**STRING METHODS**

**Introduction**

Do you have a gigantic string that you need to parse for information? Do you need to sanitize a users input to work in a function? Do you need to be able to generate outputs with variable values? All of these things can be accomplished with *string methods*!

Python comes with built-in *string methods* that gives you the power to perform complicated tasks on strings very quickly and efficiently. These string methods allow you to change the case of a string, split a string into many smaller strings, join many small strings together into a larger string, and allow you to neatly combine changing variables with string outputs.

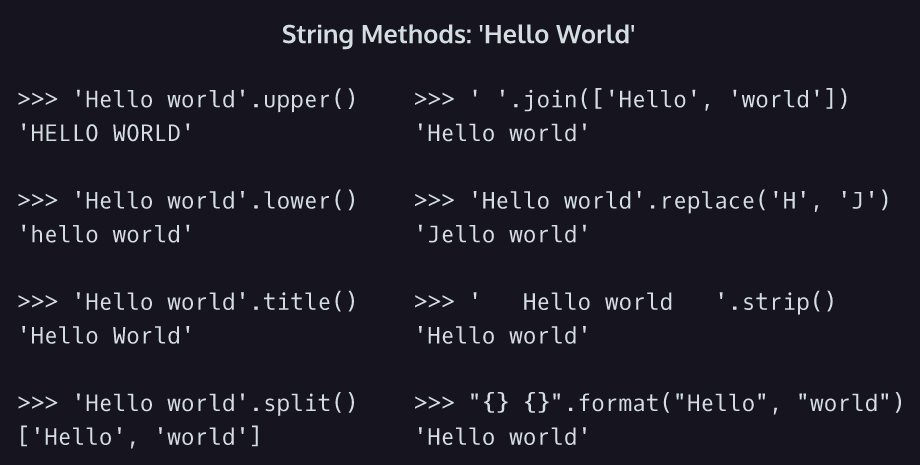
In the previous lesson, you worked with len(), which was a *function* that determined the number of characters in a string. This, while similar, was NOT a string method. String methods all have the same syntax:

string\_name.string\_method(arguments)

Unlike len(), which is called with a string as its argument, a string method is called at the end of a string and each one has its own method specific arguments.

**Instructions**

The diagram shows all of the string methods you can expect to learn in this lesson. Take a quick look at them and then let’s get started!



**Formatting Methods**

There are three string methods that can change the casing of a string. These are .lower(), .upper(), and .title().

* .lower() returns the string with all lowercase characters.
* .upper() returns the string with all uppercase characters.
* .title() returns the string in title case, which means the first letter of each word is capitalized.

Here’s an example of .lower() in action:

favorite\_song = 'SmOoTH'  
favorite\_song\_lowercase = favorite\_song.lower()  
print(favorite\_song\_lowercase)  
# => 'smooth'

Every character was changed to lowercase! It’s important to remember that string methods can only **create** new strings, they do not change the original string.

print(favorite\_song)  
# => 'SmOoTH'

See, it’s still the same! These string methods are great for sanitizing user input and standardizing the formatting of your strings.

**Instructions**

**1.**

You’re a programmer working for an organization that is trying to digitize and store poetry called *Preserve the Verse*.

You’ve been given two strings, the title of a poem and its author, and have been asked to reformat them slightly to fit the conventions of the organization’s database.

Make poem\_title have title case and save it to poem\_title\_fixed.

Checkpoint 2 Passed

**2.**

Print poem\_title and poem\_title\_fixed.

How did the string change?

Checkpoint 3 Passed

**3.**

The organization’s database also needs the author’s name to be uppercase only.

Make poem\_author uppercase and save it to poem\_author\_fixed.

Checkpoint 4 Passed

**4.**

Print poem\_author and poem\_author\_fixed.

Again, how did the string change?

**script.py**

poem\_title = "spring storm"

poem\_author = "William Carlos Williams"

poem\_title\_fixed = poem\_title.title()

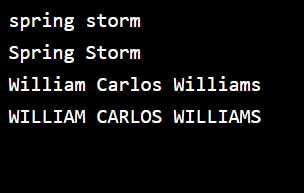
print(poem\_title)

print(poem\_title\_fixed)

poem\_author\_fixed = poem\_author.upper()

print(poem\_author)

print(poem\_author\_fixed)

****

**Splitting Strings**

.upper(), .lower(), and .title() all are performed on an existing string and produce a string in return. Let’s take a look at a string method that returns a different object entirely!

.split() is performed on a string, takes one argument, and returns a list of substrings found between the given argument (which in the case of .split() is known as the delimiter). The following syntax should be used:

string\_name.split(delimiter)

If you do not provide an argument for .split() it will default to splitting at spaces.

For example, consider the following strings:

man\_its\_a\_hot\_one = "Like seven inches from the midday sun"  
print(man\_its\_a\_hot\_one.split())  
# => ['Like', 'seven', 'inches', 'from', 'the', 'midday', 'sun']

.split returned a list with each word in the string. Important to note: if we run .split() on a string with no spaces, we will get the same string in return.

**Instructions**

**1.**

In the code editor is a string of the first line of the poem *Spring Storm* by William Carlos Williams.

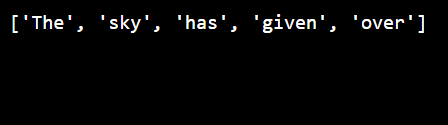
Use .split() to create a list called line\_one\_words that contains each word in this line of poetry.

**script.py**

line\_one = "The sky has given over"

line\_one\_words = line\_one.split()

print(line\_one\_words)

****

**Splitting Strings II**

If we provide an argument for .split() we can dictate the character we want our string to be split on. This argument should be provided as a string itself.

Consider the following example:

greatest\_guitarist = "santana"  
print(greatest\_guitarist.split('n'))  
# => ['sa', 'ta', 'a']

We provided 'n' as the argument for .split() so our string “santana” got split at each 'n' character into a list of three strings.

What do you think happens if we split the same string at 'a'?

print(greatest\_guitarist.split('a'))  
# => ['s', 'nt', 'n', '']

Notice that there is an unexpected extra '' string in this list. When you split a string on a character that it also ends with, you’ll end up with an empty string at the end of the list.

You can use *any* string as the argument for .split(), making it a versatile and powerful tool.

**Instructions**

**1.**

Your boss at the Poetry organization sent over a bunch of author names that he wants you to prepare for importing into the database. Annoyingly, he sent them over as a long string with the names separated by commas.

Using .split() and the provided string, create a list called author\_names containing each individual author name as it’s own string.

Checkpoint 2 Passed

**2.**

Great work, but now it turns out they didn’t want poet’s first names (why didn’t they just say that the first time!?)

Create another list called author\_last\_names that only contains the last names of the poets in the provided string.

Checkpoint 3 Passed

Hint

There are several ways to do this, but one way is to iterate through the list you created in part one and use .split(), negative indexing, and .append() to construct the new list.

**script.py**

authors = "Audre Lorde,Gabriela Mistral,Jean Toomer,An Qi,Walt Whitman,Shel Silverstein,Carmen Boullosa,Kamala Suraiyya,Langston Hughes,Adrienne Rich,Nikki Giovanni"

author\_names = authors.split(',')

print(author\_names)

subresult = []

author\_last\_names = []

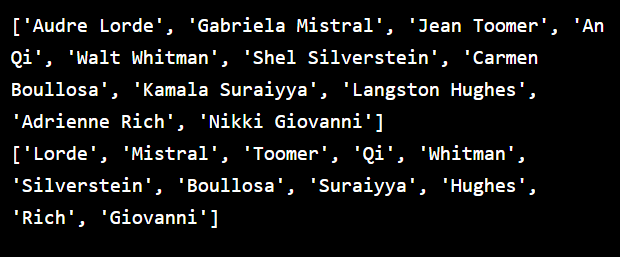
for author in author\_names:

  subresult.append(author.split(' '))

for author in subresult:

  author\_last\_names.append(author[1])

print(author\_last\_names)

****

**Splitting Strings III**

We can also split strings using *escape sequences*. Escape sequences are used to indicate that we want to split by something in a string that is not necessarily a character. The two escape sequences we will cover here are

* \n Newline
* \t Horizontal Tab

Newline or \n will allow us to split a multi-line string by line breaks and \t will allow us to split a string by tabs. \t is particularly useful when dealing with certain datasets because it is not uncommon for data points to be separated by tabs.

Let’s take a look at an example of splitting by an escape sequence:

smooth\_chorus = \  
"""And if you said, "This life ain't good enough."  
I would give my world to lift you up  
I could change my life to better suit your mood  
Because you're so smooth"""  
   
chorus\_lines = smooth\_chorus.split('\n')  
   
print(chorus\_lines)

This code is splitting the multi-line string at the newlines (\n) which exist at the end of each line and saving it to a new list called chorus\_lines. Then it prints chorus\_lines which will produce the output

['And if you said, "This life ain\'t good enough."', 'I would give my world to lift you up', 'I could change my life to better suit your mood', "Because you're so smooth"]

The new list contains each line of the original string as its own smaller string. Also, notice that Python automatically escaped the ' character in the first line and adjusted to double quotation marks to allow the apostrophe on last line when it created the new list.

**Instructions**

**1.**

The organization has sent you over the full text for William Carlos Williams poem *Spring Storm*. They want you to break the poem up into its individual lines.

Create a list called spring\_storm\_lines that contains a string for each line of *Spring Storm*.

Checkpoint 2 Passed

Hint

You will have to use .split() and the escape character for a newline, \n.

**script.py**

spring\_storm\_text = \

"""The sky has given over

its bitterness.

Out of the dark change

all day long

rain falls and falls

as if it would never end.

Still the snow keeps

its hold on the ground.

But water, water

from a thousand runnels!

It collects swiftly,

dappled with black

cuts a way for itself

through green ice in the gutters.

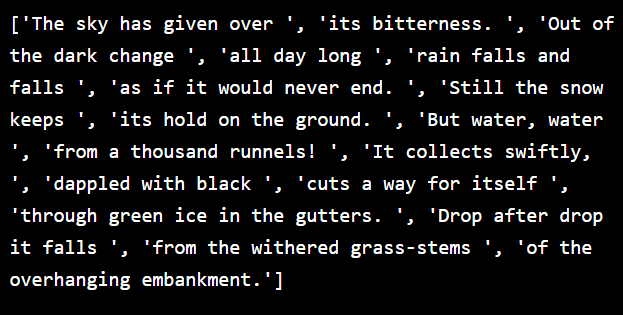
Drop after drop it falls

from the withered grass-stems

of the overhanging embankment."""

spring\_storm\_lines = spring\_storm\_text.split('\n')

print(spring\_storm\_lines)

****