Introduction to JavaScript

PROGRAMMING LANGUAGE

Steffen Holanger - Boitano

Christoffer Træen - Twoday

What is JavaScript?

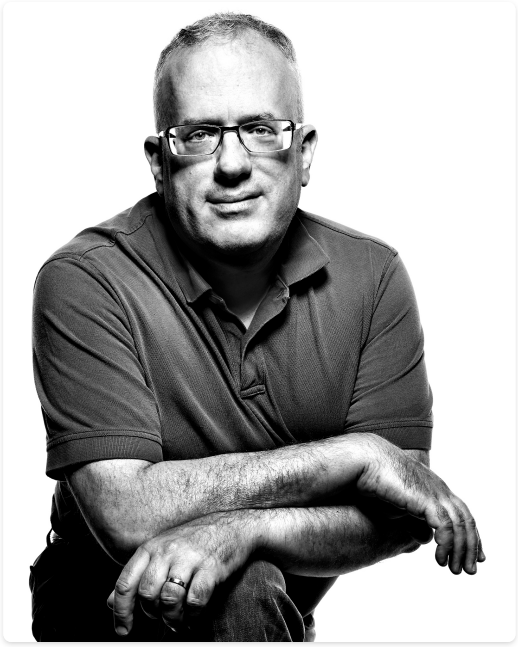
JavaScript is a high-level, interpreted programming language primarily used for:

- Creating interactive web pages
- Client-side web development
- Server-side applications (Node.js)
- Mobile applications
- Game development
- Desktop applications

ⓘ Not to be confused with Java - they're completely different languages!

A Brief History

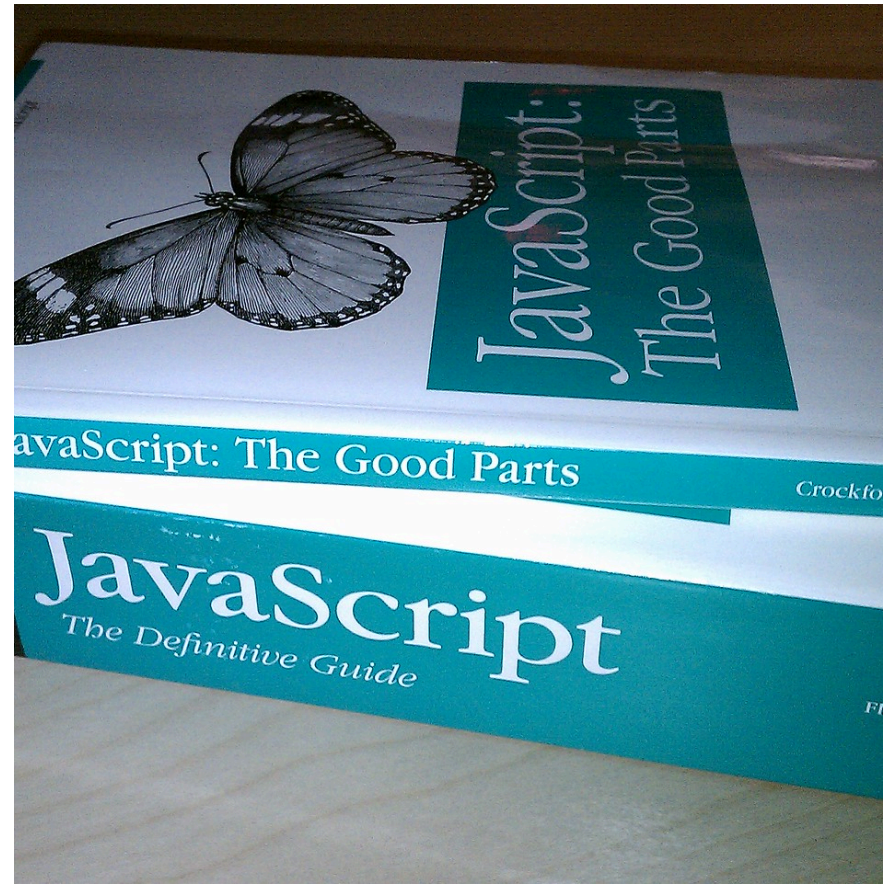
- **1995**: Created by Brendan Eich at Netscape in just 10 days
- **1996**: Submitted to ECMA International for standardization
- **1997**: ECMAScript 1 released (official standard)
- **2005**: AJAX popularized (asynchronous JS)
- **2009**: ECMAScript 5 with important improvements
- **2015**: ECMAScript 2015 (ES6) - major update
- **Now**: Yearly updates (ES2022, ES2023, etc.)



Brendan Eich, creator of JavaScript

Growing pains

ES6 Revolution (2015): Added classes, arrow functions, let/const, modules, promises, and more.



Where JavaScript Runs

BROWSER

- Client-side execution
- DOM manipulation
- Interactivity
- Limited access to system resources

```
<script>
  document
    .getElementById('demo')
    .innerHTML = 'Hello JavaScript!'
</script>
```

NODE.JS

- Server-side execution
- File system access
- Network operations
- System resource access

```
const http = require('http')
const server = http.createServer((req, res) => {
  res.end('Hello from Node.js!')
})
server.listen(3000)
```

OTHER ENVIRONMENTS

- Deno, Bun (newer runtimes)
- Embedded systems
- Mobile frameworks (React Native)
- Desktop apps (Electron)

JavaScript Basics: Syntax

By convention `camelCase` is being used for all variables.

Variables

```
var oldWay = 'Avoid using var' // Function-scoped
let mutable = 'Can be changed' // Block-scoped
const immutable = "Can't change*" // Block-scoped
```

Comments

```
// comment
// const iAmACommentedOutVariable = 'not running 🚶'
const iAmNotCommentedOut = 'Running 🏃'

/*
const iAmInACommentBlock = '🤷'
*/
```

Block scope – how it works

A block is all code between `{` and `}`

```
let a = 'Hi'
{
  console.log(a)
  a = 'hello'
  {
    console.log(a)
    const b = 'bye'
  }
  console.log(a)
  console.log(b)
  var c = 'I am global'
}

console.log(c)
```

```
Hi
hello
hello
ReferenceError: b is not defined
```

Data Types

```
// Primitive types
let name = 'John' // String
let age = 30 // Number (double-precision 64 bit)
let isActive = true // Boolean

// Special primitives
let empty = null // Null
let notDefined // Undefined

// Complex types
let person = { name, age } // Object
let colors = ['red', 'blue'] // Array
```

Type Coercion in JavaScript

JavaScript automatically converts types when needed - this is called "type coercion"

Implicit Coercion (Automatic)

```
// String + Number → String
console.log("5" + 3)      // "53"

// Number / String -> Number
console.log(12 / "6")     // 2

// converting string to number
console.log(typeof +"5" ) // number
console.log(typeof "5" )  // string
```

```
53
2
number
string
```

```
// Boolean → Number
console.log(true + 1)    // 2
console.log(false + 1)   // 1

// Comparison with ==
console.log(
  "0" == 0,              // true
  false == 0,            // true
  null == undefined,     // true
  "0" == false           // true
)
```

```
2
1
true, true, true, true
```

Explicit Coercion (Manual)

```
// To String
String(123)         // "123"
(123).toString()    // "123"

// To Number
Number("123")       // 123
parseInt("123")     // 123
+"123"              // 123

// To Boolean
Boolean(0)          // false
Boolean("")         // false
Boolean(null)       // false
Boolean(undefined)  // false
!!123               // true
```

Functions

Functions are first class citizens in javascript

```
// Traditional function
function greet(name) {
  return `Hello, ${name}!`
}

// Anonymous
(function (name) {
  return  `Hello ${name}`
})()

// Arrow function
const greet = (name) => {
  return `Hello, ${name}!`
}

// One-liner arrow function
const greet = (name) => `Hello, ${name}!`
```

Control Flow

```
// Conditionals
if (age >= 18) {
  console.log('Adult')
}

// Loops
for (let i = 0; i < 5; i++) {
  console.log(i)
}
colors.forEach((color) => {
  console.log(color)
})

while(timeIsPassing) {
  console.log("Keep living")
}

do {
  console.log("Learn to code")
} while (youCan)
```

```
for (const value of [1, 2, 3]) {
  console.log(value);
}

const person = { name: 'John', age: 30, job: 'developer' }

for (const key in person) {
  const value = person[key]
  console.log(`${key}: ${value}`)
}
```

```
1
2
3
name: John
age: 30
job: developer
```

JavaScript in Action: DOM Manipulation

```
// This would select elements in a real webpage
console.log('Simulating DOM manipulation...')
const element = document.getElementById('demo')

// Simulating element selection
console.log('Initial element:', element)

// Changing content
element.innerHTML = 'Hello, JavaScript!'
console.log('After setting innerHTML:', element)

// Simulating adding an event listener
console.log('Adding click event listener...')
element.addEventListener('click', (event) => {
  console.log('Clicking target', event.target)
  })

element.innerHTML = 'You clicked me!'
console.log('Final element state:', element)
```

Real DOM Example

```
<!DOCTYPE html>
<html>
  <body>
    <!-- Element to manipulate -->
    <h2 id="demo">A Heading</h2>
    <button id="btn">Click me</button>

    <script>
      // Get element references
      const demo = document.getElementById('demo')
      const btn = document.getElementById('btn')

      // Change content immediately
      demo.innerHTML = 'Hello JavaScript!'

      // Add event listener to button
      btn.addEventListener('click', function () {
        demo.style.color = 'red'
        demo.innerHTML = 'Text changed!'
      })
    </script>
  </body>
</html>
```

Working with Data

Objects

```
// Creating an object
const person = {
  name: 'Sarah',
  age: 28,
  isEmployed: true,
}

// Accessing properties
console.log(person.name) // "Sarah"
console.log(person['age']) // 28
person.location = 'New York' // Adding property
```

Arrays

```
// Creating an array
const fruits = ['Apple', 'Banana', 'Cherry']

// Accessing elements
console.log(fruits[0]) // "Apple"

// Array methods
fruits.push('Date') // Add to end
fruits.pop() // Remove from end
const citrus = fruits.slice(1, 2) // Extract
fruits.forEach((f) => console.log(f)) // Iterate
```

Modern Array Methods

```
const numbers = [1, 2, 3, 4, 5]

// map: transform each element
const doubled = numbers.map((x) => x * 2)
console.log('Doubled:', doubled)

// filter: keep elements that pass a test
const evenNumbers = numbers.filter((x) => x % 2 === 0)
console.log('Even numbers:', evenNumbers)

// reduce: accumulate values
const sum = numbers.reduce((acc, curr) => acc + curr, 0)
console.log('Sum:', sum)

// find: get first matching element
const found = numbers.find((x) => x > 3)
console.log('First number > 3:', found)
```

```
Doubled:, [2, 4, 6, 8, 10]
Even numbers:, [2, 4]
Sum:, 15
First number > 3:, 4
```

Asynchronous JavaScript

JavaScript handles async operations with:

- Callbacks (traditional)
- Promises (ES6)
- Async/await (modern)

Callbacks

```
function fetchData(callback) {
  setTimeout(() => {
    callback('Data received')
  }, 1000)
}

console.log('start')
fetchData(function (data) {
  console.log(data) // After 1 second: "Data received"
})
```

```
start
Data received
```

Promises

```
function fetchData() {
  return new Promise((resolve, reject) => {
    setTimeout(() => {
      resolve('Data received')
    }, 1000)
  })
}

fetchData()
  .then((data) => console.log(data))
  .catch((error) => console.error(error))
```

```
Data received
```

Async/Await

```
async function getData() {
  try {
    const data = await fetchData()
    console.log(data)
  } catch (error) {
    console.error(error)
  }
}
```

Common Use Cases

Form Validation

```
document.querySelector('form')
  .addEventListener('submit', (event) => {
    const emailInput = document.getElementById('email')
    const emailRegex = /^[^\s@]+@[^\s@]+\.[^\s@]+$/

    if (!emailRegex.test(emailInput.value)) {
      event.preventDefault()
      showError('Please enter a valid email')
    }
})
```

API Requests

```
async function getUsers() {
  try {
    const response =
    await fetch('https://api.example.com/users')

    if (!response.ok) {
      throw new Error(`HTTP error: ${response.status}`)
    }

    const users = await response.json()
    displayUsers(users)
  } catch (error) {
    console.error('Fetch error:', error)
  }
}
```

DOM Updates

```
function updateCounter() {
  const counterElement = document
    .getElementById('counter')

  let count = parseInt(counterElement.textContent)
  counterElement.textContent = count + 1

  if (count >= 10) {
    counterElement.style.color = 'red'
  }
}
```

Local Storage

```
// Save data
function saveSettings(settings) {
  localStorage
    .setItem('userSettings', JSON.stringify(settings))
}

// Load data
function loadSettings() {
  const data = localStorage.getItem('userSettings')
  return data ? JSON.parse(data) : defaultSettings
}
```

Falsy values

Value	Type	Description
null	Null	The keyword `null` — the absence of any value.
undefined	Undefined	`undefined` — the primitive value.
`false`	Boolean	The keyword `false`.
NaN	Number	`NaN` — not a number.
`0`	Number	The `Number` zero, also including `0.0`, `0x0`, etc.
`-0`	Number	The `Number` negative zero, also including `-0.0`, `-0x0`, etc.
`0n`	BigInt	The `BigInt` zero, also including `0x0n`, etc.
`""`	String	Empty string value, also including `''` and ``` ``.

JavaScript Best Practices

- Use `const` and `let` instead of `var`
- Prefer strict equality (`===`) over loose equality (`==`) — Example: `"0" === 0` is `false`, while `"0" == 0` is `true`
- Avoid global variables
- Use meaningful variable and function names
- Comment your code (but make it self-documenting)
- Handle errors properly
- Use modern ES6+ features
- Follow a style guide (Airbnb, Google, Standard)
- Break code into small, reusable functions
- Use linters (ESLint) and formatters (Prettier)
- Test your code
- Consider using TypeScript for large projects

Bad vs Good Code Example

```
// Bad code
var x = function (y) {
  if (y == null) y = 42
  var z = y + 5
  return z
}

// Good code
const addFive = (number = 42) => {
  return number + 5
}
```

Modern JavaScript Ecosystem

Resources for Learning

FOR BEGINNERS

- freeCodeCamp
- JavaScript.info
- MDN Web Docs
- Codecademy

FOR PRACTICE

- Exercism.io
- CodeWars
- LeetCode
- Frontend Mentor

ADVANCED TOPICS

- You Don't Know JS (book series)
- Eloquent JavaScript (book)
- JavaScript: The Good Parts (book)
- Frontend Masters (courses)

Thank You!

Start writing JavaScript today!

Questions? Let's discuss!