Strings, Regex & Unicode

WE'LL COVER THE FOLLOWING

String Methods

Recently introduced in JavaScript are a few string methods that are more convenient methods for working with strings. These methods include `.startsWith`, `.endsWidth`, `.includes`, and `.repeat`.

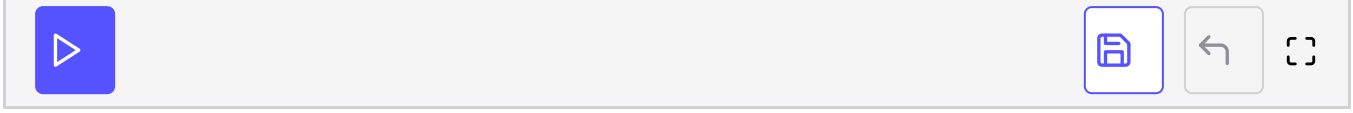
Previously if we wanted to search for the existence of some text in a string we would have to use regex. Now there are a few methods that help us make this a bit easier.

`.startsWith`

If you wanted to see if a string started with a specific bit of text, we could have created a regular expression that looked like this.

```
const truth = "JavaScript is a really fun language!";
console.log(truth.match(/^JavaScript/)); //[ 'JavaScript',
```

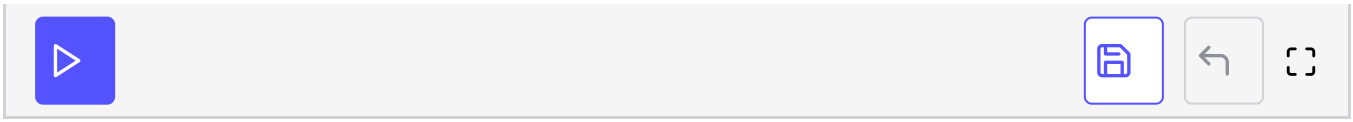
```
                                    //index: 0,
                                    //input: 'JavaScript is a really fun language!' ]
console.log(truth.match(/^fun/)); //null
```



This regular expression would search for the work `JavaScript` at the beginning of our string. With the `.startsWith` method, we cause do that same thing.

```
const truth = "JavaScript is a really fun language!";

console.log(truth.startsWith("JavaScript")); //true
console.log(truth.startsWith("fun")); //false
```

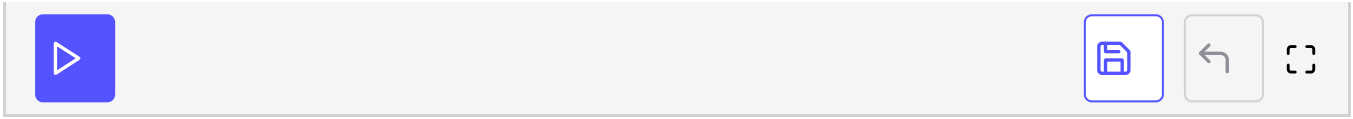


The method will return `true` or `false` if the text you are looking for appears in the at the start of the string.

`.endsWith`

Similar to `.startsWith` the `.endsWith` method will check to see if a string ends with a specific bit of text. If we wanted to do this in with a regular expression we could match it like this.

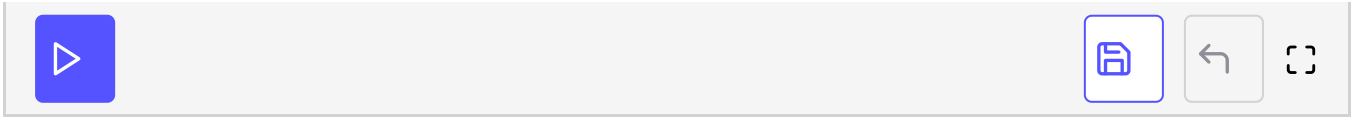
```
const truth = "JavaScript is a really fun language!";
console.log(truth.match(/language!$/)); //['language!', index: 27 ....]
console.log(truth.match(/JavaScript$/)); //null
```



With the `.endsWith` method will allow us to make this check without having to write any regex, and it will simply return `true` or `false` after the check.

```
const truth = "JavaScript is a really fun language!";

console.log(truth.endsWith('language!')); //true
console.log(truth.endsWith('JavaScript')); //false
```



`.includes`

If we can check weather a string starts or ends with some text we should also be able to check if it includes some text. The `.includes` method will allow us to do just this. This is again another convenience method for something we could perform with a regular expression.

```
const truth = "JavaScript is a really fun language!";
console.log(truth.match(/fun/g)); //["fun"]
console.log(truth.match(/a/g)); //["a", "a", "a", "a", "a", "a"]
```

The `.includes` method checks the entire string for the provided text, note the check is case sensitive!

```
const truth = "JavaScript is a really fun language!";

console.log(truth.includes('fun')); //true
console.log(truth.includes('a')); //true
console.log(truth.includes('go')); //false
```

Just like `.startsWith` and `.endsWith` the `.includes` method will return `true` or `false`.

Start position

Unlike the other methods, `.includes` takes a second optional position parameter. This is used to tell `.includes` about where it should start checking in the string for the value.

```
const truth = "JavaScript is a really fun language!";

console.log(truth.includes('J')); //true
console.log(truth.includes('J', 10)); //false
```

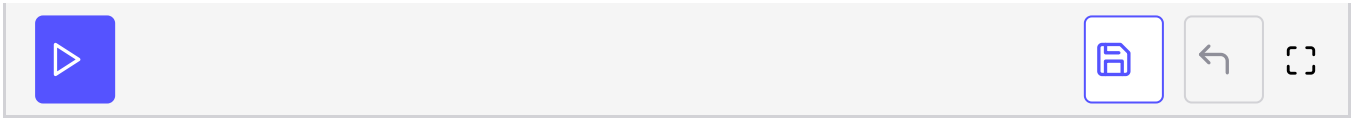
We will see this again when we get to the `.includes` method in the

ES7(ES2016) & Beyond chapter.

`.repeat`

One more method to look at is the `.repeat` method is pretty straight forward, it allows us to repeat a string a given number of times.

```
const nan = 'NaN'.repeat(6);
console.log(nan + ' BATMAN!'); //"NaNNaNNaNNaNNaNNaN BATMAN!"
```



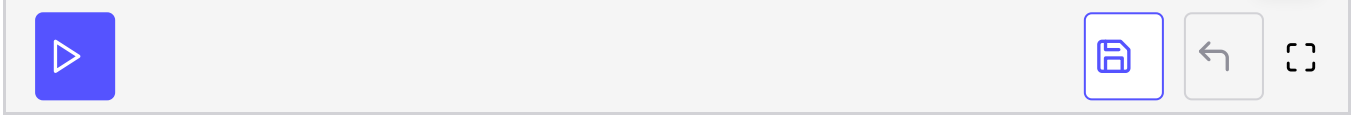
There are a few exceptions to what you can pass in as the count. It can not be a negative number and if it is a decimal number it will be rounded to an integer.

Unicode

New in ES6 for Unicode is the ability to represent Unicode as code points. Previously this was not possible because you could only represent a unicode character with up to 4 hexadecimal digits. So any Unicode character that required more that 4 you needed to create what is called a surrogate pair.

However in ES6 we can use the `\u{}` syntax to include up to 6 digits, enough to represent all the Unicode characters. It is pretty straight forward.

```
console.log('\u{1F44D}'); //👍
```



Regex

Regex in ES6 also got a few additions. There are two new flags available to use, the `y` or sticky flag, and the `u` flag. The `u` flag is used for unicode characters. For example, say we have a really cool bit of text.

```
"Man I really love 🚀, they are the best!"
```

And we wanted to see if there was the rocket ship emoji in there. Well we

And we wanted to see if there was the rocket ship emoji in there. Well we could do something like this

```
console.log("Man I really love 🚀, they are the best!".match(/\u{1F680}/u))
//[ '🚀',
//index: 18,
//input: 'Man I really love 🚀, they are the best!' ]
```

Note you could actually do this as well `.match(/🚀/)`, you will notice that we do not need to use the `u` flag here. The `u` flag is used when we pass a unicode code point `\u{}` to our expression. We can also do a range, so assume we wanted to get all the emojis used in a bit of text.

```
"Wow I really love this new phone I got 😃! Although the battery is not as good as my old on
```

Using the `u` flag we could match on a range of characters.

```
console.log("Wow I really love this new phone I got 😃! Although the battery is not as good
//["😃", "😕"]
```

This will match globally, `g`, all the faces from 😀 to 😂. And again we could just use the emjoi if we wanted `.match(/[😀-😂]/)`

Sticky match

The last thing I wanted to talk about with regex is the `y` flag, this is the sticky flag. It is used to allow us the chance to determine where to start our search. Before we dive into it, we need to look at the `lastIndex` property.

```
let regex = /really/g;
let text = "I really love pizza, is there really a better food?";
console.log(regex.exec(text)); // [ 'really', index: 2, input: 'I really love pizza, is there

console.log(regex.lastIndex); //8

console.log(regex.exec(text)); //[ 'really', index: 30, input: 'I really love pizza, is there

console.log(regex.lastIndex); //36
```

When you use the `g` flag and run an `exec` it will find the first match and set the `lastIndex` property. The next time you run the `exec` method it will use that index and start from there. With the `y` or sticky flag, we can set the `lastIndex` to let RegEx know where to start looking.

```
let regex = /really/y;
let text = "I really love pizza, is there really a better food?";
console.log(regex.exec(text)); //null
regex.lastIndex = 30;

console.log(regex.exec(text));
//[ 'really',
// index: 30,
// input: 'I really love pizza, is there really a better food?' ]
```

This can be helpful if you need to check for a bit of text starting from a specific point in your code.

Additional Resources

- https://mathiasbynens.be/notes/es6-unicode-regex
- http://www.loganfranken.com/blog/831/es6-everyday-sticky-regex-matches/