Reg. Exp.	Matches . . .	Example
`\d`	A digit character. It is equivalent to the POSIX class `[[:digit:]]`.	The expression `^\(\d{3}\) \d{3}-\d{4}$` matches `(650) 555-0100` but does not match `650-555-0100`.
`\D`	A nondigit character. It is equivalent to the POSIX class `[^[:digit:]]`.	The expression `\w\d\D` matches `b2b` and `b2_` but does not match `b22`.
`\w`	A word character, which is defined as an alphanumeric or underscore (_) character. It is equivalent to the POSIX class `[[:alnum:]_]`. If you do not want to include the underscore character, you can use the POSIX class `[[:alnum:]]`.	The expression `\w+@\w+(\.\w+)+` matches the string `jdoe@company.co.uk` but not the string `jdoe@company`.
`\W`	A nonword character. It is equivalent to the POSIX class `[^[:alnum:]_]`.	The expression `\w+\W\s\w+` matches the string `to: bill` but not the string `to bill`.
`\s`	A whitespace character. It is equivalent to the POSIX class `[[:space:]]`.	The expression `\(\w\s\w\s\)` matches the string `(a b )` but not the string `(ab)`.
`\S`	A nonwhitespace character. It is equivalent to the POSIX class `[^[:space:]]`.	The expression `\(\w\S\w\S\)` matches the string `(abde)` but not the string `(a b d e)`.
`\A`	Only at the beginning of a string. In multi-line mode, that is, when embedded newline characters in a string are considered the termination of a line, `\A` does not match the beginning of each line.	The expression `\AL` matches only the first `L` character in the string `Line1\nLine2\n`, regardless of whether the search is in single-line or multi-line mode.
`\Z`	Only at the end of string or before a newline ending a string. In multi-line mode, that is, when embedded newline characters in a string are considered the termination of a line, `\Z` does not match the end of each line.	In the expression `\s\Z`, the `\s` matches the last space in the string `L i n e \n`, regardless of whether the search is in single-line or multi-line mode.
`\z`	Only at the end of a string.	In the expression `\s\z`, the `\s` matches the newline in the string `L i n e \n`, regardless of whether the search is in single-line or multi-line mode.
`*?`	The preceding pattern element 0 or more times ("nongreedy"). This quantifier matches the empty string whenever possible.	The expression `\w*?x\w` is "nongreedy" and so matches `abxc` in the string `abxcxd`. The expression `\w*x\w` is "greedy" and so matches `abxcxd in the string abxcxd`. The expression `\w*?x\w` also matches the string `xa`.

`+?`	The preceding pattern element 1 or more times ("nongreedy").	The expression `\w+?x\w` is "nongreedy" and so matches `abxc` in the string `abxcxd`. The expression `\w+x\w` is "greedy" and so matches `abxcxd in the string abxcxd`. The expression `\w+?x\w` does not match the string `xa`, but does match the string `axa`.
`??`	The preceding pattern element 0 or 1 time ("nongreedy"). This quantifier matches the empty string whenever possible.	The expression `a??aa` is "nongreedy" and matches `aa` in the string `aaaa`. The expression `a?aa` is "greedy" and so matches `aaa` in the string `aaaa`.
`{n}?`	The preceding pattern element exactly `n` times ("nongreedy"). In this case `{n}?` is equivalent to `{n}`.	The expression `(a|aa){2}?` matches `aa` in the string `aaaa`.
`{n,}?`	The preceding pattern element at least `n` times ("nongreedy").	The expression `a{2,}?` is "nongreedy" and matches `aa` in the string `aaaaa`. The expression `a{2,}` is "greedy" and so matches `aaaa`a.
`{n,m}?`	At least `n` but not more than `m` times ("nongreedy"). `{0,m}?` matches the empty string whenever possible.	The expression `a{2,4}?` is "nongreedy" and matches `aa` in the string `aaaaa`. The expression `a{2,4}` is "greedy" and so matches `aaaa`.