CptS 111 — Lab #2

Exploring Python Basics

Learning Objectives:

- Use the IDLE Shell window to practice some operations, import a module, and practice a little with f-string string formatting
- Use the IDLE Editor window to write a Python program (script)
- Run Python programs in the Shell window
- Write programs that make use of concepts from zyBooks and save them to a directory/folder

Prerequisites:

- Have Python installed on your laptop
- Be able to run Python interactively in the IDLE Shell window

---

Task 1: Practicing some commands in the IDLE Shell window

Open the IDLE Shell window. First let's explore the difference between regular division and floor division. Enter the commands below exactly as given, and note the differences in results.

```
>>> a = 11
>>> b = 5
>>> c = 5.0
>>> a / b    # a and b are both integers; the result?
>>> a // b   # a and b are both integers; the result?
>>> a / c
>>> a // c   # a is an integer and c is a float; the result?
```

What you should have noticed is that regular division `/` always gives a float result, but for floor division `//` it depends on the type of values you use. If both numerator and denominator are integers, the result will be an integer; if either the numerator or the denominator is a float, the result will be a float.

Next let's explore the floor division and modulo operators as well as the **`divmod()`** *function. You need to know and understand how to use all three of these.* First note that Python allows us to use multiple lvalues as long as the number of values to the right of the assignment operator = is the same. This is known as **simultaneous assignment**. Enter the following:

```
>>> whole = a // b
>>> remainder = a % b
>>> whole
>>> remainder
>>> whole, remainder = a // b, a % b
>>>
>>> # Note that there are two lvalues matched to two values on the right
>>> # side of the assignment operator (=) in the statement above.
>>> # The two lvalues are separated by a comma, and the two values are
>>> # as well.  The values of whole and remainder are the same as before
>>> # because we used the same values for a and b:
>>>
>>> whole
>>> remainder
>>>
>>> # Now let's consider the divmod() function.
>>>
>>> divmod(a, b)
>>>
>>> # We see that the divmod() function gives two values as well (in
>>> # the form of a tuple).  Thus, we can use the following:
>>>
>>> whole, remainder = divmod(a, b)
>>>
>>> whole
>>> remainder
>>>
>>> # Comparing the results above, we see that the divmod() function
>>> # combines floor division and the modulo operator.  As such,
>>> # it's a more efficient method for obtaining the two results.
>>> # Let's try another more complicated example:
>>>
>>> tot_secs = 5439
>>> hrs, rem_secs = divmod(tot_secs, 3600) # 3600 secs in one hr
>>> mins, secs = divmod(rem_secs, 60)  # 60 secs in one min
>>> print(f'There is/are {hrs} hour(s), {mins} minute(s), and {secs}\
seconds in {tot_secs} seconds.')
>>>
>>> # Let's spend some time analyzing this.  We assigned 5439 seconds
>>> # to tot_secs.  Then we used the divmod() function with the arguments
>>> # tot_secs and 3600.  We chose 3600 because we want to know how many
>>> # hours are in 5439 seconds.  We assign the floor division part, i.e.,
>>> # the number of whole values, to hrs and the remainder (the modulo)
>>> # to rem_secs.  In the next line, we again use the divmod() function,
>>> # but this time we use the arguments rem_secs and 60 because we've
>>> # already used a lot of the seconds to create the hrs and only have
>>> # the remaining seconds.  We use 60 because now we want to calculate
>>> # the number of minutes in the remaining seconds, and there are 60
>>> # seconds in a minute.  We assign the whole values to mins and the
>>> # remainder to seconds, i.e., the remaining seconds are just that,
>>> # seconds!  You may need to think about this a bit, but please do
>>> # until you understand because you'll be using it in future labs and
>>> # also in a programming assignment.  BTW, the \ at the end of the
>>> # first line of the print() command allows you to continue typing on
>>> # the next line, but Python actually waits until there's an end
>>> # parenthesis, so it's not really necessary.  It can just make your
>>> # code more readable (but not in IDLE).
```

Don't forget that floor division discards the decimal remainder, resulting in a whole number. Modulo, on the other hand, gives the remainder. As stated previously, the **`divmod()`** function combines floor division **div** and the modulo function **mod** into a single function.

Next we'll import the `math` module. Do the following in precisely the order given.

```
>>> sin(a)       # Don't panic.  You didn't do anything wrong.
>>> import math
>>> sin(a)       # Still don't panic.
>>> math.sin(a)
>>> math.tan(a)
>>> math.exp(a)
>>>
>>> # When we use 'import math' to import the math module, we _must_
>>> # use dot notation together with the function.  We can also import
>>> # a module and assign it to an alias to reduce the amount of
>>> # typing we must do.
>>>
>>> import math as m
>>> m.sin(a)
>>>
>>> # Note, however, that we must still use dot notation.
>>>
>>> sin(a)
>>>
>>> # In computing, you must always take care when using floats because
>>> # they can never be exact (unlike integer values).  We can see this
>>> # when we consider the value of sin(pi) which should have a value of
>>> # 0.  Let's see what we get instead.
>>>
>>> m.sin(m.pi)
>>>
>>> # As we see, we get a very small number, but it isn't zero.  If you're
>>> # doing numerical calculations, you need to use a test, and set very
>>> # small values to zero when necessary!
```

Finally, we'll end this task by practicing some f-string string formatting.

```
>>> # Let's begin with a string.
>>> cat = 'dog'
>>> print(f''My cat's name is {cat}.'')
>>> print(f''My cat's name is {cat}, but my {cat}'s name is cat.'')
>>>
>>> # Earlier we defined 'a' to be 11 and 'b' to be 5.  Let's see what
>>> # we get if we use f-string string formatting without any type of
>>> # format specification.
>>>
>>> print(f'a = {a}')
>>>
>>> # Now let's add some zeros in front of the 11.
>>>
>>> print(f'a = {a:04d}')
>>>
>>> # Let's analyze this.  Within the curly braces, 'a' is the value
>>> # we want to print.  It's followed by a colon (:) which indicates
>>> # that what follows specifies the format.  The '0' is what will
>>> # replace any of the '4' spaces without a value, and the 'd'
>>> # means we're using a decimal integer.  Using 0's in this way is
>>> # called zero padding.  Note the difference between the results
>>> # above and for the statement that follows.
>>>
>>> print(f'b = {b:04d}')
>>>
>>> # Next, let's consider a float value.
>>> pi = 3.141592653
>>>
>>> # Explain the results you get for each of the commands below.
>>>
>>> print(f'pi = {pi}')
>>> print(f'pi = {pi:f}')
>>> print(f'pi = {pi:g}')
>>> print(f'pi = {pi:.2f}')
>>> print(f'pi = {pi:.4f}')
>>>
>>> # And really finally, as a side note, pi and m.pi aren't the same!
>>> # Explain why!
>>>
>>> pi
>>> m.pi
```

Demonstrate that you have successfully completed this task to your TA for credit.

Task 2: Using the IDLE Editor window and saving files

In this task you'll use the editing capability of IDLE to edit a file and then save it to a directory. Click on the `File` menu in IDLE and then highlight and click on `New Window`, or, alternatively, type Control-N (i.e., hold down the control key and then type the N key) (Windows) or Command-N (macOS).[1] This will open up a new window labeled `Untitled`. This window is not interactive; it is the window in which you will edit your Python programs.

Click on the `Untitled` window to select it. Next type the following

```
# Your name
# Today's date
# CptS 111
# Lab #2, Task 2

# My first program.  It's a friendly greeting.

print('Hello, World!')
print('Goodbye, World!')
```

[1]In the future we will just write C-X with the understanding that "C-" is the control key on a Windows machine and the command key on a Mac.

For "`Your name`," type your name, and for "`Today's date`," type today's date. Next you need to save this program. Click on `File` and select `Save` or type C-S.

> **Windows:** A new window will appear that requests information about how to save your file. Click the `My Documents` icon that appears on the left. Then create a new folder called `CS111` by clicking on the icon in the upper right that looks like a folder with a sparkle in one corner. A new folder will appear and the text will be selected. Type `CS111`. Now, double click on this new folder and create another new folder called `Lab2`. Now double click on `Lab2`. Find the field labeled `File name:` at the bottom of the window and type `lab2_t2.py`. Click on the `Save` button or else hit the `Return` key.
>
> **macOS:** After you type C-S and the `Save` window opens you will see a field in which you can type the name of the file. If you don't see columns of directories and files, click on the downward-pointing arrow. The window will expand and reveal some of your directories and files. You should see `Documents` centered just under the writing field. Click the button labeled `New Folder` and type `CS111` in the space for the name and then click on `Create`. Click again on the `New Folder` button, type `Lab2` in the writing field, and then click `Create`. Finally, type `lab2_t2.py` in the `Save As:` field, and click on `Save`.

The edit window should now be labeled `lab2_t2.py`. Above this window, look for the `Run` pulldown menu. Click on `Run` and then on `Run Module`. Alternatively, just hit the F5 function key (Windows) or fn-F5 (macOS). The results of your program should now appear in the interactive window (the Python Shell window).

Demonstrate that you have successfully completed this task to your TA for credit.

Task 3: Body Mass Index

The body mass index (BMI) is given by the formula

$$\frac{703 \times weight}{(height)^2}$$

where $weight$ is weight (or, more precisely, mass) in pounds and $height$ is height in inches. What BMI tells us about one's body or health is subject to debate, but we won't worry about this. Instead, we'll write a program to calculate the user's BMI.

In IDLE, open a new file (C-N), and write a program that prompts the user for his or her weight and height (in pounds and inches). The user should be able to enter `float` values, and your program should calculate the BMI and then report the BMI to the user rounded to one decimal place using `round(bmi, 1)`, where `bmi` is the dummy name for the lvalue you choose. Use string formatting as you did in Task 1 to print the results. Be sure to include the appropriate header information as you did for Task 1 here and for all programs you write in the future! A couple of examples follow.

```
Enter weight [pounds]: 160
Enter height [inches]: 69
Your BMI is 23.6.
```

Here's another example (this time with float inputs):

```
Enter weight [pounds]: 139.5
Enter height [inches]: 63.5
Your BMI is 24.3.
```

Save your program in your `Lab2` directory as `lab2_t3.py`. When it works, demonstrate to your TA that it does what it should.

Task 4: Calculating your CptS 111 score

All your assessment scores are normalized to 100%. Thus, your overall score in CptS 111 is given by:

$$\begin{aligned} score &= 0.1 * (hw + poll + lab + pa) + (0.16 * mid1) \\ &+ (0.2 * mid2) + (0.24 * final) \end{aligned}$$

where hw is your zyBook activities score, $poll$ is your iClicker polls score, lab is your labs score, pa is your programming assignments score, $mid1$ is your first midterm score, $mid2$ is your second midterm score, and $final$ is your score on the final exam.

Write a program that will prompt the user for all their scores one at a time and report their final overall score **rounded to one decimal place**. Be sure to include the same header information you used in Tasks 2 and 3.

Here are some examples:

```
Please enter the following information:

Homework score: 100
iClicker score: 100
Lab score: 100
PA score: 100
Midterm 1 score: 40
Midterm 2 score: 40
Final exam score: 40

Your score for CptS 111 is 64.
```

```
Please enter the following information:

Homework score: 80
iClicker score: 80
Lab score: 80
PA score: 80
Midterm 1 score: 91
Midterm 2 score: 91
Final exam score: 91

Your score for CptS 111 is 86.6.
```

When your program is working properly, show it to your TA and demonstrate that it works. Be sure to save it to your `Lab2` directory as `lab2_t4.py`.

After you've completed this lab, you should feel pretty good about yourself!