Python → Data types and operations → Strings → Escape sequences

Theory: Escape sequences

⊘ 11 minutes   7 / 8 problems solved

Start practicing

Let's recall the problems that you may encounter using the `print()` function.

Nobody likes to write long forms of words, so here comes apostrophe. Be careful, though, when using an apostrophe in strings in Python, you might get an error message. Let's take a look at the example below:

```
1   # Bad example
2
3   warning = 'That's my car'
4   print(warning)
```

The sentence seems to look fine, but Python will show you an error message « `EOL while scanning string literal` ». Why did that error occur? The abbreviation "EOL" stands for "End-of-line" and it means that Python went through the text and didn't find the end of the string. The sentence was divided into two parts. The first one is `"That"` and the second one is `"s my car"`.

§1. What is an escape sequence?

To avoid the described problem you should use **escape sequences**.

All escape sequences start with a backslash `\`, which is interpreted as an **escape character**. A backslash indicates that one or more characters must be handled in a special way or that the next character after it has a different meaning.

Add an escape character to our example and see that the quote is now interpreted as the literal symbol of a single quote without any errors.

```
1   # Better example
2
3   warning = 'That\'s my car'
4   print(warning)  # That's my car
```

> Don't forget that single quotes and an apostrophe in the same sentence are a bad style! According to PEP8, it is better to use double quotes in these cases.

A backslash can be used in a combination with another symbol to do some particular things. For example, if you use `\b` in the middle of the string and then try to print it, you won't see the character before `\b`. The combination `\b` means a backspace character:

```
1   print('deleted\b sign')  # delete sign
```

Take a look at some other escape sequences:

- `\n` – newline
- `\t` – horizontal tabulation
- `\r` – moves all characters after `\r` to the beginning of the line, overwriting as many characters as moved.

The use of escape sequences in a string is always the same. Nevertheless, we will give more examples in the next section.

So, what if you need to print the backslash itself?

```
1   print('\')  # SyntaxError: EOL while scanning string literal
```

The error happens because Python expects an escape sequence that is not there. In this case, we must use a double backslash `\\`

Current topic:

✓ Escape sequences `Stage 1` 8★ ⋯

Topic depends on:

✓ Quotes and multi-line strings `Stage 1` 12★ ⋯

Topic is required for:

✓ Split and join `Stage 1` 8★ ⋯

Escaping in regexps ⋯

```
1 │ print('\\')  # \
```

You add an extra `\` to tell Python that the next `\` should **not** be interpreted as the start of an escape sequence.

Now if you want to print text that contains `\`, you can double it. For example, this is useful when you need to print literally `\n`, because `print('\n')` will only output a new blank line. Double backslash will help you in such situations!

```
1 │ print('\\n')  # \n
```

And you can also write:

```
1 │ metal = '\m/'
2 │ print(metal)  # \m/
```

Why is everything correct in such a case? Our string with `\` printed correctly, because `\m` is not an escape expression. Therefore, no formatting has occurred. One more example:

```
1 │ face = '\^_^/'
2 │ print(face)  # \^_^/
```

The function `repr()` returns a printable representation of this string, thus, escape sequences are visible.

```
1 │ print(repr(face))  # '\\^_^/'
```

When we want to know the length of a string, escape sequences are also taken into account:

```
1 │ print(len(repr(face)))  # 8
```

§2. Other examples

Let's consider an example with the escape sequence `\n`:

```
1 │ # The new line
2 │
3 │ print('Hello \nWorld!')
```

The `\n` combination starts a new line, so you will see the following output:

```
1 │ Hello
2 │ World!
```

The next example shows the escape sequence `\t`. As it was said above, `\t` is used for tabulation. If you put it in the middle of a string, the two parts of the string will be divided by some blank space that is called tabulation. It is quite useful when you work with a text.

```
1 │ # The tabulation
2 │
3 │ print('Hello\tWorld!')  # Hello   World!
```

Another escape sequence that can be useful while you are working with text is `\r`. The common name for this escape sequence is a carriage return. It moves characters after `\r` to the beginning of the line, replacing the exact number of old characters. That is, if the length of the string is longer before this escape sequence, then only the required number of characters is rewritten.

```
1 │ # The characters removal
2 │
3 │ print("Hello, dear \rWorld!")  # World! dear
```

Please note that the string length remains the same!

```
1   print(len("Hello, dear \rWorld!"))  # 19
```

Escape sequences are simple to use, aren't they? Let's talk more about the length of strings. For example:

```
1    # Comparing the lengths
2
3    greeting = 'Hello, John'
4    nice_greeting = 'Hello, \nJohn'
5
6    print(greeting)
7    # Hello, John
8    print(nice_greeting)
9    # Hello,
1
0    # John
1
1
1
2    print(len(greeting))        # 11
1
3    print(len(nice_greeting))  # 12
```

After calling the `len()` function, we can see that the length of the string with an escape sequence (in this case `\n`) is greater.

Be careful while working with strings because the function `print()` doesn't show escape sequences.

Thus, we have introduced basic escape sequences so that you can work with them in Python strings!

🖹 Report a typo

525 users liked this theory. **17** didn't like it. **What about you?**

😍   🙂   😐   🙁   😡
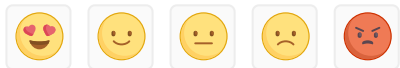
Start practicing

This content was created over 1 year ago and updated 9 days ago. Share your feedback below in comments to help us improve it!

Comments (8)   Hints (1)   Useful links (0)   Show discussion