Exercise 10.11.2

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, here's a dictionary that maps from the letters in a string to the number of times they appear.

```
def value_counts(string):
    counter = {}
    for letter in string:
        if letter not in counter:
            counter[letter] = 1
        else:
            counter[letter] += 1
    return counter
counter = value_counts('brontosaurus')
```

If we look up a letter that appears in the word, get returns the number of times it appears.

```
counter.get('b', 0)
```

```
1
```

If we look up a letter that doesn't appear, we get the default value, 0.

```
counter.get('c', 0)
```

```
0
```

Use get to write a more concise version of value_counts. You should be able to eliminate the if statement.

In [17]:
```
#Updated function to use get
def value_counts(string):
    counter = {}
    #loop through the string
    for letter in string:
        #Use the get method to return the current count in the counter dict
        current_count = counter.get(letter, 0)
        #add 1 to the current count for the letter
        counter[letter] = current_count+1
    #return the dictionary
    return counter

#Set up the test string and get the test result
test_string = 'brontosaurus'
test = value_counts(test_string)

#Create a list of the unique values in the test string
```

```
unique_letters = sorted(list(set(test_string)))

#Loop through the unique letters and print the letter and it's count
for letter in unique_letters:
    print(f'Letter: {letter}\tCount: {test.get(letter,0)}')
```

```
Letter: a       Count: 1
Letter: b       Count: 1
Letter: n       Count: 1
Letter: o       Count: 2
Letter: r       Count: 2
Letter: s       Count: 2
Letter: t       Count: 1
Letter: u       Count: 2
```

Exercise 10.11.4

Write function called find_repeats that takes a dictionary that maps from each key to a counter, like the result from value_counts. It should loop through the dictionary and return a list of keys that have counts greater than 1. You can use the following outline to get started.

```
def find_repeats(counter):

    """Makes a list of keys with values greater than 1.

    counter: dictionary that maps from keys to counts

    returns: list of keys

    """

    return []
```

In [ ]:
```
def find_repeats(counter):
    #Create an empty list
    key_list = []

    #loop through the keys and values of each dictionary item
    for k, v in counter.items():

        #Append the key to the list if the value is greater than 1
        if v>1:
            key_list.append(k)

    #return the list
    return key_list

find_repeats(test)
```

Out[ ]:
```
['r', 'o', 's', 'u']
```