CSci 1133, Spring 2015
Lab Exercise 1

Introduction to UNIX and Python

In this first lab, you will make sure your account is functional and explore a number of computational resources that will be needed during the course of the upcoming semester. You will also begin to write your first Python code!

First, register your account

If you haven't already done so, you need to register your CSE Labs account. To register your account, use a web browser to visit http://www.cselabs.umn.edu/ and then click on the "Account Information" link.

Logging in

To log in from a machine in one of the CSE computer labs (such as KHKH 2-170) pick an available Linux machine and type your *username* and *password* in the login box.

Part 1: Brief UNIX Introduction (adapted from C. Swanson, "Unix Tutorial")

The UNIX operating system dates to the early 1970's and remains one of the most efficient, robust and reliable pieces of software in use today. The "operating system" refers to the collection of programs residing on your computer that provide the interface to the actual hardware of the machine. The operating system is responsible for maintaining and organizing data on external devices, loading and running application programs, etc. You are probably familiar with graphically based operating systems such as Microsoft Windows, Apple Mac OS and others (actually, Mac OS uses the UNIX operating system for its "core" functions). Our lab computers employ the UNIX operating system, so you will need to learn a few UNIX essentials. As usual, you are encouraged to explore and learn more on your own! There is an abundant wealth of UNIX information and tutorials available on the Internet.

The UNIX Terminal

In the early days of UNIX, communication with the operating system was performed via an electro-mechanical contraption known as a "teletypewriter terminal" or "terminal" for short. It had a keyboard for generating "input" to the operating system and a paper printer for producing "output" from the operating system.

The concept of a "terminal" still applies today, but now we *emulate* the teletype device using a graphical display window. When we run the "terminal" application, our keystrokes are input directly to the operating system and the subsequent output is displayed in the window rather than being printed on a roll of paper.

Unlike many other computer systems you may have used in the past, you will be interacting with UNIX mainly using typed commands. The commands are entered into a terminal window along with required and optional *arguments*. *Arguments* are additional information that tell the computer what to do with the command. For example, if you want to rename a file, you need to provide the name of the file to be renamed. This way of communicating with UNIX is referred to as the *Command Line Interface*, or CLI.

To run programs and generally interact with the UNIX operating system in "command line" mode, you must first start the terminal application by clicking on the terminal icon (generally located on the upper left side of your display -- it looks like a little computer monitor). Start the terminal now (seek help from your TAs if you need it).

The Command Line Interface
The operating system will prompt you for a command, using a short text string like:

```
username@machine:~$
```

where *username* is your UNIX user name and *machine* is the machine you are currently logged in to. The '$' symbol lets you know that the system is ready for another command. Commands can be edited while they are still being typed using the left and right arrow keys or the backspace key. UNIX commands must be typed *exactly*, and are case-sensitive.

Most UNIX commands are shortened names that describe what they do. For example, `cp` stands for *copy* and `mv` stands for *move*. If you ever forget what a particular command does or how it is used, you can use the `man` command to learn more. `man` stands for *manual* page and will tell you everything you need to know (and more) about a particular UNIX command. Simply type: `man` *command-name* [ENTER] to display the manual page, and press [SPACE] to scroll through the pages of information. You can even use `man` to learn about `man` itself! Try it:

`man man` [ENTER]

Files and Directories
If you've used a modern computer system, you are probably familiar with the concepts of files and folders. A file is a collection of data that has a specific name associated with it. A folder is a special file that contains zero or more files or other folders. In UNIX, folders are called *directories*, but they are essentially the same thing. Files and directories allow users to store and organize their data in an easy-to-use fashion.

File and directory names are case-sensitive, and special care should be taken when naming them. Avoid using spaces or special characters except for the underscore, dash, and period. Certain special characters are forbidden in filenames, but others are allowed and can cause problems. Also, avoid giving files the same name as UNIX commands.

Navigating the File System
All of your personal files and directories are contained within your *home* directory. When you first start a new terminal, your *home* directory will always be your (active) *working* directory. The working directory is simply the directory you are currently viewing, or working in. You can use the *print working directory* command, `pwd` at any time to show you what your current working directory is. Type the following:

`pwd` [ENTER]

You should see something like:

`/home/`*username*

This is the complete *path* from the first (root) directory to your current working directory. Each directory name in the path is separated by forward slash characters.

To list all of the files and directories in your working directory, type the *list* command, `ls`:

```
    ls [ENTER]
```

All files will be listed; directories are displayed with an ending forward slash. Directories can be created and destroyed using `mkdir` (*make directory*) and `rmdir` (*remove directory*). Try making a new directory:

```
    mkdir hello [ENTER]
```

You can change to a different working directory with the *change directory* command, `cd`. To change to your newly created `hello` directory:

```
    cd hello [ENTER]
```

Now we will introduce some special names for directories. The double-dot (`..`) stands for the *parent* directory, which is the directory that contains the current working directory. We can use this to "move up" one level. Try

```
    cd ..  [ENTER]
    pwd    [ENTER]
```

Also, the tilde (~) is shorthand for your *home* directory. We could have also returned to the home directory (from anywhere) by typing:

```
    cd ~ [ENTER]
```

Now that we have returned to the home directory, we can remove our hello directory. Type

```
    rmdir hello [ENTER]
    ls  [ENTER]
```

Note that `rmdir` will only remove empty directories (that is, directories that contain no files or other directories).

Creating Python script files
Python can be used interactively (one command at a time) or in "script" mode. In "script" mode, Python reads its commands from a file. Let's create a simple Python script that will calculate the amount of a radioactive substance that remains after a specific time given an initial amount of a substance, its half-life and the elapsed time. First, create a special directory for this script. Type:

```
    mkdir myscripts [ENTER]
    cd myscripts  [ENTER]
```

There are many ways to create text files in UNIX, but for writing small program scripts it is convenient to use a text editor. A text editor is a relatively simple program (application) that works like a stripped-down word processor. It allows you to enter/modify text and save the text to a file.

There are a number of text editors available on CSELabs machines. We'll use the GEANY text editor for this lab assignment. Start the GEANY editor by typing:

```
geany decay.py [ENTER]
```

Once GEANY is ready, you can enter source code. Type in (or copy and paste, if you can figure out how) the following Python script:

```
init = eval(input("Initial Amount: "))
halflife = eval(input("Half-life: "))
time = eval(input("Elapsed-time: "))
residual = init*0.5**(time/halflife)
print("Residual amount remaining = ",residual)
```

Make sure you type in the script *exactly* as shown!

Simply typing text in the editor program does not actually create the file; you must explicitly *save* the text to the file. To save your changes, select "File" and "Save" in the pull-down menu. We'll use this script later, but for now it will provide an opportunity to learn some additional UNIX commands. Note that computer languages are very precise and Python has no tolerance for errors! Every symbol in this program has significance and must be entered *exactly* as it appears.

Close (quit) the GEANY application.

Your `decay.py` script is now saved as a file in the `myscripts` directory. Suppose we had intended to put the file in a directory called `labOne`. Making this change will give us an opportunity to practice some UNIX commands. If you aren't already there, change to your home directory:

```
cd ~ [ENTER]
```

Now make a new directory named `labOne`:

```
mkdir labOne [ENTER]
```

and change your working directory to the `myscripts` directory:

```
cd myscripts [ENTER]
```

We'll use the *copy* command, `cp`, to copy files from one directory to another. The copy command takes the form

`cp` *source-file destination-file*

where *source-file* is the name of the file to be copied, and *destination-file* is where we want to put it. Use the list command to list all the files in the `myscripts` directory:

```
ls [ENTER]
```

Now, copy the file using the `cp` command

```
cp decay.py ../labOne/ [ENTER]
```

The `../labOne/` tells the cp command to find the directory called `labOne` in the parent directory, and copy the files there. Now use the *list* command to list all of the files in the `labOne` directory, to make sure they were copied successfully:

```
ls ../labOne/ [ENTER]
```

Once you have verified that you copied the files correctly, you can delete the files in the myscripts directory using the *remove* command. Type:

```
rm decay.py [ENTER]
```

Be careful! Unlike other computer systems, there is no "undelete" or "Recycle Bin" in UNIX. Once you delete a file, it's gone for good in most cases. Now return to your home directory:

```
cd .. [ENTER] (or cd ~ [ENTER])
```

While copying myscript files into the labOne directory was good practice, we could have saved ourselves a lot of work by simply renaming the myscripts directory! Try it yourself. Use the *move* command to rename a directory:

```
mv myscripts hello [ENTER]
ls [ENTER]
```

Change it back using:

```
mv hello myscripts [ENTER]
ls [ENTER]
```

`mv` can be used to rename either files or directories.

Lastly, we will discuss two more convenient functions provided by the command line. These are *command history* and *tab completion*.

Command History
You may have noticed already that pushing the up arrow in a terminal window brings up previous commands. You can use this to your advantage to recall any command you have typed in that terminal window. This is especially useful if you make a mistake in a long command. You can simply press the up arrow until you arrive at that command, and correct the error using the left and right arrow keys. Try it:

```
makdir hello [ENTER]
```

Press the up arrow once to recall the command. Then press the left arrow until the cursor is on the k. Then press backspace to remove the erroneous a. Finally, press [ENTER] to retry the command. The cursor does not need to be at the end of the command; the terminal will take the whole line as the command regardless of cursor position.

Tab Completion

Often, you will have long filenames that are annoying to type. Tab completion will save you some typing. You can simply type the first few letters of a file or directory name, press the Tab key, and the terminal will try to complete the name for you. Try this:

```
cd ~ [ENTER]
cd h [TAB]
```

The terminal should "fill in" the rest of hello. But what happens when multiple files start with the same characters? Try this:

```
cd ~ [ENTER]
mkdir hydrogen [ENTER]
cd h [TAB]
```

The terminal should display the names of all files and directories that start with h. In this case, hello and hydrogen both qualify. Type

```
y  [TAB]
```

and the terminal should fill in the rest of hydrogen.

You've learned to use several important UNIX commands for manipulating files. If you are still unsure of what you are doing, discuss it with one of your Lab TAs before continuing.

Part 2: Running Python

The best way to learn Python is to just start playing with it… First we must start the Python3 interpreter.

There are two versions of the Python language and, unfortunately, they are not compatible with one another. We'll be using Python version 3 throughout our course, so you will need to be especially careful that you are running the correct version.

Python is a computer language "interpreter"; a command-line program that reads a line of input and responds with some output. To "invoke" the Python interpreter simply type:

```
python3 [ENTER]
```

Be sure to type `Python3` and not `Python` or you will get the wrong version! You should see something like:

```
Python 3.4.1 (default, Oct 19 2011, 12:14:35)
Type "help", "copyright", "credits" or "license" for more
 information
>>>
```

This new prompt, >>> , indicates that you are now providing Python commands to the interpreter and not to the Unix command-line interface.

Type something like:

```
4+5*2 [ENTER]
```

and Python will respond with the value of the expression:

```
14
```

Pretty cool! Now try some arithmetic expressions on your own. Note that +, - and / all have the usual meanings (addition, subtraction, division), but multiplication is represented with the * symbol.

We can save the result of an expression evaluation for use later on by simply giving it a name and using the assignment operator (=). Try this:

```
>>> x = 23.5 * 46 [ENTER]
```

Note that the "value" of the result is not shown… it has been stored in the computer's memory with the name 'x'. If we want to use this value in another expression, we just use its name:

```
>>> x**2*3.14159 [ENTER]
3671139.55199
```

Note that the two asterisks (**) is not a mistake. This is how you represent exponentiation in Python. You may find a more frequent use for the value 3.14159. So you can just name it as well:

```
>>> pi = 3.14159 [ENTER]
```

If you want to see what the value of the object named 'pi' is, you can use a built-in Python *function* to print its value to the terminal display. Type the following:

```
>>> print(pi) [ENTER]
```

Python will print out the value of the object named pi, 3.14159.

Go ahead and explore on your own! Try creating more complex expressions, storing and printing out the results.

Turtle Graphics

Python has a built-in graphics display capability that is very simple and fun to use called "turtle" graphics. To use the turtle graphics package, you must first tell the Python interpreter to load the turtle module. Type the following Python commands (exactly as shown):

```
>>> import turtle [ENTER]
>>> turtle.showturtle() [ENTER]
```

The first Python command loads the turtle graphics module, the second one causes a graphics window to be displayed with a small triangle (the 'turtle') in the middle. To draw a line, you simply *move* the turtle. Try this:

```
>>> turtle.forward(100) [ENTER]
```

Voila! You've drawn a line in the direction the turtle is facing. Its easy to change the turtle direction using a rotation method. Type in the following Python instructions:

```
>>> turtle.left(90) [ENTER]
>>> turtle.forward(100) [ENTER]
```

There are lots of fun things you can do with turtles! Let's complete the square. Type the following:

```
>>> turtle.forward(100) [ENTER]
```

Oops, forgot to *turn* the turtle first. No problem, there is a very useful turtle command that will fix our mistake. Try the following:

```
>>> turtle.undo() [ENTER]
```

So you can try things and if they don't do what you expect, no problem, just "undo" them!

Now enter commands to complete the square on your own.

Consult appendix C3 in your textbook and try out as many of the turtle commands as you have time for. Go ahead! Experiment! Its fun! You simply type "turtle", followed by a period ('.') and then the name of the turtle command (method). Be sure to use the parentheses.

Now do something creative and show one of the TAs.

When you are done experimenting and want to end your Python session, simply enter control-d (hold the "control" key down and press the 'd' key at the same time). If you need help, ask a class TA for assistance.

If you've done this successfully, you should see your Unix command-line prompt once again.

Congratulations! You have completed your first lab exercise and are now an experienced Python programmer!

Radioactive Decay
If you just can't get enough, you can take a sneak-peek at running Python scripts. Change your working directory to the `labOne` directory that contains the `decay.py` script you created earlier. To run the script, simply add the filename (`decay.py`) following the python3 command:

```
python3 decay.py [ENTER]
```

Try the following values and see how your script runs!

Initial amount = 4.
Half-Life = 140.
Elapsed Time = 420.