ARROW FUNCTIONS

Arrow Functions

arrow functions were introduced in ES6 as a new way to write functions. They have some important differences from normal functions

```
const example = (x, y) => x + y;
console.log(example(1, 2)); // Output: 3
```

Arrow Functions Use cases:

Shorter syntax: Arrow functions have a shorter and more concise syntax than normal functions, making them easier to read and write. They are particularly useful for small, simple functions that do not require a lot of additional code.

```
// Normal function
function multiply(x, y) {
  return x * y;
}

// Arrow function
const multiply = (x, y) => x * y;
```

Arrow Functions Use cases:

Implicit return: Arrow functions have an implicit return feature, which means that if the function body is a single expression, that expression is automatically returned without having to use the return keyword

```
// Normal function
function multiply(x, y) {
  return x * y;
}

// Arrow function
const multiply = (x, y) => x * y;
```

SPREAD OPERATOR

Spread Operator

The spread operator in JavaScript is denoted by three dots ... and it allows an iterable (such as an array or a string) or an object to be expanded into individual elements or components

Spread Operator

used to pass an array as individual arguments to a function

```
function sum(a, b, c) {
  return a + b + c;
}

const numbers = [1, 2, 3];

// without spread operators
console.log(sum(numbers[0],numbers[1],numbers[2]
))

// with spread operators
console.log(sum(...numbers)); // output: 6

// this doesn't work
console.log(sum(numbers));
```

Spread Operator

the spread operator can be used to combine two or more arrays into a new array.

```
const arr1 = [1, 2, 3];
const arr2 = [4, 5, 6];
const combined = [...arr1, ...arr2];

console.log(combined); // output: [1, 2, 3, 4, 5, 6]
```

Spread Operator

can be used to merge two or more objects into a new object.

```
const obj1 = { a: 1, b: 2 };
const obj2 = { c: 3, d: 4 };
const combined = { ...obj1, ...obj2 };

console.log(combined); // output: { a: 1, b: 2, c: 3, d: 4 }
```

Spread Operator

can also be used with rest parameters in function declarations . Rest parameters allow a variable number of arguments to be passed to a function, and are denoted by three dots ... before the last parameter in the function declaration

```
function sum(...numbers) {
  return numbers.reduce((total, num) => total + num, 0);
}

console.log(sum(1, 2, 3)); // output: 6
console.log(sum(4, 5, 6, 7)); // output: 22
```

Spread Operator

You can have fixed parameters alongside a variable number of arguments

```
function printDetails(name, age, ...hobbies) {
  console.log(`Name: ${name}, Age: ${age}`);
  console.log(`Hobbies: ${hobbies.join(', ')}`);
}

printDetails("John", 25, "Reading", "Swimming", "Coding");
//In this example, the first two arguments are captured by name and age, while all
subsequent arguments are captured by the hobbies array. "Swimming", "Coding");
```

Destructuring

Destructuring

allows you to extract data from arrays and objects and assign them to variables. It makes it easier to work with complex data structures by simplifying the process of extracting and assigning values.

Array Destructuring

you can extract elements from an array and assign them to individual variables

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

const [a, b, c, d] = numbers;

console.log(a); // output: 1
console.log(b); // output: 2
console.log(c); // output: 3
console.log(d); // output: 4
```

Object Destructuring

you can extract properties from an object and assign them to individual variables.

```
const person = { name: 'John', age: 30, gender: 'male' };

const { name, age, gender } = person;

console.log(name); // output: "John"
console.log(age); // output: 30
console.log(gender); // output: "male"
```

Default Values

you can extract properties from an object and assign them to individual variables.

```
const person = { name: 'John', age: 30 };

const { name, age, gender = 'male' } = person;

console.log(name); // output: "John"
console.log(age); // output: 30
console.log(gender); // output: "male"
```

Without Default Value:

If you don't provide a default value and try to destructure a non-existing property, the variable will be assigned the value undefined

```
const obj = { a: 1 };
const { a, b } = obj;
console.log(a); // Outputs: 1
console.log(b); // Outputs: undefined
```

Rest/Spread

you can also use the rest and spread operators with destructuring. The rest operator allows you to extract the remaining elements of an array or object into a new array or object, while the spread operator allows you to combine two or more arrays or objects into a single array or object

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

const [a, b, ...rest] = numbers;

console.log(a); // output: 1
console.log(b); // output: 2
console.log(rest); // output: [3, 4]
```

Rest/Spread

you can also use the rest and spread operators with destructuring. The rest operator allows you to extract the remaining elements of an array or object into a new array or object, while the spread operator allows you to combine two or more arrays or objects into a single array or object

```
const person = {
  name: 'John',
  age: 30,
  city: 'New York',
  state: 'NY'
};

const { name, age, ...location } = person;

console.log(name); // output: John
console.log(age); // output: 30
console.log(location); // output: { city: 'New York', state: 'NY' }
```

Rename the property

Usually with normal distracting the variable that you extract from the object has the same name as the property in the object but some times that might not be good because you might have another variables with the same name of the property so you would need to change the name you can do that by using (:) and put the variable name that you want after it.

```
const person = {
  firstName: 'John',
  lastName: 'Dani',
};

const { firstName: name } = person;

console.log(name); // output: John
console.log(firstName); // ReferenceError: firstName is not defined
```

Arrays

Arrays

special type of object that allows you to store a collection of values in a single variable. Unlike objects, arrays are ordered and can be accessed using an index, which is a numeric value representing the position of an element in the array.

```
const numbers = [1, 2, 3, 4, 5];
```

Array literal notation:

use square brackets [] to define the array and separate its elements with commas

```
const numbers = [1, 2, 3, 4, 5];
```

Using the new Array() constructor:

This method creates an empty array and allows us to add elements later. Here's an example:

```
const fruits = new Array();
fruits[0] = 'apple';
fruits[1] = 'banana';
fruits[2] = 'orange';
```

Using an array method:

There are several built-in array methods in JavaScript that allow you to create arrays, such as Array.from() and Array.of(). Here's an example:

```
const letters = Array.from('hello');
// Expected output: Array ['h', 'e', 'l', 'l', 'o']
const arr = Array.of('foo', 2, 'bar', true);
// Expected output: Array ["foo", 2, "bar", true]
```

Array access

Arrays use zero-based indexing, we can access arrays elements using the index number

```
console.log(numbers[2]); // Output: 3
console.log(fruits[0]); // Output: "apple"
```

Array elements types

arrays can hold any type of data. This includes numbers, strings, booleans, objects, other arrays, null, undefined, functions, and so on. This is because JavaScript is dynamically typed, which means you don't have to specify what type of data a variable will hold.

```
let array = [1, 'two', true, { name: 'Alice' }, [5, 6, 7], null, undefined, function() { console.log('hello');} ]
```

Array elements types

arrays can hold other arrays, you can create multi-dimensional arrays

```
let matrix = [[1, 2, 3], [4, 5, 6], [7, 8, 9]];
console.log(matrix[1][1]);  // logs 5
```

Array elements types

An array of objects is simply an array where each element is an object.

```
let cars = [
  { make: 'Tesla', model: 'Model 3', year: 2020 },
  { make: 'Toyota', model: 'Camry', year: 2019 },
  { make: 'Ford', model: 'Mustang', year: 2021 }
];
```

Array elements types

Just like numbers, strings, and objects, functions too can be stored in JavaScript arrays. Here's an example of an array of functions:

```
let functions = [
  function(x) { console.log('Square of x is ' + x * x); },
  function(x) { console.log('Cube of x is ' + x * x * x);
},
  function(x) { console.log('x multiplied by 2 is ' + x *
2); }
];

functions[0](5);  // logs 'Square of x is 25'
functions[1](3);  // logs 'Cube of x is 27'
functions[2](7);  // logs 'x multiplied by 2 is 14'
```

Array common operations

We can also add, delete, or modify elements in an array using array methods like push(), pop(), splice(), shift(), unshift(), and slice().

```
numbers.push(6); // Add a new element to the end of the array
fruits.pop(); // Remove the last element of the array
numbers.splice(2, 1); // Remove one element from the middle of the array
fruits.shift(); // Remove the first element of the array
numbers.unshift(0); // Add a new element to the beginning of the array
const slicedNumbers = numbers.slice(1, 4); // Extract a subset of the array
```

Iterating through Arrays

1. For loop

```
for (var i = 0; i < myArray.length; i++) {
  console.log(myArray[i]); // logs items in myArray
}
```

Iterating through Arrays

2. For...of loop

```
for (var item of myArray) {
  console.log(item); // logs items in myArray
}
```

Array Methods

forEach, map, filter Reduce, find,
some, every and join

ARRAY METHODS

JavaScript provides a rich set of built-in methods for working with arrays. Here are some of the most commonly used methods, along with examples of their usage.

ARRAY METHODS

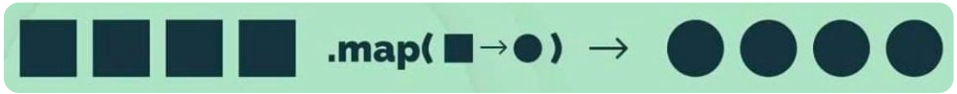
1. forEach: Executes a provided function once for each array element.

```
const numbers = [1, 2, 3, 4, 5];
numbers.forEach((num) => console.log(num));
// logs 1, 2, 3, 4, 5
```

ARRAY METHODS

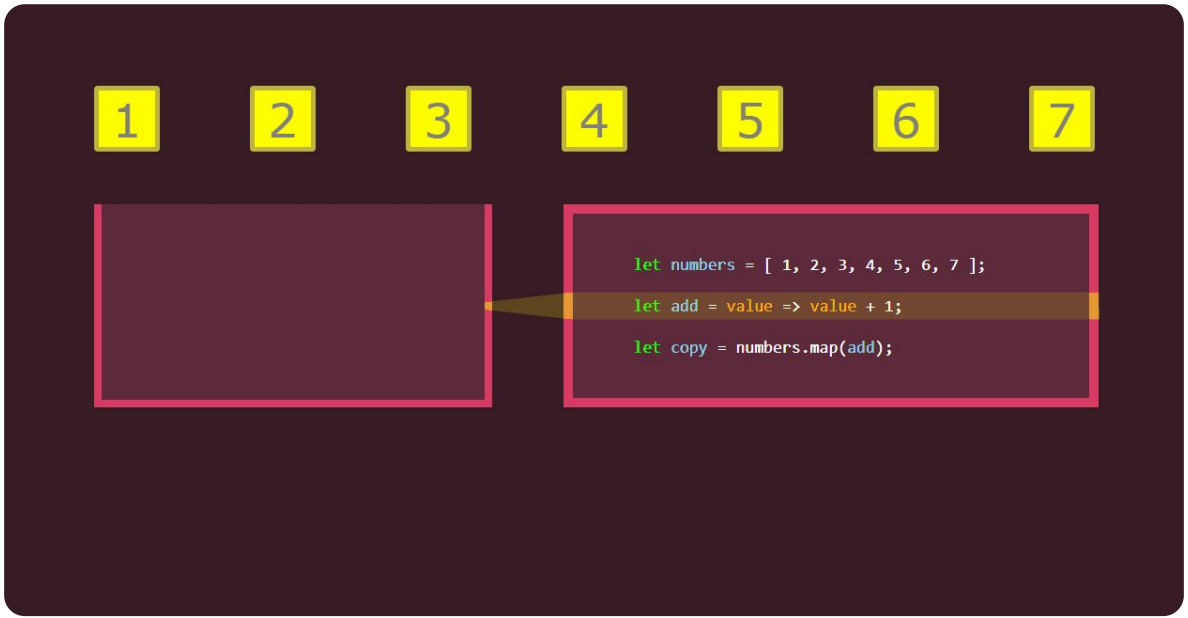
2. Map: Creates a new array with the results of calling a provided function on every element in the calling array.

```
const numbers = [1, 2, 3, 4, 5];
const doubledNumbers = numbers.map((num) => num * 2);
console.log(doubledNumbers); // logs [2, 4, 6, 8, 10]
```



ARRAY METHODS

2. Map:



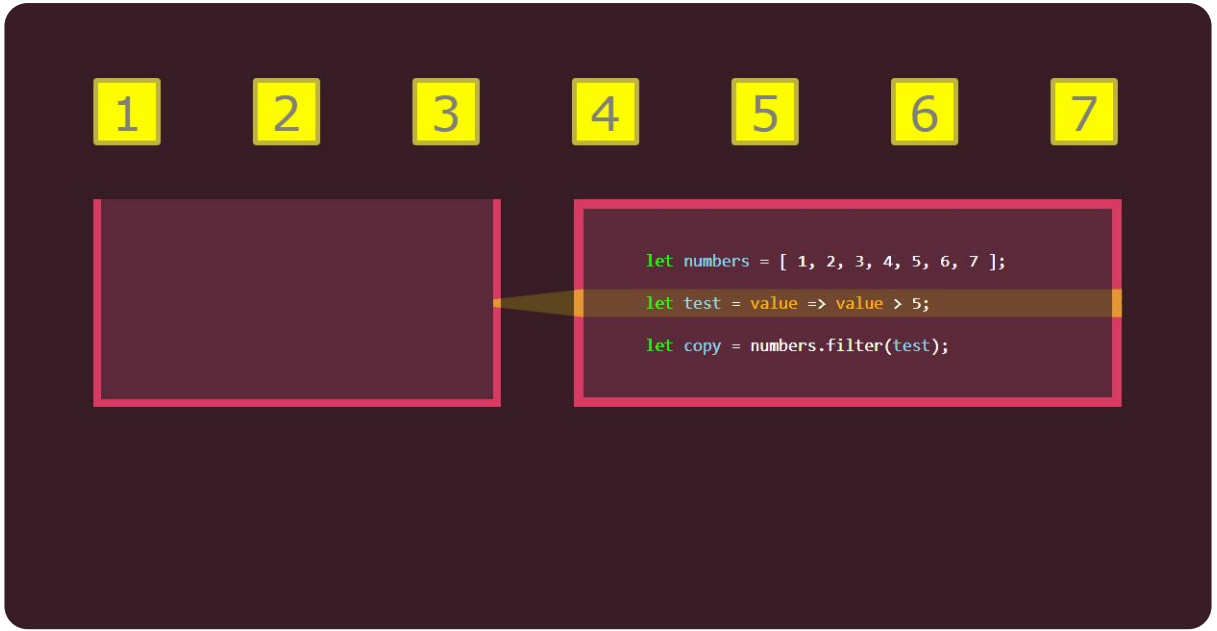
ARRAY METHODS

3. filter: Creates a new array with all elements that pass the test implemented by the provided function.

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

ARRAY METHODS

3. filter:



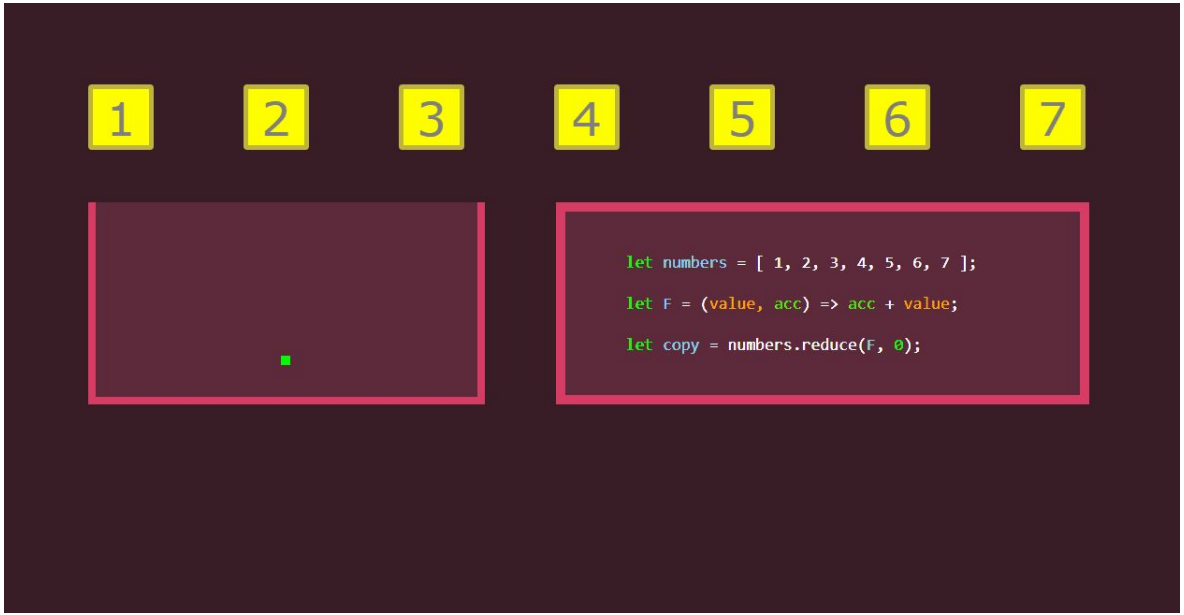
ARRAY METHODS

4. reduce: Executes a reducer function on each element of the array, resulting in a single output value.

```
const numbers = [1, 2, 3, 4, 5];
const sum = numbers.reduce((accumulator, current) => accumulator + current, 0);
console.log(sum); // logs 15
```

ARRAY METHODS

4. reduce:



ARRAY METHODS

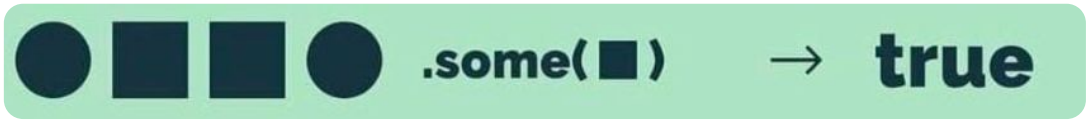
5. find: Returns the first element in the array that satisfies the provided testing function.

```
const numbers = [1, 2, 3, 4, 5];
const foundNumber = numbers.find((num) => num > 3);
console.log(foundNumber); // logs 4
```

ARRAY METHODS

6. some: Returns true if at least one element in the array satisfies the provided testing function.

```
const numbers = [1, 2, 3, 4, 5];
const hasEvenNumber = numbers.some((num) => num % 2 === 0);
console.log(hasEvenNumber); // logs true
```



ARRAY METHODS

7. every: Returns true if all elements in the array satisfy the provided testing function.

```
const numbers = [1, 2, 3, 4, 5];
const allNumbersAreLessThanTen = numbers.every((num) => num < 10);
console.log(allNumbersAreLessThanTen); // logs true
```

ARRAY METHODS

8. Join: it joins all the elements of an array into a string. It returns a new string with the elements of the array separated by a specified separator. The default separator is a comma (,), but you can specify any string to be used as the separator.

```
const colors = ["red", "green", "blue"];
const joinedString = colors.join("-");
console.log(joinedString); // logs "red-green-blue"

const letters =["r","e","d"]
const word = letters.join("");
console.log(word); // logs "red"
```

Array methods Exercise

You are given an array of products. Each product has a name and a price. Your task is to:

1. Use map to create a new array with the names of all the products.
2. Use filter to get a new array with only the products that cost more than 20.
3. Use reduce to find the total cost of all the products.
4. Use join to create a string with the names of all the products, separated by a comma and a space.

```
let products = [
  { name: 'Apple', price: 30 },
  { name: 'Banana', price: 10 },
  { name: 'Cherry', price: 20 },
  { name: 'Date', price: 40 }
];
```

Array methods Solution

```
// Using map to create a new array with names of the products
let productNames = products.map(product => product.name);

// Using filter to create a new array with products that cost more than 20
let expensiveProducts = products.filter(product => product.price > 20);

// Using reduce to find the total cost of all products
let totalCost = products.reduce((sum, product) => sum + product.price, 0);

// Using join to create a string with the names of all products separated by a comma and a space
let productNamesString = productNames.join(', ');

console.log(productNames); // ["Apple", "Banana", "Cherry", "Date"]
console.log(expensiveProducts); // [{ name: 'Apple', price: 30 }, { name: 'Date', price: 40 }]
console.log(totalCost); // 100
console.log(productNamesString); // "Apple, Banana, Cherry, Date"
```