Module 3 – Strings

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: "3_Strings.py", and select it to open the script file.

**3) Introduction to String data**.

Before we begin: I think examples that people put together for efforts like this inevitably include some reflection of that person's research efforts, hobbies, etc. Hiking up/climbing up mountains squarely falls under the 'hobby' category (when I can), and colors a bit of what you'll see below. Apologies if this is an unfamiliar topic – but honestly, any chosen example probably would be to someone. It's not at all important to the conceptual material though.

The first thing to know about string (text) data in Python – they are immutable (see code comments in Lines 7-8). In other words, once we set a variable to hold string data, we can't change its contents.

Line 11 sets a variable **mountain** to hold string data. **Run Line 11**, and view the output. Notice the single quotes surrounding the word 'Baboquivari', which is the name of a rather striking desert peak southwest of Tucson, Arizona.

The next little bit of code (**Run Lines 14-16**) offer two things. One: an example of how to import and use the **webbroswer** module. Two: images of the mountain referenced above, if interested. If not, then just make a note of how the webbroswer.open() command works for personal future reference. Or just look at the pictures and move on.

Next, I offer an example of how to access another module – and a subcomponent of another module – on Lines 22-23. Run Lines 22-23. Notice this time I don't use the "import (module) as… (shorthand for use in namespace)" formatting this time. That's not required, but does help with style/flow/simplicity. It is up to you which to implement in your own work, though.

Importing these two modules allows us to parse html documents. That could be of use in some of your own work, too, though of course always be careful when accessing information from web pages like this. In this case, I know one of the site administrators of peakbagger.com, which is what we'll be accessing in this example. Specifically, we'll be scraping the elevation and name of the mountain in question.

Knowledge of the particulars of these modules is not necessary at this point, but feel free to explore more here: https://docs.python.org/3/library/urllib.request.html, and here: https://www.crummy.com/software/BeautifulSoup/bs4/doc/.

I would recommend you **run lines 22-26 as one block**. Explore the outputs of entering either the "peak_page" or "page" variables in the console.

Then, **run Lines 28-30** as one block, and as before, explore the outputs of each variable in the console

**4) String operations**

Now, Run **lines 32 and 33**, then **run lines 35 and 36 afterwards**. Compare the output of the print statements to what you saw as outputs from the **find_elev** and **find_name** variable outputs. The **.strip() method**, a built-in method that can be applied to any string data in Python, is useful in "cleaning up" string outputs, by removing spaces or formatting away from the main text.

Next, **run Lines 41-42** and view the output. Line 41 modifies the **NameString** output and saves it into the **notitsname** variable initiated at the beginning of this line. Recall that strings are immutable, so changing the string data associated with **NameString** here isn't an option, nor is it is even desired. The modification, though, uses **index values** to take a subset of that string data and write it to a new file.

A number of Python data types work using an index that starts first at position 0, continuing through the full length of the data type in question

So, **NameString's** index scheme would look like this: [0]'B', [1]'a', [2]'b', [3],'o', [4],'q', etc.

But, Line 41 uses the [0:4] index range, and the output of NameString[0:4] is 'Babo', not 'Baboq'. In other words, the values at index position 0,1,2, and 3 (but not 4) are extracted. In Python, when you enter index range values like this, the item at the first position will be included in the output, but the second position acts as the upper boundary that is not itself included.

To see all of the allowable methods associated with string data in Python, **run Line 45** and scan through some of them to see what could be of some help to you.

**Run Lines 48 and 49 together.** The **find()** method in Python leverages this index structure to tell you where a subset of a string is in a larger object. Notice searching for 'qu' returns a value in the console output that corresponds to the **index value** of the first position identified in the search. When you search for the 'bq' string object (**Run Lines 51-52**), you'll notice that a value of '-1' is returned, which means that was not encountered. This even happens if you modify the search to include the capital letter 'B'. Even though 'B' and 'q' can be found in the string object, they are not found in consecutive order like that, so a -1 value is returned again. Searching independently for those letters would net their index positions, though.

Next, **run Line 55**, which we will use an example for using the .strip(), .lstrip(), and .rstrip() methods in lines 56-58. Notice the string object on Line 55 contains a number of space values (which are completely valid to store with string objects) on either side of the words Baboquivari Peak in the example. Also notice that using print () statements allows you to see what using each will do to the object on Line 55.

**Run Lines 56-58 together** and compare the outputs. Then, enter **bestmountain** in the input of the console. Notice no permanent changes. Pick one to of the modified ones from lines 56-58 (e.g., enter **bestmountain.lstrip()** ), and then enter **bestmountain** again as the next input in the console. While at first glance it may have appeared that using one of the .strip() methods committed a change to the string object with **bestmountain**, you can see that entering the variable name again shows this is not the case. If you want to make that change permanent, enter it in conjunction with a new variable name, such as:

```
In [61]: fixed = bestmountain.lstrip()

In [62]: fixed
Out[62]: 'Baboquivari Peak  '
```

**5) Escape Characters**

Lines 61-67 provide some templates and documentation on how to incorporate escape characters in string outputs. Note that the string object in Line 63 is initiated through the use of double quotes. One of the quirks of Python is that either single or double quotes can be used to set a string object, just so long as the use is consistent for that object. For example, x = "why' would cause a syntax error, but x = "why" or x = 'why' are both fair game.

Sometimes, though, you may wish to include quotes in the string object output in the same format as the quotes you used to initiate the string object. You can use double quotes around the edge of the string object, and use single quotes to return quotes of some form in the output (for example):

```
In [67]: print("Baboquivari is considered the 'hardest' mountain to climb in the state")
Baboquivari is considered the 'hardest' mountain to climb in the state
```

The syntax presented in your code in Line 63 preserves the double quotes in the output, though. **Run Line 63 to see**

**Run Line 65**, which adds a wrinkle by breaking up the long string object into two lines in the output, and then **run Line 67**, which builds on this by adding a tab. Similarly, you can access these characters that denote new lines or tabs when reading some file types, which will come later in Module 10.

**6) Format Flags/String Formatting Operators**

Line 70 contains a number of new characters that are useful when reporting different data types in print output statements. These help with formatting in accordance to Python's rules regarding certain data types. %s will ensure reporting of string data, %d as a signed integer decimal, and %f as floating decimal point data. There are of course others: https://python-reference.readthedocs.io/en/latest/docs/str/formatting.html

You can add precision reporting through the use %.1f, for example. Note the consistent use of the (%) – or **mod** – character throughout. Note also the mod character's position between the end of the string object and the values, separated by spaces. That structure is required and a syntax error will always be thrown if things are not presented in that order.

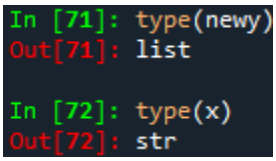
**Run Line 70**, and play around with some of the formatting. For example, what would happen if you changed the %.1f to %.2f?

**Run Lines 72-75** next and take a look at the output, which uses variables passed into the statement this time

These operators are very useful for reporting purposes when reading/reporting raw data from various sources.

**7) Using String Methods to create New Data Types**

One nice thing about Python is that you can use string methods to create entirely new data types. **Running Lines 78 through 80** show an interesting output in the print statement that looks quite different than we've seen earlier in this module. You can check data types of objects at any point by using the **type() method** in the console, as below:




Running this command will report back the particular data type of the object. In this case, the 'newy' variable holds a list object, while the 'x' variable I set earlier to 'why' is a string object. We will review lists later on in Module 8, but be aware that some of these function and commands produce certain data types by default.

This completes the structured part of Module 3. Feel free to experiment some more with setting variables and values, working with print statements, or anything else you are curious about.