Theory: Split and join

37 minutes 16 / 16 problems solved

Start practicing

6034 users solved this topic. Latest completion was about 3 hours ago.

In Python, strings and lists are quite similar. Firstly, they both pertain to **sequences**, although strings are limited to characters while lists can store data of different types. In addition, you can **iterate** both over strings and lists. However, sometimes you need to turn a string into a list or vice versa. Python has this kind of tools. The methods that will help you to accomplish this task are `split()`, `join()` and `splitlines()`.

§1. Split a string

The `split()` method divides a string into substrings by a **separator**. If the separator isn't given, whitespace is used as a default. The method returns a **list** of all the substrings and, notably, the separator itself is not included in any of the substrings.

```
# split example
definition = input()  # 'Coin of the realm is the legal money of the country'

definition.split()
# ['Coin', 'of', 'the', 'realm', 'is', 'the', 'legal', 'money', 'of', 'the', 'country']

definition.split("legal")
# ['Coin of the realm is the ', ' money of the country']
```

You can also specify how many times the split is going to be done with the `maxsplit` argument that comes after the separator. The number of elements in the resulting list will be equal to `maxsplit + 1`.

If the argument isn't specified, all possible splits are made.

```
# maxsplit example
definition = input()  # 'Coin of the realm is the legal money of the country'

definition.split("of", 1)
# ['Coin ', ' the realm is the legal money of the country']

definition.split("of")
# ['Coin ', ' the realm is the legal money ', ' the country']
```

If the separator doesn't occur in the string, then the result of the method is a list with the original string as its only element:

```
definition = input()  # 'Coin of the realm is the legal money of the country'

definition.split("hi!")  # wrong separator
# ['Coin of the realm is the legal money of the country']
```

Thus, in all cases `split()` allows us to convert a string into a list.

It may also be useful to read input directly into several variables with `split()`:

```
name, surname = input().split()  # Forrest Gump

print(name)     # Forrest
print(surname)  # Gump
```

It's pretty efficient when you know the exact number of input values. In case you don't, it's likely to result in `ValueError` with a message telling you either that there are too many values to unpack or not enough of them. So keep that in mind!

Current topic:

✓ Split and join Stage 1 8★

Topic depends on:

✓ Escape sequences Stage 1 8★

✓ Basic string methods Stage 1 13★

✓ For loop Stage 1 12★

Topic is required for:

✓ Defaultdict and Counter

Working with CSV

Processing requests Stage 1

Table of contents:

§2. Join a list

The `join()` method is used to create a string out of a collection of strings. However, its use has a number of limitations. First, the argument of the method must be an **iterable object** with strings as its elements. And second, the method must be applied to a **separator**: a string that will separate the elements in a resulting string object. See below the examples of that:

```
word_list  = ["dog", "cat", "rabbit", "parrot"]

" ".join(word_list)  # "dog cat rabbit parrot"
"".join(word_list)  # "dogcatrabbitparrot"
"_".join(word_list)  # "dog_cat_rabbit_parrot"
" and ".join(word_list)  # "dog and cat and rabbit and parrot"
```

Note that this method only works if the elements in the iterable object are **strings**. If, for example, you want to create a string of integers, it will not work. In this case, you need to convert the integers into strings explicitly or just work with strings right from the outset.

```
int_list = [1, 2, 3]
" ".join(int_list)  # TypeError!

str_list = ["1", "2", "3"]
" ".join(str_list)  # "1 2 3"
```

§3. Split multiple lines

The `splitlines()` method is similar to `split()`, but it is used specifically to split the string by the line boundaries. There are many escape sequences that signify the end of the line, but the `split()` method can only take one separator. So this is where the `splitlines()` method comes in handy:

```
# splitlines example
long_text = 'first line\nsecond line\rthird line\r\nfourth line'

long_text.splitlines()
# ['first line', 'second line', 'third line', 'fourth line']
```

The method has an optional argument `keepends` that has a True or False value. If `keepends = True` linebreaks are included in the resulting list:

```
# keepends
long_text = 'first line\nsecond line\rthird line\r\nfourth line'

long_text.splitlines(keepends=True)
# ['first line\n', 'second line\r', 'third line\r\n', 'fourth line']
```

You can also use several string methods at once. It is called **chaining,** and it works because most of the string methods return a copy of the original string:

```
# chaining example
sent = input()  # "Mary had a little lamb"
new_sent = sent.lower().split()
# ["mary", "had", "a", "little", "lamb"]
```

But do not get carried away, because the length of a line should be no more than 79 characters, and we definitely do not want to break PEP 8!

§4. Conclusion

We have learned how to convert strings to lists via the `split()` and `splitlines()` methods, and how to get strings back from lists via the `join()` method. As a recap, consider the following:

- Splitting and joining methods **do not change** the original string.
- If you need to use the "changed" string several times, you need to assign the result of the respective method to a variable.

- If you need to use this result only once, you can work with it on spot, for example, `print()` it.
- There are a lot of parameters in string methods. You can check the documentation if you need to fine-tune your program.

Report a typo

459 users liked this theory. 9 didn't like it. **What about you?**

😍 🙂 😐 🙁 😡

Start practicing

Comments (6) Hints (0) Useful links (0) Show discussion