PREV
Lesson 7. Introducing string objects: sequences of

NEXT
Lesson 9. Simple error messages

Lesson 8. Advanced string operations

After reading lesson 8, you'll be able to

- Manipulate substrings
- Do mathematical operations with strings

If you're given a long file, it's typical to read the entire file as one large string. But working with such a large string can be cumbersome. One useful thing you might do is break it into smaller substrings—most often, by new lines, so that every paragraph or every data entry could be looked at separately. Another beneficial thing is to find multiple instances of the same word. You could decide that using the word *very* more than 10 times is annoying. Or if you're reading the transcript of someone's award acceptance speech, you may want to find all instances of the word *like* and remove those before posting it.

Consider this

While researching the way that teens text, you gather some data. You're given a long string with many lines, in the following format:

- #0001: gr8 lets meet up 2day
- #0002: hey did u get my txt?
- #0003: ty, pls check for me
- ...

Given that this is originally one large string, what are some steps that you could take to make the data more approachable by analyzing it?

Answer:

1. Separate the big data string into a substring for each line.
2. Replace common acronyms with proper words (for example, pls with please).
3. Count the number of times certain words occur in order to report on the most popular acronyms.

8.1. OPERATIONS RELATED TO SUBSTRINGS

In lesson 7, you learned to retrieve a substring from a string when you knew what indices you wanted to use. You can do more-advanced operations that can give you more information regarding the composition of a string.

8.1.1. Find a specific substring in a string with find()

Suppose you have a long list of filenames on your computer and want to find out whether a specific file exists, or you want to search for a word in a text document. You can find a particular case-sensitive substring inside a larger string by using the `find()` command.

As with the commands to manipulate case, you write the string you want to do the operation on, then a dot, then the command name, and then the parentheses. For example: `"some_string".find()`. Note that the empty string, `''`, is in every string.

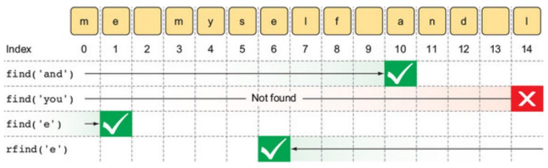
But this isn't all. In addition, you tell the command what substring you want to find by putting it in the parentheses—for example, `"some_string".find("ing")`.

The substring you want to find must be a string object. The result you get back is the index (starting from 0), in the string, where the substring starts. If more than one substring matches, you get the index of the first one found. If the substring isn't in the string, you get `-1`. For example, If you want to start looking for a substring from the end of the string instead of the beginning, you can use a different command, `rfind()`. The `r` in `rfind` stands for *reverse find*. It looks for the substring nearest to the end of the string and reports the index (starting from 0) at which the substring starts.

If you have `who = "me myself and I"`, then figure 8.1 shows how to evaluate the following:

- `who.find("and")` evaluates to `10` because the substring starts at index 10.
- `who.find("you")` evaluates to `-1` because the substring isn't in the string.
- `who.find("e")` evaluates to `1` because the first occurrence of the substring is at index 1.
- `who.rfind("el")` evaluates to `6` because the first occurrence of the substring nearest to the end of the string is at index 6.

Figure 8.1. Four examples of substrings to find in the string `"me myself and I"`. The arrows indicate the direction in which you're finding the substring. The check mark tells you the index in which you found the substring. An x tells you that the substring wasn't found.



Quick check 8.1

You're given `a = "python 4 ever&EVER"`. Evaluate the following expressions. Then try them in Spyder to check yourself:

1

`a.find("E")`

2

`a.find("eve")`

3

`a.rfind("rev")`

4

`a.rfind("VER")`

5

`a.find(" ")`

6

`a.rfind(" ")`

8.1.2. Find out whether a substring is in the string with "in"

The `find` and `rfind` operations tell you where to find a substring. Sometimes you only want to know whether the substring is in the string. This is a small variation on `find` and `rfind`. You can use the yes or no answer to this question more efficiently when you don't need to know the exact location of the substring. Because there are only two values, the answer to this question is an object of type Boolean, and the value you get back will be either `True` or `False`. The operation to find the answer to this question uses the keyword `in`. For example, `"a" in "abc"` is an expression that evaluates to `True` because the string `"a"` is in the string `"abc"`. The keyword `in` is used frequently in Python because it makes a lot of the code you write look very much like English.

Quick check 8.2

You're given `a = "python 4 ever&EVER"`. Evaluate the following expressions. Then try them in Spyder to check yourself:

1

2

```
"" in a
```

3

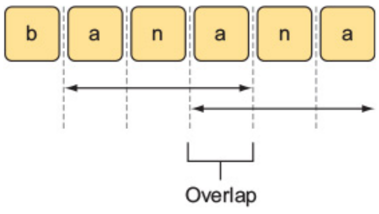
```
"2 * 2" in a
```

8.1.3. Count the number of times a substring occurs with count()

Especially when editing a document, you'll find it useful to make sure you aren't overusing words. Suppose you're editing an essay and find that within the first paragraph, you used the word *so* five times already. Instead of manually counting the number of times that word occurs in the whole essay, you can take the text you've written and automatically find the number of times the substring `"so"` occurs by using an operation on strings.

You can count the number of times a substring occurs in a string by using `count()`, which will give you back an integer. For example, if you have `fruit = "banana"`, then `fruit.count("an")` evaluates to `2`. One important point about `count()` is that it doesn't count overlapping substrings. `fruit.count("ana")` evaluates to `1` because the `"a"` overlaps between the two occurrences of `"ana"`, as shown in figure 8.2.

Figure 8.2. Counting the number of occurrences of `"ana"` in the string `"banana"`. The answer is 1 because `"a"` overlaps between the two occurrences, and the Python `count()` command doesn't take this into account.



Quick check 8.3

You're given `a = "python 4 ever&EVER"`. Evaluate the following expressions. Then try them in Spyder to check yourself:

1

```
a.count("ev")
```

2

```
a.count(" ")
```

3

```
a.count(" 4 ")
```

4

```
a.count("eVer")
```

8.1.4. Replace substrings with replace()

Suppose your son wrote a short report on his favorite fruit: apples. The morning of the day it's due, he changes his mind, hates apples, and now loves pears. You can take his entire report as a string and easily replace all instances of the word *apple* with *pear*.

A final useful string operation is to replace one substring in the string with another substring. This command operates on a string, as the previous ones do, but you have to put in two items in the parentheses, separated by a comma. The first item is the substring to find, and the second is the substring replacement. This command replaces all occurrences. For example, `"variables have no spaces".replace(" ", "_")` replaces all occurrences of the space string with the underscore string in the string `"variables have no spaces"`, and evaluates to `"variables_have_no_spaces"`.

Quick check 8.4

You're given `a = "Raining in the spring time."` Evaluate the following expressions. Then try them in Spyder to check yourself:

1

```
a.replace("R", "r")
```

2

```
a.replace("ing", "")
```

3

```
a.replace("!", ".")
```

4

```
b = a.replace("time","tiempo")
```

8.2. MATHEMATICAL OPERATIONS

You can do only two mathematical operations on string objects: addition and multiplication.

Addition, which is allowed only between two string objects, is called *concatenation*. For example, `"one" + "two"` evaluates to `'onetwo'`. When you add two strings, you put the values of each string together, in the order of the addition, to make a new string object. You may want to add one string to another if, for example, you have three people who worked on a report and wrote individual sections; all that remains is to combine the first, then the second, and lastly, the third.

Multiplication, which is allowed only between a string object and an integer, is called *repetition*. For example, `3 * "a"` evaluates to `'aaa'`. When you multiply a string by an integer, the string is repeated that many times. Multiplying a string by a number is often used to save time and for precision. For example, let's say you want to create a string representing all unknown letters when playing hangman. Instead of initializing a string as `"----------"`, you could do `"-" * 10`. This is especially useful if you don't know the size of the word to guess in advance, and you can store the size in a variable that you'll then multiply by the `"-"` character.

Quick check 8.5

Evaluate the following expressions. Then try them in Spyder to check yourself:

1

```
"la" + "la" + "Land"
```

2

```
"USA" + " vs " + "Canada"
```

3

```
b = "NYc"
c = 5
b * c
```

4

```
color = "red"
shape = "circle"
number = 3
number * (color + "-" + shape)
```

SUMMARY

In this lesson, my objective was to teach you about more operations you can do with string objects, specifically related to substrings. You learned how to find whether a substring is in a string, get its index location, count the number of times it occurred, and replace all occurrences of the substring. You also saw how to add two strings and what it means to multiply a string with a number. Here are the major takeaways:

- You can manipulate a string with just a few operations to make it look the way you'd like.

- Repeating a string means you're multiplying the string by a number.

Let's see if you got this...

Q8.1

Write a program that initializes a string with the value `"Eat Work Play Sleep repeat"`. Then, use the string manipulation commands you've learned so far to get the string `"working playing"`.

Recommended / Playlists / History / Topics / Settings / Get the App / Sign Out

PREV
Lesson 7. Introducing string objects: sequences of characters

NEXT
Lesson 9. Simple error messages