

CSCI 141  
Spring 2019

Lab 2: Linux, Type Conversions, Operators and Operands, Print  
Formatting, Errors  
Due Date: Check Canvas

## Lab and Homework Assignments

This lab introduces you to the Linux Operating System, and explores errors, printing and operands. You should have enough time to complete this lab during the lab session. Upload your submission to Canvas before the deadline. If you have questions, be sure to ask the TA. Ask questions often. Labs are your opportunity to get personalized help!

This lab is to be done on the Linux operating system. **You must use Linux to complete this lab – basic knowledge of Linux is one of the course outcomes.** For the third lab onward, and for all assignments, you have the option to use either Windows or Linux.

### 1 Logging into Linux

Linux is an operating system with much of the same functionality as the Windows or Mac operating systems, but with a slightly different feel. The lab computers are dual boot computers, and when the computer boots up you can specify which of the operating systems (Windows or Ubuntu, which is a flavor of Linux) you want to use.

To log into your CS Linux account, reboot your CS computer; when a menu appears, use the arrow keys to select Ubuntu from the set of options, and press enter to select it. The login window should look similar to what is shown in Figure 1. You can click on the circular gnome icon to the right of "Login" to view a menu where you can customize the look and feel of your Linux session. For now, we'll assume that you've left it on Ubuntu (Default). Provide your CS account username and password (the same credentials as you used for your Windows account in lab 1) to log in.

Once logged in, the overall look and feel of your desktop will vary depending on which environment you have selected. To access system settings for your desktop environment, click on the power icon in the upper right hand corner and select System Settings... To find Thonny, click the Ubuntu icon (the top-most icon in the bar on the left side of the screen. If Thonny doesn't appear among the applications there, you can type Thonny in the search bar and it should appear. Once Thonny starts, it should look very similar to the Windows version you used in last week's lab.

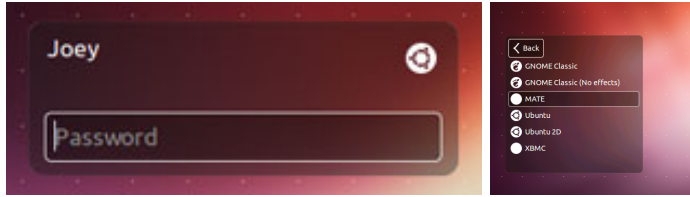


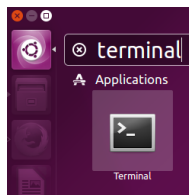
Figure 1: A typical linux Login prompt and desktop environment selector

## 2 Linux Command Line Basics

Windows, Mac OS, and Linux all provide graphical interfaces such as those you're used to using, that allow you to open programs and accomplish tasks using the mouse, icons, windows, and so on. All of these operating systems also provide another way of interacting with them, called a *Command Line Interface* (CLI). Although there is a steeper learning curve for knowing how to accomplish things using the command line, for certain tasks it can be more powerful once you know how to use it.

In this lab, you will learn the very basic elements of how to interact with the Linux command line and learn how to run Python code without the help of an IDE such as Thonny. What you will learn here is only a tiny fraction of the commands available; you can find a helpful "cheat sheet" of many commonly used Linux commands on the course website<sup>1</sup> if you want to learn more.

1. Begin by opening a command line window (also called a Terminal). Click on the Ubuntu icon in the upper left corner and type **terminal** to initiate a search; click on the Terminal icon from the results to launch a new terminal window.



2. In the terminal, the \$ sign and the space to the right of it where you type is called the command line. Commands that you issue are interpreted by a program called a shell (the default shell, or command line language, in the lab machines is bash). It is one of the many shells available in Linux.
3. You'll notice that the \$ is prepended with your username and an @ followed by the name of the computer that you are logged into. For example, `wehrwes@linux-11:~$` specifies the user `wehrwes` logged into the `linux-11` machine.
4. All of the things that you can do with the mouse when interacting with a windows environment you can also accomplish via the command line. In the following steps you will create a new directory (folder), navigate into that folder, and run python from the command line. For these instructions, code and/or sample file content or output are

<sup>1</sup>Direct link: [https://facultyweb.cs.wvu.edu/~wehrwes/courses/csci141\\_19s/labs/linuxref.pdf](https://facultyweb.cs.wvu.edu/~wehrwes/courses/csci141_19s/labs/linuxref.pdf)

shown in boxes. Type commands EXACTLY as they provided, and press return/enter to issue the command. For example:

```
whoami
```

is instructing you to type the command *whoami* on the command line and execute it by pressing return/enter. Try it out. What does the `whoami` command do?

5. Commands can take arguments, similarly to how functions in Python take arguments, except here they are not surrounded by parentheses. To create a directory, use the `mkdir` command with a single argument that is the name of the directory (folder) that you want to make. Create the directory `Lab2`.

```
mkdir Lab2
```

6. To confirm that you have made your directory, issue the `ls` command, which will list all contents of the directory where you are currently in.

```
ls
```

You should see a list with multiple items, which are the files and/or directories in your account. If done correctly, `Lab2` should be listed.

7. In graphical interface, you would double click on a folder to access that folder. In Linux, you open a directory using the command `cd`, for change directory. Enter the `Lab2` directory.

```
cd Lab2
```

8. Launch Thonny via the command line :

```
thonny &
```

The `&` is important: this allows you to continue using the terminal window AFTER you launch Thonny. Use Thonny to create a new file, *helloWorld.py*, with a single print statement : `print("hello world")`. Save the file **in your Lab2** folder. Return to the terminal, and again issue the `ls` command, and you should see the just-created file listed.

9. Just as you can run a Python program using Thonny by pressing the green Run button, you can also run a program from the command line. In the terminal window (make sure you are in your `Lab2` folder, which contains your Python program), run the `helloWorld` program by invoking the `python3` interpreter :

```
python3 helloWorld.py
```

You should see the output of your program printed to the terminal console line.

**For this lab, you must invoke the *helloWorld.py* program from the command line. To provide proof that you have done so, either upload a screenshot showing that you invoked `helloWorld` from the command line (use the screenshot tool available on Linux), or show your TA how you execute the program `helloWorld` from the command line so they can give you credit on Canvas for those points right away.**

For the remaining sections of this lab, you can run `python` either via Thonny or via the command line.

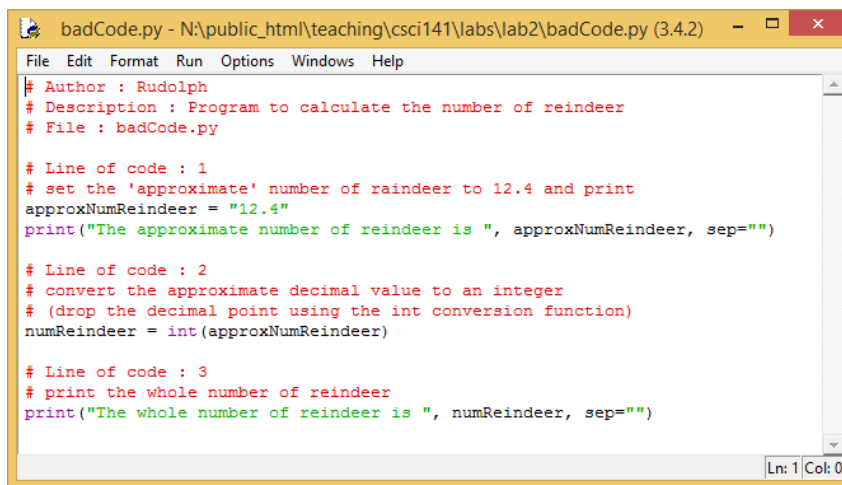
### 3 Errors, comments, and the sep argument of print

Inevitably, you will write code that will have bugs, or errors, no matter how experienced of a programmer you might be. Knowing how to find and fix bugs is a critical skill for all programmers.

Using a browser such as Firefox, go to the course website:

[https://facultyweb.cs.wvu.edu/~wehrwes/courses/csci141\\_19s/#schedule](https://facultyweb.cs.wvu.edu/~wehrwes/courses/csci141_19s/#schedule))

download the file *badCode.py*, and save it in your Lab2 folder. The contents of that file are shown in the following figure.



```
badCode.py - N:\public_html\teaching\csci141\labs\lab2\badCode.py (3.4.2)
File Edit Format Run Options Windows Help
# Author : Rudolph
# Description : Program to calculate the number of reindeer
# File : badCode.py

# Line of code : 1
# set the 'approximate' number of reindeer to 12.4 and print
approxNumReindeer = "12.4"
print("The approximate number of reindeer is ", approxNumReindeer, sep="")

# Line of code : 2
# convert the approximate decimal value to an integer
# (drop the decimal point using the int conversion function)
numReindeer = int(approxNumReindeer)

# Line of code : 3
# print the whole number of reindeer
print("The whole number of reindeer is ", numReindeer, sep="")
Ln: 1 Col: 0
```

Open the file *badCode.py* in Thonny. Read over the code, paying particular attention to the comments. Proper commenting is crucial to writing good code, so that it is easy to read by you and others. In python all comments begin with the `#` character. As a general rule:

- Include your name (the author) and a short description at the top of each .py file
- For every few lines of code, include a descriptive comment
- Use white space (blank lines) to chunk your code into logical sections

For all the labs and assignments in this course, commenting is a portion of the rubric. Get into the habit now of commenting your code well!

Notice that this code contains a brand new way of using `print`. Adding `sep=""` (short for “separator”) as the last argument to the print function will prevent print from adding spaces between items that you want printed. If you use `sep=""`, the only spaces that are included in the output are the spaces that you place explicitly into the Strings. For example:

```
print("Taking", "CSCI", 141, "woo-hoo!")
```

would print the following:

```
Taking CSCI 141 woo-hoo!
```

but

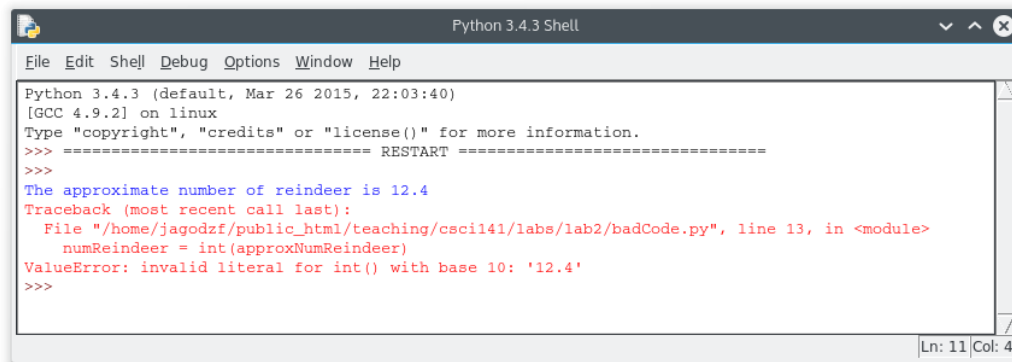
```
print("Taking", "CSCI", 141, "woo-hoo!", sep="")
```

would print the following:

TakingCSCI141woo-hoo!

In general, the `print` function defaults to " " as the separator between the arguments it prints. If you want a different separator, you can give any string to the `sep` argument, such as the empty string (`sep=""`) or a comma (`sep=","`), or any other string you'd like (e.g., `sep=" uhh, "`). Try out a few calls to `print` with different separators in the interactive Shell pane in Thonny.

*badCode.py* has a **single** intentional error. Run the code to see the error (following Figure).



```
Python 3.4.3 Shell
File Edit Shell Debug Options Window Help
Python 3.4.3 (default, Mar 26 2015, 22:03:40)
[GCC 4.9.2] on linux
Type "copyright", "credits" or "license()" for more information.
>>> ----- RESTART -----
>>>
The approximate number of reindeer is 12.4
Traceback (most recent call last):
  File "/home/jagodz/teaching/csci141/labs/lab2/badCode.py", line 13, in <module>
    numReindeer = int(approxNumReindeer)
ValueError: invalid literal for int() with base 10: '12.4'
>>>
```

Look closely at the error message. On what line number is the error? Fix the error. **Hint:** Refer to the lecture slides for fixing an error that uses the conversion function `int`. **Be sure that you modify ONLY the line of code that says `numReindeer = int(approxNumReindeer)`.** You may not modify any other line of code, nor add nor delete any line of code.

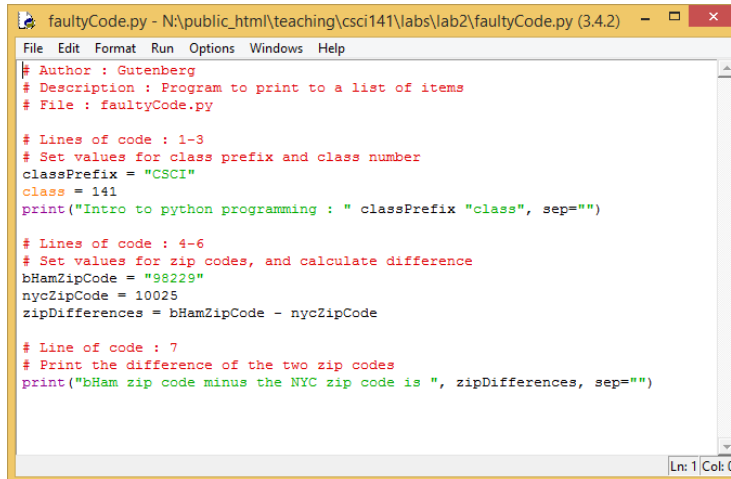
## 4 Additional errors

Download *faultyCode.py* from the course webpage and save it in your Lab2 folder. The contents of that file are shown in Figure 2. Take a close look at the code. Notice again the good commenting. The program *faultyCode.py* has intentional MULTIPLE errors. Run the code (either using Thonny or via the command line), and look closely at the error message.

Fix the errors in *faultyCode.py*. Go back and forth between fixing an error and trying to run the program. Ultimately, the goal is that the output of the program is as shown in Figure 3.

## 5 Operands and Operators

In lecture we have discussed operands and operators. Make sure that you know the definition of both. If you do not recall, review the lecture slides.



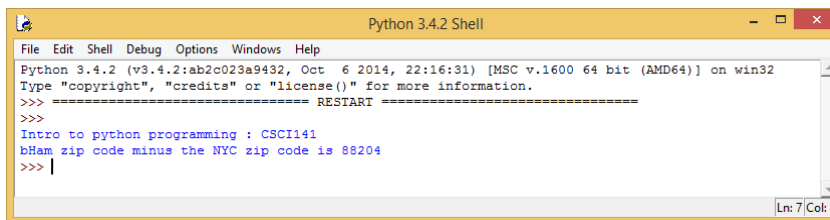
```
faultyCode.py - N:\public_html\teaching\csci141\labs\lab2\faultyCode.py (3.4.2)
File Edit Format Run Options Windows Help
# Author : Gutenberg
# Description : Program to print to a list of items
# File : faultyCode.py

# Lines of code : 1-3
# Set values for class prefix and class number
classPrefix = "CSCI"
class = 141
print("Intro to python programming : " classPrefix "class", sep="")

# Lines of code : 4-6
# Set values for zip codes, and calculate difference
bHamZipCode = "98229"
nycZipCode = 10025
zipDifferences = bHamZipCode - nycZipCode

# Line of code : 7
# Print the difference of the two zip codes
print("bHam zip code minus the NYC zip code is ", zipDifferences, sep="")
Ln: 1 Col: 0
```

Figure 2: contents of faultyCode.py



```
Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help
Python 3.4.2 (v3.4.2:ab2c023a9432, Oct 6 2014, 22:16:31) [MSC v.1600 64 bit (AMD64)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> ===== RESTART =====
>>>
Intro to python programming : CSCI141
bHam zip code minus the NYC zip code is 88204
>>> |
Ln: 7 Col: 4
```

Figure 3: Output of the corrected faultyCode.py program.

In this section, you'll solve the following programming problem: **The period key on your keyboard is broken, but you would like to want to multiply two numbers with a decimal digit.**

Let's look at an example. Suppose you want to calculate  $20.4 \times 17.7$ , but can only enter 204 and 177. The desired result is 361.08. If the user inputs the values 204 and 177, how can you convert them to 20.4 and 17.7? By using the modular and integer division operators! For example, 204 modulus 10 has a remainder of 4, which gives the decimal value .4, while 204 integer division 10 gives 20, which is the number before the decimal in 20.4.

**The Math:** Suppose that for the input values 204 and 177 you have successfully extracted the whole and decimal values (i.e., 20, 4, 17 and 7). How can you calculate the result of  $20.4 \times 17.7$ ? Multiplication of decimal values requires you to sum the following four parts:

- The first integer times the second integer
- The first integer times the second decimal
- The first decimal times the second integer
- The first decimal times the second decimal

Notice that for each decimal, we also multiply in a factor of 0.1 to make sure that it is correctly

weighted in the final product. In our example, the calculation looks like this:

$$\begin{aligned} 20.4 * 17.7 &= (20 * 17) \\ &+ (20 * 7 * 0.1) \\ &+ (4 * 0.1 * 17) \\ &+ (4 * 0.1 * 7 * 0.1) \\ \\ &= 340 + 14 + 6.8 + 0.28 \\ &= 361.08 \end{aligned}$$

1. Download *brokenCalculator.py* from the course webpage and save it to your Lab2 folder.
2. That file is incomplete. Only two lines of code have been written, which you are not allowed to edit. The rest are comments. Lines of code that say `# COMPLETE THE CODE` you will need to write. Read the comments for each section to get a sense of what code you need to write. Also, the number of `COMPLETE THE CODE` comments in that file is how many lines of python code I (Scott) wrote in my solution. It's okay if your solution uses fewer lines of code, or more, but each block of code should accomplish what the comment above it specifies.

**For the lines of code that you write, you are only allowed to use the print function, the assignment operator, and the following mathematical operators:**

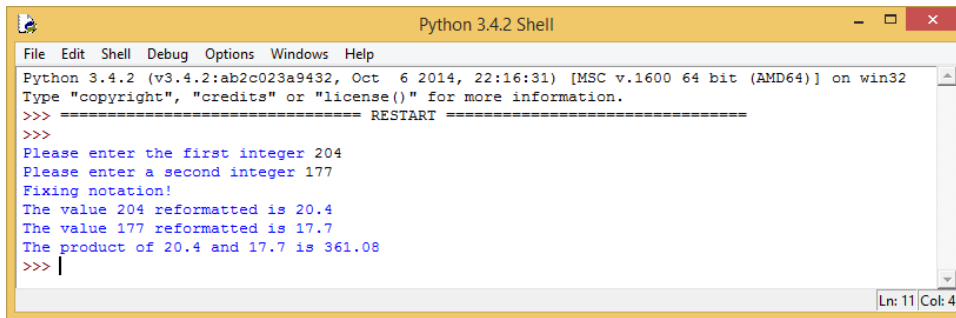
- `//`
- `%`
- `*`
- `+`

**For the lines of code that you write, you cannot use `float()`, `int()`, `nor /`.**

3. There are 7 parts to the code, labeled A through G. Here are hints for each of them:
  - **A:** This requires a single use of the print function
  - **B:** Use only the `//` and `%` operators. Follow the logic in the description above
  - **C:** This requires a single use of the print function
  - **D:** Do the same for the second integer as you did for the first integer (step B). Use only the `//` and `%` operators
  - **E:** The same as step C, but for the second integer
  - **F:** Use the Hint above for explanation on how to do this
  - **G:** This requires a single use of the print function.

**In your python code, you CAN make use of periods, but when the program is RUN, the user CAN ONLY enter integer (non decimal) numbers.**

A sample output for the completed code is shown in the following figure:



```
Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help
Python 3.4.2 (v3.4.2:ab2c023a9432, Oct 6 2014, 22:16:31) [MSC v.1600 64 bit (AMD64)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> ===== RESTART =====
>>>
Please enter the first integer 204
Please enter a second integer 177
Fixing notation!
The value 204 reformatted is 20.4
The value 177 reformatted is 17.7
The product of 20.4 and 17.7 is 361.08
>>> |
```

## Submission

Either show your working code to your TA, who will post your lab grade scores right away, else upload the following files:

- Screenshot (.png, or .jpg file) showing you invoking your helloWorld program from the command line
- Your fixed *faultyCode.py* code file
- The completed *brokenCalculator.py* file that can reproduce the output shown in the above figure

## Rubric

Screenshot showing execution of helloWorld from the linux command line	5 points
<i>faultyCode.py</i> has been fixed, and is properly commented	3
<i>brokenCalculator.py</i> uses <b>only</b> %, //, + and * operators for the lines of code that you have written	3
<i>brokenCalculator.py</i> has no syntax errors	3
<i>brokenCalculator.py</i> produces the correct output	3
<i>brokenCalculator.py</i> is properly commented	3
Total	20 points