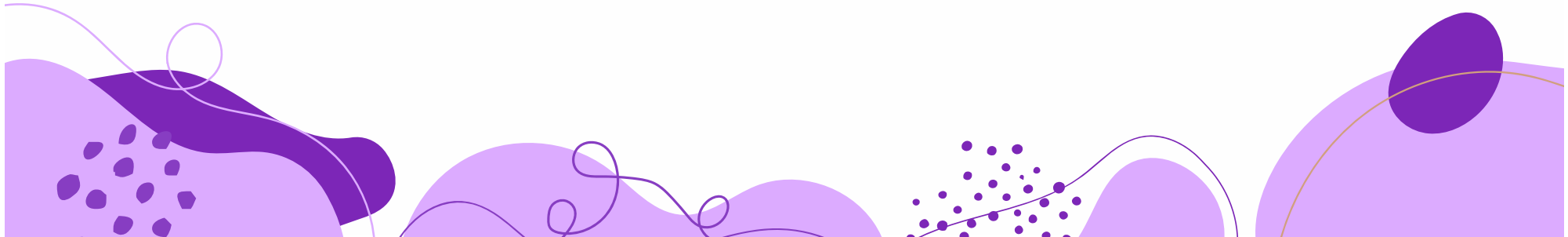
Arrays and Objects

in JavaScript

THRIVE
Women for Women

Storing and Accessing Data Dynamically



Recap: The Building Blocks of Logic

Transitioning from Logic to Data Structures

We've covered the foundational logic of JavaScript. This week, we apply that logic to structured data.

Goal: Applying these logical tools to organize and manipulate structured data.

Conditionals

Using if/else and switch for decision-making logic (e.g., grading scores).

Loops

Implementing for and while to automate repetitive tasks efficiently.

Functions

Creating reusable code blocks with parameters and return values.

Why Data Containers Matter

Bundling Information for Efficient Processing

The Analogy & Efficiency

The Analogy: Handling data without arrays/objects is like carrying groceries one apple at a time.

Efficiency: Data containers allow you to bundle, organize, and manipulate collections as a single unit.

Real-World Use

- Arrays: User lists, high scores, menu items.
- Objects: User profiles,

They turn static variables into dynamic applications .

Session Roadmap

Our Journey Through JavaScript Data

01 | Introduction and Session Overview

02 | Arrays: Basics, Creation, and Access

03 | Essential Array Methods (push, pop, map, filter)

04 | Iterating Over Arrays

05 | Objects: Key-Value Pairs and Access

06 | Iterating Over Objects

07 | Live Coding Exercises in VS Code

08 | Homework and Wrap-up

Learning Objectives

What You Will Master Today

Arrays
Learn to store and access ordered lists of data using zero-based indexing.

Objects
Understand how to store key-value pairs for structured information.

Iteration
Master looping through both structures to process data efficiently.

Arrays: The Ordered List

Definition, Creation, and Indexing

THRIVE
Women for Women

CONCEPTS

Definition: An ordered collection of values, similar to a shopping list or a playlist.

> Indexing: Arrays are zero-indexed. The first element is always at index 0.

This structure allows for extremely fast data retrieval when the position is known.

IMPLEMENTATION

```
// 1. Creation using Array Literal
const fruits = ["apple", "banana", "cherry"];

// 2 Creation of an array with mixed data
types
const mixedArray = ["apple", 42, "banana",
3.14, "cherry", true];

// 3. Accessing Elements
console.log(fruits[1]);
// Output: "banana"
```

Accessing and Modifying Arrays

Reading, Writing, and Checking Length

Operation	Syntax	Example
Read	array[index]	scores[0]
Write	array[index] = val	scores[0] = 90
Length	array.length	scores.length

```
const scores = [85, 92, 78];

// Read the first score
console.log(scores[0]); // 85

// Update the first score
scores[0] = 90;

// Check the new length
console.log(scores.length); // 3
```

Essential Array Methods (Mutating)

Changing the Original Array

THRIVE
Women for Women

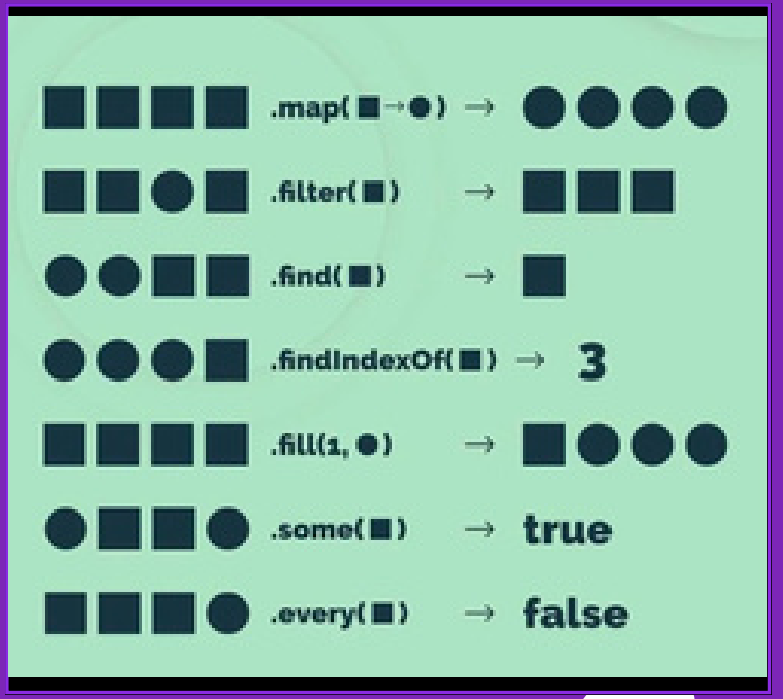
push(item) / pop()

Adds to the end / Removes from the end.

shift() / unshift(item)

Removes from the beginning / Adds to the beginning.

```
let nums = [1, 2, 3]; //
nums.push(4);                    [1, 2, 3, 4]
let    last    =    nums.pop();     // last is 4
console.log(nums);          //    [1, 2, 3]
```



Essential Array Methods (Non-Mutating)

Transforming Data Without Changing the Original

THRIVE
Women for Women

CORE METHODS

map(callback)

Creates a new array by transforming every element.

filter(callback)

Creates a new array with elements that pass a test.

f or Each (callback)

Executes a function for each element (no return value).

Pro Tip: Methods like map and filter don't mutate originals—immutability wins!

EXAMPLE

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

// Map: Transform each item
const doubled = nums.map(n => n * 2);
// [2, 4, 6, 8]

// Filter: Keep items matching condition
const evens = nums.filter(n => n % 2 === 0);
// [2, 4]

// Original array is unchanged
console.log(nums);
// [1, 2, 3, 4]
```

Iterating Over Arrays

Processing Every Item Efficiently

Traditional for Loop

Best when you need the index or specific control over the iteration flow.

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

for (let i = 0; i < fruits.length; i++) {
        console.log(fruits[i]);
}
```

Modern forEach

The cleanest way for simple iteration where you don't need to break the loop.

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

for...of Loop

The most readable way to iterate over values without managing an index.

```
for (const fruit of fruits) {
        console.log(fruit.toUpperCase());
}
```

Objects: The Key-Value Map

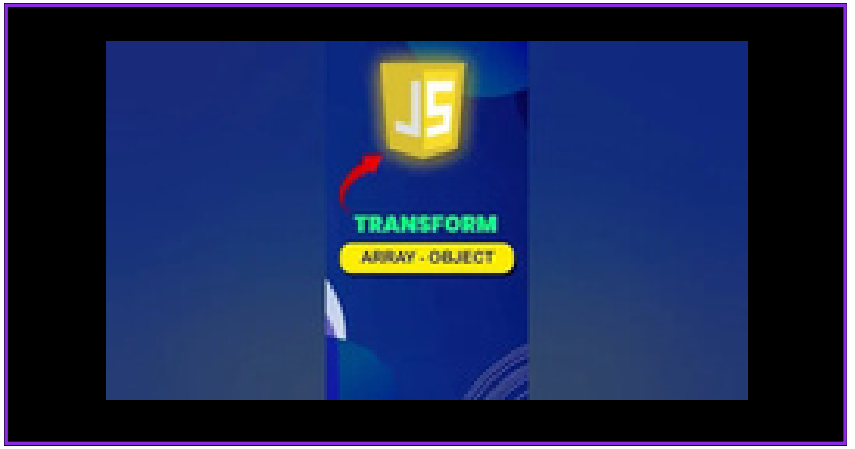
Definition, Creation, and Basic Structure

CONCEPTS

Definition: An unordered collection of key-value pairs, like a contact card or a dictionary.

> Structure: Keys are strings (or Symbols); values can be any data type, including arrays or other objects.

Objects are the primary way we represent complex, structured entities in JavaScript.




```
const user = {
    name: "Alice",
    age: 22,
    isAdmin: true,
    hobbies: ["reading", "coding"]
};
```

Accessing and Modifying Objects

Dynamic and Static Property Management

Operation	Syntax	Use Case
Dot Access	obj.key	Known property name (static).
Bracket Access	obj["key"]	Dynamic keys or variables.
Add/Update	obj.key = val	Set or change properties.

```
const user = { name: "Alice", age: 22 };

// Add a new property
user.email = "alice@email.com";

// Access a property dynamically
let prop = "age";
console.log(user[prop]); // Output: 22

// Get all keys
const keys = Object.keys(user);
```

Iterating Over Objects

Looping Through Properties and Values

TECHNIQUES

for...in Loop

Iterates over all keys of an object.

Object.keys()

Returns an array of an object's own property names.

Object.values()

Returns an array of an object's own property values.

EXAMPLE

```
const user = { name: "Bob", score: 95 };

// 1. Using for...in
for (let key in user) {
    console.log(`${key}: ${user[key]}`);
}

// 2. Using Object.keys()
Object.keys(user).forEach(key => {
    console.log(key); // "name", "score"
});

// 3. Using Object.values()
Object.values(user).forEach(val => {
    console.log(val); // "Bob", 95
});
```

Live Coding Exercise 1 & 2

Practice: Arrays and Methods

Exercise 1: Array Basics

- Create: Declare an empty array `shoppingList`.
- Add: Use `push()` to add "Milk", "Bread", "Eggs".
- Remove: Use `pop()` to remove the last item.
- Access: Print the item at index 0.
- Modify: Change index 1 to "Whole Wheat Bread".
- Loop: Use `for...of` to print every item.

Exercise 2: Array Methods

Data: Start with `[75, 88, 92, 65, 98]`

Filter: Create `passingScores` for grades >= 80.

Map: Create `curvedScores` by adding 5 points to each.

Iterate: Use `forEach` to log each curved score.

Goal: Solidify your understanding of array manipulation through hands-on practice.

Live Coding Exercise 3 & 4

Practice: Objects and Combined Data Structures

Exercise 3: The Book Profile

Create a **book** object with title, pages, and published year. >

Add an **author** property dynamically. >

Access title via dot notation and pages via bracket notation.

Loop through all keys using **Object.keys()**.

Exercise 4: Array of Objects (The CRM)

Start with an array of **user** objects (id, name, isActive).

Filter for **activeUsers** where **isActive** is true. >

Map to create a **userNames** array of strings. >

Loop to print the name of each active user.

Goal: Master complex data handling in real-world scenarios.

Homework and Wrap-up

Key Takeaways & Next Steps

- Arrays: Tidy lists you can shape with push, pop, map, and filter.
- Objects: Named info you can read with dot or bracket styles and loop through.
- Smart loops make data easier to explore.

Homework

- Array Task: Grade Scores Easy steps: Make a function processScores that gets an array like [65, 82, 78, 55, 91]. Filter scores >=70. Add 5% to them (multiply by 1.05). Use reduce to find the average.

- Object Task: Inventory Manager Description: Create an inventory object to track store items, like { "apples": 10, "bananas": 5 } (quantity as values). Write a function addItem(inventory, item, qty) to dynamically add or update items (e.g., addItem(inventory, "oranges", 8)).

Q/A