JavaScript

By Utkarsh Mehta

Introduction

Introduction to JavaScript

1. JavaScript is the world's most popular programming language.
2. JavaScript is the programming language of the Web.
3. JavaScript is easy to learn.
4. JavaScript is free to use for everyone.
5. JavaScript is an interpreted language. (Does not compile to machine code but runs line by line.)
6. Can be used inside browser(Front-end) to manipulate HTML elements.
7. Can use used on Server side (Backend) to run services, api servers, etc.
8. Can be used to develop mobile apps for iOS, Android, etc.
9. Desktop applications, IOT projects & much more.

History of JavaScript

JavaScript was invented by Brendan Eich in 1995.

It was developed for Netscape 2, and became the ECMA-262 standard in 1997.

After Netscape handed JavaScript over to ECMA, the Mozilla foundation continued to develop JavaScript for the Firefox browser. Mozilla's latest version was 1.8.5. (Identical to ES5).

Internet Explorer (IE4) was the first browser to support ECMA-262 Edition 1 (ES1).

1997 (ES1) ECMAScript 1 was released

1998 (ES2) ECMAScript 2 was released

1999 (ES3) ECMAScript 3 was released

2008 (ES4) ECMAScript 4 was abandoned

2009 (ES5) ECMAScript 5 was released

2015 (ES6) ECMAScript 6 was released

Versions & Features

Ver	Official Name	Description
ES1	ECMAScript 1 (1997)	First edition
ES2	ECMAScript 2 (1998)	Editorial changes
ES3	ECMAScript 3 (1999)	Added regular expressions Added try/catch Added switch Added do-while
ES4	ECMAScript 4	Never released
ES5	ECMAScript 5 (2009) Read More	Added "strict mode" Added JSON support Added String.trim() Added Array.isArray() Added Array iteration methods Allows trailing commas for object literals
ES6	ECMAScript 2015 Read More	Added let and const Added default parameter values Added Array.find() Added Array.findIndex()
	ECMAScript 2016 Read More	Added exponential operator (**) Added Array.includes()
	ECMAScript 2017 Read More	Added string padding Added Object.entries() Added Object.values() Added async functions Added shared memory
	ECMAScript 2018 Read More	Added rest / spread properties Added asynchronous iteration Added Promise.finally() Additions to RegExp

Setting up the Environment

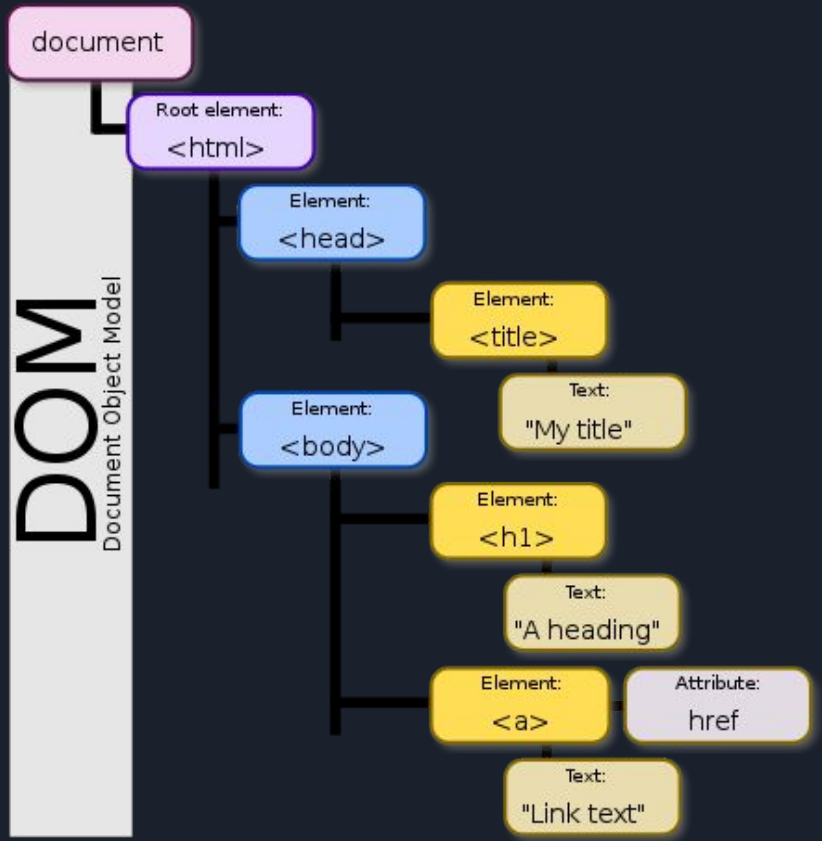
1. Demo of JavaScript in browser
2. Chrome's V8 engine (Runtime JavaScript Engine)
3. Installation of Node.js
4. Install & overview of VSCode
5. Hello World Program

Basics

DOM (Document Object Model)

The Document Object Model (DOM) is the data representation of the objects that comprise the structure and content of a document on the web.

- HTML => <html>, <head>, <body>, <p>, <h1>, etc.
- Representation of html elements of the page in the form of a tree.
- DOM can be accessed using keyword `document` in JavaScript.
- You can methods like `getElementById()`, `getElementsByName()`,etc to select a particular elements or elements from dom.
- Demo (Browser)



Basic Output

JavaScript can "display" data in different ways:

1. Writing into an HTML element, using `innerHTML`.
2. Writing into the HTML output using `document.write()`.
3. Writing into an alert box, using `window.alert()`.
4. Writing into the browser console, using `console.log()`.
5. Demo.

Variables

All JavaScript variables must be identified with unique names called identifiers.

Identifiers can be short names (like x and y) or more descriptive names (age, sum, totalVolume).

The general rules for constructing names for variables (unique identifiers) are:

- Names can contain letters, digits, underscores, and dollar signs.
- Names must begin with a letter
- Names can also begin with $ and _
- Names are case sensitive (y and Y are different variables)
- Reserved words (like JavaScript keywords) cannot be used as names let, await, async, const, etc.

You can declare variables using `let, var & const.`

VAR vs LET vs CONST

	var	let	const
Stored in Global Scope	✓	✗	✗
Function Scope	✓	✓	✓
Block Scope	✗	✓	✓
Can Be Reassigned?	✓	✓	✗

Data Types

There are different data types supported:

1. String
2. Number
3. Boolean
4. Array
5. Object
6. Undefined

Type Conversions

Original Value	Converted to Number	Converted to String	Converted to Boolean
false	0	"false"	false
true	1	"true"	true
0	0	"0"	false
1	1	"1"	true
"0"	0	"0"	**true**
"000"	0	"000"	**true**
"1"	1	"1"	true
NaN	NaN	"NaN"	false
Infinity	Infinity	"Infinity"	true
-Infinity	-Infinity	"-Infinity"	true
""	**0**	""	**false**
"20"	20	"20"	true
"twenty"	NaN	"twenty"	true
[ ]	**0**	""	true
[20]	**20**	"20"	true
[10,20]	NaN	"10,20"	true
["twenty"]	NaN	"twenty"	true
["ten","twenty"]	NaN	"ten,twenty"	true
function(){}	NaN	"function(){}"	true
{ }	NaN	"[object Object]"	true
null	**0**	"null"	false
undefined	NaN	"undefined"	false

String Methods

1. .length
2. .slice(start, end)
3. substring(start, end)
4. substr(start, length)
5. replace()
6. toUpperCase()
7. toLowerCase()
8. concat()
9. trim()
10. padStart()
11. padEnd()

Arrays and Objects

Array in JavaScript is a collection data that can be accessed using an index. Array can store multiple data types.

Array uses numbered indices.

Objects are {key, value} pairs. You can access the value by using key.

Objects use named indices.

In JavaScript, almost "everything" is an object.

Working with Date and Time

Expressions and Operations

Flow Control

If Statements

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

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

```
if (condition1) {
  //  block of code to be executed if condition1 is true
} else if (condition2) {
  //  block of code to be executed if the condition1 is false and condition2 is true
} else {
  //  block of code to be executed if the condition1 is false and condition2 is false
}
```

If statement helps us to run code blocks conditionally.

Switches

```
switch(expression) {
  case x:
    // code block
    break;
  case y:
    // code block
    break;
  default:
    // code block
}
```

The switch statement is used to perform different actions based on different conditions.

This is how it works:

- The switch expression is evaluated once.
- The value of the expression is compared with the values of each case.
- If there is a match, the associated block of code is executed.
- If there is no match, the default code block is executed.

Loops

Loops help you execute a block of code repeatedly.

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

For loop

```
for (statement 1; statement 2; statement 3) {
  // code block to be executed
}
```

Statement 1 is executed (one time) before the execution of the code block.

Statement 2 defines the condition for executing the code block.

Statement 3 is executed (every time) after the code block has been executed.

For In Loop

```
for (key in object) {
  // code block to be executed
}
```

The JavaScript for in statement loops through the properties of an Object and elements of an array.

For Of

```
for (variable of iterable) {
  // code block to be executed
}
```

The JavaScript for of statement loops through the values of an iterable object. It lets you loop over iterable data structures such as Arrays, Strings, Maps, NodeLists, and more.

- **variable** - For every iteration the value of the next property is assigned to the variable. Variable can be declared with const, let, or var.
- **iterable** - An object that has iterable properties.

While Loop

```
while (condition) {
  // code block to be executed
}
```

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

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.

```
do {
  // code block to be executed
}
while (condition);
```

Functions

A JavaScript function is a block of code designed to perform a particular task.

A JavaScript function is executed when "something" invokes it (calls it).

```
function name(parameter1, parameter2, parameter3) {
  // code to be executed
}
```

Scope

There are three types of Scopes:

1. Global Scope
2. Function Scope.
3. Block Scope

Error Handling

The try statement lets you **test a block of code for errors**.

The catch statement lets you **handle the error**.

The throw statement lets you **create custom errors**.

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

JavaScript Classes

Creating a Class

```
class ClassName {
  constructor() { ... }
}
```

Use the keyword **class** to create a class.

Always add a method named **constructor**():

Working with JSON

JSON stands for JavaScript Object Notation

JSON is a text format for storing and transporting data

JSON is "self-describing" and easy to understand

Two main methods related to JSON:

- `JSON.parse`
- `JSON.stringify`

Modern JavaScript and Asynchronous Programming

Arrow Functions

Arrow functions were introduced in ES6.

Arrow functions allow us to write shorter function syntax

```
let myFunction = (a, b) => a * b;
```

```
hello = function() {
  return "Hello World!";
}
```

```
hello = () => {
  return "Hello World!";
}
```

```
hello = () => "Hello World!";
```

```
hello = val => "Hello " + val;
```

```
hello = (val) => "Hello " + val;
```

Callback Functions

A callback is a function passed as an argument to another function

This technique allows a function to call another function

A callback function can run after another function has finished

```
function myDisplayer(some) {
  console.log(some);
}
function add(num1, num2, myCallback) {
  myCallback(num1 + num2);
}
add(5, 5, myDisplayer);
```

Promises

"Producing code" is code that can take some time

"Consuming code" is code that must wait for the result

A Promise is a JavaScript object that links producing code and consuming code

```
let myPromise = new Promise(function(myResolve, myReject) {
// "Producing Code" (May take some time)

  myResolve(); // when successful
  myReject();  // when error
});

// "Consuming Code" (Must wait for a fulfilled Promise)
myPromise.then(
  function(value) { /* code if successful */ },
  function(error) { /* code if some error */ }
);
```

Async/Await

async and await make promises easier to write

async makes a function return a Promise

await makes a function wait for a Promise