Building Dictionaries

By now you are familiar with two important concepts: 1) counting with `for` loops and 2) the dictionary `get` method. These two can actually be combined to create a useful counter dictionary, something you will likely come across again. For example, we can create a dictionary, `word_counter`, that keeps track of the total count of each word in a string.

The following are a couple of ways to do it:

Method 1: Using a `for` loop to create a set of counters

Let's start with a list containing the words in a series of book titles:

```
book_title =  ['great', 'expectations','the', 'adventures', 'of', 'sherlock','holmes','the','great','gasby','hamlet','adventures','of','h
```

Step 1: Create an empty dictionary.

```
word_counter = {}
```

Step 2. Iterate through each element in the list. If an element is already included in the dictionary, add 1 to its value. If not, add the element to the dictionary and set its value to 1.

```
for word in book_title: if word not in word_counter:
word_counter[word] = 1 else: word_counter[word] += 1
```

What's happening here?

- The `for` loop iterates through each element in the list. For the first iteration, `word` takes the value 'great'.
- Next, the if statement checks if `word` is in the `word_counter` dictionary.
- Since it doesn't yet, the statement `word_counter[word] = 1` adds *great* as a key to the dictionary with a value of 1.
- Then, it leaves the if else statement and moves on to the next iteration of the for loop. `word` now takes the value *expectations* and repeats the process.
- When the if condition is not met, it is because that `word` already exists in the `word_counter` dictionary, and the statement `word_counter[word] = word_counter[word] + 1` increases the count of that word by 1.
- Once the `for` loop finishes iterating through the list, the `for` loop is complete.

We can see the output by printing out the dictionary. Printing `word_counter` results in the following output.

```
{'great': 2, 'expectations': 1, 'the': 2, 'adventures': 2, 'of': 2, 'sherlock': 1, 'holmes': 1, 'gasby': 1, 'hamlet': 1, 'huckleberry': 1
```

Feel free to try this out yourself in the code editor at the bottom of this page.

Method 2: Using the `get` method

We will use the same list for this example:

```
book_title =  ['great', 'expectations','the', 'adventures', 'of', 'sherlock','holmes','the','great','gasby','hamlet','adventures','of','h
```

Step 1: Create an empty dictionary.

```
word_counter = {}
```

Step 2. Iterate through each element, `get()` its value in the dictionary, and add 1.

Recall that the dictionary `get` method is another way to retrieve the value of a key in a dictionary. Except unlike indexing, this will return a default value if the key is not found. If unspecified, this default value is set to None. We can use `get` with a default value of 0 to simplify the code from the first method above.

Simplify the code from the first method above.

```
for word in book_title:
    word_counter[word] = word_counter.get(word, 0) + 1
```

What's happening here?

- The `for` loop iterates through the list as we saw earlier. The `for` loop feeds 'great' to the next statement in the body of the `for` loop.
- In this line: `word_counter[word] = word_counter.get(word,0) + 1`, since the key *'great'* doesn't yet exist in the dictionary, `get()` will return the value 0 and `word_counter[word]` will be set to 1.
- Once it encounters a word that already exists in `word_counter` (e.g. the second appearance of *'the'*), the value for that key is incremented by 1. On the second appearance of 'the', the key's value would add 1 again, resulting in 2.
- Once the `for` loop finishes iterating through the list, the `for` loop is complete.

Printing `word_counter` shows us we get the same result as we did in method 1.

```
{'great': 2, 'expectations': 1, 'the': 2, 'adventures': 2, 'of': 2,
'sherlock': 1, 'holmes': 1, 'gasby': 1, 'hamlet': 1, 'huckleberry':
1, 'fin': 1}
```

Again, feel free to try this out yourself in the workspace on the **Coding Space** page.
