11.6. Introduction: Accumulating Multiple Results In a Dictionary

You have previously seen the accumulator pattern; it goes through the items in a sequence, updating an accumulator variable each time. Rather than accumulating a single result, it's possible to accumulate many results. Suppose, for example, we wanted to find out which letters are used most frequently in English.

Suppose we had a reasonably long text that we thought was representative of general English usage. For our purposes in the this chapter, we will use the text of the Sherlock Holmes story, "A Study in Scarlet", by Sir Arthur Conan Doyle. The text actually includes a few lines about the source of the transcription (Project Gutenberg), but those will not materially affect our analyses so we will just leave them in. You can access this text within this chapter with the code `open('scarlet.txt', 'r')`.

As with other files that we access in this textbook environment, this one is actually pre-loaded in your browser, not retrieved from your computer's file system. That's why this chapter may be a little slower to load than others. You can view the text of "A Study in Scarlet" at the bottom of the page.

If we want to find out how often the letter 't' occurs, we can accumulate the result in a count variable.

Save & Run Original - 1 of 1 Show in CodeLens

```
f = open('scarlet.txt', 'r')
txt = f.read()
# now txt is one long string containing all the characters
t_count = 0 #initialize the accumulator variable
for c in txt:
    if c == 't':
        t_count = t_count + 1   #increment the counter
print("t: " + str(t_count) + " occurrences")

```

```
t: 17584 occurrences
```

Activity: 1 -- ActiveCode (ac10_5_1)

We can accumulate counts for more than one character as we traverse the text. Suppose, for example, we wanted to compare the counts of 't' and 's' in the text.

Save & Run Original - 1 of 1 Show in CodeLens

```
f = open('scarlet.txt', 'r')
txt = f.read()
# now txt is one long string containing all the characters
t_count = 0 #initialize the accumulator variable
s_count = 0 # initialize the s counter accumulator as well
for c in txt:
    if c == 't':
        t_count = t_count + 1   #increment the t counter
    elif c == 's':
        s_count = s_count + 1
print("t: " + str(t_count) + " occurrences")
print("s: " + str(s_count) + " occurrences")

```

```
t: 17584 occurrences
s: 11830 occurrences
```

Activity: 2 -- ActiveCode (ac10_5_2)

OK, but you can see this is going to get tedious if we try to accumulate counts for all the letters. We will have to initialize a lot of accumulators, and there will be a very long if..elif..elif statement. Using a dictionary, we can do a lot better.

One dictionary can hold all of the accumulator variables. Each key in the dictionary will be one letter, and the corresponding value will be the count so far of how many times that letter has occurred.

Save & Run Original - 1 of 1 Show in CodeLens

```
f = open('scarlet.txt', 'r')
txt = f.read()
# now txt is one long string containing all the characters
x = {} # start with an empty dictionary
x['t'] = 0  # initialize the t counter
x['s'] = 0  # initialize the s counter
for c in txt:
    if c == 't':
```

```
        x['t'] = x['t'] + 1  # increment the t counter
    elif c == 's':
        x['s'] = x['s'] + 1  # increment the s counter

print("t: " + str(x['t']) + " occurrences")
print("s: " + str(x['s']) + " occurrences")
```

```
t: 17584 occurrences
s: 11830 occurrences
```

Activity: 3 -- ActiveCode (ac10_5_3)

This hasn't really improved things yet, but look closely at lines 8-11 in the code above. Whichever character we're seeing, t or s, we're incrementing the counter for that character. So lines 9 and 11 could really be the same.

Save & Run | Original - 1 of 1 | Show in CodeLens

```
f = open('scarlet.txt', 'r')
txt = f.read()
# now txt is one long string containing all the characters
x = {} # start with an empty dictionary
x['t'] = 0  # intiialize the t counter
x['s'] = 0  # initialize the s counter
for c in txt:
    if c == 't':
        x[c] = x[c] + 1   # increment the t counter
    elif c == 's':
        x[c] = x[c] + 1   # increment the s counter

print("t: " + str(x['t']) + " occurrences")
print("s: " + str(x['s']) + " occurrences")
```

```
t: 17584 occurrences
s: 11830 occurrences
```

Activity: 4 -- ActiveCode (ac10_5_4)

Lines 9 and 11 above may seem a little confusing at first. Previously, our assignment statements referred directly to keys, with `x['s']` and `x['t']`. Here we are just using a variable `c` whose value is 's' or 't', or some other character.

If that made perfect sense to you, skip the next two paragraphs. Otherwise, read on. Let's break down that line in a little more detail.

First, note that, as with all assignment statements, the right side is evaluated first. In this case `x[c]` has to be evaluated. As with all expressions, we first have to substitute values for variable names. `x` is a variable bound to a dictionary. `c` is a variable bound to one letter from the string that `txt` is bound to (that's what the for statement says to do: execute lines 8-11 once for each character in txt, with the variable c bound to the current character on each iteration.) So, let's suppose that the current character is the letter `s` (we are on line 11). Then `x[c]` looks up the value associated with the key 's' in the dictionary x. If all is working correctly, that value should be the number of times 's' has previously occurred. For the sake of argument, suppose it's 25. Then the right side evaluates to 25 + 1, 26. Watch this play out below.

Next Step | Reset

```
f = open('scarlet.txt', 'r')
txt = f.read()
# now txt is one long string containing all the characters
x = {} # start with an empty dictionary
x['t'] = 15 # intiialize the t counter
x['s'] = 25 # initialize the s counter
```

```
for c in txt: # we have reached the 26th s now
for 's' in txt:
if c == 't':
if 's' == 't':
elif c == 's':
elif 's' == 's':
x[c] = x[c] + 1 # increment the s counter
x['s'] = x['s'] + 1 # increment the s counter
x['s'] = 25 + 1 # increment the s counter
x['s'] = 26 # increment the s counter
```

Activity: 5 -- ShowEval (eval10_5_4)

Now we have assigned the value 26 to `x[c]`. That is, in dictionary x, we set the value associated with the key 's' (the current value of the variable c) to be 26. In other words, we have incremented the value associated with the key 's' from 25 to 26.

We can do better still. One other nice thing about using a dictionary is that we don't have to prespecify what all the letters will be. In this case, we know in advance what the alphabet for English is, but later in the chapter we will count the occurrences of words, and we do not know in advance all the of the words that may be used. Rather than pre-specifying which letters to keep accumulator counts for, we can start with an empty dictionary and add a counter to the dictionary each time we encounter a new thing that we want to start keeping count of.

Save & Run | Original - 1 of 1 | Show in CodeLens

```
f = open('scarlet.txt', 'r')
txt = f.read()
# now txt is one long string containing all the characters
x = {} # start with an empty dictionary
for c in txt:
    if c not in x:
        # we have not seen this character before, so initialize a counter for it
        x[c] = 0

    #whether we've seen it before or not, increment its counter
    x[c] = x[c] + 1

print("t: " + str(x['t']) + " occurrences")
print("s: " + str(x['s']) + " occurrences")

```

```
t: 17584 occurrences
s: 11830 occurrences
```

Activity: 6 -- ActiveCode (ac10_5_5)

Notice that in the for loop, we no longer need to explicitly ask whether the current letter is an 's' or 't'. The increment step on line 11 works for the counter associated with whatever the current character is. Our code is now accumulating counts for all letters, not just 's' and 't'.

Check your understanding

dictionaries-5-1: Which of the following will print out True if there are more occurrences of e than t in the text of A Study in Scarlet, and False if t occurred more frequently (assumming that the previous code, from dict_accum_5, has already run.)

- A. print(txt['e'] > txt['t'])
- B. print(x['e'] > x['t'])
- C. print(x[e] > x[t])
- D. print(x[c] > txt[c])
- E. print(e[x] > t[x])

Check me | Compare me

✔ x is the dictionary of counts; you want to compare the values associated with 'e' and 't'.

Activity: 7 -- Multiple Choice (question10_5_1)

Note that the print statements at the end pick out the specific keys 't' and 's'. We can generalize that, too, to print out the occurrence counts for all of the characters, using a for loop to iterate through the keys in x.

Save & Run | Original - 1 of 1 | Show in CodeLens

```
f = open('scarlet.txt', 'r')
txt = f.read()
# now txt is one long string containing all the characters
letter_counts = {} # start with an empty dictionary
for c in txt:
    if c not in letter_counts:
        # we have not seen this character before, so initialize a counter for it
        letter_counts[c] = 0

    #whether we've seen it before or not, increment its counter
    letter_counts[c] = letter_counts[c] + 1

for c in letter_counts.keys():
    print(c + ": " + str(letter_counts[c]) + " occurrences")

```

```
t: 17584 occurrences
s: 11830 occurrences
T: 699 occurrences
h: 12892 occurrences
e: 25410 occurrences
 : 43507 occurrences
P: 203 occurrences
r: 12054 occurrences
o: 15506 occurrences
j: 214 occurrences
c: 5121 occurrences
G: 235 occurrences
u: 5643 occurrences
n: 13714 occurrences
b: 2741 occurrences
g: 4017 occurrences
E: 270 occurrences
B: 156 occurrences
k: 1480 occurrences
f: 4223 occurrences
A: 389 occurrences
S: 404 occurrences
d: 9013 occurrences
y: 3721 occurrences
I: 1358 occurrences
a: 15872 occurrences
l: 7445 occurrences
,: 3137 occurrences
C: 185 occurrences
D: 205 occurrences

: 5155 occurrences
i: 12695 occurrences
```

```
w: 4745 occurrences
m: 5097 occurrences
v: 1906 occurrences
.: 2785 occurrences
Y: 201 occurrences
p: 3337 occurrences
-: 609 occurrences
L: 256 occurrences
:: 91 occurrences
J: 116 occurrences
1: 103 occurrences
2: 54 occurrences
0: 27 occurrences
8: 24 occurrences
[: 52 occurrences
#: 1 occurrences
4: 32 occurrences
]: 52 occurrences
R: 181 occurrences
9: 21 occurrences
5: 21 occurrences
F: 206 occurrences
7: 17 occurrences
3: 31 occurrences
*: 28 occurrences
O: 193 occurrences
H: 546 occurrences
U: 82 occurrences
N: 240 occurrences
K: 20 occurrences
q: 153 occurrences
': 592 occurrences
x: 330 occurrences
M: 191 occurrences
(: 23 occurrences
_: 34 occurrences
W: 286 occurrences
): 23 occurrences
z: 140 occurrences
": 1795 occurrences
?: 208 occurrences
!: 86 occurrences
;: 104 occurrences
V: 30 occurrences
6: 19 occurrences
Q: 2 occurrences
è: 1 occurrences
é: 2 occurrences
ñ: 4 occurrences
Z: 2 occurrences
&: 2 occurrences
/: 25 occurrences
%: 1 occurrences
X: 2 occurrences
@: 2 occurrences
$: 2 occurrences
```

Activity: 8 -- ActiveCode (ac10_5_6)

Note that only those letters that actually occur in the text are shown. Some punctuation marks that are possible in English, but were never used in the text, are omitted completely. The blank line partway through the output may surprise you. That's actually saying that the newline character, `\\n`, appears 5155 times in the text. In other words, there are 5155 lines of text in the file. Let's test that hypothesis.

Save & Run | Original - 1 of 1 | Show in CodeLens

```
f = open('scarlet.txt', 'r')
txt_lines = f.readlines()
# now txt_lines is a list, where each item is one
# line of text from the story
print(len(txt_lines))
print(txt_lines[70:85])

```

```
5155
['(_Being a reprint from the reminiscences of_ JOHN H. WATSON, M.D., _late\n', 'of the Army Medica
l Department._) [2]\n', '\n', '\n', '\n', '\n', 'CHAPTER I. MR. SHERLOCK HOLMES.\n', '\n', '\n',
'IN the year 1878 I took my degree of Doctor of Medicine of the\n', 'University of London, and pro
ceeded to Netley to go through the course\n', 'prescribed for surgeons in the army. Having complet
ed my studies there,\n', 'I was duly attached to the Fifth Northumberland Fusiliers as Assistant
\n', 'Surgeon. The regiment was stationed in India at the time, and before\n', 'I could join it, t
he second Afghan war had broken out. On landing at\n']
```

Activity: 9 -- ActiveCode (ac10_5_7)

Now, here are some additional problems to try.

2. Provided is a string saved to the variable name `sentence`. Split the string into a list of words, then create a dictionary that contains each word and the number of times it occurs. Save this dictionary to the variable name `word_counts`.

Save & Run 5/14/2021, 8:55:35 AM - 2 of 2 Show in CodeLens

```
sentence = "The dog chased the rabbit into the forest but the rabbit was too quick.
words = sentence.split()
word_counts = {}

for word in words:
    if word not in word_counts:
        word_counts[word] = 0
    word_counts[word] = word_counts[word] + 1

```

Activity: 10 -- ActiveCode (ac10_5_8)

Result	Actual Value	Expected Value	Notes
Pass	[('Th..., 1)]	[('Th..., 1)]	Testing that word_counts was created correctly.

Expand Differences

You passed: 100.0% of the tests

3. Create a dictionary called `char_d` from the string `stri`, so that the key is a character and the value is how many times it occurs.

Save & Run 5/14/2021, 8:56:27 AM - 2 of 2 Show in CodeLens

```
stri = "what can I do"

char_d = {}

for c in stri:
    if c not in char_d:
        char_d[c] = 0
    char_d[c] = char_d[c] + 1

```

Activity: 11 -- ActiveCode (ac10_5_9)

Result	Actual Value	Expected Value	Notes
Pass	[(' '..., 1)]	[(' '..., 1)]	Testing that char_d has been created correctly.

Expand Differences

You passed: 100.0% of the tests

Data file: `scarlet.txt`

You have attempted 12 of 11 activities on this page

11.5. Aliasing and copying">
11.7. Accumulating Results From a Dictionary">
11.5. Aliasing and copying">
Complete
11.7. Accumulating Results From a Dictionary">Next Section - 11.7. Accumulating Results From a Dictionary

| Back to top