Python Regex Non-capturing Group

Summary: in this tutorial, you'll learn about the Python regex non-capturing group to create a group but don't want to store it in the groups of the match.

Introduction to the Python regex non-capturing group

Regular expressions have two types of groups:

- Capturing groups
- Non-capturing groups

So far, you learned how to use a capturing group (https://www.pythontutorial.net/python-regex/python-regex-capturing-group/) to extract information from a bigger match or rematch the previous matched group using a backreference (https://www.pythontutorial.net/python-regex/python-regex-backreferences/) .

To do that, you create a capturing group, you place a pattern (or a rule) inside the parentheses, for example:

```
(X)
```

This syntax captures whatever match `X` inside the match so that you can access it via the `group()` method of the `Match` object.

Sometimes, you may want to create a group but don't want to capture it in the groups of the match. To do that, you can use a non-capturing group with the following syntax:

```
(?:X)
```

Python Regex no-capturing group example

The following example illustrates how to use the capturing groups to capture the major and minor versions of Python in the string `"Python 3.10"`

```
import re

s = 'Python 3.10'
pattern = '(\d+)\.(\d+)'

match = re.search(pattern, s)

# show the whole match
print(match.group())

# show the groups
for group in match.groups():
    print(group)
```

Output:

```
3.10
3
10
```

The following pattern matches one or more digits followed by the literal string (.) and one or more digits:

```
(\d+)\.(\d+)
```

It has two capturing groups. They capture the digits before and after the literal (.):

```
3
10
```

Suppose you don’t want to capture the digits before the literal character (.), you can use a non-capturing group like this:

```
import re

s = 'Python 3.10'
pattern = '(?:\d+)\.(\d+)'

match = re.search(pattern, s)

# show the whole match
print(match.group())

# show the groups
for group in match.groups():
    print(group)
```

Output:

```
3.10
10
```

In this example, we use the non-capturing group for the first group:

```
(?:\d+)
```

To capture the minor version only, you can ignore the non-capturing group in the first place like this:

```
import re

s = 'Python 3.10'
pattern = '\d+\.(\d+)'
match = re.search(pattern, s)

# show the whole match
print(match.group())

# show the groups
for group in match.groups():
    print(group)
```

Output:

```
3.10
10
```

So why do you use the non-capturing group anyway? the reason for using the non-capturing group is to save memory, as the regex engine doesn't need to store the groups in the buffer.

Summary

- Use the regex non-capturing group to create a group but don't save it in the groups of the match.