1 Getting used to Python

We assume you know how to program in some language, but are new to Python. We'll use Java as an informal running comparative example.

Here are what we think are the most important differences between Python and what you already know about programming.

1.1 Shell

Python is designed to be easy for a user to interact with. It comes with an interactive mode called a *listener* or *shell*. The shell gives a prompt (usually something like `>>>`) and waits for you to type in a Python expression or program. Then it will evaluate the expression you typed in and print out the value of the result. So, for example, an interaction with the Python shell might look like this:

```
>>> 5 + 5
10
>>> x = 6
>>> x
6
>>> x + x
12
>>> y = 'hi'
>>> y + y
'hihi'
>>>
```

So, you can use Python as a fancy calculator. And as you define your own procedures in Python, you can use the shell to test them or use them to compute useful results.

1.2 Indentation and line breaks

Every programming language has to have some method for indicating grouping. Here's how you write an if-then-else structure in Java:

```
    if (s == 1){
        s = s + 1;
        a = a - 10;
    } else {
        s = s + 10;
        a = a + 10;
    }
```

The braces specify what statements are executed in the `if` case. It is considered good style to indent your code to agree with the brace structure, but it isn't required. In addition, the semi-colons are used to indicate the end of a statement, independent of where the line breaks in the file are. So, the following code fragment has the same meaning as the previous one.

```
    if (s == 1){
s = s
+ 1;     a = a - 10;
    } else {
                s = s + 10;
    a = a + 10;
    }
```

In Python, on the other hand, there are no braces for grouping or semicolons for termination. Indentation indicates grouping and line breaks indicate statement termination. So, in Python, we'd write the previous example as

```
if s == 1:
    s = s + 1
    a = a - 10
else:
    s = s + 10
    a = a + 10
```

There is no way to put more than one statement on a single line. If you have a statement that's too long for a line, you can signal it with a backslash:

```
aReallyLongVariableNameThatMakesMyLinesLong = \
        aReallyLongVariableNameThatMakesMyLinesLong + 1
```

It's easy for Java programmers to get confused about colons and semi-colons in Python. Here's the deal: (1) Python doesn't use semi-colons; (2) Colons are used to start an indented block, so they appear on the first line of a procedure definition, when starting a `while` or `for` loop, and after the condition in an `if`, `elif`, or `else`.

Is one method better than the other? No. It's entirely a matter of taste. The Python method is pretty unusual. But if you're going to use Python, you need to remember about indentation and line breaks being significant.

1.3 Types and declarations

Java programs are what is known as *statically and strongly typed*. That means that the types of all the variables must be known at the time that the program is written. This means that variables have to be declared to have a particular type before they're used. It also means that the variables can't be used in a way that is inconsistent with their type. So, for instance, you'd declare `x` to be an integer by saying

```
int x;
x = 6 * 7;
```

But you'd get into trouble if you left out the declaration, or did

```
int x;
x = "thing";
```

because a *type checker* is run on your program to make sure that you don't try to use a variable in a way that is inconsistent with its declaration.

In Python, however, things are a lot more flexible. There are no variable declarations, and the same variable can be used at different points in your program to hold data objects of different types. So, this is fine, in Python:

```
if x == 1:
    x = 89.3
else:
    x = "thing"
```

The advantage of having type declarations and compile-time type checking, as in Java, is that a compiler can generate an executable version of your program that runs very quickly, because it can be sure what kind of data is stored in each variable, and doesn't have to check it at runtime. An additional advantage is that many programming

mistakes can be caught at compile time, rather than waiting until the program is being run. Java would complain even before your program started to run that it couldn't evaluate

```
3 + "hi"
```

Python wouldn't complain until it was running the program and got to that point.

The advantage of the Python approach is that programs are shorter and cleaner looking, and possibly easier to write. The flexibility is often useful: In Python, it's easy to make a list or array with objects of different types stored in it. In Java, it can be done, but it's trickier. The disadvantage of the Python approach is that programs tend to be slower. Also, the rigor of compile-time type checking may reduce bugs, especially in large programs.

1.4 Modules

As you start to write bigger programs, you'll want to keep the procedure definitions in multiple files, grouped together according to what they do. So, for example, I might package a set of utility functions together into a single file, called `utility.py`. This file is called a `module` in Python.

Now, if I want to use those procedures in another file, or from the the Python shell, I'll need to say

```
import utility
```

Now, if I have a procedure that file called `foo`, I can use it in this program with the name `utility.foo`. You can read more about modules in the Python documentation.

1.5 Interaction and Debugging

We encourage you to adopt an interactive style of programming and debugging. Use the Python shell a lot. Write small pieces of code and test them. It's much easier to test the individual pieces as you go, rather than to spend hours writing a big program, and then find it doesn't work, and have to sift through all your code, trying to find the bugs.

But, if you find yourself in the (inevitable) position of having a big program with a bug in it, don't despair. Debugging a program doesn't require brilliance or creativity or much in the way of insight. What it requires is persistence and a systematic approach.

First of all, have a test case (a set of inputs to the procedure you're trying to debug) and know what the answer is supposed to be. To test a program, you might start with some special cases: what if the argument is 0 or the empty list? Those cases might be easier to sort through first (and are also cases that can be easy to get wrong). Then try more general cases.

Now, if your program gets your test case wrong, what should you do? Resist the temptation to start changing your program around, just to see if that will fix the problem. Don't change any code until you know what's wrong with what you're doing now, and therefore know that the change you make is going to correct the problem.

Ultimately, for debugging big programs, it's most useful to use a software development environment with a serious debugger. But these tools can sometimes have a steep learning curve, so in this class we'll learn to debug systematically using "print" statements.

Start putting print statements at all the interesting spots in your program. Print out intermediate results. Know what the answers are supposed to be, for your test case. Run your test case, and find the first place that something goes wrong. You've narrowed in on the problem. Study that part of the code and see if you can see what's wrong. If not, add some more print statements, and run it again. **Don't try to be smart....be systematic and indefatigable!**

You should learn enough of Python to be comfortable writing basic programs, and to be able to efficiently look up details of the language that you don't know or have forgotten.

MIT OpenCourseWare
http://ocw.mit.edu

6.01SC Introduction to Electrical Engineering and Computer Science

Spring 2011

For information about citing these materials or our Terms of Use, visit: http://ocw.mit.edu/terms.