Module 9 Lesson 1

What is JavaScript?

JavaScript is a versatile, high-level programming language that is primarily known for its role in web development. Here's a detailed breakdown of its features, characteristics, and capabilities:

1. History & Background:

- Origin: Developed by Brendan Eich in 1995 while working at Netscape.
- Initial Purpose: Initially designed to add interactivity to websites during the early days of the web.
- Name Evolution: Originally named Mocha, then LiveScript, and finally renamed to JavaScript.

2. Characteristics:

- Interpreted Language: Doesn't need to be compiled. Browsers read and execute it directly.
- Dynamic: Variables in JavaScript can hold any type of value without explicit type declaration.
- Weakly Typed: Variables' data types are automatically converted as-needed (e.g., string to number).

3. Execution Environment:

- Browsers: Originally designed to run in web browsers, making web pages dynamic and interactive.
- Server-side: With the advent of platforms like Node.js, JavaScript can also be executed server-side.

4. Capabilities:

- DOM Manipulation: Can interact with the Document Object Model (DOM) to update website content, structure, and styles dynamically.
- Event Handling: Responds to user actions (like clicks or key presses).

5. Frameworks & Libraries:

- Over the years, numerous libraries (like jQuery) and frameworks (like Angular, React, and Vue) have been developed to simplify and enhance JavaScript development.

6. Evolution & Standards:

- JavaScript evolves through versions standardized by Ecma International in the ECMA-262 specification.
- It is sometimes referred to as ECMAScript because of this standardization.

7. Versatility:

- Full-stack Development: With technologies like Node.js for the backend and various frontend frameworks, it's possible to use JavaScript for both server-side and client-side development.
- Databases: Even databases like MongoDB use JavaScript for querying.

8. Popularity & Community:

- Most Used: It's one of the core technologies of the web, alongside HTML and CSS.
- Large Community: Due to its widespread use, there's a vast community, which means extensive support, abundant resources, and a multitude of third-party tools.

Why learn JavaScript?

- Popularity: Most used programming language for web development.
- Versatility: Can be used both client-side and server-side (thanks to Node.js).
- Community: Large, active developer community with abundant resources for learning and troubleshooting.
- Job Opportunities: Third most in-demand programming language as of the end of 2022.

JavaScript Versions:

- Standardized by Ecma International in ECMA-262.

    - Hence, sometimes referred to as ECMAScript.
- 13 ECMA editions as of the content's last update.
- JavaScript features are added feature-by-feature, not necessarily by ECMA edition.
    - Important to know which features are supported enough for production.
- Resources for checking feature support:
    - ECMAScript Compatibility Table.
    - Can I use?
- Most widely supported version: ES6 (a.k.a. EMCAScript 2015 or ES2015).

2. Setting up a Basic JavaScript Development Environment:

1. Embedded (or Internal) JavaScript:

How:

You place the JavaScript code directly within the HTML document using the `<script>` tags.

Example:

```
<!DOCTYPE html>
<html>
<head>
    <title>Embedded JS Example</title>
</head>
<body>
    <h1>Welcome to my website</h1>
    <script>
        alert("This is embedded JavaScript");
    </script>
</body>
</html>
```

When to Use:

- When the JavaScript is specific to a single page and won't be reused elsewhere.
- When you want to keep everything in one file for simplicity.

Considerations:

- Can increase the file size of the HTML document, which might affect loading times if the scripts are large.
- Can make the HTML document messy and harder to maintain if the scripts are extensive.

2. Inline JavaScript:

How:

You add JavaScript directly within HTML tags using event attributes like `onclick`, `onmouseover`, etc.

Example:

```
<button onclick="alert('You clicked the button!');">Click Me</button>
```

When to Use:

- For simple, one-off interactions that don't require a lot of code.
- For quick prototyping.

Considerations:

- Can make the HTML messy if overused.
- Not recommended for complex functionality as it mixes structure (HTML) with behavior (JavaScript).
- Can be a security risk due to the potential for inline event handlers to execute malicious scripts.

3. External JavaScript:

How:

You place the JavaScript code in a separate `.js` file and then link to that file from your HTML document using the `<script>` tag with the `src` attribute.

Example:

HTML File:

```
<!DOCTYPE html>
<html>
```

```
<head>
    <title>External JS Example</title>
</head>
<body>
    <h1>Welcome to my website</h1>
    <script src="externalScript.js"></script>
</body>
</html>
```

externalScript.js:

```
alert("This is external JavaScript");
```

When to Use:

- When the same script is used across multiple pages.
- For better organization and separation of concerns.

Considerations:

- Offers better maintainability and scalability.

Where to Put the `<script>` Tag and Why:

1. In the `<head>`:
    - The browser will load and execute the script before rendering the HTML.
    - This might delay page rendering, making the page appear slow.
    - Useful when the script needs to run before any part of the page is rendered (e.g., feature detection).
2. At the end of the `<body>`:
    - The browser will parse and display the HTML first, then load and execute the script.
    - This approach makes the page content visible to the user more quickly.
    - Most commonly recommended for scripts that manipulate the DOM or rely on the document being fully parsed.

Module 9 Lesson 2

Introduction to Variables in JavaScript:

Variables:

Definition: A variable is a symbolic name for a value. Variables are used to store data, which can be referenced and manipulated throughout a program.

Declaration: In JavaScript, variables can be declared using `var`, `let` (ES6), `const` (ES6).

```
var name;
let age;
const PI = 3.14;
```

---

1. Difference between `var`, `let`, and `const`:

`var`:

- Scope: Function-scoped.
- Use Case: Prior to ES6, `var` was the only way to declare variables in JavaScript. It's less used now in favor of `let` and `const`.

`let`:

- Scope: Block-scoped (e.g., inside loops or conditionals).
- Use Case: When you need a variable that might be reassigned.

`const`:

- Scope: Block-scoped.
- Re-assignment: Cannot reassign a `const` variable after it's been assigned.
- Use Case: When you have a variable whose value should not change.

---

2. CamelCase:

- Definition: A naming convention in which the first word is lowercase, and each subsequent word or abbreviation begins with an uppercase character.
- Example: `myVariableName`, `getUserInfo`, `calculateTotalAmount`.
- Use in JavaScript: Commonly used for variable names, function names, and object properties.

---

3. Declaring and Initializing Variables:

Declaration: Stating that a variable exists, optionally specifying its type.

```
let age;
const birthYear;
```

Initialization: Assigning a value to a variable for the first time.

```
age = 25;
const birthYear = 1997;  // Here, declaration and initialization happen at the same time.
```

---

4. Concatenation with Strings and Variables:

Concatenation: The process of combining strings.

Using the `+` Operator:

```
let greeting = "Hello, " + "World!";
let name = "Alice";
console.log("Hi, " + name + "!");  // Outputs: "Hi, Alice!"
```

Using Template Literals (ES6) ***(will learn in a later lesson):***

```
console.log(`Hi, ${name}!`);  // Outputs: "Hi, Alice!"
```

---

5. Index in Arrays:

Arrays in JavaScript are zero-indexed.

The first element is at index `0`, the second at index `1`, and so on.

```
let fruits = ["apple", "banana", "cherry"];
console.log(fruits[0]);  // Outputs: "apple"
```

---

6. Object Properties:

Objects in JavaScript are collections of key-value pairs.

Keys (often called properties) are unique strings, while values can be any data type.

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

Accessing Properties:

Using dot notation: `person.firstName` would give "John".

Using bracket notation: `person["lastName"]` would give "Doe".

Data Types in JavaScript:

Primitive Data Types:

Number: Represents both integers and floating-point numbers.

```
let num1 = 5;
```

```
let num2 = 5.89;
```

String: Represents a sequence of characters.

```
let greeting = "Hello, World!";
```

Boolean: Represents a true or false value.

```
let isTrue = true;
let isFalse = false;
```

Undefined: Indicates that a variable has been declared but hasn't been assigned a value yet.

```
let something;
console.log(something);  // Outputs: undefined
```

Null: Represents a deliberate absence of any value or object.

```
let emptyValue = null;
```

Complex Data Types:

Object: Used to store collections of data and more complex entities.

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

Array: Represents a list-like collection of values.

```
let colors = ["Red", "Blue", "Green"];
```

Function: A block of reusable code.

```
function greet() {
  console.log("Hello there!");
```

```
}
```

Commenting Code in JavaScript:

Single-Line Comments: Use `//` to begin a single-line comment.

```
// This is a single-line comment.
let x = 5;  // This is an inline comment.
```

Multi-Line Comments: Use `/*` to begin a multi-line comment and `*/` to end it.

```
/*
This is a multi-line comment.
It spans multiple lines.
*/
let y = 10;
```

Module 9 Lesson 3

Basic Operators in JavaScript:

1. Arithmetic Operators:

Addition (`+`): Adds two numbers.

```
5 + 3;  // Outputs: 8
```

Subtraction (`-`): Subtracts the right-hand operand from the left-hand operand.

```
5 - 3;  // Outputs: 2
```

Multiplication (`*`): Multiplies two numbers.

```
5 * 3;  // Outputs: 15
```

Division (`/`): Divides the left-hand operand by the right-hand operand.

```
5 / 3;  // Outputs: 1.666...
```

Remainder (or Modulus) (`%`): Returns the remainder after division.

```
5 % 3;  // Outputs: 2
```

Exponentiation (`**`): Raises the left operand to the power of the right operand.

```
5 ** 3;  // Outputs: 125
```

2. Order of Operations:

- Follows the PEMDAS/BODMAS rule:
    - P/B: Parentheses/Brackets
    - E/O: Exponents/Orders (i.e., powers and square roots, etc.)
    - MD: Multiplication and Division (from left to right)

    - AS: Addition and Subtraction (from left to right)

3. Type Coercion:

The process where the data type of a value is implicitly changed to another data type.

```
"5" + 3;  // Outputs: "53" (number 3 is coerced to a string)
```

4. Comparison Operators:

Equal (`==`): Checks if two values are equal after type coercion.

```
5 == "5";  // Outputs: true
```

Strict Equal (`===`): Checks if two values are equal without type coercion.

```
5 === "5";  // Outputs: false
```

Not Equal (`!=`): Checks if two values are not equal after type coercion.

Strict Not Equal (`!==`): Checks if two values are not equal without type coercion.

Greater Than (`>`), Less Than (`<`), Greater Than or Equal (`>=`), Less Than or Equal (`<=`): Used for comparing two values.

5. Logical Operators:

- AND (`&&`): Returns true if both operands are true.
- OR (`||`): Returns true if at least one operand is true.
- NOT (`!`): Returns true if the operand is false, and false if the operand is true.

6. Truthy or Falsy:

- In JavaScript, values are either "truthy" or "falsy".
- Falsy Values: `false`, `0`, `""` (empty string), `null`, `undefined`, `NaN`.
- Most other values, including all objects, are considered "truthy".

7. Increment (++) and Decrement (--):

These operators are used to increase or decrease a value by 1, respectively. They can be used in both prefix and postfix forms, which can affect the order of operations.

Prefix (++x and --x):

The value of the variable is changed before its current value is used in an expression. Example:

```
let x = 5;
let y = ++x;
console.log(x);  // Outputs: 6 (x has been incremented before assignment)
console.log(y);  // Outputs: 6 (y has the incremented value of x)
```

Postfix (x++ and x--):

The value of the variable is changed after its current value is used in an expression. Example:

```
let a = 5;
let b = a++;
console.log(a);  // Outputs: 6 (a has been incremented after assignment)
console.log(b);  // Outputs: 5 (b has the original value of a before increment)
```

8. Compound Assignment:

Compound assignments are shorthand methods to update the value of a variable based on its current value. They combine a binary operation (like addition or multiplication) and assignment into one operation.

Addition Assignment (+=):

Adds the right operand to the left operand and assigns the result to the left operand. Example:

```
let num = 10;
num += 5;  // Equivalent to num = num + 5;
console.log(num);  // Outputs: 15
```

Subtraction Assignment (-=):

Subtracts the right operand from the left operand and assigns the result to the left operand.
Example:

```
let num = 10;
num -= 5;  // Equivalent to num = num - 5;
console.log(num);  // Outputs: 5
```

Multiplication Assignment (*=):

Multiplies the left operand by the right operand and assigns the result to the left operand.
Example:

```
let num = 10;
num *= 2;  // Equivalent to num = num * 2;
console.log(num);  // Outputs: 20
```

Division Assignment (/=):

Divides the left operand by the right operand and assigns the result to the left operand.
Example:

```
let num = 10;
num /= 2;  // Equivalent to num = num / 2;
console.log(num);  // Outputs: 5
```

Terms

1. Interpreted vs. Compiled:

Interpreted:

- Definition: Interpreted languages are executed line-by-line by another program known as an interpreter. The interpreter reads the source code and executes it directly.
- Advantages:
    - Portability: The same code can run on any machine with the appropriate interpreter.
    - Faster development cycle: No need to compile, just write and run.
- Disadvantages:
    - Slower execution: The interpreter translates the code to machine instructions on-the-fly, which can be slower than pre-compiled code.
- Examples: JavaScript, Python, Ruby.

Compiled:

- Definition: Compiled languages are transformed into machine code by a compiler before execution. The source code is translated all at once and saved as a binary file, which can be executed.
- Advantages:
    - Faster execution: The translation from high-level code to machine instructions has already been done.
    - Often allows for more optimizations.
- Disadvantages:
    - Longer initial development cycle due to the need for compilation.
    - Portability issues: Binaries are often OS and architecture-specific.
- Examples: C, C++, Java (though Java uses a mix, compiling to bytecode, which is then interpreted or compiled at runtime).

---

2. Dynamic Typing:

- Definition: In dynamically-typed languages, variables do not have fixed types. Instead, a variable's type can change at runtime based on the value it holds.
- Advantages:
    - More flexible and often leads to quicker development.
- Disadvantages:
    - Potential for runtime errors if a variable doesn't hold the expected type.
- Example: In JavaScript, you can have `var x = 5;` and later assign `x = "hello";`.

---

3. Weakly Typed:

- Definition: In weakly-typed languages, implicit type conversion (or type coercion) happens. This means the interpreter/compiler will automatically change one type of data to another if it deems necessary.
- Example: In JavaScript, adding a string and a number like `"5" + 3` results in the string `"53"`. The number is coerced to a string for the operation.

---

4. Server-side vs. Client-side JS:

Server-side JavaScript:

- Definition: JavaScript that runs on the server, handling data processing, database operations, file manipulation, etc.
- Key Features:
    - Can access databases directly.
    - Executes before content is sent to the user's browser.
    - Examples include Node.js and Deno.

Client-side JavaScript:

- Definition: JavaScript that runs in the browser, after a web page is loaded. It primarily enhances user interaction and interfaces.
- Key Features:
    - Directly manipulates the Document Object Model (DOM) of a web page.
    - Limited access to the user's system for security reasons.
    - Cannot access databases directly (it needs to make requests to a server to do so).

- Is what most people traditionally think of when referring to JavaScript in web development.

Resources and Practice

Free Code Camp:
https://www.freecodecamp.org/learn/javascript-algorithms-and-data-structures/

JavaScript Tutorial (W3):
https://www.w3schools.com/js/

CodePen Module 9:
https://codepen.io/collection/ExpgKO