Lab 1: An Introduction to IDLE and Python

CSE/IT 107

NMT Computer Science

1 Introduction

The purpose of this lab is to introduce you to the Python programming environment. In this lab, you will learn how to

1. Log into Linux at the Tech Computer Center (TCC).
2. Learn to use IDLE as an interpreter.
3. Edit, compile, and run a few short Python programs.
4. Create a tarball and submit a lab to Moodle.

Throughout this semester's labs, we will be using specially formatted text to aid in your understanding. For example,

```
text in a console font within a gray box
```

is text that either appears in the terminal or is to be typed into the terminal verbatim. Even the case is important, so be sure to keep it the same! If a $ is used that indicates the terminal prompt (Linux only, see Appendix A). You do not type the $.

```
1 Text that appears in console font within a framed box is
2 sample Python code or program output.
```

Text in a `simple console font`, without a frame or background, indicates the name of a file, a function, or some action you need to perform, but not necessarily type into the terminal. Don't worry: as you become familiar with both the terminal and Python, it will become obvious which is being described!

2 Instructions

The following instructions will guide you through this lab. Because this is an introductory lab, the instructions are rather detailed and specific. However, it is important that you understand the concepts behind the actions you take—you will be repeating them throughout the semester. If you have any questions, ask the instructor, teaching assistant, or lab assistant.

2.1 Log in

To log into a TCC machine, you must use your TCC username and password. If you do not have a TCC account, you must visit the TCC Office to have one activated.

You can choose to use Linux or Windows in this class.

It should be noted that most of these instructions will work on Windows or Mac OS with Python installed, though we recommend using Linux simply to gain familiarity, as later Computer Science classes will require it.

2.2 Windows: Open IDLE

Throughout this semester, we will primarily be working with IDLE. IDLE is an integrated development environment included with every Python installation. In this lab we will cover how to use it as a calculator, as well as how to use it to write simple programs.

2.3 Linux: Intro to Linux

You can use IDLE in Linux just like you can in Windows and move on to Section 3 or you can learn the usage of the command line. Future CS classes will make extensive use of the command line.

If you wish to use the Linux command line for this class, find the introduction to Linux in Appendix A. Skip to the appendix and then come back here to follow Section 3.

It may be that the Linux instructions will take longer than the alloted lab time. You would have to work out some of the material after lab is over. (This will be the case for all future labs anyway.)

3 Python as a calculator

Forgot ever using your calculator! Python rocks as a calculator.

When you first open IDLE, you should see something like this

```
Python 3.4.1 (v3.4.1:c0e311e010fc, May 18 2014, 10:38:22) [MSC v.1600 32 bit (I
Type "copyright", "credits" or "license()" for more information.
>>>
```

Note the first line thing printed out: `Python 3.4.1`. This is the version of Python IDLE is running. Make sure you have *Python 3* or higher. If you do not, check for another program called IDLE (or IDLE 3) or, if on your own computer, install Python 3.

At the Python prompt `>>>` you can type in Python statements. For instance, to add 2 + 3

```
>>> 2 + 3
5
>>>
```

Notice how after a statement is executed, you are given a prompt to enter another statement.

The `+` is the addition operator. Other basic math operators in python include:

`-` Subtraction: subtracts right hand operand from left hand operand

```
>>> 2 - 3
-1
```

`*` Multiplication: Multiplies values on either side of the operator

```
>>> 2 * 3
6
```

`/` Division, `//` Floor division - Divides left hand operand by right hand operand

Python 3

```
>>> 2 / 3
0.6666666666666666
>>> 2 // 3
0
>>> 2 / 3.0
0.6666666666666666
>>> 2 // 3.0
0.0
>>> 2. / 3.
0.6666666666666666
>>> 2 / float(3)
0.6666666666666666
```

This is an example of a major difference between Python 2 and Python 3. If you are using Python 2 some of these calculations will give different results. Make sure you are using Python 3.

Using `/` will always include the decimal portion of the answer. If you wish to drop the decimal portion, you must use `//`. Dropping the decimal portion of the answer is called integer division (since your answer will always be an integer) or floor division (since it will always round down).

`%` Modulus: Divides left hand operand by right hand operand and returns the remainder.

```
>>> 5 % 3
2
>>> 3 % 5
3
```

`**` Exponent: Performs exponential (power) calculation on operator. `a ** b` means a raised to b.

```
>>> 10 ** 5
100000
>>> 5 ** 10
9765625
>>> 9 ** .5
3.0
>>> 5.5 ** 6.8
108259.56770464421
```

Besides fractional powers, python can handle very large numbers. Try raising 10 to the 100th power (a googol)

```
>>> 10 ** 100
10000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000L
```

The `L` at the end of the number indicates it is a long. Long integers are integers that are too big to represent with a normal integer. In general you are able to treat integers and longs as equivalent data types.

Of course these commands can be combined to form more complex expressions. They follow typical precedence rules, with `**` having higher precedence than `* / % //`, which have higher precedence than `+ -`. If operators have equal precedence they execute left to right. You can always use parenthesis to change the precedence.

```
>>> 6 * 5 % 4 - 6 ** 7 + 80 / 3 #((6 * 5) % 4) - (6 ** 7) + (80 / 3)
-279908
>>> 6 * 5 % 4
2
>>> 6 * (5 % 4)
6
>>> (6 * 5) % 4
2
```

The `#` indicates the start of a comment. Comments are ignored by the interpreter and can be used to provide reminders to yourself as to what a section of code does.

The real power of python as a calculator comes when you add variables. Using = allows you to assign a value to a variable. A variable can have any name consisting of letters, numbers, and underscores. Once a variable has been assigned, it can be used in future computations.

```
>>> x = 7
>>> y = 3 * x ** 2 + 7 * x + 6
>>> y
202
```

In general you should try to give your variables descriptive names so that it is clear what a section of code does.

```
>>> food_price = 4.50
>>> drink_price = 1.00
>>> subtotal = food_price + drink_price
>>> tax = subtotal * 0.07
>>> total = subtotal + tax
>>> total
5.885
```

4 Running Python programs

So far, you have been running Python interactively. This is a nice feature as it lets you test ideas quickly and easily. However, there are times you want to save your ideas to a text file and to run the code as a program. Your programs will be almost identical to what you enter when using the Python interpreter, though you will have to use the `print` function in order to display output.

Open IDLE's text editor from `File > New File` or with `Ctrl + N` and create a one line Python program:

```
1 print('Hello, world!')
```

`print` is a function. It will print the value of whatever is within its parentheses. In order to print plaintext (not variables), you must additionally surround it with quotes. This is called a string. It is important to note that

```
1 print("Hello, world!")
```

is equivalent to the first example.

`\n` prints a newline. This means that the text following the `\n` will start on a new line. Test this with the following example.

```
1 print('Hello, world!\nHow are you?')
```

Save your file as `hello.py`. The `py` extension indicates it is a Python file.

To run the program, press `F5`.

```
Python 3.4.1 (v3.4.1:c0e311e010fc, May 18 2014, 10:38:22) [MSC v.1600 32 bit (I
Type "copyright", "credits" or "license()" for more information.
>>>
Hello, world!
How are you?
>>>
```

You can also print out the values of variable that you may have assigned earlier in the program.

```
1 x = 5
2 y = 4
3 print(y)
4 y = y + x
5 print(y)
```

```
>>>
4
5
```

But what if you wish to print a variable's value on the same line as another string? Unfortunately, the simplest method of doing this does not work:

```
>>> x = 5
>>> print('x = ' + x)
Traceback (most recent call last):
  File "<pyshell#9>", line 1, in <module>
    print('x = ' + x)
TypeError: Can't convert 'int' object to str implicitly
```

However, we can get around this by converting `x` into a string using `str()`.

```
>>> x = 5
>>> print('x = ' + str(x))
x = 5
```

At this point it is not necessary to completely understand the differences between a string and an actual number, just remember that `x = '5'` and `x = 5` denote different things.

5 Exercises

5.1 Conversions

Download `conversions.py`. It is a simple Python program for converting temperatures from Celsius to Kelvin. We would like to modify it to also convert the temperatures to Fahrenheit. Currently, the output of the program is something like this:

```
Please input a temperature in Celsius: 10
Kelvin temperature: 283.15
```

Add code (you should not need to modify the existing lines, though you are free to) so that its output is as follows:

```
Please input a temperature in Celsius: 10
Kelvin temperature: 283.15
Fahrenheit temperature: 50.0
```

6 Submitting

You may submit your code as either a tarball (instructions below) or as a .zip file. Either one should contain all files used in the exercises for this lab. The submitted file should end with either `.zip` or `.tar.gz` depending on which you used.

For Windows, use a tool you like to create a `.zip` file. The TCC computers should have `7z` installed.

Upload your tarball or .zip file to Canvas.

6.1 Linux

Tar is used much the same way that Zip is used in Windows: it combines many files and/or directories into a single file. Gzip is used in Linux to compress a single file, so the combination of Tar and Gzip do what Zip does. However, Tar deals with Gzip for you, so you will only need to learn and understand one command for zipping and extracting.

In the terminal (ensure you are in your `lab1` directory), type the following command, replacing `firstname` and `lastname` with your first and last names:

```
tar czvf cse107_firstname_lastname_lab1.tar.gz *.py
```

This creates the file `cse107_firstname_lastname_lab1.tar.gz` in the directory. The resulting archive, which includes every python file in your `lab1` directory, is called a tarball.

To check the contents of your tarball, run the following command:

```
tar tf cse107_firstname_lastname_lab1.tar.gz *.py
```

You should see a list of your Python source code files.

Appendices

A Linux environment

A.1 Open a terminal

If you choose to use Linux throughout this semester, you will perform most actions from the command line. In order to do this, you must first open a terminal. There are many methods for opening a terminal; the following is just one.

1. Click on the `Activities` button at the top left of the screen.
2. In the text box in the upper right hand corner, type in `Terminal` and hit return.

A.2 Change your password (optional)

In order to protect the work that you do this semester, it is important that you have strong password protection of your TCC account. See Microsoft's article "Strong Passwords and Password Security" (`http://www.microsoft.com/protect/fraud/passwords/create.aspx`) for advice on creating strong passwords.

The following instructions are optional; however, they are a useful way to become familiar with command line actions.

1. In the terminal, type the following command:

```
$ passwd
```

2. Follow the prompts to change your password.

A.3 Shell

At the terminal prompt you are using what is called a shell. A shell provides a means to interact with the operating system. The shell also provides the command line for you: a means to type in commands and receive feedback from the operating system, shell, or program behind that command.

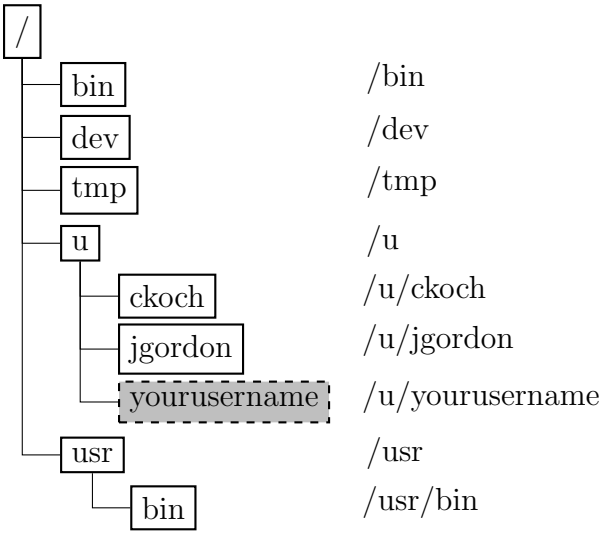
A.4 File Hierarchy

Unix has a logical file system, which means as an end user you don't care about the actual physical layout and can focus on navigating the file system.

Key: Everything begins at the root directory `/` and all other directories are children of the root directory. The file system forms a tree.

A path consists of the names of directories and is a way to navigate from the root file system to the desired directory. If you include the root directory in the path this is called an absolute path. Another way to navigate the tree is to use relative paths, which navigate the tree based on where you currently are.

For example, this is a typical file system tree close to what the TCC has with absolute file paths shown on the right:



To navigate the file system, there are a couple of useful commands:

A.4.1 pwd - print working directory

print **w**orking **d**irectory displays the directory you are in.

```
$ pwd
/home/cse
```

The directory you are currently in is called your "working" directory.

A.4.2 cd - change directory

`cd` allows you to change your working directory to another location.

Syntax:

```
$ cd <path_name>
```

```
$ cd /usr/local/bin
$ pwd
/usr/local/bin
$ cd -
```

Where are you?

```
cd -
```

Now where are you? The command `cd -` returns you to your previous working directory.

The `-` is called a command line argument or option. To view the list of options for a command, use the manual.

```
$ man cd
```

`man` is short for manual.

To move up one directory (one node) level use dot-dot:

```
$ cd ..
```

Where are you?

To move up two levels use dot-dot slash dot-dot:

```
$ cd ../..
```

To go to your home directory you can type

```
$ cd /u/<user>
```

where <user> is your user name. Alternatively, use a tilde

```
$ cd ~
```

or you can use

```
$ cd $HOME
```

where $HOME is an environment variable that stores the path to your home directory. This is more useful in shell scripting than path navigation. Use tilde instead.

Even easier than all these options is to just type `cd` without any options. This always takes you to your home directory.

```
$ cd
```

A.4.3 ls - listing the contents of a directory

`ls` allows you to list the contents of a directory: the files and directories that are located there.

```
$ cd
$ ls
<a list of file names and directories contained
with your working directory, which is currently
your home directory>
```

Adding some command options let you see other file information

```
$ ls -alFh
...
-rwxr-x---.  1 scott scott  1.0K Jan 12 14:56 down_and_out*
drwxr-xr-x.  5 scott scott  4.0K Jan 21 17:04 Downloads/
-rw-r-x---.  1 scott scott  2.2K Jan 12 11:31 .emacs
...
```

The `a` option show hidden files (aka dot-files), the `l` option produces a long listing format of file names and directories, the `F` option adds a `*` to the end of the file if is an executable and a `/` if it is a directory, and the `h` option prints out the size of the file in human readable format.

Understanding the long format option

The `-l` option of `ls` tells you quite a bit about a file. The first character of the line indicates the file type. Some file type options are:
`-` ordinary file
`d` directory
`l` symbolic link

There are other file types.

The next three groups of three letters indicate the file permissions for the *user*, *group*, and *others* (aka ugo). Common letters are `r` for the ability to read the file; `w` for the ability to write or change the contents of the file on disk; and `x` for executable or the ability to run the file. There are other file permissions.

The next item indicates how many links the file has. A directory often has several links, because each of its subdirectories includes an entry for its parent directory and that entry counts as a link to the parent.

The next two items indicate the user and the group that own the file.

The next item indicates the file size in characters (1 character = 1 byte). Use the -h option to get this in human readable format (K for kilobytes, M for megabytes, and G for gigabytes)

The next two items indicate the date and time when the file was last modified.

The last item is the file name. If you use the `-F` option, a `*` at the end of the file indicates it is an executable file and a `/` indicates it is a directory.

Finding more information

How can you see more command line options for `ls`? What is the option to sort by file size?

A.5 Other directory commands

```
mkdir
cp
mv
rm
touch
cat
less
script
```

A.6 Creating directories

Directories are very important for organizing your work, and you should get in the habit of creating a new directory for every lab this semester.

1. In the terminal, change your current working directory to your home directory by typing the following command:

   ```
   cd
   ```

2. Create a `cse107` directory by typing the following command;

   ```
   mkdir cse107
   ```

3. Change your current working directory to the `cse107` directory by typing the following command:

   ```
   cd cse107
   ```

4. Create a subdirectory for this lab in your `cse107` directory by typing the following command:

   ```
   mkdir lab1
   ```

5. Change your current working directory to the `lab1` directory by typing the following command:

   ```
   cd lab1
   ```

A.7 Using the file system

The following instructions will show you how to maneuver in the Unix file system. For more information on these commands, review the TCC publications "Summary of common Unix commands" (`http://infohost.nmt.edu/tcc/help/pubs/unixcrib/`) and "Customizing your Unix account" (`http://infohost.nmt.edu/tcc/help/pubs/dotfiles/`).

1. In the terminal, type the following command:

   ```
   $ pwd
   ```

   `pwd` stands for 'print working directory,' and the result is called the *absolute path* to this location. In this instance, the result should be something similar to

   ```
   /u/username/cse107/lab1
   ```

   where `username` is your TCC account username.

2. Type the following command:

   ```
   $ ls
   ```

`ls` stands for 'list segments' (a carry over from Multics) and it prints to the screen, *standard out*, a list of files in the current directory. At this point, you should see nothing, because you have not put any files into your `lab1` directory. If, instead, you type the following command:

```
$ ls -a
```

you should see two entries: `.` and `..`, which exist in every directory. The `.` stands for 'this directory' and the `..` stands for the 'parent directory.'

3. Try the following commands to move within your directory structures:

```
$ cd .
$ pwd
$ cd ..
$ pwd
```

4. To return to your `lab1` directory, type the following command:

```
$ cd lab1
```

5. Now type the following commands:

```
$ cd
$ pwd
```

Typing `cd` without any parameters should take you to your 'home directory.'

6. Using the commands you've just learned, return to your `lab1` directory. There is more than one way to get to the directory.

7. In your `lab1` directory, type the following command, and then find out where it takes you:

```
$ cd ../..
$ pwd
```

8. From your current directory, type the following command:

```
$ cd cse107/lab1
$ pwd
```

Notice that you can use multiple paths at once on the command line, separated by `/`. If you've used Windows for some time, you will notice that this differs from Windows path designations where the directory separator is \.

A.8 Start Python

In the shell, type `python3`. It is important that you not use `python`, but `python3`! It should look something like this:

```
$ python3
Python 3.3.2 (default, Jun 30 2014, 12:45:18)
[GCC 4.8.2 20131212 (Red Hat 4.8.2-7)] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

From this point on, you can follow instructions in Section 3 by replacing "open IDLE" or "type in IDLE" with using `python3` and `gedit` in the terminal.