Python Regular Expression Patterns List

Learn about the Python REGEX symbols

WE'LL COVER THE FOLLOWING

The following table lists the regular expression syntax that is available in Python. Note that any Regex can be concatenated to form new regular expressions; if `X` and `Y` are both regular expressions, then `XY` is also a regular expression.

Pattern	Description
`.`	Matches any single character except newline. Using `m` option allows it to match newline as well.
`^`	Matches the start of the string, and in `re.MULTILINE` (see the next lesson on how to change to multiline) mode also matches immediately after each newline.
`$`	Matches end of line. In `re.MULTILINE` mode also matches before a newline.
`[.]`	Matches any single character in brackets.

`[^.]`	Matches any single character not in brackets.
`*`	Matches 0 or more occurrences of preceding expression.
`+`	Matches 1 or more occurrence of preceding expression.
`?`	Matches 0 or 1 occurrence of preceding expression.
`{ n}`	Matches exactly `n` number of occurrences of preceding expression.
`{ n,}`	Matches `n` or more occurrences of preceding expression.
`{ n, m}`	Matches at least `n` and at most `m` occurrences of preceding expression. For example, `x{3,5}` will match from `3` to `5` `'x'` characters.
`x| y`	Matches either `x` or `y`.
`\d`	Matches digits. Equivalent to `[0-9]`.
`\D`	Matches nondigits.
`\w`	Matches word characters.
`\W`	Matches nonword characters.

`\z`	Matches end of string.
`\G`	Matches point where last match finished.
`\b`	Matches the empty string, but only at the beginning or end of a word. Boundary between word and non-word and `/B` is opposite of `/b`. Example `r"\btwo\b"` for searching two from `'one two three'`.
`\B`	Matches nonword boundaries.
`\n, \t`	Matches newlines, carriage returns, tabs, etc.
`\s`	Matches whitespace.
`\S`	Matches nonwhitespace.
`\A`	Matches beginning of string.
`\Z`	Matches end of string. If a newline exists, it matches just before newline.

Groups and Lookarounds

More details later:

Pattern	Description
`(re)`	Groups regular expressions and remembers matched text.

`(?: re)`	Groups regular expressions without remembering matched text. For example, the expression `(?:x{6})*` matches any multiple of six ‘`x`’ characters.
`(?#...)`	Comment.
`(?= ...)`	Matches if `...` matches next, but doesn’t consume any of the string. This is called a **lookahead assertion**. For example, `Scientific (?=Python)` will match `Scientific` only if it’s followed by `Python`.
`(?!...)`	Matches if `...` doesn’t match next. This is a negative lookahead assertion.
`(?<=...)`	Matches if the current position in the string is preceded by a match for `...` that ends at the current position.