Module 4 – Float and Integer Data

Instructions

1) If not already open, from the Anaconda Navigator Home screen, launch Sypder

2) In Spyder, go to "File", then "Open", then navigate to the location where you saved: "4_Numeric.py", and select it to open the script file.

3) Integer and Float Basics

The fundamental characters to remember are outlined on Lines 11-12 in the comment section, and Lines 8-9 give some hints about the order in which they are evaluated.

Run Lines 13-14 and then **16-17**, and modify/experiment to get a sense of how input/output works with these operators

See the note in the comments on Lines 19 and 20 regarding data types of inputs and how they relate to outputs. **Run lines 21 and 22**. Try typing in: type(d) in the input console after running Line 22 to confirm. Line 23 gives an example of how you can create a new data type simply by calling its namesake function. **Run Line 23**, which converts the value stored for variable **d** to an integer. You can use **str()** and **float()** to convert data to those types in similar fashion. Experiment with this using the variable **d** in the console yourself.

Run lines Lines 25-27. See the output? To produce a rounded/more manageable output, Line 28 turns to the formatting operators from the previous module. **Run Lines 28-30 together** and compare the outputs. Again, note that the format flag comes first, followed by the **mod** (%) character, then the variable.

Lines 32 and 33 show you how to use more than one – **run them next**. You will see these both throughout this workshop and when you find code from various sources, so it's helpful to see repeated examples of formatting operators. Experiment some with modifying Lines 28, 30, and 33 so you get a sense of how to work with these and set them up to your liking.

In contrast to division, which produces float data by default, note that Lines 35-39 demonstrate that addition, subtraction, and multiplication produce integer data by default. **Run lines 36-39**, and confirm by using the **type**() method for both **y** and **z** in the console. Of course, if you ever need to convert these values, you can do so by calling the namesake function: e.g., entering z = float(z) in the input console will now store a float value of the original multiplication output, which was first produced as an integer.

4) Working with User Input

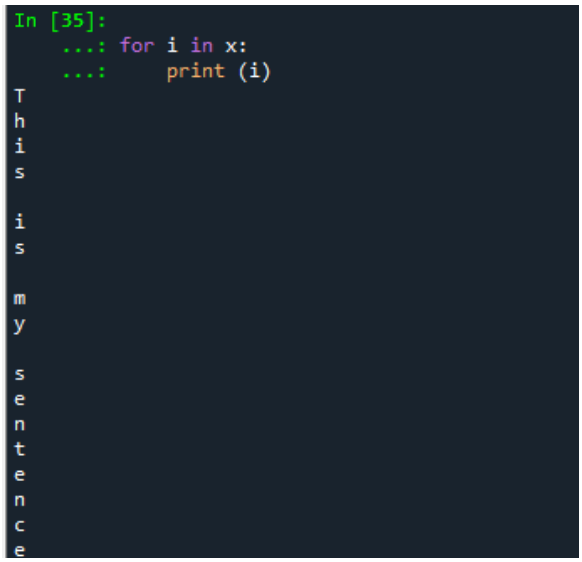
Line 42 offers a first glimpse at how to take user input in Python and integrate it with your programs. **Run Line 42-44 together**. You'll need to enter a value in meters to continue. Once you do, the program will finish. This little program takes input from a user - in this case, an elevation value in meters – and converts it to feet. It even uses a format flag to show only one value after the decimal point.

Activity: modify this short program to take a user's input in feet and convert it to meters

5) Working More with Index Values

Now that you got a brief overview of how numeric data work in Python, this section offers more details on index values, shows some examples of other allowable operations and syntax to extract subsets by using index values, and provides some additional opportunities to practice using and/or modifying them.

First**, run Line 52**, which as was the case in the previous module, is string data. Then, read the comment on Line 54 and then, **run Lines 55 and 56**. This shows each character at each position in the string data object.



Run Lines 58-60, which give you examples of how to access a single, particular value through its index position.

In contrast, Lines 63-66 give you ways to extract ranges of values (often referred to as a 'slice'). Line 63 should look quite similar to the example in the previous module. But Lines 64-66 differ. The negative values count backwards from the end of the range of values, instead of counting from the beginning (at 0). **Run these each one at a time** to compare outputs.

The open/unspecified ends on either side of the colon icon (:) also help you search through to the end of the range by default from the location of the slice. But the rules about the value on the left being included and the value on the right noting the upper boundary still apply.

Activity: Using what you know about index values, write a print statement that extracts 'is my' from the string data stored with the x variable, set on Line 52.

This completes the structured part of Module 4. Feel free to experiment some more with setting variables and values, working with print statements, or anything else you are curious about.