From Java to Python
For RHIT Students

A guide by Andy Schmitz

Why I wrote this guide…

This guide is designed to help all the people who were affected by the CSSE language change. I was caught in the middle along with many of my friends and colleagues. Instead of sitting down passively and struggling through the upper level courses, I decided to learn Python on my own. I learned it during (rare) slow homework weeks. I soon found that Python is very similar to Java. The commands are similar, the syntax is similar, and once you understand how Python "works", it's easy to translate from Java to Python. So that's why I wrote this guide. It's designed to give people a look at how Python "works". Then (if I did my job), you should be over that first "what am I looking at?" learning curve. Confidence goes up, stress goes down, grades go up, and hours spent banging your head on your desk in frustration also goes down…

Anyway, if something's wrong, incorrect, or incomplete, I take no responsibility for any confusion, mayhem, or rioting in the streets that may result. This is my official "stupidity disclaimer". But you have any corrections, ideas, enhancements, or questions, feel free to contact me at schmitar@rose-hulman.edu. - Andy Schmitz

First of all, Python is a high-level language, just like Java. And hey guess what, syntactically it's pretty similar. It's still got all the basic commands you're used to, for-loops, if-else-then statements, lists, etc… So learning to program in Python is a pretty easy step. There are obviously a few differences, but, in my opinion, they're all done in a way that makes Python a much better, much easier language. But who cares, lets actually learn something.

Top Secret Installation Information
Assuming you don't know anything about Python and don't have it downloaded, go here: http://www.python.org/ftp/python/2.5.1/python-2.5.1.msi. (Last checked: Oct 18, 2007)

Python is a scripting language, so it's typically run through a program called IDLE, it's console program. So, because it has a console, you can mess around with bits of code and debug programs in small bits. For simple examples, just start using it as a calculator. But I figure you'll soon want to do something a little bit cooler, so:

The biggest visual difference between Python and Java is the absence of those damn brackets. Instead, Python uses indentation through tabs. Just like what you already do to make Java look pretty. Alright, let's try our first code in Python (using IDLE):

Hello World. Well Hello World. It’s so nice to have you back where you belong

```
>>> for i in range(10):
     print i
```

-yields-

```
0
1
…
8
9
```

Ok, so the output prints 0-9. Woo Hoo! Don’t you just get a warm feeling inside? If you do, you’re getting a little too excited. Go grab some water and simmer down. Anyway, explaining the code, the colon marks the start of a multi-line statement. For-loops and if-statements both use them. “print i” is obvious. It’s a simpler “System.out.println(i)” But what in the world is “`range(10)`”? Well, first of all, for-loops in Python don’t take numbers. They take a list instead. So you could have used “`for i in [0,1,2,3,…,8,9]`” and it would have done the same thing. The range() command just automatically creates that list for you.

Amazing Side Note

Actually, you don’t even need a list of numbers. Try this code:

```
>>> x = [“Andy”, “Josh”, “Packard”, “Rich”]
>>> for i in x:
     print i + “ is amazing!”
```

The range() command takes 1-3 numbers. The required number is the number the list ends on (non inclusive). So `range(4)` returns `[0,1,2,3]`. If you want to start at something other than zero, that’s what the other number is (inclusive). So `range(3,6)` gives `[3,4,5]`. The third number is the increment. So `range(0,10,3)` returns `[0,3,6,9]`.

In the last side note, I introduced variables. The difference between Python and Java is that Python assumes you’re a decent programmer and know what you’re doing. You don’t need to specify what kind of variable x is. You can even change the variable type mid-program.

What the world needs now; is code sweet code

```
>>>def sumTheList(x):
     # BTW: This is a comment.
     # It starts with the pound character "#"
     # Try messing up and input an integer instead of a list.
     # Python will yell at you.
     y = 0
     for i in range(len(x)):
          y += x[i] # the += also exists in Python
     print y
>>>sumTheList([0,1,2])
-yields-  #By the way:  This command already exists.
3         #  sum(x).  It hates integers too.
```

Alright, so I introduced some other topics too. The "def" command defines a function. This can be used any number of times. If you want to create a one-time anonymous function, you can use the "lambda" command, stolen from Lisp (another fun, but even more messed up language). If you don't know why you'd ever use that, don't worry, you can erase that from your memory. Another command I used was `range(len(x))`. That's the exact same thing as saying "i<x.length" inside the for-loop statement. But, in this example, (only because it's designed for numbers) I could have said "for i in x: y+=i". This is more difficult to understand, at least for me, but it's very useful and makes for neater, more readable code. Anyway, we've examined the for-loop in lots of juicy detail. Let's move on.

For-loops are nice, but to really do work, you need if-statements. The structural idea is exactly the same (in all high-level languages). The only difference is the Java "else if" command is written as "elif" in Python. The colon/tab idea is still the same. Another feature is that the "if" doesn't need parentheses. Just one more small detail the designers included that reduces keystrokes. Ok, the last detail that's not really important, but nice to know, is that because you don't have to define data types, "true" and "false" is designated by "1" and "0". It's a bunch of tiny small junk that's easy to remember, but still needs to be said. Anyway, let's see the code…

Sweet Code-oline [bum]x3 Good times never seem so good!

```
>>>x = ["Andy", "Josh", "Packard", "Rich", "Steve McWho"]
>>>def myFunction(x):
	for i in x:
		if i == "Andy":
			print i + " is the best guy in the world!"
		elif i == "Josh":
			print i + " is the genus one"
		elif i == "Packard":
			print i + " is the comic relief"
		elif i == "Rich":
			print i + " is the 'talk about anything' one"
		else:
			print i + " isn't one of my best friends. "
```

Ok, so the only new and unexpected thing here is the single quote. In Python, both "" and '' mean the same thing. Then why does the language have both? So you can spell "can't" without using escape characters(\n, \t, etc…). `'"Bob lies" said Jason'` returns a phrase with quotes. So 95% of the time, when you're not using imbedded quotes, you can use whatever is more comfortable. And the other 5% can be done without escape characters.

Alright, so we've used lists in every single example so far, and I haven't even formally mentioned them. That foolishness stops here! We're going to discuss lists. Alright! Power to the (wo)man! First, there's the traditional list that everyone knows and loves, the `[]` as you've probably figured out. Just like an ArrayList in Java, you can nest lists, have combinations of lists and elements, stick whatever you want in them, etc. And one thing that's really nice, Python keeps the object type when you take something out. Bye-bye casting variables! So basically, those square bracket brothers will be your friends. They'll be used 99% of the time. But they've also got some cousins from out-of-state, the tuple. The what? The tuple. Ignore the funny name; just call it a fixed list. It's the equivalent of using the "FINAL" keyword to make a list. A tuple is created by using `()`. Just like Java, they're used for fixed purposes such as setting a static *(non-changing, not the Java interpretation of the word)*, reference type of list. Pretty simple; I'm going to skip the examples. The third, and most powerful (and complicated), list is called a "dictionary". They use the squiggly brackets `{}`. They are similar to hash-tables. You build them like such:

Can't touch this… Stop! Dictionary-time (As if MC Hammer ever read books)

```
>>> x={ 0:"a", #Ohhh look… multiple lines
        1:"b",
        3:"d",
        2:"c",           #Just demonstrating that you can
        "Frank": ""      #  use different kinds of keys
>>> x
```

-yields-
The ordered version of x (2 & 3 are switched). Strings are also ordered alphabetically.

```
>>>x["Frank"]
```

-yields- # The default string "operator" is the single quote. So unless you use a double
`''` # quote inside it, it will, by default, return a single quote "operator".

The first element is the key, the second is the return statement. Basically, if you don't already understand this, don't bother learning it. At least not with one, ridiculously simple example. It's for power-hungry programmers trying to do fancy-pants things.

Alright, the last topic is how to create your own programs and run them. After-all, what's the point of making a program if you have to retype it into IDLE every time you want to use it. First, create a new text document (I use notepad) and type in your program.

*** There's antimony, arsenic, aluminum, fibonaccium; And hydrogen and oxygen***

```
def fib(n):
    if n == 0:
        return 0
    elif n == 1:
        return 1
    else:
        return fib(n-1) + fib(n-2)
```

Then save that as "FibonacciProgram.py". Make sure you select "All Files" under the "Save as Type" drop bar, otherwise it will save it as "FibonacciProgram.py.txt". Also, save it wherever you installed Python at (*subfolders are ok*). If you want to use it elsewhere, I'll explain how to do that in a second. Alright, a lot of explaining, but that's Window's fault, not Python's. Now type "`import FibonacciProgram`" into IDLE. Then "`FibonacciProgram.fib(<some *small* value>)`". Amazing! Your first real program. Now, if you wanted to get around typing the "FibinacciProgram" prefix every single time, type "`from FibonacciProgram import *`". Now you can type "`fib(n)`" and get something. The first example was accessing fib() through a module. Just like using the String class to access the "String.valueOf(number)" method.

Well that's dandy and all, but let's get even more efficient. Let's create an executable file. This will allow you to use a program without using IDLE. Once again, create your program with the .py extension. Now this is a program I wrote and actually use on a normal basis. So guess what, it's a little longer than everything else. Try ctrl-c and ctrl-v…

Why does he always reference old, outdated songs? He must be seriously lame

```
import time
#This is the bell schedule
sch = [[[8,5]  ,""],
       [[9,0]  ,""],
       [[9,55] ,""],
       [[10,50],"Artificial Int."],
       [[11,45],""],
       [[12,40],"Comp Arc I"],
       [[13,35],"Comp Arc I"],
       [[14,30],""],
       [[15,25],"Diff EQ"],
       [[16,20],"School's Out"]]


def get():
    #Gets the current time in minutes
    a = convert(time.localtime()[3:5])

    #Checks the day of the week.
    if (time.localtime()[6] > 4) or (time.localtime()[6] == 3):
            print "It's the weekend, silly!  No class!"
            return

    #Determine which hour it is right now
    for i in range(len(sch)-1):
        if(a > sch[i][0]) and (a < sch[i+1][0]):
            print "It is " + str(i+1) + "th hour, " + sch[i][1]
            print "You have " + str(sch[i+1][0]-a-5)  + " minutes
before " + sch[i+1][1]
            return

    #else, if it's not school-time
    print "School's not in session right now!"
    return


#Converts [h,m] between mmmm
#If it errors trying to access the array, it must be in mmmm form.
def convert(x):
    try:
        return x[0]*60 + x[1]
    except:
        return [x/60, x%60]
```

(continued onto the next page)

```
########################
##### Run on Start #####
########################
#This converts the easy to read hh:mm format of the schedule into mmmm
for i in range(len(sch)):
        sch[i][0] = convert(sch[i][0])

get()
raw_input("Press ENTER to continue...")
```

Alright, this is the final bit of Python code, so I'm not going to explain it, I'm going to let you explore it for yourself. If you have any questions, feel free to e-mail me at schmitar@rose-hulman.edu. Anyway, if everything works properly, just double-click on the .py file. The Windows Command Prompt will come up and do it's magic. One thing I will explain, is the "raw_input" line. It's used to read input at the cmd prompt level. It's used here as a pause. If I didn't have it there, windows would open the cmd prompt, run the program, display the answer, and exit the window, all in the blink of an eye. So it allows you to actually read the output.

The last bit of code I'm giving you *(code? I thought there was no more code! I'm drowning already! Arrg! Well haha, I said the last Python code. This is DOS code! [karate hand motions])* is to allow you to save your .py files somewhere else becides in the Python directory (*like My Documents maybe?*). First create your folder somewhere. My example is "My Documents\School\Python Projects". Then go to start > run > type "cmd". Now's where it gets tricky. It may differ slightly depending on what you've done to your computer. Fortunately, for 99% of you, this should work. Type "`set pythonpath = ..\..\Documents and Settings\<your username>\My Documents\<wherever you put your folder. Ex.'School\Python Projects'>`". Make sure there are no typos. Now, you can import your own programs into IDLE without saving them in the "C:\Program Files\Python 2.5.1" folder.

Alright, that's about it. I've (hopefully) taught you the basics of Python. If you pay attention in class, you can learn the rest of it without too much stress. If you're looking for more information about Python, visit www.python.org. It's got lots of examples and tutorials. Especially ones that are better than mine. So have fun programming. Or if you're not a programmer, and you're being forced to take this class (I originally wrote this for a math major), well uh… good luck anyway!!!

-Andy Schmitz