Lesson 4 - Thomas Deneuville

October 5, 2017

1 Set Membership

One major application of sets is to check if an element is inside the set. Use the `in` keyword.

```
In [1]: A = {(1, 1, 1), 'Thomas', 'Ray', 3, -1}

        #   query in set
        B = 'Ray' in A
        print(B)
        C = 'Thomas' in A
        print(C)
        D = 100 in A
        print(D)
```

```
True
True
False
```

The standard syntax is `query in A` where `query` is what you want to look for and `A` is the set that was constructed.

2 Dictionaries

Are very similar to an English dictionary. You look up the word you want to find the definition of - we call this a **key**. And the definition associated with the word/key is called the **value**. A dictionary in Python allows you to store things with key/value pairs.

The process for dictionaries in Python is to store something referenced by a key. When you want to retrieve whatever you stored, you use the key to do it.

To create dictionaries in Python is very simple. You use the `{}` (curly braces) just like sets, but what goes inside is very different.

```
In [2]: B = {"hello": 1, "world": 2, 3: "Ray", (1, 2, 3): "Hihihihihih"}
        # What goes before the colon is the key
        # What goes after the colon is the value

        print(B["hello"]) # Give me the number 1
        print(B[3]) # Gives me Ray
        print(B[(1, 2, 3)]) # Gives me hihihihihih
```

```
1
Ray
Hihihihihih
```

It's probably useful to add more definitions / keys to the dictionary as we go. If the keys are not unique, then it will remember the most current definition associated with that key (i.e. change definitions)

```
In [3]: print(B)
        B["hello"] = [1,2,3,4,4,5]
        print(B)
        # Add definitions
        B["testing"] = range(10)
        print(B)
```

```
{'hello': 1, 3: 'Ray', 'world': 2, (1, 2, 3): 'Hihihihihih'}
{'hello': [1, 2, 3, 4, 4, 5], 3: 'Ray', 'world': 2, (1, 2, 3): 'Hihihihihih'}
{'testing': range(0, 10), 'hello': [1, 2, 3, 4, 4, 5], 3: 'Ray', 'world': 2, (1, 2, 3): 'Hihih
```

We can remove keys and associated values from the dictionary. Use the `del` command. You can only remove keys from the dictionary and their values by association go away. You cannot remove values due to the way Python is set up.

```
In [4]: del B["testing"]
        print(B)
```

```
{'hello': [1, 2, 3, 4, 4, 5], 3: 'Ray', 'world': 2, (1, 2, 3): 'Hihihihihih'}
```

If we try to delete keys that don't exist, it will give us an error

```
In [5]: del B["blah"]
```

```
        ---------------------------------------------------------------------------

        KeyError                                  Traceback (most recent call last)

        <ipython-input-5-31633f599e63> in <module>()
    ----> 1 del B["blah"]


        KeyError: 'blah'
```

2.1 Looping over a dictionary

We can certainly loop over dictionaries. The syntax is simply:

```
In [6]: for i in B: # This will iterate over the KEYS
            print(i)

hello
3
world
(1, 2, 3)
```

If you wanted to loop over the values, you would use the `values()` method that is a part of dictionaries

```
In [7]: for i in B.values():
            print(i)

[1, 2, 3, 4, 4, 5]
Ray
2
Hihihihihih
```

We can provide both the key and value inside the loop by using a method called `items()`:

```
In [8]: for (i, j) in B.items(): #i is the key and j is the value
            print("Key: ", i, ", Value: ", j)

Key:  hello , Value:  [1, 2, 3, 4, 4, 5]
Key:  3 , Value:  Ray
Key:  world , Value:  2
Key:  (1, 2, 3) , Value:  Hihihihihih
```

We can also check to see if there is a key inside a dictionary. Use the `in` syntax just like before:

```
In [9]: print(B)
        print('world' in B) # Gives me true
        print('blah' in B) # Gives false
        print((1,2,3) in B) # Gives true

{'hello': [1, 2, 3, 4, 4, 5], 3: 'Ray', 'world': 2, (1, 2, 3): 'Hihihihihih'}
True
False
True
```

In practice, if we try to obtain a key that doesn't exist in the dictionary, we will get an error

```
In [10]: print(B)

{'hello': [1, 2, 3, 4, 4, 5], 3: 'Ray', 'world': 2, (1, 2, 3): 'Hihihihihih'}


In [11]: print(B["blah"])


        ---------------------------------------------------------------------------

        KeyError                                  Traceback (most recent call last)

        <ipython-input-11-9528f972a1dc> in <module>()
    ----> 1 print(B["blah"])


        KeyError: 'blah'
```

We can assume a default value. That is, if the key doesn't exist, provide a default value instead. Use the `get` method in a dictionary. There are two input parameters. The first parameter is the key you're searching for and the second parameter is the default value in case we don't find the key.

```
In [12]: C = B.get("blah", -1) # This produces the value of -1 because "blah" key doesn't exis
         print(C)
         D = B.get("world", -1) # This produces the value 2 because "world" key does exist
         print(D)

-1
2
```

Let's try and find the key associated with a value.

```
In [13]: q = "Ray"
         print(B)
         # Iterate through each pair of key/value.  Check to see if the value is the one you w
         # print that out
         for (i, j) in B.items():
             if j == q:
                 print("Value found: ", j, ", Corresponding key: ", i)

{'hello': [1, 2, 3, 4, 4, 5], 3: 'Ray', 'world': 2, (1, 2, 3): 'Hihihihihih'}
Value found:  Ray , Corresponding key:  3
```

2.2 Dictionary Comprehensions

We can create dictionaries with comprehensions, very much like lists but make sure you respect the dictionary syntax `key: value`:

```
In [14]: A = [1, 2, 3, 4, 5]
         B = ["a", "b", "c", "d", "e"]

         C = {i:j for (i, j) in zip(A, B)}
         print(C)
```

```
{1: 'a', 2: 'b', 3: 'c', 4: 'd', 5: 'e'}
```