Python Regex Quantifiers

Summary: in this tutorial, you'll learn how to use Python regex quantifiers to define how many times a character or a character set can be repeated.

Introduction to Python regex quantifiers

In regular expressions (https://www.pythontutorial.net/python-regex/python-regular-expressions/) , quantifiers match the preceding characters or character sets a number of times. The following table shows all the quantifiers and their meanings:

Quantifier	Name	Meaning
*	Asterisk	Match its preceding element zero or more times.
+	Plus	Match its preceding element one or more times.
?	Question Mark	Match its preceding element zero or one time.
{ *n* }	Curly Braces	Match its preceding element exactly n times.
{ *n* ,}	Curly Braces	Match its preceding element at least n times.

Quantifier	Name	Meaning
{ *n* , *m* }	Curly Braces	Match its preceding element from n to m times.

Match zero or more times (*)

The quantifier (*) matches its preceding element zero or more times. For example, the following program uses the * quantifier to match any string that ends with Python :

```
import re

s = """CPython, IronPython, and JPython
       are major Python's implementation"""

matches = re.finditer('\w*Python', s)

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

In this example:

- The \w matches any single word character.
- So the \w* matches zero or more word characters.
- Therefore, the \w*Python match any zero or more characters followed by the string Python .

As a result, the \w*Python pattern matches CPython , IronPython , JPython , and Python in the string:

```
<re.Match object; span=(0, 7), match='CPython'>
<re.Match object; span=(9, 19), match='IronPython'>
<re.Match object; span=(25, 32), match='JPython'>
<re.Match object; span=(51, 57), match='Python'>
```

Match one or more times (+)

The `+` quantifier matches its preceding element one or more times. For example, the `\d+` matches one or more digits.

The following example uses the `+` quantifier to match one or more digits in a string:

```
import re

s = "Python 3.10 was released in 2021"

matches = re.finditer('\d+', s)

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

Output:

```
<re.Match object; span=(7, 8), match='3'>
<re.Match object; span=(9, 11), match='10'>
<re.Match object; span=(28, 32), match='2021'>
```

Match zero or one time (?)

The `?` quantifier matches its preceding element zero or one time.

The following example uses the (?) quantifier to match both strings `color` and `colour` :

```
import re

s = "What color / colour do you like?"

matches = re.finditer('colou?r', s)
```

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

Output:

```
<re.Match object; span=(5, 10), match='color'>
<re.Match object; span=(13, 19), match='colour'>
```

In this example, the `u?` matches zero or one character `u`. Therefore, the `colou?r` pattern matches both `color` and `colour`

Match Exactly n Times: {n}

The `{n}` quantifier matches its preceding element exactly `n` times, where `n` is zero or a positive integer.

For example, the following program uses the quantifier `{n}` to match a time string in the `hh:mm` format:

```
import re

s = "It was 11:05 AM"

matches = re.finditer('\d{2}:\d{2}', s)

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

Output:

```
<re.Match object; span=(7, 12), match='11:05'>
```

In this example, the `\d{2}` matches exactly two digits. Therefore, the `\d{2}:\d{2}` matches a string that starts with two digits, a colon `:`, and ends with two digits.

Match at least n times: {n,}

The `{n,}` quantifier matches its preceding element at least `n` times, where `n` is zero or a positive integer.

For example, the following program uses the `{n, }` quantifier to match the date strings with the `m-d-yyyy` or `mm-dd-yyyy` format:

```
import re

s = "5-5-2021 or 05-05-2021 or 5/5/2021"

matches = re.finditer('\d{1,}-\d{1,}-\d{4}', s)

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

Output:

```
<re.Match object; span=(0, 8), match='5-5-2021'>
<re.Match object; span=(12, 22), match='05-05-2021'>
```

Match from n and m times: {n,m}

The `{n,m}` quantifier matches its preceding element at least `n` times, but no more than `m` times, where `n` and `m` are zero or a positive integer. For example:

```
import re

s = "5-5-2021 or 05-05-2021 or 5/5/2021"

matches = re.finditer('\d{1,2}-\d{1,2}-\d{4}', s)
```

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

Output:

```
<re.Match object; span=(0, 8), match='5-5-2021'>
<re.Match object; span=(12, 22), match='05-05-2021'>
```

In this example, the pattern `\d{1,2}` matches one or two digits. So the pattern `\d{1,2}-\d{1,2}-\d{4}` matches a date string in the `d-m-yyyy` or `dd-mm-yyyy` format.

Summary

- Quantifiers match their preceding elements a number of times.