Module 6 – Conditional Statements and Flow Control

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: “6_ConditionalStatements.py”, and select it to open the script file.

3) Introduction to conditional statements

We saw a couple examples of these in the previous module. Indeed, these two often go hand-in-hand, as ‘while’, ‘for’, and ‘if’ anchor the major flow control statements, with some other requisite components. Conditional statements help provide greater functionality and flexibility to loops, and are central to good programming Python.

Comparison operators are also essential building blocks of flow control, and the main operators are provided and briefly explained in the commented section of the code (Lines 6-14). These Boolean expressions produce a yes/no (or True/False) output that we use to control program flow. Lines 16-20 touch on the concepts of one-way, two-way, and multi-way decisions.

Two-way decisions are those where we want the program to do a certain thing if true and something different if false: we must choose one path or the other. These are typically seen in Python as **if/else** statements.

Multi-way decisions are those where more than two outcomes are possible, and we want the program to choose a certain path from those variety of options. This is where the **elif** operation in Python comes in handy, though ‘if’ and ‘else’ also are helpful (and required!). We’ll get into examples, but think about the structure of some of the programs you have developed (or want to develop), and how these types of decisions may apply. That will guide which of the statements and methods to use. Some rules of thumb on these operations are shown below:

- If you have two choices: use ‘if’ and ‘else’
- ‘Else’ is sort of a “catch-all” for when condition is **not met**, and must be preceded by an ‘if’ test. **Else** can only be called **once**
- Else if, or -> **elif**
    - Best to use when more than two options/choices
    - Avoids excessive indentation and adds specificity to certain cases, and avoids having to create multiple **if-else** statements
    - Can check multiple expressions for **true** and executes a block of code once one of the conditions evaluates to **true**
    - Can call as many **elif** statements as possible

4) Working with Conditional Statements in Simple Flow Control

Review lines 31-40 and take a look at the program structure, where four outcomes are possible (so, this is squarely a multi-way decision). Note the order in which statements appear, the syntax, and the use of **if/elif/else** throughout. First, the program asks a user to provide their favorite number. Then, based on that entry, a series of conditional statements are used to determine the program's response, and only one response is produced.

Notice that the 'if' statement comes first, and that the 'else' statement comes last. That must be maintained, even if just for simple two-way decisions like in Lines 57-61 below.

If you have **elif** statements, as we do on Lines 35-38, they go in between the **if** and **else** statements.

Notice, too, that colons (:) must follow the conditional statement, followed by an indentation on the next line, where a response is conditionally provided.

Notice, too, that this program converts the input (x) to float data before conditional evaluation. **Run lines 32-40** numerous times, trying various numbers when prompted to see how this kind of flow control works. Think about how this relates to program structure, too. For example, a value of 6 would seem to be logically met by statements that consider whether or not that value is less than 7 **and also** 9, which are both set by **elif** statements (separately on Line 35 and Line 37), but we see that only the print statement from x conditionally being less than 7 is produced. That's what makes **elif** statements so helpful: instead of having to structure these statements in such a way that said (in words) that you wanted a value less than 7 AND greater than 5 to do something, and then another saying you wanted a value less than 9 and greater than 7 to do something else, **elif** saves the need to spell out these statements with such precision.

Activity: Try setting up your own multi-way decision and run various inputs through that program. Feel free to use the provided code as a template. Try developing your own series of outcomes where a user enters yes, no, or maybe as answers to a prompt.

5) Conditional Statements in Definite Loops: an Example

Review Lines 52-62, which return to the example raised near the end of the previous module on looping. This section of code, if you recall, does the following:

- Reads a string object that includes 17 Nevada county and county-equivalent names, separated by 16 comma characters. (Line 52)
- Splits the string object into 17 unique objects, and does so by **splitting** the string object every time it encounters a comma (,) character, and stores those 17 items in a list. (Line 53)
- Sorts those 17 items alphabetically (Line 54)
- Initiates a definite loop that iterates through each item in the now-sorted list object in order (Line 55)
    - At each iteration, it finds that first list object, then:
    - Removes any leading or trailing blank spaces using the **strip** method (this is allowable since each object in the list is itself a string object)

Now, in the previous module, I raised the quirk that Carson City is not a county in the same way that Washoe County is, and the previous definite loop in that module appended the string object "County"

after reading each string object in the list to produce a print statement. I asked you to consider some ways to address that. Conditional statements will help us here.

Line 58 initiates a simple two-way conditional statement embedded within the definite loop. Basically, it asks, in plain words:

"Hey program, can you **find** (remember that method from the string module?) the following anywhere in that string object that matches 'City'? Yes or no. If not, print the string object followed by 'County". If you do, **only** print the string object."

Since this is a two-way decision, an if/else conditional statement is appropriate. This time, though, I'm setting the 'if' statement in accordance with what I think the most likely outcomes will be: that 'City' won't be found except for one case. I only know this because I know the names of the 17 items in the list. Setting an if statement to execute if false (== -1), and then conditionally execute accordingly (Line 58-59), is perfectly valid.

Run Lines 52-62 together to see the full output, which should look something like this.

```
In [69]: NVcounties = " Washoe, Clark, Lyon, Carson City, Pershing, Douglas, White Pine, Nye, Humboldt, Lincoln, Esmeralda,
Mineral, Elko, Storey, Eureka, Churchill, Lander"
    ...: NVList = NVcounties.split(",")
    ...: NVList.sort()
    ...: for i in NVList:
    ...:     x = i.strip()
    ...:         # Recall the previous module, regarding the Carson City issue
    ...:     if i.find('City')== -1:
    ...:         print (x, 'County')
    ...:     else:
    ...:         print (x)
    ...: print ("That's Nevada!")
Carson City
Churchill County
Clark County
Douglas County
Elko County
Esmeralda County
Eureka County
Humboldt County
Lander County
Lincoln County
Lyon County
Mineral County
Nye County
Pershing County
Storey County
Washoe County
White Pine County
That's Nevada!
```

To geographers like me, this kind of "data quirk" is important. I imagine that many of you encounter similar issues in data and analysis in your fields too. Conditional statements offer a way for certain program pathways to be constructed based on situations like this.

It's also important to note that there are plenty of other valid ways that this issue could be addressed: this is merely one.

Activity: What other possible solutions could have been used here? Think about and experiment with some alternate constructions. Build another definite loop with a conditional statement to produce an output where the name of Carson City is reported correctly, and so are the other 16 counties.

This completes the structured part of Module 6. Feel free to experiment some more with setting variables and values, working with print statements, or anything else you are curious about.