UMI 202 • Web Design and Programming

Week 11

Interaction (JavaScript)

Professional course deck aligned with the source unit

```
HTML + CSS + JavaScript

structure + style + behavior

click → function → DOM update
```

Variables | Selectors | Events | Functions | Arrays | Loops | jQuery

Goal: By the end of this lesson, each JavaScript line should be connected to a visible change in the page.

Why this lesson is structured this way

Textbook coverage + beginner flow + runnable examples

Beginner flow

Start from visible behavior: select one element, change one output, then add variables, conditions, functions, events, arrays, loops, timers, and libraries.

Textbook alignment

The source unit covers variables, data types, operators, control structures, arrays, loops, functions, libraries, and jQuery. This deck keeps that coverage.

Runnable practice

Every major concept is connected to a single-file example that can be tested in the HTML playground.

Learning path: plain JavaScript first, then jQuery as a helper layer.

Learning Unit: Interaction (JavaScript)

The purpose of this unit is to turn static HTML/CSS pages into dynamic interfaces that interact with the user.

What We Will Learn

- Variables and data types
- Operators
- Control structures
- Events and functions
- Arrays and loops
- Timers
- jQuery fundamentals

Keywords

variable, array, loop, function, event, control structure, library, jQuery

Lesson Logic

Each concept is introduced first by meaning, then through a small browser-visible practice example.

Warm-Up Questions

Before the lesson begins, connect JavaScript to everyday web experiences.

Question 1

Which parts of the websites you use create interaction with the user?

Examples: buttons, forms, dropdown menus, counters, notifications, and chat panels.

Question 2

Why are popular JavaScript libraries used?

They can reduce repeated code, provide ready-made functions, support animation, simplify selection, and offer broad community support.

Main idea for today: JavaScript does not only display a page; it lets the page respond.

Lesson Map

The lesson moves from simple to complex ideas, producing a visible result at every stage.

1. Code structure

Concept → small example → mini practice

2. Selectors

Concept → small example → mini practice

3. Variables

Concept → small example → mini practice

4. Operators

Concept → small example → mini practice

5. Events + Functions

Concept → small example → mini practice

6. Control Structures

Concept → small example → mini practice

7. Arrays + Loops

Concept → small example → mini practice

8. Timers

Concept → small example → mini practice

9. jQuery

Concept → small example → mini practice

Learning Outcomes

By the end of this lesson, students should be able to read and explain small scripts.

Understand behavior

Explain how JavaScript turns a static page into an interactive page.

Use the DOM

Select HTML elements and change text, style, HTML content, or form values.

Build small tools

Use variables, conditions, functions, events, arrays, loops, and timers in simple single-file pages.

Success means predicting the visible browser result, not only memorizing syntax.

Detailed Lesson Route

The lesson moves from first script line to mini project.

Part 1–3

Web stack, script flow, selectors, DOM properties, variables, data types, strings, and operators.

Part 4–6

Control structures, functions, parameters, return values, events, and forms.

Part 7–9

Arrays, loops, timers, mini projects, libraries, jQuery, assessment, and practice.

Each section follows: concept → code → visible result → small modification.

How HTML, CSS, and JavaScript Work Together

HTML controls content structure, CSS controls appearance, and JavaScript controls behavior.

HTML

It is the skeleton of the page.
It creates elements such as headings, paragraphs, buttons, forms, and tables.

CSS

It is the visual layer of the page.
It controls color, spacing, alignment, size, responsive layout, and visual animation.

JavaScript

It is the behavior layer of the page.
It handles clicks, input checks, calculations, content updates, and interaction.

HTML answers “what is on the page?”, CSS answers “how does it look?”, and JavaScript answers “what happens and when?”

Why JavaScript Is Needed

Static pages display information; with JavaScript, the page responds to the user and the current situation.

It checks empty form fields.
It can change the view when a button is clicked.
It can create live messages or notification areas.
It can show different content based on user choice.
It adds behaviors such as timers, counters, galleries, and animations.

```
button click
    ↓
JavaScript function
    ↓
select element
    ↓
change style / text / value
```

Key distinction: HTML creates structure, CSS styles it, and JavaScript creates the behavior.

JavaScript Code Structure: Three Key Writing Rules

For readable code, dot notation, camelCase naming, and consistent statement endings are important.

1. Dot Notation

A dot is used to reach properties and methods.

Example: element.style.color

2. camelCase

For multi-word property names, each word after the first begins with a capital letter.

backgroundColor
fontSize
borderRadius

3. Statement Ending

Using semicolons at the end of JavaScript statements is a clean and safe habit.

```
document.getElementById("title").style.fontSize = "52px";
```

Ways to Add JavaScript

JavaScript can be written inside a script tag or connected through an external .js file.

Inline JavaScript

A <script> tag is added inside the HTML file, and JavaScript is written directly inside it.

It is practical for small examples and classroom experiments.

External JavaScript

The code is stored in a separate .js file. The HTML page connects to it with the src attribute.

This is cleaner for larger projects.

```
<script>
  alert("Hello JavaScript");
</script>

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

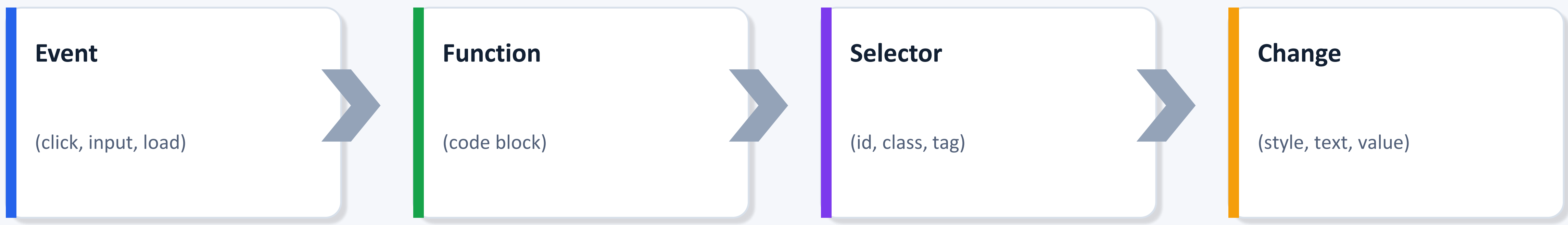
Select → Change → Observe

A simple mental model for every interactive page.

Core formula: select something, change something, and look at the browser result.

General Working Logic

An event happens, a function runs, JavaScript reaches an HTML element with a selector, and then changes it.




```
document.getElementById("message").innerText = "New content";
```

First DOM Change: Read It Slowly

The first example already contains the complete beginner logic.

```
<p id="status">HTML loaded.</p>

<script>
  const statusLine = document.getElementById("status");
  statusLine.textContent = "JavaScript ran after the HTML was loaded.";
</script>
```

Step-by-step reading

- The paragraph already exists in the HTML.
- getElementById("status") finds that paragraph.
- textContent replaces the visible text.
- The browser result proves that JavaScript changed the DOM.

First Runnable Example

The first visible JavaScript action is a simple text change on the page.

```
<p id="status">HTML loaded.</p>

<script>
  document.getElementById("status").innerText =
    "JavaScript ran.";
</script>
```

Key Point

1. HTML first provides visible text.
2. JavaScript finds the paragraph element with a selector.
3. innerText changes the text.
4. The visible result is now different.

The first goal is to connect code with the visible browser result.

Selectors: How JavaScript Reaches HTML Elements

Without a selector, the content, value, or style of an element cannot be changed.

ID Selector

Targets one unique element.

getElementById()

Class Selector

Selects all elements with the same class as a list-like collection.

getElementsByClassName()

Name Selector

Used for form elements with the same name value.

getElementsByName()

Tag Selector

Selects all elements with the same tag name.

getElementsByTagName()

Except for the ID selector, most selectors can return multiple elements, so index numbers such as [0], [1], and [2] are needed.

ID Selector: Working with One Element

The ID selector is especially useful for one title, one paragraph, or one result box.

```
<h1 id="title">Title</h1>
<p id="content">Old content</p>
<input id="fullName" value="">

<script>
  document.getElementById("title").style.fontSize = "52px";
  document.getElementById("content").innerHTML = "<b>Bold text</b>";
  document.getElementById("fullName").value = "Mustafa Ozer";
</script>
```

Explanation

- style changes CSS properties.
- innerHTML can insert HTML code inside an element.
- innerText changes only visible text.
- value reads or writes the value of form controls.

style, innerHTML, innerText, and value

After the selector finds the element, the next step is to decide which property will change.

style

Changes style properties.
Examples: color, size, background.

innerHTML

Can change both HTML code and text.

innerText

Changes only visible text.

value

Reads or writes the values of form elements such as input, select, and textarea.

For safety and simplicity, innerText is more controlled when only text is needed; innerHTML should be used deliberately when HTML output is required.

Choosing the Correct DOM Property

After selection, decide what kind of change is needed.

Text only

Use innerText or textContent when the output is plain visible text.

HTML content

Use innerHTML only when the output must include HTML tags such as <b>, <li>, or <div>.

Form value / style

Use value for form controls and style for CSS properties.

Safe habit: if the output is only text, prefer textContent or innerText.

ClassName Selector: Multiple Elements

Results returned by a class selector behave like a collection; the first element has index number 0.

```
<span class="letters">J</span>
<span class="letters">S</span>
<span class="letters">cript</span>

<script>

document.getElementsByClassName("letters")[0].style.fontSize
= "22px";
  document.getElementsByClassName("letters")[1].style.color
= "white";
</script>
```

Important Logic

Three elements with class="letters" are selected.

[0] → J
[1] → S
[2] → cript

The code affects only the elements at the selected indexes.

Name and TagName Selectors

name is useful for form groups; tagName is useful for groups of the same HTML element.

```
document.getElementsByName("userName")[0]
  .style.backgroundColor = "red";

document.getElementsByTagName("span")[2]
  .style.borderBottomStyle = "solid";
```

Name Selector

It is especially useful for selecting radio buttons, checkboxes, and form fields that share the same name value.

TagName Selector

It can select all div, span, p, or similar tags on the page. It is usually used with an index or a loop.

Modern Selector Bridge

querySelector and querySelectorAll use CSS-style selector syntax.

Modern selectors

- querySelector returns the first matching element.
- querySelectorAll returns all matching elements.
- forEach can update each selected element.
- This prepares students for arrays and loops.

```
const firstCard = document.querySelector(".mini-card");
firstCard.style.background = "#dbeafe";

const allCards = document.querySelectorAll(".mini-card");
allCards.forEach(function (card, index) {
  card.textContent = `${index + 1}. ${card.textContent}`;
});
```

CSS selector knowledge becomes useful directly inside JavaScript.

Variables and Data Types

Variables are named containers that temporarily store data in memory while code is running.

Why It Is Used

- Reuse the same value
- Perform calculations
- Store user input
- Write results to the page
- Keep the program state

Source Unit Approach

In the source unit, variables are introduced with the var keyword. This is enough to understand the basic idea of storing data.

Modern Note

In modern JavaScript, const and let are preferred more often; however, var can still help explain the basic idea of storing data.

Data Types: Number, String, Boolean, Object

In JavaScript, a variable type is usually determined automatically from the assigned value.

```
var number = 52;         // number
var ratio = 1.618;       // decimal number
var city = "Ordu";       // string
var choice = true;       // boolean

var staff = {
  name: "Sinan",
  height: 1.87,
  weight: 92,
  job: "Chef"
};
```

Key Idea

Numbers are not written in quotation marks.
Strings are written in quotation marks.
Boolean values can only be true or false.
Objects hold multiple property-value pairs under one structure.

String Concatenation Mistake

Numbers written inside quotation marks are treated as text, not mathematical numbers.

```
var a = "1";
var b = "98";
var c = "8";

var total = a + b + c;
alert(total);  // 1988
```

Why Not 107?

Because a, b, and c are written inside quotation marks. JavaScript joins them as strings: "1" + "98" + "8" → "1988"

Key question: Is this value a number, or text that only looks like a number?

String vs Number: The 1988 Trap

A value can look like a number but still behave like text.

Why it happens

- Values inside quotation marks are strings.
- The + operator joins strings.
- Input values usually arrive as strings.
- Convert text to number before mathematical calculation.

```
const a = "1";
const b = "98";
const c = "8";

const total = a + b + c;
alert(total); // 1988

const numericTotal = Number(a) + Number(b) + Number(c);
alert(numericTotal); // 107
```

Key question: Is this value a number, or text that only looks like a number?

Variable Naming Rules

Variable names should be readable and follow JavaScript rules.

Names should be unique.
They can start with a letter, underscore (_), or dollar sign ($).
They cannot start with a number.
Special characters other than underscore and dollar sign cannot be used.
They are case-sensitive: Score and score are different.
JavaScript keywords such as var, if, and else cannot be used as variable names.

```
var studentName = "Aylin";
var _score = 85;
var $state = true;

// Incorrect examples:
var 2score = 90;
var class = "A";
```

Modern Variable Habits

The source unit uses var; modern JavaScript usually uses const and let.

const

Use const when the variable should not be reassigned. Most values can start as const.

let

Use let when the value will change, such as a counter or a timer value.

var

Older examples and textbooks often use var. Understand it, but prefer const/let in new code.

For beginner clarity: use const by default, switch to let only when the value changes.

Operators: Calculation, Assignment, Comparison, Logic

Operators are the calculation and decision-making language of the program.

Arithmetic

+ - * / %
++ --

Addition, subtraction, multiplication, division, modulo, increment, decrement

Assignment

= += -= *= /= **=

Assign a value to a variable or update an existing value

Comparison

< > == <= >= !=

The result is true or false.

Logical

&& || !

Used to evaluate multiple conditions together.

Arithmetic and Assignment Operators

They are often used in counters, score calculations, and averages.

```
var a = 8;
var b = 2;

var total = a + b;  // 10
var fark = a - b;    // 6
var carpim = a * b;  // 16
var bolum = a / b;   // 4
var kalan = a % b;   // 0

a += 5;           // a = a + 5
b--;              // b = b - 1
```

Explanation

Assignment operators may look confusing at first.

Read A += 5 as “add 5 to the current value of A.”

Comparison and Logical Operators

Comparison operators are the foundation of decision structures; logical operators build multiple conditions.

```
var score = 78;

score >= 60      // true
score < 50       // false
score != 100     // true

var user = "Ali";
var password = "12345";

if (user == "Ali" && password == "12345") {
  // login approved
}
```

Short Meanings

&& → and
Both conditions must be true.

|| → or
At least one condition can be true.

! → not
It reverses the result.

Arithmetic Plus Comparison in One Output

A calculation can immediately become a true/false decision.

```
const quiz = 72;
const project = 88;
const average = (quiz + project) / 2;
const passed = average >= 70;

document.getElementById("scoreGrid").innerHTML = `
  <div>Quiz: ${quiz}</div>
  <div>Project: ${project}</div>
  <div>Average: ${average}</div>
  <div>Passed? ${passed}</div>
`;
```

Calculation flow

- First calculate the average.
- Then compare it with the pass threshold.
- The comparison result is true or false.
- Finally show all values in the browser.

Events: How the Page Responds to the User

Events are interaction points created by the user or the browser.

Everyday Examples

- Clicking a button
- Typing into an input
- Loading the page
- Moving the mouse over an element
- Pressing a key
- Leaving a selected field

Code Logic

When an event happens, code can run directly or a function can be called.

Core Sentence

“When the user does this, run this function.”

This sentence is the basis of event logic.

Common JavaScript Events

Understanding when each event is used is more important than memorizing event names.

onLoad

When an element or page loads

onClick

When the user clicks

onDblClick

On double click

onFocus

When an element receives focus

onBlur

When focus leaves the element

onMouseOver

When the mouse is over the element

onMouseOut

When the mouse leaves the element

onKeyDown

When a keyboard key is pressed

Event Names as Triggers

An event is the trigger; the function creates the result.

Mouse events

click, dblclick, mouseover, mouseout, mousedown, mouseup

Keyboard and input

keydown, input, focus, blur, submit

Page lifecycle

load and DOM-ready patterns run code when the page is available.

Do not memorize only the name; connect each event to a user action or browser state.

Event + Function Relationship

If an event action is very short, it can be written in the tag; if the code grows, it should be placed inside a function.

```
<button onclick="darkMode()">Dark Mode</button>
<button onclick="lightMode()">Light Mode</button>

<script>
function darkMode() {
  document.getElementsByTagName("body")[0].style.color = "white";
  document.getElementsByTagName("body")[0].style.backgroundColor =
"black";
}
</script>
```

Key Idea

When functions are used with events, page behavior becomes organized.

onclick only triggers the action; the main work happens inside the function.

Inline Events and addEventListener

Both styles connect user action to code; they organize behavior differently.

Inline event

The event is written directly in the HTML tag. It is easy to see in early textbook examples.

```
<button onclick="darkMode()">Dark</button>
```

addEventListener

The event is connected inside JavaScript. It keeps HTML cleaner and is common in modern projects.

```
button.addEventListener("click", function(){ ... });
```

Both mean the same idea: when this happens, run this code.

What Is a Function?

A function is a named, reusable block of code created to perform a specific task.

Why It Is Used

- Reduces repeated code
- Groups code
- Improves readability
- Works with events
- Allows the same action to be called again

```
function functionName()
{
  // code to run
}

functionName();
```

Basic Parts

function keyword
Function name
Parentheses ()
Curly braces {}
Code that runs inside

Functions Without Parameters

If the action does not need extra information from outside, a function without parameters is enough.

```
<input type="button" onclick="resizeImage()">
<img src="sunset.jpg" id="image">

<script>
function resizeImage() {
  document.getElementById("image").style.width = "150px";
  document.getElementById("image").style.height = "150px";
}
</script>
```

Explanation

When the button is clicked, the same action always happens: the element with id="image" becomes 150px wide and 150px high.

Because no external value is needed, the function is called with empty parentheses.

Practice: Gallery with Functions Without Parameters

In the gallery example, each button brings a different image to the front; zIndex changes the layer order.

```
function showImage1() {
  document.getElementById("image1").style.zIndex = 2;
  document.getElementById("image2").style.zIndex = 1;
  document.getElementById("image3").style.zIndex = 1;
}

function showImage2() {
  document.getElementById("image1").style.zIndex = 1;
  document.getElementById("image2").style.zIndex = 2;
  document.getElementById("image3").style.zIndex = 1;
}
```

Explanation

The images are placed on top of one another.

The image with the larger zIndex value appears on top.

The buttons do not change the image source; they only change the layer order.

Functions with Parameters

A parameter is a value sent into a function from outside; it determines which element or value the function works with.

```
function newSize(a) {
  document.getElementById(a).style.width =
"150px";
  document.getElementById(a).style.height =
"150px";
}

newSize("image");
```

Why It Is More Flexible

The same function can be called with different id values.

A parameter makes the function flexible instead of fixed.

A parameter is the information sent to a function.

Practice: Changing Border Radius with a Parameter

Radio buttons send different values to the same function; the function stays the same, but the result changes.

```
<input type="radio" onclick="roundCorner('0px')"> 0 pixels
<input type="radio" onclick="roundCorner('10px')"> 10 pixels
<input type="radio" onclick="roundCorner('25px')"> 25 pixels
<input type="radio" onclick="roundCorner('50px')"> 50 pixels

<script>
function roundCorner(radiusValue) {
  document.getElementById("box").style.borderRadius =
radiusValue;
}
</script>
```

Main Point

Instead of writing four separate functions, one function is used.

The parameter sent to the function creates the difference.

The this Keyword

this is used to send the element where the event happened into the function.

```
<input type="text" value="0" onkeypress="squareNumber(this)">
<p id="result">0</p>

<script>
function squareNumber(a) {
  var number = a.value;
  var result = number * number;
  document.getElementById("result").innerHTML = result;
}
</script>
```

Key Idea

this represents the input element where the event occurred.

a.value reads the current value of that input.

The result is written into another element.

The this Parameter with Number Conversion

this sends the current element into the function.

```
<input type="text" value="0" oninput="squareValue(this)">
<p id="result">0</p>

<script>
function squareValue(inputElement) {
  const number = Number(inputElement.value);
  const result = number * number;
  document.getElementById("result").textContent = result;
}
</script>
```

Why this version is clearer

- this means “this exact input element”.
- inputElement.value reads what the user typed.
- Number(...) converts text input into a number.
- The output updates the paragraph.

The return Command

return sends the result produced by a function back outside the function.

```
function megaByte(a) {
  var result = a * 1024;
  return result;
}

var x = megaByte(100);
alert(x); // 102400

function combineName(ad, soyad) {
  var fullName = ad + " " + soyad;
  return fullName;
}
```

Why It Matters

A function does not only have to write to the screen.

It can first calculate a result and then return it so the result can be used elsewhere.

Return Values in a Calculator

A function can calculate first and return a result later.

```
function calculateTotal(price, quantity, taxRate) {
  const subtotal = price * quantity;
  const tax = subtotal * taxRate;
  return subtotal + tax;
}

const total = calculateTotal(120, 2, 0.10);
document.getElementById("summary").textContent = total;
```

Clean calculation pattern

- The function does not directly update the page.
- It returns a calculated value.
- Another line decides where to show that value.
- This separation makes larger code easier to manage.

Practice: Messaging Interface

The user writes a message into the input, presses the button, and JavaScript adds a new message box into the HTML.

```
function sendMessage() {
  var message = document.getElementById("message").value;
  var d = new Date();
  var hour = d.getHours();
  var minute = d.getMinutes();

  document.getElementById("messageBoxes").innerHTML +=
    "<div class='sentMessage'>" + message +
    "<span class='time'>" + hour + ":" + minute + "</span></div>";

  document.getElementById("message").value = "";
}
```

Concept Links

This example combines several concepts at once:
- reading value
- Date object
- adding with innerHTML
- appending to existing content with +=
- clearing the input

Mini Form Validation Flow

Read, clean, check, and then show feedback.

Validation steps

- Read the input values.
- Use trim() to remove extra spaces.
- Check empty values first.
- Check the email pattern simply.
- Write clear feedback to the page.

```
const studentName = document.getElementById("studentName").value.trim();
const studentEmail = document.getElementById("studentEmail").value.trim();

if (!studentName || !studentEmail) {
  feedback.textContent = "Please fill in both fields.";
} else if (!studentEmail.includes("@")) {
  feedback.textContent = "The email address must contain @.";
} else {
  feedback.textContent = `Ready to submit: ${studentName} / ${studentEmail}`;
}
```

This combines forms, strings, and if/else validation flow.

Message Interface with Template Literals

A user action can create a new visible interface block.

```
function sendMessage() {
  const message = document.getElementById("message").value.trim();
  if (!message) return;

  const date = new Date();
  const time = `${date.getHours()}:${date.getMinutes()}`;

  document.getElementById("messageBoxes").innerHTML += `
    <div class="sent-message">
      <span>${message}</span>
      <small>${time}</small>
    </div>`;

  document.getElementById("message").value = "";
}
```

Concept links

- Read the input value.
- Stop if the message is empty.
- Create a time value with Date.
- Append new HTML to the message list.
- Clear the input after sending.

Control Structures: Decision Points in a Program

Control structures decide which code runs depending on a condition.

if

Runs if the condition is true.
Skipped if the condition is false.

if-else

If true, one path runs; if false, another path runs.

else if

Checks multiple ranges or situations.

switch

Chooses among fixed known values.

Without control structures, the page responds the same way to every user; with control structures, it reacts based on the situation.

The if Structure

if runs code only when the condition is true.

```
if (condition) {
  // code to run if the condition is true
}

function hideAd() {
  var d = new Date();
  var minute = d.getMinutes();

  if (minute >= 15) {
    document.getElementById("ad").style.display = "none";
  }
}
```

Reading the Condition

If the minute value is 15 or greater, the ad is hidden.

If the condition is false, the if block does not run.

Multiple Conditions

&& and || allow multiple conditions to be used together.

```
if (floorNo == 8 && apartmentNo == 32) {
  // runs if both values are correct
}

if (month == "June" || month == "July" || month ==
"August") {
  // runs if it is one of these three months
}
```

Short Logic

&&: all conditions must be true.

||: at least one condition must be true.

Common examples include username + password, summer months, and valid ranges.

The if-else Structure

else represents the alternative path that runs when the condition is false.

```
function changeMode() {
  if (document.getElementById("yes").checked == true) {
    document.getElementsByTagName("body")[0].style.backgroundColor = "black";
    document.getElementsByTagName("body")[0].style.color = "white";
  } else {
    document.getElementsByTagName("body")[0].style.backgroundColor = "white";
    document.getElementsByTagName("body")[0].style.color = "black";
  }
}
```

Check the Branch

If the radio button is checked, the dark style runs.

If it is not checked, the light style runs.

This example shows how a decision structure changes the visible result.

The else if Structure

An else-if chain is used when there are multiple ranges or possible results.

```
function checkResult() {
  var netScore = document.getElementById("netScore").value;

  if (netScore < 0) {
    screen.innerText = "Invalid number!";
  } else if (netScore < 10) {
    screen.innerText = "Below average";
  } else if (netScore < 25) {
    screen.innerText = "Near average";
  } else if (netScore <= 40) {
    screen.innerText = "Above average";
  } else {
    screen.innerText = "Invalid number!";
  }
}
```

Important Point

Conditions are read from top to bottom.

When the first true condition is found, later conditions are not checked.

A wrong order can produce a wrong result.

Decision Making with Visual Classes

Conditions can change both text and visual state.

```
const score = 84;
const badge = document.getElementById("gradeBadge");

if (score >= 85) {
  badge.textContent = "Excellent";
  badge.className = "badge success";
} else if (score >= 60) {
  badge.textContent = "Needs revision";
  badge.className = "badge warn";
} else {
  badge.textContent = "Redo the task";
  badge.className = "badge danger";
}
```

UI decision flow

- The condition chooses the message.
- className changes the visual badge state.
- This is closer to real interface behavior.
- Try scores below 60, between 60 and 84, and 85 or above.

The switch-case Structure

Used to check which fixed value a variable matches.

```
var date = new Date();
var day = date.getDay();

switch (day) {
  case 0:
    alert("Sunday");
    break;
  case 1:
    alert("Monday");
    break;
  default:
    alert("Unknown day");
}
```

Key Details

switch reads one value.
case runs the matching block.
break exits the switch.
default runs when there is no match.

Matching data types matter.

switch for Visual Modes

A fixed mode value can choose one visual state.

```
const mode = "focus";
const box = document.getElementById("themeBox");

switch (mode) {
  case "calm":
    box.style.background = "#0f766e";
    break;
  case "focus":
    box.style.background = "#1d4ed8";
    break;
  case "review":
    box.style.background = "#9333ea";
    break;
  default:
    box.style.background = "#475569";
}
```

Mode logic

- switch is useful for fixed labels.
- case checks exact matches.
- break prevents fall-through.
- default handles unknown values.

Arrays: Holding Many Values with One Name

An array stores multiple related values under one variable name in an organized order.

```
// Without an array
var city0 = "Izmir";
var city1 = "Istanbul";
var city2 = "Ankara";

// With an array
var cities = ["Izmir", "Istanbul", "Ankara", "Bursa",
"Ordu"];
```

Why It Is Better

It keeps similar values in one structure.

Values are read by index number.

It becomes very powerful when used with loops.

The first element has index number 0. This is a basic habit for JavaScript arrays.

Defining, Assigning, and Reading Arrays

Arrays can be defined empty or with values assigned directly.

```
var cities = [];
var cars = [];

var cities = ["Izmir", "Istanbul", "Ankara"];

cities[0] = "Izmir";
cities[1] = "Istanbul";

document.getElementById("screen").innerHTML = cities[0];
alert("You are connecting to the web page from " + cities[1]);
```

Key Idea

cities[0] gives the first value.
cities[1] gives the second value.

The same indexing logic also applies to lists returned by class, name, and tag selectors.

Array Methods

Array methods make it easier to add, remove, sort, and search list items.

push()

Adds a new item to the end of the array.

length

Returns the number of items.

sort()

Sorts from A to Z.

reverse()

Reverses the order.

pop()

Removes the last item.

indexOf()

Returns the index number of the searched value.

```
products.push("Pencil");
products.length;
products.sort();
products.reverse();
products.pop();
products.indexOf("Pencil");
```

Practice: Storing Course Averages with Arrays

Grades are stored in separate arrays, averages are calculated, and the overall average is found.

```
var math = [60, 90, 100];
var date = [80, 80];
var physics = [100, 90, 75];
var averages = [];

var ort1 = (math[0] + math[1] + math[2]) / 3;
var ort2 = (history[0] + history[1]) / 2;
var ort3 = (physics[0] + physics[1] + physics[2]) / 3;

averages.push(ort1);
averages.push(ort2);
averages.push(ort3);
```

Improvement Idea

This example can later be improved with loops.

For now, each index is written manually.
Once loops are learned, repeated calculations become shorter.

Practice: Digital Dictionary

The index number of the English word is found; the Turkish equivalent at the same index is printed to the page.

```
var english = ["apple", "bottle", "computer", "book"];
var turkish = ["elma", "şişe", "bilgisayar", "kitap"];

function translateWord() {
  var searched =
document.getElementById("searchWord").value;
  var indexNo = english.indexOf(searched);

  if (indexNo >= 0) {
    document.getElementById("turkish").innerText =
turkish[indexNo];
  }
}
```

Main Logic

The two arrays work in parallel.

english[0] = apple
turkish[0] = elma

indexOf finds the index of the searched word.

Digital Dictionary with indexOf

Two arrays can work together when their indexes match.

```
const english = ["apple", "bottle", "computer", "book", "mouse"];
const turkish = ["elma", "şişe", "bilgisayar", "kitap", "fare"];

function translateWord() {
  const search = document.getElementById("searchWord").value.trim();
  const index = english.indexOf(search);

  if (index >= 0) {
    document.getElementById("result").textContent = turkish[index];
  } else {
    document.getElementById("result").textContent = "Not found";
  }
}
```

Parallel arrays

- english[0] matches turkish[0].
- indexOf finds the English word position.
- The same index gives the Turkish translation.
- This is a clear bridge from arrays to searching.

Loops: Managing Repeated Code

Loops are used when the same operation must be repeated many times.

Why Loops?

Instead of writing 500 if statements for 500 products, one if statement can be placed inside a loop.

For

for is usually used when the number of repetitions is known.

While / Do-While

These loops run while a condition is true. do-while runs at least once.

Arrays and loops together are the basis for repeated cards, lists, and table rows on a web page.

Counters

Counters increase or decrease a variable value step by step.

```
<h1 id="counter">0</h1>
<input type="button" onclick="arttir()" value="Increase Number">
<input type="button" onclick="azalt()" value="Decrease Number">

<script>
function arttir() {
  document.getElementById("counter").innerText++;
}
function azalt() {
  document.getElementById("counter").innerText--;
}
</script>
```

Important Point

++ increases the value by 1.
-- decreases the value by 1.

This example combines DOM selection and assignment logic.

Counter State with let

A changing variable remembers the current UI state.

```
let count = 0;
const countValue = document.getElementById("countValue");

function increase() {
  count += 1;
  countValue.textContent = count;
}

function decrease() {
  count -= 1;
  countValue.textContent = count;
}
```

State idea

- count starts at 0.
- Each action changes the stored value.
- The page displays the stored value.
- This introduces the idea of interface state.

The for Loop

A for loop is generally used when the number of repetitions is known.

```
for (var i = 0; i < 8; i++) {
  // i = 0, 1, 2, 3, 4, 5, 6, 7
}

function changeColors() {
  var colors = ["red", "green", "blue", "yellow", "pink", "orange", "gray", "white"];

  for (var i = 0; i < 8; i++) {
    var randomIndex = Math.floor(Math.random() * 8);
    document.getElementsByTagName("div")[i].style.backgroundColor = colors[randomIndex];
  }
}
```

Explanation

The loop visits 8 boxes one by one.
A random color is selected for each box.
The TagName selector result is used with an index.

for Loop with Random Colors

Loops remove repeated manual code.

```
function changeColors() {
  const colors = ["red", "green", "blue", "yellow", "pink", "orange", "gray", "white"];
  const boxes = document.getElementsByClassName("box");

  for (let i = 0; i < boxes.length; i += 1) {
    const randomIndex = Math.floor(Math.random() * colors.length);
    boxes[i].style.backgroundColor = colors[randomIndex];
  }
}
```

Random color logic

- The loop visits every box.
- Math.random creates a random decimal.
- Math.floor turns it into an integer index.
- The index selects one color from the array.

Nested for Loop: Chessboard

Nested loops are used when row and column logic is needed.

```
function chessboard() {
  var i, j;

  for (i = 0; i < 8; i++) {
    for (j = 0; j < 8; j++) {
      if ((j + i) % 2 == 0) {
        boardFrame.innerHTML += "<div class='whiteSquare'></div>";
      } else {
        boardFrame.innerHTML += "<div class='blackSquare'></div>";
      }
    }
  }
}
```

Mathematical Logic

i represents the row, and j represents the column.

(j + i) % 2 determines the square color.

This example combines loop + condition + HTML generation.

Nested for Loop: Chessboard Logic

A grid needs rows and columns, so two loops work together.

```
function createChessboard() {
  let html = "";

  for (let row = 0; row < 8; row += 1) {
    for (let col = 0; col < 8; col += 1) {
      if ((row + col) % 2 === 0) {
        html += `<div class="white-square"></div>`;
      } else {
        html += `<div class="black-square"></div>`;
      }
    }
  }

  document.getElementById("board").innerHTML = html;
}
```

Grid reasoning

- The outer loop controls rows.
- The inner loop controls columns.
- (row + col) % 2 alternates the color.
- The final HTML is written once to the board.

while and do-while Loops

while checks the condition first; do-while runs the code at least once and checks the condition at the end.

```
var number1 = 0;
while (number1 <= 10) {
  document.getElementById("screen1").innerHTML += number1;
  number1++;
}

var number2 = 10;
do {
  document.getElementById("screen2").innerHTML += number2;
  number2--;
} while (number2 >= 0);
```

Difference

while: condition first, then execution.

do-while: execution first, then condition.

Therefore, do-while runs at least once.

Loop Safety

A loop must move toward its stopping condition.

while

Checks the condition before running. If the condition is false at the start, it may not run at all.

do-while

Runs once first, then checks the condition. Useful when one execution is required.

Infinite loop risk

If the counter never changes or the condition never becomes false, the browser may freeze.

Always check: initial value, condition, and counter update.

Timers

Timers allow code to run after a delay or repeatedly at specific intervals.

setTimeout

Runs code once after the set time.

Example: hide an ad after 5 seconds.

setInterval

Runs repeatedly at a set interval.

clearInterval is used to stop it.

```
setTimeout(function(){ ad.style.display = "none"; }, 5000);

var timer = setInterval(function(){ counter--; }, 1000);
clearInterval(timer);
```

Timer Countdown Core

Some interactions depend on time, not direct user input.

```
let seconds = 5;
const timerId = setInterval(function () {
  seconds -= 1;
  timerValue.textContent = seconds;

  if (seconds === 0) {
    clearInterval(timerId);
    timerBox.classList.add("done");
  }
}, 1000);
```

Timer behavior

- setInterval repeats the function.
- 1000 means one second.
- clearInterval stops the repeating timer.
- classList.add changes the visual state at the end.

Practice: Melody from Random Notes

An array, a random number, and setInterval are used together to add a new note every 2 seconds.

```
var metronom;

function startMelody() {
  var notes = ["do", "re", "mi", "fa", "sol", "la", "si"];

  metronom = setInterval(function() {
    var randomIndex = Math.floor(Math.random() * 7);
    document.getElementById("staff").innerText += notes[randomIndex] + "_";
  }, 2000);
}

function stopMelody() {
  clearInterval(metronom);
}
```

Concepts

- Array: notes
- Random number: Math.random()
- Integer conversion: Math.floor()
- Repetition: setInterval
- Stop: clearInterval

Timed UI State

A timer can update both content and CSS classes.

Content update

The timer changes visible numbers or messages at regular intervals.

Example: countdown value, status note, progress text.

State update

When time is complete, classList.add can switch the element into a success or finished visual state.

Timer examples should stay small: the key concept is time-based change.

Popular JavaScript Libraries

A library is a collection of ready-made functions under one structure.

Purpose

To do more with less code.

Selection, animation, data handling, and interface development become easier.

Examples

jQuery
React
Angular
Vue

Different ecosystems serve different needs.

Note

Using a library is not a magic solution without core JavaScript; the basic logic is still the same.

Plain JavaScript First, Then Libraries

The correct learning order for zero-knowledge students.

Foundation first

Students must understand selection, events, variables, and DOM updates in plain JavaScript.

Then shortcuts

jQuery becomes easier because it is a helper layer over the same ideas.

Assessment benefit

Students who understand plain JavaScript can read, debug, and explain code more confidently.

A library is a shortcut after you understand the road.

What Is jQuery?

jQuery is a popular library that allows JavaScript commands to be written more shortly and practically.

```
<script
src="https://code.jquery.com/jquery-
3.6.0.js"></script>
```

It can be used together with standard JavaScript.
Selectors are shorter.
Event functions are easier to define.
Style, content, and value changes become easier.
It supports effects and animation.
It is open source and has broad documentation.

Connection

To use jQuery, the relevant script link can be added inside the HTML head or near the end of the body.

A Library Is Not Magic

jQuery simplifies patterns that already exist in JavaScript.

Plain JavaScript idea

Select an element, attach an event, change style/content/value, and observe the result.

jQuery helper idea

Use shorter syntax for selecting, events, effects, content insertion, and removal.

The underlying browser behavior is still selection, event, and DOM update.

Using jQuery Selectors

jQuery selectors are similar to CSS selector syntax and shorten standard JavaScript selectors.

```
// Standard JavaScript
document.getElementById("content")
document.getElementsByClassName("content")
document.getElementsByTagName("p")

// jQuery
$("#content")
$(".content")
$("p")
```

Commentary

In jQuery, # selects an id, . selects a class, and a plain tag name selects a tag.

The same selector can be used to change style, attach events, or update content.

The ready Function

ready makes the jQuery code inside it run when the page is fully ready.

```
$(document).ready(function() {
  // jQuery code that runs when the
page is ready
});
```

Why It Is Used

If JavaScript runs before HTML elements exist, a selector may not find the element.

ready reduces this problem.

Core sentence: “When the page is ready, do these actions.”

jQuery Effect Functions

jQuery effect functions can hide, show, fade, slide, and animate elements.

hide / show

Hides or shows an element over a set duration. When hidden, another element may take its place.

fadeOut / fadeIn

Changes visibility gradually.

slideUp / slideDown

Hides or shows the element by sliding it up or down.

animate

Changes selected CSS properties over time.

```
$(".image").hide(3000);
$("div").fadeOut(1000);
$("#content").slideUp(500);
$("#box").animate({ left: "250px", width: "150px" }, 1000);
```

jQuery Effects in Context

Effects are useful only when they support the interface purpose.

hide / show

Hide or show an element, optionally over a duration.

fade / slide

Gradually change visibility or slide elements up and down.

animate

Change selected CSS properties over time.

Use effects to guide attention; avoid unnecessary motion.

jQuery Event Functions

With jQuery, the event and the code to run can be written in the same block.

```
$(document).ready(function() {
  $("#button1").click(function() {
    $("#screen").text("Button clicked.");
  });
});
```

Comparison

In standard HTML event handling, onclick is written in the tag.

In jQuery, the event is connected to the selected element inside the script.

This structure is cleaner in larger projects.

Reading and Writing Values with jQuery

jQuery performs standard innerHTML, innerText, and value operations with shorter functions.

html()

Reads or writes HTML content.

Standard equivalent: innerHTML

text()

Reads or writes text only.

Standard equivalent: innerText

val()

Reads or writes the value of a form element.

Standard equivalent: value

```
var a = $("#description").val();
$("#content1").html("<h3>" + a + "</h3>");
$("#content2").text(a);
```

Adding and Removing Content with jQuery

With jQuery, content can be added at the beginning, end, before, or after selected elements.

append()

Adds to the end of the selected element

prepend()

Adds to the beginning of the selected element

before()

Adds before the selected element

after()

Adds after the selected element

remove()

Removes the element and its children

empty()

Empties the element content

```
$("p").append("New content");
$("#box4").empty();
$("#box4").remove();
$("span").css({ backgroundColor: "tomato", color: "white" });
```

Practice: Box Operations with jQuery

The box example shows jQuery selection, style changes, content insertion, and element removal together.

```
$(document).ready(function() {
  $("#action1").click(function() {
    $("span").css({ "backgroundColor": "tomato", "color": "white" });
  });

  $("#action2").click(function() {
    $("#box2").before("<span>New<br>Box</span>");
  });

  $("#action7").click(function() {
    $("#box4").remove();
  });
});
```

Concept Combination

This practice combines:
- ready
- click
- selector
- css()
- before()
- remove()

The library logic becomes visible in practice.

jQuery DOM Manipulation Methods

Content can be inserted or removed with short methods.

Method meanings

- prepend adds inside the beginning.
- append adds inside the end.
- before adds before the selected element.
- after adds after the selected element.
- empty clears inside content.
- remove deletes the element itself.

```
$("#prependBtn").click(function () {
  $("#frame").prepend("<span>New Box</span>");
});

$("#appendBtn").click(function () {
  $("#frame").append("<span>New Box</span>");
});

$("#emptyBtn").click(function () {
  $("#box4").empty();
});

$("#removeBtn").click(function () {
  $("#box4").remove();
});
```

Draw where the new content goes: inside beginning, inside end, before, or after.

Mini Project: Interactive Study Panel

At the end of this unit, students can build a small single-file interactive page.

HTML Structure

Input field
Buttons
Result area
List area
Status message

JavaScript Logic

Read value
Check it
Add to array
List with a loop
Print the result

Interactions

click
input
submit
timer
class/style change

Learning goal: being able to say, “When I press a button, I can intentionally create a change on the page.”

Mini Project Workflow

A small project connects the moving parts.

Start with a goal

Example: add study tasks and instantly see the updated list.

Build data flow

HTML structure first, then array, then render function, then event.

Test states

Try empty input, normal input, repeated clicks, and reset behavior.

Good small projects combine input handling, arrays, functions, rendering, and feedback.

Study Planner Core Script

Arrays + render function + click event.

```
const tasks = ["Review Flexbox notes", "Finish Week 10 assignment"];
const taskInput = document.getElementById("taskInput");
const taskList = document.getElementById("taskList");
const addTaskBtn = document.getElementById("addTaskBtn");

function renderTasks() {
  let html = "";
  tasks.forEach(function (task, index) {
    html += `<div class="task-item">${index + 1}. ${task}</div>`;
  });
  taskList.innerHTML = html;
}

addTaskBtn.addEventListener("click", function () {
  const value = taskInput.value.trim();
  if (!value) return;
  tasks.push(value);
  taskInput.value = "";
  renderTasks();
});

renderTasks();
```

Concept map

- tasks stores the data.
- renderTasks rebuilds the visible list.
- The click event reads input and pushes a new task.
- The empty-input guard prevents blank tasks.

In-Class Practice Flow

Each topic can be practiced through a short explanation, code inspection, and one small change.

1. Understand

Define the concept briefly and simply.
Connect it to where it is used.
Use a familiar web example.

2. Run

Open the single-file example in the playground.
Read the code.
Observe the result in the browser.

3. Modify

Change one value.
Predict the result.
If something fails, check the selector, value, and event flow.

Playground Practice Sequence

A practical rhythm for learning JavaScript by doing.

Practice pattern: run the code, change one value, predict the result, then test.

Common Mistakes

This list summarizes common debugging problems in beginner JavaScript work.

Writing an id incorrectly or using different names in HTML and JavaScript.
Forgetting [0] when using a collection returned by a class selector.
Treating numbers inside quotation marks as real numbers.
Writing if conditions in the wrong order.
Defining a function but forgetting to call it.
Forgetting clearInterval for setInterval.
Using the $ sign without connecting jQuery.

```
// Debug checklist
1. Is the selector correct?
2. Does the element exist on the page?
3. Is the event really firing?
4. Is the value the expected type?
5. Is there a console error?
```

Common Beginner Mistakes

Use these as live debugging checkpoints.

Selector mistakes

Wrong id/class name, selecting before the element exists, or forgetting [0] after collection selectors.

Value mistakes

Forgetting quotes around strings, treating input text as a number, or mixing = with comparison.

Flow mistakes

Forgetting to call a function, forgetting break in switch, or creating an infinite loop.

Debug checklist: selector → element exists → event fires → value type → console error.

Assessment Map

Quiz and exam questions can focus on concept definitions, code reading, and small code completion tasks.

Concept Questions

Selector types
Data types
Operators
Loop types
Functions and return

Code Reading

What appears on the screen when this code runs?
Which if block runs?
What is the array index result?

Code Completion

Write the missing selector.
Connect the event function.
Complete the loop condition.
Build the jQuery ready block.

Success is measured not only by memorizing code, but by being able to predict the visible result of code.

Mini Quiz Prompts

These questions check understanding, not memorization.

Quick check prompts

1. What does document.getElementById("message") select?
2. Why does "1" + "98" + "8" produce 1988?
3. What is the difference between let and const?
4. Why does getElementsByClassName("box")[0] need [0]?

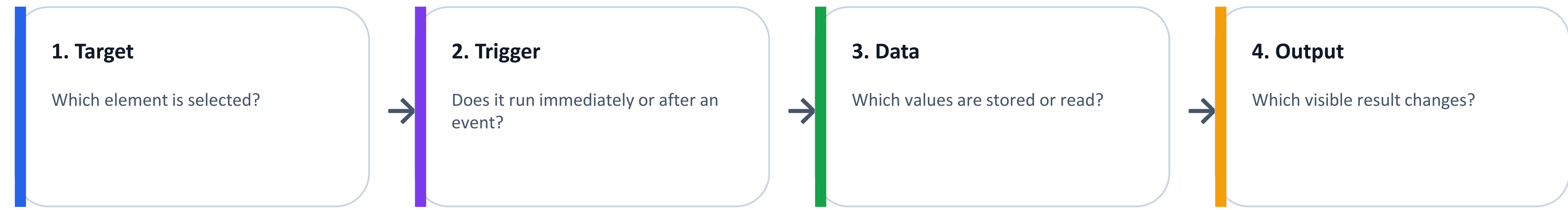
More prompts

5. What does preventDefault do in a form submit event?
6. When should a function return a value?
7. Why is a loop better than repeating the same DOM update?
8. What problem does jQuery try to make easier?

A strong answer connects code to visible browser behavior.

Code Reading Checkpoints

Before running a script, read it in this order.



This reading order helps students explain short scripts in quizzes and exams.

Unit Summary

This unit positions JavaScript as a basic tool for web page interaction.

JavaScript adds a behavior layer on top of HTML and CSS.
Page elements cannot be changed without selectors.
Variables and data types carry program data.
Operators produce calculations and decisions.
Events and functions start user interaction.
Control structures determine the program path.
Arrays, loops, and timers manage repeated work.
jQuery makes some operations shorter and more practical.

Final Message

The value of code is understood through the behavior it creates on the page.

Unit Skills Map

The lesson connects JavaScript concepts to browser behavior.

Structure and selection

Script tags, execution flow, selectors, DOM properties, and visible updates.

Logic and reuse

Variables, data types, operators, conditions, functions, parameters, and return values.

Interaction and repetition

Events, forms, arrays, loops, timers, mini projects, libraries, and jQuery.

The value of code is understood through the behavior it creates on the page.

Next Step: Student Practice

Each topic should be practiced again through small changes.

1. Run the Example

Open the single-file example in the playground and observe the result.

2. Change a Value

Change one text, number, color, duration, or condition value.

3. Explain the Behavior

What happened on the page after the change? Which line caused it?

Main goal: JavaScript should be learned as visible browser behavior, not as abstract syntax.

Rebuild, Modify, Explain

Practice by changing examples, not by copying them unchanged.

Rebuild

Choose one example and recreate it from memory using the same concept.

Modify

Change text, numbers, colors, timing, conditions, or input fields.

Explain

Describe the selector, the event, the stored values, and the visible result.

Final habit: every line of JavaScript should be connected to a browser result.