

Theory: Basic string methods

🕒 19 minutes 10 / 10 problems solved

Start practicing

11613 users solved this topic. Latest completion was about 1 hour ago.

As you already know, the string is one of the most important data types in Python. To make working with strings easier, Python has many special built-in string methods. We are about to learn some of them.

An important thing to remember, however, is that the string is an **immutable** data type! It means that you cannot just change the string in-place, so most string methods **return a copy** of the string (with several exceptions). To save the changes made to the string for later use you need to create a **new variable** for the copy that you made or **assign the same name** to the copy. So, what to do with the output of the methods depends on whether you are going to use the original string or its copy later.

§1. "Changing" a string

The first group of string methods consists of the ones that "change" the string in a specific way, that is they return the copy with some changes made.

The syntax for calling a method is as follows: a string is given first (or the name of a variable that holds a string), then comes a period followed by the method name and parentheses in which arguments are listed.

Here's a list of common string methods of that kind:

- `str.replace(old, new[, count])` replaces all occurrences of the `old` string with the `new` one. The `count` parameter is optional, and if specified, only the first `count` occurrences are replaced in the given string.
- `str.upper()` converts all characters of the string to the upper case.
- `str.lower()` converts all characters of the string to the lower case.
- `str.title()` converts the first character of each word to upper case.
- `str.swapcase()` converts upper case to lower case and vice versa.
- `str.capitalize()` changes the first character of the string to the title case and the rest to the lower case.

And here's an example of how these methods are used (note that we don't save the result of every method):

```
1 message = "bonjour and welcome to Paris!"
2
3 print(message.upper()) # BONJOUR AND WELCOME TO PARIS!
4 # `message` is not changed
5 print(message) # bonjour and welcome to Paris!
6
7 title_message = message.title()
8 # `title_message` contains a new string with all words capitalized
9 print(title_message) # Bonjour And Welcome To Paris!
10
11
12 print(message.replace("Paris", "Lyon")) # bonjour and welcome to Lyon!
13
14 replaced_message = message.replace("o", "!", 2)
15
16 print(replaced_message) # b!nj!ur and welcome to Paris!
17
18
19 # again, the source string is unchanged, only its copy is modified
20
21 print(message) # bonjour and welcome to Paris!
```

Current topic:

✓ [Basic string methods](#)

Stage 113★...

Table of contents:

- [1 Basic string methods](#)
- [§1. "Changing" a string](#)
- [§2. "Editing" a string](#)
- [§3. Conclusions](#)
- [Feedback & Comments](#)

§2. "Editing" a string

Often, when you read a string from somewhere (a file or the input) you need to edit it so that it contains only the information you need. For instance, the input string can have a lot of unnecessary whitespaces or some trailing combinations of characters. The "editing" methods that can help with that are `strip()`, `rstrip()` and `lstrip()`.

- `str.lstrip([chars])` removes the leading characters (i.e. characters from the left side). If the argument `chars` isn't specified, leading whitespaces are removed.
- `str.rstrip([chars])` removes the trailing characters (i.e. characters from the right side). The default for the argument `chars` is also whitespace.
- `str.strip([chars])` removes both the leading and the trailing characters. The default is whitespace.

The `chars` argument, when specified, is a string of characters that are meant to be removed from the very end or beginning of the word (depending on the method you're using). See how it works:

```
1 whitespace_string = "    hey    "
2 normal_string = "incomprehensibilities"
3
4 # delete spaces from the left side
5 whitespace_string.lstrip() # "hey    "
6
7 # delete all "i" and "s" from the left side
8 normal_string.lstrip("is") # "ncomprehensibilities"
9
10
11 # delete spaces from the right side
12
13 whitespace_string.rstrip() # "    hey"
14
15
16 # delete all "i" and "s" from the right side
17
18 normal_string.rstrip("is") # "incomprehensibilitie"
19
20
21 # no spaces from both sides
22
23 whitespace_string.strip() # "hey"
24
25
26 # delete all trailing "i" and "s" from both sides
27
28 normal_string.strip("is") # "ncomprehensibilitie"
```

Keep in mind that the methods `strip()`, `lstrip()` and `rstrip()` get rid of all possible combinations of specified characters:

```
1 word = "Mississippi"
2
3 # starting from the right side, all "i", "p", and "s" are removed:
4 print(word.rstrip("ips")) # "M"
5
6
# the word starts with "M" rather than "i", "p", or "s", so no chars are removed from the left side:
7 print(word.lstrip("ips")) # "Mississippi"
8
9 # "M", "i", "p", and "s" are removed from both sides, so nothing is left:
10 print(word.strip("Mips")) # ""
```


Use them carefully, or you may end up with an empty string.

§3. Conclusions



Thus, we have considered the main methods for strings. Here is a brief recap:

- While working with string, you have to remember that strings are **immutable**, thus all the methods that "change" them only return the copy of a string with necessary changes.
- If you want to save the result of the method call for later use, you need to assign this result to a variable (either the same or the one with a different name).
- If you want to use this result only once, for example, in comparisons or just to print the formatted string, you are free to use the result on spot, as we did within `print()`.

 Report a typo

 Thanks for your feedback!

Write here how we could improve this theory



Start practicing

[Comments \(21\)](#)[Hints \(0\)](#)[Useful links \(1\)](#)[Show discussion](#)