JavaScript Strings

JavaScript strings are for storing and manipulating text.

A JavaScript string is zero or more characters written inside quotes.

JavaScript String Length

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

Extracting String Parts

There are 3 methods for extracting a part of a string:

- `slice(start, end)`
- `substring(start, end)`
- `substr(start, length)`

JavaScript String slice()

`slice()` extracts a part of a string and returns the extracted part in a new string.

Note

JavaScript counts positions from zero.

First position is 0.

Second position is 1.

JavaScript String substring()

`substring()` is similar to `slice()`.

The difference is that start and end values less than 0 are treated as 0 in `substring()`.

If you omit the second parameter, `substring()` will slice out the rest of the string.

JavaScript String substr()

`substr()` is similar to `slice()`.

The difference is that the second parameter specifies the **length** of the extracted part.

Replacing String Content

The `replace()` method replaces a specified value with another value in a string:

Note

The `replace()` method does not change the string it is called on.

The `replace()` method returns a new string.

The `replace()` method replaces **only the first** match

If you want to replace all matches, use a regular expression with the /g flag set. See examples below.

By default, the `replace()` method replaces **only the first** match:

Note

You will learn a lot more about regular expressions in the chapter JavaScript Regular Expressions.

JavaScript String ReplaceAll()

In 2021, JavaScript introduced the string method `replaceAll()`:

Note

`replaceAll()` is an ES2021 feature.

`replaceAll()` does not work in Internet Explorer.

Converting to Upper and Lower Case

A string is converted to upper case with `toUpperCase()`:

A string is converted to lower case with `toLowerCase()`:

Note

All string methods return a new string. They don't modify the original string.

Formally said:

Strings are immutable: Strings cannot be changed, only replaced.

JavaScript String trim()

The `trim()` method removes whitespace from both sides of a string:

JavaScript String trimStart()

ECMAScript 2019 added the String method `trimStart()` to JavaScript.

The `trimStart()` method works like `trim()`, but removes whitespace only from the start of a string.

JavaScript Numbers

JavaScript has only one type of number. Numbers can be written with or without decimals.

JavaScript Numbers are Always 64-bit Floating Point

Unlike many other programming languages, JavaScript does not define different types of numbers, like integers, short, long, floating-point etc.

JavaScript numbers are always stored as double precision floating point numbers, following the international IEEE 754 standard.

This format stores numbers in 64 bits, where the number (the fraction) is stored in bits 0 to 51, the exponent in bits 52 to 62, and the sign in bit 63:

Adding Numbers and Strings

WARNING !!

JavaScript uses the + operator for both addition and concatenation.

Numbers are added. Strings are concatenated.

If you add two numbers, the result will be a number:

The JavaScript interpreter works from left to right.

First 10 + 20 is added because x and y are both numbers.

Then 30 + "30" is concatenated because z is a string.

In the last example JavaScript uses the + operator to concatenate the strings.

NaN - Not a Number

`NaN` is a JavaScript reserved word indicating that a number is not a legal number.

Trying to do arithmetic with a non-numeric string will result in `NaN` (Not a Number):

Never write a number with a leading zero (like 07).
Some JavaScript versions interpret numbers as octal if they are written with a leading zero.

By default, JavaScript displays numbers as **base 10** decimals.

But you can use the `toString()` method to output numbers from **base 2** to **base 36**.

Hexadecimal is **base 16**. Decimal is **base 10**. Octal is **base 8**. Binary is **base 2**.

JavaScript Number Methods

The toString() Method

The `toString()` method returns a number as a string.

All number methods can be used on any type of numbers (literals, variables, or expressions):

The toExponential() Method

`toExponential()` returns a string, with a number rounded and written using exponential notation.

The toFixed() Method

`toFixed()` returns a string, with the number written with a specified number of decimals:

The toPrecision() Method

`toPrecision()` returns a string, with a number written with a specified length:

The valueOf() Method

`valueOf()` returns a number as a number.

In JavaScript, a number can be a primitive value (typeof = number) or an object (typeof = object).

The `valueOf()` method is used internally in JavaScript to convert Number objects to primitive values.

There is no reason to use it in your code.

All JavaScript data types have a `valueOf()` and a `toString()` method.

The Number() Method

The `Number()` method can be used to convert JavaScript variables to numbers:

Number Methods Cannot be Used on Variables

The number methods above belong to the JavaScript **Number Object**.

These methods can only be accessed like `Number.isInteger()`.

Using X.isInteger() where X is a variable, will result in an error:

`TypeError X.isInteger is not a function`.

The Number.isInteger() Method

The `Number.isInteger()` method returns `true` if the argument is an integer.

The Number.isSafeInteger() Method

A safe integer is an integer that can be exactly represented as a double precision number.

The `Number.isSafeInteger()` method returns `true` if the argument is a safe integer.

JavaScript Arrays

Why Use Arrays?

If you have a list of items (a list of car names, for example), storing the cars in single variables could look like this:

```
let car1 = "Saab";
let car2 = "Volvo";
let car3 = "BMW";
```

However, what if you want to loop through the cars and find a specific one? And what if you had not 3 cars, but 300?

The solution is an array!

An array can hold many values under a single name, and you can access the values by referring to an index number.

Creating an Array

Using an array literal is the easiest way to create a JavaScript Array.

Syntax:

```
const array_name = [item1, item2, ...];
```

It is a common practice to declare arrays with the `const` keyword.

Arrays are Objects

Arrays are a special type of objects. The `typeof` operator in JavaScript returns "object" for arrays.

But, JavaScript arrays are best described as arrays.

Array Elements Can Be Objects

JavaScript variables can be objects. Arrays are special kinds of objects.

Because of this, you can have variables of different types in the same Array.

Associative Arrays

Many programming languages support arrays with named indexes.

Arrays with named indexes are called associative arrays (or hashes).

JavaScript does **not** support arrays with named indexes.

In JavaScript, **arrays** always use **numbered indexes**.

The Difference Between Arrays and Objects

In JavaScript, **arrays** use **numbered indexes**.

In JavaScript, **objects** use **named indexes**.

Arrays are a special kind of objects, with numbered indexes.

When to Use Arrays. When to use Objects.

- JavaScript does not support associative arrays.
- You should use **objects** when you want the element names to be **strings (text)**.
- You should use **arrays** when you want the element names to be **numbers**.

JavaScript new Array()

JavaScript has a built-in array constructor `new Array()`.

But you can safely use `[]` instead.

How to Recognize an Array

A common question is: How do I know if a variable is an array?

The problem is that the JavaScript operator `typeof` returns "`object`":

JavaScript Array Methods

JavaScript Array length

The `length` property returns the length (size) of an array:

JavaScript Array toString()

The JavaScript method `toString()` converts an array to a string of (comma separated) array values.

The `join()` method also joins all array elements into a string.

It behaves just like `toString()`, but in addition you can specify the separator:

Popping and Pushing

When you work with arrays, it is easy to remove elements and add new elements.

This is what popping and pushing is:

Popping items **out** of an array, or pushing items **into** an array.

Shifting Elements

Shifting is equivalent to popping, but working on the first element instead of the last.

JavaScript Array shift()

The `shift()` method removes the first array element and "shifts" all other elements to a lower index.

JavaScript Array unshift()

The `unshift()` method adds a new element to an array (at the beginning), and "unshifts" older elements:

Changing Elements

Array elements are accessed using their **index number**:

Array **indexes** start with 0:

[0] is the first array element
[1] is the second
[2] is the third ...

JavaScript Array delete()

Warning !

Array elements can be deleted using the JavaScript operator `delete`.

Using `delete` leaves `undefined` holes in the array.

Use pop() or shift() instead.

Merging (Concatenating) Arrays

The `concat()` method creates a new array by merging (concatenating) existing arrays:

Flattening an Array

Flattening an array is the process of reducing the dimensionality of an array.

The flat() method creates a new array with sub-array elements concatenated to a specified depth.

JavaScript Sorting Arrays

Sorting an Array

The `sort()` method sorts an array alphabetically:

Reversing an Array

The `reverse()` method reverses the elements in an array.

Numeric Sort

By default, the `sort()` function sorts values as **strings**.

This works well for strings ("Apple" comes before "Banana").

However, if numbers are sorted as strings, "25" is bigger than "100", because "2" is bigger than "1".

Because of this, the `sort()` method will produce incorrect result when sorting numbers

JavaScript Date Objects

Note

Date objects are static. The "clock" is not "running".

The computer clock is ticking, date objects are not.

JavaScript Date Output

By default, JavaScript will use the browser's time zone and display a date as a full text string:

Wed Aug 09 2023 18:20:39 GMT+0530 (India Standard Time)

You will learn much more about how to display dates, later in this tutorial.

Creating Date Objects

Date objects are created with the `new Date()` constructor.

JavaScript Math Object

The JavaScript Math object allows you to perform mathematical tasks on numbers.

The Math Object

Unlike other objects, the Math object has no constructor.

The Math object is static.

All methods and properties can be used without creating a Math object first.

Math Properties (Constants)

The syntax for any Math property is : `Math.property`.

Math.round()

`Math.round(x)` returns the nearest integer:

Math.ceil()

`Math.ceil(x)` returns the value of x rounded **up** to its nearest integer:

Math.floor()

`Math.floor(x)` returns the value of x rounded **down** to its nearest integer:

Math.trunc()

`Math.trunc(x)` returns the integer part of x:

Math.sign()

`Math.sign(x)` returns if x is negative, null or positive:

Math.pow()

`Math.pow(x, y)` returns the value of x to the power of y:

Math.sqrt()

`Math.sqrt(x)` returns the square root of x:

Math.random()

`Math.random()` returns a random number between 0 (inclusive), and 1 (exclusive):

JavaScript if, else, and else if

Conditional statements are used to perform different actions based on different conditions.

Conditional Statements

Very often when you write code, you want to perform different actions for different decisions.

You can use conditional statements in your code to do this.

In JavaScript we have the following conditional statements:

- Use `if` to specify a block of code to be executed, if a specified condition is true

- Use `else` to specify a block of code to be executed, if the same condition is false
- Use `else if` to specify a new condition to test, if the first condition is false
- Use `switch` to specify many alternative blocks of code to be executed

The `switch` statement is described in the next chapter.

The if Statement

Use the `if` statement to specify a block of JavaScript code to be executed if a condition is true.

Syntax

```
if (condition) {
  //  block of code to be executed if the condition is true
}
```

JavaScript For Loop

Loops can execute a block of code a number of times.

JavaScript Loops

Loops are handy, if you want to run the same code over and over again, each time with a different value.

Note that `if` is in lowercase letters. Uppercase letters (If or IF) will generate a JavaScript error.

Different Kinds of Loops

JavaScript supports different kinds of loops:

- `for` - loops through a block of code a number of times
- `for/in` - loops through the properties of an object
- `for/of` - loops through the values of an iterable object
- `while` - loops through a block of code while a specified condition is true
- `do/while` - also loops through a block of code while a specified condition is true

JavaScript While Loop

Loops can execute a block of code as long as a specified condition is true.

The While Loop

The `while` loop loops through a block of code as long as a specified condition is true.

If you forget to increase the variable used in the condition, the loop will never end. This will crash your browser.

The Do While Loop

The `do while` loop is a variant of the while loop. This loop will execute the code block once, before checking if the condition is true, then it will repeat the loop as long as the condition is true.

JavaScript Regular Expressions

A regular expression is a sequence of characters that forms a search pattern.

The search pattern can be used for text search and text replace operations.

What Is a Regular Expression?

A regular expression is a sequence of characters that forms a **search pattern**.

When you search for data in a text, you can use this search pattern to describe what you are searching for.

A regular expression can be a single character, or a more complicated pattern.

Regular expressions can be used to perform all types of **text search** and **text replace** operations.

Syntax

```
/pattern/modifiers;
```

Using String Methods

In JavaScript, regular expressions are often used with the two **string methods**: `search()` and `replace()`.

The `search()` method uses an expression to search for a match, and returns the position of the match.

The `replace()` method returns a modified string where the pattern is replaced.

JavaScript Errors

Throw, and Try...Catch...Finally

The `try` statement defines a code block to run (to try).

The `catch` statement defines a code block to handle any error.

The `finally` statement defines a code block to run regardless of the result.

The `throw` statement defines a custom error.

Errors Will Happen!

When executing JavaScript code, different errors can occur.

Errors can be coding errors made by the programmer, errors due to wrong input, and other unforeseeable things.

JavaScript try and catch

The `try` statement allows you to define a block of code to be tested for errors while it is being executed.

The `catch` statement allows you to define a block of code to be executed, if an error occurs in the try block.

The JavaScript statements `try` and `catch` come in pairs

JavaScript Throws Errors

When an error occurs, JavaScript will normally stop and generate an error message.

The technical term for this is: JavaScript will **throw an exception (throw an error)**.

JavaScript will actually create an **Error object** with two properties: **name** and **message**.

The throw Statement

The `throw` statement allows you to create a custom error.

Technically you can **throw an exception (throw an error)**.

The exception can be a JavaScript `String`, a `Number`, a `Boolean` or an `Object`:

The finally Statement

The `finally` statement lets you execute code, after try and catch, regardless of the result:

JavaScript Window - The Browser Object Model

The Browser Object Model (BOM) allows JavaScript to "talk to" the browser.

The Browser Object Model (BOM)

There are no official standards for the **B**rowser **O**bject **M**odel (BOM).

Since modern browsers have implemented (almost) the same methods and properties for JavaScript interactivity, it is often referred to, as methods and properties of the BOM.

The Window Object

The `window` object is supported by all browsers. It represents the browser's window.

All global JavaScript objects, functions, and variables automatically become members of the window object.

Global variables are properties of the window object.

Global functions are methods of the window object.

Even the document object (of the HTML DOM) is a property of the window object:

```
window.document.getElementById("header");
```

is the same as:

```
document.getElementById("header");
```

Window Size

Two properties can be used to determine the size of the browser window.

Both properties return the sizes in pixels:

- `window.innerHeight` - the inner height of the browser window (in pixels)
- `window.innerWidth` - the inner width of the browser window (in pixels)

The browser window (the browser viewport) is NOT including toolbars and scrollbars.

Other Window Methods

Some other methods:

- `window.open()` - open a new window
- `window.close()` - close the current window
- `window.moveTo()` - move the current window
- `window.resizeTo()` - resize the current window

JavaScript Window Screen

Window Screen

The `window.screen` object can be written without the window prefix.

Properties:

- `screen.width`
- `screen.height`
- `screen.availWidth`
- `screen.availHeight`
- `screen.colorDepth`
- `screen.pixelDepth`

Window Screen Width

The `screen.width` property returns the width of the visitor's screen in pixels.

JavaScript Window Location

The `window.location` object can be used to get the current page address (URL) and to redirect the browser to a new page.

Window Location

The `window.location` object can be written without the window prefix.

Some examples:

- `window.location.href` returns the href (URL) of the current page
- `window.location.hostname` returns the domain name of the web host
- `window.location.pathname` returns the path and filename of the current page
- `window.location.protocol` returns the web protocol used (http: or https:)
- `window.location.assign()` loads a new document

Window Location Href

The `window.location.href` property returns the URL of the current page.

JavaScript Window History

The `window.history` object contains the browsers history.

Window History

The `window.history` object can be written without the window prefix.

To protect the privacy of the users, there are limitations to how JavaScript can access this object.

Some methods:

- `history.back()` - same as clicking back in the browser
- `history.forward()` - same as clicking forward in the browser

Window History Back

The `history.back()` method loads the previous URL in the history list.

Window History Forward

The `history.forward()` method loads the next URL in the history list.

This is the same as clicking the Forward button in the browser.

JavaScript Window Navigator

The `window.navigator` object contains information about the visitor's browser.

Window Navigator

The `window.navigator` object can be written without the window prefix.

Some examples:

- `navigator.cookieEnabled`
- `navigator.appCodeName`
- `navigator.platform`

Browser Cookies

The `cookieEnabled` property returns true if cookies are enabled, otherwise false:

JavaScript Popup Boxes

JavaScript has three kind of popup boxes: Alert box, Confirm box, and Prompt box.

Alert Box

An alert box is often used if you want to make sure information comes through to the user.

When an alert box pops up, the user will have to click "OK" to proceed.

Confirm Box

A confirm box is often used if you want the user to verify or accept something.

When a confirm box pops up, the user will have to click either "OK" or "Cancel" to proceed.

If the user clicks "OK", the box returns **true**. If the user clicks "Cancel", the box returns **false**.

Syntax

```
window.confirm("sometext");
```

Prompt Box

A prompt box is often used if you want the user to input a value before entering a page.

When a prompt box pops up, the user will have to click either "OK" or "Cancel" to proceed after entering an input value.

If the user clicks "OK" the box returns the input value. If the user clicks "Cancel" the box returns null.

Syntax

```
window.prompt("sometext","defaultText");
```

JavaScript Timing Events

Timing Events

The `window` object allows execution of code at specified time intervals.

These time intervals are called timing events.

The two key methods to use with JavaScript are:

- `setTimeout(function, milliseconds)`
  Executes a function, after waiting a specified number of milliseconds.
-

- `setInterval(function, milliseconds)`
  Same as setTimeout(), but repeats the execution of the function continuously.

The `setTimeout()` and `setInterval()` are both methods of the HTML DOM Window object.

JavaScript Cookies

Cookies let you store user information in web pages.

What are Cookies?

Cookies are data, stored in small text files, on your computer.

When a web server has sent a web page to a browser, the connection is shut down, and the server forgets everything about the user.

Cookies were invented to solve the problem "how to remember information about the user":

- When a user visits a web page, his/her name can be stored in a cookie.
- Next time the user visits the page, the cookie "remembers" his/her name.

When a browser requests a web page from a server, cookies belonging to the page are added to the request. This way the server gets the necessary data to "remember" information about users.

None of the examples below will work if your browser has local cookies support turned off.

Create a Cookie with JavaScript

JavaScript can create, read, and delete cookies with the `document.cookie` property.