Web Testing Lab: JavaScript (Without Jest or Unit Testing Framework)

Introduction: In this lab, you'll focus on writing and running tests directly in JavaScript, without using any testing frameworks like Jest. You'll manually handle assertions and validate inputs using boundary tests, regular expressions, and dynamic parsing.

Materials: - A Code editor (e.g., VS Code, Atom) - Web browser (for running and testing JavaScript code)

Activity 1: Set Up a Simple HTML Page

1. Create a new folder for this activity and open your code editor inside it.
2. Write a basic HTML file (`index.html`):

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Grade Submission</title>
</head>
<body>
    <h1>Submit your grade</h1>
    <input type="number" id="gradeInput" placeholder="Enter your grade">
    <button id="submitBtn">Submit</button>
    <p id="message"></p>

    <script src="script.js"></script>
</body>
</html>
```

Activity 2: Handle Input and Display a Message

Goal: Handle the user's input (grade submission) and display a "Pass" or "Fail" message based on the boundary value of 50.

In your script.js, write the following code:

```
document.addEventListener('DOMContentLoaded', function() {
    // This ensures that the DOM is fully loaded before interacting with elements. We don't

    const submitBtn = document.getElementById('submitBtn');

    // Attach event listener to the button after the DOM is ready.
```

```
    submitBtn.addEventListener('click', function() {
        // Handle form submission
        const grade = parseInt(document.getElementById('gradeInput').value);
        const message = grade >= 40 ? 'Pass' : 'Fail';

        // Update the message element with the result
        document.getElementById('message').innerText = message;
    });
});
```

This will display:

"Pass" if the grade is 40 or higher, "Fail" if the grade is below 40, *A validation message if the input is outside of the 0-100 range.

Activity 3: Manual Testing for Boundary Values

Goal: Manually check your boundary values by entering grades like 0, 39, 40, 100, and invalid inputs (e.g., negative numbers, or values greater than 100). Here are some sample boundary cases to test:

- Test Case 1: Enter 49. Expected result: "Fail"
- Test Case 2: Enter 50. Expected result: "Pass"
- Test Case 3: Enter 100. Expected result: "Pass"
- Test Case 4: Enter -1. Expected result: "Invalid grade. Please enter a number between 0 and 100."
- Test Case 5: Enter 101. Expected result: "Invalid grade. Please enter a number between 0 and 100."
- Test Case 6: Enter abc (non-numeric input). Expected result: "Invalid grade. Please enter a number between 0 and 100."

You can run these tests manually by opening index.html in your browser, entering the values into the input box, and checking the results. Have a go at automating one test by manipulating the DOM in JS. If you open up the inspector (right click -> inspect) you will be able to access DOM elements programmatically. Right click -> Copy JS Path -> Switch tabs to the console and manipulate the object directly. Play around with the input box by setting things like the innerText = 'value' and simulate a button click by selecting the element then using the dot syntax e.g. button.click(). You can reuse some of your tests from the previous lab here if you want to build this up.

Activity 4: Regular Expressions in JavaScript

Goal: Let's add a feature to validate non-numeric inputs using regular expressions and check if the input is a valid number.

Replace the script.js logic with the following:

```
document.getElementById('submitBtn').addEventListener('click', function() {
    const input = document.getElementById('gradeInput').value;
    const numberPattern = /^[0-9]+$/; // Regular expression for numeric input

    if (!numberPattern.test(input)) {
        document.getElementById('message').innerText = 'Invalid input. Please enter only nu
        return;
    }

    const grade = parseInt(input);

    if (grade < 0 || grade > 100) {
        document.getElementById('message').innerText = 'Invalid grade. Please enter a numbe
    } else {
        const message = grade >= 40 ? 'Pass' : 'Fail';
        document.getElementById('message').innerText = message;
    }
});
```

What This Does: This uses a regular expression to ensure that only numeric values are accepted. If a user enters non-numeric characters, a validation message will be shown.

Try These Inputs: Enter letters like abc, 1a2, or symbols like !@#. The message should display: "Invalid input. Please enter only numbers."

Activity 5: Parsing Input Dynamically

Goal: Explore dynamic parsing of different input types and how JavaScript handles them.

Extend the functionality to handle different types of input—like floating-point numbers or handling spaces.

```
document.getElementById('submitBtn').addEventListener('click', function() {
    const input = document.getElementById('gradeInput').value.trim();
    // Remove leading/trailing spaces
    const numberPattern = /^[0-9]+(\.[0-9]+)?$/;

    // Accept integer and floating-point numbers

    if (!numberPattern.test(input)) {
        document.getElementById('message').innerText = 'Invalid input. Please enter a valid
        return;
    }

    const grade = parseFloat(input); // Use parseFloat to handle decimals
```

```
    if (grade < 0 || grade > 100) {
        document.getElementById('message').innerText = 'Invalid grade. Please enter a numbe
    } else {
        const message = grade >= 40 ? 'Pass' : 'Fail';
        document.getElementById('message').innerText = message;
    }
});
```

Key Points: Trim function removes leading/trailing spaces. Regular expression now supports both integers and floating-point numbers

`(^[0-9]+(\.[0-9]+)?$)`

parseFloat is used to handle decimal inputs. We could do more here. Read this. https://bito.ai/resources/sanitize-input-javascript-javascript-explained/

Activity 6

Let's have a go at something a bit more engaging.

Navigate to: https://doc.gold.ac.uk/~smcgr004/grades.html

You will want to save these to localstorage objects. We're going to show off all of the things you can do entirely client side, without needing a server! Cookies are also an option.

Run this script to parse the table. Make sure you look up how tables work in HTML as a reminder. https://www.w3schools.com/html/html_tables.asp

```
// Get the table element from the page
const table = document.querySelector('table');

// Parse table rows and cells
const rows = [...table.querySelectorAll('tr')].map(row => {
  const cells = [...row.querySelectorAll('td, th')];
  return cells.map(cell => cell.innerText.trim());
});

// Store the table data in sessionStorage
sessionStorage.setItem('gradesTable', JSON.stringify(rows));

console.log('Table data has been stored in sessionStorage:', rows);
```

This is now saved in a local storage object. If you change browsers, tabs etc then it will lose this. They're only held for a brief period. Cookies are slightly more persistent, but if you are working with lots of data manipulation you will want to host a server in Node or Python.

Now you can click the link at the bottom of the page that takes you to a checker. Remember some of the names and grades (or take a screenshot) and validate these inputs in the second form. Inspect the HTML to see how the Javascript is referencing your localstorage objects JSON.parse(sessionStorage.getItem('gradesTable')); against the inputs on the page.

Extension

If you want to copy and paste the two webpages from my website and run them locally you can. Some ideas of activities you could do:

- Work to parse the entire row of grades rather than just the first grade for each student.
- Work with this broken HTML table at https://doc.gold.ac.uk/~smcgr004/dodgy.html and introduce some of your error checking JS code from last week to either validate the data (correct) or replace it with a standard value. This could be 0, NA, NaN or maybe you want to calculate the mean, median or mode from a column/row and 'guesstimate' their grade? If your logic from the lab was sound it should 'catch' all of these issues. Did you miss something? Try combining some of the techniques we have seen today.
- Introduce some further error checking mechanisms. You can do this on the JSON sessionStorage object, treating it as if it's a database and then parse values as arrays.

```
// Assuming the data is stored in sessionStorage under the key 'gradesTable'
const storedGrades = JSON.parse(sessionStorage.getItem('gradesTable'));

// Get the entire session storage data for 'gradesTable'
console.log("All grades:", storedGrades);

// Filter and retrieve only the data where the first column (Student Name) is 'Michael Brown
const michaelBrownData = storedGrades.filter(row => row[0] === 'Michael Brown');

console.log("Michael Brown's data:", michaelBrownData[0]);
```

- Build some logic into the dodgy.html (aptly named) page to warn you if some grades are high. Hint: A z-score of ± 3.5 or higher typically indicates a data point is far away from the mean and could be considered an outlier. Calculate the mean then look for any data that's above or below that mean value by 3.5 standard deviations.

Selenium Introduction

If you want to go a bit further you can scrape the content, load it into Python (Pandas of course) and output it as an excel spreadsheet. Nifty! I'll leave it up to you decide how far you go with this, but Pandas gives you some pretty

hardcore capabilities. You can repeat the checks we've done today and do clever things like forward fills, backfills and even imputation to 'guesstimate' values using machine learning!

Boilerplate code. Will need adapting based on a) Selenium version b) OS c) Browser. Caveat emptor. Performance on the lab machines is mixed at best. You'll almost certainly have to tweak the sleep time depending on network traffic.

```
pip install selenium pandas openpyxl
```

```
from selenium import webdriver
from selenium.webdriver.common.by import By
import pandas as pd
import time

# Initialize the webdriver (make sure you have ChromeDriver or another WebDriver installed)
driver = webdriver.Chrome()  # You can specify the path to chromedriver here if it's not in

# Navigate to the webpage
url = "https://doc.gold.ac.uk/~smcgr004/dodgy.html"
driver.get(url)

# Give it a few seconds to load the page
time.sleep(2)

# Locate the table in the HTML page
table = driver.find_element(By.TAG_NAME, 'table')

# Extract all the rows
rows = table.find_elements(By.TAG_NAME, 'tr')

# Prepare data for pandas DataFrame
data = []
for row in rows:
    # Get all the columns (td elements) for each row
    cols = row.find_elements(By.TAG_NAME, 'td')
    # If it's a header row, use th instead of td
    if not cols:
        cols = row.find_elements(By.TAG_NAME, 'th')
    # Extract text for each column and append to the data list
    data.append([col.text for col in cols])

# Create a DataFrame from the extracted table data
df = pd.DataFrame(data)

# Close the browser window
```

```
driver.quit()

# Save the DataFrame as an Excel spreadsheet
df.to_excel('student_grades.xlsx', index=False, header=False)

print("Table has been saved to 'student_grades.xlsx'.")
```

Even more advanced: You can even go full hardcore with the unittests if you want to. See here https://selenium-python.readthedocs.io/getting-started.html