String

What is a String Object in JavaScript?

The String object is used to represent and manipulate a sequence of characters. Strings 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. JavaScript's native String primitive comes with various common string functions.

Creating Strings

```
const str1 = "A string primitive";
const str2 = 'Another string primitive';
const str3 = `Yet another string primitive`;
const str4 = new String("A String object");
```

String literals can be specified using single or double quotes, which are treated identically, or using the backtick character

String Character access

There are two ways by which we can access an individual character in a string.
The first one the charAt() method:

```
'dog'.charAt(1) // gives value "o"
```

.charAt(index) takes an index (which starts at 0) and returns the character at that index location in the string.
The other way is to treat the string as an array-like object, where individual characters correspond to a numerical index:

```
'dog'[1] // gives value "o"
```

In string if we have to access multiple characters, we can use .substring(startIndex, endIndex), which will return the characters between the specified indices.

```
'YouTube'.substring(1,2); // returns 'o'
 YouTube'.substring(3,7); // returns 'tube'
```

If you do not pass a second parameter (endIndex), it will return all the character values from the specified start position until the end.

```
return 'YouTube'.substring(1); // returns 'outube'
```

String Comparison

Most programming languages have a function that allows you to compare strings. In JavaScript, this can be done simply by using less-than and greater-than operators.

```
var a = 'a';
var b = 'b';
console.log(a < b); // prints 'true'
```

This can be really useful for comparing strings when sorting algorithms.

However, if you are comparing two strings of different lengths, it starts comparing from the start of the string until the length of the smaller string.

```
var a = 'add';
var b = 'b';
console.log(a < b); // prints 'true'
```

In this example, a and b are compared. Since a is smaller than b, a < b evaluates to true.

```
var a = 'add';
```

```
var b = 'ab';
console.log(a < b); // prints 'false'
```

In this example, after 'a' and 'b' are compared, 'd' and 'b' are compared. Processing cannot continue because everything in 'ab' has been looked at. This is the same as comparing 'ad' with 'ab'.

```
console.log('add'<'ab' == 'ad'<'ab'); // prints 'true'
```

String Search

To find a specific string within a string, you can use .indexOf(searchValue[, fromIndex]). This takes a parameter that is the string to be searched as well as an optional parameter for the starting index for the search. It returns the position of the matching string, but if nothing is found, then -1 is returned. Note that this function is case sensitive.

```
'Red Dragon'.indexOf('Red'); // returns 0
'Red Dragon'.indexOf('RedScale'); // returns -1
'Red Dragon'.indexOf('Dragon', 0); // returns 4
'Red Dragon'.indexOf('Dragon', 4); // returns 4
'Red Dragon'.indexOf(", 9); // returns 9
```

To check for the occurrence of a search string inside a larger string, simply check whether -1 was returned from .indexOf.

```
function existsInString (stringValue, search) {
return stringValue.indexOf(search) !== -1;
}
console.log(existsInString('red','r')); // prints 'true';
console.log(existsInString('red','b')); // prints 'false';
```

You can use an additional parameter to search after a certain index in a string. An example is counting occurrences of certain letters . In the following example, the occurrences of the character 'a' will be counted:

```
var str = "He's my king from this day until his last day";
var count = 0;
var pos = str.indexOf('a');
while (pos !==-1) {
count++;
pos = str.indexOf('a' pos + 1);}
console.log (count); // prints '3'
```

Finally, startsWith returns true (boolean) if the string starts with the specified input, and endsWith checks whether the string ends with the specified input.

```
'Red Dragon'.startsWith('Red'); // returns true
'Red Dragon'.endsWith('Dragon'); // returns true
'Red Dragon'.startsWith('Dragon'); // returns false
'Red Dragon'.endsWith('Red'); // returns false
```

String Decomposition

String Split

For decomposing a string into parts, you can use **.split(separator)**, which is a great utility function. It takes one parameter (the separator) and creates an array of substrings.

```
var test1 = 'chicken,noodle,soup,broth';
test1.split(","); // ["chicken", "noodle", "soup", "broth"]
```

Passing an empty separator will create an array of all the characters.

```
var test1 = 'chicken';
test1.split(""); // ["c", "h", "i", "c", "k", "e", "n"]
```

This is useful for when there are items listed in a string. The string can be turned into an array to easily iterate through them.

String Replace

.replace(string, replaceString) replaces a specified string within a string variable with another string.

```
"Wizard of Oz".replace("Wizard","Witch"); // "Witch of Oz"
```

Regular Expressions

Regular expressions (regexes) are a set of characters that define a search pattern. Learning how to use regexes is a massive task of its own, but as a JavaScript developer, it is important you know the basics of regexes.

JavaScript also comes with the native object RegExp, which is used for regular expressions.

The constructor for the RegExp object takes two parameters: the regular expression and the optional match settings, as shown here:

i Perform case-insensitive matching

g Perform a global match (find all matches rather than stopping after first match)

m Perform multiline matching

RegExp has the following two functions:

• **search():** Tests for matches in a string. This returns the index of the match.

• **match():** Tests for matches. This returns all the matches.

The JavaScript String object also has the following two regex-related functions that accept the RegExp object as an argument:

• **exec():** Tests for matches in a string. This returns the first match.

• **test():** Tests for matches in a string. This returns true or false.

Basic Regex

Here are the basic regex rules:

^: Indicates the start of a string/line

\d: Finds any digit

[abc]: Finds any character between the brackets

[^abc]: Finds any character not between the brackets

[0-9]: Finds any digit between the brackets

[^0-9]: Finds any digit not between the brackets

(x|y): Finds any of the alternatives specified

The following returns index 11, which is the index of the character D, which is the first character of the matched regex:

```
var str = "JavaScript DataStructures";
var n = str.search(/DataStructures/);
console.log(n); // prints '11'
```

String Summary

Function	Usage
charAt (index)	Accesses a single character at index
substring(startIndex, endIndex)	Accesses part of string from start Index to endIndex
str1 › str2	Returns true if str1 is lexicographically bigger than str2
index0f(str, startIndex)	Index of the desired str starting at startIndex
str.split(delimiter)	Breaks a string into an array with the specified delimiter
str.replace(original, new)	Replaces original with new

Regex Summary

Regex	**Pattern Usage**
/\d+/	Any numeric characters
/^\d+$/	Only numeric characters
/^[0-9]*. [0-9]*[1-9]+$/	Float numeric characters
/[a-zA-Zo-9]/	Only alphanumeric characters