Zipf's Law (part 2)

In the previous section, we computed the frequency of each word in a corpus using parallel lists, but we discussed why this isn't the best solution. Here we will discuss another data structure that is going to let us have direct access from *word -> count*.

Dictionaries

A Python dictionary is written like this:

```
{ "key1" : 20, "key2" : 30 }
```

This creates a dictionary where "key1" maps to 20 and "key2" maps to 30. Just like a list, you can refer to an item of the dictionary using square brackets.

```
>>> { "key1" : 20, "key2" : 30 }["key1"]
20
```

What happens when you use a key that isn't in the dictionary?

```
>>> { "key1" : 20, "key2" : 30 }["key3"]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 'key3'
```

If you're worried about this happening, you can check for a particular key using *in*, which is another one of Python's builtin comparatives:

```
>>> "key3" in { "key1" : 20, "key2" : 30 }
False
```

So a common pattern is code like this:

```
if key in counts:
   print("the value is", counts[key])
else:
   print("the value is 0")
```

Again, it's always safe to write this out, but you can abbreviate it using the *get* method, which takes two arguments, the *key* and a *default*, and returns either the value of *dict[key]* or *default* if that value isn't present.

```
print("the value is", counts.get(key, 0))
```

To add a new value, on the other hand, you can use square brackets even if the key isn't already there.

```
>>> mydict = { "key1" : 20, "key2" : 30 }
>>> mydict
{'key2': 30, 'key1': 20}
>>> mydict["key3"] = 40                #just mydict["key3"] is an error
>>> mydict
{'key3': 40, 'key2': 30, 'key1': 20}
```

Dictionaries and loops

Dictionaries support most of the same operations as lists (like *len*, which counts the number of keys). You can also loop through the dictionary using *for*. By default, this gives you all the keys:

```
>>> for key in { "key1" : 20, "key2" : 30 }:
...     print(key)
...
key2
key1
```

Unlike a list, the entries in the dictionary aren't in any particular order.

You can also get key-value pairs using the *items* method (and values using the *values* method).

```
>>> for key,value in { "key1" : 20, "key2" : 30 }.items():
...     print(key, "->", value)
...
key2 -> 30
key1 -> 20
```

What does the following code print?

```
>>> for item in { "key1" : 20, "key2" : 30}.items():
...     print(item)
```

And this one?

```
>>> for key,value in { "key1" : 20, "key2" : 30}.values():
...     print(key, "->", value)
```

Ok, back to the actual program

We are going to rewrite the word counting program with a dictionary this time (instead of with parallel lists). Before jumping into this, think about what your code should be doing.

What will be the content of our dictionary? We want to create a dictionary that is going to contain every word of the corpus and their frequencies. So what are the keys and what are the values of the dictionary?

How are we going to construct this dictionary? We will loop through the lines of the corpus and tokenize the lines to get words. Then, what do we do for each word in the lines? Either the word is already in the dictionary, and we just increment its frequency, or it is not yet in the dictionary. What do we do if it is not yet in the dictionary? We add it to the dictionary, with a frequency count of 1.

Time your script. If you've done this right, it should be a lot faster than the script with parallel lists!