Introduction to JavaScript, Developer Tools & DOM

JavaScript basics, introduction to Developer Tools, using the Document Object Model (DOM) and creating an interactive Memory Game.

FH University of Applied Sciences

TECHNIKUM

WIEN

What is JavaScript?

JavaScript is a **client-side scripting language** for the web.

- **Client-Side**: Runs on the user's browser, as opposed to the server.
- **Adds Interactivity**: Allows dynamic updates, animations, and real-time data.
- **Works with HTML & CSS**: JavaScript controls behavior, while HTML provides structure, and CSS manages design.

Example:

```javascript
console.log("Hello, World!");
```

This outputs "Hello, World!" to the Console, a key debugging tool we'll explore further.

JavaScript in the Browser

JavaScript can be embedded directly in HTML files.

- **Included with** `<script>` **Tags**: JavaScript is placed inside HTML using `<script>` tags.
- **Executed in the Browser**: The browser interprets and runs JavaScript as soon as it encounters it.

Example of Including JavaScript in HTML:

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>JavaScript Example</title>
</head>
<body>
    <h1>Welcome to JavaScript!</h1>
    <script>
        console.log("Hello from JavaScript inside HTML!");
    </script>
</body>
</html>
```

When this page is loaded, "Hello from JavaScript inside HTML!" is printed to the Console.

Introduction to Developer Tools

Developer Tools are essential for **debugging, testing, and optimizing** JavaScript in the browser.

1. **Opening Developer Tools**:
   - Press **F12** or right-click and select **Inspect** to open.

2. **Key Tabs**:
   - **Elements**: Displays HTML and CSS for structure and styling.
   - **Console**: Executes and logs JavaScript, shows errors and warnings.
   - **Sources**: Holds JavaScript files, allows breakpoints and step-by-step debugging.
   - **Network**: Monitors network requests, useful for fetching external data.

The JavaScript Console

The Console allows you to:

- **Run JavaScript directly**: You can enter and execute code live.
- **Print debugging output**: `console.log()` helps trace code execution.
- **View errors and warnings**: Console alerts you to issues in your code.

Example Usage of `console.log()`:

```javascript
let name = "Alice";
console.log("Hello, " + name); // Output: Hello, Alice
```

Practical Tips:

- Use `console.log(variableName);` to view variables at key points.
- Check for error messages to understand unexpected behavior.

Variables in JavaScript

JavaScript provides three ways to declare variables:

1. `let` : Block-scoped, used for variables that might change.
2. `const` : Block-scoped, used for constants that won't change.
3. `var` : Global or function-scoped, outdated for most modern uses.

Examples:

```javascript
let age = 20;        // Block-scoped, mutable
const pi = 3.14159;  // Block-scoped, immutable
var country = "Austria"; // Global scope, less preferred
```

Good Practices:

- Use `let` and `const` in modern code for readability and scope control.
- Use `const` for values that shouldn't change, like configuration data.

JavaScript Data Types

Key **data types** in JavaScript:

- **Number**: Numeric values, e.g., `42`, `3.14`
- **String**: Text values, enclosed in quotes, e.g., `"Hello"`
- **Boolean**: Logical values, `true` or `false`
- **Array**: Lists of values, e.g., `['apple', 'banana', 'grape']`
- **Object**: Key-value pairs, useful for structured data, e.g., `{ name: 'Alice', age: 20 }`

Examples:

```javascript
let fruits = ['apple', 'banana', 'grape'];
let person = { name: "Alice", age: 20 };

console.log(fruits); // Output: ["apple", "banana", "grape"]
console.log(person.name); // Output: "Alice"
```

Arrays and objects are essential for organizing complex data in JavaScript.

Using the Console: Debugging with `console.log()`

The Console object provides helpful debugging functions:

- `console.log()` : Prints standard information.
- `console.error()` : Shows error messages.
- `console.warn()` : Displays warnings for potential issues.

Example:

```javascript
console.log("Info: Process started.");
console.error("Error: Something went wrong!");
console.warn("Warning: Check input values.");
```

Tips:

- Use `console.log()` frequently to monitor program flow.
- Use `console.error()` to highlight critical failures.

Basic Operators in JavaScript

JavaScript operators allow calculations, comparisons, and logic:

1. **Arithmetic Operators**: `+`, `-`, `*`, `/`
2. **Comparison Operators**: `==`, `===`, `!=`, `!==`, `>`, `<`, `>=`, `<=`
3. **Logical Operators**: `&&` (and), `||` (or), `!` (not)

Example:

```javascript
let x = 5;
let y = 10;

console.log(x + y);        // Output: 15
console.log(x === y);      // Output: false
console.log(x < y && x > 0); // Output: true
```

Operators are essential for setting conditions and calculations within your code.

🏗️ Introduction to the DOM (Document Object Model)

The **Document Object Model (DOM)** represents the structure of a webpage as an **interactive tree**.

- **Each HTML element is a node** in the tree.
- JavaScript can **access, modify, and manipulate** these elements dynamically.
- Changes in the DOM instantly update the webpage.

Example: HTML Document Structure

```html
<!DOCTYPE html>
<html>
<head>
    <title>DOM Example</title>
</head>
<body>
    <h1 id="title">Welcome to the DOM</h1>
    <button id="changeText">Click Me</button>
</body>
</html>
```

🌳 Understanding the DOM Tree

Every webpage is structured as a **DOM Tree**:

```
Document
 ├── <html>
 │    ├── <head>
 │    │    ├── <title>
 │    ├── <body>
 │         ├── <h1 id="title">
 │         ├── <button id="changeText">
```

JavaScript allows **selecting, modifying, and handling events** on these elements.

🎯 Selecting DOM Elements

To interact with elements, JavaScript provides various selection methods:

1. Selecting Elements by ID

```javascript
let heading = document.getElementById("title");
console.log(heading.textContent); // Output: Welcome to the DOM
```

2. Selecting Elements by Class

```javascript
let buttons = document.getElementsByClassName("btn");
```

3. Selecting Elements with Query Selectors

```javascript
let heading = document.querySelector("#title");
let allButtons = document.querySelectorAll("button");
```

✏️ Modifying the DOM

Once selected, elements can be changed dynamically.

Changing Text Content

```
let title = document.getElementById("title");
title.textContent = "Hello, DOM!";
```

Changing Styles

```
title.style.color = "blue";
title.style.fontSize = "24px";
```

Adding and Removing Elements

```
let newParagraph = document.createElement("p");
newParagraph.textContent = "This is a new paragraph!";
document.body.appendChild(newParagraph);
```

Functions in JavaScript

Functions are reusable blocks of code that perform a specific task.

- **Define** with the `function` keyword.
- **Parameters** allow inputs to customize the function.
- **Return** statements output values to the rest of the code.

Example:

```javascript
function greet(name) {
    return "Hello, " + name + "!";
}


console.log(greet("Alice")); // Output: Hello, Alice!
```

Benefits:

- Functions reduce code repetition and increase readability.
- Functions can return values or directly modify the program.

Hands-on: Functions and Debugging with the Console

1. **Create a Simple Addition Function**:

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

2. **Use `console.log()` for Debugging**:

   - Insert `console.log()` to monitor variable values and function outputs.

3. **Set Breakpoints**:

   - Use Developer Tools to pause execution and inspect values line-by-line.

Example with Debugging:

```javascript
let total = add(5, 10);
console.log("Total:", total); // Output: Total: 15
```

Debugging Techniques in JavaScript

1. **Using** `console.log()`:
   - `console.log(variable);` to view key points and variable states.

2. **Breakpoints in Developer Tools**:
   - Breakpoints pause execution at selected lines, allowing you to inspect variables and understand the code's flow.

Example with Breakpoints:

```javascript
function add(a, b) {
    console.log("Adding:", a, "+", b); // Track inputs
    return a + b;
}
add(3, 7); // Observe in Console
```

Demo: Creating a Simple Memory Game

Project Objective:

Build a Memory Game using JavaScript, HTML, and CSS.

1. **Step 1**: Load an HTML and CSS template for the game structure.
2. **Step 2**: Implement JavaScript logic for flipping and matching cards.
3. **Step 3**: Debug the code using Console output and breakpoints.

🛠️ Hands-on: Memory Game & the DOM

To build a **Memory Game**, we need to:

1. **Dynamically generate cards** in the DOM.

2. **Listen for click events** to flip cards.

3. **Check for matches** and track score.

Example: Creating Card Elements in JavaScript

```javascript
let card = document.createElement("div");
card.classList.add("card");
card.textContent = "🍎";
document.body.appendChild(card);
```

Memory Game Logic: Flipping and Matching Cards

1. **Initialize the Card Array**:

```javascript
let cards = ['apple', 'apple', 'banana', 'banana', 'grape', 'grape'];
```

2. **Shuffle the Cards**:

```javascript
function shuffle(array) {
    array.sort(() => Math.random() - 0.5);
}
shuffle(cards);
console.log(cards); // Check shuffled order in Console
```

This prepares the deck for a randomized layout.

Memory Game Functions: Flip and Match

1. **flipCard()**:
   - This function flips a card when clicked.
   - Use `console.log()` to display which card is flipped and track the game flow.

2. **checkMatch()**:
   - Compares two flipped cards to check for a match.
   - Debug matching logic with `console.log()` and breakpoints.

Example:

```javascript
function flipCard(card) {
    console.log("Flipped:", card);
    // Flip logic here
}
```

Each function improves game interactivity by providing actions for the player.

Next Steps & Weekly Exercise

- **Explore Developer Tools further**: Practice using Console, breakpoints, and Network tabs.
- **Practice with the DOM**: Try selecting and modifying elements on a webpage.
- **Weekly Exercise**: Build the basic structure of a Memory Game, integrating debugging tools.
- **Goal**: A functional game with Console-based debugging support.

Summary and Q&A

- **JavaScript Basics Recap**: Variables, operators, functions, debugging tools.

- **Developer Tools**: Key elements in the Console and Sources tabs.

- **DOM = structured representation of a webpage**

- **Q&A**: Open session to address questions.

- **Exercise Preparation**: Set up and start implementing the Memory Game project for next week.