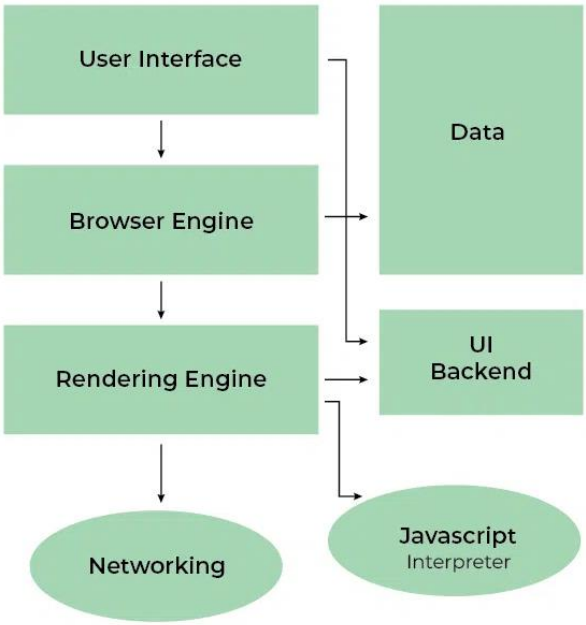
MCA-III Semester
303-WEB TECHNOLOGIES
(Unit 1&2 Questions Internal Examination-I)

1. Explain & Describe the BROWSER ARCHITECTURE?

A web browser consists of several modular and interdependent components that work together to fetch, process, and display web pages. The architecture typically follows a multi-process or multi-threaded design in modern browsers for enhanced stability and security.



Key Components of Browser Architecture

1. User Interface

- **Description**: The visible part of the browser that interacts with the user.
- **Components**:
  - Address bar for entering URLs.
  - Navigation buttons (Back, Forward, Refresh, Home).
  - Bookmarks, settings, and other UI controls.
- **Function**: Receives user input and displays content or feedback.

2. Browser Engine

- **Description**: The intermediary layer between the UI and the rendering engine.
- **Function**:
  - Translates user actions (like navigating to a URL) into commands for the rendering engine.
  - Manages browser state and coordinates interactions.

3. Rendering Engine

- **Description**: The core component that renders HTML, CSS, and JavaScript into a visual representation.
- **Examples**: Blink (used by Chrome), WebKit (used by Safari), Gecko (used by Firefox).
- **Function**:
  - Parses HTML and builds the DOM (Document Object Model).
  - Parses CSS to create the CSSOM (CSS Object Model).
  - Combines the DOM and CSSOM into the Render Tree.
  - Lays out and paints the content on the screen.

4. Networking

- **Description**: Handles all network operations.
- **Function**:
  - Sends HTTP/HTTPS requests to web servers.
  - Downloads web resources (HTML, CSS, JavaScript, images, etc.).
  - Implements caching and cookies for performance and session management.
  - Ensures secure communication using SSL/TLS protocols.

5. JavaScript Engine

- **Description**: A separate component for executing JavaScript code.
- **Examples**: V8 (Chrome, Edge), SpiderMonkey (Firefox), JavaScriptCore (Safari).
- **Function**:
  - Parses and compiles JavaScript code.
  - Executes JavaScript efficiently for interactive web applications.

6. UI Backend

- **Description**: Provides support for drawing widgets like buttons, windows, and scrollbars.
- **Function**:
  - Uses system-specific APIs to render UI components.
  - Interfaces with the rendering engine for consistent visuals.

7. Data Storage

- **Description**: Stores data for offline use and session continuity.
- **Types of Data**:
  - Cookies
  - Cache
  - IndexedDB
  - LocalStorage and SessionStorage
- **Function**: Provides mechanisms for persistent storage of user data and web content.

8. Security and Sandboxing

- **Description**: A modern browser feature that isolates processes to enhance security.
- **Function**:
  - Prevents a compromised website from affecting other tabs or the system.
  - Separates processes for rendering, networking, and JavaScript execution.

2. Differences between CHROME and a SEARCH ENGINE

Google Chrome:

1. **Web Browser**: Google Chrome is a web browser developed by Google. It allows users to access and navigate websites on the internet. It supports features like tabs, extensions, and private browsing to enhance the browsing experience.
2. **Performance & Speed**: Known for its fast performance, Chrome uses the Blink rendering engine, which optimizes the loading and display of web pages. It also includes features like prefetching resources to speed up website loading times.

Search Engine:

1. **Information Retrieval Tool**: A search engine is a tool or system that helps users find information on the internet by searching through vast amounts of data indexed from websites. Examples include Google, Bing, and Yahoo.
2. **Search Algorithm**: Search engines use complex algorithms to rank websites based on relevance and other factors such as keywords, page quality, and backlinks. This determines the order in which results are shown to users after a query is entered.

Google Chrome	Search Engine
Software used to access websites and display web content.	Online tool for searching and retrieving web content.
Enables users to navigate the web and interact with web apps.	Helps users find specific information on the internet.
Displays and renders web pages using web technologies.	Provides search results (links to websites) based on queries.
Directly loads web pages when given a URL.	Requires a browser to function and display results.
URLs or links entered in the address bar.	Keywords or queries entered into the search bar.
Full webpage rendered visually.	A list of links or resources matching the search query.
Can operate independently (e.g., visiting direct URLs).	Requires a browser to be accessed.
Tab management, extensions, offline access, developer tools.	Advanced search filters, algorithms, and ranking.
Developed by Google (or other browser providers).	Search engines are typically platforms (Google, Bing).
Examples: Google Chrome, Mozilla Firefox, Safari, Microsoft Edge.	**Examples:** Google Search, Bing, Yahoo, DuckDuckGo.

3. Create the login form by using CSS styles.

Key Elements of a Login Form

1. **Username/Email Field:**
   - Input field to accept the user's unique identifier (e.g., username, email).
2. **Password Field:**
   - Secure input field to accept the user's password, usually masked (e.g., ***).
3. **Submit Button:**
   - Button to send the login credentials to the server for validation.
4. **Additional Features:**
   - Links for "Forgot Password?" or "Sign up."
   - Checkbox for "Remember Me" functionality.

LoginForm.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Login Form</title>
    <link rel="stylesheet" href="styles.css">
</head>
<body>
    <div class="login-container">
        <form class="login-form">
            <h2>Login</h2>
            <div class="input-group">
                <label for="username">Username</label>
                <input type="text" id="username" name="username" placeholder="Enter your username"
required>
            </div>
            <div class="input-group">
                <label for="password">Password</label>
                <input type="password" id="password" name="password" placeholder="Enter your password"
required>
            </div>
            <button type="submit" class="login-btn">Login</button>
                <p class="signup-link"><a href="#">Forget Password?</a></p>

            <p class="signup-link">Don't have an account? <a href="#">Sign up</a></p>
        </form>
    </div>
</body>
</html>
```

styles.css

```
* {
    margin: 0;
    padding: 0;
    box-sizing: border-box;
    font-family: Arial, sans-serif;
}

/* Body Styling */
body {
    display: flex;
    justify-content: center;
    align-items: center;
    height: 100vh;
    background-color: #f4f4f4;
}

/* Login Form Container */
.login-container {
    background-color: #ffffff;
    padding: 20px 30px;
    border-radius: 10px;
    box-shadow: 0 4px 10px rgba(0, 0, 0, 0.1);
    width: 100%;
    max-width: 400px;
}

/* Form Heading */
.login-form h2 {
    text-align: center;
    margin-bottom: 20px;
    color: #333333;
}

/* Input Group */
.input-group {
    margin-bottom: 15px;
}

.input-group label {
    display: block;
    margin-bottom: 5px;
    font-weight: bold;
    color: #555555;
}

.input-group input {
    width: 100%;
    padding: 10px;
    font-size: 16px;
    border: 1px solid #dddddd;
    border-radius: 5px;
    outline: none;
    transition: border-color 0.3s;
}

.input-group input:focus {
    border-color: #007BFF;
}

/* Login Button */
.login-btn {
    width: 100%;
    padding: 10px;
    background-color: #007BFF;
    color: #ffffff;
    font-size: 16px;
    font-weight: bold;
    border: none;
    border-radius: 5px;
    cursor: pointer;
    transition: background-color 0.3s;
}

.login-btn:hover {
    background-color: #0056b3;
}

/* Signup Link */
.signup-link {
    margin-top: 10px;
    text-align: center;
    color: #555555;
}

.signup-link a {
    color: #007BFF;
    text-decoration: none;
    font-weight: bold;
}

.signup-link a:hover {
    text-decoration: underline;
}
```

4. Explain Types of CSS

CSS (Cascading Style Sheets) describes the HTML elements which are displayed on the screen, paper, or other media. It saves a lot of time by controlling the layout of multiple web pages simultaneously. CSS sets the font size, font family, color, and background color on the page.

It also allows us to add effects or animations to the website. With CSS, we can display animations such as buttons, effects, loaders or spinners, and even animated backgrounds.

Without CSS, websites would not look attractive. There are three main types of CSS:

1. Internal/Embedded CSS
2. External CSS
3. Inline CSS

1. Internal CSS

Internal CSS is written inside the <style> tag in the <head> section of the HTML document. This CSS style is an effective way to style a single page. However, using CSS styles for multiple web pages can be time-consuming, as the styles need to be added to each web page.

How to use Internal CSS:

1. Open the HTML page and locate the <head> section.
2. Add the following code after the <head> tag:

```
<style type="text/css">
  body {
    background-color: black;
  }
  h1 {
    color: white;
    padding: 50px;
  }
</style>
```

3. After adding the internal CSS, the complete HTML file will look like this:

```
<!DOCTYPE html>
<html>
<head>
  <style>
    body {
      background-color: black;
    }
    h1 {
      color: red;
      padding: 50px;
    }
  </style>
</head>
<body>
  <h2>CSS types</h2>
  <p>Mother Theresa Institute of Computer Applications</p>
</body>
</html>
```

We can also use selectors (class and ID) in the stylesheet, like this:

```
.class {
   property1 : value1;
   property2 : value2;
   property3 : value3;
}

#id {
   property1 : value1;
   property2 : value2;
   property3 : value3;
}
```

Pros of Internal CSS:

- Does not require uploading multiple files as the code is added directly to the HTML page.

Cons of Internal CSS:

- Adding the code to the HTML document can reduce the page size and increase the loading time of the webpage.

2. External CSS

In external CSS, we link the web pages to an external .css file, which is created using a text editor. External CSS is a more efficient method for styling a website because by editing the .css file, we can change the styles across the whole site at once.

How to use External CSS:

1. Create a new .css file with a text editor and add the CSS rules, like this:

```
.xleftcol {
   float: right;
   width: 35%;
   background: #608800;
}
.xmiddlecol {
   float: right;
   width: 35%;
   background: #eff3df;
}
```

2. Add a reference to the external .css file right after the <title> tag in the <head> section of the HTML document:

```
<link rel="stylesheet" type="text/css" href="style.css" />
```

Pros of External CSS:

- The file structure is cleaner and smaller in size.
- The same .css file can be used for multiple web pages.

Cons of External CSS:

- Pages cannot be displayed correctly until the external CSS is loaded.
- Uploading many CSS files can increase the download time of a website.

3. Inline CSS

Inline CSS is used to style a specific HTML element. It involves adding a style attribute to each HTML tag without using selectors. While managing a website with only inline CSS can be difficult, it is useful in certain situations where we do not have access to external CSS files.

Example of Inline CSS:

```
<!DOCTYPE html>
<html>
<body style="background-color: white;">
   <h1 style="color: Red; padding: 20px;">Welcome to Web Technologies</h1>
   <p style="color: blue;">Happy Birthday to you!</p>
</body>
</html>
```

Pros of Inline CSS:

- Allows you to create CSS rules directly within the HTML page.
- No need to create or upload a separate CSS file.

Cons of Inline CSS:

- Adding CSS rules to each HTML element is time-consuming and can clutter the HTML structure.
- It styles individual elements, which can affect the page size and download time.

5. Define the Scripting. What is JavaScript ?

Definition of Scripting

Scripting refers to writing small programs, called scripts, to automate tasks, enhance user interactions, or control the behavior of a system. Unlike full-fledged programming languages, scripting languages often operate within a runtime environment (like a web browser) to execute pre-written instructions without the need for compilation.

Characteristics of Scripting

1. **Lightweight:** Scripts are typically small and designed for specific tasks.
2. **Interpreted:** Scripts are executed line by line by an interpreter rather than being compiled into machine code.
3. **Dynamic:** They often allow dynamic typing, late binding, and run-time manipulation of variables.
4. **Embedded:** Scripts are frequently embedded into larger systems, such as web pages or server environments.

Examples of scripting languages: JavaScript, Python, PHP, Ruby, Perl.

JavaScript

JavaScript is a **high-level, lightweight, interpreted programming language** primarily used to create interactive and dynamic content on web pages. It is one of the core technologies of the World Wide Web, alongside HTML and CSS, and is executed by web browsers.

Key Features of JavaScript

1. **Client-Side Execution**: Most JavaScript code runs directly in the browser, enhancing webpage interactivity without needing server communication.
2. **Event-Driven**: Responds to user actions like clicks, keystrokes, or mouse movements.
3. **Dynamic Typing**: Variables can hold different types of data at different times.
4. **Prototype-Based**: Uses prototypes instead of traditional class-based inheritance.
5. **Asynchronous Programming**: Supports asynchronous operations using promises, callbacks, and async/await.
6. **Platform Independence**: Runs on any platform with a web browser.

Applications of JavaScript

1. Web Development

- Creates dynamic and interactive web pages.
- Powers client-side functionalities like animations and form validation.
- Works with HTML and CSS for a complete front-end experience.
- Enables real-time updates without page reloads via AJAX.
- Used in frameworks like React and Angular for building complex web apps.

2. Mobile App Development

- Utilized in frameworks like React Native for cross-platform apps.
- Allows sharing a single codebase between iOS and Android.
- Facilitates building responsive, high-performance mobile applications.
- Supports integration with device features like cameras and GPS.
- Simplifies the mobile development process for web developers.

3. Server-Side Development

- Node.js enables JavaScript for backend development.
- Suitable for building scalable server-side applications.
- Handles asynchronous programming efficiently for real-time services.
- Integrates seamlessly with databases like MongoDB and MySQL.
- Ideal for developing APIs and microservices.

4. Game Development

- JavaScript is used for developing browser-based games.
- Supports rendering 2D and 3D graphics using WebGL and Canvas.
- Game frameworks like Phaser and Babylon.js simplify game logic.
- Enables creating multiplayer games with WebSocket support.
- Accessible for casual game development, playable on most browsers.

5. Browser Extensions

- Powers extensions for Chrome, Firefox, and other browsers.
- Facilitates interaction with browser APIs for customization.
- Commonly used for tools like ad blockers and productivity extensions.
- Enhances user experience with tailored browsing functionalities.
- Simplifies development with frameworks like Web Extensions.

Advantages of JavaScript

1. **Speed**: Executes directly in the browser without server-side delays.
2. **Versatility**: Can be used for both frontend and backend development.
3. **Rich Interfaces**: Enables the creation of interactive features like drag-and-drop and animations.
4. **Community Support**: Vast libraries and frameworks like React, Angular, and Vue make development faster.

Disadvantages of JavaScript

1. **Security Issues**: Since it runs on the client side, it is susceptible to attacks like Cross-Site Scripting (XSS).
2. **Browser Dependence**: Behavior may vary slightly between browsers, though modern browsers adhere closely to standards.
3. **Debugging Complexity**: Interpreted nature makes some errors harder to debug.

Simple Example of JavaScript

```
<!DOCTYPE html>
<html>
<head>
   <title>JavaScript Example</title>
</head>
<body>
   <h1 id="greeting">Hello, World!</h1>
   <button onclick="changeText()">Click Me</button>

   <script>
      function changeText() {
         document.getElementById("greeting").innerHTML = "You clicked the button!";
      }
   </script>
</body>
</html>
```

6. Brief discussion about the Control Structure using JavaScript?

Control structures are programmatic constructs that determine how your code executes. They provide mechanisms for making decisions, repeating code blocks, and changing the flow of execution based on specific conditions. By effectively utilizing these structures, you can write code that is more responsive, adaptable, and efficient.

JavaScript offers a rich set of control structures, categorized into three primary types:

- **Conditional Statements:** These structures allow your code to make decisions based on conditions. They execute specific code blocks only when the defined conditions are met.
- **Looping Statements:** These structures enable you to repeat a block of code a specific number of times or until a particular condition is met. This is crucial for iterating through data collections and performing repetitive tasks.
- **Jumping Statements:** These structures alter the normal flow of execution within a loop or conditional statement. They provide mechanisms for exiting loops before the loop stops or skipping iterations.

Conditional Statements

Conditional statements are the tools JavaScript uses to make choices. They allow your code to execute different code blocks based on the evaluation of conditions. Here, we explore the essential conditional statements:

- **if Statement:** This is the most fundamental conditional statement. It evaluates a condition and executes a block of code if the condition is true. Optionally, you can include an else block that executes if the condition is false.

EX:-

```
let grade = 85;
if (grade >= 90) {
  console.log("Excellent!");
} else {
  console.log("Keep practicing!");
}
```

- **else if Statement:** This allows for chaining multiple conditions within an if statement. It lets you check several things one by one, and only runs the code for the first thing that's true.

EX:-
```
let grade = 85;
if (grade >= 90) {
  console.log("Excellent!");
} else if (grade >= 80) {
  console.log("Great job!");
} else {
  console.log("Keep practicing!");
}
```

- **switch Statement:** This structure is used for multi-way branching based on the value of an expression. It provides a cleaner alternative to nested if statements when dealing with several possible conditions that evaluate to different values.

EX:-
```
let day = "Tuesday";
switch (day) {
  case "Monday":
    console.log("Start of the week!");
    break;
  case "Tuesday":
  case "Wednesday":
  case "Thursday":
    console.log("Midweek grind!");
    break;
  case "Friday":
    console.loop("TGIF!");
    break;
  default:
    console.log("Enjoy the weekend!");
}
```

Looping Statements

Looping statements are fundamental for iterating through data collections, executing code blocks repeatedly until a condition is met, and automating repetitive tasks. JavaScript offers three primary looping structures:

- **for Loop:** This handy loop lets you do something over and over again easily. It keeps track of where you are using a counter, and stops when a condition isn't met anymore.

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

In this example, the loop initializes i to 0, checks if i is less than 5 (the condition), and increments i by 1 after each iteration. This allows the loop to iterate five times, printing the value of i in each iteration.

- **while Loop:** This loop continues executing a code block as long as a specified condition remains true. The condition is checked before each iteration, and the loop continues as long as the condition evaluates to true.

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

In this example, the loop continues as long as count is less than 3. The count++ statement increments count after each iteration, eventually causing the condition to become false and the loop to stop.

- **do-while Loop:** This loop guarantees that the code block executes at least once, even if the initial condition evaluates to false. The condition is checked after the code block executes. The loop continues iterating as long as the condition remains true.

EX:-
```
let input = "";
do {
  input = prompt("Enter your name (or 'quit' to exit):");
} while (input.toLowerCase() !== "quit");
console.log("Hello,", input);
```

This example utilizes a do-while loop to continuously prompt the user for input until they enter “quit”. The loop executes at least once, ensuring the user gets a chance to enter their name.

Jumping Statements

JavaScript provides two primary jumping statements that change the normal flow of execution within loops:

- **break Statement:** This statement instructs to jump out of the loop, even if the loop isn’t finished yet. It’s often used to exit a loop before the loop stops when a specific condition is met within the loop.

EX:-
```
let numbers = [1, 5, 2, 8, 3];
for (let i = 0; i < numbers.length; i++) {
  if (numbers[i] % 2 === 0) {
    console.log("Found an even number:", numbers[i]);
    break;
  }
}
```

In this example, the break statement within the if block exits the loop as soon as an even number is found.

- **continue Statement:** This statement skips the current iteration of the enclosing loop and proceeds to the next iteration. It allows you to selectively skip specific iterations based on conditions within the loop.

EX:-
```
let numbers = [1, 5, 2, 8, 3];
for (let i = 0; i < numbers.length; i++) {
  if (numbers[i] % 2 === 0) {
    continue;
  }
  console.log("Odd number:", numbers[i]);
}
```

Here, the continue statement skips any even numbers in the array, printing only the odd numbers. Jumping statements should be used with good judgment, as excessive use can make code harder to read and maintain. However, when used appropriately, they can improve the control flow of your loops.

7. What is Operator's. Example program in JavaScript?

JavaScript Operators

JavaScript Operators are symbols used to perform specific mathematical, comparison, assignment, and logical computations on operands. They are fundamental elements in JavaScript programming, allowing developers to manipulate data and control program flow efficiently. Understanding the different types of operators and how they work is important for writing effective and optimized JavaScript code.

1. **Arithmetic Operators**:
   These are used for basic mathematical calculations on numbers.
   - **Addition(+):** Adds two values.
   - **Subtraction(-):** Subtracts the second value from the first.
   - **Multiplication(*):** Multiplies two values.
   - **Division(/):** Divides the first value by the second.
   - **Modulus Division(%):** Returns the remainder when dividing.

2. **Comparison Operators**:
   These operators compare two values and return a boolean (true or false).
   - **==:** Checks if two values are equal.
   - **===:** Strict equality, checks if two values are equal and of the same type.
   - **Not Equal to(!=):** Checks if two values are not equal.
   - **Greater than(>):** Checks if the left value is greater than the right.
   - **Less than(<):** Checks if the left value is less than the right.

3. **Logical Operators**:
   These are used to perform logical operations on boolean values.
   - **AND(&&):** Logical AND, returns true only if both operands are true.
   - **OR(||):** Logical OR, returns true if at least one operand is true.
   - **NOT(!):** Logical NOT, inverts the boolean value.

4. **Assignment Operators**:
   These are used to assign values to variables.
   - **=:** Simple assignment.
   - **+=:** Adds the value to the variable.
   - **-=:** Subtracts the value from the variable.

Example Program:

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <title>JavaScript Operators Example</title>
</head>
<body>
   <h1>JavaScript Operators Example</h1>

   <script>
      // Arithmetic Operators
      let a = 15;
```

```
    let b = 5;
    let addition = a + b;        // Addition
    let subtraction = a - b;     // Subtraction
    let multiplication = a * b;  // Multiplication
    let division = a / b;        // Division
    let modulus = a % b;         // Modulus
    let exponentiation = a ** b; // Exponentiation

    console.log("Arithmetic Operators:");
    console.log("a + b = " + addition);        // 20
    console.log("a - b = " + subtraction);     // 10
    console.log("a * b = " + multiplication);  // 75
    console.log("a / b = " + division);        // 3
    console.log("a % b = " + modulus);         // 0
    console.log("a ** b = " + exponentiation); // 759375

    // Assignment Operators
    a += 10; // a = a + 10
    b -= 2;  // b = b - 2
    console.log("\nAssignment Operators:");
    console.log("New value of a (a += 10) = " + a); // 25
    console.log("New value of b (b -= 2) = " + b);  // 3

    // Comparison Operators
    let isEqual = (a === b);  // Checks if a is equal to b
    let isGreater = (a > b);  // Checks if a is greater than b
    let isNotEqual = (a != b); // Checks if a is not equal to b

    console.log("\nComparison Operators:");
    console.log("a === b: " + isEqual);  // false
    console.log("a > b: " + isGreater);  // true
    console.log("a != b: " + isNotEqual); // true

    // Logical Operators
    let isAdult = true;
    let hasPermission = false;

    let canAccess = isAdult && hasPermission; // Checks if both conditions are true
    let canAccessOr = isAdult || hasPermission; // Checks if at least one condition is true
    let notAllowed = !hasPermission; // Checks the negation of hasPermission

    console.log("\nLogical Operators:");
    console.log("isAdult && hasPermission: " + canAccess);       // false
    console.log("isAdult || hasPermission: " + canAccessOr);     // true
    console.log("!hasPermission: " + notAllowed);                // true
  </script>

</body>
</html>
```

8. Define Function in JavaScript and Validations?

Function in JavaScript

A function in JavaScript is a block of reusable code that performs a specific task. Functions help you to avoid redundancy, enhance code readability, and allow code to be modular.

Syntax for defining a function:

```
function functionName(parameters) {
    // code to be executed
}
```

- **functionName**: The name of the function.
- **parameters:** (Optional) Values that are passed into the function to work with. They are also known as arguments.
- **code to be executed:** The block of code that runs when the function is called.

Example of a basic function:

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

// Calling the function
greet("Alice"); // Output: Hello, Alice!
greet("Bob");   // Output: Hello, Bob!
```

Types of Functions in JavaScript:

1. **Function Declaration:** A standard way of defining a function.

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

2. **Function Expression:** Functions are defined and assigned to variables.

```
const multiply = function(a, b) {
return a * b;
};
```

3. **Arrow Function :** A shorter syntax for writing functions.

```
const divide = (a, b) => a / b;
```

4. **Anonymous Function:** Functions that do not have a name, typically used in function expressions.

```
setTimeout(function() {
console.log("This is an anonymous function!");
}, 1000);
```

Validations in JavaScript

Validations are important in JavaScript to ensure that data is correct and that errors are prevented. For example, checking if a user input is in the correct format or meets certain criteria.

Common Types of Validations:

1. **String Validation:** Checking if a string is not empty.

```
function isNotEmpty(input) {
    if (input.trim() === "") {
        console.log("Input cannot be empty.");
        return false;
    } return true;
```

```
}
console.log(isNotEmpty("Hello")); // true
console.log(isNotEmpty(""));      // false
```

2. **Email Validation:** Checking if an email address is in the correct format using Regular Expressions (RegEx).

```
function validateEmail(email) {
    const regex = /^[a-zA-Z0-9._-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,4}$/;
    if (regex.test(email)) {
        console.log("Valid email address.");
    } else {
        console.log("Invalid email address.");
    }
}
validateEmail("test@example.com");
validateEmail("invalid-email");
```

3. **Number Validation:** Checking if an input is a valid number.

```
function isNumber(input) {
    if (isNaN(input)) {
        console.log("Input is not a number.");
        return false;
    }
    return true;
}

console.log(isNumber("123"));  // true
console.log(isNumber("abc"));  // false
```

4. **Password Strength Validation:** Checking if a password meets the criteria (length, special characters, etc.).

```
function validatePassword(password) {
    const regex = /^(?=.*[A-Za-z])(?=.*\d)[A-Za-z\d@$!%*?&]{8,}$/;
    if (regex.test(password)) {
        console.log("Password is strong.");
    } else {
        console.log("Password must be at least 8 characters long, include a letter and a
number.");
    }
}

validatePassword("Strong1Password");  // Password is strong.
validatePassword("weak");
```

5. **Form Validation:** Checking multiple fields in a form.

```
function validateForm() {
    const name = document.getElementById("name").value;
    const email = document.getElementById("email").value;

    if (!isNotEmpty(name) || !validateEmail(email)) {
        alert("Please fill out the form correctly.");
        return false; }   return true; }
```

Example Program:

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <title>Form Validation Example</title>
</head>
<body>
   <h2>Signup Form</h2>
   <form onsubmit="return validateForm()">
      <label for="name">Name:</label><br>
      <input type="text" id="name" name="name"><br><br>

      <label for="email">Email:</label><br>
      <input type="text" id="email" name="email"><br><br>

      <label for="password">Password:</label><br>
      <input type="password" id="password" name="password"><br><br>

      <input type="submit" value="Submit">
   </form>
   <script>
      function validateForm() {
         const name = document.getElementById("name").value;
         const email = document.getElementById("email").value;
         const password = document.getElementById("password").value;

         // Validate Name
         if (name.trim() === "") {
            alert("Name is required.");
            return false;
         }
         // Validate Email
         const emailRegex = /^[a-zA-Z0-9._-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,4}$/;
         if (!emailRegex.test(email)) {
            alert("Please enter a valid email.");
            return false;
         }
         // Validate Password
         const passwordRegex = /^(?=.*[A-Za-z])(?=.*\d)[A-Za-z\d@$!%*?&]{8,}$/;
         if (!passwordRegex.test(password)) {
            alert("Password must be at least 8 characters long, and contain at least one letter and one
number.");
            return false;
         }

         alert("Form submitted successfully!");
         return true;  // Allow form submission
      }
   </script>
</body>
</html>
```