Intermediate JavaScript Programming

LESSON 3: Working with Arrays and Objects Efficiently

Learning Objectives:

By the end of this lesson, participants will be able to:

- Extract values from arrays and objects using destructuring.
- Use spread/rest operators to combine and decompose data.
- Apply `map`, `filter`, and `reduce` for data manipulation.
- Avoid pitfalls that arise when using these tools without clear mental models.

Lesson Outline:

I. Destructuring Arrays and Objects (15 min)

Destructuring is a syntax that lets you unpack values from arrays or properties from objects into distinct variables.

Array Destructuring:

```
const numbers = [1, 2, 3];
const [first, second] = numbers;
console.log(first);  // 1
console.log(second); // 2
```

You can skip elements:

```
const [a, , c] = [10, 20, 30];
console.log(c); // 30
```

Object Destructuring:

```
const person = { name: "Greg", age: 65 };
const { name, age } = person;
console.log(name); // Greg
```

If the variable name is different from the property:

```
const { name: fullName } = person;
console.log(fullName); // Greg
```

Commentary:

Destructuring can look elegant, but overuse or misuse can make code harder to read.

While it provides a compact way to extract values, it relies entirely on property names matching variable names, which reduces predictability. For learners or teams focused on clarity, the long form is preferable:

```
const name = person.name;
const age = person.age;
```

This style is easier to read, trace, and debug — especially when dealing with changing data structures or shared code. For the remainder of this course, examples will avoid destructuring in favor of explicit assignments. It's often more clear to extract a property in a separate statement if the code is meant for learners or long-term maintainability.

II. Spread and Rest Operators (15 min)

The `...` operator is used in two related ways:

- **Spread** — to expand elements or properties
- **Rest** — to collect remaining items into a new variable

Spread Example (Arrays):

```
const a = [1, 2];
const b = [3, 4];
const combined = [...a, ...b];
console.log(combined); // [1, 2, 3, 4]
```

Spread Example (Objects):

```
const original = { x: 1, y: 2 };
const copy = { ...original, z: 3 };
console.log(copy); // { x: 1, y: 2, z: 3 }
```

Rest Example:

```
const [head, ...tail] = [1, 2, 3, 4];
console.log(head); // 1
console.log(tail); // [2, 3, 4]
```

Commentary:

Because `...` means two things depending on context, it's easy to misread what a line is doing. Always read the whole line to determine whether it's spreading or gathering.

III. Array Methods: map, filter, reduce (20 min)

These methods are used for transforming and analyzing data. They do not mutate the original array.

map:

Creates a new array by applying a function to each item.

```
const numbers = [1, 2, 3];
const doubled = numbers.map(function(n) {
  return n * 2;
});
console.log(doubled); // [2, 4, 6]
```

filter:

Creates a new array by keeping only the elements that match a condition.

```
const numbers = [1, 2, 3, 4];
const even = numbers.filter(function(n) {
  return n % 2 === 0;
});
console.log(even); // [2, 4]
```

reduce:

Reduces an array to a single value by accumulating. The second argument to `reduce` is the **initial value** of the accumulator.

```
const numbers = [1, 2, 3, 4];
const sum = numbers.reduce(function(acc, value) {
  return acc + value;
}, 0);
console.log(sum); // 10
```

Sidebar: What about `forEach`?

`forEach` lets you loop through elements, but it doesn't return a new array or value. It is used for side effects such as logging or updating the DOM.

```
const numbers = [1, 2, 3];
numbers.forEach(function(n) {
  console.log(n);
});
// Output: 1
//         2
//         3
```

Commentary:

These methods are powerful but can be opaque to readers unfamiliar with functional patterns. Beginners often confuse `map` and `forEach`, or misuse `reduce` with complex logic. When in doubt, explain your intention in a comment or use named helper functions for clarity.

IV. Combining Techniques (5 min)

You can chain these methods together for expressive logic:

```
const numbers = [1, 2, 3, 4, 5];
const result = numbers
  .filter(function(n) { return n % 2 !== 0; })
  .map(function(n) { return n * n; });

console.log(result); // [1, 9, 25]
```

Commentary:

Method chaining is concise, but deeply chained logic can hide what's going on. Break up the code with named steps if it helps clarity.

V. Recap & Q&A (5 min)

- Review destructuring and spread/rest syntax.
- Emphasize clarity and intention when using `map`, `filter`, and `reduce`.
- Encourage use of full syntax over shorthands while learning.

Final Multiple-Choice Question:

Which method would you use to compute the total of an array? A. map B. reduce C. filter D. join

(Answer: B. reduce — it collapses an array into a single value.)