ZOLAXTECH

learn. build. secure.

JAVASCRIPT

CODE GUIDE

In every session

Objectives • Live-coding walkthroughs • Try-it-yourself exercises • Homework

www.zolaxtech.com.et

Table of Contents

Course Overview & Schedule

This guide is built for a JavaScript course run over 2 months, meeting 2 days per week, with each session lasting 1 hour 30 minutes 16 sessions in total. Every session in this guide is self-contained: it has clear objectives, a suggested time breakdown, fully explained code you can type along with, and exercises to test yourself before the next class.

How This Guide Is Organized?

- Session 1 is theory only it covers what JavaScript is and gets your computer set up with VS Code, so no coding exercises are attached to it.
- Sessions 2–16 are hands-on: each one teaches a concept, walks through code examples, and ends with Try-It-Yourself exercises plus a short homework challenge.
- A full Answer Key for every exercise in the guide is provided in the final chapter try each exercise yourself first before checking it.
- A Capstone Project in Session 16 ties everything together into one real mini-application.

16-Session Schedule

#	Week / Day	Topic
1	Week 1 · Day 1	Introduction to JavaScript & Setting Up VS Code (Theory)
2	Week 1 · Day 2	Variables & Data Types
3	Week 2 · Day 1	Operators & Expressions
4	Week 2 · Day 2	Control Flow & Loops
5	Week 3 · Day 1	Functions Part 1 (Declarations, Arrow Functions, Parameters)
6	Week 3 · Day 2	Functions Part 2 (Closures & Higher-Order Functions)
7	Week 4 · Day 1	Arrays Part 1 (CRUD & Searching)
8	Week 4 · Day 2	Arrays Part 2 (map, filter, reduce)
9	Week 5 · Day 1	Strings & Template Literals
10	Week 5 · Day 2	Objects, Destructuring & Spread/Rest

#	Week / Day	Topic
11	Week 6 · Day 1	Scope, Hoisting & the this Keyword
12	Week 6 · Day 2	Classes, OOP & Prototypes
13	Week 7 · Day 1	Error Handling & Introduction to Async
14	Week 7 · Day 2	Promises, async/await & Built-ins
15	Week 8 · Day 1	DOM Manipulation & Events
16	Week 8 · Day 2	JSON, Modules, Map/Set, Regex & Capstone Project

Note for Instructors

The week/day labels above are placeholders swap them for your actual calendar dates. Every session is designed to fit comfortably inside 90 minutes for a beginner audience, with a little buffer for questions.

SESSION 1

Introduction to JavaScript & Environment Setup

Week 1 · Day 1 • Duration: 1 hr 30 min

By the end of this session, you will be able to:

- Explain what JavaScript is and where it runs
- Install Visual Studio Code on your computer
- Install and configure the essential VS Code extensions for JavaScript
- Create and run your very first JavaScript file

Segment	Time	Focus
Welcome & what is JS	15 min	History, where JS runs, why it matters
Installing VS Code	20 min	Download, install, tour of the interface
Installing extensions	20 min	Extensions panel, recommended list
First script + Live Server	25 min	Create a file, run it, see it live
Q&A / troubleshooting	10 min	Fix install issues, confirm everyone is ready

Chapter 1 Theory: What Is JavaScript, and How Do We Get Set Up?

JavaScript (JS) is the programming language of the web. It runs inside every modern browser and powers everything interactive on a page button clicks, animations, form validation, live chat, and more. Alongside

HTML (structure) and CSS (style), JavaScript is the third pillar of front-end web development, and thanks to Node.js it also runs on servers, in command-line tools, and even on small devices.

1.1 Where JavaScript Runs

- In the browser attached to a web page to make it interactive.
- On a server using Node.js, to build APIs and backend services.
- Almost everywhere else mobile apps (React Native), desktop apps (Electron), and more.

1.2 Installing Visual Studio Code (VS Code)

VS Code is a free, lightweight, and extremely popular code editor built by Microsoft. It's the editor we'll use throughout this entire course.

1. Go to https://code.visualstudio.com in your browser.
2. Click the big Download button it automatically detects your operating system (Windows, macOS, or Linux).
3. Run the installer. On Windows, make sure to check "Add to PATH" during installation. On macOS, drag the app into your Applications folder.
4. Open VS Code once installation finishes you should see a Welcome tab.

1.3 Installing Node.js (so we can run JS outside the browser)

5. Go to https://nodejs.org.
6. Download the LTS (Long-Term Support) version it's the most stable.
7. Run the installer with the default options.
8. Verify it worked: open a terminal and type node -v it should print a version number like v20.11.0.

1.4 The Best VS Code Extensions for JavaScript

Extensions add superpowers to VS Code. To install any of these: click the Extensions icon in the left sidebar (or press Ctrl+Shift+X / Cmd+Shift+X), search the name, and click Install.

Extension	What it does
Prettier – Code Formatter	Automatically formats your code neatly every time you save
ESLint	Flags likely bugs and style issues as you type
Live Server	Runs your HTML page with auto-refresh on every save
JavaScript (ES6) code snippets	Type short shortcuts that expand into common JS patterns
Error Lens	Shows errors and warnings inline, right next to the broken line
Auto Rename Tag	Renames the matching closing HTML tag automatically
Path Intellisense	Autocompletes file paths as you type import/src paths

Extension	What it does
Better Comments	Color-codes comments (TODO, warnings, highlights) for readability
Material Icon Theme	Adds clear, distinct file-type icons to the file explorer
GitLens	Shows who changed each line and when, if you're using Git

1.5 Recommended VS Code Settings

Open Settings (Ctrl+, or Cmd+,), search for each setting below, and enable it:

- Editor: Format On Save turn this ON so Prettier tidies your code automatically.
- Files: Auto Save set to onFocusChange so you rarely lose work.
- Default Formatter set to Prettier.

1.6 Your First Project Folder

9. Create a folder on your computer called js-course.
10. In VS Code, choose File → Open Folder and select it.
11. Inside VS Code's file explorer, create a new file called index.html.
12. Create a second file called script.js in the same folder.

1.7 Writing and Running Your First Script

Type this into index.html:

```
<!DOCTYPE html>
<html>
<head><title>My First JS Page</title></head>
<body>
  <h1>Hello, ZolaxTech!</h1>
  <script src="script.js" defer></script>
</body>
</html>
```

Then type this into script.js:

```
console.log('Hello from JavaScript!');
document.querySelector('h1').style.color = 'teal';
```

Right-click index.html and choose "Open with Live Server" (this option appears once the Live Server extension is installed). Your browser opens automatically, and the heading should turn teal. Open the browser's DevTools (F12) and check the Console tab you should see your logged message.

1.8 Running JavaScript with Node.js Instead

You can also run plain .js files directly from the terminal, without a browser at all this is how we'll run most exercises in this course.

```
// In VS Code, open the integrated terminal: Ctrl+` (backtick) or View → Terminal
node script.js
// Output: Hello from JavaScript!
// Note: document.querySelector will NOT work here  that line only works in a browser.
```

1.9 Handy VS Code Keyboard Shortcuts

Shortcut (Win/Linux)	Shortcut (Mac)	Action
Ctrl+`	Cmd+`	Toggle the integrated terminal
Ctrl+S	Cmd+S	Save the file
Ctrl+/	Cmd+/	Comment / uncomment a line
Ctrl+P	Cmd+P	Quickly open any file by name
Ctrl+Shift+X	Cmd+Shift+X	Open the Extensions panel
Alt+Down/Up	Option+Down/Up	Move the current line down/up
Ctrl+D	Cmd+D	Select the next occurrence of the current word

✓Session 1 Setup Checklist

☐ VS Code is installed and opens without errors

☐ Node.js is installed (node -v prints a version number)

☐ Prettier, ESLint, Live Server, and Error Lens extensions are installed

☐ Format On Save is turned on

☐ You created js-course/index.html and js-course/script.js and successfully ran both via Live Server AND via node script.js

There are no coding exercises for Session 1 just make sure every box above is checked before Session 2, since every session from here on assumes your environment is ready to go.

SESSION 2

Variables & Data Types

Week 1 · Day 2 • Duration: 1 hr 30 min

By the end of this session, you will be able to:

- Declare variables correctly using let and const
- Identify JavaScript's core data types
- Convert values between types safely

Segment	Time	Focus
Recap Session 1	5 min	Quick check-in on setup
let, const & var	20 min	Declaring variables, scope rules
Data types	25 min	Strings, numbers, booleans, etc.
Type conversion	20 min	Explicit vs implicit coercion
Try-it-yourself + wrap-up	20 min	Live practice, assign homework

2.1 Declaring Variables

Create a file called session2.js inside your js-course folder for today's exercises. Type each example, then run it with node session2.js in the terminal.

```
let age = 25;        // can be reassigned later
age = 26;             // ✅fine

const name = 'Zola';  // cannot be reassigned
// name = 'Sam';       ❌TypeError

const user = { name: 'Zola' };
user.name = 'Sam';     // ✅fine   we're changing a property, not the variable itself
```

Best Practice

Default to const for everything. Only switch to let when you know a value will change. Avoid var completely it ignores block scope and causes confusing bugs.

2.2 The Core Data Types

```
typeof 'hello'      // 'string'
typeof 42           // 'number'
typeof true         // 'boolean'
typeof undefined    // 'undefined'
typeof null         // 'object'  (a famous, harmless JS quirk)
typeof {}           // 'object'
typeof []           // 'object'  (use Array.isArray() to check for arrays specifically)
```

2.3 Type Conversion

```
// Explicit conversion (you do it on purpose)
String(123)     // '123'
Number('42')    // 42
Number('abc')   // NaN
Boolean(0)      // false
```

```
// Implicit coercion (JS does it automatically   be careful!)
'5' + 3   // '53'  (string wins)
'5' - 3   // 2     (number wins for -, *, /)
```

Try It Yourself

Try It Yourself #1

Declare a const called city with your city's name, and a let called temperature set to any number. Log both to the console using node.

Try It Yourself #2

Predict the output of '10' + 5 and '10' - 5 on paper first, then run them and check if you were right.

Try It Yourself #3

Use typeof to check the type of true, null, and an empty array write down each result before running the code.

Session Homework

Write a script that declares 5 variables of 5 different types (string, number, boolean, object, array) and logs the typeof each one. Bring it to the next session.

SESSION 3

Operators & Expressions

Week 2 · Day 1 • Duration: 1 hr 30 min

By the end of this session, you will be able to:

- Use arithmetic, assignment, and comparison operators correctly
- Understand the difference between == and ===
- Use logical operators, the ternary operator, and modern operators like ?? and ?.

Segment	Time	Focus
Recap & homework review	10 min	Check Session 2 homework together
Arithmetic & assignment	20 min	+, -, *, /, %, **, +=, etc.
Comparison & equality	20 min	== vs ===, why it matters
Logical & modern operators	20 min	&&, ||, ??, ?., ternary
Try-it-yourself + wrap-up	10 min	

3.1 Arithmetic & Assignment

```
let x = 10;
x += 5;   // 15
x -= 3;   // 12
x *= 2;   // 24
x /= 4;   // 6
x %= 4;   // 2   (remainder)
x **= 3;  // 8   (exponent)
```

3.2 Comparison: == vs ===

```
5 == '5'    // true    loose equality converts types first
5 === '5'   // false   strict equality checks type too

null == undefined   // true
null === undefined  // false
```

Golden Rule

Always use === and !== in your own code. == causes surprising bugs like '' == 0 being true.

3.3 Logical Operators & Short-Circuiting

```
true && false   // false
true || false   // true
!true           // false

const user = null;
const name = user && user.name;   // stops early → null, no crash
```

3.4 Modern Operators: ?? and ?.

```
let count = 0;
count || 10   // 10   WRONG if 0 is valid!
count ?? 10   // 0    correct, only null/undefined fall back

const user = { profile: null };
user.profile?.email;   // undefined, no crash (instead of a TypeError)
```

3.5 The Ternary Operator

```
const age = 20;
const status = age >= 18 ? 'adult' : 'minor';
console.log(status);   // 'adult'
```

Try It Yourself

Try It Yourself #1

Write an expression that checks if a variable score is greater than or equal to 50, using the ternary operator to log 'Pass' or 'Fail'.

Try It Yourself #2

Predict, then test: what does [] == false return? What about [] === false?

Try It Yourself #3

Use ?? to give a variable username a fallback value of 'Guest' only when username is null or undefined.

Session Homework

Write a small script that takes two numbers and an operator symbol (+, -, *, /) as variables, and uses a series of if/else statements with comparison operators to print the correct result.

SESSION 4

Control Flow & Loops

Week 2 · Day 2 • Duration: 1 hr 30 min

By the end of this session, you will be able to:

- Control your program's flow with if/else and switch
- Repeat code using for, while, and do...while loops
- Iterate arrays and objects with for...of and for...in

Segment	Time	Focus
Recap & homework review	10 min	
if/else & switch	20 min	Branching logic
for, while, do...while	25 min	Classic loops
for...of vs for...in	20 min	Iterating arrays vs objects
Try-it-yourself + wrap-up	15 min	

4.1 if / else if / else

```
const score = 85;
if (score >= 90) {
  console.log('Grade: A');
} else if (score >= 80) {
```

```
  console.log('Grade: B');
} else {
  console.log('Grade: F');
}
// Grade: B
```

4.2 switch

```
const day = 'Mon';
switch (day) {
  case 'Sat':
  case 'Sun':
    console.log('Weekend!');
    break;
  case 'Mon':
    console.log('Back to work');
    break;
  default:
    console.log('Midweek grind');
}
```

4.3 for, while, do...while

```
for (let i = 0; i < 5; i++) { console.log(i); }   // 0 1 2 3 4

let i = 0;
while (i < 3) { console.log(i); i++; }

let j = 0;
do { console.log(j); j++; } while (j < 3);   // runs at least once
```

4.4 for...of vs for...in

```
const fruits = ['apple', 'banana', 'cherry'];
for (const fruit of fruits) { console.log(fruit); }  // the VALUES
for (const index in fruits) { console.log(index); }  // the KEYS ('0','1','2')
```

Rule of Thumb

Use for...of for arrays when you need the values. Use for...in only for plain objects, when you need the keys.

Try It Yourself

Try It Yourself #1

Write a for loop that logs every even number from 1 to 20.

Try It Yourself #2

Write a switch statement that prints the number of days in a given month name (assume no leap years).

Try It Yourself #3

Given const colors = ['red','green','blue'], use for...of to print each color in uppercase.

Session Homework

Write a program using a while loop that prints the multiplication table (1x to 10x) for a number you choose.

SESSION 5

Functions Part 1

Week 3 · Day 1 • Duration: 1 hr 30 min

By the end of this session, you will be able to:

- Write function declarations, expressions, and arrow functions
- Use default and rest parameters
- Understand hoisting differences between function types

Segment	Time	Focus
Recap & homework review	10 min	
Declarations vs expressions	20 min	Hoisting differences
Arrow functions	25 min	Syntax shortcuts
Default & rest parameters	20 min	
Try-it-yourself + wrap-up	15 min	

5.1 Function Declarations vs Expressions

```
// Declaration   hoisted, callable before its definition line
function greet(name) { return `Hello, ${name}!`; }

// Expression   NOT hoisted, must be defined before use
const greet2 = function (name) { return `Hi, ${name}!`; };
```

5.2 Arrow Functions

```
const add = (a, b) => a + b;         // implicit return, no braces needed
```

```
const square = n => n * n;            // single param   parens optional
const sayHi = () => console.log('Hi'); // no params   parens required
```

5.3 Default & Rest Parameters

```
function greet(name = 'Guest') { return `Hello, ${name}!`; }
greet();          // 'Hello, Guest!'

function sum(...numbers) {
  return numbers.reduce((total, n) => total + n, 0);
}
sum(1, 2, 3, 4);  // 10
```

Try It Yourself

Try It Yourself #1

Write an arrow function isEven(n) that returns true if n is even.

Try It Yourself #2

Write a function greetUser(name, greeting = 'Hello') that returns a greeting using the default value when only one argument is passed.

Try It Yourself #3

Write a function total(...prices) that adds up any number of prices passed to it.

Session Homework

Build a function calculateBill(...items) where each item is a price, that returns the total plus 15% tax, rounded to 2 decimal places.

SESSION 6

Functions Part 2: Closures & Higher-Order Functions

Week 3 · Day 2 • Duration: 1 hr 30 min

By the end of this session, you will be able to:

- Explain what a closure is and why it's useful
- Write and use higher-order functions
- Build simple function factories

Segment	Time	Focus
Recap & homework review	10 min	

Segment	Time	Focus
Closures	30 min	The counter example, private data
Higher-order functions	25 min	Functions that take/return functions
Try-it-yourself + wrap-up	25 min	

6.1 Closures

A closure is a function that 'remembers' variables from the scope it was created in, even after that outer function has finished running.

```
function makeCounter() {
  let count = 0;             // private variable
  return function () {
    count++;
    return count;
  };
}
const counter = makeCounter();
console.log(counter());   // 1
console.log(counter());   // 2
```

6.2 Higher-Order Functions

```
// Takes a function as an argument
function repeat(n, action) {
  for (let i = 0; i < n; i++) action(i);
}
repeat(3, i => console.log('Iteration', i));

// Returns a function (a 'function factory')
function multiplyBy(factor) {
  return n => n * factor;
}
const double = multiplyBy(2);
console.log(double(5));   // 10
```

Try It Yourself

Try It Yourself #1

Write a closure-based function makeGreeter(greeting) that returns a function taking a name and combining it with the greeting.

Try It Yourself #2

Write a higher-order function applyDiscount(percentage) that returns a function taking a price and applying that discount.

Try It Yourself #3

Predict what happens if you call makeCounter() twice into two different variables do the counts share state or not? Test it.

Session Homework

Build a private 'bank account' using a closure: a function createAccount(balance) that returns an object with deposit(amount) and getBalance() methods, where balance can't be accessed directly from outside.

SESSION 7

Arrays Part 1: CRUD & Searching

Week 4 · Day 1 • Duration: 1 hr 30 min

By the end of this session, you will be able to:

- Create, read, update, and delete array items
- Search arrays using find, includes, and indexOf
- Destructure arrays

Segment	Time	Focus
Recap & homework review	10 min	
Creating & accessing arrays	15 min	
Adding/removing items	25 min	push, pop, shift, unshift, splice, slice
Searching arrays	20 min	find, includes, indexOf, some, every
Try-it-yourself + wrap-up	20 min	

7.1 Creating & Accessing

```
const fruits = ['apple', 'banana', 'cherry'];
console.log(fruits[0]);        // 'apple'
console.log(fruits.length);    // 3
```

7.2 Adding & Removing Items

Method	What it does	Mutates?
push(item)	Adds to the end	Yes

Method	What it does	Mutates?
pop()	Removes from the end	Yes
unshift(item)	Adds to the beginning	Yes
shift()	Removes from the beginning	Yes
splice(start, count)	Removes/inserts anywhere	Yes
slice(start, end)	Copies a range (no mutation)	No

```
const nums = [1, 2, 3];
nums.push(4);        // [1, 2, 3, 4]
nums.pop();          // nums is now [1, 2, 3]
nums.splice(1, 1);   // removes 1 item at index 1 → [1, 3]
```

7.3 Searching Arrays

```
const nums = [5, 12, 8, 130, 44];
nums.find(n => n > 10);       // 12
nums.includes(8);              // true
nums.some(n => n > 100);       // true
nums.every(n => n > 0);        // true
```

7.4 Array Destructuring

```
const colors = ['red', 'green', 'blue'];
const [first, second] = colors;   // first='red', second='green'
const [, , third] = colors;        // third='blue'
```

Try It Yourself

Try It Yourself #1

Given const tasks = ['wash', 'cook'], push 'clean' to the end and unshift 'wake up' to the beginning. Log the final array.

Try It Yourself #2

Given const nums = [3, 7, 1, 9, 4], use .find() to get the first number greater than 5.

Try It Yourself #3

Destructure const point = [10, 20] into x and y variables and log 'x: 10, y: 20'.

Session Homework

Build a simple to-do list manager using an array: functions addTask(list, task), removeTask(list, index), and a way to print all current tasks.

SESSION 8

Arrays Part 2: map, filter, reduce

Week 4 · Day 2 • Duration: 1 hr 30 min

By the end of this session, you will be able to:

- Transform arrays with map ()
- Filter arrays with filter ()
- Collapse arrays into single values with reduce ()

Segment	Time	Focus
Recap & homework review	10 min	
map()	20 min	Transform every item
filter()	20 min	Keep matching items
reduce()	25 min	Collapse to one value
Try-it-yourself + wrap-up	15 min	

None of these three methods mutate the original array each one returns a brand-new array (or value).

```
const nums = [1, 2, 3, 4, 5];

const doubled = nums.map(n => n * 2);          // [2, 4, 6, 8, 10]
const evens = nums.filter(n => n % 2 === 0);   // [2, 4]
const total = nums.reduce((sum, n) => sum + n, 0);  // 15
```

How reduce works, step by step

Starting value: 0. Step 1: 0+1=1. Step 2: 1+2=3. Step 3: 3+3=6. Step 4: 6+4=10. Step 5: 10+5=15 that final number is what reduce returns.

8.1 Chaining Them Together

```
const prices = [10, 25, 40, 15, 60];

const total = prices
  .filter(p => p > 15)     // keep prices above 15
  .map(p => p * 0.9)        // apply a 10% discount
  .reduce((sum, p) => sum + p, 0);  // add them all up
console.log(total);
```

Try It Yourself

Try It Yourself #1

Given const nums = [4, 9, 15, 22, 30], use .filter() and .map() together to return the even numbers, each doubled.

Try It Yourself #2

Use .reduce() to find the maximum value in const nums = [3, 77, 12, 45, 9] without using Math.max.

Try It Yourself #3

Given an array of objects const people = [{name:'A',age:20},{name:'B',age:15}], use .filter() to keep only adults (age >= 18).

Session Homework

Given an array of orders (objects with item and price), use map/filter/reduce to calculate the total cost of orders over $20.

SESSION 9

Strings & Template Literals

Week 5 · Day 1 • Duration: 1 hr 30 min

By the end of this session, you will be able to:

- Use the most common string methods
- Build dynamic strings with template literals
- Convert between strings and numbers

Segment	Time	Focus
Recap & homework review	10 min	
Template literals	15 min	Embedded expressions, multi-line strings
Common string methods	35 min	slice, split, replace, trim, case methods
Number ↔ string conversion	15 min	
Try-it-yourself + wrap-up	15 min	

9.1 Template Literals

```
const name = 'Zola';
const greeting = `Hello, ${name}! Today is day ${1 + 2}.`;
const multiLine = `Line one
Line two`;
```

9.2 Common String Methods

Method	Example	Result
length	'hello'.length	5
toUpperCase()	'hi'.toUpperCase()	'HI'
trim()	' hi '.trim()	'hi'
includes()	'hello'.includes('ell')	true
slice(a,b)	'hello'.slice(1,3)	'el'
split(sep)	'a,b,c'.split(',')	['a','b','c']
replace()	'hi'.replace('h','H')	'Hi'
padStart(n,c)	'5'.padStart(2,'0')	'05'

9.3 Converting Numbers & Strings

```
Number('42')        // 42
parseInt('42px')    // 42
(42).toString()     // '42'
```

Try It Yourself

Try It Yourself #1

Given const first='Zola', last='Tech', use a template literal to build the string 'Zola Tech says hi!'.

Try It Yourself #2

Given const sentence = 'the quick brown fox', use .split(' ') to turn it into an array of words, then .join('-') back into a dashed string.

Try It Yourself #3

Write code that checks if a string email includes '@' before treating it as valid.

Session Homework

Write a function titleCase(sentence) that capitalizes the first letter of every word in a sentence, using split/map/join.

SESSION 10

Objects, Destructuring & Spread/Rest

Week 5 · Day 2 • Duration: 1 hr 30 min

By the end of this session, you will be able to:

- Create, read, update, and delete object properties

- Destructure objects to pull out values quickly
- Use the spread operator to copy and merge objects

Segment	Time	Focus
Recap & homework review	10 min	
Objects basics	20 min	Create/read/update/delete, dot vs bracket
Object destructuring	20 min	
Spread & rest	25 min	Arrays and objects
Try-it-yourself + wrap-up	15 min	

10.1 Object Basics

```
const user = { name: 'Zola', age: 28 };
console.log(user.name);        // dot notation
console.log(user['age']);      // bracket notation
user.email = 'zola@zolaxtech.com';   // add a property
delete user.age;                      // remove a property
```

10.2 Object Destructuring

```
const user = { name: 'Zola', age: 28, role: 'Dev' };
const { name, role } = user;
const { city = 'Unknown' } = user;   // default if missing
```

10.3 Spread & Rest

```
// Arrays
const a = [1, 2, 3];
const b = [...a, 4, 5];        // [1,2,3,4,5]

// Objects
const base = { a: 1, b: 2 };
const extended = { ...base, c: 3 };   // {a:1, b:2, c:3}

// Rest in a function
function logAll(first, ...rest) { console.log(first, rest); }
```

Try It Yourself

Try It Yourself #1

Given const car = {brand:'Toyota', model:'Corolla', year:2022}, destructure brand and year into variables.

Try It Yourself #2

Use spread to create a new object that copies const settings = {theme:'dark'} and adds fontSize: 14 without modifying the original.

Try It Yourself #3

Write a function describeUser({name, age}) that takes a destructured object parameter directly and returns a sentence.

Session Homework

Given an array of product objects, write a function that returns a new array where every product has an added onSale: true property, using map and object spread without mutating the originals.

SESSION 11

Scope, Hoisting & the this Keyword

Week 6 · Day 1 • Duration: 1 hr 30 min

By the end of this session, you will be able to:

- Explain global, function, and block scope
- Predict hoisting behavior for var, let, and const
- Correctly reason about what this refers to in different situations

Segment	Time	Focus
Recap & homework review	10 min	
Scope	20 min	Global/function/block
Hoisting	20 min	var vs let/const vs functions
The this keyword	30 min	Objects, arrow functions, call/apply/bind
Try-it-yourself + wrap-up	10 min	

11.1 Scope

```
let globalVar = 'everywhere';
function outer() {
  let functionVar = 'only inside outer()';
  if (true) {
    let blockVar = 'only inside these {}';
  }
  // blockVar is NOT accessible here
}
```

11.2 Hoisting

```
console.log(hoistedVar);   // undefined (not an error)   var is hoisted
var hoistedVar = 5;

console.log(hoistedLet);   // ✕ReferenceError   temporal dead zone
let hoistedLet = 5;
```

11.3 this in Objects vs Arrow Functions

```
const user = {
  name: 'Zola',
  greet() { console.log(`Hi, I'm ${this.name}`); }  // this = user
};
user.greet();   // 'Hi, I'm Zola'

const timer = {
  seconds: 0,
  start() {
    setInterval(() => {
      this.seconds++;   // arrow function keeps 'this' = timer ✓
    }, 1000);
  }
};
```

11.4 call, apply, bind

```
function introduce(greeting) { console.log(`${greeting}, I'm ${this.name}`); }
const person = { name: 'Zola' };
introduce.call(person, 'Hello');
const bound = introduce.bind(person);
bound('Hey');
```

Try It Yourself

Try It Yourself #1

Predict the output of logging a let variable declared inside an if-block, accessed outside it. Test to confirm.

Try It Yourself #2

Create an object counter with a count property and an increment() method using a regular function confirm this.count updates correctly.

Try It Yourself #3

Use .bind() to create a greetZola function from a general greet(name) function, permanently bound to name = 'Zola'.

Session Homework

Explain in your own words (a few sentences, in a comment) why arrow functions are usually better than regular functions inside setTimeout/setInterval callbacks tied to an object method.

SESSION 12

Classes, OOP & Prototypes

Week 6 · Day 2 • Duration: 1 hr 30 min

By the end of this session, you will be able to:

- Define classes with constructors and methods
- Use inheritance with extends and super
- Understand how the prototype chain powers classes under the hood

Segment	Time	Focus
Recap & homework review	10 min	
Defining classes	20 min	Constructor, methods
Inheritance	25 min	extends, super, getters/setters
Prototypes	20 min	Object.create, prototype chain
Try-it-yourself + wrap-up	15 min	

12.1 Defining a Class

```
class User {
  constructor(name, age) {
    this.name = name;
    this.age = age;
  }
  greet() { return `Hi, I'm ${this.name}`; }
}
const zola = new User('Zola', 28);
console.log(zola.greet());
```

12.2 Inheritance

```
class Admin extends User {
  constructor(name, age, permissions) {
    super(name, age);
    this.permissions = permissions;
  }
  greet() { return `${super.greet()} (Admin)`; }
```

```
}
```

12.3 Getters, Setters, Static & Private Fields

```
class Circle {
  #radius;                        // private field
  constructor(radius) { this.#radius = radius; }
  get area() { return Math.PI * this.#radius ** 2; }
  set radius(value) { this.#radius = value; }
  static describe() { return 'A circle has a radius and area.'; }
}
```

12.4 Prototypes Under the Hood

```
const animal = { eats: true, walk() { console.log('Animal walks'); } };
const rabbit = Object.create(animal);   // rabbit's prototype = animal
rabbit.jumps = true;
console.log(rabbit.eats);   // true   found via the prototype chain
```

Try It Yourself

Try It Yourself #1

Create a class Book with title, author, and a describe() method that returns a sentence combining both.

Try It Yourself #2

Create a class EBook that extends Book and adds a fileSizeMB property, overriding describe() to mention the file size too.

Try It Yourself #3

Add a private #isRead field and a markAsRead() method to Book, along with a getter isRead that safely exposes it.

Session Homework

Design a small class hierarchy for a game: a base class Character with name and health, and two subclasses Warrior and Mage that each override an attack() method differently.

SESSION 13

Error Handling & Introduction to Async

Week 7 · Day 1 • Duration: 1 hr 30 min

By the end of this session, you will be able to:

- Handle errors gracefully with try/catch/finally
- Throw and create custom errors

- Understand why asynchronous code exists and how callbacks work

Segment	Time	Focus
Recap & homework review	10 min	
try/catch/finally	20 min	
Custom errors	20 min	
Callbacks & the event loop	30 min	Why async code exists
Try-it-yourself + wrap-up	10 min	

13.1 try / catch / finally

```
try {
  const data = JSON.parse('{ invalid json }');
} catch (error) {
  console.error('Something went wrong:', error.message);
} finally {
  console.log('This always runs.');
}
```

13.2 Throwing & Custom Errors

```
function withdraw(balance, amount) {
  if (amount > balance) throw new Error('Insufficient funds');
  return balance - amount;
}

class ValidationError extends Error {
  constructor(message) {
    super(message);
    this.name = 'ValidationError';
  }
}
```

13.3 Callbacks

JavaScript is single-threaded but handles slow operations (timers, network requests) without freezing, using an event loop. The original way to handle 'do this later' code was the callback function.

```
function fetchData(callback) {
  setTimeout(() => { callback('Data loaded!'); }, 1000);
}
fetchData((result) => console.log(result));   // after 1 second
```

Why this matters next session

Deeply nested callbacks create 'callback hell'. Promises (next session) were designed to fix exactly this problem keep this example in mind.

Try It Yourself

Try It Yourself #1

Wrap a JSON.parse() call on invalid JSON in a try/catch and log a friendly error message instead of crashing.

Try It Yourself #2

Create a custom error class NotFoundError and throw it from a function findUser(id) when the id doesn't exist in a small array.

Try It Yourself #3

Write a callback-based function delayedGreet(name, callback) that calls callback with a greeting string after a 2-second setTimeout.

Session Homework

Write a function safeDivide(a, b) that throws a custom error DivisionError if b is 0, and returns the result otherwise. Call it inside a try/catch and log either the result or the error message.

SESSION 14

Promises, async/await & Built-ins

Week 7 · Day 2 • Duration: 1 hr 30 min

By the end of this session, you will be able to:

- Create and consume Promises
- Rewrite Promise chains using async/await
- Run multiple promises together and use key built-in objects (Math, Date)

Segment	Time	Focus
Recap & homework review	10 min	
Promises	25 min	.then/.catch/.finally
async/await	25 min	Cleaner syntax over promises
Promise.all & friends + built-ins	20 min	Math, Date, Number quick tour
Try-it-yourself + wrap-up	10 min	

14.1 Promises

```
function fetchData() {
  return new Promise((resolve, reject) => {
    setTimeout(() => {
      const success = true;
      success ? resolve('Data loaded!') : reject('Error loading data');
    }, 1000);
  });
}
fetchData().then(console.log).catch(console.error).finally(() => console.log('Done'));
```

14.2 async / await

```
async function loadData() {
  try {
    const result = await fetchData();   // pauses here until resolved
    console.log(result);
  } catch (error) {
    console.error(error);
  }
}
loadData();
```

14.3 Running Multiple Promises

Method	Resolves when...	Rejects when...
Promise.all()	Every promise fulfills	Any single promise rejects
Promise.allSettled()	Every promise settles either way	Never rejects
Promise.race()	The first promise settles	If that first one rejects

14.4 Quick Tour: Math & Date

```
Math.round(4.7);   // 5
Math.floor(4.7);   // 4
Math.random();      // 0 to <1
Math.floor(Math.random() * 10) + 1;  // random whole number 1-10

const now = new Date();
now.getFullYear();  // e.g. 2026
now.toISOString();
```

Try It Yourself

> ✍ **Try It Yourself #1**

Rewrite the fetchData().then() example from 14.1 to use async/await with try/catch instead.

Try It Yourself #2

Create two Promises that resolve after different setTimeout delays, and use Promise.all() to wait for both.

Try It Yourself #3

Write a function randomInt(min, max) using Math.random() and Math.floor() that returns a random whole number in the given range.

Session Homework

Simulate a mini API: write a function getUser(id) returning a Promise that resolves with a user object after a delay, or rejects if the id isn't in a small hardcoded list. Consume it with async/await and handle both outcomes.

SESSION 15

DOM Manipulation & Events

Week 8 · Day 1 • Duration: 1 hr 30 min

By the end of this session, you will be able to:

- Select and modify HTML elements with JavaScript
- Create and remove elements dynamically
- Respond to user interaction with event listeners

Segment	Time	Focus
Recap & homework review	10 min	
Selecting & changing elements	25 min	querySelector, textContent, classList
Creating & removing elements	20 min	
Events	25 min	addEventListener, event object, delegation
Try-it-yourself + wrap-up	10 min	

This session runs in the browser, not Node use Live Server on an index.html + script.js pair, since document only exists in a browser.

15.1 Selecting & Changing Elements

```
const title = document.querySelector('#title');
title.textContent = 'New Title';
title.style.color = 'teal';
title.classList.add('highlight');
```

```
title.classList.toggle('active');
```

15.2 Creating & Removing Elements

```
const newItem = document.createElement('li');
newItem.textContent = 'New list item';
document.querySelector('#myList').appendChild(newItem);
newItem.remove();
```

15.3 Listening for Events

```
const button = document.querySelector('#saveBtn');
button.addEventListener('click', (event) => {
  console.log('Clicked!', event.target);
});
```

15.4 Event Delegation

```
document.querySelector('#list').addEventListener('click', (event) => {
  if (event.target.tagName === 'LI') {
    console.log('You clicked:', event.target.textContent);
  }
});
// Works even for <li> items added to the list LATER
```

Try It Yourself

Try It Yourself #1

Build a page with a button that, when clicked, changes a heading's text and background color.

Try It Yourself #2

Add an input box and a button that, on click, creates a new <li> in a list using the input's value.

Try It Yourself #3

Use event delegation on the list from Exercise 2 so clicking any item logs its text and removes it from the page.

Session Homework

Build a simple to-do list app in the browser: an input + 'Add' button that adds items to a list, and clicking any item toggles a 'completed' CSS class on it.

SESSION 16

JSON, Modules, Map/Set, Regex & Capstone Project

Week 8 · Day 2 • Duration: 1 hr 30 min

By the end of this session, you will be able to:

- Convert data between JS objects and JSON
- Split code across files using import/export
- Use Map, Set, and basic regular expressions
- Apply everything learned in a final capstone project

Segment	Time	Focus
Recap & homework review	10 min	
JSON	10 min	stringify/parse
Modules	10 min	import/export
Map, Set & Regex	20 min	Quick practical tour
Capstone project work time	30 min	Build & get help live
Wrap-up & course close	10 min	Q&A, next steps

16.1 JSON

```
const user = { name: 'Zola', age: 28 };
const jsonString = JSON.stringify(user);   // '{"name":"Zola","age":28}'
const parsed = JSON.parse(jsonString);      // back to a real object
```

16.2 Modules

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

// app.js
import { add } from './mathUtils.js';
```

16.3 Map & Set

```
const scores = new Map();
scores.set('Zola', 95);
scores.get('Zola');      // 95

const ids = new Set([1, 2, 2, 3]);   // Set(3) {1, 2, 3}   duplicates removed
const unique = [...new Set([1, 1, 2])];  // [1, 2]
```

16.4 Basic Regex

```
const emailPattern = /^[\w.-]+@[\w.-]+\.\w+$/;
emailPattern.test('user@zolaxtech.com');  // true
'2026-07-31'.split('-');   // ['2026', '07', '31']
```

Capstone Project Personal Task Tracker

Bring together everything from the course into one small but complete browser application. This is the final project for the course work on it during the last part of Session 16 and finish it as homework if needed.

Requirements

- An HTML page with an input field, an 'Add Task' button, and a list to display tasks (DOM + Events Session 15).
- Each task is stored as an object {id, text, done} inside an array (Objects & Arrays Sessions 7–10).
- Clicking a task toggles its done property and adds a 'completed' CSS style (classList Session 15).
- A 'Clear Completed' button removes all done tasks using. filter () (Arrays Session 8).
- Tasks are saved to and loaded from localStorage using JSON.stringify/JSON.parse (JSON 16.1) so the list survives a page refresh.
- Wrap at least one part of the logic (like adding a task) in a small class TaskManager using what you learned about classes (Session 12).
- Bonus: use async/await with a fake setTimeout-based 'save to server' function to simulate saving asynchronously (Session 14).

Course Complete!

Congratulations on finishing all 16 sessions. You've covered the same ground as the ZolaxTech 'Complete JavaScript Cheat Sheet' keep that as your quick-reference companion, and keep building small projects to cement everything you've learned.

Answer Key

Solutions to selected Try-It-Yourself exercises are below, organized by session. Always attempt each exercise yourself first.

Session 2

```
// Ex.1
const city = 'Addis Ababa';
let temperature = 24;
console.log(city, temperature);
```

Session 3

```
// Ex.1
const score = 65;
console.log(score >= 50 ? 'Pass' : 'Fail');
```

Session 4

```
// Ex.1
for (let i = 1; i <= 20; i++) {
  if (i % 2 === 0) console.log(i);
}
```

Session 5

```
// Ex.1
const isEven = n => n % 2 === 0;
```

Session 6

```
// Ex.1
function makeGreeter(greeting) {
  return function (name) { return `${greeting}, ${name}!`; };
}
const sayHello = makeGreeter('Hello');
sayHello('Zola');   // 'Hello, Zola!'
```

Session 7

```
// Ex.1
const tasks = ['wash', 'cook'];
tasks.push('clean');
tasks.unshift('wake up');
```

```
// ['wake up', 'wash', 'cook', 'clean']
```

Session 8

```
// Ex.1
const nums = [4, 9, 15, 22, 30];
const result = nums.filter(n => n % 2 === 0).map(n => n * 2);
// [8, 44, 60]
```

Session 9

```
// Ex.1
const first = 'Zola', last = 'Tech';
console.log(`${first} ${last} says hi!`);
```

Session 10

```
// Ex.1
const car = { brand: 'Toyota', model: 'Corolla', year: 2022 };
const { brand, year } = car;
```

Session 11

```
// Ex.2
const counter = {
  count: 0,
  increment() { this.count++; }
};
counter.increment();
console.log(counter.count);   // 1
```

Session 12

```
// Ex.1
class Book {
  constructor(title, author) {
    this.title = title;
    this.author = author;
  }
  describe() { return `${this.title} by ${this.author}`; }
}
```

Session 13

```
// Ex.1
```

```
try {
  JSON.parse('{ invalid json }');
} catch (e) {
  console.log('Invalid JSON provided.');
}
```

Session 14

```
// Ex.1
async function loadData() {
  try {
    const result = await fetchData();
    console.log(result);
  } catch (e) {
    console.error(e);
  }
}
```

Session 15

```
// Ex.1 (in script.js, run with Live Server)
const btn = document.querySelector('#changeBtn');
const heading = document.querySelector('h1');
btn.addEventListener('click', () => {
  heading.textContent = 'Changed!';
  heading.style.background = 'teal';
});
```

About This Guide

This student code guide was designed to run alongside a live, instructor-led course. It intentionally paces topics for beginners across 16 sessions if your class needs more time on any topic, sessions can be split further without losing the overall structure.

About ZolaxTech

This guide was built by ZolaxTech to make programming education practical and approachable. Visit www.zolaxtech.com for more course guides and hands-on learning resources.