Python RegEx

In this tutorial, you will learn about regular expressions (RegEx), and use Python's re module to work with RegEx (with the help of examples).

A **Re**gular **Ex**pression (RegEx) is a sequence of characters that defines a search pattern. For example,

```
^a...s$
```

The above code defines a RegEx pattern. The pattern is: **any five letter string starting with *a* and ending with *s***.

A pattern defined using RegEx can be used to match against a string.

Expression	String	Matched?
`^a...s$`	`abs`	No match
	`alias`	Match
	`abyss`	Match
	`Alias`	No match
	`An abacus`	No match

Specify Pattern Using RegEx

To specify regular expressions, metacharacters are used. In the above example, `^` and `$` are metacharacters.

MetaCharacters

Metacharacters are characters that are interpreted in a special way by a RegEx engine. Here's a list of metacharacters:

[] . ^ $ * + ? {} () \ |

`[]` - Square brackets

Square brackets specifies a set of characters you wish to match.

Expression	String	Matched?
`[abc]`	`a`	1 match
	`ac`	2 matches
	`Hey Jude`	No match
	`abc de ca`	5 matches

Here, `[abc]` will match if the string you are trying to match contains any of the `a`, `b` or `c`.

You can also specify a range of characters using `-` inside square brackets.

- `[a-e]` is the same as `[abcde]`.
- `[1-4]` is the same as `[1234]`.
- `[0-39]` is the same as `[01239]`.

You can complement (invert) the character set by using caret `^` symbol at the start of a square-bracket.

- `[^abc]` means any character except *a* or *b* or *c*.
- `[^0-9]` means any non-digit character.

`.` - Period

A period matches any single character (except newline `'\n'`).

Expression	String	Matched?
`..`	`a`	No match
	`ac`	1 match
	`acd`	1 match
	`acde`	2 matches (contains 4 characters)

`^` - Caret

The caret symbol `^` is used to check if a string **starts with** a certain character.

Expression	String	Matched?
`^a`	`a`	1 match
	`abc`	1 match
	`bac`	No match
`^ab`	`abc`	1 match
	`acb`	No match (starts with `a` but not followed by `b`)

`$` - Dollar

The dollar symbol `$` is used to check if a string **ends with** a certain character.

Expression	String	Matched?
`a$`	`a`	1 match
	`formula`	1 match
	`cab`	No match

`*` - Star

The star symbol `*` matches **zero or more occurrences** of the pattern left to it.

Expression	String	Matched?
	`mn`	1 match
	`man`	1 match
`ma*n`	`maaan`	1 match
	`main`	No match (`a` is not followed by `n`)
	`woman`	1 match

`+` - **Plus**

The plus symbol `+` matches **one or more occurrences** of the pattern left to it.

Expression	String	Matched?
	`mn`	No match (no `a` character)
	`man`	1 match
`ma+n`	`maaan`	1 match
	`main`	No match (a is not followed by n)
	`woman`	1 match

`?` - **Question Mark**

The question mark symbol `?` matches **zero or one occurrence** of the pattern left to it.

Expression	String	Matched?
	`mn`	1 match
	`man`	1 match
`ma?n`	`maaan`	No match (more than one `a` character)
	`main`	No match (a is not followed by n)
	`woman`	1 match

`{}` - **Braces**

Consider this code: `{n,m}`. This means at least *n*, and at most *m* repetitions of the pattern left to it.

Expression	String	Matched?
	`abc dat`	No match
	`abc daat`	1 match (at `daat`)
`a{2,3}`	`aabc daaat`	2 matches (at `aabc` and `daaat`)
	`aabc daaaat`	2 matches (at `aabc` and `daaaat`)

Let's try one more example. This RegEx `[0-9]{2, 4}` matches at least 2 digits but not more than 4 digits

Expression	String	Matched?
	`ab123csde`	1 match (match at `ab123csde`)
`[0-9]{2,4}`	`12 and 345673`	3 matches (`12`, `3456`, `73`)
	`1 and 2`	No match

`|` - Alternation

Vertical bar `|` is used for alternation (`or` operator).

Expression	String	Matched?
	`cde`	No match
`a|b`	`ade`	1 match (match at `ade`)
	`acdbea`	3 matches (at `acdbea`)

Here, `a|b` match any string that contains either *a* or *b*

`()` - Group

Parentheses `()` is used to group sub-patterns. For example, `(a|b|c)xz` match any string that matches either *a* or *b* or *c* followed by *xz*

Expression	String	Matched?
	`ab xz`	No match
`(a|b|c)xz`	`abxz`	1 match (match at `abxz`)
	`axz cabxz`	2 matches (at `axzbc cabxz`)

`\` - Backslash

Backlash `\` is used to escape various characters including all metacharacters. For example,

`\$a` match if a string contains `$` followed by `a`. Here, `$` is not interpreted by a RegEx engine in a special way.

If you are unsure if a character has special meaning or not, you can put `\` in front of it. This makes sure the character is not treated in a special way.

Special Sequences

Special sequences make commonly used patterns easier to write. Here's a list of special sequences:

`\A` - Matches if the specified characters are at the start of a string.

Expression	String	Matched?
`\Athe`	`the sun`	Match
	`In the sun`	No match

`\b` - Matches if the specified characters are at the beginning or end of a word.

Expression	String	Matched?
`\bfoo`	`football`	Match
	`a football`	Match
	`afootball`	No match
`foo\b`	`the foo`	Match
	`the afoo test`	Match
	`the afootest`	No match

`\B` - Opposite of `\b`. Matches if the specified characters are **not** at the beginning or end of a word.

Expression	String	Matched?
`\Bfoo`	`football`	No match
	`a football`	No match
	`afootball`	Match
`foo\B`	`the foo`	No match
	`the afoo test`	No match
	`the afootest`	Match

`\d` - Matches any decimal digit. Equivalent to `[0-9]`

Expression	String	Matched?
`\d`	`12abc3`	3 matches (at `12abc3`)
	`Python`	No match

`\D` - Matches any non-decimal digit. Equivalent to `[^0-9]`

Expression	String	Matched?
`\D`	`1ab34"50`	3 matches (at `1ab34"50`)
	`1345`	No match

`\s` - Matches where a string contains any whitespace character. Equivalent to `[ \t\n\r\f\v]`.

Expression	String	Matched?
`\s`	`Python RegEx`	1 match
	`PythonRegEx`	No match

`\S` - Matches where a string contains any non-whitespace character. Equivalent to `[^ \t\n\r\f\v]`.

Expression	String	Matched?
`\S`	`a b`	2 matches (at `a b`)
		No match

`\w` - Matches any alphanumeric character (digits and alphabets). Equivalent to `[a-zA-Z0-9_]`. By the way, underscore _ is also considered an alphanumeric character.

Expression	String	Matched?
`\w`	`12&": ;c`	3 matches (at `12&": ;c`)
	`%"> !`	No match

`\W` - Matches any non-alphanumeric character. Equivalent to `[^a-zA-Z0-9_]`

Expression	String	Matched?
`\W`	`1a2%c`	1 match (at `1a2%c`)
	`Python`	No match

`\Z` - Matches if the specified characters are at the end of a string.

Expression	String	Matched?
`Python\Z`	`I like Python`	1 match
	`I like Python Programming`	No match
	`Python is fun.`	No match

Tip: To build and test regular expressions, you can use RegEx tester tools such as regex101. This tool not only helps you in creating regular expressions, but it also helps you learn it.

Now you understand the basics of RegEx, let's discuss how to use RegEx in your Python code.