

CRASH COURSE

# JavaScript Crash Course

from zero JavaScript → able to build small interactive programs

---

## DURATION

**Roughly 4-6 hours**

... or spread it across a few days

## START POINT

**Zero JavaScript**

no prior programming needed

## OUTCOME

**Small interactive programs**

written by you, from scratch

Here's a compact course designed to take you from zero JavaScript → able to build small interactive programs. You can work through it in roughly 4-6 hours, or spread it across a few days.

# Contents

- 1** What JavaScript is
- 2** Where JavaScript fits
- 3** A short history
- 4** JavaScript vs Java
- 5** How to run JavaScript
- 6** Variables
- 7** Operators
- 8** Strings
- 9** Conditions
- 10** Functions
- 11** Arrays
- 12** Objects
- 13** Loops
- 14** Destructuring
- 15** Spread syntax
- 16** The DOM
- 17** Events
- 18** Async JavaScript
- 19** Modules
- 20** Classes
- 21** Mini Project: Counter
- 22** Mini Project: Todo Logic
- 23** The 80/20 of JavaScript

# 01 What JavaScript is

*JavaScript is the programming language that makes websites interactive.*

## HTML

provides structure

## CSS

provides appearance

## JavaScript

provides behavior

**For example:**

### JavaScript

```
console.log("Hello, world!");
```

`console.log()` prints something to the browser's developer console.

# Where JavaScript fits in the front-end

*JavaScript (JS) is a computer programming language used to make websites and applications dynamic and interactive.*



## Front-end languages

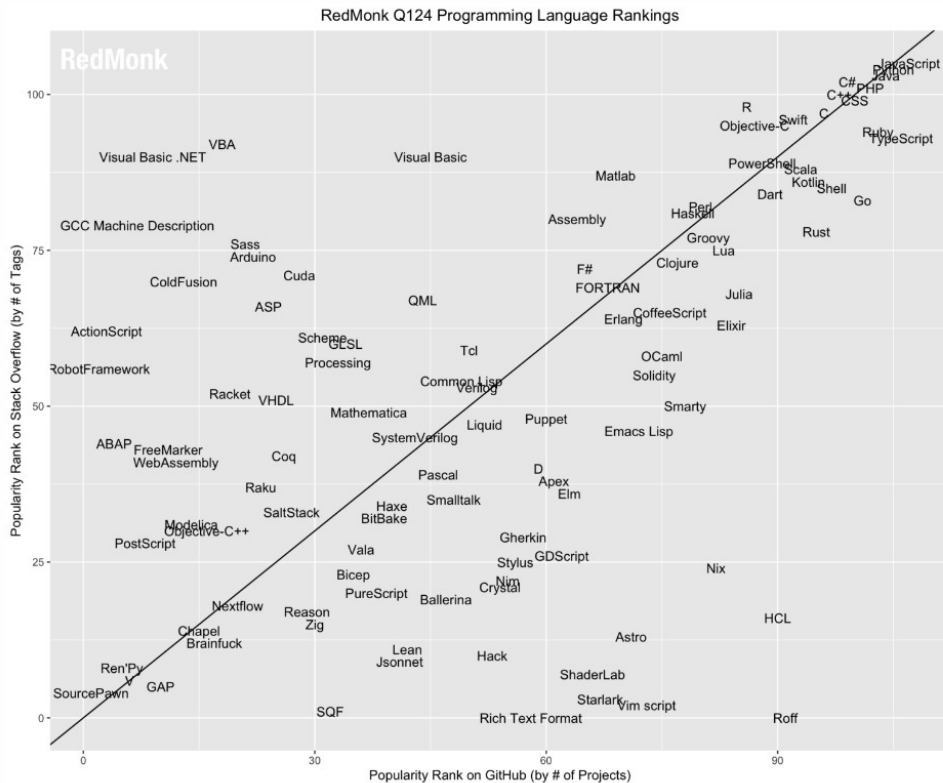
**HTML** — what do you want on your website?

**CSS** — how do you want it to look?

**JavaScript** — what do you want it to do?

## 02 Ranking of Programming Languages (2024)

*RedMonk ranks languages by GitHub projects and Stack Overflow activity.*



### Top tier (both axes)

JavaScript · Python · Java · C# · PHP · C++ · CSS · C

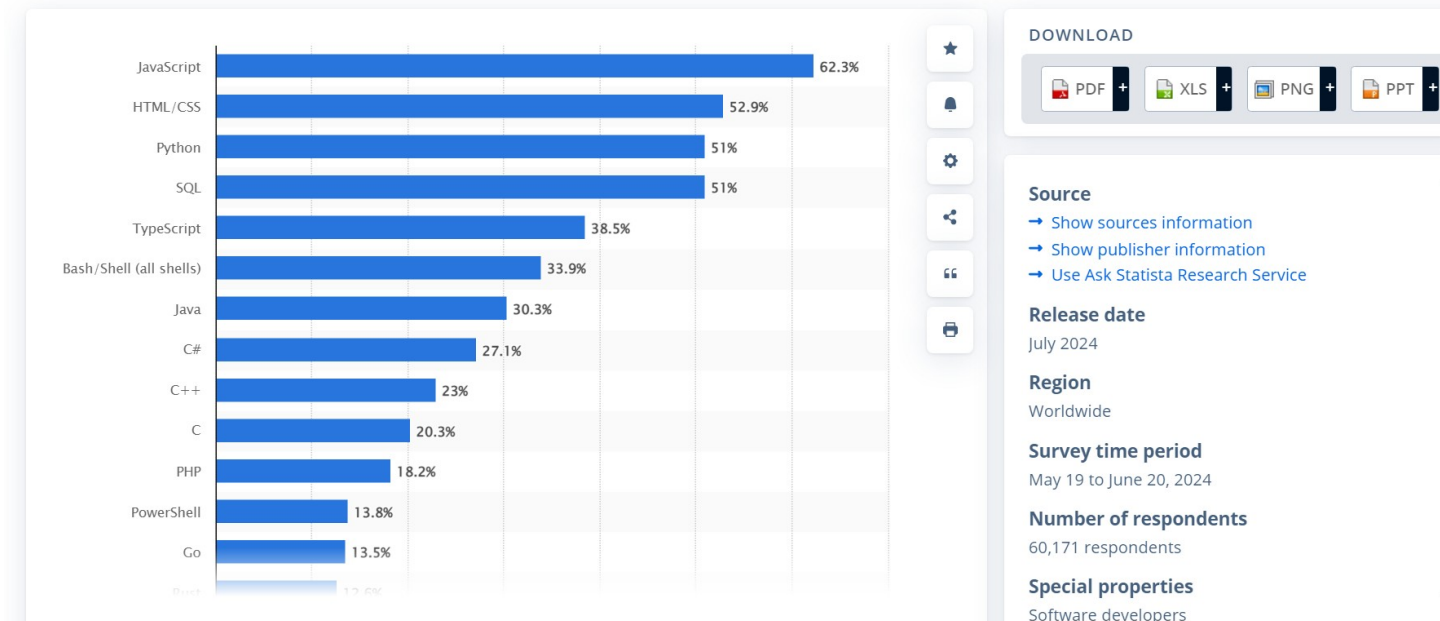
The y-axis is popularity rank on Stack Overflow (by number of tags); the x-axis is popularity rank on GitHub (by number of projects). JavaScript sits in the top group on both.

Source: <https://redmonk.com/sograde/2024/03/08/language-rankings-1-24/>

## 02 Most used programming languages (2024)

*Most used programming languages among developers worldwide, 2024.*

Most used programming languages among developers worldwide as of 2024



### JavaScript leads again

**62.3%** of surveyed developers use JavaScript — ahead of HTML/CSS (52.9%), Python (51%) and SQL (51%).

60,171 developers surveyed · 2024

Source: <https://www.statista.com/statistics/793628/worldwide-developer-survey-most-used-languages/>

# A short history of JavaScript

*From a browser scripting language to a yearly-released standard.*

| Year | Event / Version           | Key Milestones                                                                         |
|------|---------------------------|----------------------------------------------------------------------------------------|
| 1995 | JavaScript by Netscape    | JavaScript (Mocha/LiveScript) created for dynamic web content.                         |
| 1996 | Microsoft JScript         | Microsoft introduces JScript in Internet Explorer.                                     |
| 1997 | ECMAScript 1 (ES1)        | ECMAScript standard created by ECMA International.                                     |
| 1999 | ECMAScript 3 (ES3)        | Introduced regular expressions, <code>try/catch</code> , first widely adopted version. |
| 2009 | ECMAScript 5 (ES5)        | Added JSON support, strict mode, improved array methods.                               |
| 2015 | ECMAScript 6 (ES6/ES2015) | Major update: let/const, arrow functions, classes, modules, Promises.                  |
| 2016 | ECMAScript 7 (ES7) onward | Annual releases with incremental features.                                             |
| 2024 | ECMAScript 2024 (ES2024)  | Latest: Function decorators, class field improvements, RegExp updates.                 |

**1995** created at Netscape

**1997** ECMAScript standard

**2015** ES6: let/const, arrow functions, classes, modules, Promises

**2016** → annual ES releases

# JavaScript vs Java

*Two languages with similar names and almost nothing else in common.*

| Aspect             | JavaScript                                   | Java                                                                      |
|--------------------|----------------------------------------------|---------------------------------------------------------------------------|
| Type               | Interpreted, scripting language              | Compiled, object-oriented programming language                            |
| Primary Use        | Client-side (browser) web development        | General-purpose, platform-independent applications (desktop, web, mobile) |
| Execution          | Runs in the browser (via JavaScript engine)  | Runs on the Java Virtual Machine (JVM)                                    |
| Syntax             | Dynamic typing, more flexible syntax         | Static typing, strict object-oriented syntax                              |
| Concurrency        | Single-threaded, asynchronous (event-driven) | Multi-threaded using threads and concurrency tools                        |
| Inheritance        | Prototype-based inheritance                  | Class-based inheritance                                                   |
| Popular Frameworks | React, Angular, Node.js                      | Spring, Hibernate, Android SDK                                            |
| Developed by       | Originally Netscape, now ECMA International  | Sun Microsystems (now Oracle)                                             |

## Why the confusion?

JavaScript was named for marketing, not for Java. The two differ in typing, execution model and inheritance:

**JavaScript** — interpreted, dynamic typing, runs in the browser, prototype-based.

**Java** — compiled to bytecode, static typing, runs on the JVM, class-based.

# How to run JavaScript

*Three ways to execute JavaScript today.*

## 1. In the Browser

- Open browser (e.g., Chrome)
- Open Developer Tools (Right-click > Inspect > Console tab)
- Type JavaScript code in the Console

## 2. In HTML Files

- Embed JavaScript in `<script>` tags within an HTML file
- Open the HTML file in a browser
- Example: `<script>console.log("Hello, JavaScript!");</script>`

## 3. Using Node.js

- Install Node.js
- Create a .js file (e.g., app.js)
- Run the file in terminal: `node app.js`

# Your first HTML page with JavaScript

*Embed JavaScript in `<script>` tags within an HTML file.*

## HTML

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Document</title>
</head>
<body>

  <script>
    console.log("Hello World");
    alert("Hello World");
  </script>
</body>
</html>
```

## What happens

`console.log()` writes to the developer console.

`alert()` shows a pop-up dialog in the browser.

# Variables

*Variables store information.*

- Use **const** by default. Use **let** when the value needs to change.
- You'll occasionally encounter **var** in older JavaScript, but you generally shouldn't use it in new code.

## let & const

```
let name = "Alex";  
let age = 25;  
const country = "Canada";  
  
console.log(name);  
console.log(age);
```

## the important distinction

```
let score = 10;  
score = 20; // allowed  
  
const pi = 3.14159;  
pi = 4; // error  
  
var age = 25; // Use let instead of var
```

# Variables — basic data types

*Basic data types*

## data types

```
const name = "Alex";      // string
const age = 25;           // number
const isStudent = true;   // boolean
const nothing = null;     // null
let unknown;              // undefined
const list = [1, 2, 3];   // array
const user = { name: "John", age: 25 }; // object
```

You can inspect a value's type — use the `typeof()` method:

## typeof

```
console.log(typeof "hello"); // "string"
console.log(typeof 42);      // "number"
console.log(typeof true);    // "boolean"
```

## 06

# Variables — rules for naming

*Rules for naming JavaScript variables.*

- Must start with a letter, `_`, or `$` (cannot start with a number).
- Can contain letters, digits, `_`, and `$` (no spaces, no special characters).
- Case-sensitive (`user`  $\neq$  `User`).
- Cannot use reserved keywords (e.g., `let`, `class`, `return`).
- Use **camelCase** by convention for readability (`totalPrice`, `userName`).
- Make names descriptive and meaningful, not cryptic.

## Good vs bad:

### naming

```
let totalAmount = 100; // ✓ good
```

```
let 2price = 100; // ✗ bad
```

```
let return = 5; // ✗ bad
```

## Reading input from the user:

### prompt / alert

```
<script>
  name = prompt("What is your name?");
  alert("Hello " + name);
  console.log("Hello " + name);
</script>
```

## 07 Operators — arithmetic & comparison

*JavaScript supports normal arithmetic:*

### arithmetic

```
const x = 10;  
const y = 3;  
  
console.log(x + y); // 13  
console.log(x - y); // 7  
console.log(x * y); // 30  
console.log(x / y); // 3.333...  
console.log(x % y); // 1  
console.log(x ** y); // 1000
```

Comparisons produce booleans:

### comparisons

```
10 > 5      // true  
10 < 5      // false  
10 === 10   // true  
10 !== 5    // true
```

# Operators — strict equality & logic

*Prefer strict equality:*

## === vs ==

```
5 === "5" // false
5 == "5"  // true
```

=== compares both value and type, which makes behavior more predictable.

Logical operators are **AND (&&)**, **OR (||)**, **NOT (!)**:

## logical

```
let isAdult = true;
let hasID = false;

console.log(isAdult && hasID); // false
console.log(isAdult || hasID); // true
console.log(!isAdult);       // false
```

## with plain values

```
true && false // AND
true || false // OR
!true         // NOT
```

## 07 Operators — the Math object

*The Math object provides useful methods for mathematical operations.*

### common methods

```
console.log(Math.sqrt(16));      // 4
console.log(Math.pow(2, 3));     // 8
console.log(Math.abs(-10));      // 10
console.log(Math.floor(4.7));    // 4
console.log(Math.ceil(4.2));     // 5
console.log(Math.random());      // e.g. 0.3456
console.log(Math.PI);            // 3.141592653589793
```

### random range

```
// Random number between 1 and 10
let randomNum = Math.floor(Math.random() * 10) + 1;

console.log(randomNum);
```

**Math.sqrt(16)** → 4

**Math.floor(x)** rounds down, **Math.ceil(x)** rounds up

**Math.random()** returns a number between 0 and 1

# Strings

*Strings represent text.*

## concatenation

```
const firstName = "John";  
const lastName = "Smith";  
const fullName = firstName + " " + lastName;
```

Modern JavaScript commonly uses **template literals**:

## template literals

```
const name = "John";  
const age = 30;  
  
console.log(`My name is ${name} and I am ${age} years old.`);
```

## 08 Strings — declaring them

*Strings can be created using single quotes ('), double quotes ("), or template literals (`).*

### three ways

```
let singleQuote = 'Hello';
let doubleQuote = "World";
let templateLiteral = `Hello, ${doubleQuote}!`;

console.log(templateLiteral); // Hello, World!
```

### Only backticks interpolate

`${...}` only works inside template literals — single and double quotes give you a plain string.

Use `+` or template literals to combine strings:

### concatenation

```
let fullName = firstName + " " + lastName;
let welcome = `Hello, ${firstName} ${lastName}!`;
```

## 08 Strings — useful methods

*Useful string methods include:*

### string methods

```
const text = "JavaScript";
```

```
text.length  
text.toUpperCase()  
text.toLowerCase()  
text.includes("Script")  
text.indexOf("Script")  
text.slice(0, 4)
```

Called on the string with **dot notation**:

```
text.length → 10
```

```
text.toUpperCase() → "JAVASCRIPT"
```

```
text.toLowerCase() → "javascript"
```

```
text.includes("Script") → true
```

```
text.indexOf("Script") → 4
```

```
text.slice(0, 4) → "Java"
```

# Conditions

*Conditions allow programs to make decisions.*

## if / else

```
const age = 20;

if (age >= 18) {
  console.log("Adult");
} else {
  console.log("Minor");
}
```

You can have multiple conditions:

## else if

```
const score = 85;

if (score >= 90) {
  console.log("A");
} else if (score >= 80) {
  console.log("B");
} else if (score >= 70) {
  console.log("C");
} else {
  console.log("Needs improvement");
}
```

## 09 Conditions — the ternary operator

*A compact conditional is the ternary operator:*

### ternary operator

```
const age = 20;  
const status = age >= 18 ? "adult" : "minor";
```

### in practice

```
let isLoggedIn = true;  
let message = isLoggedIn ? "Welcome back!" : "Hello there!"  
console.log(message); // Welcome back!
```

Think:

```
condition ? valueIfTrue : valueIfFalse
```

# Functions

*Functions package reusable behavior.*

## function declaration

```
function greet(name) {  
    return `Hello, ${name}!`;  
}  
  
console.log(greet("Sam"));
```

Modern JavaScript also uses **arrow functions** heavily:

## arrow function

```
const greet = (name) => {  
    return `Hello, ${name}!`;  
};
```

## 10 Functions — short arrows & mental model

*A short version:*

short arrow

```
const double = number => number * 2;  
console.log(double(5)); // 10
```

A useful mental model is:

input



function



output

For example:

calculateTax

```
const calculateTax = price => price * 0.15;  
calculateTax(100); // 15
```

## 10 Functions — expressions & when to use which

*Functions take inputs (parameters) and return a result.*

### function expression

```
const add = function(x, y) {  
  return x + y;  
};  
  
console.log(add(5, 3)); // 8
```

### arrow (ES6)

```
const multiply = (x, y) => x * y;  
console.log(multiply(4, 3)); // 12
```

### When to use which:

**Declarations** — best for global, reusable functions or helper methods in vanilla JS or outside React components.

**Expressions** — useful for defining localized, non-hoisted functions in both vanilla JS and React, especially when scoping control is required.

**Arrow functions** — the preferred choice in React for functional components, event handlers, and concise, lexically-scoped callback functions.

# 11 Arrays

*Arrays store collections of values.*

## basics

```
const fruits = ["apple", "banana", "orange"];

console.log(fruits[0]); // apple
console.log(fruits[1]); // banana
```

- Array positions start at 0, not 1.
- Useful operations:

## push / pop

```
const fruits = ["apple", "banana"];

fruits.push("orange");
fruits.pop();

console.log(fruits.length);
```

and the other end of the array — add to start, remove first:

## unshift / shift

```
let animals = ["Cat", "Dog", "Elephant"];

animals.unshift("Tiger"); // Add to start
animals.shift();          // Remove first element
```

## 11 Arrays — map, filter, find

*Three extremely important array methods are map, filter, and find.*

### map() — Transform every item

```
const numbers = [1, 2, 3, 4];  
  
const doubled = numbers.map(number => number * 2);  
  
console.log(doubled);  
// [2, 4, 6, 8]
```

### filter() — Keep certain items

```
const numbers = [1, 2, 3, 4, 5];  
  
const bigNumbers = numbers.filter(number => number > 3);  
  
console.log(bigNumbers);  
// [4, 5]
```

### find() — Find one item

```
const users = [  
  { name: "Alice", age: 22 },  
  { name: "Bob", age: 35 }  
];  
  
const user = users.find(user => user.name === "Bob");  
  
console.log(user);
```

**Mastering array methods will make modern JavaScript considerably easier.**

## 11 Arrays — looping through them

*Looping through arrays with an index.*

### indexed for loop

```
let colors = ["Red", "Green", "Blue"];  
for (let i = 0; i < colors.length; i++) {  
  console.log(colors[i]);  
}
```

**i** counts from 0 while **i < colors.length**, and **colors[i]** reads the item at that position.

## 12 Objects

*Objects group related information.*

### object literal

```
const person = {  
  name: "Alice",  
  age: 25,  
  city: "Montreal"  
};  
  
console.log(person.name);  
console.log(person.age);
```

### Object properties store key-value pairs:

#### bracket access

```
console.log(person["city"]); // Montreal
```

### Properties can change:

#### mutation

```
person.age = 26;  
console.log(person.age); // 26
```

## 12 Objects — methods & real-world data

*Objects can contain functions called methods:*

### methods

```
const person = {  
  name: "Alice",  
  
  greet() {  
    console.log(`Hello, I'm ${this.name}`);  
  }  
};  
  
person.greet();
```

Most real-world JavaScript data is essentially combinations of **objects and arrays**:

### objects + arrays

```
const users = [  
  {  
    id: 1,  
    name: "Alice",  
    skills: ["JavaScript", "HTML"]  
  },  
  {  
    id: 2,  
    name: "Bob",  
    skills: ["Python", "SQL"]  
  }  
];
```

## 12 Objects — looping through them

*Looping through objects with for...in.*

### for...in

```
let car = { brand: "Toyota", model: "Corolla", year: 2020 };

for (let key in car) {
  console.log(key + ": " + car[key]);
}
```

### output

```
// brand: Toyota
// model: Corolla
// year: 2020
```

**for...in** gives you each **key**, and **car[key]** reads the matching value.

Use it when you want to walk every property of one object; use **for...of** when you want the values of an array.

## 13 Loops

*Loops repeat code.*

### for

```
for (let i = 0; i < 5; i++) {  
  console.log(i);  
}
```

### output

```
0  
1  
2  
3  
4
```

Usually cleaner for arrays — **for...of**:

### for...of

```
const fruits = ["apple", "banana", "orange"];  
  
for (const fruit of fruits) {  
  console.log(fruit);  
}
```

Arrays also commonly use **forEach**:

### forEach

```
fruits.forEach(fruit => {  
  console.log(fruit);  
});
```

## 13 Loops — while

*Loops are used to repeat blocks of code.*

### while

```
let count = 0;

while (count < 3) {
  console.log("Count:", count);
  count++;
}
```

### output

```
// Count: 0
// Count: 1
// Count: 2
```

Use **for** when you know how many times to repeat; use **while** when the loop should run until a condition stops being true.

**count++** is what eventually makes the condition false — without it the loop never ends.

# Destructuring

*Modern JavaScript uses destructuring frequently.*

Instead of:

the long way

```
const person = {  
  name: "Alice",  
  age: 25  
};  
  
const name = person.name;  
const age = person.age;
```

you can write:

object destructuring

```
const { name, age } = person;
```

Arrays work similarly:

array destructuring

```
const colors = ["red", "green", "blue"];  
  
const [first, second] = colors;
```

# Spread syntax

*The ... operator can copy or combine values.*

## Arrays:

### arrays

```
const a = [1, 2];
const b = [3, 4];

const numbers = [...a, ...b];

console.log(numbers);
// [1, 2, 3, 4]
```

## Objects:

### objects

```
const user = {
  name: "Alice",
  age: 25
};

const updatedUser = {
  ...user,
  age: 26
};
```

- This pattern appears constantly in modern frontend development.

# The DOM

*Now JavaScript becomes useful for webpages.*

Suppose your HTML contains:

HTML

```
<h1 id="title">Hello</h1>
<button id="button">Click me</button>
```

JavaScript can access those elements:

querySelector

```
const title = document.querySelector("#title");
const button = document.querySelector("#button");
```

And modify them:

modify

```
title.textContent = "JavaScript is working!";
```

Or change CSS:

style

```
title.style.color = "red";
```

*Events let JavaScript react to users.*

## listening

```
const button = document.querySelector("#button");  
  
button.addEventListener("click", () => {  
  console.log("Button clicked!");  
});
```

Combine that with DOM manipulation:

## DOM + events

```
const title = document.querySelector("#title");  
const button = document.querySelector("#button");  
  
button.addEventListener("click", () => {  
  title.textContent = "You clicked the button!";  
});
```

This concept powers a huge amount of frontend development:



## 18 Async JavaScript

*JavaScript often needs to wait for things like servers, databases, timers, or files.*

### A simple example:

#### setTimeout

```
setTimeout(() => {  
  console.log("2 seconds later");  
}, 2000);
```

Modern asynchronous JavaScript revolves around **Promises** and **async/await**.

#### async / await

```
async function getUsers() {  
  const response = await fetch("https://example.com/api/users");  
  const users = await response.json();  
  
  console.log(users);  
}
```

### The key idea:

await = wait for this asynchronous operation to finish

# Async JavaScript — error handling

*Use error handling around operations that can fail:*

## try / catch

```
async function getUsers() {  
  try {  
    const response = await fetch("https://example.com/api/users");  
  
    if (!response.ok) {  
      throw new Error("Request failed");  
    }  
  
    const users = await response.json();  
    console.log(users);  
  } catch (error) {  
    console.error(error);  
  }  
}
```

**try** → attempt the operation

**catch** → handle the failure

**throw** → signal a problem

# Modules

*As programs grow, code gets separated into files.*

**math.js** — exports:

```
math.js  
  
export function add(a, b) {  
  return a + b;  
}
```

**app.js** — imports:

```
app.js  
  
import { add } from "./math.js";  
  
console.log(add(2, 3));
```

- This allows applications to be divided into reusable components.

# Classes

*You'll encounter classes in JavaScript codebases:*

class Person

```
class Person {  
  constructor(name, age) {  
    this.name = name;  
    this.age = age;  
  }  
  
  greet() {  
    return `Hello, I'm ${this.name}`;  
  }  
}  
  
const alice = new Person("Alice", 25);  
console.log(alice.greet());
```

You don't need to master classes immediately. **Functions, objects, and arrays are more important for a beginner.**

# Mini Project: Counter

*Here's your first complete interactive program.*

## HTML:

### HTML

```
<h1 id="count">0</h1>

<button id="minus">-</button>
<button id="plus">+</button>

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

A display (**#count**), two buttons (**#minus** / **#plus**), and the script that makes it work.

## 21 Mini Project: Counter — the JavaScript

*JavaScript:*

app.js

```
let count = 0;

const display = document.querySelector("#count");
const plusButton = document.querySelector("#plus");
const minusButton = document.querySelector("#minus");

function updateDisplay() {
  display.textContent = count;
}

plusButton.addEventListener("click", () => {
  count++;
  updateDisplay();
});

minusButton.addEventListener("click", () => {
  count--;
  updateDisplay();
});
```

State lives in **count** — every click changes state, then **updateDisplay()** re-renders the page.

## 21 Mini Project: Counter — what it teaches

*Despite being tiny, this project contains several core concepts:*

```
graph TD; A[variable] --> B[DOM elements]; B --> C[event listeners]; C --> D[functions]; D --> E[state changes]; E --> F[UI updates];
```

variable  
↓  
DOM elements  
↓  
event listeners  
↓  
functions  
↓  
state changes  
↓  
UI updates

**That same basic pattern scales surprisingly far.**

# Mini Project: Todo Logic

*Try this without looking up the solution first.*

Create an array:

the data

```
const todos = [];
```

Write these functions:

the API

```
addTodo("Learn JavaScript");
```

```
removeTodo(0);
```

```
showTodos();
```

**addTodo(...)** adds a todo · **removeTodo(...)** removes one · **showTodos()** lists them all

## 22 Mini Project: Todo Logic — a solution

*A simple implementation could look like:*

app.js

```
const todos = [];  
  
function addTodo(todo) {  
  todos.push(todo);  
}  
  
function removeTodo(index) {  
  todos.splice(index, 1);  
}  
  
function showTodos() {  
  todos.forEach((todo, index) => {  
    console.log(`${index + 1}. ${todo}`);  
  });  
}  
  
addTodo("Learn variables");  
addTodo("Learn functions");  
addTodo("Build a project");  
  
showTodos();
```

Your next challenge is to connect this logic to HTML buttons and inputs.

# The 80/20 of JavaScript

*Don't try to memorize the entire language. Focus first on:*

Variables  
▼  
Conditions  
▼  
Functions  
▼  
Arrays  
▼  
Objects  
▼  
Array methods  
▼  
DOM  
▼  
Events  
▼  
async/await

**If you can comfortably use those concepts, you have enough JavaScript knowledge to start building real projects.**

A good progression is **Counter → Calculator → Todo App → Quiz App → Weather/API App → then a framework such as React.**

The most important part is writing code yourself rather than only reading tutorials.