ENSF-381: Full Stack Web Development Laboratory

Ahmad Abdellatif

Department of Electrical & Software Engineering

University of Calgary

Lab 3

Objectives

Welcome to the ENSF381 course lab. The main objective of this lab is to build a solid understanding of fundamental JavaScript concepts and how to apply them in practice. Students will develop skills in working with variables, strings, and dates, writing and using functions, and manipulating core data structures such as arrays and objects. By completing the tasks and testing code in the browser's developer console, students will gain hands-on experience writing dynamic and efficient JavaScript programs.

Groups

Lab instructions must be followed in groups of two students. Submissions from groups with fewer than two students will not be accepted.

Constraint

Use only the JavaScript functions and data structures covered in lectures or in this lab session.

Submission

- You must submit the complete source code, ensuring it can be executed without any modifications.
- If requested by the instructor, you may need to submit additional documentation (e.g., word documents and images).
- The link to the GitHub repository created for this lab must be provided alongside your submission on D2L. If you submit a word document as a part of this lab, include the link to the GitHub repository within that document as well.
- Only one member of the group needs to submit the assignment, but the submission must include the names and UCIDS of all group members.

Deadline

Lab exercises must be submitted by **11:55 PM on the same day as the lab session**.

Exercise 1: Exploring JavaScript Fundamentals

Objective: We will practice fundamental JavaScript concepts, including working with variables, strings, template literals, and dates, to develop a strong foundation in the language. You will execute the tasks in your browser's developer console to observe the output.

Create an HTML File

1. Open a text editor.
2. Create a new file and save it as exercise1.html.
3. Add the basic structure of an HTML document:

```
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Lab3 -- Exercise 1</title>
</head>

<body>
  <h1>JavaScript Lab Exercise 1</h1>
  <script>
    // Write your JavaScript code here
  </script>
</body>

</html>
```

Using the Developer Console

Before starting with exercise 1, we need to familiarize ourselves with the browser's developer console, where we will view the JavaScript outputs.

1. **Open the Developer Console**:
    - In your web browser (e.g., Chrome, Firefox, or Edge), right-click anywhere on the page and select Inspect or Inspect Element.
    - Navigate to the Console tab in the developer tools panel that appears.

2. **Write and Test JavaScript Code**: JavaScript code placed inside the <script> tag of your HTML file will run when the page loads, with any outputs or errors appearing in the console. You may use console.log(var) whenever you need to inspect the content of a variable var.

3. **Clear the Console**: to make it easier to track your work, clear the console between tests by clicking the Clear Console button (a trash can icon) or pressing *Ctrl+L* (Windows/Linux) or *Cmd+K* (Mac).

Exercise 1A: Variable Declaration

1. Declare one variable with var, one variable with let, and one variable with const. Assign a different value to each variable.

2. Change the value of the variable declared using let and observe the behaviour.

3. Try to reassign the value of the const variable and note what happens.

4. Wrap the invalid assignment from *step3* in a try…catch block. Within catch, log the message "Error: cannot reassign const".

Exercise 1B: String Operations

1. Declare two string variables:

```
let string1 = "Hello, World!"
let string2 = "JavaScript is fun!"
```

2. Use a method to find the length of string1 and display it in the console.

3. Concatenate string1 and string2 and log the result in the console.

4. Create a new variable string3 by replacing the word "fun" in string2 with "awesome", then display string3 in the console.

Exercise 1C: String Templates

1. Declare the following variables for the tasks in this part:

```
let name = "Alice"
let age = 25
```

2. Create a template literal using backticks (`) and the variables name and age so that the result reads: "My name is Alice and I am 25 years old".
   Hint: Insert variables into the template literal using the ${variableName} syntax.

3. Log the resulting template string in the console.

Exercise 1D: Iteration

1. Declare two arrays, one for student names and one for ages:

```
const students = ["Alice", "Tom", "Charlie"]
const ages = [19, 21, 20]
```

2. Use a for loop to iterate through the students array by index.

3. In each iteration, reuse the same template literal format from the previous section ("My name is ... and I am ... years old"), using matching indexes from both arrays, and log it to the console. For example:

```
for (let i = 0; i < students.length; i++) {
  let studentInfo = `My name is ${students[i]} and I am ${ages[i]} years old`;
  console.log(studentInfo);
}
```

Exercise 1E: Working with Dates

1. Create a variable to store the current date using JavaScript's Date object. This is a new concept, so you may need to research the Date object online for this lab.
2. Log the full date in the console in the en-CA and en-US formats.
3. Use methods from the Date object to:
    - Extract and log the year (e.g., 2026).
    - Extract and log the month as a number (e.g., 1 for February). Note that these are zero-indexed unlike the output we got from logging the date in *step 2*.
    - Extract and log the day of the week from the date object (e.g., Friday). ***Hint***: Use the weekday option with toLocaleDateString.
4. Determine and log the current season using the extracted month value. ***Hint***: You may use if...else to map month numbers to seasons. Assume Northern Hemisphere seasons:
    - Winter: 11, 0, 1
    - Spring: 2, 3, 4
    - Summer: 5, 6, 7
    - Fall: 8, 9, 10

Submission

1. Fill out the Answer_sheet.docx, including the names and UCIDs of all group members.
2. Create a new GitHub repository and upload the code for Exercise 1 to the new repository.

Exercise 2: Building a Simple Calculator

Objective: In this exercise, we will work on creating and experimenting with JavaScript functions to deepen your understanding of their concepts and usage. Follow the instructions carefully and test your code using the browser's developer console.

Exercise 2A: Declaring Functions

1. Repeat the steps from "Create an HTML File" in Exercise 1 to create a new file named exercise2.html. In the newly created file, write a function named add that accepts two numbers as parameters and returns their sum.
2. Write another function named subtract that accepts two numbers as parameters and returns their difference.
3. Use your browser's developer console to call each function with sample inputs and confirm the results.

Exercise 2B: Using Default Parameters

1. Comment out the add and subtract functions declared in the previous section “Declaring Functions”
2. Copy and modify your add function to include a default value for the second parameter (e.g., 5).
3. Similarly, modify the subtract function to include a default value for the second parameter.
4. Test the updated functions by calling them with one and two arguments in the developer console. Observe the behaviour when you omit the second argument.

Exercise 2C: Implementing Arrow Functions

1. Comment out the implementations of add and subtract from Exercise 2B.
2. Re-implement add and subtract using arrow function syntax.

3. Define two new functions, multiply and divide, using arrow function syntax.
    - The multiply function should return the product of two numbers.
    - The divide function should return the quotient of two numbers, where the **second argument is the divisor**. Prevent division by zero by logging an error message if the divisor is 0.

Test all four functions in the developer console to verify their functionality.

Exercise 2D: Understanding Callback Functions

1. Create a function named calculator with three parameters: num1, num2, and operation.
    - num1: the first number
    - num2: the second number
    - operation: a callback function that takes num1 and num2 in that order and returns the result of applying operation to them.
2. Inside calculator, call the provided callback function with the two numbers and return the result.
3. Test your calculator function in the developer console by passing it:
    - Two numbers and the add function as the callback.
    - Two numbers and the divide function as the callback.

Submission:

Upload the code for Exercise 2 to the same repository you created for Exercise 1.

Exercise 3: Managing Data with JavaScript

Objective: The objective of this exercise is to help students practice managing and manipulating data in JavaScript using arrays, objects, and destructuring. We will learn how to create, update, and access structured data, enabling them to build dynamic and organized programs.

Exercise 3A: Arrays and Basic Methods

1. Repeat the steps from "Create an HTML File" in Exercise 1 to create a new file named *exercise3.html.*
2. Create a JavaScript file named *exercise3.js* within the same folder as *exercise3.html*.
3. Replace the <script>…</script> block with <script src="./exercise3.js" defer></script> **All subsequent steps in this exercise should be done within *exercise3.js*.**
4. Declare an array named classRoster containing the following student names: Alice, Tom, Charlie, Diana, Evan
5. Use the toString method to convert the array into a string and log it to the console.
6. Log the initial classRoster to show that it remains unchanged after using toString.
7. Use a method (e.g., push) to add two more students: Fiona and Nancy.
8. Remove the first student in the list (Alice) using the shift method and log their name to the console. **Do not** remove a name by directly editing the array declaration.
   Hint: the shift method is used to remove the first element from an array and return it.
9. Log the updated array to the console and confirm all changes.
10. Log the length of the updated classRoster array to confirm the total number of students.

Exercise 3B: Objects with Nested Structures

1. Create an object named classInfo with the following properties:
   a. className: A string, 'ENSF381: Full-Stack Web Development'.
   b. instructor: A string, 'Dr. Smith'.
   c. students: The classRoster array created earlier.

   d. details: A nested object with these properties:
      - semester: 'Winter'
      - year: 2025

2. Add a new property, schedule, to the classInfo object. This property should be an array containing the class meeting days: *Monday, Wednesday, Friday*.
3. Update the instructor property to 'Dr. Abdellatif'. Use the appropriate syntax for this change (e.g., classInfo.instructor = 'Dr.Abdellatif'). **Do not** directly re-declare the object.
4. Log the values of the className, instructor, and students properties to the console.
5. Access and log the semester property from the nested details object.
6. Log the updated classInfo object to confirm all changes.
7. Destructure the className and students properties from classInfo into individual variables. Then, log these variables to the console to confirm their values.
8. Destructure the semester and year properties from the nested details object in classInfo into variables. Then, log these variables to confirm they are correctly assigned.
9. Destructure the first two student names from the classRoster array into variables named student1 and student2.
10. After destructuring, student1 and student2 should contain the first two names, and remainingStudents should include the rest.
11. Log student1, student2, and remainingStudents to confirm their values.

Submission:

1. Upload the code for Exercise 3 to the same repository you created for Exercise 1.
2. Compress your Exercise 1, Exercise 2, and Exercise 3 files into a single zip file and upload the compressed file to D2L. Also, upload the completed Answer_sheet.docx.