JavaScript string

Summary: in this tutorial, you'll learn about the JavaScript `string` primitive type and how to use it to define strings.

Introduction to the JavaScript strings

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

To create literal strings, you use either single quotes (`'`) or double quotes (`"`) like this:

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

ES6 introduced template literals that allow you to define a string backtick (`) characters:

```
let name = `John`';
```

The template literals allow you to use single quotes and double quotes inside a string without the need to escape them. For example:

```
let mesage = `"I'm good". She said";
```

Also, you can place the variables and expressions inside a template literal. JavaScript will replace the variables with their value in the string. This is called string interpolation. For example:

```
let name = 'John';
let message = `Hi, I'm ${name}.`;

console.log(message);
```

Output:

```
Hi, I'm John.
```

In this example, JavaScript replaces the `name` variable with its value inside the template literal.

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!';
```

Getting the length of the string

The `length` property returns the length of a string:

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

> Note that JavaScript has the `String` type (with the letter `S` in uppercase), which is the primitive wrapper type of the primitive `string` type. Therefore, you can access all properties and methods of the String type from a primitive string.

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';

console.log(className);
```

Output:

```
btn btn-primary none
```

Converting values to string

To convert a non-string value to a string, you use one of the following:

- String(n);
- ” + n
- n.toString()

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. For example:

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

In this example:

- First, declare the `status` variable and initialize it with the value of `false` .
- Second, convert the status variable to a string using the `toString()` method.
- Third, convert the string back to a boolean value using the `Boolean()` function. The Boolean() function converts the string `"false"` to a boolean value. The result is true because `"false"` is a non-empty string.

> Note that only the string for which the `Boolean()` returns `false`, is the empty string (`''`);

Comparing strings

To compare two strings, you use comparison operators such as `>`, `>=`, `<`, `<=`, and `==` operators.

The comparison operators compare strings based on the numeric values of the characters. And it may return a string order that is different from the one used in dictionaries. For example:

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

However:

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

Summary

- JavaScript strings are primitive values and immutable.
- Literal strings are delimited by single quotes (`'`), double quotes (`"`), or backticks (`).
- The `length` property returns the length of the string.
- Use the comparison operators `>, >=, <, <=, == to compare strings.