Regular Expressions with JavaScript

By Amit Kishore

Agenda - Understanding Regular Expression

Part–1 - Basics

- History
- What ?
- Why ?
- How to create ?
- Where to test ?
- How it works ?

Part-2 - Working with JavaScript

- How to create ?
- How to use ?

Part-3 - Into the RegExp World

- Meta Characters
- Character set
- Repetitions
- Grouping
- Anchored Expressions
- Lookahead Assertions
- Using Unicode

Part –I

Basics

History

- Idea was developed by Mathematician Stephen Keene in 1950.

- First common use was in "grep" command a Unix based text-processing utilities.

(grep stands for Global Regular Expression Print or Global Search for regular Expression and print matching)

- In 1980, Perl was the first programming language to use regular expression.

- And in 1997, Regular Expression standardized so that we can use same principles everywhere.

What is Regular Expression ?

- A regular expression is a sequence of characters that defines a search pattern in text.

Example:

Text: "East or West, JavaScript is the best."

Pattern: "est"

Why use a Regular Expression ?

- Kind of optional but mandatory skill to have.

- Matching Pattern in a password. (e.g one lower, one upper, one number etc.)

- Checking if Email or Phone Number is valid or not.

- What if we have to find out how many times a word appeared in a string.

How to create a Regular Expression ?

/pattern/

Text: "East or West, JavaScript is best."

Pattern: /est/

Where to test ?

https://regex101.com/

How Pattern Matching Works?

Text: "here we help, hello everyone!"

RegExp: /hello/

Usually, the string-matching algorithm behind Regular Expression takes O(m*n) time.

Where, m is the length of regular expression and n is the length of string.

https://en.wikipedia.org/wiki/Regular_expression

End of Part-1

Part – 2

Working with JavaScript

Working with JavaScript

- Just like everything else Regular Expression is also an object in JavaScript.

- And, just like other user defined objects we can create RegExp object in two different ways.

!!Example of array and objects!!

How to create Regular Expression?

1. Literal Syntax - /pattern/

   e.g: let regex = /hello/

2. Constructor Function – RegExp(pattern)

   e.g: let regex = new RegExp(/hello/);

How to use RegExp ?

- Have 2 methods on Regex Object.

  1. test(str) - returns true or false
  2. exec(str) - returns array of matching result.

- Have 4 methods on String.

  1. match(pattern) - returns array of matching result same as exec() but has a difference.
  2. search(pattern) - returns index if match is found.
  3. replace(pattern, "text2") - replaces the matching text with the other text.
  4. split(pattern) - takes a string and convert it into an array based on delimiter.

Time to do some practice!

let array = ["201-333-3922","201-338-3322","524-203-3201","221-2012-2015","201-223-4201"];

Filter out the array or create new array which contains numbers starting with "201"

Output:

let array = ["201-333-3922","201-338-3322","201-223-4201"];

Flags

- These are used to modify the behavior of searching.

- It is optional and is denoted using a single lowercase alphabetic character.

- Multiple flags can be combined together to be used at once.

With Literal Regex: /pattern/flags;

With Regex Object: new RegExp(pattern , "flags");

Common Flags:

- i – case insensitive
- g – global match
- m – multiline mode
- s – allows . (dot) to match multiline character.
- u – enable Unicode mode.
- y - "Sticky Mode" searching at exact position in the text.

Meta Characters

- Characters that have special meaning during pattern processing.

- While constructing the Regular Expression these will be used almost all the time.

^ $. * + ? = ! : | \ / ( ) [ ] { }

The wildcard (.) metacharacter: It means any single character.

e.g. "That thing is so hot."

- What if we have to match (.) dot Itself ?
- how to decide which character to escape ?
- dot (.) has an exception with new line character.

Control Statements

\t - tab

\n - new line

\r - carriage return

Character Set

- Group of characters used for matching a single characters.
- use [...] to define character set.

e.g. Grey or Gray Advisor or Adviser

** Inside character set meta character do not act as meta character.

- to specify range in character set we use -

[a-z] [A-Z] [0-9]

* what if we have to match - in character set.

e.g. I have tried 4 - 5 times and will try again.
How to match 12 - 45 ?

* Excluding a character set: ^ carrot symbol in the start of character set.

e.g. "I have tried 4 - 5 times and tired of being trying."

Metacharacters we may need to escape: - ^ \ ]

Shorthand for character sets

e.g. "a string with numbers (12345)"

\d - [0-9]
\w - [a-zA-Z0-9_] word character
\s - [\t\r\n] space

Shorthand for negate character sets

\D - [^0-9]
\W - [^a-zA-Z0-9_] non word character
\W - [^ \t\r\n] space

Practice!

let array = ["201-333-3922","201-338-3322","201-203-32091","221-2012-20515","201-223-4201"];

Filter out the array or create new array which contains numbers starting with "201" and

Should be in the format as nnn-nnn-nnnn

** do by range and then by shorthand and then by repetition.

Output:

let array = ["201-333-3922","201-338-3322","201-223-4201"];

Repetition in pattern:

+ - match one or more occurrences of item that is on the left.
? - match zero or one occurrences.
* - match zero or more occurrences.

e.g.

"warning warning! Warning!! Warning!!"

"Take 1 apple from 10 apples."

"Hello to EveryOnE."

** Regex is greedy in nature that is it matches as much character as possible.

 This is a para2

** convert greedy nature into lazy.

Specify Repetition amount

{ min, max } - matches min to max occurrences.

{ min } - matches min or exact occurrences.

{ min, } - matches min or more occurrences.

e.g. "Find all the words having 3, 4 or 5 characters."

"#FF0000 #CCDE87 #5EF74A #CCDDEE"

Match SSN - nnn-nnn-nnnn

"202-222-3232, 201-223-3222, 323-343-5542, 143-334-2222"

** convert greedy nature into lazy.

e.g. Validate numbers which can be in any of the format.

(nnn)-nnn-nnnn, nnn.nnn.nnnn, nnn-nnn-nnnn, nnnnnnnnnn, (nnn)nnn-nnn

Anchored Expression

Use to specify position in a regex.

1. Match start and end.
   ^ - match at the start of each line.
   $ - match at the end of each line.

   e.g. "This is a bat."

2. Word boundary.
   \b - word boundary – pattern bounded by a non-word character. [^a-zA-Z0-9_]
   \B - non word boundary - pattern bounded by a word character. [a-zA-Z0-9_]

   e.g.
   "plan to plant trees on this planet."

Group

- () are used to group a pattern or party of regex.

e.g.
"a1b2c3d2 c7d2p2d6 d68h2db2 r2k1g3j4d6"

Alternate Group

- use pipe | to specify an alternate group.

e.g.
"Today is Wednesday and tomorrow is thursday & then we have friday."

** JS example of using a group with dates

Capturing Group

- by default, every group is a capturing group.
- it is called capturing group as it captures the pattern to be used later on.
- It matched the exact text instead of the pattern.
- we use back references to re-use capturing group.

e.g. same date examples.

Non-Capturing group

- regex engine do not capture the group.
- when to use it ?

Named Capturing group

- instead of using the default capturing we can give name to each capturing group.
- to name a group use (**?<name>**) at the start of the group.
- to access it use **\k<name>**

e.g. "a1a1 b2c4 d5h3 d2d2 k2k3 p2p2"

Lookahead group (?=)

- when we don't want to include matching pattern to be a part of the result.
- if we use multiple lookahead, each group starts the match from the start.

e.g. "google.com youtube.com facebook.com"

** JS exec() method.

Thank You