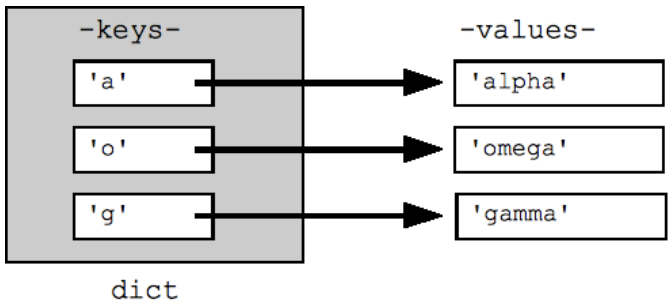
CMSC 201 Fall 2016
Lab 11 – Tuples and Dictionaries

Assignment: Lab 11 – Tuples and Dictionaries
Due Date: During discussion, November 14th through 17th
Value: 10 points

Part 1: Dictionaries

A very useful data type built into Python is the ***dictionary***. Dictionaries are sometimes found in other languages as "associative memories" or "associative arrays." Dictionaries are data structures that map a key to a value. So, in the example below, we have a dictionary that maps the key '**`a`**' to the value '**`alpha`**'; the key '**`o`**' to the value '**`omega`**'; and the key '**`g`**' to the value '**`gamma`**'.



(example from https://developers.google.com/edu/python/dict-files)

We can create this dictionary with this line of code:

```
greek = {'a': 'alpha', 'o': 'omega', 'g': 'gamma'}
```

Dictionaries may look a lot like lists, but there are a few key differences:

1. A dictionary uses curly braces instead of square brackets
2. A dictionary is made up of (key, value) pairs
3. The key and value are separated by a colon (`:`)
4. The (key, value) pairs are separated by a comma (`,`)
5. The keys must be unique (just like the indexes of a list are unique)

Lists are indexed by ***order***, which we see as a range of numbers. Dictionaries are indexed by ***association***, or their key values. Keys can be any immutable type, and every key in a dictionary must be unique. Strings, floats, and integers are common choices for a key.

Part 2: Dictionary Functions

We can start by looking at how we could create a simple dictionary. Let's create a new dictionary called **animals**.

```
animals = {"Clifford" : "dog",    "Hedwig" : "owl",
           "George"   : "monkey", "Kha"    : "snake",
           "Laika"    : "dog"}
```

In this dictionary, we have mapped famous animals, using their name as the key, and their species as the value. Since there may be multiple animals of the same species (*e.g.*, Clifford and Laika are both dogs), it makes sense to use the unique value (the name) as the key.

Using a dictionary, we can perform a number of operations. The examples below use the **animals** dictionary defined above.

A. **Iterate** through the dictionary:

```
for name in animals:
        print(name, "is a famous", animals[name])
```

OUTPUT:

```
Laika is a famous dog
George is a famous monkey
Clifford is a famous dog
Kha is a famous snake
Hedwig is a famous owl
```

B. **Access** a specific entry:

```
print("Kha is the", animals["Kha"], \
      "from 'The Jungle Book'")
```

OUTPUT:

```
Kha is the snake from 'The Jungle Book'
```

C. **Add** something to the dictionary:

```
animals["Punxsutawney Phil"] = "groundhog"
```

D. **Updating** the value of something in the dictionary:

```
animals["Hedwig"] = "snowy owl"
```

E. **Deleting** something from the dictionary:

```
# Laika was a Soviet space dog, the first
# animal to orbit the Earth; she did not
# survive more than a few hours in space :(
del animals["Laika"]
```

F. **Checking if a key is present** in the dictionary:

```
"Laika" in animals
# this will return False, as Laika's no longer in
the dictionary

"Clifford" in animals
# this will return True
```

G. Dictionaries also have methods that enable some additional functionality. In addition to the commands and examples above, here are some of the more helpful methods we can use.

All of these return a "view" by default, so we must cast them to a list to use them.

a. `list(animals.items() )`

i. Returns a list of **animals**'s (key, value) tuple pairs

```
[('Punxsutawney Phil', 'groundhog'),
 ('Hedwig', 'snowy owl'), ('George', 'monkey'),
 ('Clifford', 'dog'), ('Kha', 'snake')]
```

b. `list(animals.values() )`

i. Returns a list of dictionary **animals**'s values

```
['groundhog', 'snowy owl', 'monkey', 'dog',
 'snake']
```

c. `list(animals.keys() )`

i. Returns a list of dictionary **animals**'s keys

```
['Punxsutawney Phil', 'Hedwig', 'George',
 'Clifford', 'Kha']
```

Part 3: State Capitals

After logging into GL, navigate to the **Labs** folder inside your **201** folder. Create a folder there called **lab11**, and go inside the newly created **lab11** directory.

```
linux2[1]% cd 201
linux2[2]% cd Labs
linux2[3]% pwd
/afs/umbc.edu/users/k/k/k38/home/201/Labs
linux2[4]% mkdir lab11
linux2[5]% cd lab11
linux2[6]% pwd
/afs/umbc.edu/users/k/k/k38/home/201/Labs/lab11
linux2[7]%
```

Once you're in the folder, you will need to copy the starter file from my public directory. Type (all on one line – don't forget the rest of the command!):
cp /afs/umbc.edu/users/k/k/k38/pub/cs201/given_capitals.py capitals.py

To open the file for editing, type
emacs capitals.py
and hit enter.

The first thing you should do in your file is complete the comment header block, filling in your name, section number, email, and the date.

Then you can start completing the code, following the comments in the file and the instructions on the following page.

For Lab 11, you will be implementing an application that can tell the user the capital of any state. First things first, download the file containing the states and their capitals by running this command inside your lab11 folder:
`cp /afs/umbc.edu/users/k/k/k38/pub/cs201/stateCaps.txt .`

The file contains the states and their respective capitals.

In order to complete your lab, you will need to implement a program that does the following tasks:

1. Read in the **`stateCaps.txt`** file (you can hardcode the filename). *(HINT: You will need to take a look at how the file is formatted to be able to extract the parts you need!)*

2. Write a function to store the data from the file in a dictionary, where the state names are the key, and the capital is the value. *(HINT: Your function should return the dictionary. The* **`split()`** *function should prove useful in getting the data out of the file.)*

3. Ask the user to input a state – check if the word appears in the dictionary's keys. *(HINT: Use the* **`in`** *keyword discussed earlier.)*
    a. If the word doesn't appear in the dictionary, tell the user that.
    b. If the word does appear in the dictionary, return the translation.

4. Allow the user to keep looking up state capitals as long as they like; if they type "**`exit`**", the program should finish.

5. If the user types "list" it lists each of the state names (*HINT: Use the* **`keys()`** *method to list the keys.*)

You can find a sample run of the program on the next page.

Here is a sample run of the program, with the user input in blue. Note that your "list" will likely not be in the same order, since the dictionary values will be in a different order.

```
bash-4.1$ python capitals.py
Welcome to the State Capital Lookup System

        Please enter the state you want the capital of.
        (Use 'list' for choices, or 'exit' to quit): list
Your choices of states are:
Oregon
West Virginia
Iowa
Hawaii
Utah
Colorado
Rhode Island
Connecticut
Alaska
[etc etc -- your program should print out all 50 states]

        Please enter the state you want the capital of.
        (Use 'list' for choices, or 'exit' to quit): mayrland
Sorry, mayrland is not a state.

        Please enter the state you want the capital of.
        (Use 'list' for choices, or 'exit' to quit): Maryland
The capital of Maryland is Annapolis

        Please enter the state you want the capital of.
        (Use 'list' for choices, or 'exit' to quit): Minnesota
The capital of Minnesota is Saint Paul

        Please enter the state you want the capital of.
        (Use 'list' for choices, or 'exit' to quit): EXIT
Sorry, EXIT is not a state.

        Please enter the state you want the capital of.
        (Use 'list' for choices, or 'exit' to quit): exit
Thank you for using the State Capital Lookup System!
```

Part 4: Completing Your Lab

To test your program, first enable Python 3, then run **`capitals.py`**. Try asking it to show the capitals of different states than the ones shown in the sample run above.

Since this is an in-person lab, you do not need to use the **`submit`** command to complete your lab. Instead, raise your hand to let your TA know that you are finished.

They will come over and check your work – they may ask you to run your program for them, and they may also want to see your code. Once they've checked your work, they'll give you a score for the lab, and you are free to leave.

IMPORTANT: If you leave the lab without the TA checking your work, you will receive a **zero** for this week's lab. Make sure you have been given a grade before you leave!