# Meta characters in regex

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

# Concatenation of patterns

/cat/

# Alternation

/(cat|dog)/ – matches cat or dog

/(cat\|dog)/ – matches ‘cat|dog’

/\(cat\|dog\)/ – matches ‘(cat|dog)’

# Control character escapes

/n – line feed

/r – carriage return

/t – tab

# Ignoring case:

/launch/i – matches launch, Launch, LAUNCH etc.

# Character Classes

They are patterns that let you specify a set of characters that you want to match (as in this OR this OR…)

## Set of Characters

Patterns within [], ex: [abc].

Such a pattern matches a single occurrence of any of the characters between parentheses

Useful in case of validating user inputs ex: to validate that user input is a number between 1-5 /[12345]/

Useful for selecting upper/lower case selection ex: /[hH]oover/ matches hoover, Hoover but not HOOVER.

**Note**: While writing character classes, it is a good practice to group characters by type: digits, uppercase letters, lowercase letters, whitespaces and non-alphanumeric characters.

Concatenating character classes: /[abc][12]/

Number of metacharacters inside a character class dwindles to these: ^ / - [].

You can use ^ as a non-meta character if it isn’t the first character in the class and you can use – if it is the first character.

## Range of Characters

Represents natural sequence of characters in an abbreviated form.

Ex: [a-z], [0-9]

matching all hexadecimal numbers: /[0-9A-Fa-f]/

## Negated Classes

Matches all the characters not identified in the range.

Uses a ^ to turn a range to negation.

Ex: [^aeiou] matches any character except the lower-case vowels.

# Character class shortcuts

We use character classes so often that most regex engines have built-in shortcuts to use these classes.

## Any Character

/./ (a period)

Matches any character: alphanumeric, punctuation, whitespace, some control character etc.

Note: A .(period) inside the [] is a literal and does not match, any character!

## Whitespace

\s denotes a whitespace character

\S denotes a non-whitespace character

Whitespace characters are: space(‘ ‘), tab(\t), vertical tab(\v), carriage return(\r), line feed(\n), form feed(\f).

Thus /\s/ is equal to /[ \t\v\r\n\f]/

## Digits

\d matches any digit. [\d] is equivalent to [0-9]

\D matches any character that is not a digit: [^0-9]

## Alphanumerics/Word characters

\w matches any alpha numeric character including underscore: [0-9A-Za-z\_]

\W matches any character that is not an alphanumeric character or an underscore.

[See more](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Regular_expressions/Character_classes) character shortcuts (Javascript).

Use of character class shortcuts make your regex easy to type and read.

# Anchors

Tells regex engine, where the matches can begin and where they can end. They provide a way to limit how a regex matches a particular string.

In JS, the meta-characters ^ and $ are anchors that match the beginning (^) or ending ($) of a string.

Ex: /^c.t/ig matches “catastrophe” but not “wildcat” or even “cot\n”. If you want to include cot\n you can use the flag “\m” as /^c.t/mig

JS treats \n as a separate line when you use the /m flag.

## Word boundary (\b) and Non-word (\B) boundary

A word boundary occurs:

* + Between any pair of characters, one of which is a word character and another is not
  + At the beginning of a string if the first character is a word character
  + At the end of a string if the last character is a word character

A non-word boundary occurs

* + Between any pair of characters, both of which are word characters or both of which are non-word characters
  + At the beginning of a string if the first character is a non-word character
  + At the end of a string if the last character is a non-word character

# Quantifiers

\*:   
Matches 0 or more occurrences of pattern to its left. Ex: pattern /\b\d\d\d\d\*\b/ matches 3 or more continuous digits with a word boundary on either side.

+:   
Matches 1 or more occurrences of pattern to its left.

?:  
Matches 0 or 1 occurrence of a pattern. An optional pattern that occurs once or doesn’t occur at all.

## Ranges

To specify the repeat count more precisely, Ex: 9 digit phone number, words with 4 to 7 letters etc.

Range quantifiers use curly braces “{}” with one or two numbers and an optional comma

p{m}: matches precisely m occurrences of pattern p

p{m,}: matches m or more occurrences of pattern p

p{m,n}: matches m or more occurrences of pattern p but not more than n

## Greedy vs. Lazy quantifiers

All quantifiers we’ve seen so far are greedy, i.e they will match the longest pattern possible.

Ex: matching /a[abc]\*c/ against xabcbcbacy will match “abcbcbac”. Notice that it does not match abc , abcbc or ac.

If you want to do a lazy matching, by adding a ‘?’ after the main quantifier:

Ex: /a[abc]\*?c/ against xabcbcbacy will match both abc and ac.

# Capture Groups

Grouping a pattern within () and providing it with a backreference so that it can be reused later.

Ex: We want to capture text between quoted strings (‘’ or ““): we use regex /[‘”].\*?[‘”]/, however these would also capture ‘ ” and “ ’ which is not what we want.

To avoid this we can capture the first group that was matched and reuse it using capture groups and backreferencing: /([‘”]).\*?\1/.

Here /1 is a backreference. A regex can have multiple capture groups like /2, /3 ….

## String transformations

we use “.match()” method to match string with regex, but we can transform strings using “.replace()” method. Capture groups are mainly used for such transformations.