Python Regex Capturing Group

Summary: in this tutorial, you'll learn about Python regex capturing groups to create subgroups for a match.

Introduction to the Python regex capturing groups

Suppose you have the following path that shows the news with the id 100 on a website:

```
news/100
```

The following regular expression (https://www.pythontutorial.net/python-regex/python-regular-expressions/) matches the above path:

```
\w+/\d+
```

> Note that the above regular expression also matches any path that starts with one or more word characters, e.g., `posts` , `todos` , etc. not just `news` .

In this pattern:

- `\w+` is a word character set (https://www.pythontutorial.net/python-regex/python-regex-character-set/) with a quantifier (https://www.pythontutorial.net/python-regex/python-regex-quantifiers/) (+) that matches one or more word characters.
- `/` mathes the forward slash `/` character.
- `\d+` is digit character set with a quantfifer (`+`) that matches one or more digits.

The following program uses the \w+/\d+ pattern to match the string ' `news/100'` :

```
import re

s = 'news/100'
pattern = '\w+/\d+'

matches = re.finditer(pattern,s)
for match in matches:
    print(match)
```

Output:

```
<re.Match object; span=(0, 8), match='news/100'>
```

It shows one match as expected.

To get the `id` from the path, you use a capturing group. To define a capturing group for a pattern, you place the rule in parentheses `:`

```
(rule)
```

For example, to create a capturing group that captures the `id` from the path, you use the following pattern:

```
'\w+/(\d+)'
```

In this pattern, we place the rule `\d+` inside the parentheses `()`. If you run the program with the new pattern, you'll see that it displays one match:

```
import re

s = 'news/100'
pattern = '\w+/(\d+)'

matches = re.finditer(pattern, s)
for match in matches:
    print(match)
```

Output:

```
<re.Match object; span=(0, 8), match='news/100'>
```

To get the capturing groups from a match, you the `group()` method of the `Match` object:

```
match.group(index)
```

The `group(0)` will return the entire match while the `group(1)`, `group(2)`, etc., return the first, second, ... group.

The `lastindex` property of the `Match` object returns the last index of all subgroups. The following program shows the entire match (`group(0)`) and all the subgroups:

```
import re

s = 'news/100'
pattern = '\w+/(\d+)'

matches = re.finditer(pattern, s)
for match in matches:
```

```
    for index in range(0, match.lastindex + 1):
        print(match.group(index))
```

Output:

```
news/100
100
```

In the output, the `news/100` is the entire match while `100` is the subgroup.

If you want to capture also the resource (`news`) in the path (`news/100`), you can create an additional capturing group like this:

```
'(\w+)/(\d+)'
```

In this pattern, we have two capturing groups one for `\w+` and the other for `\d+` . The following program shows the entire match and all the subgroups:

```
import re

s = 'news/100'
pattern = '(\w+)/(\d+)'

matches = re.finditer(pattern, s)
for match in matches:
    for index in range(0, match.lastindex + 1):
        print(match.group(index))
```

Output:

```
news/100
news
100
```

In the output, the `news/100` is the entire match while `news` and `100` are the subgroups.

Named capturing groups

By default, you can access a subgroup in a match using an index, for example, `match.group(1)` . Sometimes, accessing a subgroup by a meaningful name is more convenient.

You use the named capturing group to assign a name to a group. The following shows the syntax for assigning a name to a capturing group:

```
(?P<name>rule)
```

In this syntax:

- `()` indicates a capturing group.
- `?P<name>` specifies the name of the capturing group.
- `rule` is a rule in the pattern.

For example, the following creates the names:

```
'(?P<resource>\w+)/(?P<id>\d+)'
```

In this syntax, the `resource` is the name for the first capturing group and the `id` is the name for the second capturing group.

To get all the named subgroups of a match, you use the `groupdict()` method of the `Match` object. For example:

```
import re

s = 'news/100'
pattern = '(?P<resource>\w+)/(?P<id>\d+)'

matches = re.finditer(pattern, s)
```

```
for match in matches:
    print(match.groupdict())
```

Output:

```
{'resource': 'news', 'id': '100'}
```

In this example, the `groupdict()` method returns a dictionary where the keys are group names and values are the subgroups.

More named capturing group example

The following pattern:

```
\w+/d{4}/d{2}/d{2}
```

matches this path:

```
news/2021/12/31
```

And you can add the named capturing groups to the pattern like this:

```
'(?P<resource>\w+)/(?P<year>\d{4})/(?P<month>\d{1,2})/(?P<day>\d{1,2})'
```

This program uses the patterns to match the path and shows all the subgroups:

```
import re

s = 'news/2021/12/31'
pattern = '(?P<resource>\w+)/(?P<year>\d{4})/(?P<month>\d{1,2})/(?P<day>\d{1,2})'

matches = re.finditer(pattern, s)
```

```
for match in matches:
    print(match.groupdict())
```

Output:

```
{'resource': 'news', 'year': '2021', 'month': '12', 'day': '31'}
```

Summary

- Place a rule of a pattern inside parentheses () to create a capturing group.
- Use the `group()` method of the `Match` object to get the subgroup by an index.
- Use the `(?P<name>rule)` to create a named capturing group for the rule in a pattern.
- Use the `groupdict()` method of the `Match` object to get the named subgroups as a dictionary.