Regular Expression in Python

Regular Expressions, often shortened as regex, are a sequence of characters used to check whether a pattern exists in a given text (string) or not. If you have ever used search engines, search and replace tools of word processors and text editors - you've already seen regular expressions in use. They are used at the server side to validate the format of email addresses or passwords during registration, used for parsing text data files to find, replace, or delete certain string, etc. They help in manipulating textual data, which is often a prerequisite for data science projects involving text mining.

In Python, regular expressions are supported by the re module. That means that if you want to start using them in your Python scripts, you must import this module with the help of **import**:

```
import re
```

The re library in Python provides several functions that make it a skill worth mastering.

This chapter will walk you through the important concepts of regular expressions with Python. You will start with **importing re** - Python library that supports regular expressions. Then you will see how **basic/ordinary characters** are used for performing matches, followed by **wild or special characters**. Next, you will learn about using **repetitions** in your regular expressions. You will also learn how to create **groups and named groups** within your search for ease of access to matches.

Basic Patterns: Ordinary Characters

You can easily tackle many basic patterns in Python using ordinary characters. Ordinary characters are the simplest regular expressions. They match themselves exactly and do not have a special meaning in their regular expression syntax.

Examples are 'A', 'a', 'X', '5'.

Ordinary characters can be used to perform simple exact matches:

```
import re

pattern = r"Cookie"
sequence = "Cookie"
if re.match(pattern, sequence):
    print("Match!")
else:
    print("Not a match!")
```

Output

```
Match!
```

Most alphabets and characters will match themselves, as you saw in the example.

The ***match()*** function returns a match object if the text matches the pattern. Otherwise, it returns None.

Wild Card Characters: Special Characters

Special characters are characters that do not match themselves as seen but have a special meaning when used in a regular expression. For simple understanding, they can be thought of as reserved metacharacters that denote something else and not what they look like.

Let us check out some examples to see the special characters in action...

But before you do, the examples below make use of two functions namely: ***search()*** and ***group()***.

With the **search** function, you scan through the given string/sequence, looking for the first location where the regular expression produces a match.

The **group** function returns the string matched by the re.

. - A period. Matches any single character except the newline character.

```
import re

print(re.search(r'Co.k.e', 'Cookie').group())
```

Output

```
Cookie
```

^ - A caret. Matches the start of the string.

```
import re

print(re.search(r'^Hello', "Hello World!").group())
```

Output

```
Hello
```

$ - Matches the end of string.

This is helpful if you want to make sure a document/sentence ends with certain characters.

```
import re

print(re.search(r'python$', "Python! Let's learn
python").group())
```

Output

```
python
```

[abc] - Matches a or b or c.

[a-zA-Z0-9] - Matches any letter from (a to z) or (A to Z) or (0 to 9).

```
import re

print(re.search(r'[0-6]', 'Number: 5').group())
```

Output

```
5
```

```
import re

# Matches any character except 5
print(re.search(r'Number: [^5]', 'Number: 0').group())
```

Output

```
Number: 0
```

\w - Lowercase 'w'. Matches any single letter, digit, or underscore.

\W - Uppercase 'W'. Matches any character not part of \w (lowercase w).

```
import re

print("Lowercase w:", re.search(r'Co\wk\we',
'Cookie').group())

# Matches any character except single letter, digit or
underscore
print("Uppercase W:", re.search(r'C\Wke', 'C@ke').group())

# Uppercase W won't match single letter, digit
print("Uppercase W won't match, and return:",
re.search(r'Co\Wk\We', 'Cookie'))
```

Output

```
Lowercase w: Cookie
Uppercase W: C@ke
Uppercase W won't match, and return: None
```

\s - Lowercase 's'. Matches a single whitespace character like: space, newline, tab, return.

\S - Uppercase 'S'. Matches any character not part of \s (lowercase s).

```
import re

print("Lowercase s:", re.search(r'Learn\spython', 'Learn
python').group())
print("Uppercase S:", re.search(r'cook\Se', "Let's eat
cookie").group())
```

Output

```
Lowercase s: Learn python
Uppercase S: cookie
```

\d - Lowercase d. Matches decimal digit 0-9.

\D - Uppercase d. Matches any character that is not a decimal digit.

```
import re

print("How many python courses do you want? ",
re.search(r'\d+', '2 python').group())
```

Output

```
How many python courses do you want?  2
```

\t - Lowercase t. Matches tab.

\n - Lowercase n. Matches newline.

\r - Lowercase r. Matches return.

\A - Uppercase a. Matches only at the start of the string. Works across multiple lines as well.

\Z - Uppercase z. Matches only at the end of the string.

\b - Lowercase b. Matches only the beginning or end of the word.

Repetitions

It becomes quite tedious if you are looking to find long patterns in a sequence. Fortunately, the **re** module handles repetitions using the following special characters:

+ - Checks if the preceding character appears one or more times starting from that position.

```
import re

print(re.search(r'Py+thon', 'Pyyyython').group())
```

Output

```
Pyyyython
```

*** - Checks if the preceding character appears zero or more times starting from that position.**

```
import re

print(re.search(r'Ca*o*kie', 'Cookie').group())
```

Output

```
Cookie
```

? - Checks if the preceding character appears exactly zero or one time starting from that position.

```
import re

print(re.search(r'Pythoi?n', 'Python').group())
```

Output

```
Python
```

But what if you want to check for an exact number of sequence repetition?

For example, checking the validity of a phone number in an application. **re** module handles this very gracefully as well using the following regular expressions:

{x} - Repeat exactly x number of times.

{x,} - Repeat at least x times or more.

{x, y} - Repeat at least x times but no more than y times.

```
import re

print(re.search(r'\d{9,10}', '0987654321').group())
```

Output

```
0987654321
```

Grouping in Regular Expressions

The **group** feature of regular expression allows you to pick up parts of the matching text. Parts of a regular expression pattern bounded by parenthesis **()** are called *groups*. The parenthesis does not change what the expression matches, but rather forms groups within the matched sequence. You have been using the **group()** function all along in this tutorial's examples. The plain **match.group()** without any argument is still the whole matched text as usual.

Let us understand this concept with a simple example. Imagine you were validating email addresses and wanted to check the user's name and host. This is when you would want to create separate groups within your matched text.

```
import re

statement = 'Please contact us at: info@xyz.com'
match = re.search(r'([\w\.-]+)@([\w\.-]+)', statement)
if statement:
    print("Email address:", match.group())  # The whole
matched text
    print("Username:", match.group(1))  # The username (group
1)
    print("Host:", match.group(2))  # The host (group 2)
```

Output

```
Email address: info@xyz.com
Username: info
Host: xyz.com
```