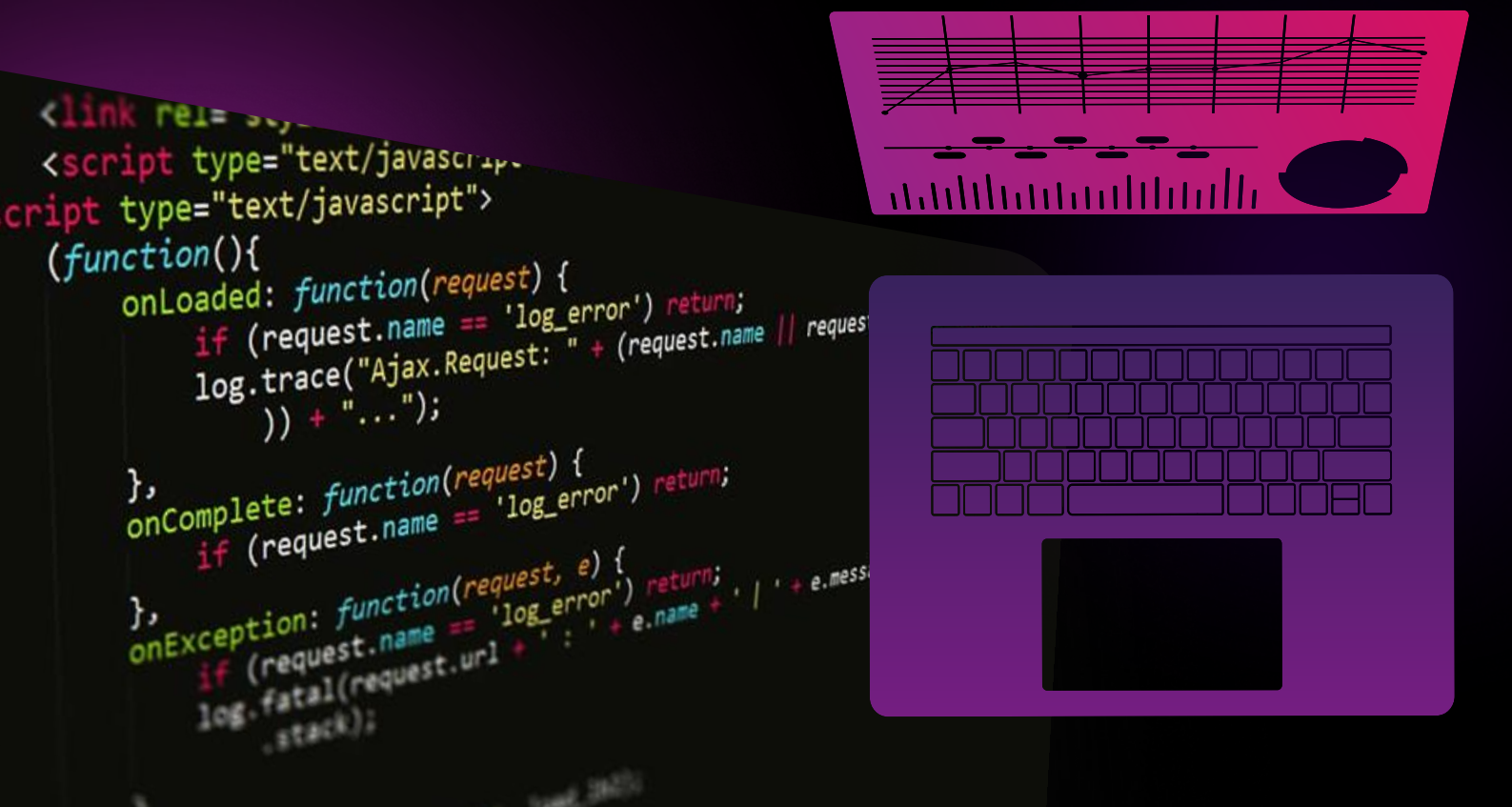
heycoach

HOW TO MASTER JS IN 15 DAYS



DAY 01

🚀 What is Javascript?

JavaScript is a versatile, high-level programming language used for web development to add interactivity and functionality to websites.

🚀 Why one should learn Javascript?

Learning JavaScript is essential for web developers as it allows them to create dynamic and interactive web applications.

🚀 Where it is used?

JavaScript is primarily used for client-side web development, but it can also be used in server-side development (Node.js) and in various application environments.

First 'Hello world!' program in Javascript.

```
console.log("Hello World!");
```

DAY 02

🚀 Understanding basic Syntax

Topics:
Basic syntax and comments in Javascript
JavaScript code is written with a combination of keywords, variables, and operators. Comments are used to explain code and are not executed by the browser.

Example:

- / This is a single-line comment

- /* This is a
multi-line comment */

🚀 Console object

The console object is used for debugging and logging information to the browser's developer console.

Examples:

```
console.log("This is a log message.");
console.error("This is an error message.");
console.warn("This is a warning message.");
```

🚀 Script Tag

Where to place it - The <script> tag is used to include JavaScript code in an HTML document. You can place it either in the <head> or at the end of the <body> element.

Example
(placing it in the <head>):

```
//HTML code
<!DOCTYPE html>
<html>
<head>
   <script>
      // JavaScript code goes here
   </script>
</head>
<body>
   <!-- Rest of the HTML content -->
</body>
</html>
```

🚀 Alert, prompt and confirm

alert(): Displays a dialog box with a message and an "OK" button.

Example:

```
alert("This is an alert!");
```

prompt(): Displays a dialog box with a message and an input field for the user to enter text. It returns the text entered by the user.

Example:

```
const userInput = prompt("Enter your name:");
console.log("User entered: " + userInput);
```

confirm(): Displays a dialog box with a message and "OK" and "Cancel" buttons. It returns true if the user clicks "OK" and false if they click "Cancel."

Example:

```
const userConfirmed
confirm("Are you sure you want to proceed?");
if (userConfirmed) {
console.log("User confirmed.");
}  else {
console.log("User canceled.");
}
```

DAY 03

🚀 Variables and Data Types in Javascript

Topics:
Javascript variables and scope
JavaScript variables are used to store data values. They can be declared using the var, let, or const keywords. The scope of a variable determines where it can be accessed.

Example:

```
// Variable 'x' is declared with global scope
var x = 5;

// Variable 'y' is declared with local (function) scope
function exampleFunction() {
    let y = 10;
}
```

🚀 var vs let vs const (popular interview question

var has function-level scope and can be redeclared within the same scope. let has block-level scope and can be reassigned but not redeclared within the same scope. const has block-level scope and cannot be reassigned or redeclared within the same scope.

Example:

```
var a = 5;
let b = 10;
const c = 15;

a = 8; // Valid for 'var'
b = 12; // Valid for 'let'
// c = 20; // Invalid for 'const', it cannot be reassigned
// let b = 25; // Invalid, redeclaration not allowed for 'let' in the same scope
```

🚀 Data types

JavaScript has several data types, including:

1) Number: for numeric values (e.g., 5, 3.14)
2) String: for text (e.g., "Hello, World!")
3) Boolean: for true/false values (e.g., true)
4) Object: for collections of key-value pairs (e.g., { name: "John", age: 30 })
5) Array: for ordered lists of values (e.g., [1, 2, 3])
6) Null: represents the intentional absence of any object value
7) Undefined: represents an uninitialized variable or missing property

🚀 Primitive and non-primitive data types

Primitive data types in JavaScript are immutable, meaning they cannot be changed after creation. They include:

1) Number
2) String
3) Boolean
4) Null
5) Undefined
6) Symbol (ES6)

Example of a primitive type:

```
let num = 42; // 'num' is a primitive (number) type
```

🚀 **Non-primitive data** types (also known as reference types) include:

1) Object
2) Array
3) Function

Example of a non-primitive type:

```
let person = { name: "Alice", age: 25 }; // 'person' is
```

📚 Further Reading:

- https://www.w3schools.com/js/js_variables.asp
- https://www.w3schools.com/js/js_datatypes.asp
- https://developer.mozilla.org/en-US/docs/Web/JavaScript/Data_structures

YouTube tutorials for hands-on practice:

- https://www.youtube.com/watch?v=pZQeBJsGoDQ&list=PLjwm_8O3suyM61TZY1w5ufD12nRQCtd2N&index=2
- https://www.youtube.com/watch?v=ufHT2WEkkC4&list=PLjwm_8O3suyM61TZY1w5ufD12nRQCtd2N&index=3
- https://www.youtube.com/watch?v=Q4p8vRQX8uY&list=PLuOW_9lll9ahR1blWXxgSlL4y9iQBnLpR&index=2
- https://www.youtube.com/watch?v=Icev9Oxf0WA&list=PLuOW_9lll9ahR1blWXxgSlL4y9iQBnLpR&index=3
- https://www.youtube.com/watch?v=qpU3WIqRz9I&list=PLuOW_9lll9ahR1blWXxgSlL4y9iQBnLpR&index=4

❓ Practice questions:

Q1) Differentiate between null and undefined in JavaScript.

Q2) What is the type of operator in JavaScript, and how is it used to determine data types?

Q3) What will be the output when you use type of operator for an array in Javascript?

Q4) Declare a variable named ‘organization’ and assign the value ‘HeyCoach’ to it.

DAY 04

🚀 Operators in Javascript

Topics:
Types of operators in Javascript.
JavaScript supports various types of operators, including:

Arithmetic Operators: Used for basic mathematical operations.

```
let x = 5 + 3; // Addition
let y = 10 - 4; // Subtraction
```

Comparison Operators: Used to compare values and return a Boolean result.

```
let a = 5 > 3; // Greater than (true)
let b = 10 === 10; // Equal (strictly equal, true)
```

Logical Operators: Used for logical operations.

```
let p = true && false; // Logical AND (false)
let q = true || false; // Logical OR (true)
```

Assignment Operators: Used to assign values to variables.

```
let num = 7; // Assignment
num += 3; // Addition assignment (num is now 10)
```

Unary Operators: Operate on a single operand.

```
let neg = -5; // Unary negation
let not = !true; // Unary NOT (false)
```

🚀 Operator precedence

Operator precedence determines the order in which operations are evaluated in an expression. For example, multiplication is performed before addition. Parentheses can be used to override precedence.

Example:

```
let result = 5 + 3 * 2;
// Multiplication has higher precedence (result is 5 + 6 = 11)
```

Go through the precedence table given in the link below for better understanding.

=, ==, === (popular interview question)

- = is the assignment operator used to assign a value to a variable.
- == is the equality operator used to compare values for equality with type coercion.
- === is the strict equality operator used to compare values for equality without type coercion.o

Example:

```
let num = 5;
let str = "5";

console.log(num == str);
// true (loose equality with type coercion)
 console.log(num === str);
// false (strict equality without type coercion)
```

🚀 Type conversion

Type conversion in JavaScript refers to changing the data type of a value. It can be implicit (automatic) or explicit (manual).

Example of explicit type conversion (using parseInt()):

```
let str = "42";
let num = parseInt(str); // Explicitly convert string to number
```

Coercion (slightly advanced topic as well as a popular interview question)

o Coercion refers to the automatic conversion of values from one data type to another in JavaScript, often during operations like addition or comparison.

Example:

```
let result = 5 + "5";
// Coercion: number to string (result is "55")
```

Understanding these concepts is fundamental to working effectively with JavaScript, and questions related to operators, precedence, equality, type conversion, and coercion are common in interviews and assessments.

📚 Further reading:

- https://www.w3schools.com/js/js_operators.asp
- https://www.w3schools.com/js/js_precedence.asp
- https://www.guru99.com/difference-equality-strict-
- operator-javascript.html#:~
- :text=perform%20any%20conversion.-,KEY%20
- DIFFERENCES,datatype%20
- and%20compares%20two%20values.
- https://www.w3schools.com/js/js_type_conversion.asp
- https://www.freecodecamp.org/news/coercion-and-
- type-conversion-in-javascript/

YouTube tutorials for hands-on practice:

- https://www.youtube.com/watch?v=4UwdF2Ia8rY&lis
- t=PLjw
- m_8O3suyM61TZY1w5ufD12n
- RQCtd2N&index=4
- https://www.youtube.com/watch?v=IsV8JQgSW1s&list
- =&index=6
- https://www.youtube.com/watch?v=wFiVtqe1osM

❓ Practice questions:

Q1) What will be printed to the console for the following code snippet, and why?

```
var x = '5';
var y = 9;
Console.log(x + y);
```

Q2) Explain truthy and false values and provide examples of each. Also illustrate how they are used in case of Boolean coercion in Javascript?

Q3) Write a JavaScript program to convert a specified number into an array of digits.

Q4) Write a JavaScript program to find the area of a triangle where three sides are 5, 6, 7.

DAY 05

🚀 Conditional Statements

Topics:
Learn about if-else statements
if-else statements in JavaScript are used for conditional execution of code. They allow you to specify a block of code to run if a condition is true and another block if the condition is false.

Example:

```
let num = 10;
 if (num > 5) {
    console.log("Number is greater than 5");
}
else {
    console.log("Number is not greater than 5");
}
```

🚀 Switch statements

switch statements provide a way to perform different actions based on different conditions. It's often used when you have multiple possible values to check.

Example:

```
let day = "Monday";

switch (day) {
    case "Monday":
        console.log("It's the start of the workweek.");
        break;
    case "Friday":
        console.log("It's almost the weekend!");
        break;
    default:
        console.log("It's an ordinary day.");
}
```

🚀 Ternary operator (popular interview question)

The ternary operator ? : allows you to write a compact version of an if-else statement in a single line.

Example:

```
let age = 18;

let canVote = (age >= 18) ? "Yes" : "No";

console.log("Can vote: " + canVote);
// "Can vote: Yes"
```

🚀 Loops – for, while, do-while

for loops are used for iterating a specific number of times.

Example

```
for (let i = 0; i < 5; i++) {
    console.log("Iteration " + i);
}
```

while loops repeatedly execute a block of code as long as a condition is true.

```
let count = 0;
while (count < 3) {
    console.log("Count: " + count);
    count++;
}
```

🚀 do-while loops

are similar to while loops, but they ensure that the block of code is executed at least once before checking the condition.

```
let n = 1;
do { console.log("Number: " + n);
    n++;
} while (n <= 3);
```

Break and continue statements

break is used to exit a loop prematurely.

```
for (let i = 0; i < 5; i++) {
    if (i === 3) {
        break;
    }
    console.log("Iteration " + i);
}
```

continue is used to skip the current iteration of a loop and proceed to the next.

```
for (let i = 0; i < 5; i++) {
    if (i === 2) {
        continue;
    }
    console.log("Iteration " + i);
}
```

Further Reading:

- https://www.w3schools.com/js/js_if_else.asp
- https://www.w3schools.com/js/js_switch.asp
- https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Conditional_operator
- https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Loops_and_iteration

YouTube tutorials for hands-on practice:

- https://www.youtube.com/watch?v=wT-1T7Ws5qY&list=PLjwm_8O3suyM61TZY1w5ufD12nRQCtd2N&index=5
- https://www.youtube.com/watch?v=_Xf6g1ZYxiI&list=PLjwm_8O3suyM61TZY1w5ufD12nRQCtd2N&index=6
- https://www.youtube.com/watch?v=s5Lu4QTjeL0&list=PLu0W_9lII9ahR1blWXxgSlL4y9iQBnLpR&index=7
- https://www.youtube.com/watch?v=XKyyM1VWtUE&list=PLu0W_9lII9ahR1blWXxgSlL4y9iQBnLpR&index=9
- https://www.youtube.com/watch?v=drEjyBSu33w&list=PLu0W_9lII9ahR1blWXxgSlL4y9iQBnLpR&index=10

? Practice questions:

Q1) Write a JavaScript program to print all even numbers from 1 to 100 using for loop.

Q2) Write a JavaScript program to add numbers from 1 to 1000 using while loop

.

Q3) Write a JavaScript program to calculate the factorial of a number using any loop.

Q4) Write a JavaScript program to check whether a number is prime or not and print “Yes” if it’s prime, else print “No”.

DAY 06

🚀 Functions in Javascript

Topics:
Basics of Functions
Functions in JavaScript are blocks of reusable code that perform a specific task. They allow you to encapsulate logic and execute it by calling the function's name.

Example:

```
function greet(name) {
    console.log("Hello, " + name + "!");
}
greet("Alice"); // Output: "Hello, Alice!"
```

🚀 Functions in Javascript

Arguments vs parameters

Parameters: These are the named variables defined in a function's declaration. They act as placeholders for values that will be provided when the function is called.

Example:

```
function add(a, b) { // 'a' and 'b' are parameters
    return a + b;
}
```

Arguments: These are the actual values passed into a function when it is invoked. They match the parameters by position.

Example:

```
let result = add(3, 5); // 3 and 5 are arguments
```

🚀 Pass by value vs pass by reference

Pass by Value:
Primitive data types (e.g., numbers, strings) are passed by value. This means that a copy of the value is passed to the function, and changes within the function do not affect the original value.

Example:

```
function modifyValue(x) {
    x = 10;
}
let num = 5;
modifyValue(num);

console.log(num); // Output: 5 (unchanged)
}
```

🚀 Pass by Reference:

Objects and arrays are passed by reference. This means that a reference to the original object is passed to the function, allowing changes to affect the original object.

Example:

```
function modifyArray(arr) {
    arr.push(4);
}
let myArray = [1, 2, 3];
modifyArray(myArray);
console.log(myArray); // Output: [1, 2, 3, 4] (modified)
```

🚀 Recursive functions

A recursive function is a function that calls itself in order to solve a problem. It typically consists of a base case and a recursive case.

Example (calculating factorial):

```
function factorial(n) {
    if (n <= 1) {
        return 1; // Base case
    } else {
        return n * factorial(n - 1); // Recursive case
    }
}
```

🚀 Callback functions

Callback functions are functions that are passed as arguments to other functions and are executed later, often after some asynchronous operation completes.

Example (using setTimeout with a callback):

```
function greet(name, callback) {
    setTimeout(function () {
        console.log("Hello, " + name + "!");
        callback();
    }, 1000);
}

function sayGoodbye() {
    console.log("Goodbye!");
}

greet("Alice", sayGoodbye);
```

📚 Further Reading:

- https://www.w3schools.com/js/js_functions.asp
- https://www.freecodecamp.org/news/what-is-the-difference-between-parameters-and-arguments-in-javascript/#:~:text=A%20parameter%20is%20one%20of,to%20x%20and%20y%2C%20respectively.
- https://medium.com/nodesimplified/javascript-pass-by-value-and-pass-by-reference-in-javascript-fcf10305aa9c
- https://www.freecodecamp.org/news/recursion-in-javascript/
- https://www.w3schools.com/js/js_callback.asp

YouTube tutorials for hands-on practice:

- https://www.youtube.com/watch?v=a_gwOwkbhZ0&list=PLu0W_9lII9ahR1blWXxgSlL4y9iQBnLpR&index=11
- https://www.youtube.com/watch?v=GNLw19RMyZc&list=PLjwm_8O3suyM61TZY1w5ufD12nRQCtd2N&index=7
- https://www.youtube.com/watch?v=_FJvnslu1co&list=PLjwm_8O3suyM61TZY1w5ufD12nRQCtd2N&index=8

❓ Practice questions:

Q1) Write a JavaScript function called greet that takes a person's name as an argument and returns a greeting message (e.g., "Hello, John!").

Q2) Write a Javascript program to calculate the power of an number (a^b) using recursion.

Q3) Compare and contrast recursion and iteration in JavaScript. In what situations would you prefer to use recursion over iteration, and vice versa?

Q4) Write a Javascript program to implement a calculator. It should have functions for various arithmetic operations and the choices can be provided using a switch statement.

DAY 07

🚀 Arrays, Strings and Objects

Topics:

Arrays and array methods

Arrays: Arrays are ordered collections of values, which can be of any data type. They are often used to store lists of items.

Example:

```
let colors = ["red", "green", "blue"];
```

🚀 Array Methods:

JavaScript provides numerous array methods for manipulating and working with arrays.

Examples include:

- **push()**: Adds elements to the end of an array.
- **pop()**: Removes the last element from an array.
- **forEach()**: Iterates over each element in an array.
- **filter()**: Creates a new array with elements that meet a condition.
- **map()**: Creates a new array by applying a function to each element.

Example:

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

numbers.push(6); // [1, 2, 3, 4, 5, 6]

let doubled = numbers.map(function (num) {
    return num * 2;
}); // [2, 4, 6, 8, 10, 12]
```

🚀 String Methods:

JavaScript provides various string methods for manipulating and working with strings. Examples include:

- **toUpperCase():** Converts a string to uppercase.
- **substring():** Extracts a portion of a string.
- **indexOf():** Returns the position of a substring within a string.
- **split():** Splits a string into an array of substrings based on a delimiter.

Example:

```
let sentence = "This is a sample sentence.";

let upperCaseSentence = sentence.toUpperCase(); // "THIS IS A SAMPLE SENTENCE."

let words = sentence.split(" "); // ["This", "is", "a", "sample", "sentence."]
```

🚀Javascript Objects

Objects:
Objects in JavaScript are collections of key-value pairs. They are versatile data structures used to represent complex data and are extensively used in web development, often to represent things like users, products, or other entities.

Example:

```
let person = {
    firstName: "John",
    lastName: "Doe",
    age: 30
};
```

🚀 Working with Objects:

You can access object properties using dot notation (object.property) or bracket notation (object["property"]), and you can add, update, or delete properties dynamically.

Example:

```
person.age = 31; // Update property
person["gender"] = "Male"; // Add property
delete person.lastName; // Delete property
```

📚 Further Reading:

- https://www.w3schools.com/js/js_arrays.asp
- https://www.w3schools.com/js/js_array_methods.asp
- https://www.w3schools.com/js/js_strings.asp
- https://www.w3schools.com/js/js_string_methods.asp
- https://www.w3schools.com/js/js_objects.asp

YouTube tutorials for hands-on practice:

- https://www.youtube.com/watch?v=a_Bz5ciBHQ0&list=PLu0W_9lII9ahR1blWXxgSlL4y9iQBnLpR&index=16
- https://www.youtube.com/watch?v=BLIrBThPTXc&list=PLu0W_9lII9ahR1blWXxgSlL4y9iQBnLpR&index=17
- https://www.youtube.com/watch?v=Yafji9PB1lM&list=PLu0W_9lII9ahR1blWXxgSlL4y9iQBnLpR&index=13
- https://www.youtube.com/watch?v=8yg4RUEnalk&list=PLu0W_9lII9ahR1blWXxgSlL4y9iQBnLpR&index=14
- https://www.youtube.com/watch?v=TCXQsQL5kPo&list=PLjwm_8O3suyM61TZY1w5ufD12nRQCtd2N&index=9
- https://www.youtube.com/watch?v=h13yzap50z4&list=PLjwm_8O3suyM61TZY1w5ufD12nRQCtd2N&index=10

? Practice questions:

Q1) Write a Javascript program to declare an empty array called fruits. Add three different fruit names to the array using various array methods.

Q2) Write a JavaScript program to sort a given array using Merge Sort algorithm.

Q3) Write a JavaScript program that searches for a specific word within a given string and returns the count of its occurrences using the index Of method.

Q4) Write a Javascript program to check whether a given string is a palindrome or not.

Q5) Declare an array of objects, where each object is a person with properties for name, age, and city. Assign values to these properties. Use for Each method to traverse each person in the array and then print his/her details.

DAY 08

🚀Javascript Events

Topics:

Understanding Events in Javascript

Events in

- JavaScript refer to interactions or occurrences that take place in the browser, such as user actions (like clicks or keyboard input) or system events (like page loading or resizing).

- JavaScript allows you to respond to these events by executing code when they occur. Events play a vital role in creating interactive web applications.

.

🚀 Mouse Events:

- **click:** Triggered when a mouse button is clicked.
- **mousedown**: Fired when a mouse button is pressed down.
- **mouseup:** Fired when a mouse button is released.
- **mouseover and mouseout:** Triggered when the mouse enters or exits an element.

🚀Keyboard Events:

- **keydown**: Fired when a key is pressed down.
- **keyup**: Triggered when a key is released.
- **keypress**: Fired when a key is pressed and released.

🚀Form Events:

- **submit:** Triggered when a form is submitted.
- **change**: Fired when the value of an input element changes.
- **focus and blur:** Triggered when an element gains or loses focus.

🚀 Window and Document Events:

- **load:** Fired when a web page or image has finished loading.
- **unload:** Triggered when the page is about to be unloaded (rarely used).
- **resize:** Fired when the browser window is resized.

Custom Events:
Developers can create and dispatch their custom events for specific use cases.

🚀 Window and Document Events:

- **load:** Fired when a web page or image has finished loading.
- **unload:** Triggered when the page is about to be unloaded (rarely used).
- **resize:** Fired when the browser window is resized.

Custom Events:

Developers can create and dispatch their custom events for specific use cases.

Event listeners

Event listeners are functions in JavaScript that "listen" for specific events to occur. They are attached to HTML elements and execute a specified function when the associated event occurs.

Example:

```
let button = document.getElementById("myButton");

button.addEventListener("click", function() {
    alert("Button clicked!");
});
```

🚀 Event handlers

Event handlers are attributes in HTML elements that specify the JavaScript code to execute when an event occurs. While event listeners provide more flexibility and are preferred in modern web development, event handlers are still used in simpler cases.

Example (using an event handler in HTML):

```
<button id="myButton" onclick="myFunction()">Click Me</
button>

<script>
function myFunction() {
    alert("Button clicked!");
}
</script>
```

In this example, the onclick attribute in the HTML button element directly specifies the JavaScript function to run when the button is clicked.

📚 Further Reading:

- https://www.w3schools.com/js/js_events.asp
- https://www.freecodecamp.org/news/javascript-events-explained-in-simple-english/
- https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Building_blocks/Events

YouTube tutorials for hands-on practice:

- https://www.youtube.com/watch?v=R7mu7nKFc7w
- https://www.youtube.com/watch?v=Y3f_ih-2jGk&list=PLuOW_9lII9ahR1blWXxgSlL4y9iQBnLpR&index=47
- https://www.youtube.com/watch?v=rFq0HVOdDo4
- https://www.youtube.com/watch?v=XF1_MlZ5l6M

❓ Practice questions:

Q1) How do you attach an event listener to an HTML element in JavaScript? Provide an example of attaching a click event listener to a button element.

Q2) What are inline event handlers in HTML? Explain their advantages and disadvantages compared to using external event listeners.

Q3) Describe the event object in JavaScript. What information does it typically contain, and how can you access it within an event handler?

Q4) Write a JavaScript program that displays text “TEXT-GROWING” with increasing font size in the interval of 100ms in RED COLOR, when the font size reaches 50pt it displays “TEXT-SHRINKING” in BLUE color. Then the font size decreases to 5pt.

DAY 09

🚀 DOM (Document Object Model)

Topics:

Introduction to DOM
The Document Object Model (DOM) is a programming interface for web documents. It represents the structure of an HTML or XML document as a tree-like structure in which each node represents part of the document. The DOM provides a way for developers to interact with and manipulate web pages dynamically using programming languages like JavaScript.

DOM Navigation

DOM navigation involves traversing and accessing elements within the DOM tree. Common navigation methods include:

- **getElementById():** Retrieves an element by its unique id attribute.
- **getElementsByClassName()**: Returns a collection of elements with the specified class name.
- **getElementsByTagName():** Retrieves a collection of elements with the specified tag name.
- **querySelector()**: Selects the first element that matches a CSS selector.
- **querySelectorAll():** Selects all elements that match a CSS selector.

🚀 Parent and child relationships:

Example:

```
let myElement = document.getElementById("myId");
let elementsWithClass =
document.getElementsByClassName("myClass");
let paragraphElements =
document.getElementsByTagName("p");
let firstDiv = document.querySelector("div");
let allDivs = document.querySelectorAll("div");
```

🚀 DOM Properties and Methods

The DOM provides various properties and methods to interact with elements. Common properties include innerHTML, textContent, value, and src, which allow you to read or modify element content and attributes. Methods include setAttribute(), appendChild(), removeChild(), and addEventListener() for manipulating elements and responding to events.

Example (changing text content and setting an attribute):

```
let myElement = document.getElementById("myId");
myElement.textContent = "New content";
myElement.setAttribute("src", "newimage.jpg");
```

DOM Elements

DOM elements represent the individual parts of an HTML document, such as headings, paragraphs, buttons, and images. Elements can be selected, created, modified, or deleted using JavaScript.

Example
(creating and appending a new element):

```
let newParagraph = document.createElement("p");
newParagraph.textContent = "This is a new paragraph.";
document.body.appendChild(newParagraph);
```

📚 Further Reading:

- https://www.w3schools.com/js/js_htmldom.asp
- https://www.w3schools.com/js/js_htmldom_navigation.asp
- https://www.w3schools.com/js/js_htmldom_methods.asp
- https://www.w3schools.com/js/js_htmldom_elements.asp

YouTube tutorials for hands-on practice:

- https://www.youtube.com/watch?v=DcjNkHtDj8A&list=PLu71SKxNbfoBuX3f4EOACle2y-tRC5Q37&index=31
- https://www.youtube.com/watch?v=Rhj6KWFw7AA&list=PLuOW_9lII9ahR1blWXxgSlL4y9iQBnLpR&index=31
- https://www.youtube.com/watch?v=78VMefnkMcM&list=PLuOW_9lII9ahR1blWXxgSlL4y9iQBnLpR&index=34
- https://www.youtube.com/watch?v=M8AUk6gDe2c&list=PLuOW_9lII9ahR1blWXxgSlL4y9iQBnLpR&index=

? Practice questions:

Q1) Differentiate between HTML and the DOM. How does the DOM relate to the structure of an HTML document?

Q2) Compare and contrast get Element Byld and query Selector methods in terms of DOM element selection. When would you use one over the other?

Q3) How can you change the text content of an HTML element using JavaScript? Provide code examples for both getting and setting text content.

Q4) Write a Javascript program to add a click event listener to a button element.

Q5) Write a Javascript program to select all <p> elements on a webpage.

DAY 10

🚀 DOM (Document Object Model)

Topics:

DOM Nodes, Node lists, Collections
DOM Nodes: DOM nodes are individual elements or parts of a web page represented as objects in the Document Object Model (DOM). Each element, attribute, or piece of text in an HTML document is a node. Nodes can be elements (e.g.,), attributes (e.g., class="example"), or text content.

o Node Lists: Node lists are collections of DOM nodes, often returned by methods like getElementsByTagName, getElementsByClassName, or querySelectorAll. They are similar to arrays but lack some of the array methods.

Example:

```
let paragraphs = document.getElementsByTagName("p"); // Returns a node list of <p> elements.
```

🚀 Collections:

Collections are similar to node lists but are specific to certain HTML elements, such as HTMLCollection for forms or tables. They also lack some array methods.

Example (accessing a form collection):
```
let forms = document.forms; // Returns an HTMLCollection of forms.
```

Creating New Elements in DOM
You can create new DOM elements dynamically using JavaScript's createElement method and then append them to the DOM using methods like appendChild or insertBefore.

Example (creating and appending a new paragraph element):

```
let newParagraph = document.createElement("p");
newParagraph.textContent = "This is a new paragraph.";
document.body.appendChild(newParagraph);
```

Editing and Removing Elements from DOM

Editing Elements:
You can modify the content, attributes, or style properties of existing DOM elements using various properties and methods. For instance, you can use textContent to change the text content or setAttribute to change an attribute.

Example (changing the text content and an attribute):

```
let myElement = document.getElementById("myId");
myElement.textContent = "New content";
myElement.setAttribute("src", "newimage.jpg");
```

Removing Elements:
To remove elements from the DOM, use methods like removeChild or remove.

Example (removing an element):

```
let parentElement = document.getElementById("parent");
let childElement = document.getElementById("child");
parentElement.removeChild(childElement);
```

📚 Further Reading:

- https://www.w3schools.com/js/js_htmldom_nodes.asp
- https://www.w3schools.com/js/js_htmldom_collections.asp
- https://www.w3schools.com/js/js_htmldom_nodelist.asp
- https://www.w3schools.com/jsref/met_document_createelement.asp
- https://www.w3schools.com/jsref/met_element_remove.asp

YouTube tutorials for hands-on practice:

- https://www.youtube.com/watch?v=Ab6K57WjWTE&list=PLu71SKxNbfoBuX3f4EOACle2y-tRC5Q37&index=32
- https://www.youtube.com/watch?v=xAvTgCsCHLs&list=PLu71SKxNbfoBuX3f4EOACle2y-tRC5Q37&index=33
- https://www.youtube.com/watch?v=VQIY-X_eeTE&list=PLu71SKxNbfoBuX3f4EOACle2y-tRC5Q37&index=34

❓Practice questions:

Q1) Describe how to create a new HTML element in the DOM using JavaScript. Provide an example of creating a new <li> element within an unordered list.

Q2) Explain how to set attributes for newly created DOM elements. Provide an example of setting the src attribute for an <img> element.

Q3) Describe how to change the value of an existing attribute of a DOM element. Provide an example of changing the href attribute of an anchor (<a>) element.

Q4) Explain how to remove an element from the DOM tree using JavaScript. Provide an example of removing a list item (<li>) from an ordered list.

Q5) How can you replace an existing DOM element with another element? Give an example of replacing a with a .

🚀 OOPS (Object Oriented Programming)

Topics:
Intro to Object Oriented Programming

Object-Oriented Programming (OOP)

- OOP is a programming paradigm that revolves around the concept of objects. In OOP, objects are instances of classes, which are templates or blueprints for creating objects.
- OOP promotes the organization of code into reusable and modular components called classes, and it focuses on concepts such as encapsulation, inheritance, abstraction and polymorphism.

🚀 Creating Classes and Objects

Classes:
In OOP, classes are used to define the structure and behavior of objects. A class serves as a blueprint, specifying properties (attributes) and methods (functions) that objects of that class will have.
Example of defining a class in JavaScript (ES6 syntax):

```
class Person {
    constructor(name, age) {
        this.name = name;
        this.age = age;
    }

    greet() {
        console.log(`Hello, my name is ${this.name} and I am ${this.age} years old.`);
    }
}
```

🚀Objects:

Objects are instances of classes. They are created based on the blueprint defined by a class and can have their own unique property values while inheriting the methods defined in the class.

Example of creating objects from a class:

```
let person1 = new Person("Alice", 30);
let person2 = new Person("Bob", 25);

person1.greet(); // Output: "Hello, my name is Alice and I am 30 years old."
person2.greet(); // Output: "Hello, my name is Bob and I am 25 years old."
```

🚀 Prototypes

Prototypes in JavaScript are mechanisms that allow objects to inherit properties and methods from other objects. Every JavaScript object has an associated prototype, which is an object from which it inherits properties and methods.

Example of adding a method to an object's prototype:

```
function Person(name, age) {
    this.name = name;
    this.age = age;
}

Person.prototype.greet = function() {
    console.log(`Hello, my name is ${this.name} and I am ${this.age} years old.`);
};

let person1 = new Person("Alice", 30);
person1.greet(); // Output: "Hello, my name is Alice and I am 30 years old."
```

🚀 Constructor method

A constructor method is a special method in a class that is automatically called when an object is created from that class. It is used to initialize the object's properties.

Example of a constructor method:

```
class Person {
    constructor(name, age) {
        this.name = name;
        this.age = age;
    }
}

let person = new Person("Alice", 30);
```

🚀 Static methods

Static methods in a class are methods that belong to the class itself rather than instances of the class. They are typically used for utility functions or operations that do not require access to instance-specific data.

Example of a static method:

```
class MathUtils {
    static add(x, y) {
        return x + y;
    }
}

let sum = MathUtils.add(5, 3); // Call the static method directly on the class
console.log(sum); // Output: 8
```

📚 Further Reading:

- https://www.w3schools.in/javascript/object-oriented
- https://www.w3schools.com/js/js_object_prototypes.asp
- https://www.w3schools.com/js/js_object_constructors.asp
- https://www.w3schools.com/js/js_class_static.asp
- https://www.freecodecamp.org/news/object-oriented-javascript-for-beginners/

YouTube tutorials for hands-on practice:

- https://www.youtube.com/watch?v=fe6L8bNC_Yw&list=PLuOW_9lII9ahR1blWXxgSlL4y9iQBnLpR&index=74
- https://www.youtube.com/watch?v=eDxrLEQbLvO&list=PLuOW_9lII9ahR1blWXxgSlL4y9iQBnLpR&index=75
- https://www.youtube.com/watch?v=7RpdfkSyJfU&list=PLuOW_9lII9ahR1blWXxgSlL4y9iQBnLpR&index=76
- https://www.youtube.com/watch?v=0E2akQ5E-5Y&list=PLuOW_9lII9ahR1blWXxgSlL4y9iQBnLpR&index=77
- https://www.youtube.com/watch?v=__l6ZOPaijs&list=PLuOW_9lII9ahR1blWXxgSlL4y9iQBnLpR&index=81

❓ Practice questions:

Q1) How do you create a class in JavaScript? Provide an example of defining a class for a "Car" with properties like company, model, and year, fuel and methods like start(), applyBrakes(), blowHorn().

Q2) Explain the difference between a class and an object in JavaScript. How do you create instances (objects) of a class?

Q3) Explain the prototype chain in JavaScript and how it allows objects to inherit properties and methods from their prototypes.

Q4) Provide an example of a constructor method for a "Person" class that takes parameters for name and age and sets them as properties of the object.

Q5) Give an example of a static method in a class for calculating the area of a circle without creating an instance of the class.

DAY 12

🚀 OOPS (Object Oriented Programming)

Topics:
Encapsulation is one of the four fundamental principles of object-oriented programming (OOP). It involves bundling data (attributes or properties) and the methods (functions) that operate on that data into a single unit called a class. Encapsulation helps to protect the internal state of an object by providing controlled access to its properties and methods.

Example of encapsulation in JavaScript:

```
class Person {
    constructor(name, age) {
        this.name = name; // Encapsulated property
        this.age = age;   // Encapsulated property
    }

    // Encapsulated method
    greet() {
        console.log(`Hello, my name is ${this.name} and I am ${this.age} years old.`);
    }
}

let person = new Person("Alice", 30);
person.greet(); // Accessing the encapsulated method
```

🚀 Inheritance

Inheritance is another fundamental OOP concept that allows a class to inherit properties and methods from another class. It promotes code reuse and the creation of a hierarchy of classes.

Example of inheritance in JavaScript:

```
class Animal {
    constructor(name) {
        this.name = name;
    }speak() {
        console.log(`${this.name} makes a sound.`);
    }
}

class Dog extends Animal { // Dog inherits from Animal
    constructor(name, breed) {
        super(name); // Call the parent class constructor
        this.breed = breed;
    }bark() {
        console.log(`${this.name} barks loudly.`);
    } }
 let dog = new Dog("Buddy", "Golden Retriever");
dog.speak(); // Accessing the inherited method from Animal
dog.bark();  // Accessing the method specific to Dog
```

🚀 Polymorphism

Polymorphism allows objects of different classes to be treated as objects of a common superclass. It enables flexibility and dynamic behavior based on the specific type of object.

Example of polymorphism in JavaScript (using inheritance):

javascript
Copy code

```
class Shape {
    area() {
        return 0;
    }
}

class Circle extends Shape {
    constructor(radius) {
        super();
        this.radius = radius;
    }
```

```
🚀    area() {
        return Math.PI * this.radius ** 2;
    }
}

class Rectangle extends Shape {
    constructor(width, height) {
        super();
        this.width = width;
        this.height = height;
    }

    area() {
        return this.width * this.height;
    }
}

let shapes = [new Circle(5), new Rectangle(4, 6)];

shapes.forEach(shape => {
    console.log(`Area: ${shape.area()}`);
});
```

🚀 Abstraction

Abstraction is the process of simplifying complex reality by modeling classes based on the essential properties and behaviors an object should have. It hides the irrelevant details and focuses on what's necessary for the specific problem being solved.

Example of abstraction in JavaScript (using a simplified Vehicle class):

```
class Vehicle {
    constructor(make, model) {
        this.make = make;   // Essential property
        this.model = model; // Essential property
    }

    startEngine() {
        console.log("Engine started."); // Essential behavior
    }

    // Other methods and properties can be added for
specific types of vehicles
}
```

🚀 Getters and Setters

Getters and setters are methods in a class used to control access to the properties of objects. They allow you to get (retrieve) and set (update) the values of properties with additional logic if needed.

Example of getters and setters in JavaScript:

```
class Circle {
    constructor(radius) {
        this._radius = radius; // Private property (convention using an underscore)
    }

    get radius() {
        return this._radius;
    }

    set radius(value) {
        if (value < 0) {
            console.log("Radius cannot be negative.");
            return;
        }
        this._radius = value;
    }
```

```
    area() {
        return Math.PI * this._radius ** 2;
    }
}

let circle = new Circle(5);
console.log(circle.radius); // Accessing the getter
circle.radius = 7;          // Accessing the setter
console.log(circle.area());  // Calculating the area
```

Getters and setters provide control and validation over property access, ensuring data integrity and facilitating encapsulation.

📚 Further Reading:

- https://viktor-kukurba.medium.com/object-oriented-programming-in-javascript-2-inheritance-447368f57a26
- https://viktor-kukurba.medium.com/object-oriented-programming-in-javascript-4-encapsulation-4f9165cd26f9
- https://viktor-kukurba.medium.com/object-oriented-programming-in-javascript-3-polymorphism-fb564c9f1ce8
- https://viktor-kukurba.medium.com/object-oriented-programming-in-javascript-1-abstraction-c47307c469d1
- https://www.honeybadger.io/blog/javascript-oop/

▶ YouTube tutorials for hands-on practice:

- https://www.youtube.com/watch?v=9loYq8W8rsg
- https://www.youtube.com/watch?v=NsF4BrRfOGM&list=PLu0W_9lII9ahR1blWXxgSlL4y9iQBnLpR&index=82
- https://www.youtube.com/watch?v=HXDKho8S2yM
- https://www.youtube.com/watch?v=YkhLw5tYR6c
- https://www.youtube.com/watch?v=jM0WcyQWMSM

❓ Practice questions:

Q1) What are the four pillars of OOPS? Explain each with a suitable example.

Q2) Explain the concept of access modifiers (private, protected, public) in relation to encapsulation. How can they be emulated in JavaScript?

Q3) Provide an example of inheritance in JavaScript, where a subclass inherits properties and methods from a superclass.

Q4) Explain the difference between compile-time (static) and runtime (dynamic) polymorphism. Provide examples for each in JavaScript.

Q5) Give an example of abstraction in JavaScript, where you define an abstract class or method and explain how it can be implemented in derived classes.

.....

DAY 13

🚀 Error Handling and JSON

Topics:

Errors in Javascript
Errors in JavaScript represent unexpected or undesirable behavior in a program. Common error types include SyntaxError (errors in code syntax), ReferenceError (trying to access an undefined variable), TypeError (incompatible data types or operations), and more. Errors can occur during code execution and need to be handled appropriately.

Example of a ReferenceError:

```
console.log(undefinedVariable); // ReferenceError: undefinedVariable is not defined
```

🚀 Handling Errors –

Throw, Try, Catch, Finally
Throw: The throw statement is used to throw an exception (error) when a specific condition is met.

Example:

```
function divide(a, b) {
    if (b === 0) {
        throw new Error("Division by zero is not allowed.");
    }
    return a / b;
}
```

Try-Catch: The try block is used to enclose the code that might throw an exception, and the catch block is used to handle the exception when it occurs.

Example:

```
try {
    let result = divide(10, 0);
    console.log(result);
} catch (error) {
    console.error("An error occurred:", error.message);
}
```

🚀 Finally:

The finally block is used to specify code that will be executed regardless of whether an exception was thrown or not. It is often used for cleanup operations.

Example:

```
try {
    // Code that might throw an exception
} catch (error) {
    // Handle the exception
} finally {
    // Cleanup code (always executed)
}
```

🚀 JSON (Javascript Object Notation)

JSON, which stands for JavaScript Object Notation, is a lightweight data interchange format used to exchange data between a server and a client, or between different parts of an application. JSON is easy for humans to read and write and is easy for machines to parse and generate. It is often used in web development for data serialization and communication.

Example of JSON data:

```
{
    "name": "John Doe",
    "age": 30,
    "email": "john@example.com"
}
```

📚 Further Reading:

- https://www.tutorials for hands-on practicepoint.com/javascript/javascript_error_handling.htm
- https://www.w3schools.com/js/js_errors.asp
- https://www.w3schools.com/jsref/jsref_obj_error.asp
- https://www.w3schools.com/js/js_json.asp
- https://www.w3schools.com/js/js_json_xml.asp

YouTube tutorials for hands-on practice:

- https://www.youtube.com/watch?v=fRv2ng_srrM
- https://www.youtube.com/watch?v=WRNBQCl_cPU
- https://www.youtube.com/watch?v=uOQBGKfldIg
- https://www.youtube.com/watch?v=whNFPBEI-wM

❓ Practice questions:

Q1) What are runtime errors in JavaScript, and how do they differ from syntax errors?

Q2) What is the purpose of the throw statement in JavaScript, and when is it typically used in error handling?

Q3) Explain the properties and methods available in the JavaScript Error object. How can you create custom error objects in JavaScript?

Q4) Create a custom error class called ValidationError that extends the Error object. Provide an example of how to throw and catch instances of this custom error.

Q5) Compare JSON and XML as data interchange formats. Highlight their key differences and explain when it's preferable to use one over the other.

🚀 Important Modern JS features

(Extensively used in React and in general as well nowadays)

Topics:

Arrow Functions

Arrow functions are a concise way to write functions in JavaScript, introduced in ES6. They provide a shorter syntax compared to traditional function expressions and have a lexically scoped this, meaning they inherit the this value from the surrounding code.

Example of arrow function:

```
const add = (a, b) => a + b;
console.log(add(2, 3)); // Output: 5
```

- Rest and Spread operators

🚀 Rest and Spread operators

Rest Operator (...):
The rest operator allows you to represent an indefinite number of arguments as an array within a function.

Example:

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

console.log(sum(2, 3, 4)); // Output: 9
```

Spread Operator (...):

The spread operator is used to split an array or object into individual elements or properties.

Example (spreading an array):

```
const arr1 = [1, 2, 3];
const arr2 = [4, 5, ...arr1, 6];
console.log(arr2); // Output: [4, 5, 1, 2, 3, 6]
```

🚀 Array and Object destructuring

Array Destructuring: Array destructuring allows you to extract values from an array into individual variables.

Example:

```
const numbers = [1, 2, 3];
const [a, b, c] = numbers;
console.log(a, b, c); // Output: 1 2 3
```

Object Destructuring:

Object destructuring lets you extract properties from an object and assign them to variables with matching names.

Example:

```
const person = { name: "John", age: 30 };
const { name, age } = person;
console.log(name, age); // Output: John 30
```

🚀 Map, Filter and Reduce methods

Map: The map() method creates a new array by applying a function to each element in an existing array.

Example:

```
const numbers = [1, 2, 3];
const doubled = numbers.map(num => num * 2);
console.log(doubled); // Output: [2, 4, 6]
```

Filter: The filter() method creates a new array with elements that pass a specified condition.

Example:

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

console.log(evenNumbers); // Output: [2, 4]
```

📚 Further Reading:

- https://www.w3schools.com/js/js_arrow_function.asp
- https://www.freecodecamp.org/news/javascript-rest-vs-spread-operators/
- https://www.freecodecamp.org/news/array-vs-object-destructuring-in-javascript/
- https://www.freecodecamp.org/news/javascript-map-reduce-and-filter-explained-with-examples/

YouTube tutorials for hands-on practice:

- https://www.youtube.com/watch?v=tJOJPealurs
- https://www.youtube.com/watch?v=sOZwwL_-cBA
- https://www.youtube.com/watch?v=_BsE5kmJk6Q
- https://www.youtube.com/watc

? Practice Question

Q1) What is the main advantage of using an arrow function over traditional function expressions?

Q2) Explain how the this keyword behaves differently in arrow functions compared to regular functions. Provide an example to illustrate the difference.

Q3) What is the purpose of the map() method in JavaScript arrays, and how does it differ from the forEach()?

Q4) What is difference between the rest and spread operators? Give suitable examples.

Q5) How is array destructuring different from object destructuring? Give suitable examples.

DAY 15

🚀 Asynchronous Javascript

(One of the hottest topics of JS nowadays)

Topics:

Synchronous vs Asynchronous Code

Synchronous Code: In synchronous code execution, statements are executed one after the other, and each statement must complete before the next one begins. This means that if one operation takes a long time, it can block the entire program.

Example of synchronous code:

```
console.log("Step 1");
console.log("Step 2");
console.log("Step 3");
```

🚀 Callbacks and callback hell

Callbacks:

Callbacks are functions passed as arguments to other functions, to be executed after the completion of an asynchronous operation. They are a common pattern for handling asynchronous code in JavaScript

Example:

```
function fetchData(callback) {
    // Simulating an asynchronous operation
    setTimeout(() => {
        const data = "Some data from the server";
        callback(data);
    }, 1000);
}

fetchData((data) => {
    console.log("Data received:", data);
});
```

🚀 Callback Hell:

Callback hell, also known as "Pyramid of Doom," occurs when multiple nested callbacks make the code hard to read and maintain. This happens when dealing with complex asynchronous operations.

Example of callback hell:

```
asyncFunction1((result1) => {
asyncFunction2((result2) => {
asyncFunction3((result3) => {
// ...and so on
});
});
```

🚀 Promises in Javascript

Promises are a more structured and readable way to handle asynchronous operations. A promise represents a value that may not be available yet but will be at some point in the future. Promises can be in one of three states: pending, resolved (fulfilled), or rejected.

Example of using a promise:

```
function fetchData() {
    return new Promise((resolve, reject) => {
        // Simulating an asynchronous operation
        setTimeout(() => {
            const data = "Some data from the server";
            resolve(data); // Promise is resolved with the data
        }, 1000);
    });

}fetchData()
    .then((data) => {
        console.log("Data received:", data);
    })
    .catch((error) => {
        console.error("Error:", error);
    });
```

🚀Event loop

The event loop is a core concept in JavaScript's concurrency model. It manages the execution of code, including asynchronous operations, by processing the call stack, message queue, and callback queue. The event loop ensures that JavaScript remains single-threaded and non-blocking.

Example (simplified explanation of the event loop):

```
console.log("Step 1");

setTimeout(() => {
    console.log("Step 2 (after time}, 1000);

console.log("Step 3");

// Output: Step 1, Step 3, Step 2 (after timer)
```

📚 Further Reading:

- https://www.freecodecamp.org/news/asynchronous-javascript/
- https://www.w3schools.com/js/js_asynchronous.asp
- https://www.w3schools.com/js/js_promise.asp
- https://www.w3schools.com/js/js_async.asp
- https://developer.mozilla.org/en-US/docs/Web/JavaScript/Event_loop
-

YouTube tutorials for hands-on practice:

- https://www.youtube.com/watch?v=zgt5oTD3rRc
- https://www.youtube.com/watch?v=bLre6Uf4Op0
- https://www.youtube.com/watch?v=ZYb_ZU8LNxs
- https://www.y

❓ Practice Question

Q1) How does JavaScript handle long-running tasks like network requests or file I/O without blocking the main thread?

Q2) Explain the states of a Promise (pending, fulfilled, rejected) and how error handling is done with Promises.

Q3) Explain the difference between using promises and using async/await.

Q4) Describe the role of the call stack, callback queue, and message loop in the Event Loop process.

🚀 Congratulations for mastering Javascript!

Thinking for next steps?

Learn a frontend framework – React.js, Angular.js or Vue.js
Learn Backend Technologies – Node.js, Express.js

Following one of the above steps will either make you a Frontend or Backend developer depending on what you choose.
Following both of them will lead you to the path of Full-Stack Developer. Yes, you can master MERN/MEAN stack once you master the concepts of Javascript.

If you want we can make such roadmaps for the above mentioned frameworks as well.
Let us know in the comments which technology's roadmap do you want next.

Stay tuned for such content with HeyCoach.