9.4: LOOPING AND DICTIONARIES

Contributed by Chuck Severance
Clinical Associate Professor (School of Information) at University of Michigan

If you use a dictionary as the sequence in a `for` statement, it traverses the keys of the dictionary. This loop prints each key and the corresponding value:

CODE 9.4.1 (PYTHON):

```
%%python3

counts = { 'chuck' : 1 , 'annie' : 42, 'jan': 100}
for key in counts:
    print(key, counts[key])
```

run restart

Here's what the output looks like:

```
jan 100
chuck 1
annie 42
```

Again, the keys are in no particular order.

We can use this pattern to implement the various loop idioms that we have described earlier. For example if we wanted to find all the entries in a dictionary with a value above ten, we could write the following code:

CODE 9.4.1 (PYTHON):

```
%%python3

counts = { 'chuck' : 1 , 'annie' : 42, 'jan': 100}
for key in counts:
    if counts[key] > 10 :
        print(key, counts[key])
```

run restart

The `for` loop iterates through the *keys* of the dictionary, so we must use the index operator to retrieve the corresponding *value* for each key. Here's what the output looks like:

```
jan 100
annie 42
```

We see only the entries with a value above 10.

If you want to print the keys in alphabetical order, you first make a list of the keys in the dictionary using the `keys` method available in dictionary objects, and then sort that list and loop through the sorted list, looking up each key and printing out key-value pairs in sorted order as follows:

CODE 9.4.1 (PYTHON):

```
%%python3
```

```
counts = { 'chuck' : 1 , 'annie' : 42, 'jan': 100}
lst = list(counts.keys())
print(lst)
lst.sort()
for key in lst:
    print(key, counts[key])
```

run restart

Here's what the output looks like:

```
['jan', 'chuck', 'annie']
annie 42
chuck 1
jan 100
```

First you see the list of keys in unsorted order that we get from the `keys` method. Then we see the key-value pairs in order from the `for` loop.