JavaScript RegExp Syntax Cheat Sheet v1.3.4, PDF Download

By niat786 • January 15, 2021

Table of Contents

JavaScript RegExp Defination

The JavaScript RegExp object explains the patterns of characters. These patterns are used in search, validation, and search and replace functionality in JS programming.

JavaScript RegExp Syntax

```
// Syntax
/pattern/modifiers;

//Example
let pattern = /infopediya/i


//
```

In above example the `/infopediya/i` is example of JS RegExp.

The `/infopediya/` is a pattern and `/i` is a modifier. we will explain the modifiers in JavaScript RegExp Syntax Cheat Sheet.

How to use JavaScript Regular Expressions?

How to create a pattern using a constructor?

It is easy to create a pattern using the `new RegExp()`. It creates a new RegExp object.

```
//Example of the constructor

let pattern = new RegExp("Hello World", "g");
console.log(pattern);

//
```

Here the **pattern** variable will give you the " /Hello World/g " as output. The new RegExp accepts two parameters **the string and modifier**.

In the above example ' **Hello World** ' is the string pattern and **'g'** is a modifier given as the parameters to `new RegExp()`.

What are modifiers in regular expressions?

The modifiers add a particular behavior/functionality to the whole pattern.

For example, if we want to search a case-insensitive piece of text in a paragraph then the **Case Insensitive** flag comes in handy. The symbol for **Case Insensitive** is **" /i "**.

The **/hello world/i** will match " **Hello World** " as well.

Name	Symbol	Definition
global	/g	searches globally and do not stop on first matching.
case insensitive	/i	matches upper and lower case simultaneously
multiline	/m	If matching starts with or ends with your condition on each line then it returns the matching string. E.g if the new line starts with **'The'** **(^The)** or ends with **'The'(The$)**

Examples

JavaScript RegExp /g Modifier

```
//Example

 let sample_text =  'Lorem ipsum dolor sit amet consectetur adipisicing elit. Dolor nisi voluptatibus sequi Lorem soluta possimus dolores libero veritatis iste, saepe esse';


 //search 'Lorem' without /g flag
 let result1 = sample_text.match(/Lorem/);
 console.log(result1);
 //Output:["Lorem"]


 //now search 'Lorem' with /g flag
 let result2 = sample_text.match(/Lorem/g);
 console.log(result2);
 //Output: ["Lorem", "Lorem"]
```

JavaScript RegExp /i Modifier

```
 // Example /i

 let sample_text =  'Lorem ipsum IPSUM dolor sit amet consectetur adipisicing elit. Dolor nisi voluptatibus sequi Lorem soluta possimus dolores libero veritatis iste, saepe esse';

 //search 'Lorem' without /i flag
 let result1 = sample_text.match(/ipsum/g);
 console.log(result1);
 //Output:["ipsum"]


 //now search 'Lorem' with /i flag
 let result2 = sample_text.match(/ipsum/gi);
 console.log(result2);
 //Output: ["ipsum", "IPSUM"]
```

JavaScript RegExp /m Modifier

Note: The **/m flage** is case sensitive. Use it with **/gi** to search globally with case insesitive matches.

```
 // Example /m
 //  \n shows new line
 // $ sign shows ending and ^ sign shows beginning in RegExp.
 // we will explain these signs latter here.

 let sample_text =  'This is paragraph one.\nThis is paragraph two. Paragraph three';

 //each line starts with this
 let result1 = sample_text.match(/^This/mg);
 console.log(result1);
 //Output:["This"]


 //each line ends with three
 let result2 = sample_text.match(/three$/mg);
 console.log(result2);
 //Output: []
```

What are JavaScript Regular Expression Quantifiers?

Quantifiers simply show the **quantity**. consider the following example.

the @ symbol repeats atleat one time and underscore **(_)** repeats one or more times.

The @ symbol is required but **(_)** is optional.

```
 //Example Quantifiers

 let email1 = "email@example.com";
 let email2 = "email_123_123_123@example.com";

 //
```


Now how to set validation rules for such conditions? The Quantifiers resolve this problem.

Quantifier	Details
+	should repeat atleast one time
*	should repeat zero or more times
?	used for optional things (zero or one time)
{x}	find sequence (e.g `/\d{4}/g` will find a sequence of 4 digit numbers) x should be a number. Here x is 4.
{x, y}	x to y(e.g `/\d{4,6}/g` will find a sequence of 4 to 6 digit numbers)
{x,}	at least **x** (e.g `/\d{4,}/g` will find a sequence of at least 4 digit numbers)
$	at the end of it(e.g The **/x$ /g** matches any string with **x** at the end of it)
^	at the beginning of it(e.g The **/^x /g** matches any string with **x** at the beginning of it)
?=	followed by a specific string (e.g **a sample text**. here '**sample**' is followed by '**text**').
?!	Not followed by. This is opposite to **?=**

JavaScript RegExp [abc] Expression

This expression is used to find any characters given inside the brackets.

You can specify simple characters as `[abcd]`.

you can specify any range as from zero to nine `[0-9]`. Or from uppercase A to lowercase z `[A-z]`.

```
 // Example [abc] Expression

  let str = "This is a sentence.";
   let pattern = /[T]/g;
   let result = str.match(pattern);

   console.log(result);

 // output: ["T"]

 //
```

JavaScript RegExp [^abc] Expression

This expression is used to find any characters accept given inside the brackets.

This is opposite to `[abc]` Expression.

```
 // Example [^abc] Expression

  let str = "This is a sentence.";
   let pattern = /[^T]/g;
   let result = str.match(pattern);

   console.log(result);

 // output: ["h", "i", "s", " ", "i", "s", " ", "a", " ", "s", "e", "n", "t", "e", "n", "c", "e", "."]

 //
```

JavaScript RegExp . Metacharacter

The dot metacharacter is very powerful. it is used to match:

1: Every single character

2: A single character

consider the following example

```
 // Example for dot metacharacter

 // Find simple character

   let str = "Get and set values of variables.";
   let pattern = /.et/g;
   let result = str.match(pattern);
   console.log(result);
   //Output: ["Get", "set"]

 //OR match every character


   let str = "Get and set values of variables.";
   let pattern = /./g;
   let result = str.match(pattern);
   console.log(result);
   // Output: ["G", "e", "t", " ", "a", "n", "d", " ", "s", "e", "t", " ", "v", "a", "l", "u", "e", "s", " ", "o", "f", " ", "v", "a", "r", "i", "a", "b", "l", "e", "s", "."]

 //
```

List of JavaScript RegExp Metacharacters

A dot (.)	used to match single character or every character
\w	Find word character e.g A-z, 0-9, and (_) underscore
\W	Find a non-word character
\d	Find Digits from 0-9
\D	Find non-digit character
\s	It searches whitespaces
\S	It searches non-whitespaces
\b	Find a match at the beginning and at the end of a word
\B	Do not match with the beginning or end of a word e.g, **\B**cat will search words where it will not start with the **cat** or '**cat\B**' (should not ends with the **cat**).
\n	Find new line character
\f	used to find a form feed character (returns position). **Form feed** is a page-breaking ASCII control **character**.
\r	It is used to find a carriage return character. (It means to **return** to the beginning of the line without advancing downward.)
\t	Find a tab character.
\v	Match vertical tab character
\0	Find NUL character, it returns the position where NULL found.
\xxx	match the Latin character corresponding to octal number i.e 107 G. It returns NULL on not found.
\uxxxx	match the Unicode character corresponding to hexadecimal number i.e **/ \u0057 /** for **W**. It returns NULL on not found.
\xdd	match the Latin character corresponding to hexadecimal number i.e **/ \x57 /** for **W**. It returns **NULL** on not found.

Download JavaScript RegExp Syntax Cheat Sheet, the PDF File Here.

Download PDF File

References

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/RegExp

https://www.w3schools.com/jsref/jsref_obj_regexp.asp

https://regexr.com/