely complicated and complex, such as building a computer model of a city.

But, even with big problems, the individual building blocks of a solution are generally simple, and it is from those simple blocks that we can build complex computer code. We start with some big problem to solve and decompose that problem in to a series of smaller problems, each of which we can plug into a computer using computer code.

In order for a problem to be reproduced in code, we need a representation of the problem that captures enough of reality in order to produce a reasonable solution. For example, say you have 5 apples and 4 oranges, and you want to write an equation to calculate the number of pieces of fruit you have. You need a variable for apples, a variable for oranges, and a variable for fruit. These variables represent an abstraction of the original problem.

Today, we're starting programming by looking at the basic building blocks of code: variables. We use variables to represent something in our code and the type of the variable depends upon what we're representing.

First, let's look at this weeks homework where you printed Hello World to the terminal.

The print statement is a command built in to Python. When you run the .py file, the Python interpreter knows to display whatever is contained between the ().

```
print("Hello world")

#Can print other strings by changing the words in the print statement
print("Welcome to CSCI1300")
print("Here is yet another message")
```

When you create a variable, a location is setup in memory with the variable name as an identifier for that location and the value at that location. For example, you can create a variable called "a" with a value of 5 using

```
a = 5
```

The syntax there is left-hand-side = right-hand-side

After that command executes, there is a location in memory that is identified as a and that location holds a value of 5. In this case, the identifier is recognized until the program quits. The name of the variable here is arbitrary, there's nothing special about using the letter a.

You can access "a" and print it's value by doing

```
print(a)
```

We can do this again, creating another variable called b and give it a value of 6, and then print out the value of b.

```
b = 6
print(b)
```

Variables have types. The a and b we've created both hold whole numbers, called integers, so we say that a and b are integers, that's their type.

There are other types we're going to discuss today. The types you will use most frequently in this class are integers, floats, and strings. There are also booleans, lists, and dictionaries in Python.

Another variable type is called a string. Strings contain one or more letters and the value of the variable is the string. For example, we can create a variable called "aStr" that has a location in memory and holds the value "this is a string".

```
aStr = "this is a string"
# To check the value of aStr, do
print(aStr)
```

There are also numbers with decimal values. These are called floats in Python, which is short for floating point. An example of a float is

```
c = 4.5
# To check the value of c, use the print statement
```

```
print(c)
```

The value of a string doesn't necessary need to be a letter. We can also have strings that are numbers. For example, a string called x with the value "2.2" and a string called y with the value "4.4".

```
x = "2.2"
y = "4.4"
print(x)
print(y)
```

Another Python variable is called a boolean, which can only have two values, True or False. These True of False values reduce down to a 1 or 0, on or off. For example, to create a boolean variable called myBool that is set to True, you do

```
myBool = True
print(myBool)
```

Notice the color coding of the keyword True. It's blue, indicating that Pythong recognizes it. If you change the capitalization so that True is true, the highlighting goes away and you'll get an error when you run the code. Python is case-sensitive, True is different than true.

```
myBool = False
```

Python also has a variable type called a list, which can store multiple items together as one variable, and each of those items can actually have different types

For example, here we have a variable called myList where the first item is an integer, second item is a float, and third item is a string.

```
myList = [4, 6.5, 'a']
print(myList)
```

myList has three items in and to access each individual item, we use the index of the item. Indexing starts at 0, so the first item is myList[0], the second is myList[1], and the third is myList[2]

```
print(myList[0])
print(myList[1])
print(myList[2])
```

Later in the semester, we'll talk about two other data types, one is a dictionary and the other is a set

There are times when you will want to take a variable that is one type and make it behave like it's a different type. We call this casting. For example, if you have a string and you want to be a number, you would cast it as a float, like so

```
x = float(x)
print(x)
```

The float command is built in to Python and to use it, anything between the parentheses is called an argument to the float function. float will produce a new value that is a type float, using the argument.

Notice here that both the argument to the float function and the return value of the float function are both called x. It's the same x. You may be wondering how you can have two things, one on each side of an = sign that are not actually equal. The left-hand side does not equal the right hand side.

Recall what `x` is, it's a memory location. What's happening here is we're saying take the value at the `x` location, convert it to a float, and return it back to the `x` location. You'll see this over and over again in programming, where variables in programming and the `=` sign in programming is not the same as the = sign from math class.

There is also a conversion from float to integer. Here we use the Python keyword int to convert the existing value of `x` to an integer `x`. Depending on the value in `x`, we may actually change its value. Since integers don't have decimal values, if `x` has a decimal, it will be truncated, and you will be left with the whole number portion only.

```
x = int(x)
print(x)
```

The type of the variable matters here because of how the variable is handled when you do operations with it. If you add two numbers together, you get the sum of those numbers, but if you add two strings together, you get a new string that is both strings appended together.

For example, let's create float variables and two string variables with numbers and look at what happens when they are added. This is an operation on two variables

```
a = 5.1
b = 4.5
c = a + b
print(c)

a = "5.1"
b = "4.5"
c = a + b
print(c)
```

The + operator can also be used on strings that do not have numbers. For example, adding two strings creates a new string

```
my_str = "hello world "
my_str2 = "how are you today"
new_str=my_str + my_str2
print(new_str)
```

In the examples with + so far, both variables have been of the same type. If you try to add two variables with types where the + operator has different meaning, you will get an error. For example, adding an integer and a string will result in Python being confused, is the + meant to be the addition of two numbers, or the appending of two strings

For example, we can create two new variables with different types and add them, and then watch the error when we print the result.

```
n = 5
s = "the number is "
# uncomment the next line to see the error
# sn = s + n
# print(sn)
```

The previous print statement produced a type error. Python sees the s first and then the + and parses the command as appending two strings, and expects that the variable to the right of the + will be a string. When the n is reached and it's not a string, an error is generated.

To solve this problem, you can cast one of the variables to match the type of the other variable. The variable that you choose depends of what you're trying to achieve and the values of the variables. The options here are to cast s as a float or integer, or cast n as a string. Since s isn't a numeric value, we can't cast it as such. However, we can cast n as a string, and then 5 becomes "5".

```
sn = s + str(n)
print(sn)
```

There are also standard operations on numbers that you are familiar with. An example of dividing two numbers:

```
a = 5
b = 6
c = a / b
# c is a floating point number, it has a decimal component
print(c)

# multiplication
z = a*b
print(z)

# exponentiation, notice the double *
# x2^2
z = a**2
print(z)

# x2^y2
z = a**b
print(z)

# you can even combine operators in one equation
z2 = a + (b * z) / a
print(z2)
```

Exercise

Given a circle with a radius of 10 units. Calculate the area and circumference of the circle.

> The formula for area is `pi*r**2`
> The formula for circumference is `2*pi*r`

Answer

```
radius = 10
# for a variable name, I'm using something meaningful, area and `circ`
area = 3.14 * radius**2
circ = 2 * 3.14 * radius
print("the area is:", area)
print("the circumference is:", circ)
```