**Topic:  Date methods , callback, for each and map**

**Date Methods**

**Creating Dates**

Var now=new Date();

**Specific Date and Time**

let specificDate = new Date('2024-06-12T10:20:30Z');

**Getting Date Components**

let year = now.getFullYear();// year

let month = now.getMonth(); //0 -11

let day = now.getDate(); // 1-31

**Day of the Week**

let dayOfWeek = now.getDay(); // 0-6 (0 = Sunday, 6 = Saturday)

**Hours, Minutes, Seconds, Milliseconds**

let hours = now.getHours(); // 0-23

 let minutes = now.getMinutes(); // 0-59

let seconds = now.getSeconds(); // 0-59

let milliseconds = now.getMilliseconds(); // 0-999

**Setting Date Components**

now.setFullYear(2025);

now.setMonth(6); // July

 now.setDate(15);

**Set Hours, Minutes, Seconds, Milliseconds**

now.setHours(15);

now.setMinutes(30);

now.setSeconds(45);

now.setMilliseconds(500);

**Formatting Date and Time**

JavaScript provides methods to format dates as strings in different formats:

**toDateString()**

let dateStr = now.toDateString(); // e.g., "Wed Sep 22 2024"

**toTimeString()**

let timeStr = now.toTimeString(); // e.g., "15:30:45 GMT+0530 (India Standard Time)"

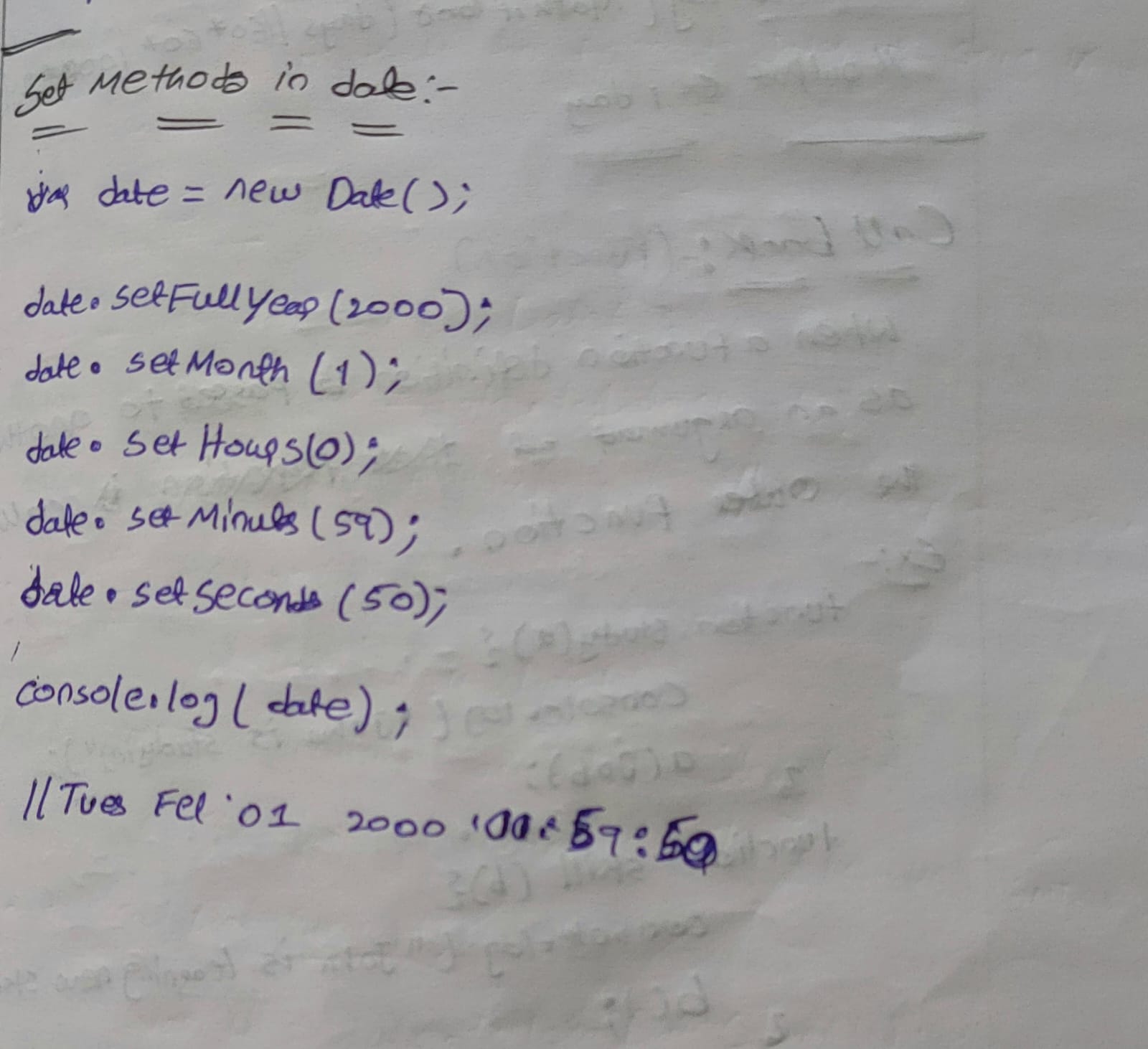
**toLocaleDateString()**

let localDateStr = now.toLocaleDateString(); // e.g., "9/22/2024" in US format

**toLocaleTimeString()**

let localTimeStr = now.toLocaleTimeString(); // e.g., "3:30:45 PM" in US format

These methods allow you to present dates in a human-readable or standardized format.



**Age calculator**

    <input type="date" id="el1" />

    <button onclick="fun()">clicke me</button>

    <script>

      function fun() {

        var d = document.getElementById("el1").value;

        var olddate = new Date(d);

        var newdate = new Date();

**var year = newdate.getFullYear() - olddate.getFullYear();**

**var milliseconds = newdate - olddate;**

**var days = Math.floor(milliseconds / (60 \* 60 \* 24 \* 1000));**

        console.log(days);

        console.log(year);

      }

    </script>

**Callbacks**

callback is a function defintion is passed as an argument to another function and is executed after some operation has been completed. This is a powerful feature that allows for asynchronous programming, enabling tasks to run concurrently without blocking the main execution thread.

**Defining the Callback:**

      // Step 1: Define the callback function

      function myCallback() {

        console.log("Callback function executed!");

      }

**The callback function can be defined separately or inline.**

**// Defining separately**

      function myCallback() {

        console.log("Callback executed!");

      }

      doSomething(myCallback);

**// Defining inline**

      doSomething(function () {

        console.log("Callback executed!");

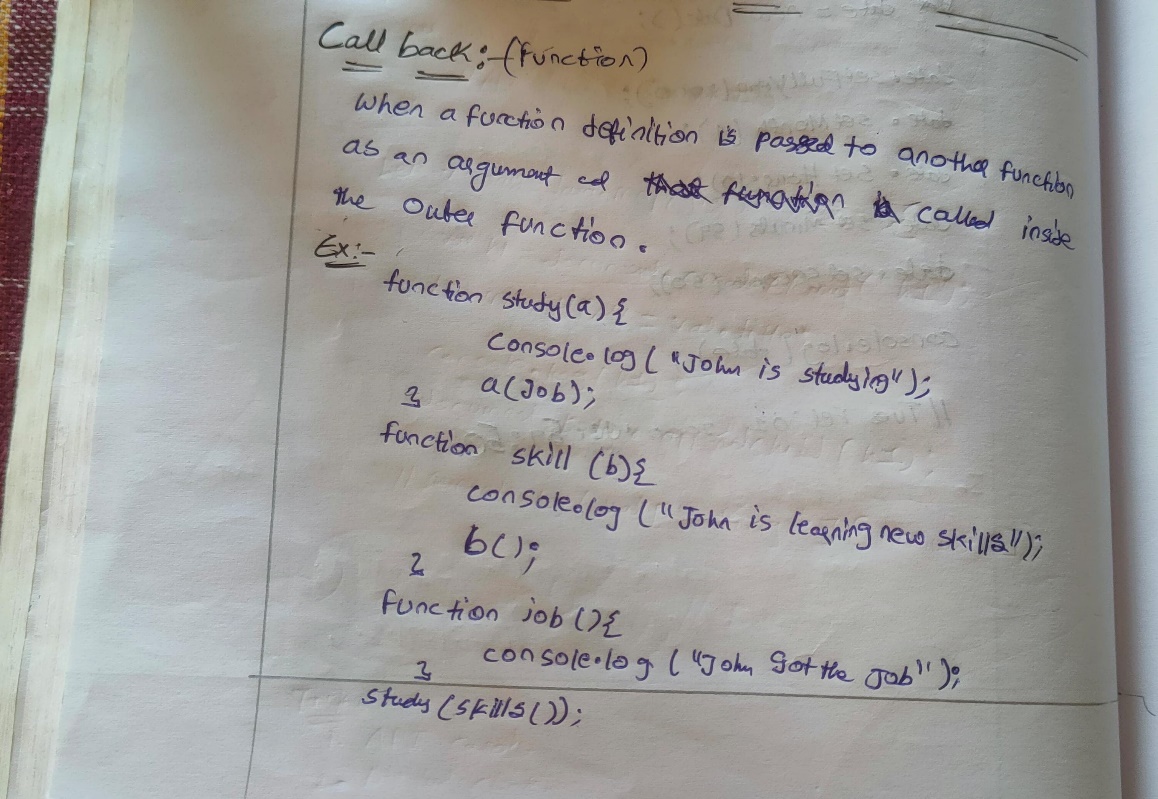
      });

**Step 1: Define a Function that Takes Another Function as a Parameter(HOF)**

**Step 2: Define a function that takes another function as a parameter**

**Step 3: Execute the callback function**

**Step 4: Pass the callback function as an argument**



**Ex-1**

function myFirst() {

  console.log("Hello");

}

function mySecond(a) {

  console.log("Goodbye");

}

var a=myFirst();

mySecond(a);

//Hello

//Goodbye

**Ex-2**

function myFirst() {

  console.log("Hello");

}

function mySecond() {

    myFirst()

  console.log("Goodbye");

}

mySecond();

The problem with the first example above, is that you have to call two functions to display the result.

The problem with the second example, is that you cannot prevent the function from displaying the result when it invokes every time.

function myFirst() {

  console.log("Hello");

}

function mySecond(a) {

  console.log("Goodbye");

}

mySecond(myFirst());//can call two functions at a time

mySecond()//one function

**Usage of Callbacks**

Callbacks are commonly used in situations where you want to perform tasks asynchronously,

**such as:**

Event Handling

Array Methods

Higher-Order Functions

Asynchronous Operations

**Array Iteration and Transformation Methods:**

**Array for each**

Array.forEach() is a method in JavaScript used to iterate over elements in an array. It executes a provided function once for each array element

**Syntax:**

array.forEach(function(value, index, array) {

    // Your code here

});

**value**: The current item in a array.

**index (optional):** The index of the item in the array.

**array (optional):** The array that forEach()

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

numbers.forEach(function(val, index) {

    console.log(`Element at index ${index} is ${val }`);

});

//output

// Element at index 0 is 1

// Element at index 1 is 2

// Element at index 2 is 3

// Element at index 3 is 4

// Element at index 4 is 5

**You can also use arrow functions for a more concise syntax:**

numbers.forEach((number, index) => {

    console.log(`Element at index ${index} is ${number}`);

});

One important thing to note about forEach() is that it doesn't return anything. It simply iterates over the array. If you need to transform the elements of the array and create a new array based on those transformations, you might want to use methods like map() instead.

array.forEach(function(element) {

    return element \* 2; // This return statement has no effect

});

**Map method:**

The map() method in JavaScript is used to create a new array by calling a provided function on every element in the calling array. It doesn't change the original array; instead, it returns a new array with the results of applying the provided function to each element.

**Syntax:**

const newArray = array.map(function callback(currentValue, index, array) {

  // Return element for newArray

});

**value:** The current item in a array.

**index (optional):** The index of the item in the array.

**array (optional):** The array that forEach()

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

const doubledNumbers = numbers.map(function(number) {

  return number \* 2; // Return value determines the value

});

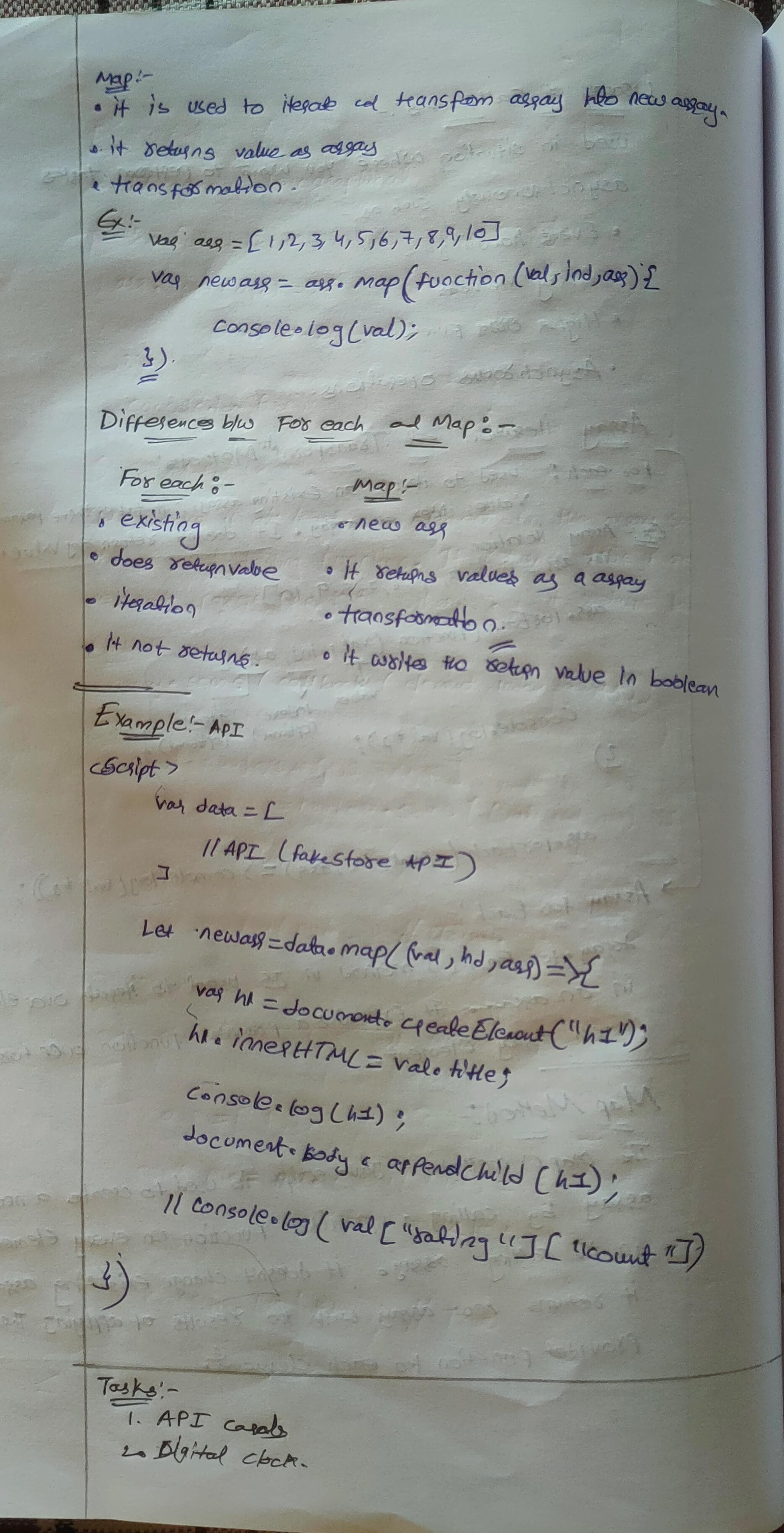
console.log(doubledNumbers); // Output: [2, 4, 6, 8, 10]

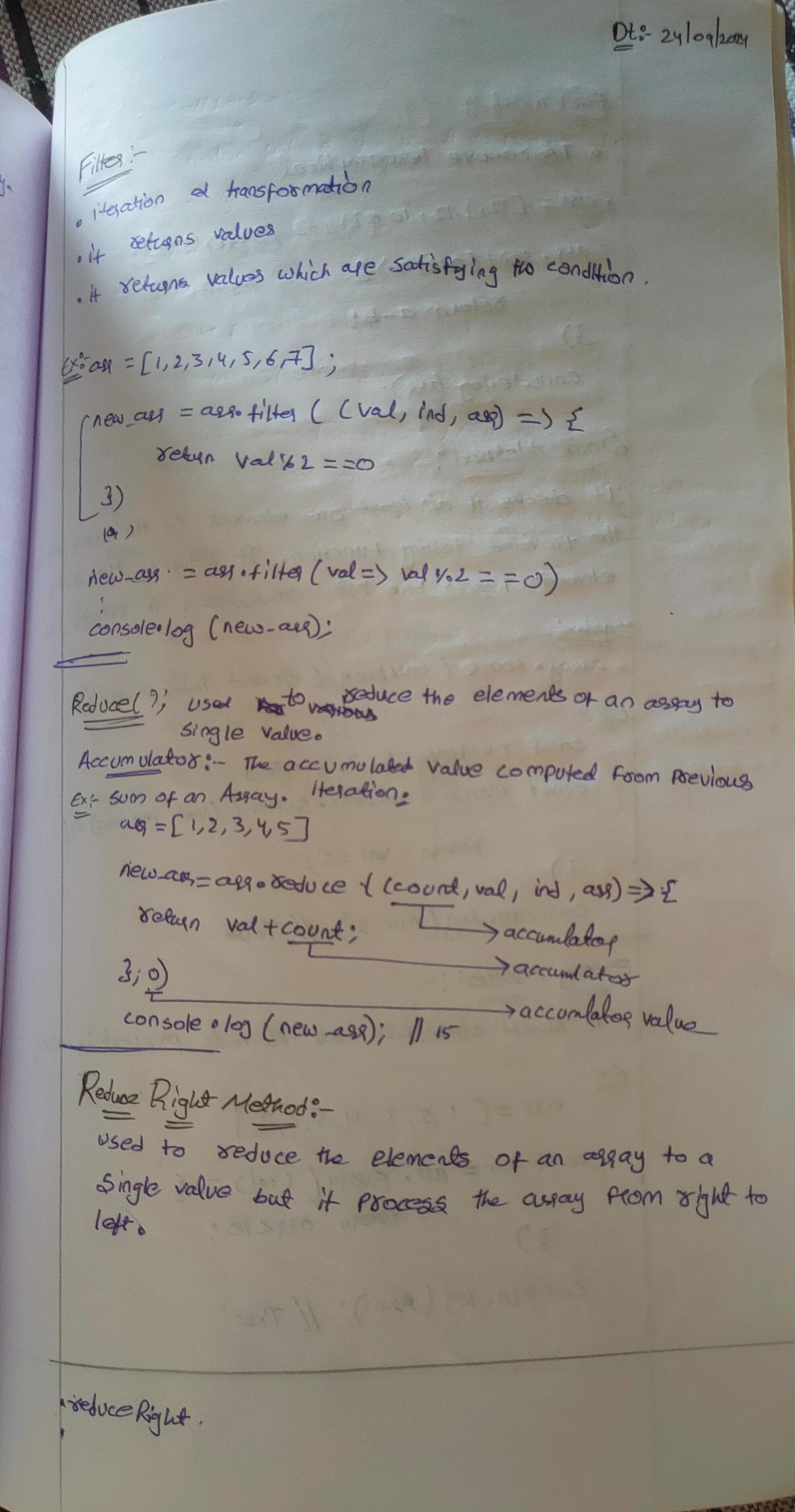
**You can also use arrow functions for more concise syntax:**

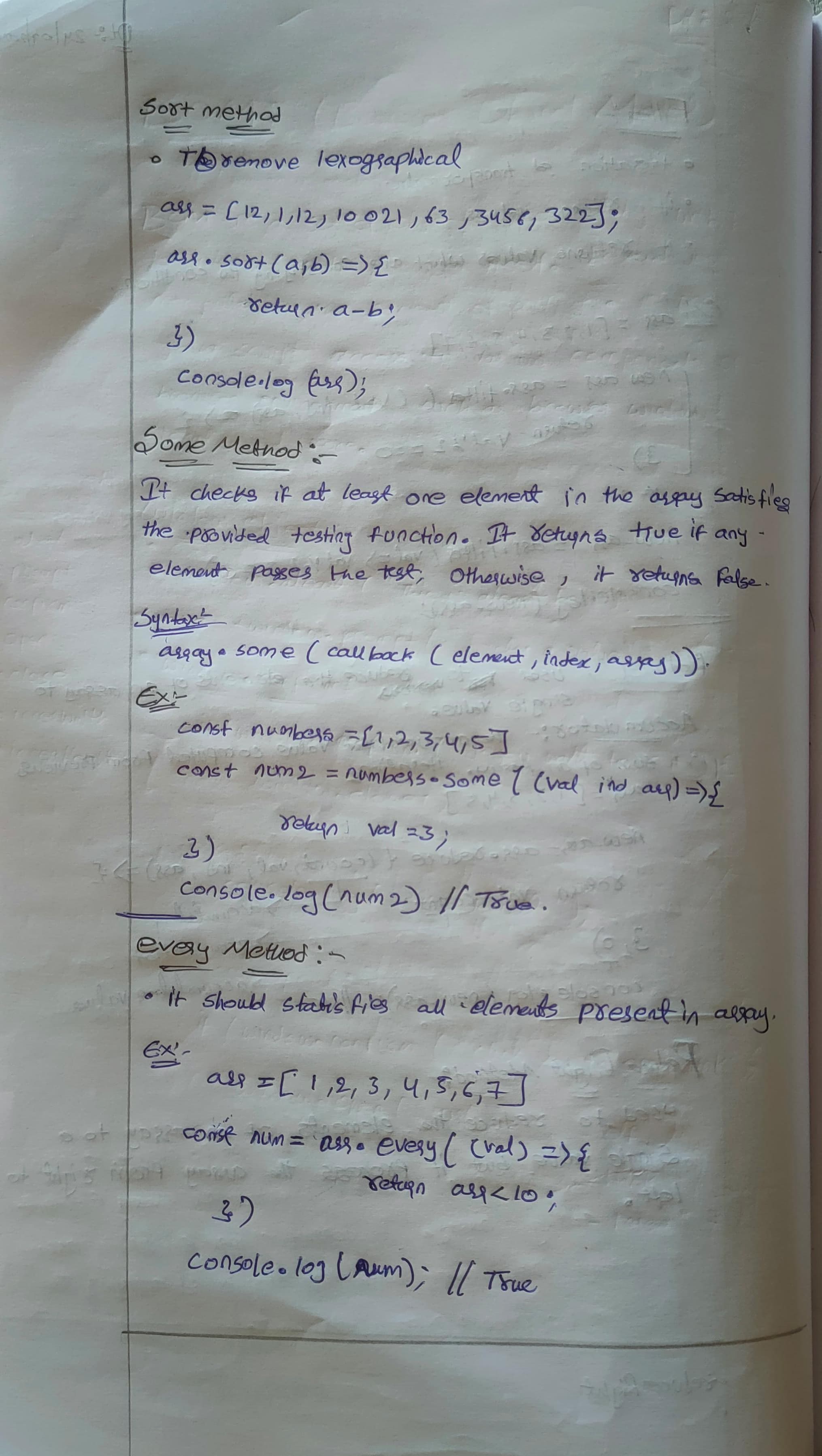
const numbers = [1, 2, 3, 4, 5];

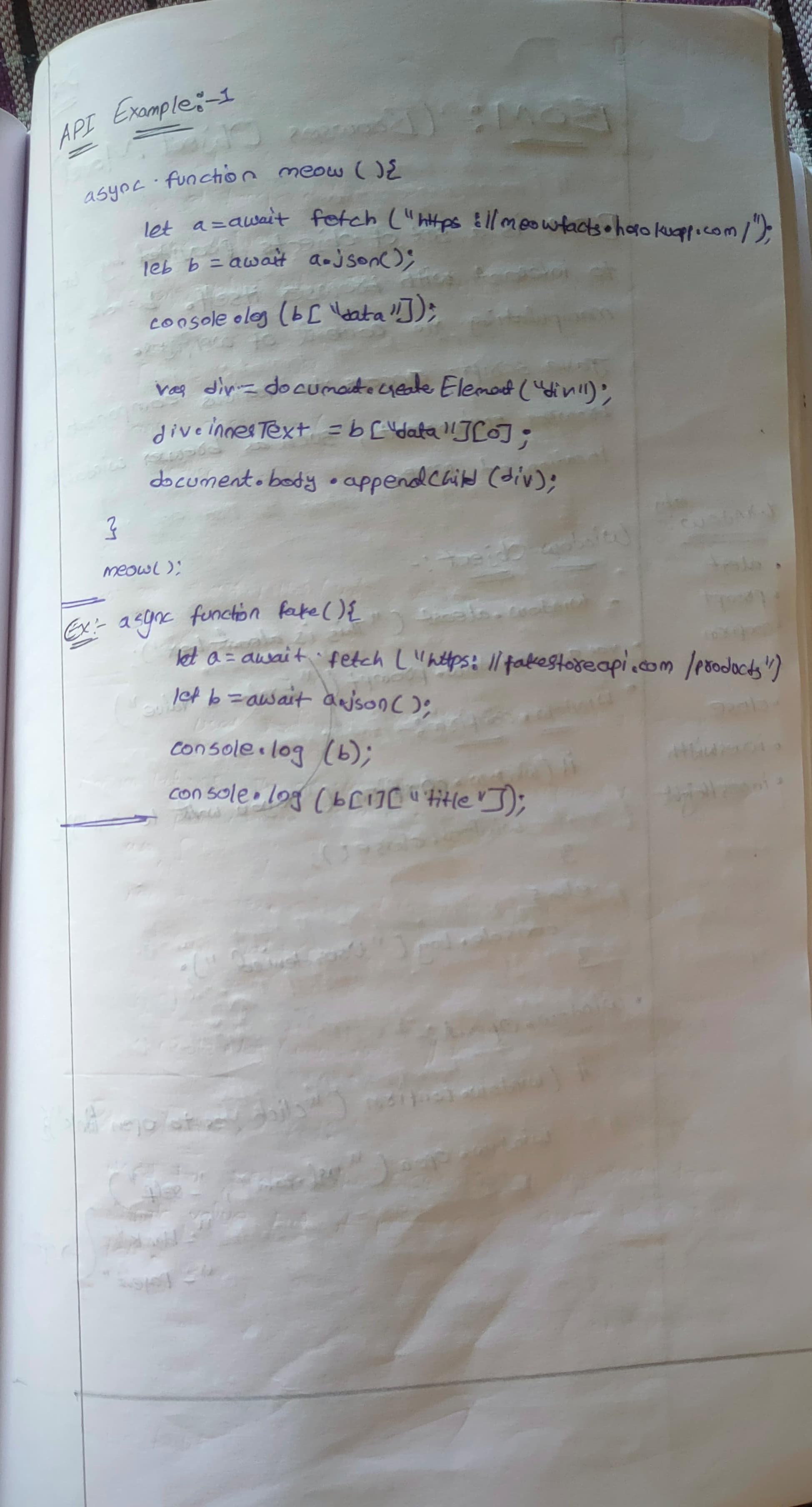
const doubledNumbers = numbers.map(number => number \* 2);

console.log(doubledNumbers); // Output: [2, 4, 6, 8, 10]









**Task:**