19. map, filter & reduce

📘 Index (Table of Contents)

❓ Questions and Answers

1. What Are Higher-Order Functions in JavaScript?

Q: What defines a higher-order function in JavaScript?

A: A higher-order function is a function that either takes one or more functions as arguments, returns a function as a result, or both.

Analogy: Think of higher-order functions as tools in a toolbox. Just as a toolbox allows you to store and use various tools (functions), higher-order functions allow you to store and manipulate other functions, making your code more flexible and modular.

---

2. How Does the `map()` Function Work?

Q: What is the purpose of the `map()` function in JavaScript?

A: The `map()` function creates a new array populated with the results of calling a provided function on every element in the calling array.

Example:

```
const numbers = [1, 2, 3, 4];
const squares = numbers.map(num => num * num);
console.log(squares); // [1, 4, 9, 16]
```

Explanation: Here, `map()` iterates over each element in the `numbers` array, applies the provided function (`num => num * num`), and returns a new array with the squared values.

---

3. What Is the Purpose of the `filter()` Function?

Q: How does the `filter()` function operate in JavaScript?

A: The `filter()` function creates a new array with all elements that pass the test implemented by the provided function.

Example:

```
const numbers = [1, 2, 3, 4, 5];
const evens = numbers.filter(num => num % 2 === 0);
console.log(evens); // [2, 4]
```

Explanation: In this case, `filter()` examines each element in the `numbers` array and includes it in the new array if it satisfies the condition (`num % 2 === 0`), effectively filtering out the odd numbers.

---

4. How Is the `reduce()` Function Used?

Q: What does the `reduce()` function do in JavaScript?

A: The `reduce()` function executes a reducer function (that you provide) on each element of the array (from left to right) to reduce it to a single value.

Example:

```
const numbers = [1, 2, 3, 4];
const sum = numbers.reduce((acc, num) => acc + num, 0);
console.log(sum); // 10
```

Explanation: Here, `reduce()` iterates through each element in the `numbers` array, accumulating the sum of the elements. The `0` is the initial value of the accumulator (`acc`).

---

5. How Do `map()`, `filter()`, and `reduce()` Differ?

Q: What distinguishes `map()`, `filter()`, and `reduce()` from each other?

A: Each function serves a unique purpose:

- `map()`: Transforms each element in the array and returns a new array of the same length.
- `filter()`: Creates a new array with all elements that pass the test implemented by the provided function, potentially resulting in a shorter array.
- `reduce()`: Reduces the array to a single value by applying a function against an accumulator and each element (from left to right).

Analogy: Consider an array as a list of ingredients:

- `map()`: Changes each ingredient (e.g., chopping vegetables).
- `filter()`: Selects only certain ingredients (e.g., picking only ripe fruits).
- `reduce()`: Combines all ingredients into a single dish (e.g., making a soup).

---

6. What Are Some Best Practices for Using These Functions?

Q: How can developers effectively use `map()`, `filter()`, and `reduce()`?

A: Best practices include:

- **Use Descriptive Callback Functions**: Ensure that the functions passed to `map()`, `filter()`, and `reduce()` are descriptive and self-explanatory.
- **Avoid Side Effects**: The callback functions should not have side effects; they should not modify external variables.
- **Chain Methods When Appropriate**: You can chain these methods to perform complex transformations and filtering operations in a readable manner.

Example:

```
const numbers = [1, 2, 3, 4, 5];
const result = numbers
  .map(num ⇒ num * 2)
  .filter(num ⇒ num > 5)
  .reduce((acc, num) ⇒ acc + num, 0);
console.log(result); // 18
```

Explanation: In this example, `map()` doubles each number, `filter()` retains numbers greater than 5, and `reduce()` calculates the sum of the remaining numbers.

🔑 Summary and Key Takeaways

- **Higher-Order Functions**: Functions that take other functions as arguments or return functions as results.
- `map()`: Transforms each element in an array and returns a new array of the same length.
- `filter()`: Creates a new array with all elements that pass the test implemented by the provided function.
- `reduce()`: Reduces the array to a single value by applying a function against an accumulator and each element.
- **Best Practices**: Use descriptive callback functions, avoid side effects, and chain methods when appropriate to write clean and efficient code.