Search string - RegEx read direction - left to right

-->

/pattern/ - Match first occurrence if any.

/ pattern/g - global: Match all occurrences.

/pattern/m - multiline. - match line by line not as a hole document. In some language it is inserted as an option XXX.Multiline.

**Strategy**

Greedy Strategy - match as **much** as possible before giving control to the next expression part.

Lazy Strategy - match as **little** as possible before giving control to the next expression part.

**Metacharacters**

. - Match any single character except newline

\ - Escape the next Metacharacter and / in the expression.

[ - Begin a character set.

] - End a character set.

- - Range of characters. Only a Metacharacter inside the set.

^ - [^pattern] - Negative character set, when using in set.

**Repetition metacharacters**

\* - Zero or more times. Apply for preceding item.

+ - One or more times. Apply for preceding item.

? - Zero or one time. Apply for preceding item.

**Quantify metacharacters**

{ - Start quantified repetition of preceding item.

} - End quantified repetition of preceding item.

? - Make preceding quantifier lazy.

**Group metacharacter**

( - Start grouped expression.

) - End grouped expression.

**Alternation metacharacters**

| - Match previous or next expression.

**Anchor metacharacters - refer to a position not to a character**

^ - Start of string/line.

$ - End of string/line.

\A - Start of string, never end of line.

\Z - End of string, never end of line.

\b - Word boundary (start/end of word). Include \_

\B - Not a word boundary.

**Backreference**

\1 through \9 - Backreference for position 1 to 9. This is reference not to the expression, but to the

value from the expression.

or

$1 through $9

?: - Specify a non-capturing group

**Lookaround Assertions**

**Lookahead Assertion**

?= - Positive lookahead assertion

?! - Negative lookahead assertion

**Lookbehind Assertion**

?<= - Positive lookbehind assertion

?<! - Negative lookbehind assertion

**Unicode**

\u - Unicode indicator.

\x - Unicode indicator in PHP, Perl

\X - Unicode wildcard - Match any single character and line brakes. in PHP and Perl

\p - Unicode property - Match character with property - /pattern\p{L}/

Properties are:

L - Letter; M - Mark; Z - Separator; S - Symbol; N - Number; P - Punctuation; C - Other

\P - Unicode not property - Match character that do not have property - /pattern\P{L}/

**Character set**

Metacharacter inside a set are already escaped except **]** **-** **^** **\** . Those need to be escaped and others don't.

[pattern] - Match only one character which **is** part of the pattern in the set. Order doesn't matter.

[pattern-pattern] - Match a single character which is part of the range.

[^pattern] - Match only one character which is **NOT** a part of the pattern in the set.

[metacharacters] - Do not need to escape metacharacters. Escape only ] - ^ \

Shorthand for character set: \d - Digit [0-9]

\w - Word character [a-zA-Z0-9\_]

!!!! Important : \s - Whitespace [ \t\r\n]

[^\d\s] - **NOT** digit **or** space \D - Not digit [^0-9]

[\D\S] - **NOT** digit **or** **NOT** space \W - Not word character [^a-zA-Z0-9\_]

\S - Not whitespace [^ \t\r\n]

**Quantified Repetition**

{min, max} - min, max are positive numbers

- min must always be included and can be zero;

- max is optional;

\d{4, 8} - Match numbers with four to eight digits;

\d{4} - Match numbers with exactly four digits (min is max);

\d{4,} - Match numbers with four or more digits (max is infinie);

**Bacreference**

/(A?)B/ matches "B" and capture "" - element **is** optional, the group/capture **is not** optional.

/(A)?B/ matcher "B" and does not capture anything - element **is not** optional, group capture **is** optional. Except JavaScript -behave like (A?)B.

?: - Usage - optimize for speed and saving space for capturing the important groups

/(?:pattern)/

? = "Give this group a different meaning"

: = "The meaning is non-capturing"

**Lookaround Assertions**

**Lookahead Assertion**

?= - Positive lookahead assertion. If it is true, then try to match the rest of the expression, else if it is false, the hole expression is false.

/(?=pattern)/

example: /(?=seashore)sea/ matches "sea" in "seashore" but not "seaside".

/sea(?=shore)/ matches "sea" in "seashore" but not "seaside".

?! - Negative lookahead assertion. Exclude patterns from the search.

/(?!pattern)/

example: /(?!seashore)sea/ matches "sea" in "seaside" but not "seashore".

/sea(?!shore)/ matches "sea" in "seaside" but not "seashore".

last occurance - (xx)(?!.\*(xx))

**Lookbehind Assertion**

?<= - Positive lookbehind assertion. If it is true, then try to match the rest of the expression, else if it is false, the hole expression is false.

/(?<=pattern)/

example: /(?<=base)ball/ matches "ball" in "baseball" but not "football".

?<! - Negative lookbehind assertion. Exclude patterns from the search.

/(?<!pattern)/

example: /(?<!base)ball/ matches "ball" in "football" but not "baseball".

if we have only assertion and the expression is true, then after the evaluation of the expression the pointer will be at the beginning of the first match. Then we can easily insert or replace characters.