Python Regex Backreferences

Summary: in this tutorial, you'll learn about Python regex backreferences and how to apply them effectively.

Introduction to the Python regex backreferences

Backreferences like variables (https://www.pythontutorial.net/python-basics/python-variables/) in Python. The backreferences allow you to reference capturing groups (https://www.pythontutorial.net/python-regex/python-regex-capturing-group/) within a regular expression (https://www.pythontutorial.net/python-regex/python-regular-expressions/) .

The following shows the syntax of a backreference:

```
\N
```

Alternatively, you can use the following syntax:

```
\g<N>
```

In this syntax, `N` can be 1, 2, 3, etc. that represents the corresponding capturing group.

> Note that the `\g<0>` refer to the entire match, which has the same value as the `match.group(0)` .

Suppose you have a string with the duplicate word `Python` like this:

```
s = 'Python Python is awesome'
```

And you want to remove the duplicate word (`Python`) so that the result string will be:

```
Python is awesome
```

To do that, you can use a regular expression with a backreference.

First, match a word with one or more characters and one or more space:

```
'\w+\s+'
```

Second, create a capturing group that contains only the word characters:

```
'(\w+)\s+'
```

Third, create a backreference that references the first capturing group:

```
'(\w+)\s+\1'
```

In this pattern, the `\1` is a backreference that references the (`\w+`) capturing group.

Finally, replace the entire match with the first capturing group using the `sub()` function from the `re` module:

```
import re
```

```
s = 'Python Python is awesome'

new_s = re.sub(r'(\w+)\s+\1', r'\1', s)

print(new_s)
```

Output:

```
Python is awesome
```

More Python regex backreference examples

Let’s take some more examples of using backreferences.

1) Using Python regex backreferences to get text inside quotes

Suppose you want to get the text within double quotes:

```
"This is regex backreference example"
```

Or single quote:

```
'This is regex backreference example'
```

But not mixed of single and double-quotes. The following will not match:

```
'not match"
```

To do this, you may use the following pattern:

```
'[\'"](.*?)[\'"]'
```

However, this pattern will match text that starts with a single quote (') and ends with a double quote (") or vice versa. For example:

```
import re

s = '"Python\'s awsome". She said'
pattern = '[\'"].*?[\'"]'

match = re.search(pattern, s)

print(match.group(0))
```

It returns the `"Python'` not `"Python's awesome"` :

```
"Python'
```

To fix it, you can use a backreference:

```
r'([\'"]).*?\1'
```

The backreference `\1` refers to the first capturing group. So if the subgroup starts with a single quote, the `\1` will match the single quote. And if the subgroup starts with a double-quote, the `\1` will match the double-quote.

For example:

```
import re

s = '"Python\'s awsome". She said'
pattern = r'([\'"])(.*?)\1'

match = re.search(pattern, s)
print(match.group())
```

Output:

```
"Python's awsome"
```

2) Using Python regex backreferences to find words that have at least one consecutive repeated character

The following example uses a backreference to find words that have at least one consecutive repeated character:

```
import re

words = ['apple', 'orange', 'strawberry']
pattern = r'\b\w*(\w)\1\w*\b'

results = [w for w in words if re.search(pattern, w)]

print(results)
```

Output:

```
['apple', 'strawberry']
```

Summary

- Use a backreference `\N` to reference the capturing group `N` in a regular expression.