LEXICON

JavaScript Essentials

To make web pages interactive and provide dynamic functionality on the client side

Objectives

- Introduction to JavaScript
- JavaScript Variables & Operators
- Functions & Scope
- Errors
- Objects and Arrays
- DOM Manipulation
- Events

Introduction to JavaScript

- JavaScript is the **programming language** for the **Web**.
- To create interactive and dynamic web content = **client-side scripting.**
- To modify and manipulate web page content in real-time.
- It is essential for **front-end** development.
- It can be used for **server-side** development.
- The most common environment for JavaScript is within **web browsers**.
- JavaScript can also run on the server side using **Node.js**.
- With Similarities to C-family languages.

Variables

- Variables are used to store data values temporarily.
- To Declare variables:
  1. var (older way, less recommended):
     - Variables declared with **var** are either limited to the function they are declared in or, if not in a function, they become global.
     - Example: `var x = 10;`
  2. let (introduced in ES6):
     - Variables declared with let are block-scoped, meaning their scope is limited to the block in which they are declared, like loops or conditionals.
     - Example: `let y = 20;`
  3. const (introduced in ES6):
     - Variables declared with const are also block-scoped, and their value cannot be reassigned after declaration.
     - Example: `const z = 30;`

Datatypes

- Primitive & Reference data types.

  Number: Represents numeric values, including integers and floating-point numbers.

  String: Represents textual data, enclosed in single (') or double (") quotes.

  Boolean: Represents true or false values. Example: **true** or **false**.

  Undefined: Represents a variable that has been declared but not assigned a value.

  Null: Represents the intentional absence of any object or value.

  Object: Represents complex data structures and collections of key-value pairs.

  Array: Represents ordered collections of values.

```
let description;
let age = 22;
let title = "Developer";
```

```
let person = { firstName: "Mehrdad", lastName: "Javan"};
let numbers = [1, 2, 3, 4, 5];
const pi = 3.14;
```

Dynamic types

- Different from a statically typed language
- Type is deduced in JavaScript by type inference
- The type of a variable can be changed
- The type of a value can be coerced or converted into another type

```
let myVariable; // No type declaration
myVariable = 42; // Now it's a number
myVariable = "Hello, World!"; // Now it's a string
myVariable = { key: "value" }; // Now it's an object

let num = 5;
let str = "10";
let result = num + str; // Result is "510" (string concatenation)
```

Operators

- Most operators in JS look and work the same

+, -, %, =, *, /, ?, >, >=, ++

- Checking equality in JS needs to be strict

=== (checking equal value and equal type)
== (checking equal value)
!== (not equal value and not equal type)

Functions – syntax options

- To encapsulate and reuse blocks of code.
- function functionName(param1, param2) { }
- Const functionName = function(param1, param2) { }
- Const functionName = (param1, param2) => { }
- () => ()

```
function greet(name) {
    return "Hello, " + name + "!";
}
let message = greet("JavaScript");
console.log(message); // "Hello, JavaScript!"

// Arrow functions provide a concise way to write functions.
let multiply = (a, b) => a * b;
```

Function Scope

- Visibility and Accessibility of variables declared within a function.

Function Scope:

- Variables declared with `var` are function-scoped.
- They are accessible only within the function in which they are defined.

Block Scope:

- Variables declared with `let` and `const` are block-scoped.
- They are accessible within the block (a pair of curly braces) in which they are defined.
- If defined within a function, they are accessible only within that function.

Global Scope:

- Variables declared outside of any function or block have global scope.
- They can be accessed from any part of the code.

Errors

- Are known as exceptions or runtime errors.

- **Syntax Errors:**
  - Occur due to violations of JavaScript language rules.
  - Prevent code execution and typically involve missing or incorrect syntax elements.
- **Reference Errors:**
  - Happen when trying to access variables or objects that are not defined or out of scope.
  - Often result from mistyped variable names or accessing nonexistent object properties.
- **Type Errors:**
  - Arise when operations are performed on values of the wrong type.
  - Common examples include attempting to call non-functions as functions or accessing properties on undefined values.
- **Range Errors:**
  - Occur when accessing arrays or performing string operations with invalid indices or lengths.
  - Typically triggered when attempting to access elements beyond the array's boundaries.
- **Custom Errors:**
  - Can be created using the **Error** constructor to provide customized error messages and additional context for specific issues in the code.

```
throw new Error("This is a custom error message.");
```

Objects

- Objects are a fundamental concept in the language.
- JavaScript uses objects to represent data.
- Objects are like containers with keys and values.
- Keys are unique identifiers, and values can be anything.
- Objects are used to model complex data structures.

```
let person = {
    id: 1,
    firstName: "Mehrdad",
    lastName: "Javan",
    isTeacher: true
  };

console.log(person.firstName); // Mehrdad
console.log(person[“id"]); // 1
```

```
let car = {
    make: "Volvo",
    model: "V60",
    start: function() {
      console.log("Engine started!");
    },
  };
  car.start(); // Engine started!
```

Arrays

- Arrays are a fundamental data structure.
- To store collections of items or values.
- Elements in an array can be of different data types.
- Elements in an array are accessed by their index, starting from 0.
- JavaScript provides built-in methods for working with arrays(**push**, **pop**, **splice**, **concat** & etc.)
- Array iteration methods (**forEach**, **map**, **filter**, & **reduce**)

```
let fruits = ["apple", "banana", "cherry", "date"];
console.log(fruits[0]); // "apple"
console.log(fruits[2]); // "cherry"
fruits[1] = "grape"; // Modifies the second element
console.log(fruits.length); // 4
for (let i = 0; i < fruits.length; i++) {
    console.log(fruits[i]);
}
```

Use JavaScript in HTML

There is two ways to integrate **Javascript** into **HTML** document

1. By writing code inside <script> tag

```
<div id="demo">Some content</div>
<script>
    document.getElementById("demo").innerHTML = "My First JavaScript";
</script>
```

2. By having the code in a separate file and link it to the page

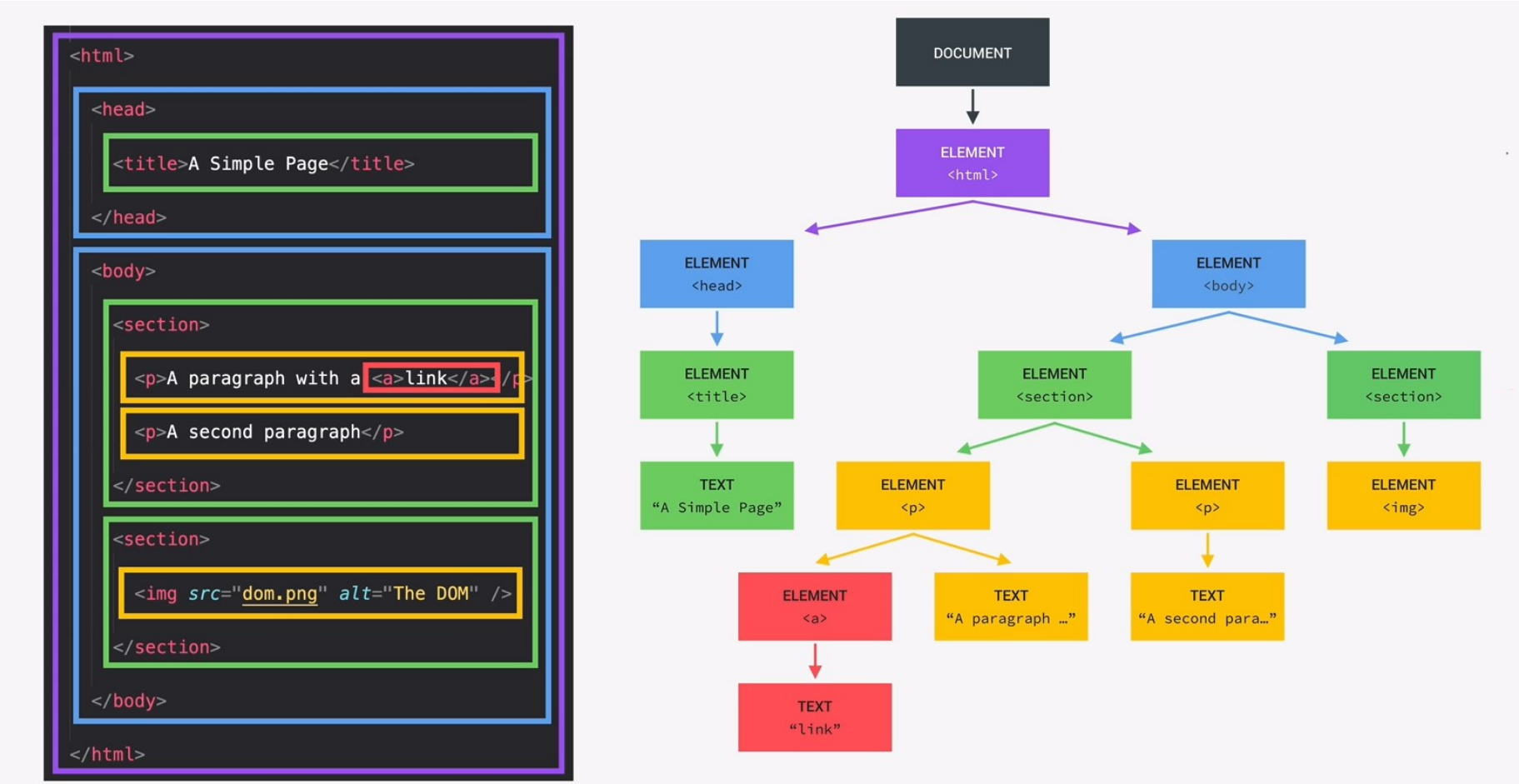
```
<div id="demo">Some content</div>
<script src="myjavascriptfile.js"></script>
```

Document Object Model (DOM)

- A programming interface (a toolkit) for **web documents**.
- Allows JavaScript to **interact with and manipulate** HTML and XML documents.
- Represents the **structure of a document** with a logical **tree**.
- Every **branch**, or every **element** in the DOM tree, is represented by a **node**.
- A node can represent various things like **elements**, **attributes**, or **text**.
- Every node in the DOM tree corresponds to a JavaScript **object**.
- Each object corresponds to an element in the document.
- DOM elements are organized in a parent-child relationship.
- Use DOM methods to modify the structure, or content of the document.

HTML DOM

- When a web page is loaded, the browser creates a **D**ocument **O**bject **M**odel of the page.



To access the HTML DOM Elements

Use methods and properties to interact with elements and make dynamic changes.

Finding HTML Elements

Method	Description
document.getElementById(*id*)	Find an element by element id
document.getElementsByTagName(*name*)	Find elements by tag name
document.getElementsByClassName(*name*)	Find elements by class name

```
const element = document.getElementById('myId’);

const elements = document.getElementsByClassName('myClass’);

const paragraphs = document.getElementsByTagName('p’);

const firstElement = document.querySelector('.myClass');
```

To access the HTML DOM Elements

Changing HTML Elements

Property	Description
element.innerHTML = *new html content*	Change the inner HTML of an element
element.attribute = new value	Change the attribute value of an HTML element
element.style.*property = new style*	Change the style of an HTML element
Method	Description
element.setAttribute*(attribute, value)*	Change the attribute value of an HTML element

```
const element = document.getElementById("main");
element.innerHTML =  "new content";

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

To access the HTML DOM Elements

Adding and Deleting Elements

Method	Description
document.createElement(*element*)	Create an HTML element
document.removeChild(*element*)	Remove an HTML element
document.appendChild(*element*)	Add an HTML element
document.replaceChild(*new, old*)	Replace an HTML element
document.write(*text*)	Write into the HTML output stream

To create a table element

```
// Create a table element
const table = document.createElement('table');
table.setAttribute('id', 'myTable');

// Create table header (th) row
const headerRow = table.insertRow();
const headerCell1 = headerRow.insertCell(0);
const headerCell2 = headerRow.insertCell(1);
headerCell1.textContent = 'Name';
headerCell2.textContent = 'Age';

// Create table rows with data
const dataRows = [   ['John Doe', '25'],   ['Jane Smith', '30’] ];

dataRows.forEach(data => {
  const row = table.insertRow();
  data.forEach((cellData, index) => {
    const cell = row.insertCell(index);
    cell.textContent = cellData;
  });

});

// Add the table to the body of the HTML document
document.body.appendChild(table);
```

DOM Events

- In JavaScript, events are actions that happen in the browser.
- JavaScript makes web pages interactive by catching and responding to events.
- Examples of HTML events:
  - Click a button
  - Type on the keyboard
  - Resize the window
  - Move the mouse over an element
  - Change something in an input field

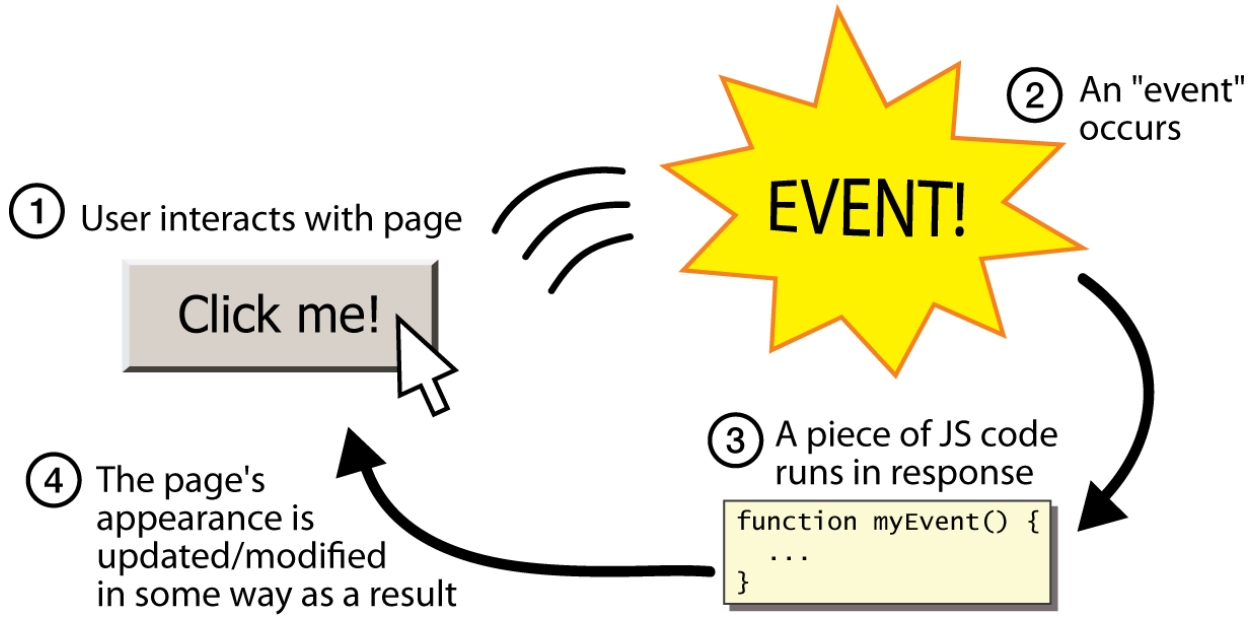
DOM Event Listener

- Attaches functions to respond to specific events on HTML elements.
- Use **addEventListener** method to add an event listener to an element.
- Use **removeEventListener** method to remove an event listener on an element.

```
element.addEventListener("click", myFunction );
```

What is an Event ?

JavaScript's interaction with **HTML** is handled through ***events*** that occur when the user or the browser manipulates a page. When the page loads, it is called an ***event***. When the user clicks a button, that click too is an ***event***. Other examples include ***events*** like pressing any key, closing a window, resizing a window, etc.



Reference

- https://www.tutorialspoint.com/javascript/javascript_overview.htm
- https://www.w3schools.com/
- https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model/Introduction