JAVASCRIPT

AWESOME TIPS

CAIO RIBEIRO PEREIRA

JavaScript Awesome Tips

A collection of tips and tricks for JavaScript

Caio Ribeiro Pereira

This book is for sale at
http://leanpub.com/javascript-awesome-tips

This version was published on 2019-03-29

I dedicate this book to my family who always supports me and motivates me since the beginning of my life.

A special thanks to Aline Brandariz Santos, for being the smile of my life, for making me happy, for supporting me and encouraging me.

A special thanks to Mrs. Charlotte Bento de Carvalho, my cousin Cláudio Souza, my friends Leandro Alvares da Costa and Bruno Alvares da Costa, Paulo Silveira and Adriano Almeida. They had an important role in my life to influencing me be who I am today, and consequently, write this book.

Thanks to all readers from https://udgwebdev.com, after all, the essence of this book is based on many posts from this blog.

Finally, thank you, dear reader for purchasing this book, I hope you enjoy it!

Contents

Preface

JavaScript is dominating the world

Today, JavaScript is in everywhere, it's a strongly language used in browsers, servers (via Node.js or other JS engines for backend), mobile (via Cordova, PhoneGap, React Native and more), IoT, SmartTVs and maybe more. If you are a web developer, you have had to play a little bit in JavaScript at some point in your career and more and more JavaScript is conquering more space in the world. This language since 2015 has been receiving important upgrades, such as the emergence of ES6, ES7, ES8, ES9 and ES10 (aka ECMAScript), which are implementations of new features to support this language, and this has brought new powers and especially new ways of solving problems in a simpler way.

About this book's audience

This book is intended for all developers who have already worked or are working with JavaScript and have at least the basic or intermediate skills in this language, and seek to learn more and more about about some practical tips, hacks and tricks.

How to read this book

This book will introduce a number of problems and practical solutions to code in JavaScript, there is no need to read each chapters of this book in sequential order, after all this is a cookbook

with a collection of practical's recipes to apply in your JavaScript's projects.

Chapter 1: String Tips

Creating slug strings using regex

If you need to create a "slug" version of a string, for example, transform the phrase: "JavaScript is awesome" to "javascript-is-awesome", you can easily create the following function:

```
function slugify(content) {
  return content.toLowerCase().replace(/\s/g, '-').trim();
}

slugify("Writing JavaScript Better");
// "writing-javascript-better"
```

Or if you want to inject this function into the String object, making it as a native function for strings, write the following code:

```
String.prototype.slugify = function() {
  return this.toLowerCase().replace(/\s/g, '-').trim();
}

"Writing JavaScript Better".slugify();
// "writing-javascript-better"
```

Interpolating data in a string

Since the ES6 (aka ES2015) implementation came out, it is possible to apply data interpolation into a string in an elegant way, you probably have already seen a lot of code similar to this:

```
const name = "John Connor";
const message = "I came from future!";
let template = "<p>";
template += "<h3>" + name + "</h3>";
template += "<span>" + message + "</span>";
template += "</p>";
```

But there is a cleaner way, thanks to ES6's feature called *Template Strings*, you can interpolate data into a string, without write complex concatenation's code, this will remove unnecessary use of the **+** operator for string interpolation, see below how to use it:

```
const name = "John Connor";
const message = "I came from future!";
const template = `
 <p>
    <h3>${name}</h3>
    <span>${message}</span>
 </p> `;
```

Repeating Strings in a Row

When it comes to repeating strings, the most common way is to write code similar to this:

```
let content = '';
const msg = 'Hello! ';
const repeat = 3;
for (let i = 0; i < repeat; i++) {
  content += msg;
}
console.log(content);
// Hello! Hello! Hello!
```

Or something with less code, like this:

```
const repeat = Array(3);
const msg = 'Hello! ';
const content = repeat.map(() => msg).join('');
console.log(content);
// Hello! Hello! Hello!
```

But now, the ES6 has a simple and better function to perform this simple task of repeating strings based on the number of iterations which is informed in the first parameter, allowing in a single line to execute this task in a semantic, readable and performatic way, to do this task, you can just need to use the `String.prototype.repeat()` function:

```
console.log('Hello!'.repeat(3));
// Hello! Hello! Hello!
```

Differences between substring() and substr()

First of all, do you know the main difference between the functions: `substring()` and `substr()`?

Basically, both functions use the same argument input, the first argument is the **starting point index**, but the difference is in the **the second argument** of both functions, take a look:

```
"JavaScript".substr(4, 6); // "Script"
"JavaScript".substring(4, 6); // "Sc"
```

- In `substr(index, quantity)` the second parameter returns the **number of characters** starting from the **index** value.
- In `substring(start, end)` this function returns a new string based on the **start and end index positions**.

Capitalizing strings

Natively you can make a string fully uppercase (`String.prototype.toUpperCase()`) or lowercase (`String.prototype.toLowerCase()`), but there is no function for capitalizing a string, which basically makes the first character to uppercase and make lowercase the rest of a string, but you can create the logic of this function in one line of code:

```
function capitalize(s) {
  return `${s.charAt(0).toUpperCase()}${s.substr(1).toLow\
erCase()}`;
}

console.log(capitalize('JAVASCRIPT')); // "Javascript"
```

If you want to make this function as native to the String's object, just write a prototype function for String, like this:

```
String.prototype.capitalize = function() {
  return `${this.charAt(0).toUpperCase()}${this.substr(1)\
.toLowerCase()}`;
}

console.log('JAVASCRIPT'.capitalize()); // "Javascript"
```

Editing query strings in the browser

Instead of wasting your time by increasing complexity code to manually edit url's string in order to include query strings parameters, you probably wrote or saw something like this:

```
const lang = 'en-us';
const section = 'books';
let url = 'https://crpwebdev.github.io';
url += '?lang ='+ lang;
url += '&section ='+ section;
console.log(url); // "https://crpwebdev.github.io?lang=en\
&section=books"
```

But there is a safe and native way to edit url's strings by using the URL API, and you can manipulate query strings in a secure way by using his `set()` and `get()` functions, this feature exists **only for JavaScript in the browser via HTML5 API**:

```
const lang = 'en-us';
const section = 'books';
const url = new URL('https://crpwebdev.github.io');
url.searchParams.set('lang', lang);
url.searchParams.set('section', section);
console.log(url); // "https://crpwebdev.github.io?lang=en\
&section=books"
```

Three ways to convert string to array

Currently there are three ways to apply *string splitting* in a single line of code, this concept is basically about transforming a **string** into an **array of chars**, and the advantages of this technique will enable the use of array's function to manipulate strings:

By using the classic `String.prototype.split()` method with **empty string** in the first parameter:

```
const title = 'Book';
// Empty string is required in argument
console.log(title.split('')); // ['B', 'o', 'o', 'k']
// Split without argument or using undefined it returns a\
n array with a single string
console.log(title.split()); // ['Book']
```

By using `Array.from()` with the string to be manipulated in the first parameter:

```
const title = 'Book';
console.log(Array.from(title)); // ['B', 'o', 'o', 'k']
```

By combining array with spread operator:

```
const title = 'Book';
console.log([...title]); // ['B', 'o', 'o', 'k']
```

Applying replace all

The `String.prototype.replace()` function is very useful when you need to replace certain characters of a string, which makes it much more interesting, is this function allows the use of **regular expression** to manipulate complex strings. And thanks to that, it is possible to make this function replace in a single execution several occurrences, instead of executing multiple times the same function, and you just need to use the `/g` in the end of the expression, take a look:

```
const title = 'JavaJavaScript';
console.log(title.replace(/Java/, '')); // "JavaScript"
console.log(title.replace(/Java/g, '')); // "Script"
```

Searching for words in an object's attributes

This tip is very useful, because you can search for any occurrence of a word through the attributes of an object, and you can do it in a single line of code, by using the combination of `Object.values().toString().includes('string to search')`:

```
const person = {
  name: 'John Connor',
  twitter: '@john'
};

Object.values(person).toString().includes('Connor'); //\
 true
```

Basically, the function `Object.values(person)` will convert the values of all object's attributes to array, and then you can use `Array.prototype.toString()` to make all items become a single string, and finally you can use the `String.prototype.include()` function to search if there is an occurrence of a word inside the whole string, if yes, it will return `true`.

Extracting content from string HTML tags

If one day you need to scrap contents from a string that contains HTML tags, you can remove these tags by using the `String.prototype.replace()` function with the following code:

```
const content = '<h1>JavaScript</h1> <h2>is the best!</h2\
>';
const text = content.replace(/<[a-zA-Z/][^>]*>/g, '');
console.log(text); // "JavaScript is the best!"
```

Chapter 2: Number Tips

Casting number to currency's number

Since the implementation of ES6 in JavaScript, you can convert numbers to "**currency's number**" natively by using the `Number.prototype.toLocale` function, see this code below:

```
(10.9).toLocaleString(); // "10.90"
(1002.5).toLocaleString('en-US'); // "1,002.50"
(5.55).toLocaleString('en-US', {
  // Changing decimal digits
  minimumFractionDigits: 2,
  maximumFractionDigits: 2
});
```

The best of this, is you can avoid doing some workarounds by using `Math.abs()` or `Number.prototype.toFixed()` to perform this number's conversion, and another cool stuff, this function is safe against **floating bugs**, which still exist in JavaScript, for example:

```
// Floating bug
0.1 + 0.2 // 0.30000000000000004
// Avoid floating bug
(0.1 + 0.2).toLocaleString(); // "0.3"
```

For more details about locales and other useful parameters for this function, you can study on Mozilla's documentation: Developers Mozilla: Number.prototype.toLocaleString()[1]

[1] https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Number/toLocaleString

Casting string to numbers using operator +

This magic is very cool and also is very simple to do it. But the code can be less easy to understand than by using `Number('100')` to perform the cast of string to number, anyway, it's always good to learn something new. In this case, you can use `+` to convert numeric strings to number, when the string is not a number it will return `NaN` *(Not a Number)*, take a look this example below:

```
const stringNumber = '100';
console.log(+stringNumber); // 100
```

Casting date to numbers using the same operator +

Just as the `+` operator can cast a `string number` to `number`, you can also use the same operator to converts a `Date` object to a number that represents a date in milliseconds, this hack can be very useful when you need to get a date's timestamp:

```
console.log(+new Date()); // 1410480611962
```

Note: If you wanna use a more readable implementation than use `+` operator, you can always use `Number(new Date())`, `Date.now()` or `Date.prototype.getTime()` to get the milliseconds from the a date.

Rounding numbers using operator ~~

The use of ~~ operator for rounding numbers, in my opinion, it's very weird, but it's also an alternative than use `Math.floor()` function:

```
console.log(Math.floor(10.99)); // 10
console.log(~~10.99); // 10
```

Checking if number is odd or even

To find out if a number is even or odd, you can use the `%` operator, this function is responsible to get the rest of a division between two numbers, and in this case, if the rest of a number divided by 2 is equals zero, it means the number is even, if it's equal one than the number is odd:

```
// Odd number
console.log(12 % 2 === 0); // true
// Even number
console.log(13 % 2 === 1); // true
```

Calculating the age of a date like a boss

After learning the `+` operator to convert `String` or `Date` to `Number`, and the use of the weird ~~ operator to perform the round of a number, now you can create a minimalistic function to calculate

the age from a date. And the most cool of this function is that is possible to write in only one line of performatical code, see this example below:

```
function calculateAge(date) {
  return ~~((Date.now() - date.getTime()) / 31557600000);
}
// The date I'm writing this code is 2019/03/03
console.log(calculateAge(new Date(1990, 5, 4))); // 28
```

Note: What this number `31557600000` means? It's basically the duration of a year in milliseconds. If you wanna see the performance about a function using this magical number, you can take a look a benchmark in this link: jsperf.com/birthday-calculation[2]

[2] https://jsperf.com/birthday-calculation