Lecture

String & Methods

Topics

Strings

- **Length**
- **Concatenation**
- **Comparing**

String Methods

- **toString()**
- **indexOf()**
- **substring()**
- **replaceAll()**
- **splice()**
- **split()**

Strings

JavaScript Strings

JavaScript strings are primitive values. JavaScript strings are also immutable. It means that if you process a string, you will always get a new string. The original string doesn't change.

To create literal strings in JavaScript, you use single quotes or double quotes:

```
let str = 'Hi';
let greeting = "Hello";
```

```
console.log(greeting);
```

Creating a String

JStrings are useful for holding data that can be represented in text form. Some of the most-used operations on strings are to check their length, to build and concatenate them using the + and += string operators, checking for the existence or location of substrings with the indexOf() method, or extracting substrings with the substring() method.

```
const string1 = "A string primitive";
const string2 = 'Also a string primitive';
```

Strings can be created as primitives, from string literals, or as objects, using the String() constructor:

```
const string4 = new String("A String object");
```

Escaping special characters

To escape special characters, you use the backslash \ character. For example:

Windows line break: '\r\n'
Unix line break: '\n'
Tab: '\t'
Backslash '\'

The following example uses the backslash character to escape the single quote character in a string:

```
let str = 'I\'m a string!';
```

String.Length

The **length** property of a String object contains the length of the string, in UTF-16 code units.

length is a read-only data property of string instances.

```
let str = "Good Morning!";
console.log(str.length);  // 13
```

Accessing Characters

To access the characters in a string, you use the array-like [] notation with the zero-based index.

The following example returns the first character of a string with the index zero:

```
let str = "Hello";
console.log(str[0]); // "H"
```

To access the last character of the string, you use the length - 1 index:

```
let str = "Hello";
console.log(str[str.length -1]); // "o"
```

Concatenating strings via + operator

To concatenate two or more strings, you use the + operator:

```
let name = 'John';
let str = 'Hello ' + name;

console.log(str); // "Hello John"
```

If you want to assemble a string piece by piece, you can use the += operator:

```
let className = 'btn';
className += ' btn-primary'
className += ' none';
```

Comparing Strings

To compare two strings, you use the operator >, >=, <, <=, and == operators.

These operators compare strings based on the numeric values of JavaScript characters. In other words, it may return the string order that is different from the one used in dictionaries.

```
let result = 'a' < 'b';
console.log(result); // true
```

```
let result = 'a' < 'B';
console.log(result); // false
```

Video: Strings

String Methods

object.toString()

To convert a non-string value to a string, in a number of ways.

Note that the toString() method doesn't work for undefined and null.

When you convert a string to a boolean, you cannot convert it back via the Boolean():

```
let status = false;
let str = status.toString(); // "false"
let back = Boolean(str); // true
```

String.UpperCase()

The **toUpperCase()** method returns the calling string value converted to uppercase (the value will be converted to a string if it isn't one). *We have the opposite method string.LowerCase()*

```
str.toUpperCase()
```

```
const sentence = 'The quick brown fox jumps over the lazy dog.';

console.log(sentence.toUpperCase());
// expected output: "THE QUICK BROWN FOX JUMPS OVER THE LAZY DOG."
```

String.indexOf()

The **indexOf()** method returns the index within the calling String object of the first occurrence of the specified value, starting the search at fromIndex.

indexOf() is case sensitive and returns -1 if the value is not found.

```
str.indexOf(searchValue [, fromIndex])
```

```
const paragraph = 'The quick brown fox jumps over the lazy dog. If the dog barked, was it really lazy?';

const searchTerm = 'dog';
const indexOfFirst = paragraph.indexOf(searchTerm);

console.log(indexOfFirst); // 40
```

String.substring()

The **substring()** method returns the part of the string between the start and end indexes, or to the end of the string

```
str.substring(indexStart[, indexEnd])
```

```
const str = 'Mozilla';

console.log(str.substring(1, 3));
// expected output: "oz"

console.log(str.substring(2));
// expected output: "zilla"
```

String.concat()

The **concat()** method concatenates the string arguments to the calling string and returns a new string.

It is strongly recommended that the assignment operators (+, +=) are used instead of the **concat()** method.

```
str.concat(str2 [, ...strN])
```

```
let hello = 'Hello, '
console.log(hello.concat('Kevin', '. Have a nice day.'))
// Hello, Kevin. Have a nice day.
```

String.replaceAll()

The **replaceAll()** method returns a new string with all matches of a pattern replaced by a replacement.

The pattern can be a string or a RegExp, and the replacement can be a string or a function to be called for each match.

```
const p = 'The quick brown fox jumps over the lazy dog. If the dog reacted, was it really lazy?';


console.log(p.replaceAll('dog', 'monkey'));
// expected output: "The quick brown fox jumps over the lazy monkey. If the monkey reacted, was it really lazy?"
```

String.slice()

The **slice()** method extracts a section of a string and returns it as a new string, without modifying the original string.

```
str.slice(beginIndex[, endIndex])
```

```
const str = 'The quick brown fox jumps over the lazy dog.';

console.log(str.slice(31));
// expected output: "the lazy dog."

console.log(str.slice(4, 19));
// expected output: "quick brown fox"

console.log(str.slice(-4));
// expected output: "dog."
```

String.split()

The **split()** method divides a String into an ordered list of substrings, puts these substrings into an array, and returns the array. The division is done by searching for a pattern; where the pattern is provided as the first parameter in the method's call.

```
const splitStr = str.split(separator, limit);
```

- separator - a string indicating where each split should occur
- limit - a number for the amount of splits to be found

String.split() cont..

Since we used the period (.) as the separator string, the strings in the output array do not contain the period in them – the output separated strings do not include the input separator itself.

```
const str = "Hello. I am a string. You can separate me.";
const splitStr = str.split("."); // Will separate str on each period character

console.log(splitStr); // [ "Hello", " I am a string", " You can separate me", "" ]
console.log(str); // "Hello. I am a string. You can separate me."
```

Video: Split vs Slice