LESSON 51

Interactive Quiz Game

40 minutes | Introduction to Web Frontend | Outcome 1.22 | Outcome 2.6 | Outcome 2.7 | Outcome 2.16 | Outcome 2.20 | Outcome 1.7 | Outcome 3.3

WHAT THIS LESSON DOES

In this lesson, you'll build your own interactive quiz game using HTML, CSS, JavaScript, and jQuery. Create multiple-choice questions, check answers, track scores, and add a timer for an extra challenge, resulting in a fully functional game.

CURRICULUM OUTCOMES

Outcome	What the specification asks for
1.22	read, write, test, and modify computer programs
2.6	construct algorithms using appropriate sequences, selections/conditionals, loops and operators to solve a range of problems, to fulfil a specific requirement
2.7	implement algorithms using a programming language to solve a range of problems
2.16	use data types that are common to procedural high-level languages
2.20	identify and fix/debug warnings and errors in computer code and modify as required
1.7	develop algorithms to implement chosen solutions
3.3	use appropriate programming languages to develop an interactive website that can display information from a database that meets a set of users' needs

WHAT STUDENTS SHOULD BE ABLE TO DO

- Understand the fundamentals of creating an interactive web application using HTML, CSS, and JavaScript.
- Develop skills in structuring a quiz game with dynamic content and user interactions.
- Apply jQuery for efficient DOM manipulation and event handling.
- Implement logic for answer validation, score tracking, and time-based challenges.
- Enhance problem-solving abilities by customising and expanding a web-based project.

BEFORE THE LESSON

- Confirm VS Code is installed and that students can create folders and files in it, since the whole lesson depends on a QuizGame folder with css and js subfolders.
- The jQuery CDN link must load, so test that code.jquery.com is reachable through the school filter before class. If it is blocked, download jquery-3.6.0.min.js and host it locally.
- Build the finished quiz yourself first. The step descriptions are truncated, so have the full working index.html, styles.css and script.js ready to share as reference.

Interactive Quiz Game

HOW THE LESSON RUNS

- Introduction to Interactive Quiz Game
- Setting Up Your Project Folder
- Creating the HTML Structure
- Adding CSS Styling
- Setting Up the JavaScript File and Quiz Data
- Implementing Answer Checking and Score Display
- Adding a Timer
- Challenge: Expand Your Quiz
- Wrap Up

TEACHING MOVES

- 2. Setting Up Your Project Folder. Insist everyone opens the folder itself as the workspace, not a parent folder. Walk the room checking the Explorer pane shows QuizGame at the top, or later relative file paths will silently fail.
- 3. Creating the HTML Structure. Point out the two path references at the top: the stylesheet link to css/styles.css and the jQuery script tag. Stress these folders and files must exist and be named exactly, or nothing renders or runs.
- 4. Adding CSS Styling. Have them open index.html in the browser before styling, then after, so they see CSS as separate from structure. Confirm the css folder was created first, not the file loose in the root.
- 5. Setting Up the JavaScript File and Quiz Data. Read the quizData array aloud as a list of objects, each with question, options and answer. This structure is the whole engine. Make sure answer text matches an option exactly, spelling and case.
- 6. Implementing Answer Checking and Score Display. Slow down on the comparison that checks the selected option against the stored answer. This is where a typo in the data silently marks correct answers wrong. Test all three questions before moving on.
- 7. Adding a Timer. Explain setInterval and clearInterval as start and stop. Warn that forgetting to clear the timer when advancing leaves old countdowns running and skipping questions. Have them test letting the clock run out.
- 8. Challenge: Expand Your Quiz. Give a concrete first target: add two questions by copying an object in the array. Only then move to styling or a restart button, so weaker students get an achievable win.

WHAT TO WATCH FOR

- Students think a page that displays correctly is therefore working. A quiz that renders but always marks answers wrong looks fine on screen. Insist on functional testing: click through every question, deliberately choose a wrong answer, and check the score. Looking right is not the same as being right.
- Files placed in the wrong folder or misnamed, so the browser cannot find the CSS or script and shows an unstyled, dead page with no obvious error. Open the browser developer tools console and network tab together. Show them a 404 for a missing file so they learn to diagnose path problems rather than guessing.
- The answer string in quizData does not exactly match an option string, so correct answers never register. Compare the two side by side. Show that 'Paris' and 'paris ' with a trailing space are different to the computer even though they look the same.

DIFFERENTIATION

- Emerging: Give the finished files as a reference so they copy in stages and see each stage run, rather than being stuck on a blank page. Pair them with a partner and have them narrate what each folder and file is for before typing.
- Developing: Ask them to add one new question unaided, which forces them to read and reuse the quizData object structure. Have them explain aloud where the answer check happens before they trust it.
- Proficient: Push toward the Higher Level treatment: separate the quiz data into its own file, or add correct and incorrect visual feedback and a restart button that resets score and timer cleanly. This build is strong evidence for a web development strand of the Coursework Project. Ask them to note the design decisions they made.