JavaScript

Table of Contents

[Basics 2](#_Toc100577375)

[Front-End 3](#_Toc100577376)

[History 4](#_Toc100577377)

[Datatypes 5](#_Toc100577378)

[Including JavaScript 5](#_Toc100577379)

[Basic Scripting 6](#_Toc100577380)

[Conditional Statements 7](#_Toc100577381)

[Loose and Strict Equality and Inequality Operators 8](#_Toc100577382)

[User Input 8](#_Toc100577383)

[Functions 8](#_Toc100577384)

[Arrays 9](#_Toc100577385)

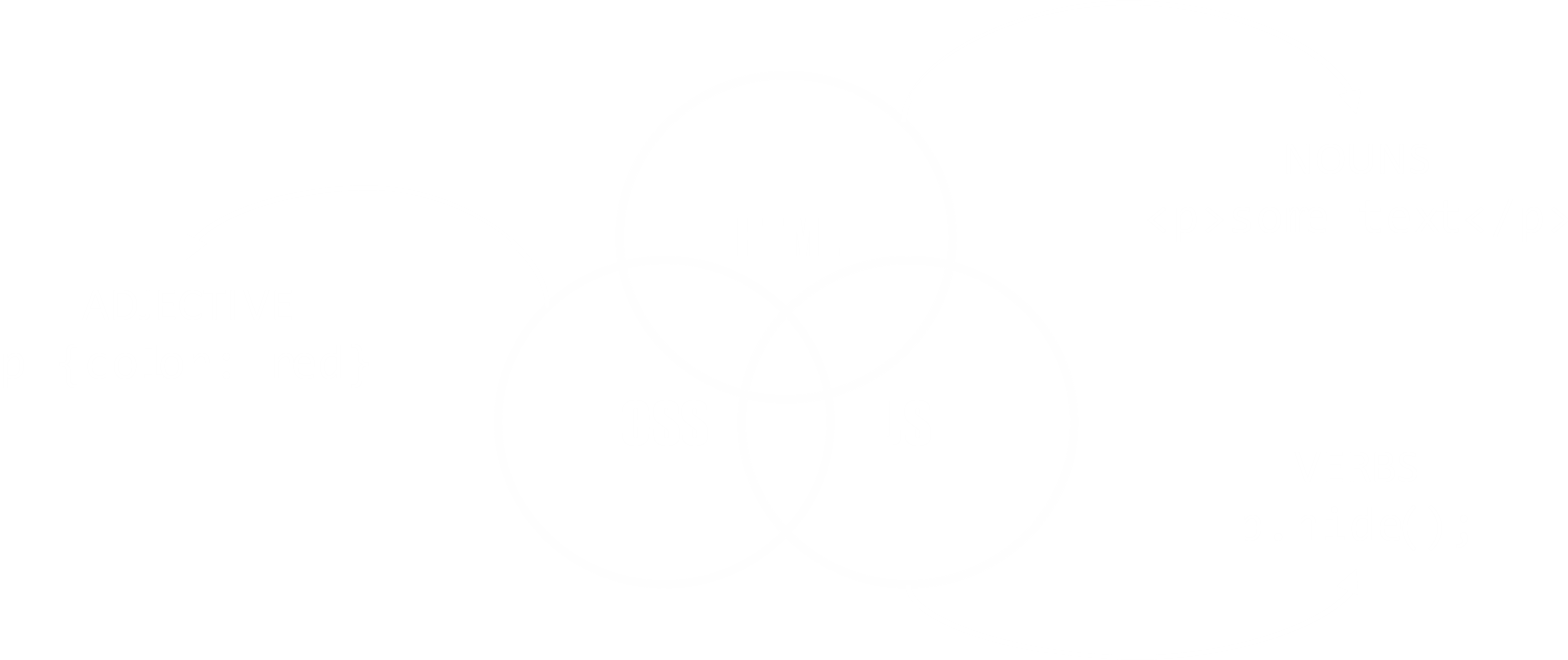
[Objects 10](#_Toc100577386)

## Basics

* High-Level language, meaning we do not need to worry about things like memory management
* Object-Oriented language
* Used in both the front-end (in frameworks like Angular, React, etc.) and the back-end (in frameworks like Node)
* Can be used to create both dynamic and interactive content for web pages
* Supports both C++ and Java syntax
* Supports dynamic typing, meaning variable data types do not need to be explicitly specified
* Code is interpreted, not compiled, meaning instead of using a compiler, an interpreter embedded in all modern browsers is used

## Front-End

The front-end is basically made up of three parts, HTML, CSS and JavaScript. HTML provides the content of the page, CSS provides the styling for that content, and JavaScript provides **behaviour** for that content. An example should make this clear.



Here, we are using HTML to add a paragraph of text using the <p> tag. CSS is being used to specify a colour for all <p> tags. JavaScript is being used to hide the <p> tags.

## History

1995 Brendan Eich creates JavaScript (called Mocha) while working for Netscape

1996 Name changes from Mocha to LiveScript to JavaScript

Microsoft launches Internet Explorer, copying JavaScript (named JScript)

1997 First official standard for JavaScript, ECMAScript1 (ES1) is released by ECMA

2009 ECMAScript 5 released with new features

2016 ES6 (ES2015) released

biggest update ever for JavaScript

contains most of the core features used today

ECMA starts releasing an update annually

All versions of JavaScript are **backwards compatible**. This was done to ensure that modern browsers being used to navigate to very old websites do not cause the websites to break.

On the flip side, to ensure that older browsers do not break when they navigate to websites that use newer features they do not support, certain techniques called **transpiling** and **polyfilling** are used. A tool called **Babel** is used to transpile modern JS code back to ES5. ES5 is being considered the baseline since it is the most used JavaScript version that is supported by all browsers (up to Internet Explorer v9, released in 2011).

Modern JS is called ES6+. Browser compatibility for ES6 features can be found [here](http://kangax.github.io/compat-table/es6/).

## Datatypes

As mentioned before, JavaScript supports **dynamic typing**, meaning we do not have to specify a datatype when we create a variable. However, the values we assign to those variables do have datatypes.

JavaScript supports the following **primitive datatypes**:

let someNumber = 5; *// all numbers are floating-point*let someString = "Hello"; *// sequence of characters*let someBoolean = true; *// Booleans*let someVariable; *// undefined*let someValue = null; *// No value*

JS

In addition to these, JavaScript also supports **objects**, since it is an object-oriented language, the **Symbol** datatype and the **BigInt** datatype, which is used for numbers larger than .

## Including JavaScript

We can include JavaScript code into an HTML page in two ways, using inline code and using an external JavaScript file.

To include **Inline JavaScript**, we can place the JavaScript code inside <script> tags in the header section of the HTML page.

<head>  
 <script>  
 *alert*("This is an alert message.")  
 </script>  
</head>

HTML

The above code will cause a popup message to appear in the browser when the page first loads.

To use an **external JavaScript file**, we can include the file anywhere in the HTML page like this:

<script src="script.js"></script>

HTML

## Basic Scripting

*// single line comment  
  
/\*  
 Multi-line comment  
 \*/  
  
alert*("This is an alert.") *// alert messages*var a = 20; *// variables before ES6; function-scoped (global)*let b = 20; *// variables from ES6 onwards; block-scoped (local)*const c = 20; *// constants; cannot be undefined*console.log("Hello World"); *// printing to browser console*console.log(typeof a); *// printing a's datatype*

*// template literals*let d = `The total is ${a + b}`; *// computations*let e = `This is a two-line  
message`;  
  
let f = *Number*("20"); *// typecasting*f = 20 - "10"; *// implicit typecasting (type coercion); result = 10*f = 20 + "5"; *// type coercion risky; result = 205; concat has precedence*

for (let i=0; i<10; i++) {  
 console.log(i);  
 break;  
}

JS

## Conditional Statements

let someBoolean = true;  
  
*// if-else statement*if (someBoolean) {}  
else {}  
  
*// switch statement*switch (someBoolean) {  
 case true:  
 break;  
 default:  
 break;  
}

JS

## Loose and Strict Equality and Inequality Operators

10 == "10"; *// loose equality; returns true due to type coercion*10 === "10"; *// strict equality; returns false; best to use in most cases  
// also applies to != and !==, the loose and strict inequality operators*

JS

## User Input

let userInput = *prompt*("Enter a number: ") *// causes input pop-up*

*// input is always a string*

JS

## Functions

*// function declaration*function *totalFruits*(apples, oranges) {  
 return *Number*(apples) + *Number*(oranges);  
}  
  
*// function call*let total = *totalFruits*(10, 20);  
  
*// function expression*let *age* = function (birthYear) {  
 return 2020 - birthYear;  
}  
  
*// function call*let currentAge = *age*(2000);

*// arrow function*let *age* = (birthYear) => 2020 - birthYear;

*// function call*let currentAge = *age*(2000);

JS

Function declarations can be called before they are declared but function expressions cannot.

## Arrays

const boys = ["John", "Alex", "Adrian"]; *// array initialization*const girls = new Array("Cassandra", "Lara", "Yuna");girls[2] = "Amina"; *// can change single elements; but not entire array*let person1 = boys[0]; *// access element*

let totalBoys = boys.length; *// array length*const boyAndGirlAge = ["John", 25, "Yuna", 24]; *// different data types*

const people = [boys, girls]; *// array of arrays*boys.push("Alan"); *// adds element to array; returns new length of array*boys.pop(); *// removes element at end; returns popped element*boys.unshift("Johnathan"); *// adds element to beginning*boys.shift(); *// removes first element*let index = boys.indexOf("Johnathan"); *// returns -1 if not present*

let present = boys.includes("Alex"); *// returns true/false*

JS

## Objects

*// objects use key-value pairs*const employeeDetails = {  
 Name: "John",  
 Position: "Engineer",  
 Age: 25,  
 Salary: 1200,  
 Gadgets: ["Smartphone", "iPad", "Smartwatch"],  
};

*// property access*let name1 = employeeDetails.Name;  
let name2 = employeeDetails["Name"];  
  
*// adding properties*employeeDetails.LastName = "Doe";  
employeeDetails["LastName"] = "Doe";

*// method declarations must use function expressions or arrow functions*const employeeDetails2 = {  
 Name: "John",  
 birthYear: 2000,  
 calculateAge1: function(birthYear) {  
 return 2020 - birthYear;  
 },  
 calculateAge2: () => {  
 return 2020 - this.birthYear;  
 },  
 calculateAge3: () => {  
 this.age = 2020 - this.birthYear; *// creates new property* return this.age;  
 }  
};  
  
*// method call*employeeDetails2.calculateAge2();  
employeeDetails2["calculateAge1"](2000);  
employeeDetails2.calculateAge3();  
let age = employeeDetails2.age;

JS