Complete Notes On

JavaScript ...

Instagram : @Curious_.programmer

Telegram : @Curious-coder

Written By :

Divya (CSE Student)

classmate
Date
Page

Index

1. Introduction to JavaScript

- JavaScript is a high-level, dynamic, and versatile programming language primarily used for web development.

- It was created by Brendan Eich in 1995 while he was working at Netscape Communications Corporation. Initially, it was named "Mocha" and later "LiveScript" before finally being called "JavaScript."

1.1. History Of JavaScript:

- JavaScript was developed to add interactivity to web pages, enabling developers to create dynamic and responsive content for users.

- It gained widespread adoption due to its ability to run on any web browser, making it a cross-platform language.

- In 1997, JavaScript was standardized under the name ECMAScript by the European Computer Manufacturers Association (ECMA), which is why you'll often hear the terms "JavaScript" and "ECMAScript" used interchangeably.

1.2. Purpose of JavaScript:

- JavaScript's primary purpose is to enable client-side scripting on web pages. This mean it allows developers to manipulate the content and behavior of web pages directly within the user's web browser.

- With JavaScript, we can dynamically update web pages elements, validate form inputs, and respond to user interactions like clicks, mouse movements, and keyboard input.

- Additionally, JavaScript can interact with web servers through AJAX (Asynchronous JavaScript and XML) to fetch data without requiring a page reload.

1.3. Setting up the development environment:

To start coding in JavaScript, we need a development environment set up.

Here are the basic Steps:

1. Text Editor:
Choose a text editor or an Integrated Development Environment (IDE) that supports JavaScript syntax highlighting. Examples include Visual Studio Code, Sublime Text, or Atom.

2. Web Browser:
JavaScript runs in web browsers, so we need a modern browser like Chrome, Firefox, or Edge to execute our JavaScript code.

3. HTML file:
Create a new HTML file that will serve as the container for our JavaScript code. This will include our HTML structure and link to the JavaScript code using the script tag.

4. JavaScript code:
Write the JavaScript code within the script tags in the HTML file or in a separate .js file, which we link to our HTML file.

5. Testing:
Open the HTML file in our web browser to see the result of our JavaScript Code.

As we progress in our JavaScript learning journey, we may explore using more advanced development tools, frameworks, and libraries to stream our workflow and build complex applications.

2. Basics of JavaScript

JavaScript, being a fundamental programming language, has several essential concepts that form it's core.
Let's dive into the basics of JavaScript, including variables and data types, operators and expressions, and control structures.

2.1. Variables and Data Types:

- Variables are containers used to store data values in JavaScript. We can declare variables using the 'var', 'let' or 'const' keyword.

For Example:

```
var age = 30;
let name = 'John';
Const PI = 3.14;
```

- JavaScript has various data types, including:

  - Primitive data Types:
    numbers, strings, booleans, null, undefined, and symbols.

- complex data types: objects (arrays, functions, and objects themselves).

2.2. Operators and Expressions:

- Operators are symbols used to perform operations on variables and values.
- JavaScript supports various types of operators, such as arithmetic, assignment, comparision, logical e.t.c.

Examples:

```
Var  x = 10;
var  y = 5;

var  sum = x+y;
var  difference = x-y;

var  isTrue = x > y;
```

- Expressions are combinations of variables, and values, and operators that evaluate to a single value

For Example:

```
var result = (x+y) * 2;
```

2.3 Control Structures:

- Control statements or structures allows us to control the flow of our code, making decisions or repeating actions based on conditions.

• If-else:

It allows us to execute a block of code if a certain condition is true or another block of code if the condition is false.

Example:

```
var age = 18;

if (age >= 18) {
  console.log("You can vote.");
}
  else {
    console.log("You can't vote");
}
```

- Loops :

for Loop : It allows us to execute a block of data / code repeatedly based on a specified condition.

Example :

```
for (var i = 1 ;  i <= 5 ;  i++) {

    console.log ("Iteration : " + i);

}
```


- while loop : It executes a block of code as long as a specified condition is true.

Example :

```
var count = 1 ;

while ( count <= 5 ) {

    console.log ("Count : " + count );

    count ++ ;

}
```

• do-while loops: Similar, to a while loop, but it executes the code at least once before checking the condition.

```
var num = 1;

do {

    console.log ("Number: " + num);

    num ++;

}

while (num <= 5);
```

3. Functions and Scope:

3.1. Defining functions:

In JavaScript, a function is a reusable block of code that performs a specific task. It allows us to encapsulate logic and execute it whenever needed.

Functions are defined using the 'function' keyword, followed by a name, a set of parentheses '()', and curly braces '{}' containing the function's body.

Example:


```
function greet() {

    console.log("Hello!");

}
```

3.2 Function Parameters and Return values:-

- Parameters are placeholders for values that we can pass to a function when calling it. They enable us to customize the behaviour of the function dynamically. Parameters are

defined inside the parentheses of the function declaration.

Example:

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

- Return values: Functions can also return values using the 'return' statement. The returned value can then be used in other parts of the code.

Example:

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


3.3. Scopes and Closures:

- Scope refers to the context in which variables and functions are accessible in our code. In JavaScript there are

two main types of scope:

- GLOBAL SCOPE:
Variables declared outside any function have global scope and can be accessed from anywhere in the code.

- LOCAL SCOPE:
Variables declared inside a function have local scope and are only accessible within that function.

Example of Local Scope:

```
function myFunction () {
        var x = 10;
        console.log(x);
}
console.log(x);
```


- Closures : A closure is a powerful feature in JavaScript that allows a function to remember and access its lexical scope even when it's executed outside that scope. This means a function can retain access to its parent function's variables even after the parent function has finished

executing.

Example of a closure:

```
function outerFunction() {
    var outerVariable = "I am from the
                outer function";

function innerFunction() {
    console.log(outerVariable);
}
  return innerFunction;
}

var closureExample = outerFunction();
closureExample();
```

4. Arrays and Objects:

4.1. Working with Arrays:

- Arrays are data structures in JavaScript used to store collections of elements. They allow us to group multiple values into a single variable. Arrays are created using square brackets '[]' and each element is separated by comma.

Example:

```
var fruits = ['apple', 'banana', 'orange'];
```

- Accessing elements: We can access individual elements in an array using their index, which starts from 0.

Example:

```
var firstFruit = fruits[0];
```

- Modifying elements: We can modify elements in an array by assigning new values to their corresponding indices.

Example:

```
fruits[1] = 'grape';
```

- Array methods: JavaScript provides various built-in methods to manipulate arrays such as 'push()', 'pop()', 'shift()', 'unshift()', 'splice()' and more.

4.2. Creating and manipulating Objects:

- Objects are collections of key-value pairs. Each key is a property that represents a name or identifier, and each value is the associated data.

Objects are enclosed in curly braces '{}'.

Example:

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

- Accessing Object Properties: We can access properties of an object using dot notation or square brackets.

Example:

```
var personName = person.name;
var personAge = person['age'];
```

- Modifying Object Properties:
We can modify object properties by assigning new values to them.

Example:


```
person.age = 31;
person['city'] = 'San Francisco';
```

- Nested Objects: Objects can also contain other objects as properties, creating nested structures.

Example:

```
var student = {
        name: 'Alice',
        age: 25,
        address: {
        street: '123 Main St',
        city: 'Los Angeles'
    }
};
```

4.3. Array and Object methods:

Arrays and objects have built-in methods that allow us to perform various operations efficiently.

Some common methods include:

- Array methods:

'push()', 'pop()', 'shift()', 'unshift()', 'splice()', 'concat()', 'slice()', 'indexOf()', 'forEach()', 'map()', 'filter()' and more.

- Object methods:

'Object.keys()', 'Object.values()', 'object.entries()', and 'Object.assign()'.

Example of using array and object methods:

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

numbers.push(6);
numbers.pop();
```

```
var student = {
        name : 'Bob',
        age : 22,
        city : 'chicago'
};

var keys = Object.keys(student);
var values = Object.values(student);
```

5. DOM Manipulation

5.1. Introduction to the Document Object Model:

- The Document Object Model (DOM) is a programming interface for HTML and XML documents. It represents the structure of a web page as a hierarchial tree of objects. Each element in the HTML document is represented as a node in the DOM tree, allowing developers to interact with and manipulate the content and structure of a web page dynamically.

5.2. Accessing and Modifying HTML Elements:

- JavaScript can be used to access and modify HTML elements in the DOM, enabling dynamic updates to the web page's content and appearance.

- Accessing elements: We can use various methods to access elements by their ID, class, tag name, or other attributes.

Examples:

```
var headingElement = document.getElementById('heading');

var elementsWithClass = document.getElementsByClassName('highlight');

var allParagraphs = document.getElementsByTagName('p');
```

- Modifying elements: Once we have access to an element, we can change its content, attributes, or styles using JavaScript.

Examples:

```
headingElement.textContent = 'New Heading';

var imageElement = document.getElementById('myImage');

imageElement.src = 'new-image.jpg';

var paragraphElement = document.getElementById('myParagraph');

paragraphElement.style.color = 'blue';
```

5.3. Handling Events:

- Events are actions or occurrences that happen in the web page, such as clicking a button, moving the mouse, or pressing a key. JavaScript allows us to handle these events and perform specific actions in response.

- Adding event listeners: We can attach event listeners to elements to listen for specific events and trigger functions when those events occur.

Examples:

```
var myButton = document.getElementById('myButton');

myButton.addEventListener('click',
    function() {
        alert('Button clicked!');
});

document.addEventListener('keydown',
    function(event){
        console.log('key pressed:',
            event.key);
});
```

DOM manipulation is a crucial aspect of front-end web development, allowing developers to create interactive and dynamic user experiences.

With JavaScript and the DOM, we can build engaging web applications and respond to user interactions effectively.

6. Asynchronous JavaScript

6.1. Introduction to Asynchronous Programming:

Asynchronous programming in JavaScript allows tasks to run independently without blocking the execution of the rest of the code. It's essential for handling time-consuming operations, such as making API requests, reading files, or waiting for user input, without freezing the user interface.

6.2. Callback functions:

A common approach to asynchronous programming in JavaScript is using callback functions.

A callback is a function that is passed as an argument to another function and executed later, usually when an asynchronous task is complete.

```
function fetchDataFromServer(callback){
    setTimeOut(function(){
        var data = { name: 'John',
                     age: 30 };
```

```
    callback(data);
  }, 1000);
}

function processData(data) {
  console.log('Data received', data);
}

fetchDataFromServer(processData);
```


6.3. Promises and async/await:

- Promises are an improved way of dealing with asynchronous operations in JavaScript.
  They represent a value that may be available now or in the future, either resolved (successful) or rejected (failed).

```
function fetchDataFromServer() {
  return new Promise(function(resolve, reject) {
    setTimeout(function() {
      var data = { name: 'John', age: 30};
```

```
resolve (data);
 }, 1000);
});

fetchDataFromServer()
    .then (function (data){
        console.log ('Data received:', data);
})
    .catch (function (error) {
      console.error ('Error:', error.message);
});
```


- 'async / await' is a modern syntax introduced in ES2017 that simplifies working with promises. It allows us to write asynchronous code in a more synchronous-like manner, making it easier to read and maintain.

```
async function fetchData (){
    try {
        var data = await fetchDataFromServer();
        console.log ('Data received :', data);
}
 catch (error) {
        console.error ('Error:', error.message);
    }
}
fetchData();
```

Asynchronous JavaScript is vital for handling time-consuming tasks and ensuring a smooth user experience in web applications.

Callbacks, promises, and 'async/await' are powerful tools that enable developers to write efficient and responsive code for handling asynchronous operations.

7. Error Handling

7.1. Handling Exceptions and errors:

Error handling in JavaScript is the process of managing unexpected situations or exceptions that may occur during the execution of our code. Errors can be caused by various factors, such as incorrect input, network issues, or logical mistakes in the code.

7.2. Handling Exceptions:

In JavaScript, exceptions are raised when an error occurs during the execution of a statement or a function. We can handle exceptions using 'try', 'catch', and optionally 'finally' blocks.


```
try {
    var result = someFunction();
}
catch (error) {
    console.error('An error occured:',
        error.message);
} finally {
    console.log('The try-catch block has
        finished');
}
```

By using the try-catch block, we can gracefully handle errors and prevent our application from crashing.

7.3 Debugging Techniques:

Debugging is the process of identifying and fixing issues in our code.

JavaScript provides various techniques to debug our code and find the root cause of errors.

- **console.log():** The simplest way to debug is by adding 'console.log()' statements in our code to output variables or messages to the browser's console.

- **console.error():** Use 'console.error()' to print error messages to the console explicitly. This is helpful for highlighting critical issues in our code.

- **Breakpoints:** Modern browsers ~~often~~ offer developer tools that allow us to set breakpoints in our code. A break point stops the execution of our code at a specific line, enabling us to

inspect variables and step through the code to understand its flow better.

- **Debugger Statement**: We can insert the 'debugger' statement in our code, and when the code is executed, it will pause at that line if the developer tools are open.

- **Stack Trace**: When an exception occurs, the stack trace provides information about the sequence of function calls leading to the error. It helps us to track the flow of execution and identify the source of the error.

- **Browser Developer Tools**: Use the browser's developer tools to inspect and debug our code. They offer features like console logs, network monitoring, performance analysis, and more.

8. ES6 and Beyond

8.1. New features introduced in ECMAScript 6 and later versions:

- ES6, short for ECMAScript 6, is the sixth edition of the ECMAScript standard, which is the specification that defines the JavaScript language.
- It was released in 2015 and introduced several new features and enhancements to JavaScript.
- Since then, newer versions of ECMAScript have been released, each bringing additional improvements and features.

Here are some notable features introduced in ECMAScript 6 and beyond:

1. Let and const:

Introduced in ES6, 'let' and 'const' are block-scoped variables. 'let' allows us to declare variables that can be reassigned, while 'const' declares constants whose value cannot be changed.

2. Arrow Functions:

Arrow functions provide a concise syntax for writing functions in JavaScript. They have implicit returns and lexically bind the value of 'this'.

3. Classes:

ES6 introduced class syntax for creating objects, supporting constructors, methods and inheritence. It is more familiar to developers coming from other object-oriented programming languages.

4. Template Literals:

Template literals allow us to create multiline strings and embed expressions within ~~brack~~ backticks (` `) instead of using single or double quotes.

5. Destructing Assignment:

Destructing assignment allows us to extract values from arrays or objects into separate variables using a concise syntax.

6. Spread and Rest operators:

The spread operator ('...') spreads the elements of an array into individual elements, while the rest parameter ('...') collects individual arguments into an array.

7. Promises:

Promises provide a better way to work with asynchronous operations in JavaScript, making it easier to handle and success and failure conditions.

8. Modules:

ES6 introduced a standardized module system using 'import' and 'export' statements, allowing for better code organization and reuse.

9. Default parameters:

We can define default values for function parameters if no value is passed or if the argument is 'undefined'.

10. Symbol :

Symbols are unique and immutable data types introduced in ES6, useful for creating non-enumerable properties and avoiding naming collisions.

11. Iterators and Generators :

Iterators allow us to loop through data structures, while generators are ~~struc~~ functions that can pause and resume their execution, useful for creating iterators.

12. Enhanced Object Literals :

ES6 introduced enhancements to object literals, allowing us to define method and computed property names.

13. Async / Await :

Introduced in ES2017, 'async/await' is a powerful syntactic feature that simplifies working with promises and handling asynchronous code in a more synchronous-like manner.

These features significantly improve JavaScript's readability, maintainability, and expressiveness, making it easier for developers to build complex applications with fewer lines of code and better organization.

9. Project Work

9.1. Building a Small JavaScript appication or website:

Project work in JavaScript involves building a small application or website using our coding skills and knowledge of JavaScript. It is an excellent way to apply what we have learned and gain hands-on experience in real-world scenarios.

Here are some key steps and considerations when working on a JavaScript project:

1. Define the project scope:

- Decide on the type of project we want to build. It could be a simple to-do list app, a weather app, a calculator, a portfolio website, or anything that interests us.

- Clearly define the features and functionality we want to implement in our project.

2. Plan and Design:

- Create a rough sketch or wireframe of our application's user interface. Decide on the layout, components, and how users will interact with it.
- Plan the project's structure, including the different JavaScript functions, modules and files we'll need.

3. Set-up the development environment:

- Ensure we have a text editor or IDE of our choice and a modern web browser.

- Set up a version control system like Git to track changes and mange our code effectively.

4. Implement the Functionality:

- Write the necessary HTML, CSS and JavaScript code to create the desired features.
- Pay attention to code organization and best practices to make our code readable and maintainable.

5. Test and Debug:

- Test the application throughly to identify and fix and any bugs or issues.
- Use debugging techniques like 'console.log()' or browser developer tools to inspect variables and troubleshoot problems.

10. Working with Dates and Times:

Working with dates and times is a common task in web development, as it enables us to display time sensitive information, schedule events, and perform various date related calculations.

In this chapter, we will explore how to create, manipulate, format and display dates in JavaScript.

10.1. Creating and manipulating Date objects:

In JavaScript, we can work with dates using the date object, which provides methods for creating and manipulating dates.

To create a new Date object, we can use the new Date() constructor.

```
var currentDate = new Date();
console.log(currentDate);
                // output the current date
                    and time
```

We can also create specific dates by passing year, month, day, hr,

minute, second and millisecond values to the Date Constructor.

```
var specificDate = new Date(2023, 6,
                   15, 12, 30, 0, 0);
console.log(specificDate);

// output Sun Jul 15 2023 12:30:00
```


To manipulate dates, we can use various methods such as 'setFullyear(), setMonth(), setDate(), setHours(), setMinute(), setSecond(), and SetMillisecond().

```
var date = new date();
date.setFullYear(2023);
date.setMonth(7);
date.setDate(23);
console.log(date);

//output : wed Dec 05 2023 09:00:00
```

10.2 Formatting and displaying dates:

To display data in a more readable format, we can use various methods to extract specific date and components and format them as needed.

```
var date = new Date();
var year = date.getFullYear();
var month = date.getMonth() + 1;
// Adding 1 to match the actual month
var day = date.getDate();
var hours = date.getHours();
var minutes = date.getMinutes();
var seconds = date.getSeconds();
console.log(year + '-' + month + '-' +
    day + '-' + hours + ':' + minutes
    + ':' + seconds);
//output: Current date and time in
    'yyyy-mm-dd' 'HH:mm:ss'
    format
```


Alternatively, we can use 'toLocalString()' method to get a localised date and time representator.

```
• console.log(date.toLocalString());
    // output: localised date and
    time representation.
```

10.3. Performing Date calculations and comparisons:

JavaScript provides several methods to perform date calculations and comparisons. We can use methods like getTime() to get the timestamp of a date and

perform mathematical operations on dates.

```
var date1 = new Date(2023, 6, 15);
var date2 = new Date(2023, 8, 20);

var differenceInMilliseconds = date2.getTime() - date1.getTime();
var differenceInDays = differenceInMilliseconds / (1000 * 60 * 60 * 24);
console.log(differenceInDays);

// Difference in days between the 2 dates
```


We can also compare dates using comparison operators like (<, >, =, >=).

10.4 Working with Time Zones:

Dealing with time zones can be challenging, as different locations have different time effects. JavaScript Date object uses the host system's time zone by default, but we can work with different time zones using libraries like Moment.js or the built-in 'toLocalString()' method with appropriate options.

```
var date = new Date();
console.log(date.toLocaleString('en-US',
    { timeZone: 'America/NewYork' })
);

// Output: Date and time in New York
           time zone.
```

11. AJAX and Fetch API

Asynchronous JavaScript and XML (AJAX) and fetch API are powerful tools that allow web developers to create dynamic and interactive web applications.

In this chapter, we will explore asynchronous programming, making HTTP requests with the fetch API, handling responses, manipulating data and working with JSON data.

11.1. Introduction to Asynchronous Programming:

In traditional programming, code executes in a sequential manner, one line after another. Asynchronous programming, on the other hand, allows tasks to run independently of the main program flow. This is essential for tasks that may take time to complete, such as making HTTP requests to external servers or fetching data from a database.

Asynchronous programming is crucial for building responsive web applications, as it prevents long-running tasks from blocking the user interface and provides a smoother user experience.

11.2. Making HTTP requests with Fetch API:

The fetch API is a modern and native JavaScript API that simplifies making HTTP requests. It provides a simple and easy to use interface to fetch resources (such as JSON data, images or text) from the server.

Here's an example of making a GET request using the fetch API.

```
fetch ('https://api.example.com/data')
  .then (response => response.json())
  .then (data => console.log (data))
  .catch (error => console.log ('Error
      fetching data', error));
```

In this example, fetch is used to make a GET request to the specified URL. The response is converted to JSON format using the JSON() method. The data is then logged to the console.

11.3 Handling responses and data manipulation:

Once we receive the response from the server, we can handle it and manipulate.
For example: we can extract specific information, display it on the web page, or use it for other tasks.

```
fetch('https://api.example.com/data')
  .then(response => response.json())
  .then(data => {
    // manipulate and use data.
    console.log(data);
    document.getElementById('result').textContent = data.message;
  })
  .catch(error => console.error('Error fetching data', error));
```

In this example, the data received from the server is used to set the content of an element with the ID 'result' on the web page.

11.4. Working with JSON data:

JSON (JavaScript Object Notation) is a popular data format used for exchanging data between a web server and web client. The fetch API automatically converts the response to JSON using

the 'JSON()' method, making it easy to work with JSON data.

```
fetch('https://api.example.com/data')
  .then(response => response.json())
  .then(data => console.log(data))
  .catch(error => console.error
      ('Error fetching data:', error));
```

JSON data is usually structured as key-value pairs, making it easy to access and manipulate specific data elements.

12. Introduction to JavaScript Frameworks and Libraries

JavaScript frameworks and Libraries are essential tools for web developers, as they provide a structured and efficient way to build complex and interactive web applications.

In this chapter, we will explore popular JavaScript frameworks, introduce libraries, discuss their role in building interactive web applications, and examine the pros and cons of using JavaScript frameworks.

12.1 Overview of popular JavaScript frame works:

JavaScript frameworks are comprehensive tools that provide a complete structure and set of functionalities for building web applications.

Some of the most popular JavaScript frameworks include React, Angular and vue.

- React:
  Developed and maintained by facebook. React is a declarative compoment-

based JavaScript library for building user interfaces. It allows developers to create reusable UI components, making it easier to manage complex applications.

- **Angular:**
Developed and maintained by Google. Angular is a comprehensive JavaScript framework that provides a complete solution for building large scale applications. It offers powerful features such as data binding, dependency injection and routing.

- **Vue:**
Vue is a progressive and user-friendly JavaScript framework that allows developers to build interactive user interface. It provides a gentle learning curve and can be integrated incrementally into existing projects.

12.2. Introduction to Libraries:

JavaScript libraries on the other hand are collections of pre-written code that simplify and speed up development tasks. They are focused on providing specific functionalities,

making them lightweight and easy to use.

- **JQuery:**
JQuery is one of the most popular JavaScript libraries. It simplifies DOM manipulation, event handling, and AJAX calls, making it easier to work with web page elements and interact with servers.

- **Lodash:**
Lodash is a utility library that provides a wide range of helper functions to work with arrays, objects, strings and more. It improves code efficiency and readability by offering methods for common data manipulation tasks.

12.3. Using frameworks and Libraries for building interactive web applications:

JavaScript frameworks and libraries are essential for building interactive and dynamic web applications. They offer a structured architecture and a wealth of tools that enable developer to manage complex data, handle user interactions, and update our interface efficiently.

Using frameworks and libraries allows developers to save time and effort by leveraging pre-built solutions for common tasks. This, inturn, improves development speed and helps deliver more robust and feature rich applications.

12.4. Pros and cons of using JavaScript Frameworks:

Using JavaScript can be beneficial for many reasons:

- **Structured Architecture.**
  Frameworks provide a structured architecture that promotes organization and maintainability.

- **Reusability:**
  Components and frameworks can be reused across different parts of the application, promoting code reuse.

- **Efficiency:**
  Frameworks offer pre-optimised solutions for common tasks improving development efficiency.

However, there are also some cons to consider.

- **Learning curve:**
  Some frameworks have a steep learning curve, especially for beginners.

- **Complexity:**
  Large frameworks can introduce unnecessay complexity for smaller projects.

- **Performance:**
  Some frameworks may add overhead to the application, affecting performance.