

BCA/IMCA SEMESTER 1

Fundamentals of Web Technology (BC01001031)

1. Introduction to JavaScript

What is JavaScript?

- **JavaScript** is a high-level, interpreted programming language used primarily to create interactive effects and dynamic content in web browsers.
- It is a **client-side scripting language**, which means it runs directly in the browser, as opposed to server-side languages (like PHP or Node.js), which run on the server.

Core Features:

- **Dynamic typing:** JavaScript doesn't require declaring types for variables. For example, a variable can store a string at one moment and a number the next.
- **Event-driven:** JavaScript enables responsive user interfaces by responding to events such as button clicks, mouse movements, and form submissions.
- **Single-threaded:** JavaScript executes one instruction at a time but uses asynchronous functions like `setTimeout()` or `fetch()` to handle concurrency (i.e., tasks running in the background).

Example:

```
// A simple example
console.log("Hello, JavaScript!");
```

2. Basic Syntax

Syntax Rules:

- **Statements:** Instructions written in JavaScript to be executed.
 - Example: `let x = 5;`
- **Semicolons:** While JavaScript often auto-inserts semicolons, it's good practice to manually place them to avoid errors.
- **Block of code:** A block of code is enclosed in curly braces `{ }`. Blocks are often used in functions, loops, and conditionals.

Example:

```
if (x > 10) {  
  console.log("x is greater than 10");  
} else {  
  console.log("x is less than or equal to 10");  
}
```

3. Variables and Identifiers

Variables:

- **var**: Function-scoped (hoisted to the top of the function). It can be re-declared and updated.
- **let**: Block-scoped, and it's preferred for modern JavaScript due to its predictable behavior.
- **const**: Also block-scoped, but the variable cannot be reassigned after its initialization.

Best Practice:

- Always use **let** or **const** to avoid issues related to **var** (like unintentional re-declarations or hoisting).

Example:

```
let name = 'John'; // Can be reassigned  
const pi = 3.14; // Cannot be reassigned
```

Identifiers:

- Identifiers are names used for variables, functions, etc. They must start with a letter, \$, or , and can include numbers after the first character.
 - JavaScript has **reserved keywords** that can't be used as identifiers (e.g., `return`, `function`, `class`).
-

4. Data Types and Values

JavaScript has two categories of data types:

- **Primitive Data Types:**
 - **String**: A sequence of characters enclosed in single or double quotes.
 - **Number**: Can represent integers or floating-point numbers.
 - **Boolean**: Represents `true` or `false`.
 - **Null**: A special type that represents the intentional absence of any object value.
-

- **Undefined:** A variable that has been declared but not assigned a value.
 - **Symbol:** Used for unique identifiers (new in ES6).
- **Reference Data Types:**
 - **Object:** A collection of key-value pairs.
 - **Array:** A list-like object that can hold multiple values.

Example:

```
let name = 'Alice'; // String
let age = 30;       // Number
let isActive = true; // Boolean
let person = {name: 'Alice', age: 30}; // Object
```

5. Scope

What is Scope?

Scope refers to the accessibility of variables, functions, and objects in certain parts of the code during execution.

- **Global Scope:** Variables declared outside any function or block are globally accessible.
- **Local Scope:** Variables declared inside a function are only accessible within that function.
- **Block Scope:** Introduced with `let` and `const`, variables declared inside a block (e.g., within `{ }`) are confined to that block.

Example:

```
let globalVar = "I am global"; // Global scope

function testScope() {
  let localVar = "I am local"; // Local scope
  if (true) {
    let blockVar = "I am block scoped"; // Block scope
    console.log(blockVar); // Accessible here
  }
  console.log(localVar); // Accessible here
  // console.log(blockVar); // Error: blockVar is not defined
}

testScope();
```

6. Literals

What Are Literals?

Literals are fixed values used directly in code to represent data.

- **String Literal:** A sequence of characters enclosed in quotes.
- **Number Literal:** A numeric value like 10 or 3.14.
- **Boolean Literal:** The values `true` or `false`.
- **Object Literal:** A collection of key-value pairs.
- **Array Literal:** A collection of values in square brackets.

Example:

```
let person = {name: 'John', age: 30}; // Object literal
let numbers = [1, 2, 3, 4];           // Array literal
```

7. Reserved Words

Reserved words are keywords that JavaScript has predefined and cannot be used as identifiers (e.g., variable names).

Examples of reserved words:

- **Control flow:** `if`, `else`, `for`, `while`, `switch`
 - **Data types:** `null`, `true`, `false`
 - **Functions:** `return`, `function`, `yield`
 - **Others:** `class`, `let`, `const`, `new`
-

8. Operators and Statements

Operators:

- **Arithmetic Operators:** Used to perform mathematical operations.
 - `+` (addition), `-` (subtraction), `*` (multiplication), `/` (division), `%` (modulus)
- **Comparison Operators:** Used to compare two values.
 - `==` (loose equality), `===` (strict equality), `!=` (inequality), `>` (greater than), `<` (less than)
- **Logical Operators:** Used to combine multiple expressions.
 - `&&` (AND), `||` (OR), `!` (NOT)
- **Assignment Operators:** Used to assign values to variables.
 - `=`, `+=`, `-=`, `*=`, `/=`
- **Ternary Operator:** A shorthand for `if-else`.
 - `let result = (x > 10) ? 'Greater' : 'Smaller';`

Control Flow:

- **`if`, `else if`, `else`:** Conditional statements that run code based on certain conditions.
-

- **switch:** A more readable alternative to multiple `if` statements for handling many conditions.

Example:

```
let x = 20;
if (x > 10) {
  console.log("Greater than 10");
} else {
  console.log("10 or less");
}
```

9. Functions

Defining Functions:

Functions are blocks of code designed to perform a particular task.

- **Function Declaration:** Can be called before or after it's defined.

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

- **Function Expression:** A function assigned to a variable.

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

- **Arrow Functions (ES6):** A shorter syntax for writing functions, especially useful for anonymous functions.

```
const greet = () => console.log("Hello!");
```

- **Parameters & Return Values:** Functions can accept parameters and return values.

```
function divide(a, b) {
  return a / b;
}
let result = divide(10, 2); // 5
```

10. Objects

JavaScript Objects:

Objects in JavaScript are collections of key-value pairs.

- **Object Creation:**

```
let person = {
  name: 'John',
  age: 30,
  greet: function() {
    console.log('Hello');
  }
};
person.greet(); // Outputs 'Hello'
```

- **Built-in Objects:**

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

```
console.log(Math.PI); // 3.14159...
console.log(Math.random()); // Random number between 0 and 1
```

- **String Object:** Provides methods for manipulating strings.

```
let str = "Hello";
console.log(str.length); // 5
console.log(str.toUpperCase()); // "HELLO"
```

- **Date Object:** Handles date and time.

```
let date = new Date();
console.log(date.getFullYear()); // Current year
```

11. Regular Expressions

What is a Regular Expression (Regex)?

A regular expression is a pattern used to match character combinations in strings. It's a powerful tool for searching, replacing, and validating input.

- **Basic Syntax:**

- `/pattern/`: A simple regex pattern.
- **Special characters:** `.` (any character), `*` (zero or more), `+` (one or more), `?` (optional), `^` (start of string), `$` (end of string)

Example:

```
let regex = /hello/i; // 'i' for case-insensitive matching
let testString = 'Hello, World!';
console.log(regex.test(testString)); // true
```

12. DOM Event Handling

What is the DOM (Document Object Model)?

The DOM represents the structure of the HTML document as a tree of objects. JavaScript allows us to manipulate the DOM to change the content and structure of a webpage dynamically.

Event Handling:

- **Event Listeners:** Functions that wait for specific events (like clicks, keypresses) to happen and then execute when the event occurs.

```
document.getElementById('btn').addEventListener('click', function() {  
    alert('Button clicked!');  
});
```

- **Event Object:** Contains information about the event, such as the target element and the type of event.

```
document.getElementById('btn').addEventListener('click', function(event) {  
    console.log(event.target); // Logs the element clicked  
});
```

This more detailed explanation should give you a much deeper understanding of each topic. You can try implementing each section with your own code and experimenting with different examples to further solidify your learning!