Notes - Tuesday 12/22/20

Day 3 of Python Bootcamp LFG.

Dictionaries

Dictionaries are unordered mappings for storing objects. Mappings are collections of objects that are stored using a key. A sequence stores objects using the position. Mappings won't have an order and can't be sorted. They can't be indexed or sliced.

There are dictionaries that can be ordered called `ordereddict` which will be covered later.

Lists store objects in an ordered sequence. Dictionaries store objects using a key value pairing. The value is a Python object. This value pairing allows the user to grab the object without knowing the index location. Retrieving something from a dictionary can be faster than getting it from a list.

The syntax is:

```{'key1':'value1','key2':'value2'}```

Note the single quote ' is used.

To retreive something from the dictionary, you'd type in the dictionary name then pass the value you want. The syntax is:

`urdictionary['urkeyhere']`

Dictionaries can hold (among other things):

- Lists
- Floating point numbers
- Strings
- Integers
- Other dictionaries

I can call dictionaries within dictionaries or list index positions within dictionaries. I would first retreive the object as seen above with the basic command `urdictionary['urkeyhere']` but then I'd addon the second object or command I'm trying to pass. For example, if I wanted to retrive an index from the dictionary then call out index position 2, my command would look like:

`urdictionary['urkeyhere'][2]`
```

If I want to add to an existing dictionary, I would simply call the dictionary I previously defined, then add a new key. The syntax would be:

`urdictionary['urnewkeyhere'] = urnewobjecthere`

This command could be used to create a new object and key in the dictionary or modify an existing key with a new object.

Dictionaries have functions. Some examples are:

- Keys - Shows all the keys in the dictionary
- Values - Shows all the values aka objects in the dictionary
- Items - Shows everything in the dictionary but as a tuple ("Too-pul") which will be discussed in a later topic.

The syntax would be: `dictionary.function()`

Remember like before, remember to include () or else you're asking Python to tell you what the function could do. Instead of actually executing the function.

Dictionary Manipulation Exercise

Let's do a basic example of manipulating a dictionary that has a list of letters

`list = [a,b,c,d]`

Let's say I want to retreive "c" then save it as a variable and capitalize it.

The long way looks like:

```
1 alphabet_dict = {"k1":['a','b','c','d']}
2 alphabet_dict
3 # Set up the dictionary and called it to confirm it's good.
4 alphabet_list = alphabet_dict['k1']
5 alphabet_list
6 # Set the variable alphabet_list to be the k1 key of the dictionary then called it to confirm it's good.
7 letter_c = alphabet_list[2]
8 letter_c
9 # Set the variable to pull "c" from the variable alphabet_list
10 letter_c.upper()
```

```
11 # Used function upper to capitalize "C"
```

Or you could stack calls on top of each other to make it easier. Python allows this flexibility. So easier:

```
1 alphabet_dict['k1'][2].upper()
2 # Did work from above lines in one step:
3 # First we re-use the alphabet_dict dictionary that was defined
4 # Second we call the key 'k1'
5 # Third we call index position 2
6 # Last we use function upper to capitalize that value
```

Tuples

Say "too-puls"

`tuple = (1,2,3,4,5)`

Tuples are like lists except they can't be changed ("immutable"). Once an object is inside a tuple, it can't be changed. The features and manipulation of a tuple is like a list but not as flexible. But tuples have count and index functions.

Tuples are often used for objects that are passed between things and it keeps your data safe and not over-written.

Sets

Sets are unordered collections of unique elements. There's only one representative of that object. So "a" or 2 can only be in the set once.

Syntax is:

```
1 ursethere = set()
2 ursethere.add(urobjecthere)
```

We'll learn more about sets later in the course. But a use for sets right now is you could take a list of numbers and get a unique list of the values.

So if I have a list of random numbers, casting the list into a set will show me the unique numbers in that set.

```
1 list_num = [4,4,4,4,4,3,3,3,3,6,7,7,7,8,89,9]
2 set(list_num)
3 {3, 4, 6, 7, 8, 9, 89}
4 # Line 3 is the output of Line 2
```

Booleans

Booleans are operators that allow you to convey True (aka integer 1), False (aka integer 0), or None. statements. They're really helpful when dealing with control flow and logic. Make sure that your boolean is Capitalized. Otherwise Python thinks you're creating a variable.

For example it plays a part in comparison operators like > and ==. If I ask Python if `1 > 2` (Is one greater than 2?) or `1 == 1` (Is one equal to 1?), it will output a boolean answer.

Files - Simple Input/Output with Files in Jupyter Notebooks

The following notes all only work in Jupyter Notebooks which came from iPython notebooks.

In Jupyter Notebooks, in a cell I can directly create a text file. The command is `%%writefile`. The syntax is `%%writefile urfilename.txt```. Then on the second line after `%%writefile``` I can start writing the actual text I want to add to the file.

I could then create a variable to open the file. If I get a `Errno 2` when I try to open the file, I spelled the file name wrong or I'm pulling from the wrong directory. The syntax would be `urvariable4filename = open('urfilename')`.

To check your current directory, type `pwd` into a Jupyter notebook cell. Just like how you would check the current directory in the terminal.

If I opened the file with a variable, there's functions I can do. It's similar to the functions we previously covered before with the `urvariable4filename.function()`. And of course, remember the (). These functions include:

- Read - it will list out the entire contents of the file
  - \n means new line which is an escape sequence which was previously covered
  - Running read will move the cursor to the end. To reset the cursor, you need to use Seek.
- Seek - this is how you reset the cursor in read. Syntax is `urvariable4filename.seek(0)`

- Read Lines - it will read through the lines in a more legible manner and you can perform other things to this readlines function. Syntax is `urvariable4filename.readlines()`
- Open - This just opens up the file that you want to work on. Simply using this open method will require that you close the file.
- Close - Remember to close out your files so that the computer doesn't think Python is still using the file.

To open files in another location, you have to change the file path when running open.

- For Linux the syntax is: `myfile = open("/Users/YouUserName/Folder/myfile.txt")`
- For Windows the syntax is: `myfile = open("C:\\Users\\YourUserName\\Home\\Folder\\myfile.txt")`

To open files AND then set modes and do something with the files, I'll want to use `with open` so the syntax would be:

```
with open ('urvariable4filename',mode='urmodehere') as urvariablehere:
    whatyouwannadohere(urvariablenamehere.function())
    # note the indentation
```

You could use this method to:

- Read what's in the file using in the indented line `print(urvariablenamehere.read())`
- Append the file or add on to it using in the indented line `urvariablenamehere.write('urtexthere')`
- Write the file using

%%writefile Modes

- Read only = mode='r'
- Write only - will overwrite or create new files = mode='w'
- Append only - add onto existing files = mode='a'
- Reading and Writing = mode='r+'
- Wrtiring and Reading - will overwrite or create new files = mode'w+'

Revision #2
Created 22 December 2020 21:49:39 by cba88
Updated 23 December 2020 01:02:08 by cba88