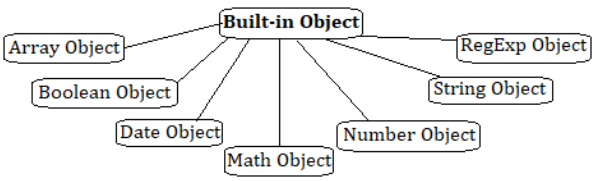
MODULE 2: Front End Development(JAVASCRIPT)

JAVASCRIPT

Q.1) Explain built-in objects in JavaScript.

⇒



Built-in Objects in JavaScript

JavaScript has several built-in objects that provide powerful methods and properties to perform various tasks. These objects are part of the JavaScript language, and you don't need to create them; they are readily available for use.

Common Built-in Objects in JavaScript

1. **String**:
   - Used to work with text data.
   - Provides methods like `.length`, `.toUpperCase()`, `.toLowerCase()`, and `.slice()`.
   - Example:

```
let greeting = "Hello, World!";
console.log(greeting.toUpperCase()); // Output: "HELLO, WORLD!"
```

2. **Number**:
   - Used to perform calculations on numerical values.
   - Includes properties like `MAX_VALUE` and methods like `toFixed()`, which rounds a number.
   - Example:

```
let num = 123.456;
console.log(num.toFixed(2)); // Output: "123.46"
```

3. **Array**:
   - Used to store collections of data in a single variable.
   - Provides methods like `.push()`, `.pop()`, `.map()`, and `.filter()`.
   - Example:

```
let fruits = ["apple", "banana", "cherry"];
fruits.push("orange");
console.log(fruits); // Output: ["apple", "banana", "cherry", "orang
e"]
```

4. **Math**:
   - Provides mathematical functions and constants.

- Includes methods like `Math.random()`, `Math.floor()`, and `Math.max()`.
- Example:

```
let randomNum = Math.random(); // Generates a random number between 0 and 1
console.log(randomNum);
```

5. **Date**:
   - Used to work with dates and times.
   - Has methods like `.getFullYear()`, `.getMonth()`, `.getDate()`, and `.getTime()`.
   - Example:

```
let today = new Date();
console.log(today.getFullYear()); // Output: current year (e.g., 2023)
```

6. **JSON** (JavaScript Object Notation):
   - Used to work with JSON data.
   - Provides methods like `JSON.parse()` to convert JSON strings to objects and `JSON.stringify()` to convert objects to JSON strings.
   - Example:

```
let jsonData = '{"name": "Alice", "age": 25}';
let user = JSON.parse(jsonData);
console.log(user.name); // Output: "Alice"
```

Each of these objects offers a set of properties and methods to work with different types of data in JavaScript, making it easier to perform tasks like text manipulation, calculations, array operations, and more.

Q.2) explain document object model in details. [vvip]

⇒

Document Object Model (DOM) in JavaScript

The Document Object Model (DOM) is a programming interface for HTML and XML documents. It represents the structure of a webpage, allowing JavaScript to interact with and modify HTML elements, attributes, and styles.

Structure of the DOM

The DOM represents the webpage as a tree structure where each element, attribute, and text in the HTML document becomes a node. This tree-like structure allows JavaScript to access and change the document's content, structure, and styling dynamically.

Example HTML:

```
<!DOCTYPE html>
<html>
<head>
  <title>DOM Example</title>
</head>
<body>
  <h1>Hello World!</h1>
  <p>This is a sample paragraph.</p>
</body>
</html>
```

In the DOM, this HTML is structured as a hierarchy:

```
- Document
 - html
  - head
   - title ("DOM Example")
  - body
   - h1 ("Hello World!")
   - p ("This is a sample paragraph.")
```

DOM Methods and Properties

Using JavaScript, we can access and modify the DOM:

1. **Accessing Elements**:
    - `.getElementById()`: Selects an element by its ID.

```
document.getElementById("exampleId");
```

- `.getElementsByClassName()`: Selects elements by their class.
- `.getElementsByTagName()`: Selects elements by their tag name.
- `.querySelector()` / `.querySelectorAll()`: Selects elements using CSS selectors.

2. **Modifying Elements**:
   - `.innerHTML`: Changes or retrieves the HTML inside an element.

```
let paragraph = document.getElementById("myParagraph");
paragraph.innerHTML = "New text content!";
```

   - `.style`: Changes CSS styles of an element.

```
paragraph.style.color = "blue";
```

3. **Creating and Removing Elements**:
   - `createElement()`: Creates a new HTML element.
   - `appendChild()`: Adds a new element as a child.
   - `removeChild()`: Removes an existing element.

Example: Manipulating the DOM

Here's an example of a button that, when clicked, changes the text and color of a paragraph.

```
<!DOCTYPE html>
<html>
<head>
  <title>DOM Manipulation Example</title>
</head>
<body>

<h1 id="heading">Welcome to My Page</h1>
<p id="myParagraph">This is the original text.</p>
<button onclick="changeContent()">Click Me</button>
```

```
<script>
   function changeContent() {
      let paragraph = document.getElementById("myParagraph");
      paragraph.innerHTML = "Text has been changed!";
      paragraph.style.color = "green";
   }
</script>

</body>
</html>
```

In this example:

- `getElementById("myParagraph")` selects the paragraph.
- `innerHTML` changes the text inside the paragraph.
- `style.color` modifies the color to green.

The DOM enables JavaScript to interact with HTML documents, making them dynamic and interactive.

Q.3) explain Exception handling in Java script with suitable example.

⇒

Exception Handling in JavaScript

Exception handling in JavaScript allows you to manage errors gracefully so the code can recover or handle issues without breaking. JavaScript provides the `try`, `catch`, `finally`, and `throw` statements to handle exceptions.

Key Parts of Exception Handling

- The `try` block contains code that may throw an error.

1. **try**:

2. **catch**:
    - The `catch` block executes if an error occurs in the `try` block.
3. **finally**:
    - The `finally` block runs after the `try` and `catch` blocks, regardless of whether an error occurred. It's useful for cleanup code.
4. **throw**:
    - The `throw` statement allows you to create custom errors and throw them.

Example of Exception Handling in JavaScript

```
try {
    let num = 10;
    if (num > 5) {
        throw "Number is too high"; // Custom error thrown
    }
    console.log("Number is fine");
} catch (error) {
    console.log("An error occurred: " + error); // Catches the error and displa
ys a message
} finally {
    console.log("This code always runs, regardless of errors.");
}
```

In this example:

- The `throw` statement raises a custom error if `num` is greater than 5.
- `catch` captures and logs the error message.
- `finally` executes at the end, ensuring that cleanup or final tasks are completed even if an error occurred.

Benefits of Exception Handling:

- **Error Management**: Prevents code from crashing and allows handling errors systematically.
- **Debugging**: Helps identify and log issues in the code for debugging.

- **Clean Code Flow**: Ensures code runs smoothly, improving user experience by managing unexpected errors.

Q.4) explain the event handling in Java script with suitable example.

⇒

Event handling in JavaScript allows you to detect and respond to various events triggered by user interactions (like clicks, key presses, mouse movements) or by the browser itself. JavaScript offers built-in event listeners and functions to manage these events.

Common JavaScript Events

- **onclick**: Triggered when an element is clicked.
- **onmouseover**: Triggered when the mouse pointer hovers over an element.
- **onkeypress**: Triggered when a key is pressed.
- **onload**: Triggered when the page or an element finishes loading.

Example of Event Handling in JavaScript

Let's create a button that displays an alert message when clicked.

```
<!DOCTYPE html>
<html>
<head>
  <title>Event Handling Example</title>
</head>
<body>

<button onclick="showMessage()">Click Me</button>

<script>
  function showMessage() {
    alert("Button was clicked!");
  }
```

```
</script>

</body>
</html>
```

In this example:

- The `onclick` event is attached to the button.
- When the button is clicked, the `showMessage()` function is executed, displaying an alert box.

Adding Event Listeners with `addEventListener()`

The `addEventListener()` method allows attaching multiple event listeners to an element and offers more flexibility.

Example:

```
<!DOCTYPE html>
<html>
<head>
    <title>Event Handling with addEventListener</title>
</head>
<body>

<button id="myButton">Click Me</button>

<script>
    let button = document.getElementById("myButton");

    // Adds a click event listener to the button
    button.addEventListener("click", function() {
        alert("Button was clicked using addEventListener!");
    });
</script>

</body>
```

```
</html>
```

Here, `addEventListener("click", ...)` listens for the click event on the button, and the anonymous function inside displays an alert.

Q.5) explain how JavaScript can hide HTML elements with suitable example

⇒

JavaScript allows you to hide HTML elements dynamically using the `.style` property. You can set the `display` or `visibility` property of an element to hide it.

Methods to Hide Elements

1. **Using `display` Property**:
   - Setting `display: "none"` removes the element from the page layout.
2. **Using `visibility` Property**:
   - Setting `visibility: "hidden"` makes the element invisible but keeps its space reserved on the page.

Example of Hiding an Element with JavaScript

Let's hide a paragraph when a button is clicked.

```
<!DOCTYPE html>
<html>
<head>
  <title>Hide Element Example</title>
</head>
<body>

<p id="myParagraph">This is a paragraph that will be hidden.</p>
<button onclick="hideParagraph()">Hide Paragraph</button>

<script>
```

```
    function hideParagraph() {
        let paragraph = document.getElementById("myParagraph");
        paragraph.style.display = "none"; // Hides the paragraph
    }
</script>

</body>
</html>
```

In this example:

- The `onclick` event on the button triggers the `hideParagraph()` function.
- Inside `hideParagraph()`, `style.display = "none"` hides the paragraph element entirely.

Example of Hiding with `visibility`

If you want the space of the hidden element to remain, use `visibility: "hidden"`:

```
paragraph.style.visibility = "hidden";
```

This makes the element invisible but reserves its space on the page, which can be useful for UI layouts where spacing matters.

CODE

Q.6) write a code to process online alumni information for your college. Create forms to get name, address, date of birth and email ID. Write Java script code to validate the following.

i) user has filled all the fields prior to form submission

ii) valid email ID(with “@” and “.”)

iii) age validation on using DOB(≥22 years).

⇒

Here's a complete implementation of a form to process online alumni information, including HTML for the form and JavaScript for validation based on the specified criteria.

HTML and JavaScript Code

HTML File: `alumni_form.html`

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.
0">
    <title>Alumni Information Form</title>
    <script src="alumni_script.js" defer></script>
    <style>
        body {
            font-family: Arial, sans-serif;
            background-color: #f4f4f4;
            padding: 20px;
        }
        form {
            background: white;
            padding: 20px;
            border-radius: 5px;
            box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);
            max-width: 400px;
            margin: auto;
        }
        input[type="text"], input[type="email"], input[type="date"] {
            width: 100%;
            padding: 10px;
            margin: 10px 0;
            border: 1px solid #ccc;
            border-radius: 4px;
        }
```

```
        input[type="submit"] {
            background-color: #5cb85c;
            color: white;
            border: none;
            padding: 10px;
            cursor: pointer;
            border-radius: 4px;
        }
        input[type="submit"]:hover {
            background-color: #4cae4c;
        }
    </style>
</head>
<body>

<h2>Alumni Information Form</h2>
<form id="alumniForm" onsubmit="return validateAlumniForm()">
    <label for="name">Name:</label>
    <input type="text" id="name" required>

    <label for="address">Address:</label>
    <input type="text" id="address" required>

    <label for="dob">Date of Birth:</label>
    <input type="date" id="dob" required>

    <label for="email">Email ID:</label>
    <input type="email" id="email" required>

    <input type="submit" value="Submit">
</form>

</body>
</html>
```

JavaScript File: `alumni_script.js`

```
function validateAlumniForm() {
    const name = document.getElementById("name").value;
    const address = document.getElementById("address").value;
    const dob = new Date(document.getElementById("dob").value);
    const email = document.getElementById("email").value;

    // Check if all fields are filled (handled by 'required' attribute in HTML)
    if (!name || !address || !dob || !email) {
        alert("Please fill all the fields.");
        return false;
    }

    // Validate email
    const emailPattern = /^[^\\s@]+@[^\\s@]+\\.[^\\s@]+$/;
    if (!emailPattern.test(email)) {
        alert("Please enter a valid email ID.");
        return false;
    }

    // Calculate age
    const today = new Date();
    let age = today.getFullYear() - dob.getFullYear();
    const monthDiff = today.getMonth() - dob.getMonth();
    if (monthDiff < 0 || (monthDiff === 0 && today.getDate() < dob.getDate
())) {
        age--;
    }

    // Validate age
    if (age < 22) {
        alert("Age must be 22 years or older.");
        return false;
    }

    alert("Form submitted successfully!");
    return true;
}
```

Q.7) create a form which has the following fields name, age,email ID, password. Using Java script validated each field as follows.

i) name should be between A-Z

ii) age should be between 0 to 100.

iii) email ID should contain “@”

iv) password should contain 1 Uppercase, 1 digit, 1 special character and length should be minimum 8.

⇒

Form Validation for Name, Age, Email ID, and Password

HTML File: `form_validation.html`

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.
0">
    <title>Form Validation</title>
    <script src="form_validation_script.js" defer></script>
</head>
<body>

<h2>Form Validation</h2>
<form id="validationForm" onsubmit="return validateForm()">
    Name: <input type="text" id="name" required><br><br>
    Age: <input type="number" id="age" required><br><br>
    Email ID: <input type="email" id="email" required><br><br>
    Password: <input type="password" id="password" required><br><br>
    <input type="submit" value="Submit">
</form>
```

```
</body>
</html>
```

JavaScript File: `form_validation_script.js`

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

    // Validate name (only A-Z)
    if (!/^[A-Z]+$/.test(name)) {
        alert("Name should contain only alphabets (A-Z ).");
        return false;
    }

    // Validate age
    if (age < 0 || age > 100) {
        alert("Age should be between 0 and 100.");
        return false;
    }

    // Validate email
    if (!email.includes("@")) {
        alert("Email should contain '@' symbol.");
        return false;
    }

    // Validate password
    const passwordPattern = /^(?=.*[A-Z])(?=.*\d)(?=.*[!@#$%^&*])[A-Za-z
\d!@#$%^&*]{8,}$/;
    if (!passwordPattern.test(password)) {
        alert("Password must contain 1 uppercase letter, 1 digit, 1 special chara
cter, and be at least 8 characters long.");
```

```
        return false;
    }

    alert("Form submitted successfully!");
    return true;
}
```

Q.) write a Java script to check password and confirm password same or not.

⇒

Password and Confirm Password Validation

HTML File: confirm_password.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.
0">
    <title>Confirm Password</title>
    <script src="confirm_password_script.js" defer></script>
</head>
<body>

<h2>Check Password</h2>
<form id="passwordForm" onsubmit="return checkPasswords()">
    Password: <input type="password" id="password" required><br><br>
    Confirm Password: <input type="password" id="confirmPassword" requi
red><br><br>
    <input type="submit" value="Submit">
</form>

</body>
```

```
</html>
```

JavaScript File: `confirm_password_script.js`

```
function checkPasswords() {
    const password = document.getElementById("password").value;
    const confirmPassword = document.getElementById("confirmPasswor
d").value;

    if (password !== confirmPassword) {
        alert("Passwords do not match.");
        return false;
    }

    alert("Passwords match!");
    return true;
}
```

Q.8) write a Java script code to accept a name and password from user and validate the data as follows.

i) name should not be empty.

ii) password should not be less than six characters

⇒

Name and Password Validation

HTML File: `name_password_form.html`

```
html
Copy code
<!DOCTYPE html>
```

```
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.
0">
    <title>Name and Password Validation</title>
    <script src="name_password_script.js" defer></script>
</head>
<body>

<h2>Name and Password Validation</h2>
<form id="namePasswordForm" onsubmit="return validateNamePassword
Form()">
    Name: <input type="text" id="name" required><br><br>
    Password: <input type="password" id="password" required><br><br>
    <input type="submit" value="Submit">
</form>

</body>
</html>
```

JavaScript File: `name_password_script.js`

```
javascript
Copy code
function validateNamePasswordForm() {
    const name = document.getElementById("name").value;
    const password = document.getElementById("password").value;

    // Validate name
    if (!name) {
        alert("Name should not be empty.");
        return false;
    }

    // Validate password length
    if (password.length < 6) {
```

```
        alert("Password should be at least six characters.");
        return false;
    }

    alert("Form submitted successfully!");
    return true;
}
```

Q.9) Write a Java script that reads 10 numbers and display the count of negative numbers, the count of positive number and the count of zero from the list.

⇒

Count of Positive, Negative, and Zero Numbers

HTML File: `count_numbers.html`

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.
0">
    <title>Count Numbers</title>
    <script src="count_numbers_script.js" defer></script>
</head>
<body>

<h2>Count Positive, Negative, and Zero Numbers</h2>
<form id="numberForm" onsubmit="return countNumbers()">
    <label>Enter 10 numbers (comma separated): </label>
    <input type="text" id="numbers" required><br><br>
    <input type="submit" value="Count">
```

```
</form>

<p id="result"></p>

</body>
</html>
```

JavaScript File: `count_numbers_script.js`

```
function countNumbers() {
    const numbers = document.getElementById("numbers").value.split(",").
map(Number);

    if (numbers.length !== 10) {
        alert("Please enter exactly 10 numbers.");
        return false;
    }

    let positiveCount = 0;
    let negativeCount = 0;
    let zeroCount = 0;

    numbers.forEach(num => {
        if (num > 0) positiveCount++;
        else if (num < 0) negativeCount++;
        else zeroCount++;
    });

    document.getElementById("result").innerText = `Positive: ${positiveCoun
t}, Negative: ${negativeCount}, Zero: ${zeroCount}`;
    return false; // Prevent form submission
}
```