CMSC 201 Spring 2018

Lab 13 – Dictionaries

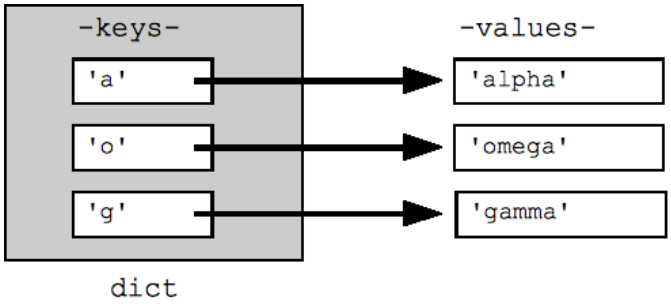
Assignment: Lab 13 – Dictionaries
Due Date: **During discussion**, April 30th through May 3rd
Value: 10 points (8 points during lab, 2 points for Pre Lab quiz)

This week's lab will give you practice with using dictionaries.

(Having concepts explained in a new and different way can often lead to a better understanding, so make sure to pay attention as your TA explains.)

Part 1A: Review – Dictionaries

A very useful data type available in 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**'.



(Image 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 keys must be unique (just like the indexes of a list are unique)
    a. The keys can only be immutable data types

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 1B: Review – 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:

```
keys = list( animals.keys() )
for i in range(len(keys)):
    print(keys[i], "is a famous", \
              animals[ keys[i] ])
```

OUTPUT:

```
Laika is a famous dog
George is a famous monkey
```

(and so on)

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. **Safely access** a specific entry:

```
print("Kha is the", animals.get("Kha"), \
      "from 'The Jungle Book'")
# if there was no "Kha" key, this would simply
# print out None, rather than crashing
```

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

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

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

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

F. **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"]
```

G. **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
```

H. 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.

These both return a "view" by default, so we must cast them to a list to use them.

a. **`list( animals.values() )`**
   Returns a list of the values in dictionary **`animals`**

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

b. **`list( animals.keys() )`**
   Returns a list of the keys in dictionary **`animals`**

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

Part 1C: New Material – Lists as Values

Although we didn't discuss it in detail during class, it is possible to use lists as the value in a (key, value) pair, and to update the list as new items are entered that belong with that key. In order to do so, though, we must pay attention to and handle two different scenarios:

1. The key does not yet exist in the dictionary
    a. We must create a new key and start the list from scratch
2. The key already exists in the dictionary
    a. We must append to the existing list, without overwriting it

To accomplish these, the first thing to do is use the `.get()` function to access a key's value. The `.get()` function is safer than square brackets, because if the key does not exist it will return `None` (where square brackets would cause an error).

If we see that the key already exists, we can create a key:value pair with a single element list for the value. If it does already exist, we simply append to the existing list. For example, code to accomplish that might look like the following.

```
# if the key doesn't already exist in the dictionary
if myDict.get(theKey) == None:
    myDict[theKey] = [newValue]
else:
    myDict[theKey].append(newValue)
```

And when we want to access something in the list of values, we'll need to first index into the dictionary (using the key) and then into the list (using regular indexes).

```
# print all the elements of a key's value list
valueList = myDict[knownKey]
for i in range(len(valueList)):
    print(valueList[i])
```

Part 2: Exercise

In this lab, you'll be downloading the start of a program that uses a dictionary to store information about Pokémon and the list of moves each of them has.

Tasks

Starting:

- ☐ Copy the **`given_pokeQuery.py`** file from Dr. Gibson's **`pub`** directory
  - ☐ It should have been renamed to be **`pokeQuery.py`**
- ☐ Copy the **`pokedex.txt`** file from Dr. Gibson's **`pub`** directory

Programming:

- ☐ Open the file and examine the code
- ☐ Complete each of the following functions:
  - ☐ **`createPokeDex()`**, which should store the data in a dictionary
  - ☐ **`checkMoveExists()`**, which should check to see if each Pokémon has a specific move
  - ☐ **`main()`**, which should call the **`checkMoveExists()`** function, and handle the result correctly

General:

- ☐ Run and test your code as needed
- ☐ Show your work to your TA

If you get stuck, don't forget what you learned in Lab 09!

Remember, a "debug statement" is a **`print()`** statement that gives you more information on what exactly is going on. Placing a **`print()`** statement inside your code, can show you what is going on in the "background" of your program. Each time the code is run, the information in your debug statement will be printed to the screen, allowing you to trace what is happening with your program.

For example, you might want to see what the current dictionary looks like, or what the recursive function is being given as parameters.

Part 3A: Downloading the File

First, create the **`lab13`** folder using the **`mkdir`** command – the folder needs to be inside your **`Labs`** folder as well.

Next, copy a file into your **`lab13`** folder using the **`cp`** command. (The command should be all on one line.)

```
cp /afs/umbc.edu/users/k/k/k38/pub/cs201/given_pokeQuery.py
pokeQuery.py
```

This will copy the file **`given_pokeQuery.py`** from Dr. Gibson's public folder into your current folder, and will change the file's name to **`pokeQuery.py`** instead.

You will **also** need to copy the **`pokedex.txt`** file, in which all of the information about the different Pokémon is stored. Don't forget the period at the end!

```
cp /afs/umbc.edu/users/k/k/k38/pub/cs201/pokedex.txt .
```

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

Part 3B: Completing the Program

For Lab 13, you will be implementing an application that will allow the user to search for all Pokémon inside of their Storage System that have a certain move (which will be entered in).

The file contains every Pokémon (151 Generation One Pokémon) that the user owns. You should open up the text file and examine its contents and how it is formatted.

Follow the instructions inside the **`pokeQuery.py`** file to complete the lab.

However, here are some hints:

- It is important to look at the layout of the text file **AND** read the function headers to understand what to do.
- The name of each Pokémon should be used as a key.
- The move set should be turned into a list and then assigned as a value for that Pokémon.

(See the next page for sample output.)

Here is some sample output of the program, with the user input in **blue**.

```
bash-4.1$ python pokeQuery.py
   Hello, and welcome to your Pokemon Storage System!
   Would you like to search your box for all Pokemon
   that possess a specified move?

Please enter a move (type 'STOP' to finish): Whine
You do not have any Pokemon that know Whine
Please enter a move (type 'STOP' to finish): Bark
You do not have any Pokemon that know Bark
Please enter a move (type 'STOP' to finish): Boof
You do not have any Pokemon that know Boof
Please enter a move (type 'STOP' to finish): Woof
You do not have any Pokemon that know Woof
Please enter a move (type 'STOP' to finish): Howl
You do not have any Pokemon that know Howl
Please enter a move (type 'STOP' to finish): Slobber
You do not have any Pokemon that know Slobber
Please enter a move (type 'STOP' to finish): FireFang
>> You have a Arcanine that knows FireFang
You have 1 Pokemon that know FireFang
Please enter a move (type 'STOP' to finish): Lick
>> You have a Jynx that knows Lick
>> You have a Gengar that knows Lick
>> You have a Ditto that knows Lick
>> You have a Haunter that knows Lick
>> You have a Lickitung that knows Lick
>> You have a Gastly that knows Lick
You have 6 Pokemon that know Lick
Please enter a move (type 'STOP' to finish): STOP
Shutting Down...
```

(See more sample output, showing a longer run, on the following page.)

Here is more sample output of the program, with the user input in **blue**.

```
bash-4.1$ python pokeQuery.py
   Hello, and welcome to your Pokemon Storage System!
   Would you like to search your box for all Pokemon
   that possess a specified move?

Please enter a move (type 'STOP' to finish): Bubble
>> You have a Poliwhirl that knows Bubble
>> You have a Horsea that knows Bubble
>> You have a Kingler that knows Bubble
>> You have a Squirtle that knows Bubble
>> You have a Poliwrath that knows Bubble
>> You have a Poliwag that knows Bubble
>> You have a Krabby that knows Bubble
>> You have a Seadra that knows Bubble
You have 8 Pokemon that know Bubble
Please enter a move (type 'STOP' to finish): Gust
>> You have a Zapdos that knows Gust
>> You have a Pidgey that knows Gust
>> You have a Pidgeotto that knows Gust
>> You have a Spearow that knows Gust
>> You have a Fearow that knows Gust
>> You have a Butterfree that knows Gust
>> You have a Moltres that knows Gust
>> You have a Articuno that knows Gust
You have 8 Pokemon that know Gust
Please enter a move (type 'STOP' to finish): STOP
Shutting Down...
```

Part 4: Completing Your Lab

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.

Tasks

Starting:

- ☐ Copy the **`given_pokeQuery.py`** file from Dr. Gibson's **`pub`** directory
  - ☐ It should have been renamed to be **`pokeQuery.py`**
- ☐ Copy the **`pokedex.txt`** file from Dr. Gibson's **`pub`** directory

Programming:

- ☐ Open the file and examine the code
- ☐ Complete each of the following functions:
  - ☐ **`createPokeDex()`**, which should store the data in a dictionary
  - ☐ **`checkMoveExists()`**, which should check to see if each Pokémon has a specific move
  - ☐ **`main()`**, which should call the **`checkMoveExists()`** function, and handle the result correctly

General:

- ☐ Run and test your code as needed
- ☐ Show your work to your TA

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!