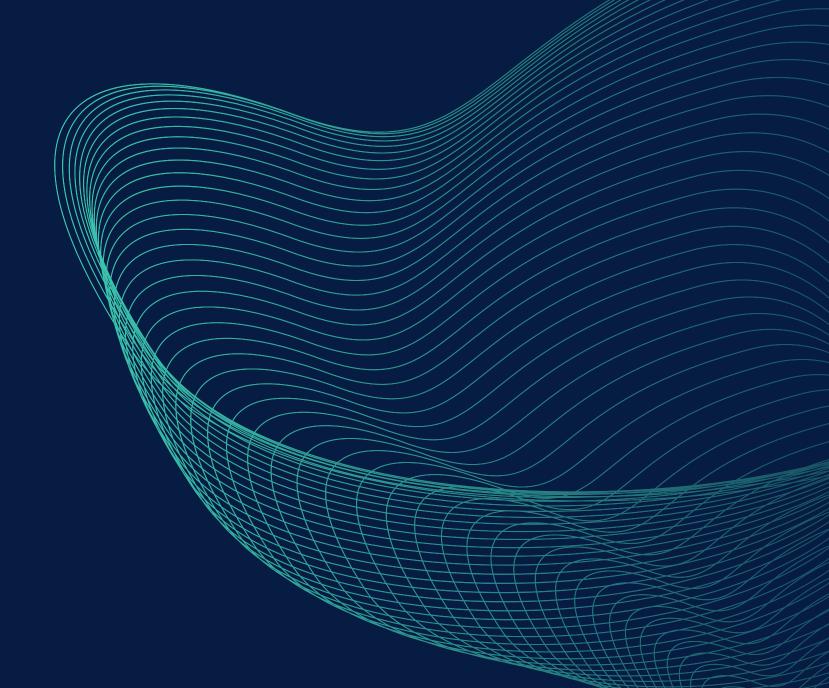


ECMA Script

By Rohan Chhetry



Let and Const

- 'let' and 'const' are block-scoped declarations introduced in ES6.
- let`allows variable reassignment, while `const` is used for constants that cannot be reassigned.

Block scope refers to the region of code within curly braces {} in which variables and constants are defined and accessible. In JavaScript, the introduction of let and const in ECMAScript 6 (ES6) brought block-scoped declarations, distinguishing them from var, which is function-scoped.

Let and Const

Block-scope

```
if (true) {
  let x = 10; // block-scoped
  console.log(x); // Output: 10
}
```

console.log(x); // Error: x is not defined outside the block

Function-scope

```
if (true) {
  var y = 20; // function-scoped
  console.log(y); // Output: 20
}

console.log(y); // Output: 20 (accessible
  outside the block)
```

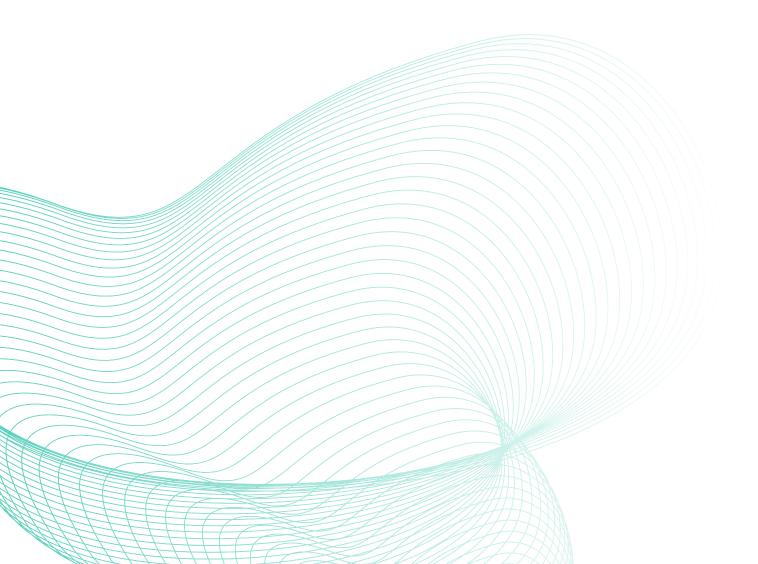
Problem 1

Declare a variable using let and another variable as a constant using const. Assign values to them and try to reassign a new value to the constant.

Solution 1

```
let variable1 = "I can be reassigned";
const constant1 = "I cannot be reassigned";
variable1 = "I am reassigned"; // Valid
// constant1 = "This will result in an error"; // Invalid
```

Map, Reduce, Filter



Мар

map(): Applies a function to all elements of an array and returns a new array.

Reduce

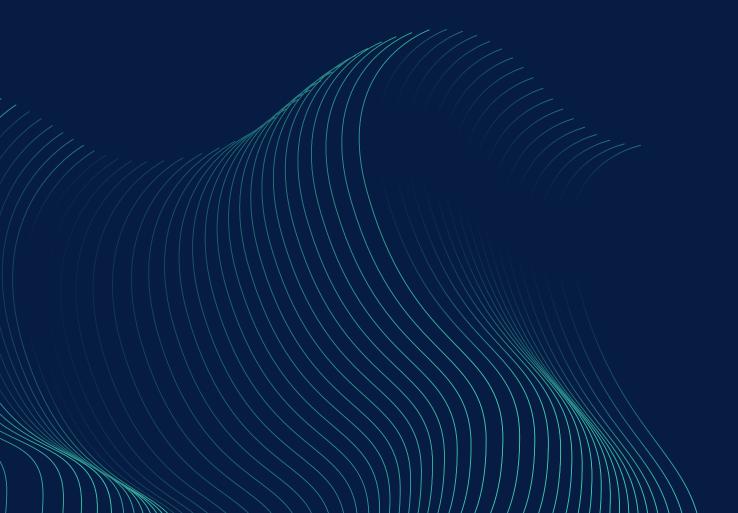
reduce(): Reduces an array to a single value by applying a function to each element

Filter

filter(): Creates a new array with elements that pass a test.

Problem 2

Given an array of numbers, use map, reduce, and filter to find the sum of squared even numbers.



Solution 2

```
const numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10];
// Map: Square each number
const squaredNumbers = numbers.map(num => num ** 2);
// Filter: Select even numbers
const evenNumbers = squaredNumbers.filter(num => num % 2 === 0);
// Reduce: Sum the squared even numbers
const sum = evenNumbers.reduce((acc, num) => acc + num, 0);
console.log(sum); // Output: 220
```

Arrow Function

- Arrow functions provide a concise syntax for writing function expressions.
- They do not have their own this and are not suitable for methods.

```
// Regular function expression
const add = function(a, b) {
  return a + b;
};
```

```
// Arrow function
const addArrow = (a, b) => a + b;
```

- 1. Concise Syntax
- 2. Implicit Return
- 3. No Binding of 'this'
- 4. Shorter Syntax for single Parameter
- 5. No `arguments` Object
- 6. Use Cases
- 7. Not Suitable for Methods
- 8. Lexical Scoping

Arrow Function

Question 3

Convert a regular function to an arrow function that takes two parameters and returns their sum

Arrow Function

Solution 3

```
// Regular function
function add(a, b) {
 return a + b;
// Arrow function
const addArrow = (a, b) => a + b;
console.log(add(5, 3)); // Output: 8
console.log(addArrow(5, 3)); // Output: 8
```

- Template literals allow embedding expressions inside string literals using backticks.
- They support multi-line strings and string interpolation

Template Literals

Problem 4

Create a template literal that includes variables for name and age in a sentence

Solution 4

```
const name = "John";
const age = 25;
```

const sentence = `My name is \${name} and I
am \${age} years old.`;

console.log(sentence); // Output: My name is John and I am 25 years old.