Agenda

- JavaScript Regular Expressions
- JavaScript Browser Debugging

JavaScript Regex

In this tutorial, you will learn about JavaScript regular expressions (Regex) with the help of examples.

In JavaScript, a **Reg**ular **Ex**pression (RegEx) is an object that describes a sequence of characters used for defining a search pattern. For example,

```
/^a...s$/
```

The above code defines a RegEx pattern. The pattern is: **any five letter string starting with** `a` **and ending with** `s` .

A pattern defined using RegEx can be used to match against a string.

String	Match
abs	No match
alias	Match
abyss	Match
Alias	No Match
An abacus	No Match

Create a RegEx

There are two ways you can create a regular expression in JavaScript.

1. **Using a regular expression literal :** The regular expression consists of a pattern enclosed between slashes `/` . For example, Here, `/abc/` is a regular expression.

```
cost regularExp = /abc/;
```

2. **Using the RegExp() constructor function :** You can also create a regular expression by calling the RegExp() constructor function. For example,

```
const reguarExp = new RegExp('abc');
```

For example,

```
const regex = new RegExp(/^a...s$/);
console.log(regex.test('alias')); // true
```

In the above example, the string `alias` matches with the RegEx pattern `/^a...s$/` . Here, the `test()` method is used to check if the strir matches the pattern.

There are several other methods available to use with JavaScript RegEx. Before we explore them, let's learn about regular expressions themselves.

Specify Pattern Using RegEx

To specify regular expressions; Brackets, Quantifier, Metacharacters are used. In the above example (`/^a...s$/`), `^` and `$` are metacharacters.

`\` **- Backslash**

Backslash `\` is used to escape various characters including all metacharacters. For example,

`\$a` match if a string contains `$` followed by `a` . Here, `$` is not interpreted by a RegEx engine in a special way.

If you are unsure if a character has special meaning or not, you can put `\` in front of it. This makes sure the character is not treated in a special way.

Metacharacters are characters that are interpreted in a special way by a RegEx engine. Here's a list of metacharacters:

Metacharacter	Description
\w	Find a word character
\W	Find a non-word character
\d	Find a digit
\D	Find a non-digit character
\s	Find a whitespace character
\S	Find a non-whitespace character
\b	Find a match at the beginning/end of a word, beginning like this: \bHI, end like this: HI\b
\B	Find a match, but not at the beginning/end of a word
\0	Find a NULL character
\n	Find a new line character
\f	Find a form feed character
\r	Find a carriage return character
\t	Find a tab character
\v	Find a vertical tab character
\xxx	Find the character specified by an octal number xxx
\xdd	Find the character specified by a hexadecimal number dd
\udddd	Find the Unicode character specified by a hexadecimal number dddd
.	Find a single character, except newline or line terminator

Brackets are used to find a range of characters

Expression	Description
[abc]	Find any character between the brackets
[^abc]	Find any character NOT between the brackets
[0-9]	Find any character between the brackets (any digit)
[^0-9]	Find any character NOT between the brackets (any non-digit)
(x	y)

Quantifier	Description
n+	Matches any string that contains at least one n.
n*	Matches any string that contains zero or more occurrences of n
n?	Matches any string that contains zero or one occurrences of n
n{X}	Matches any string that contains a sequence of X n's

Quantifier	Description
n{X,Y}	Matches any string that contains a sequence of X to Y n's
n{X,}	Matches any string that contains a sequence of at least X n's
n$	Matches any string with n at the end of it
^n	Matches any string with n at the beginning of it
?=n	Matches any string that is followed by a specific string n
?!n	Matches any string that is not followed by a specific string n

o build and test regular expressions, you can use RegEx tester tools such as regex101. This tool not only helps you in creating regular expressions, but also helps you learn it.

Now you understand the basics of RegEx, let's discuss how to use RegEx in your JavaScript code.

JavaScript Regular Expression Methods

As mentioned above, you can either use `RegExp()` or regular expression literal to create a RegEx in JavaScript.

```
const regex1 = /^ab/;
const regex2 = new Regexp('/^ab/');
```

In JavaScript, you can use regular expressions with `RegExp()` methods: `test()` and `exec()` .

There are also some string methods that allow you to pass RegEx as its parameter. They are: `match()` , `replace()` , `search()` , and `split()` .

exec()	Executes a search for a match in a string and returns an array of information. It returns null on a mismatch.
test()	Tests for a match in a string and returns true or false.
match()	Returns an array containing all the matches. It returns null on a mismatch.
matchAll()	Returns an iterator containing all of the matches.
search()	Tests for a match in a string and returns the index of the match. It returns -1 if the search fails.
replace()	Searches for a match in a string and replaces the matched substring with a replacement substring.
split()	Break a string into an array of substrings.

Example 1: Regular Expressions

```
const string = 'Find me';
const pattern = /me/;

// search if the pattern is in string variable
const result1 = string.search(pattern);
console.log(result1); // 5

// replace the character with another character
const string1 = 'Find me';
string1.replace(pattern, 'found you'); // Find found you

// splitting strings into array elements
const regex1 = /[\s,]+/;
const result2 = 'Hello world! '.split(regex1);
console.log(result2); // ['Hello', 'world!', '']

// searching the phone number pattern
```

```
const regex2 = /(\d{3})\D(\d{3})-(\d{4})/g;
const result3 = regex2.exec('My phone number is: 555 123-4567.');
console.log(result3); // ["555 123-4567", "555", "123", "4567"]
```

Regular Expression Flags

Flags are used with regular expressions that allow various options such as global search, case-insensitive search, etc. They can be used separately or together.

Flags	Description
g	Performs a global match (find all matches)
m	Performs multiline match
i	Performs case-insensitive matching

Example 2: Regular Expression Modifier

```
const string = 'Hello hello hello';

// performing a replacement
const result1 = string.replace(/hello/, 'world');
console.log(result1); // Hello world hello

// performing global replacement
const result2 = string.replace(/hello/g, 'world');
console.log(result2); // Hello world world

// performing case-insensitive replacement
const result3 = string.replace(/hello/i, 'world');
console.log(result3); // world hello hello

// performing global case-insensitive replacement
const result4 = string.replace(/hello/gi, 'world');
console.log(result4); // world world world
```

Example 3: Validating the Phone Number

```
// program to validate the phone number

function validatePhone(num) {

    // regex pattern for phone number
    const re = /^\(?([0-9]{2})\)?[-. ]?([0-9]{5})[-. ]?([0-9]{5})$/g;

    // check if the phone number is valid
    let result = num.match(re);
    if (result) {
        console.log('The number is valid.');
    }
    else {
        let num = prompt('Enter number in XX-XXXXX-XXXXX format:');
        validatePhone(num);
    }
}

// take input
```

```
let number = prompt('Enter a number XX-XXXXX-XXXXX');

validatePhone(number);
```

Output

```
Enter a number XX-XXXX-XXXXX: 9513164998
Enter number in XX-XXXXX-XXXXX format: 91-95131-64998
The number is valid
```

Example 4: Validating the Email Address

```
// program to validate the email address

function validateEmail(email) {

    // regex pattern for email
    const re = /\S+@\S+\.\S+/g;

    // check if the email is valid
    let result = re.test(email);
    if (result) {
        console.log('The email is valid.');
    }
    else {
        let newEmail = prompt('Enter a valid email:');
        validateEmail(newEmail);
    }
}

// take input
let email = prompt('Enter an email: ');

validateEmail(email);
```

Output

```
Enter an email: helloworld
Enter a valid email: students@almabetter.com
The email is valid.
```

JavaScript Browser Debugging

You can and will encounter errors while writing programs. Errors are not necessarily bad. In fact, most of the time, they help us identify issues with o code. It is essential that you know how to debug your code and fix errors.

Debugging is the process of examining the program, finding the error and fixing it.

There are different ways you can debug your JavaScript program.

1. Using console.log()

You can use the `console.log()` method to debug the code. You can pass the value you want to check into the `console.log()` method and verify the data is correct.

The syntax is:

```
console.log(object/message);
```

You could pass the object in `console.log()` or simply a message string.

In the previous tutorial, we used `console.log()` method to print the output. However, you can also use this method for debugging. For example,

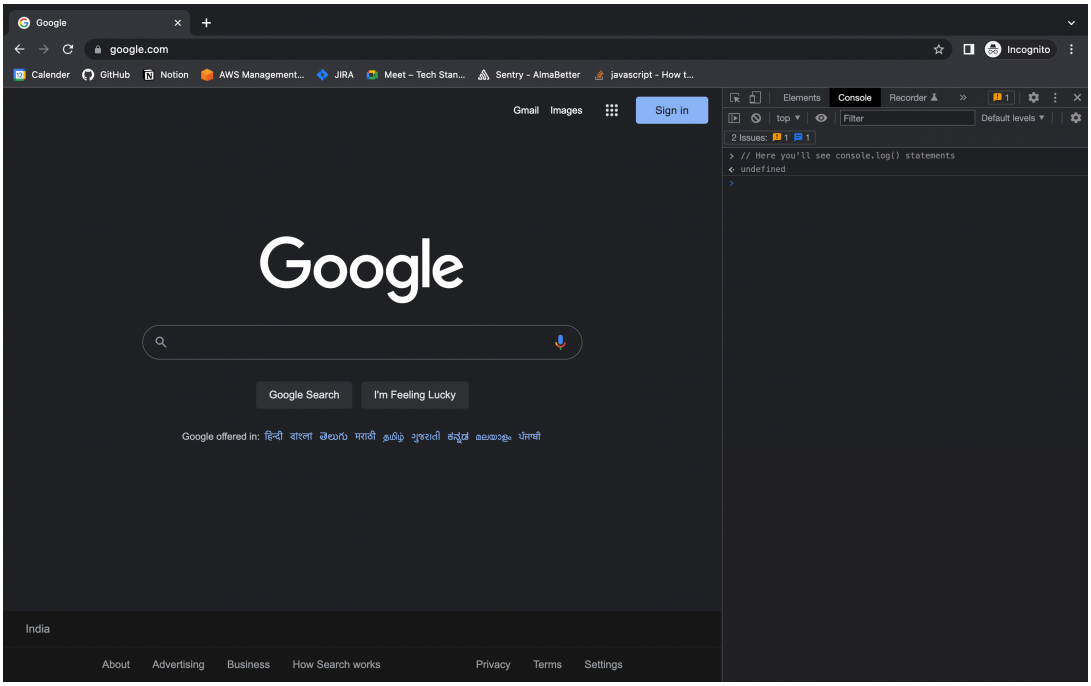
```
let a = 5;
let b = 'asdf';
let c = a + b;

// if you want to see the value of c
console.log(c);

// then do other operations
if(c) {
    // do something
}
```

Using `console.log()` method in the browser opens the value in the debugger window.

You can open developer tools and select *Console* to load this window.



Working of console.log() method in browser

The `console.log()` is not specific to browsers. It's also available in other JavaScript engines.

2. Using debugger

The `debugger` keyword stops the execution of the code and calls the debugging function.

The `debugger` is available in almost all JavaScript engines.

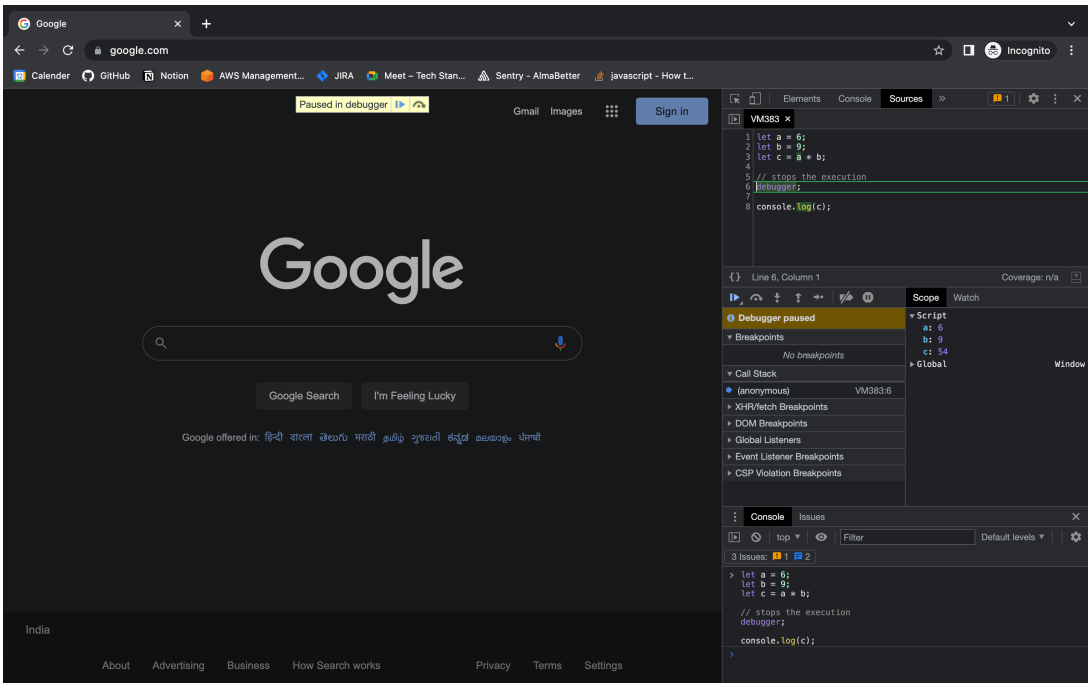
Let's see an example,

```
let a = 6;
let b = 9;
let c = a * b;

// stops the execution
debugger;
```

```
console.log(c);
```

Let's see how you can use debugger in a Chrome browser.

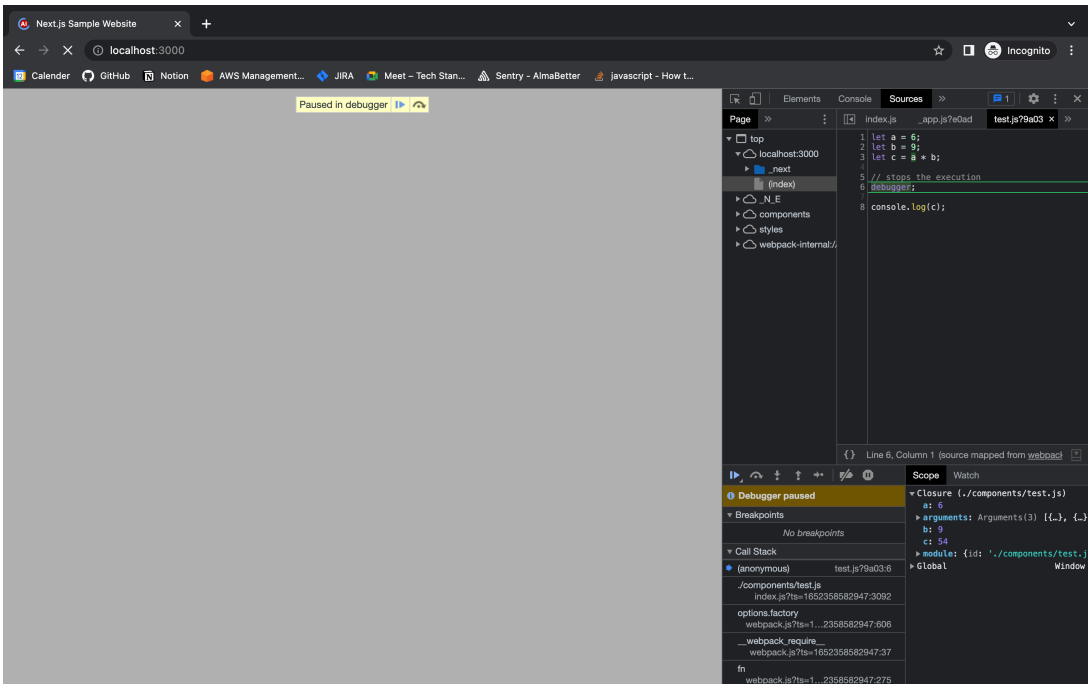


Working of debugger in the browser

The above program pauses the execution of the program in the line containing the `debugger` .

You can then resume the flow control after examining the program.

The rest of the code will execute when you resume the script by pressing play in the console.



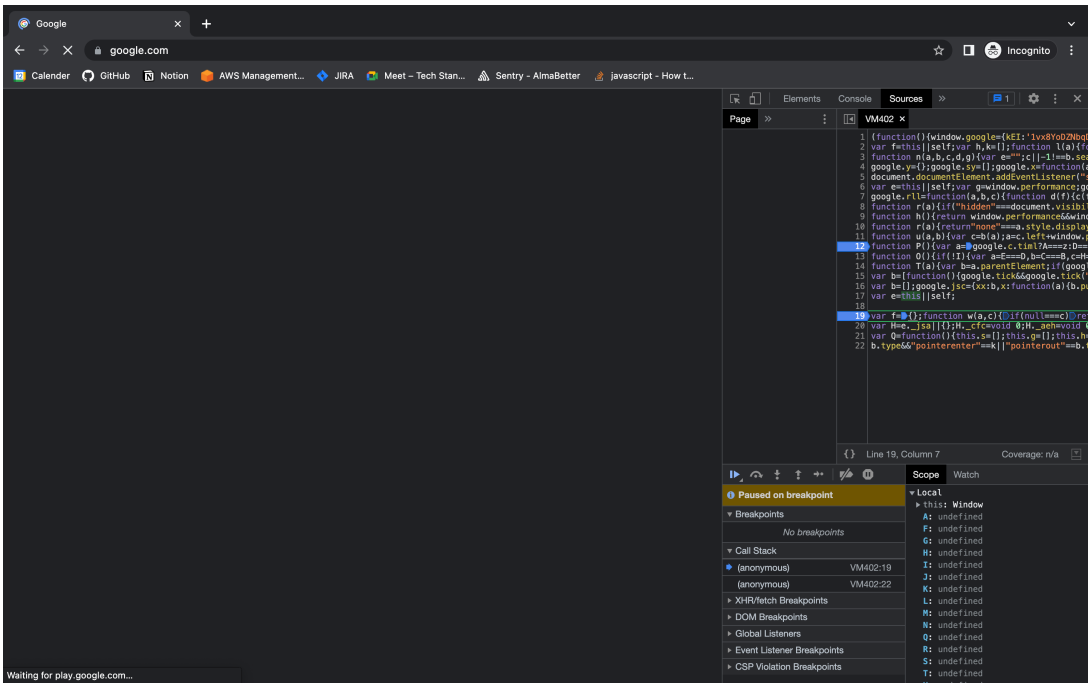
Working of debugger in the browser

3. Setting Breakpoints

You can set breakpoints for JavaScript code in the debugger window.

JavaScript will stop executing at each breakpoint and lets you examine the values. Then, you can resume the execution of code.

Let's see an example by setting a breakpoint in the Chrome browser.



Working of breakpoints in the browser

You can set breakpoints through the Developers tool anywhere in the code.

Setting breakpoints is similar to putting a debugger in the code. Here, you just set breakpoints by clicking on the line number of the source code instea of manually calling the debugger function.

Interview Questions

> What are the tools or techniques used for debugging JavaScript code

ou can use below tools or techniques for debugging JavaScript

1. Chrome Devtools
2. debugger statement
3. Good old console.log statement

> What is a debugger statement

The debugger statement invokes any available debugging functionality, such as setting a breakpoint. If no debugging functionality is available, th statement has no effect. For example, in the below function a debugger statement has been inserted. So execution is paused at the debugger stateme just like a breakpoint in the script source.

```
function getProfile() {
  // code goes here
  debugger;
  // code goes here
}
```

> What is the purpose of breakpoints in debugging

You can set breakpoints in the JavaScript code once the debugger statement is executed and the debugger window pops up. At each breakpoir JavaScript will stop executing, and let you examine the JavaScript values. After examining values, you can resume the execution of code using the pla button.

> How do you search a string for a pattern

You can use the test() method of regular expression in order to search a string for a pattern, and return true or false depending on the result.

```
var pattern = /you/;
console.log(pattern.test("How are you?")); //true
```

What is a RegExp object

RegExp object is a regular expression object with predefined properties and methods. Let's see the simple usage of RegExp object,

```
var regexp = new RegExp("\\w+");
console.log(regexp);
// expected output: /\w+/
```

What are regular expression patterns

Regular Expressions provide a group of patterns in order to match characters. Basically they are categorized into 3 types,

1. **Brackets:** These are used to find a range of characters. For example, below are some use cases,

1. [abc]: Used to find any of the characters between the brackets(a,b,c)
2. [0-9]: Used to find any of the digits between the brackets
3. (a|b): Used to find any of the alternatives separated with |

2. **Metacharacters:** These are characters with a special meaning For example, below are some use cases,

1. \d: Used to find a digit
2. \s: Used to find a whitespace character
3. \b: Used to find a match at the beginning or ending of a word

3. **Quantifiers:** These are useful to define quantities For example, below are some use cases,

1. n+: Used to find matches for any string that contains at least one n
2. n*: Used to find matches for any string that contains zero or more occurrences of n
3. n?: Used to find matches for any string that contains zero or one occurrences of n

---

Thank You