9.2: DICTIONARY AS A SET OF COUNTERS

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

Suppose you are given a string and you want to count how many times each letter appears. There are several ways you could do it:

1. You could create 26 variables, one for each letter of the alphabet. Then you could traverse the string and, for each character, increment the corresponding counter, probably using a chained conditional.
2. You could create a list with 26 elements. Then you could convert each character to a number (using the built-in function `ord`), use the number as an index into the list, and increment the appropriate counter.
3. You could create a dictionary with characters as keys and counters as the corresponding values. The first time you see a character, you would add an item to the dictionary. After that you would increment the value of an existing item.

Each of these options performs the same computation, but each of them implements that computation in a different way.

An *implementation* is a way of performing a computation; some implementations are better than others. For example, an advantage of the dictionary implementation is that we don't have to know ahead of time which letters appear in the string and we only have to make room for the letters that do appear.

Here is what the code might look like:

CODE 9.2.1 (PYTHON):

```
%%python3

word = 'brontosaurus'
d = dict()
for c in word:
    if c not in d:
        d[c] = 1
    else:
        d[c] = d[c] + 1
print(d)
```

run restart

We are effectively computing a *histogram*, which is a statistical term for a set of counters (or frequencies).

The `for` loop traverses the string. Each time through the loop, if the character `c` is not in the dictionary, we create a new item with key `c` and the initial value 1 (since we have seen this letter once). If `c` is already in the dictionary we increment `d[c]`.

Here's the output of the program:

```
{'b': 1, 'r': 2, 'o': 2, 'n': 1, 't': 1, 's': 2, 'a': 1, 'u': 2}
```

The histogram indicates that the letters `'a'` and "b" appear once; "o" appears twice, and so on.

Dictionaries have a method called `get` that takes a key and a default value. If the key appears in the dictionary, `get` returns the corresponding value; otherwise it returns the default value. For example:

```
>>> counts = { 'chuck' : 1 , 'annie' : 42, 'jan': 100}
>>> print(counts.get('jan', 0))
100
>>> print(counts.get('tim', 0))
0
```

We can use `get` to write our histogram loop more concisely. Because the `get` method automatically handles the case where a key is not in a dictionary, we can reduce four lines down to one and eliminate the `if` statement.

CODE 9.2.1 (PYTHON):

```
%%python3

word = 'brontosaurus'
d = dict()
for c in word:
    d[c] = d.get(c,0) + 1
print(d)
```

run restart

The use of the `get` method to simplify this counting loop ends up being a very commonly used "idiom" in Python and we will use it many times in the rest of the book. So you should take a moment and compare the loop using the `if` statement and `in` operator with the loop using the `get` method. They do exactly the same thing, but one is more succinct.