Other Common String Methods

WE'LL COVER THE FOLLOWING

- Slicing A String

Besides formatting, strings can do a number of other useful tricks.

```
s = '''Finished files are the re-
 sult of years of scientif-
 ic study combined with the
 experience of years.'''
                                    #①
print(s.splitlines())               #②
#['Finished files are the re-',
# 'sult of years of scientif-',
# 'ic study combined with the',
# 'experience of years.']

print(s.lower())                    #③
#finished files are the re-
#sult of years of scientif-
#ic study combined with the
#experience of years.

print(s.lower().count('f'))         #④
#6
```

① You can input multiline strings in the Python interactive shell. Once you start a multiline string with triple quotation marks, just hit `ENTER` and the interactive shell will prompt you to continue the string. Typing the closing triple quotation marks ends the string, and the next `ENTER` will execute the command (in this case, assigning the string to `s`).

② The `splitlines()` method takes one multiline string and returns a list of strings, one for each line of the original. Note that the carriage returns at the end of each line are not included.

end of each line are not included.

③ The `lower()` method converts the entire string to lowercase. (Similarly, the `upper()` method converts a string to uppercase.)

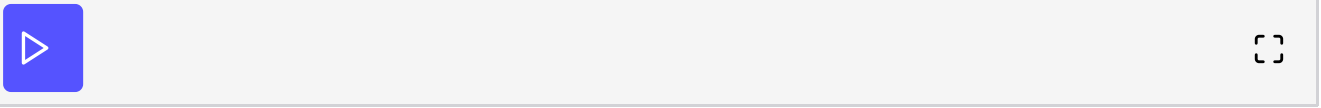
④ The `count()` method counts the number of occurrences of a substring. Yes, there really are six “f”s in that sentence!

Here’s another common case. Let’s say you have a list of key-value pairs in the form `key1=value1&key2=value2`, and you want to split them up and make a dictionary of the form `{key1: value1, key2: value2}`.

```
query = 'user=pilgrim&database=master&password=PapayaWhip'
a_list = query.split('&')                                    #①
print (a_list)
#['user=pilgrim', 'database=master', 'password=PapayaWhip']

a_list_of_lists = [v.split('=', 1) for v in a_list if '=' in v]  #②
print (a_list_of_lists)
#[['user', 'pilgrim'], ['database', 'master'], ['password', 'PapayaWhip']]

a_dict = dict(a_list_of_lists)                               #③
print (a_dict)
#{'user': 'pilgrim', 'database': 'master', 'password': 'PapayaWhip'}
```



① The `split()` string method has one required argument, a delimiter. The method splits a string into a list of strings based on the delimiter. Here, the delimiter is an ampersand character, but it could be anything.

② Now we have a list of strings, each with a key, followed by an equals sign, followed by a value. We can use a list comprehension to iterate over the entire list and split each string into two strings based on the first equals sign. The optional second argument to the `split()` method is the number of times you want to split. `1` means “only split once,” so the `split()` method will return a two-item list. (In theory, a value could contain an equals sign too. If you just used `'key=value=foo'.split('=')`, you would end up with a three-item list `['key', 'value', 'foo']`.)

③ Finally, Python can turn that list-of-lists into a dictionary simply by passing it to the `dict()` function.

> *The previous example looks a lot like parsing query parameters in a **URL**, but real-life **URL** parsing is actually more complicated than this. If you're dealing with url query parameters, you're better off using the urllib.parse.parse_qs() function, which handles some non-obvious edge cases.*

Slicing A String

Once you've defined a string, you can get any part of it as a new string. This is called `slicing` the string. Slicing strings works exactly the same as slicing lists, which makes sense, because strings are just sequences of characters.

```
a_string = 'My alphabet starts where your alphabet ends.'
print (a_string[3:11])           #①
#alphabet

print (a_string[3:-3])           #②
#alphabet starts where your alphabet en

print (a_string[0:2] )           #③
#My

print (a_string[:18])            #④
#My alphabet starts

print (a_string[18:] )           #⑤
# where your alphabet ends.
```

① You can get a part of a string, called a "slice", by specifying two indices. The return value is a new string containing all the characters of the string, in order, starting with the first slice index.

② Like slicing lists, you can use negative indices to slice strings.

③ Strings are zero-based, so `a_string[0:2]` returns the first two items of the string, starting at `a_string[0]`, up to but not including `a_string[2]`.

④ If the left slice index is 0, you can leave it out, and 0 is implied. So `a_string[:18]` is the same as `a_string[0:18]`, because the starting 0 is

implied.

⑤ Similarly, if the right slice index is the length of the string, you can leave it out. So `a_string[18:]` is the same as `a_string[18:44]`, because this string has 44 characters. There is a pleasing symmetry here. In this 44-character string, `a_string[:18]` returns the first 18 characters, and `a_string[18:]` returns everything but the first 18 characters. In fact, `a_string[:n]` will always return the first n characters, and `a_string[n:]` will return the rest, regardless of the length of the string.