420-LCU-05 Programming in Python - Lab Exercise 9

March 29, 2022

Goals for this lab:

- Experiment with dictionaries.
- Read data from a file store in a nested dictionary.
- This lab is designed as a tutorial that walks you through many of the dictionary basics covered in Lecture 9.

Your submission for this lab should be in 1 python file. Include the identification section at the top of your file. Type all the code in your Python file and run your code after each part. Whenever there is a question, type the answer in a comment. Indicate the number for each question. No need to copy answers from IDLE session. All code can be included in the same `.py` file.

1 Basics

As we started discussing in class, a dictionary (abbreviated `dict` in Python) is a *data structure* that is similar to a list, but more flexible, in that any immutable object (string, integer or float) can be used as a *key* which is paired with an arbitrary *value*.

1. First let's create a dictionary. Type the following code into the Python shell:

```
colors = {'red':10, 'blue':20, 'green':30 }
print(colors)
```

   What is printed? Is the order the same as what you typed? Remember that the ordering of elements in a dictionary is *arbitrary*.

2. Now we'll add some items to the dictionary:

```
colors['yellow'] = 40
colors['orange'] = 50
print(colors)
```

   Note the order of elements.

3. We mentioned that a dictionary is iterable. To see how this works, first try this:

```
for key in colors:
    print(key)
```

   This will simply print each key in the dictionary.

4. If we want the values as well as the keys, we have several options. First, you can always retrieve the values by indexing the dictionary with the key:

```
for key in colors:
    print(key, colors[key]) # index dictionary by key
```

   Another option is to use the `items()` method, which returns an iterable list of *key, value* pairs:

```
for key, val in colors.items():
    print(key, val)
```

   If you *only* care about examining the values, there is another option, the `values()` method:

```
for val in colors.values():
    print(val)
```

   this can be useful, for example, if you only care about the values.

5. Given the above, write a one-line expression that would compute the sum of all of the values on the dictionary.

6. As with lists, indexing allows you to modify values. Write the code to add one to each value in the `colors` dictionary.

7. If need to work with the keys in a predictable sequence, one option we have is to sort them! Try this:

```
for key in sorted(colors):
    print(key)
```

8. Attempting to read a value using a nonexistent key will trigger an exception. Try this and note what is printed:

```
print(colors['pink'])
```

   Now, you can comment this line and continue.

9. Luckily, we have other options for accessing items in a dictionary. One of the most useful is the `get()` method. Its first argument (after `self`, of course) is the key to look up. Try the following:

```
print(colors.get('red'))
print(colors.get('pink'))
```

   Notice what happens, and copy the result into your report.

10. `get` can take another, optional argument, that is used to replace missing values:

```
print(colors.get('red', -1))
print(colors.get('pink', -1))
```

    What happens this time?

11. Since dictionaries are mutable, they support both the `clear()` and `copy()` methods. Try the following:

```
x = dict()
y = x
z = x.copy()
print(x is y, x is z)
print(x == y, x == z)
z[2.0] = 'two'
y[3.14159] = 'pi'
y[2.71828] = 'e'
print(x, z)
y.clear()
print(x, z)
```

    What happens to `x`? To `z`? Why?

12. You can construct dictionaries in several ways. If we start from a list of keys, we can use the `fromkeys` method. This method is unusual in that it does not use or modify a dictionary, in fact, it only creates a new one. Therefore, it can be called through the dictionary class:

```
from string import ascii_lowercase
letters = dict.fromkeys(ascii_lowercase, 0)
print(letters)
```

13. When we have a list of keys and a list of values, we can construct a dictionary by combining both using `zip()`. Here's an example using Montreal's population from the last four censuses:

```
k = [2001, 2006, 2011, 2016] #years of censuses
v = [1039534, 1620693, 1649519, 1704694] #population
d = dict(zip(k, v))
print(d)
```

    Notice that here we have an example of 'sparse' data. We're associating a year (the key) with the population (the value). It's a bit like a list, but data is stored only for certain indices.

14. The keys in a dictionary must be unique, whereas values need not be unique. However, in many cases both will be unique. If so, we can *invert* the dictionary, so that keys become values and vice-versa.
    Given the dictionary d created in the previous step (that's the only input we have). See if you can figure out a single-line expression that will invert the population dictionary *d*. Hint: create another dictionary `inv_d`.

2 Application 1 - Letter frequencies

In this part, you will write a function to get the letter frequencies in the file `alice.txt` that was provided with Lecture 10 examples. This file is structured in lines. However, for the purpose of this exercise, it can be read as one long stream of characters. Use the method read() to read the file.

- define the function: `def LetterFrequencies(fx)`. The function takes a text file as parameter and returns a dictionary that gives the count of each letter (capitals and small) in the text file.
- Your short function counts the frequencies of the letters of the alphabet as they appear in the book.
- The character frequencies will be represented in a dictionary, where each key will be a letter ('a'..'z','A'..'Z'), and the value will be the number of times that each letter was found in the text file.
- Your code differentiates upper case and lower case letters and ignores all nonalphabet characters.
- After calling the function, Write code to print the content of the result dictionary in alphabetical order. Print frequencies of all lower case letters followed by upper case letters. The resulting dictionary includes the letters found in the text only. For Example:

```
x="alice.txt"
dx=LetterFrequencies(x)
# Insert code to print the dictionary dx in required order.
```