Background

Scripting?

Programming languages are classified as either *interpreted* or *compiled*.

A compiled language is transformed into a *binary* format that runs only on a specific type or types of computer. The advantage to a compiled program is that it runs really fast. The computer's CPU simply needs to read small instructions and execute them one after another. The speed comes at a price: as mentioned, only certain computers can run the compiled program.

An alternative to a compiled language like C or Ada is is an interpreted language. `BASIC` is an example of an interpreted language. More commonly used today are *scripting languages*: Perl, Ruby, Python, shell scripting, DOS batch programs, and so forth.

(To make things complicated, some languages like Java and C#, are both compiled and interpreted.)

With the speed of computers today, simple programs will run incredibly fast whether interpreted or compiled. It isn't until intensive number crunching and complicated loops are used that a compiled language has a huge advantage over an interpreted.

Why?

Why do we care about any of this? We're gonna do some scripting! The first 12 weeks (at least) of lecture can be easily supplemented by doing some real programming in lab. Few if any students have any experience programming so we want it to be as simple as possible. When it comes to programming languages, the easiest ones are often those that include a large *library* of powerful *functions* to do a lot of work for you. The one that we're going to play with is Python.

Hello, world!

To make scripts easy to run, we need to set the first in the file to something specific that tells the computer what program should run the script. This line is called a *shebang*, *hashbang*, *hashpling*, or *pound bang*.

For our Python code, the shebang will be:

```
#!/usr/bin/env python
```

Once we have this line, we can simply start writing our program. That's all the setup that's required.

One of the basic things that many programs need to do is print out text. In Python, the command to print text to the screen is simply `print`. Let's print the standard *Hello, World!* text to the screen:

```
print ''Hello, World!''
```

Notice that the text we want to print is in quote marks. Without them, Python will try to find *variables* with those names. More on variables later.

Our program is now two lines long:

```
#!/usr/bin/env python
print ''Hello, World!''
```

So how do we run it? Simply type the name of the file in an SSH window:

```
cse rnelson/p> helloworld.py
Hello, World!
cse rnelson/p>
```

Did your output not quite match? If the program is not set to *+x* (e**x**ecutable). The *chmod* program will let us set the script to be executable:

```
cse rnelson/p> helloworld.py
helloworld.py: Permission denied.
cse rnelson/p> chmod +x helloworld.py
cse rnelson/p> helloworld.py
Hello, World!
cse rnelson/p>
```

Now if you try running it again (by typing the filename and hitting enter), you should see your message.

Yawn

That was boring. What fun is programming if all we can do is print out some text to the screen? Let's get some data from the user. The easiest way to get text is the `raw_input()` function: `variable = raw_input(''prompt text'')`.

```
#!/usr/bin/env python
text = raw_input(''Enter some text: '')
print ''You typed: '', text
```

That program will prompt the user to enter text and then echo it back to them. Fancy!

Numbers

The information stored in `text` is a *string*. If what it contains is a number, you still can't do any math on it as-is. You need to cast, convert, parse (fairly interchangeable terms) the string into a number.

To convert a string into an integer, Python includes a `int()` function. The following program will take input from the user and store it into an integer; the integer can then be used in mathematical operations (as shown)

```
#!/usr/bin/env python
numstring = raw_input(''Enter a number: '')
x = int(numstring)
minusx = -x
print ''0 - x = '', minusx
```

Let's try running it:

```
cse rnelson/p> ./math.py
Enter a number: 5
0 - x =  -5
cse rnelson/p>
```

Awesome! Can we break it? Give it a word instead of a number:

```
cse rnelson/p> ./math.py
Enter a number: hi
Traceback (most recent call last):
  File "./math.py", line 3, in <module>
    x = int(numstring)
ValueError: invalid literal for int() with base 10: 'hi'
cse rnelson/p>
```

Turns out *hi* isn't a valid number in base 10.

Week 3 Lab: Basic Python

Directions

You will be writing two programs in Python this week.

Program 1: Hello World

Write a program that does nothing but prints out the text *Bonjour, le monde!*. Make sure you don't have any extra spaces before, after, or within the text. *Bonjour*, comma, space, *le*, space, *monde*, exclamation point.

Save this file as `hello.py`.

Program 2: Calculator

Write a program that gets two numbers from the user and performs the four basic math operations (addition, subtraction, multiplication, and division) on them. The numbers may be integer values or real values; to accommodate both, parse them as real numbers using the `float()` function.

1. Addition: $1 + 2$
2. Subtraction: $1 - 2$
3. Multiplication: $1 * 2$
4. Division: $1/2$

The following is an example run of a possible solution:

```
cse rnelson/p> ./calc.py
X: 1
Y: 2
3.0
-1.0
2.0
0.5
cse rnelson/p>
```

Notice my prompts; `X`, colon, space and `Y`, colon, space. The user typed in 1 and 2 for the two values and then the values of the four operations (in the order given above) are displayed, each on their own line.

Save this file as `calc.py`.

Handing in

I have not had a handin account setup for the lab yet. When I do, you will submit code through a web-based submission system used by the department. Assuming I have the account by that point, these programs (as well as the HTML stuff from last week) will be due at 3:20 PM next Wednesday (4 Feb. 2009).

Grading

Area	Points
Hello works:	5
Hello identical output:	3
Hello correct filename:	2
Calc works:	5
Calc identical output:	3
Calc correct filename:	2
Total:	20