Lesson 2 - Thomas Deneuville

September 28, 2017

1 Sets

They are very much like lists but they store no duplicates. It only retains unique items. If you try to add any duplicates to the set, it ignores it.

Two applications for sets. One application is of course to remove duplicates. The next application is to check to see if the set contains something. Based off of Bloom Filters - very powerful fast technology that checks for the membership of elements.

You can `add`, `remove` elements from a list. They are also very similar to mathematical sets. You can find what is common between them with `intersection` and also you can join them together through `union`.

You can only put in **immutable** elements in a `set`. Integers, `float`, other `sets`, tuples, etc.

```
In [1]: s = {1, 5, 5, 5, 5, 6, 6, 6, 'hello', 'how', 'are', 'are', 'you', (0,0,0),(0,0,0)} # M
        #s = set([1, 5, 5, 5, 5, 6, 6, 6, 'hello', 'how', 'are', 'are', 'you', (0,0,0),(0,0,0)
        print(s)
```

```
{'are', 1, 5, 6, (0, 0, 0), 'how', 'you', 'hello'}
```

As you add into `sets`, the order of how you add elements in is no longer maintained. The reason why is because the random order is a consequence of a set to facilitate fast access.

2 Loop through a set

As previously, we can iterate through a set, but the order is no longer maintained

```
In [2]: for i in s: # For each element in the set s, let's print it out
            print(i)
```

```
are
1
5
6
(0, 0, 0)
how
you
hello
```

We can add and remove elements

```
In [3]: s.add(5)
        s.add(6)
        print(s) # Ignores adding in duplicates

        s.remove(5) # Remove 5 from the set
        print(s)

        # If we try to remove an element that doesn't exist, it will spit out an error
        s.remove(5)
```

```
{'are', 1, 5, 6, (0, 0, 0), 'how', 'you', 'hello'}
{'are', 1, 6, (0, 0, 0), 'how', 'you', 'hello'}
```

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

        KeyError                                  Traceback (most recent call last)

        <ipython-input-3-1c808d4982ef> in <module>()
          7
          8 # If we try to remove an element that doesn't exist, it will spit out an error
    ----> 9 s.remove(5)


        KeyError: 5
```

We can also determine a set that contains what is common between them both. Use the `intersection` method

```
In [4]: s2 = {1, 4, 5, 5, 5, 6, 6, 'hello', 'how', 'are', 'are', 'you'}
        s3 = {1, 1, 1, 1, 2, 2, 2, 3, 3, 3}

        print(s2)
        print(s3)

        # Find what is common between the two sets
        s4 = s2.intersection(s3)
        print(s4)

        # Join two sets together
        s5 = s2.union(s3)
        print(s5)
```

```
{'are', 1, 4, 5, 6, 'how', 'you', 'hello'}
{1, 2, 3}
```

```
{1}
{'are', 1, 2, 3, 4, 5, 6, 'how', 'you', 'hello'}
```

As an application, we can remove duplicates from a `list`. To do this, create a `set` out of this list then convert back to a `list`

```
In [5]: A = {1, 1, 1, 2, 2, 2, 3, 3, 3, 4, 4, 4, 5, 5, 5, 6}
        B = list(A)
        print(B)
```

```
[1, 2, 3, 4, 5, 6]
```

If you want to change the order of elements, convert to a `list` and change the order in however way you see fit.

3 One last thing about `lists`

```
In [6]: A = [5, 6, 7, 8, 9, 10]
        C = A # This creates a view - any changes you make to C, A will remember
        # This performs a SHALLOW copy
        print(A)
        print(C)
        C[2] = -1
        print(A)
        print(C)
```

```
[5, 6, 7, 8, 9, 10]
[5, 6, 7, 8, 9, 10]
[5, 6, -1, 8, 9, 10]
[5, 6, -1, 8, 9, 10]
```

If you want to copy the list, you create a deep copy or simply copy all elements to a new list

```
In [7]: F = list(A) # Creates a new list out of the original variable
        # F = A[:] # Step is omitted.  Begin and end are not specified, begin = 0, end goes to
        # If you perform list slicing, then that copies the list as well
        print(A)
        print(F)
        F[4] = -100
        print(A)
        print(F)
```

```
[5, 6, -1, 8, 9, 10]
[5, 6, -1, 8, 9, 10]
[5, 6, -1, 8, 9, 10]
[5, 6, -1, 8, -100, 10]
```

4 Functions

Very similar to a black box. You put in inputs, you process these inputs and you (optionally) return outputs. The keyword to use is the `def` keyword.

```
In [8]: def print_name(name): # print_name is the name of your function.  name is an input par
            print("Hello " + name +".  My name is Ray") # Simply prints off this message with
```

To call a function, simply use the name defined in the `def` keyword line and specify whatever input parameters are required for it.

```
In [9]: print_name("Thomas")
        print_name("Stewart")
```

```
Hello Thomas.  My name is Ray
Hello Stewart.  My name is Ray
```

The input parameters change depending on what you supply to the function

You can also get functions to return elements. Use the `return` keyword. A function will return a result that you can store inside a variable.

```
In [10]: def perimeter_of_a_rectangle(l, w): # perimeter_of_a_rectangle is the function name a
             p = 2*l + 2*w # This creates a variable p that calculates the perimeter
             return p # Return the perimeter
```

Any variables you create in the function, they have **local scope**. Once the function completes, those variables are not available to you anymore.

The above function takes in two variables. The length and width.

```
In [11]: o = perimeter_of_a_rectangle(3, 4)
         print(o)
```

```
14
```

You can make **default parameters**. If you don't specify them, they will assume a default value

```
In [12]: def perimeter_of_a_rectangle2(l, w=2): # If you don't specify w, the assumed value is
             p = 2*l + 2*w
             return p
```

```
In [13]: o2 = perimeter_of_a_rectangle2(3, 4)
         o3 = perimeter_of_a_rectangle2(2) #w = 2
         print(o2, o3)
```

```
14 8
```

You can also specify input parameters to be in whatever order you want. In the function, you specify the actual variable name followed by what you want to assign it to

```
In [14]: def test_function(param1, param2=7, param3="hello"): # Creating a function called tes
                                                             # 3 input parameters - two of th
                                                             # values if you don't specify th
             param2 = 7 * param2 # Let's change param2 to 7 times as much
             param3 = "goodbye" + param3 # Concatenate a string goodbye in front of hello
             return (param1, param2, param3) # Return anything you want.  In this case, I'm go

In [15]: A = test_function(8, 10, "test")
         B = test_function(param2=10, param1=-100, param3="bye") # The parameters are out of o
         C = test_function(10) # param1 is NOT optional.  You must specify it
         D = test_function(param2=100, param1=-1000) # You can also specify random order of pa
         print(A)
         print(B)
         print(C)
         print(D)

(8, 70, 'goodbyetest')
(-100, 70, 'goodbyebye')
(10, 49, 'goodbyehello')
(-1000, 700, 'goodbyehello')
```