Regular Expression Basic Syntax Reference

Characters		
Character	**Description**	**Example**
Any character except `[\^$.|?*+()`	All characters except the listed special characters match a single instance of themselves. `{` and `}` are literal characters, unless they're part of a valid regular expression token (e.g. the `{n}` quantifier).	`a` matches `a`
`\` (backslash) followed by any of `[\^$.|?*+()` `{}`	A backslash escapes special characters to suppress their special meaning.	`\+` matches `+`
`\Q...\E`	Matches the characters between `\Q` and `\E` literally, suppressing the meaning of special characters.	`\Q+-*/\E` matches `+-*/`
`\xFF` where FF are 2 hexadecimal digits	Matches the character with the specified ASCII/ANSI value, which depends on the code page used. Can be used in character classes.	`\xA9` matches © when using the Latin-1 code page.
`\n`, `\r` and `\t`	Match an LF character, CR character and a tab character respectively. Can be used in character classes.	`\r\n` matches a DOS/Windows CRLF line break.
`\a`, `\e`, `\f` and `\v`	Match a bell character (`\x07`), escape character (`\x1B`), form feed (`\x0C`) and vertical tab (`\x0B`) respectively. Can be used in character classes.	
`\cA` through `\cZ`	Match an ASCII character Control+A through Control+Z, equivalent to `\x01` through `\x1A`. Can be used in character classes.	`\cM\cJ` matches a DOS/Windows CRLF line break.
Character Classes or Character Sets [abc]		
Character	**Description**	**Example**
`[` (opening square bracket)	Starts a character class. A character class matches a single character out of all the possibilities offered by the character class. Inside a character class, different rules apply. The rules in this section are only valid inside character classes. The rules outside this section are not valid in character classes, except for a few character escapes that are indicated with "can be used inside character classes".	
Any character except `^-]\` add that character to the possible matches for the character class.	All characters except the listed special characters.	`[abc]` matches `a`, `b` or `c`

`\` (backslash) followed by any of `^-]\`	A backslash escapes special characters to suppress their special meaning.	`[\^\]]` matches `^` or `]`
`-` (hyphen) except immediately after the opening `[`	Specifies a range of characters. (Specifies a hyphen if placed immediately after the opening `[`)	`[a-zA-Z0-9]` matches any letter or digit
`^` (caret) immediately after the opening `[`	Negates the character class, causing it to match a single character *not* listed in the character class. (Specifies a caret if placed anywhere except after the opening `[`)	`[^a-d]` matches `x` (any character except a, b, c or d)
`\d`, `\w` and `\s`	Shorthand character classes matching digits, word characters (letters, digits, and underscores), and whitespace (spaces, tabs, and line breaks). Can be used inside and outside character classes.	`[\d\s]` matches a character that is a digit or whitespace
`\D`, `\W` and `\S`	Negated versions of the above. Should be used only outside character classes. (Can be used inside, but that is confusing.)	`\D` matches a character that is not a digit
`[\b]`	Inside a character class, `\b` is a backspace character.	`[\b\t]` matches a backspace or tab character

Dot

Character	Description	Example
`.` (dot)	Matches any single character except line break characters \r and \n. Most regex flavors have an option to make the dot match line break characters too.	`.` matches `x` or (almost) any other character

Anchors

Character	Description	Example
`^` (caret)	Matches at the start of the string the regex pattern is applied to. Matches a position rather than a character. Most regex flavors have an option to make the caret match after line breaks (i.e. at the start of a line in a file) as well.	`^.` matches `a` in `abc\ndef`. Also matches `d` in "multi-line" mode.
`$` (dollar)	Matches at the end of the string the regex pattern is applied to. Matches a position rather than a character. Most regex flavors have an option to make the dollar match before line breaks (i.e. at the end of a line in a file) as well. Also matches before the very last line break if the string ends with a line break.	`.$` matches `f` in `abc\ndef`. Also matches `c` in "multi-line" mode.
`\A`	Matches at the start of the string the regex pattern is applied to. Matches a position rather than a character. Never matches after line breaks.	`\A.` matches `a` in `abc`

`\Z`	Matches at the end of the string the regex pattern is applied to. Matches a position rather than a character. Never matches before line breaks, except for the very last line break if the string ends with a line break.	`.\Z` matches `f` in `abc\ndef`
`\z`	Matches at the end of the string the regex pattern is applied to. Matches a position rather than a character. Never matches before line breaks.	`.\z` matches `f` in `abc\ndef`

Word Boundaries

Character	Description	Example
`\b`	Matches at the position between a word character (anything matched by `\w`) and a non-word character (anything matched by `[^\w]` or `\W`) as well as at the start and/or end of the string if the first and/or last characters in the string are word characters.	`.\b` matches `c` in `abc`
`\B`	Matches at the position between two word characters (i.e the position between `\w\w`) as well as at the position between two non-word characters (i.e. `\W\W`).	`\B.\B` matches `b` in `abc`

Alternation

Character	Description	Example
`|` (pipe)	Causes the regex engine to match either the part on the left side, or the part on the right side. Can be strung together into a series of options.	`abc|def|xyz` matches `abc`, `def` or `xyz`
`|` (pipe)	The pipe has the lowest precedence of all operators. Use grouping to alternate only part of the regular expression.	`abc(def|xyz)` matches `abcdef` or `abcxyz`

Quantifiers

Character	Description	Example
`?` (question mark)	Makes the preceding item optional. Greedy, so the optional item is included in the match if possible.	`abc?` matches `ab` or `abc`
`??`	Makes the preceding item optional. Lazy, so the optional item is excluded in the match if possible. This construct is often excluded from documentation because of its limited use.	`abc??` matches `ab` or `abc`
`*` (star)	Repeats the previous item zero or more times. Greedy, so as many items as possible will be matched before trying permutations with less matches of the preceding item, up to the point where the preceding item is not matched at all.	`".*"` matches `"def" "ghi"` in `abc "def" "ghi" jkl`
`*?` (lazy star)	Repeats the previous item zero or more times. Lazy, so the engine first attempts to skip the previous item, before trying permutations with ever increasing matches of the preceding item.	`".*?"` matches `"def"` in `abc "def" "ghi" jkl`
`+` (plus)	Repeats the previous item once or more. Greedy, so as many items as possible will be matched before trying permutations with less matches of the preceding item, up to the point where the preceding item is matched only once.	`".+"` matches `"def" "ghi"` in `abc "def" "ghi" jkl`
`+?` (lazy plus)	Repeats the previous item once or more. Lazy, so the engine first	`".+?"` matches

	matches the previous item only once, before trying permutations with ever increasing matches of the preceding item.	`"def"` in `abc "def" "ghi" jkl`
`{n}` where n is an integer >= 1	Repeats the previous item exactly n times.	`a{3}` matches `aaa`
`{n,m}` where n >= 0 and m >= n	Repeats the previous item between n and m times. Greedy, so repeating m times is tried before reducing the repetition to n times.	`a{2,4}` matches `aaaa`, `aaa` or `aa`
`{n,m}?` where n >= 0 and m >= n	Repeats the previous item between n and m times. Lazy, so repeating n times is tried before increasing the repetition to m times.	`a{2,4}?` matches `aa`, `aaa` or `aaaa`
`{n,}` where n >= 0	Repeats the previous item at least n times. Greedy, so as many items as possible will be matched before trying permutations with less matches of the preceding item, up to the point where the preceding item is matched only n times.	`a{2,}` matches `aaaaa` in `aaaaa`
`{n,}?` where n >= 0	Repeats the previous item n or more times. Lazy, so the engine first matches the previous item n times, before trying permutations with ever increasing matches of the preceding item.	`a{2,}?` matches `aa` in `aaaaa`