CptS 111 — Lab #1

Getting Started

Objectives:

- Learn how labs will be run and lab tasks scored
- Get Python up and running on your laptop
- Get to know your labmates
- Practice some Python commands in the IDLE Shell window

Lab Procedure: Each lab consists of tasks that must be done in order, and you'll get one point for each completed task. You also get a point for showing up to your lab on time and working diligently on your lab tasks. Your overall grade for each lab will be a percentage score based on your total points, including the attendance point, with a maximum grade of 100%. You must demonstrate to your TA that you've completed a task or you won't get credit for it. Credit is binary: you either get credit or you don't. The same procedure will be followed for all labs.

Your TA will help you with the tasks as needed. You're also free to ask for help from students at your table and to offer it as well. **However, try each task yourself first because, seriously, labs are where you learn to code.**

Task 1: Getting Acquainted

Introduce yourselves and get to know a little about each other besides your names and where you live. If you've decided on a major, what is it? If you're undecided, what are you considering? What are your hobbies? What's one fun fact about yourself? Are you from a country other than the United States? Are you fluent in another language? Do you play a sport? Are you famous for something? Are you a twin? You get the idea!

Task 2: Installing Python on Your Laptop

Be sure that you have a correct version of Python on your laptop (3.6.x-3.11.x). If you don't, download it from `http://python.org/download`. If you run into problems, ask for help. Mac users, you might also need to install ActiveTcl from `http://www.activestate.com/activetcl/downloads`. Chromebook users, see the YouTube video in the Useful Links module in Canvas.

Task 3: Using the IDLE Shell Window

IDLE is an "Integrated Development Environment" (IDE) that's installed with Python; it allows you to run the Python interpreter. Please use IDLE during labs to make life easier for your TA. All instructions will be given with IDLE in mind. IDLE has an interactive window called the Shell window where you can enter commands and run programs; it also has an Editor window which allows you to write and save programs. The IDLE IDE is simple compared to other IDEs, but as you'll discover, it still provides some help with commands.

Windows: To start IDLE, click the Start button in the lower left corner of the screen. Type "IDLE" and your first option should be an app called "IDLE (Python 3.xx 64 bit)." Click on this option which should bring up a window labeled Tk Python Shell. The interactive Python prompt (>>>) indicates that you can enter Python commands that will be executed by the Python interpreter when you press the return key.

macOS: In a Finder window look for a directory called Python 3.x in the Applications folder. Click on the directory, and then double click on IDLE. This will open up the IDLE Shell window where you can enter Python commands. The interactive Python prompt (>>>) indicates that you can enter Python commands that will be executed by the Python interpreter when you press the return key.

We use the Shell window to run Python interactively and also to run programs created in an Editor window (we'll use the Editor window in Lab #2). Type the statements that are in boldface font in the Shell window. Note that text in boldface font in labs and PAs means it must be typed in. **Be sure to read and understand the comments; don't enter the commands without thinking because if you do, you won't learn anything!**

```
>>> # Assign the integer literal 2 to the lvalue cat.
>>> cat = 2
>>>
>>> # type() returns the data type of its argument.  What should it
>>> # be?  Try to figure this out before entering the command!
>>> type(cat)
>>>
>>> # Assign the string 'dog' to the lvalue cat.
>>> cat = 'dog'
>>>
>>> # Now what is cat's type?  Again, answer this before entering the
>>> # command!
>>> type(cat)
>>>
>>> # We can prompt a user for input (type a word after the colon).
>>> entry = input('Enter a word: ')
>>> entry
>>> type(entry)
>>>
>>> # Whatever is entered is ALWAYS a string.  The = sign is the
>>> # assignment operator (not the same as in math!), and it assigns
>>> # the value returned by input() to the name 'entry' in the example
>>> # above.  input() always returns a string, but we can convert the
>>> # string using, e.g., the int() or float() functions, and this
>>> # value is then assigned to the name by the assignment operator =.
>>> integer = int(input('Enter an integer: '))
>>> type(integer)
>>>
>>> # Cool, eh?  Note that we used nested functions above, i.e., the
>>> # int(input()).  The inner function is executed first, and the
>>> # outer function is executed second.  We can nest lots of functions,
>>> # but usually we limit ourselves for readability.
```

```
>>> #
>>> # Python ignores comment lines which start with an octothorpe
>>> # (AKA hash or pound sign).
>>> # This is a comment.
>>>
>>> # Comment lines can be placed next to a command, but they should
>>> # be used carefully because they can be easily overlooked.
>>> type(cat)  # No comment.
>>>
>>> # See what happens in the Idle Shell window when you type a comment
>>> # without using an octothorpe.
>>> SpongeBob
>>>
>>> # Next let's print something.
>>> print('The cat is a', cat, '.')
>>>
>>> # In the  previous output, the period didn't appear where it
>>> # should have.  sep has a default value of a blank space, but if
>>> # we use sep='', we can change it to no space.  Note, however,
>>> # that we have to add a space before the second single quote to
>>> # print the sentence correctly.
>>> print('The cat is a ', cat, '.', sep='')
>>>
>>> # Try using sep='*' in the command above.  Analyze the result.
>>>
>>> # Now give this a try!
>>> print('5\n4\n3\n2\n1\nBlastoff!')
```

In the final item, we printed a single string, but `\n` is a newline character that prints as a new line! Figure out how to print the following output as a single string:

```
1
2 Hello-O-World, and
3
4
5 Goodbye-O-World!
6
```

The function `print()`, by default, returns you to a new line (this can be changed!). To create two blank lines, you must use the newline character 3 times. `print()` returns you to the IDLE prompt at the beginning of the first blank line, but to retain the blank line, you must include a newline. The second newline takes you to the second blank line, and the third newline retains the second blank line and then takes you to the beginning of the text `Goodbye-O-World!`. If you think this is a little confusing, it's because it is, but you can try using no newlines, then one newline, then two newlines, then three newlines, and this should help you see what's happening.

If you use IDLE in Windows, you can press on the Alt and p keys at the same time repeatedly to repeat previous commands. In macOS, you use the Control and p keys.

Show your completed work to your TA to get credit and then move on to the next task. If your TA is busy, just move on to the next task and flag them down when they're close by.

Task 4: A Little More Practice in the IDLE Shell Window

Next, let's do some numerical work in the IDLE Shell Window.

```
>>> # We can use the IDLE Shell window like a calculator!  However, we
>>> # have to be careful with the precedence of math operators, i.e.,
>>> # the order of operations which is: ** > *, /, //, % > +, - where
>>> # *, /, //, and % all have the same precedence and + and - have
>>> # the same precedence.  However, parentheses can be used to change
>>> # the order of operation, and if - is used as a unary operator,
>>> # i.e., it only operators on one number, to negate a value, then
>>> # it will take precedence over all other operators.  Try the
>>> # following:
>>> 2 * 3 + 5 / 7 ** 9
>>>
>>> 2 * (3 + 5) / 7 ** 9
>>>
>>> (2 * 3 + 5) / 7 ** 9
>>>
>>> 2 * (3 + 5 / 7) ** 9
>>>
>>> 4 * -11
>>>
>>> # Clearly, we need to be careful, but it's pretty handy to be able
>>> # to do calculations so easily.  Next, let's practice using
>>> # augmented assignment which allows us to be a little lazy.
>>> # First, we'll initialize a value for x by assigning 1 to the
>>> # lvalue x.
>>> x = 1
>>> print(x)
>>>
>>> # Suppose we want to increase the value of x by 1.  We can do
>>> # the following:
>>> x = x + 1
>>> print(x)
>>>
>>> # However, we can use augmented assignment as well.
>>> y = 1
>>> print(y)
>>>
>>> y += 1
>>> print(y)
>>>
>>> # y += 1 is identical to y = y + 1, but it's less work.  We can
>>> # use any other math operator with augmented assignment, but
>>> # the addition operator is probably the most common.
```

Show your work to your TA to get credit for this task.

Congratulations on finishing your first lab! You're off to a good start!