CMSC 201 Spring 2016
Lab 01 – Debugging

Assignment: Lab 01 – Debugging
Due Date: During discussion, February 8th through February 11th
Value: 10 points

In Lab 0, we logged onto GL and set up folders for 201 in your home directory. We also created a simple Python program, and turned it in using the **`submit`** command. We'll be using many of these skills in this lab as well.

As we've discussed in class, testing and debugging your programs is a large part of being a successful programmer. Sometimes, you may even have to debug other people's code!

In this lab, we'll be creating two files: **`lab1.py`** will be a file you create, and **`errors.py`** will be a file you copy into your directory, before finding and fixing the errors it contains. Both files will be counted as part of the grade for Lab 1.

Part 1: Creating a complete Python program

You'll learn how to copy files into your account from an instructor's directory, and you'll write your first complete Python program. We'll also learn a few more useful emacs shortcuts.

Step 1:

The first thing you'll do is download a file that will configure the emacs editor we'll be using, to customize the way the emacs program behaves. While in your *home directory*, copy the **`.emacs`** file into your current folder by using the **`cp`** command below (**`cp`** simply stands for "copy").

```
cp /afs/umbc.edu/users/k/k/k38/pub/cs201/.emacs .
```

There are three parts to the command, and all three are important:

1. "**`cp`**" is the command, and in this case it stands for copy
2. "**`afs/umbc.edu/users/k/k/k38/pub/cs201/.emacs`**" is where the file you are copying is located
3. "**`.`**" (a single period) is where the file will be copied to (In this case, to the current folder, and with the same filename.)

The period "**`.`**" in front of the file name indicates the file is a hidden file. If you simply type ls, you won't see it listed. To double check that you successfully copied the file, you need to use the command **`ls -a`**. The **`-a`** means "all" and will show all the files in that directory, even hidden ones.

Step 2:

Next, you are going to create your first complete Python file, entirely from scratch. First, create the **`lab1`** folder using the **`mkdir`** command -- the folder needs to be inside your **`Labs`** folder as well. *(For a reminder of how to create and navigate folders, refer to the instructions for Lab 0.)*

Next, create a file called lab1.py by opening it up for editing with emacs:

```
emacs lab1.py
```

You'll want to reproduce all of the text below inside your **`lab1.py`** file, making sure to include all of the "**#**" signs, and to follow the capitalization shown.

```
# File:         lab1.py
# Author:       Katherine Gibson
# Date:         1/1/2016
# Lab Section:  01
# UMBC email:   k38@umbc.edu
# Description:  This program shows the proper setup of code
#               in a Python file, and greets the user with
#               the name of the programmer.


def main():

    # introduces the programmer
    print("Hello, my name is YOUR_NAME)

main()
```

Step 3:

If you haven't already, update the information at the top of the file so that it matches your details. Also make sure to replace "**`YOUR_NAME`**" in the **`print()`** statement with *your* name.

Step 4:

The pound symbols "**#**" you have in the **lab1.py** file are used to tell Python that any text after the pound symbol on that line is a *comment*. Comments are ignored by Python, and the text following a pound symbol does not need to follow any of Python's syntax rules.

Programmers use comments to explain what the code is doing, to leave notes to themselves, and to document things about the code. For example, the comments at the top of the file are called a "header comment block," and record who created the file, when, and what the file is supposed to do.

Step 5:

Let's learn a few basic emacs commands. These commands are very useful when you're programming or working in emacs.

Below, you'll find a reference sheet of the emacs commands covered so far. You'll also find a few commands that are useful in the terminal as well

Command	Meaning
CTRL+A	Go to the *beginning* of the current line
CTRL+E	Go to the *end* of the current line
CTRL+K	Remove everything on the line *after* the cursor ("kill")
CTRL+Y	Paste the line cut by the **CTRL+K** command ("yank")
CTRL+X, CTRL+S	Save the file and **stay** in emacs
CTRL+X, CTRL+C	Save the files and **close** emacs

Here are some useful commands to use in the terminal (not in emacs)

"TAB" in terminal	Hitting the tab key will autocomplete based on the available filenames. For example, typing "**emacs la**" and hitting tab will autocomplete "**la**" to "**lab1.py**"
"up arrow" in terminal	Hitting the up arrow will recall your previous command to the terminal. Hitting it again will pull up the command before that one; repeat as necessary.

Part 2: Finding and Fixing Code Errors

In this part of the lab, you'll be working on a Python file full of errors. We'll first explain how to find them, and how to understand the error messages. Following that, you'll solve the errors on your own.

Step 6:

First, let's try running the lab1.py file you have created. Save your file and exit emacs. Before running your program, make sure you enable Python version 3:

```
/usr/bin/scl enable python33 bash
```

Step 7:

If you copied everything exactly, you will get an error like this:

```
linux3[11]% /usr/bin/scl enable python33 bash
bash-4.1$ python lab1.py
  File "lab1.py", line 14
    print("Hello, my name is YOUR_NAME)
                                       ^
SyntaxError: EOL while scanning string literal
```

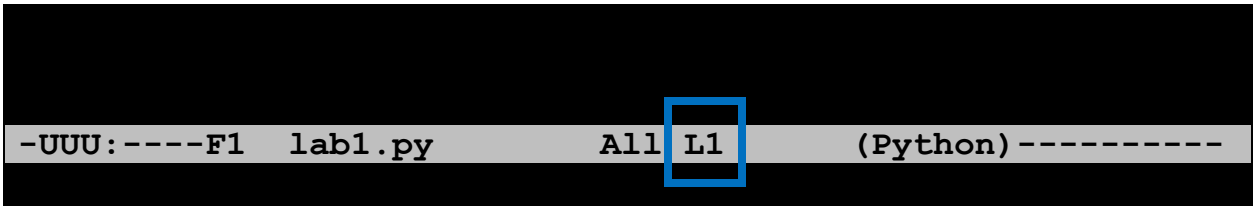
There are a couple key pieces present in this error message:

1. We know the name of the file and the line on which the error occurred In this case, it's our **`lab1.py`** file, and the error is on line 14
2. Python has attempted to pinpoint the error even further for us, using the "**^**" symbol you can see on the second-to-last line
3. Python has told us what kind of error it is, and some details:
    a. It is a syntax error
    b. Python reached EOL (End of Line)
    c. While scanning a "string literal"

Step 8:

Hopefully you've spotted the error already – the **`print()`** statement is missing the closing quotation marks. To fix it, we'll need to open **`lab1.py`** again for editing.

Once you do that, take a look at the bottom of the screen, and you should see something like this:



The "**`L1`**" you see there stands for "Line 1" – the error was on line 14, so move your cursor down until you've reached "**`L14`**" instead. Fix the error by adding the closing quotation mark, and save and exit again.

Try running your program again – if you fixed the error correctly, it should work and display the greeting you wrote.

If it doesn't work, use what you've learned to find the "new" error and fix it. Python often will only display the first error it finds, so you may find yourself doing this when working on your assignments as well.

However, if there is more than one error, start with the message at the *bottom* first. Fix that one error, and then try to run your program again.

Step 9:

Now it's time to put your bug fixing abilities to the test! Copy a file called errors.py into your **`lab1`** folder using the **`cp`** command.

```
cp /afs/umbc.edu/users/k/k/k38/pub/cs201/errors.py .
```

Before you jump into trying to fix the bugs, take a moment to read the code and figure out what the program should be doing. Then use your new knowledge of finding and fixing bugs to create an **`errors.py`** file that runs without any errors.

Step 10:

Once **`errors.py`** works, add two lines to the file's header comment block.

```
# Fixed by:    YOUR_NAME
# Date fixed: TODAYS_DATE
```

Step 11:

Test your fixed **`errors.py`** Python program with different inputs to ensure that is runs correctly. Make sure you have enabled Python version 3 before testing:

```
/usr/bin/scl enable python33 bash
```

If your testing finds a bug, fix it, and try running the program again.

(HINT: For different inputs, try big numbers and negative numbers. Remember than an integer is a whole number, so your program won't work if you give it decimals or letters.)

(REMINDER: The headers of your files (the block of comments at the top) are very important. Double check that you have correctly completed the headers for the **`errors.py`** *and* **`lab1.py`** *files.)*

Part 3: Completing Your Lab

Since this is an in-person lab, you do not need to use the **`submit`** command to complete your lab. Instead, raise your hand to let your TA know that you are finished.

They will come over and check your work – they may ask you to run your program for them, and they may also want to see your code. Once they've checked your work, they'll give you a score for the lab, and you are free to leave.

IMPORTANT: If you leave the lab without the TA checking your work, you will receive a **zero** for this week's lab. Make sure you have been given a grade before you leave!