Lab 1

CS111: Scientific Data and Dynamics — Spring 2016
Instructor: David White — Due: 2016-02-03

Welcome to your first lab! In this lab you will get to experiment with the Python interpreter, familiarize yourself with IDLE (the editor you will use to write programs), and write your first complete Python program. You will have a partner for this lab, but only for Part 6. All the early parts should be done alone.

Part 0: Organizing your work

I encourage all of you to save all your work on Denison's Novell drives – these drives are backed up regularly, giving a smaller chance of losing work. You may find them to be useful for other courses as well.

If you are using a Mac laptop (your own or one of Denison's):

(1) Click on Finder.

(2) On the menu bar use:

Go → Connect to Server...

server name = cifs://sapphire.cc.denison.edu

Use your myDenison username and password

(3) Choose Personal from the list

(4) Repeat the above steps with cifs://ruby.cc.denison.edu as the server to access the Shared drive.

If you are using your own Windows laptop, follow the instructions here: `https://helpdesk.denison.edu/helpdesk/WebObjects/Helpdesk.woa/wa/FaqActions/view?faqId=1221` (requires you to be logged into myDenison).

Both: On your desktop, look for your Personal workspace. Within this space, create a directory called cs111. Inside your cs111 directory **create a folder called lab1** to save your work into.

Note: Be very careful at the end of every class to save all your work into your personal workspace (saffire.cc.denison.edu) as any files that you save locally on the laptop will be gone the next time you log in.

Part 1: Visualizing a program's execution

At the following link, also available on the course webpage, the following code has been copied into a visualization website called Python Tutor. This website simulates the precise steps that Python follows when executing code you give it, i.e. when you click the Run button in IDLE. If the links are not working, just copy and paste the code there directly.

```
def boxVolume(length,width,height):
    volume = length*width*height
    return volume

def costOfWood(unitCost,amount):
    return (unitCost * amount)

def main():
    v = boxVolume(5,3,3)
    cost = costOfWood(5.25,v)
    print("Building the box will cost $",cost)

main()
```

Here is a high-level description of what this code does. The code first computes the volume of a box (in units of cubic feet, i.e. $feet^3$). It then computes the cost of building such a box out of solid wood, where the cost of a cubic foot of wood is $5.25. Visualize the execution of this code by clicking Forward over and over again, and answer the following questions. If you're having trouble, please read the rest of the lab, especially Part 5.

(1) Which line numbers are defining functions (there are three)?

(2) Which line numbers are calls to functions?

(3) Which line numbers are the first four lines of code executed?

(4) Why isn't the line `volume = length*width*height` executed before the line `main()`?

(5) In Step 7 of 16, what are the values of the variables `length, width, height`?

(6) In Step 10 of 16, when the value of the variable volume is returned, which line number is that value returned to? What happens to that numerical value when it appears on that line?

(7) What parameter values are sent to the function `costOfWood` in line 10?

(8) What is returned (implicitly) by `main()`?

Now click Edit code and modify the code so that it finds the price of a box of dimension $3 \times 2 \times 4$. Report the result of this computation. You do not need to put this code into the shared drive or write about it in your Lab Report, since I've already given you a high-level description of the code above.

Part 2: Typing up a program in Python

Open IDLE (the blue and yellow icon program) as you did in class. Create a new window in IDLE and type in the code below (changing the comment at the top to include your name and the date):

```
# A program to calculate compound interest
# Author: (Enter your name here)
# Date: (Enter the date here)
```

```
print("This is a program to calculate compound interest.")
principal = float(input("Enter the principal amount (dollars): "))
timeInYears = int(input("Enter the time (years): "))
rate = int(input("Enter the interest rate (percent): "))
newAmount = principal * (1 + rate/100.0) ** timeInYears
print("After", timeInYears, "years the amount will be", newAmount, "dollars.")
```

Now save this file as <username>_compoundInterest.py into your Lab1 folder in the folder CS111/Assignment Inbox/<YOUR USERNAME> on the shared drive. For example, mine would be called whiteda_compoundInterest.py. Upload all your files from this lab to this Lab1 folder. Once the file is saved, you can run it by clicking the Run button or Command+F5. Notice that this program has no functions whatsoever. In the future we will want our programs to have main() functions.

Congratulations! You have your first running program in IDLE.

We will now determine what this program does. The first several lines start with a #. In Python, this is the symbol for comments (parts of the program that the interpreter should ignore). They are meant solely for anyone reading the code to have a better understanding of what it does. Though your code would run the same without them, it is always good practice to include information about it (e.g., author, date started, description of program, etc.) at the beginning. You can also add comments to the main body of the code, if there is something complicated that you want to make a note of for the reader of the code (e.g., the person grading your assignment, another person you are working with on a project, or even yourself when you come back to some code months later).

Now that you have your first program, look at the CS111 Project Report Guidelines handout. In part 3 it explains that you're supposed to **write a high level description** of your program. For Lab1, **you should also trace your program and write a line-by-line description of what it does**. You should reference the program via line numbers, and write a line for each of the 6 lines of code that Python sees. Don't forget to include a docstring for every function you hand in.

Part 3: Messing it up. Now that you have a working program, let's see what happens when you deliberately add some errors into it. The idea is to learn the error messages produced by the interpreter so that when you get errors in more complex programs in the future you can understand what's going on.

Make the following changes in your program, making sure to revert to the original version before introducing the next change. **Save the error reports** that you see for each error in the Results and Conclusions section of your lab report, along with a **one-line explanation of why** you got that error message (or didn't get one). Does the error message only appear on some inputs? If so, state which ones in your report.

(1) Capitalize the "p" in the first "print" statement.

(2) Change all the double quotes in the first print statement to single quotes.

(3) Remove the word "float", along with the first "(" and the last ")" on that line.

(4) Replace "float" with "int". Be sure to try several different inputs when testing.

(5) Change “100.0” to “10.0” in the computation statement. Did you get an error message? Was the output computed correct?

(6) Indent the line that begins with rate.

(7) Misspell the variable name “newAmount” in the final print statement.

(8) Run the correct program, but use “CS” (or any other string) as one of the inputs.

Part 4: Types of Errors. Record your answers to the questions in this section in the Results and Conclusions section of your lab report.

A *syntax error* is caused by a violation of the syntax, or grammar, of Python. The Python interpreter sees these errors on all inputs. Several examples are given in Section 2.3.

Question 1. What happens on your screen when Python detects a syntax error? Which of the errors in Part 3 are syntax errors?

Question 2. Create another syntax error and record the change you would need to make to the code in order to get this syntax error, as well as an explanation of why it is a syntax error.

An *exception* is the sort of error which arises only during execution and not necessarily on all inputs. Consider the following code:

```
distanceTravelled = int(input("Enter the distance travelled (meters): "))
timeInSeconds = int(input("Enter the time that has passed (seconds): "))
velocity = distanceTravelled/timeInSeconds
```

Question 3. Can you think of any inputs for which this would cause an exception error?

The third main type of error which can occur is a *logical error*, i.e. if the algorithm does not perform as expected and so does not correctly compute the desired output. Again, the algorithm may not fail on all choices of input, and this is why creating a wide battery of tests is a good idea. Consider the following code:

```
x = 0.0
y = 0.0
z = 0.0
x = float(input("Enter the first number: "))
y = float(input("Enter the second number: "))
y = float(input("Enter the third number: "))
mean = (x+y+z) / 3.0
```

Question 4. Can you spot the logical error in this code? Give examples of inputs (first, second, third) for which this algorithm would produce correct output and for which it would produce incorrect output.

Hint: if you’re having trouble finding it, consider using Python Tutor to go through this code step by step.

Because logical errors are so difficult to spot, you should always test your code before submitting it. Because syntax errors and exceptions are so easy to catch, you should never submit code which creates such errors when I run it. Submitting such code will result in a hefty penalty to your grade on that assignment.

Part 5: Writing your own program. Okay, time to try your hand at writing a program all of your own. Write a program `username_ageInSeconds.py` (where username is *your* Denison username) that prompts the user to enter his/her age in years, months, and days (it should have 3 different `input` statements) and prints on the screen how many seconds you have been alive. To make things simple, you may ignore the existence of leap years and assume that all months have 30 days. Remember to add documentation at the top (your name, the date, what the program does). Remember that we want our programs to have `main()` functions which interface with the user, call our other functions, print output to the screen, and which don't return anything.

Your program should have two functions. One should be a `main()` function and the other should be a function called `ageCalculation` that takes three parameters (years, months, days) and returns a number of seconds. The `main()` function should get the input from the user and store it as variables, then call `ageCalculation` with those inputs as the parameters, then it should print the result of the computation along with a string telling the user what this number represents (i.e. their age in seconds). Your function `ageCalculation` should have a comment which describes what it does, what parameters it takes, and what it returns. Follow the model on page 118 of the textbook.

Once you have written the program (and it runs without any errors) test it on the following inputs, and verify that it is working correctly (i.e. that there are no logical errors):

- 1 year
- 20 years, 2 months
- 18 years, 7 months, 6 days

Include the results in your lab report. In order for the main method to be run (i.e. in order for computation to occur) you must put a non-indented line

```
main()
```

at the very bottom of your code. This should be the only non-indented line which does not start with the keyword 'def'.

Part 6: Testing your Code. In this part of the lab you will work with your partner. First, check to make sure you both have code which correctly completes Part 5.

Question 5. Did you both come up with the same solution? If the solution was different, explain why both are correct.

With your partner you are to write a program called `username_conversions.py`, which has the ability to do several different conversions. In this file you will create two functions which do conversions and a main() function. Both conversion functions should have comments as in the previous section (following the model on page 118).

The first is a function `heightInInches` which takes as parameter a float representing height in meters. Your function should compute this height in feet and inches, and return these two values, e.g. via a return statement like

```
return(ft,inches)
```

People usually give their height with an integer number of feet and inches (e.g. 6 feet 3 inches), so use the int() command to round down to an integer once the computation is finished. Warning: Don't round off feet before figuring out how many inches there are!

The second is a function `weightInPounds` which takes as parameter a float which is weight in kilograms, computes the weight in pounds, and returns that value. This should be kept as a float rather than rounded to an integer.

Trade code with your partner. You are now looking at someone else's code, but of course that's okay because it's your lab partner and you can always share lab related code with your lab partner.

Question 6. Can you understand what these functions do based on the comments alone? Does the code look cluttered or is it easy to read like the code in the textbook? Do any lines run over the right edge of the IDLE window (so that you can't read them without scrolling)?

Write feedback about the style of the code you are reading and include it in Section 5 of your lab report, additionally giving it to your partner.

Come up with at least 5 test values for each of the two functions your partner wrote, in a way analogous to Part 5. Try to think of common errors and which values they might occur on. Think of boundary cases (e.g. what happens at zero?). **Write your test cases and the correct outputs in your lab report**. In your `main()` function, call your partner's function from above (either `weightInPounds` or `heightInInches`) on the test cases you came up with, and print the results. For example, this might look like `print(weightInPounds(73.4))`, inside your `main` function.

Question 7. Does your partner's code work correctly on these test cases? Do you think it'll work correctly on all inputs? (Note that if you vouch for your partner's code and it's wrong you'll lose points)

Write your conclusions in Section 5 of the lab report and also share them with your partner. If you diagnosed a problem with your partner's code he/she can fix it before uploading the lab to the shared drive. Make sure you both upload the same code (and comments too) because I may only read one of them.

Congratulations! You've now completed your first lab and learned how to write programs, write and call functions, respond to errors, and test your code.

Documentation and Style

Ten percent of your lab grade will be based on following these documentation and style conventions:

- Every file you hand in should have your name/the project name/start date on top.

- Your code should be well-documented. As a rule of thumb, you should aim for at least one comment for every three lines of code.
- Use descriptive variable names. For example, `timeInSeconds` is good, but `t` is not.
- Assign to variables any values that have meaning or that you use more than once.

Submitting your lab

Upload to your folder on the Shared Drive the following files by class time on February 3rd:

- `username_compoundInterest.py`
- `username_ageInSeconds.py`
- `username_conversions.py`
- Lab Report, following the specifications in the handout, answering all questions in this document, and including all error reports, explanations, errors you created, test values you created, and feedback on your partner's code. Do not forget to write the introduction (why was this lab important?) and a concluding paragraph in the conclusions section (what did you learn from this lab?). You and your partner should have the same code, except for the username. The docstring on top of each file can have both your names.