

INTERNATIONAL ADVANCED LEVEL

INFORMATION TECHNOLOGY

UNIT 2

Topic 9 – Understanding the Functions of JavaScript



Academic Year: 2024-2025

✍ Prepared by: Irikefe Hope Oniovosa

1. The Document Object Model (DOM)

Concept Explanation

- **Definition of the DOM:**

The Document Object Model (DOM) is a programming interface for HTML and XML documents. It represents the page as a tree-like structure where each node is an object corresponding to a part of the document (such as elements, attributes, and text). This abstraction allows programming languages like JavaScript to interact with the page's content, structure, and style.

- **How JavaScript Uses the DOM:**

JavaScript interacts with the DOM to access and update web page elements in real time. This means you can:

- **Retrieve Elements:** Find elements by their ID, class, tag name, or using query selectors.
- **Modify Content:** Change the text, HTML, or attributes of elements.
- **Alter Styles:** Dynamically update CSS styles for any element.
- **Respond to User Events:** Attach event listeners to elements to respond to clicks, key presses, and other user interactions.

Demonstration

1. Inspecting the DOM:

- **Step 1:** Open a simple HTML page in a web browser.
- **Step 2:** Use browser developer tools (F12 or right-click and “Inspect”) to view the DOM tree.

Explain that each HTML tag is represented as a node in this tree.

2. Code Example – Changing an Element's Content and Style:

HTML Sample:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>DOM Demo</title>
</head>
<body>
  <h1 id="page-title">Welcome to My Website</h1>
  <p class="description">This is a sample paragraph.</p>
  <button id="change-button">Change Text</button>
  <script src="script.js"></script>
</body>
</html>
```

JavaScript (script.js)

```
// Select the h1 element using its ID
const titleElement = document.getElementById('page-title');

// Change its text content
titleElement.textContent = 'Hello, JavaScript!';

// Change its style (color and font size)
titleElement.style.color = 'blue';
titleElement.style.fontSize = '2.5rem';

// Add an event listener to the button to change the paragraph text when clicked
const button = document.getElementById('change-button');
button.addEventListener('click', function() {
  const paragraph = document.querySelector('.description');
  paragraph.textContent = 'The text has been changed!';
  paragraph.style.backgroundColor = 'lightyellow';
});
```

- **Explanation:**

- `document.getElementById('page-title')` selects the `<h1>` element by its ID.
- `textContent` is used to update the element's displayed text.
- Style properties are updated directly on the element.

- An event listener is added to a button to demonstrate interactive DOM manipulation.

Interactive Exercise

1. Task for Students:

- Provide a sample HTML page similar to the demonstration above.
- Ask students to write JavaScript code that:
 - Selects a specific element (for example, a `<p>` tag with a class "info").
 - Changes the text content of that element.
 - Updates its CSS style (for example, change the background color or font style).

2. Guidance:

- Encourage students to experiment with different DOM selection methods (`getElementById`, `querySelector`, `getElementsByClassName`).
- Remind them to use browser developer tools to inspect the DOM and verify their changes.
- Discuss any challenges encountered during the exercise.

3. Outcome:

- Students will gain hands-on experience with accessing and manipulating the DOM using JavaScript.
- They will better understand how dynamic content updates enhance user interaction on a web page.

2. Regular Expressions

A. Concept Explanation

- **Definition:**

Regular expressions (regex or regexp) are patterns used to match character combinations within strings. They provide a powerful way to search, edit, or validate text based on specific patterns.

- **Importance in Validation:**

Regular expressions are crucial for validating user input such as:

- Email addresses
- Phone numbers
- Passwords

This ensures that the data entered into a form or application meets the required format and standards.

B. Key Topics

1. Matching Characters:

- **Letters:**
 - [a-z] matches any lowercase letter.
 - [A-Z] matches any uppercase letter.
 - [a-zA-Z] matches any letter regardless of case.
- **Numbers:**
 - [0-9] matches any digit.
- **Punctuation and Symbols:**
 - Specific characters like . (dot), ! (exclamation), or @ (at symbol) can be matched explicitly or via escape sequences (e.g., \.).

2. Repeating Characters and Digits:

- **Quantifiers:**
 - * matches zero or more occurrences.
 - + matches one or more occurrences.
 - ? matches zero or one occurrence.
 - {n} matches exactly n occurrences.
 - {n,} matches n or more occurrences.

- {n,m} matches between n and m occurrences.
- **Example:**
 - `/\d{3}-\d{2}-\d{4}/` might match a pattern like a Social Security Number in the format 123-45-6789.

3. Constructing Patterns:

- Patterns are built by combining literals, character classes, quantifiers, and other special characters.

- **Example for an Email:**

- A simple email regex might look like:

```
/^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$/
```

- This pattern checks for:
 - One or more letters, numbers, dots, underscores, or special characters before the @.
 - A valid domain name after the @.
 - A top-level domain of at least two letters.

C. Demonstration

1. JavaScript Example: Validating an Email Address

```
// Define a regular expression for a simple email validation
const emailRegex = /^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$/;

// Test the regex with a sample email
const testEmail = "student@example.com";

// Using test() method to check if the email matches the pattern
if (emailRegex.test(testEmail)) {
  console.log("The email is valid.");
} else {
  console.log("The email is invalid.");
}
```

- **Explanation:**
 - The regex is enclosed in / delimiters.
 - The caret ^ and dollar sign \$ ensure the entire string is checked.
 - The pattern includes various character classes and quantifiers to validate the email format.

D. Interactive Exercise

1. Activity Overview:

Provide students with a set of input strings (e.g., various email addresses, phone numbers, and passwords) along with criteria for what constitutes valid data.

2. Task:

- Ask students to write their own regular expressions that match the given criteria.
- For example:
 - **Email Validation:** Must include an @ symbol, have a domain, and a valid TLD.
 - **Phone Number:** In a format such as (123) 456-7890 or 123-456-7890.
 - **Password:** Must be at least 8 characters long, contain at least one digit, one uppercase letter, and one lowercase letter.

3. Discussion and Testing:

- Students can work individually or in pairs to construct their regex.
- Use online regex testers like regex101.com for immediate feedback.
- Share results and discuss different approaches to solving the validation problems.

Wrap-Up

• Recap the Key Points:

- Regular expressions are powerful tools for text matching and validation.
- They use a combination of literals, character classes, quantifiers, and anchors.
- Practical applications include validating emails, phone numbers, and passwords.

• Q&A Session:

- Ask students, “What does the + quantifier do in a regex?”
- Ask, “How would you modify the email regex to require a specific domain, for example, @school.edu?”

3. Programmed Functionality

A. Adding JavaScript to Web Pages

1. Inline Scripts:

- **Usage:** Directly include JavaScript within an HTML element using the onclick attribute or similar.

- **Example:**

```
<button onclick="alert('Hello!')">Click Me</button>
```

- **Pros/Cons:** Quick for small scripts but mixes content and behavior.

2. Internal `<script>` Tags:

- **Usage:** Place JavaScript within a `<script>` tag inside the HTML document (commonly within the `<head>` or at the end of `<body>`).

- **Example:**

```
<script>  
  console.log('Page loaded');  
</script>
```

- **Pros/Cons:** Keeps code within the document but can clutter HTML if overused.

3. External JavaScript Files:

- **Usage:** Link an external .js file using the `<script src="path/to/script.js"></script>` tag.

- **Example:**

```
<script src="js/main.js"></script>
```

- **Pros/Cons:** Separates structure from behavior, promoting reusability and maintainability.

B. Fundamental Programming Concepts

1. Comments and Assignment:

- **Comments:** Use `//` for single-line comments and `/* ... */` for multi-line comments.

```
// This is a single-line comment
```

```
/* This is a  
multi-line comment */
```

- **Assignment:**

```
let message = "Hello, world!";
```

2. Data Structures:

- **Arrays:** One-dimensional lists of values.

```
let fruits = ["apple", "banana", "cherry"];
```

- **Two-Dimensional Arrays:** Arrays within arrays.

```
let matrix = [  
  [1, 2, 3],  
  [4, 5, 6],  
  [7, 8, 9]  
];
```

3. Control Structures:

- **Selection (if/else):**

```
if (score >= 50) {  
  console.log("Pass");  
} else {  
  console.log("Fail");  
}
```

- **Repetition/Iteration (for, while):**

```
for (let i = 0; i < fruits.length; i++) {  
  console.log(fruits[i]);  
}
```

4. Variables and Subprograms:

- **Declaring Variables:**

```
let name = "Alice";  
const PI = 3.14;
```

- **Functions (Subprograms):**

```
function greet(person) {  
  return "Hello, " + person + "!";  
}  
  
console.log(greet("Alice"));
```

5. Object Orientation:

- **Objects:**

Encapsulate data and functions.

```
let person = {  
  name: "Bob",  
  age: 30,  
  greet: function() {  
    return "Hi, I'm " + this.name;  
  }  
};  
  
console.log(person.greet());
```

C. Event Handling

1. Concept:

- Events are actions (like clicks, key presses, mouse movements) that trigger JavaScript functions.

2. Types of Events:

- **User Interface (UI) Events:** Clicks, mouseovers.
- **Keyboard Events:** keydown, keyup.

- **Mouse Events:** click, dblclick.
- **Focus/Blur Events:** When elements gain or lose focus.
- **Form Events:** Submission, change, input.

3. **Demonstration: Button Click Example**

HTML:

```
<button id="myButton">Click Me</button>
<p id="output">Original Text</p>
```

JavaScript:

```
// Select the button and paragraph elements
const button = document.getElementById('myButton');
const output = document.getElementById('output');

// Add an event listener to handle clicks
button.addEventListener('click', function() {
  output.textContent = "The text has been changed!";
  output.style.color = "green";
});
```

D. Integration with HTML and CSS

- **Interactive Components:**

JavaScript works together with HTML and CSS to create interactive components:

- **Slideshows:** Cycle through images with animations.
- **Modal Boxes/Images:** Display overlay content when triggered.
- **Filter and Sort Lists:** Dynamically update list content based on user input.
- **Pop-Ups and Tabbed Content:** Show/hide sections of the page.

E. Interactive Exercise

1. Task:

- In small groups or individually, have students write a simple JavaScript program that:
 - Uses the DOM to modify page content (e.g., change text or style of an element).
 - Validates user input using a regular expression (e.g., ensuring an email address is correctly formatted).
 - Responds to an event (e.g., a button click) to trigger a function.
 - **Optional:** Create a small interactive component like a basic slideshow or a pop-up message.

2. Instructions:

- Provide students with a starter HTML file that includes a few elements (a form, a button, and some text).
- Encourage students to write the JavaScript code in an external file and link it to the HTML.
- Allow time for testing and debugging.
- Invite groups to share their solutions and discuss the challenges they faced.

Wrap-Up

• Recap Key Points:

- Different methods of adding JavaScript to web pages (inline, internal, external).
- Fundamental programming concepts including variables, data structures, control structures, functions, and objects.
- The importance of event handling in creating interactive web experiences.
- How JavaScript integrates with HTML and CSS to enhance web page functionality.

• Q&A Session:

- Ask questions such as:
 - “How does the DOM allow JavaScript to interact with a web page?”
 - “Can you give an example of when you would use a for loop in JavaScript?”
 - “Why might you choose to place your JavaScript code in an external file?”

4. Summary and Review

Recap Key Points

1. DOM Manipulation:

- The DOM (Document Object Model) is a tree-like representation of an HTML document.
- JavaScript can access and update the DOM in real time, which allows dynamic content updates (e.g., changing text, styles, or adding new elements).

2. Regular Expressions:

- Regular expressions are patterns used to match specific sequences of characters within strings.
- They are crucial for validating user input (e.g., ensuring an email or phone number is in the correct format).

3. Fundamental Programming Concepts:

- Core concepts include variables, data structures (arrays, objects), control structures (if/else, loops), functions, and basic object-oriented ideas.
- These concepts form the building blocks of any JavaScript program.

4. Event Handling:

- JavaScript responds to user interactions (such as clicks, key presses, or form submissions) by attaching event listeners.
- Event handling allows for interactive and responsive web pages.

5. Integration with HTML and CSS:

- JavaScript works together with HTML (structure) and CSS (styling) to create dynamic, interactive web components like slideshows, modal boxes, and pop-ups.
- This integration enhances user experience and makes web pages more engaging.

Quick Q&A Session

- **Question:** “How does the DOM help in updating a web page dynamically?”

Expected Answer: The DOM represents the HTML document as a tree, which JavaScript can manipulate—adding, removing, or updating elements—thus enabling real-time changes to the page without a full reload.

- **Question:** “Can you give an example of a regular expression pattern used for validation?”

Expected Answer: One example is an email validation regex:

```
/^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$/
```

This pattern checks that an email address has valid characters before and after the "@" symbol and ends with a valid domain.

- **Question:** “What are some common events that trigger JavaScript functions?”

Expected Answer: Examples include:

- Click events (e.g., when a user clicks a button)
- Keyboard events (e.g., keydown or keyup)
- Mouse events (e.g., mouseover, mouseout)
- Form events (e.g., submit or change)
- Focus/blur events (when an element gains or loses focus)