Due Date: Monday, April 07, 2025 @ 11:59 PM

Assignment 3: JavaScript Basics Practice

Objective: This assignment aims to practice fundamental JavaScript concepts, including variable initialization, conditional statements, loops, and problem-solving while utilizing unit tests to validate your solutions.

Description: In this assignment, you will work on five tasks to reinforce your understanding of JavaScript basics. We've provided you with a starter code file called `starter.js`. Your task is to rename this file to your student ID (e.g., `123123123.js`) and complete the code to solve each task according to the instructions. This assignment is designed to help you strengthen your foundational JavaScript skills in a fun and interactive way.

Objectives:

- Familiarize yourself with JavaScript variable initialization.
- Gain experience in working with conditional statements and loops.
- Practice using arrays and manipulating their elements.
- Develop problem-solving skills by completing various coding tasks.
- Improve your ability to include and interact with external JavaScript files in an HTML document.

Outcomes: Upon completing this assignment, you should be able to:

1. Initialize JavaScript variables with specific values.
2. Apply conditional statements to make decisions based on variable values.
3. Implement loops to perform repetitive tasks efficiently.
4. Manipulate arrays to achieve specific results.
5. Include external JavaScript files in HTML documents for enhanced functionality.

Assignment Structure:

- 10 Marks - Part One: Displaying and Encoding Information
- 18 Marks - Part 2: Comparing Numbers with a Twist
- 18 Marks - Part Three: Loop and Concatenation with Conditions
- 24 Marks - Part Four: Advanced Array Index and Value Analysis

- 30 Marks - Part Five: Array Rearrangement and Summation with Parity

Each part of the assignment corresponds to a specific JavaScript function in your `studentID.js` file. You will complete the code inside these functions to accomplish the specified tasks. Use function names from `part 1 to part 5`, and make sure to return values from these functions.

We provide an assignment3.html file to evaluate your code and verify its correctness. Please refrain from making any changes to this HTML file; it is used to run the unit tests for your code.

Setup and Configuration:

Follow these steps to set up and complete the JavaScript Basics Practice assignment:

1. **Download the Assignment Files:**

- Download the assignment files from https://comp1234.gblearn.com/winter2025/as3/.
- You should download the following files:
  - `assignment3.html`: This HTML file runs unit tests on your code.
  - `starter.js`: A starter JavaScript file containing function placeholders for each part of the assignment.
- For your convenience a zip file is placed on the same directory containing all the necessary files.

2. **Rename the Starter File:**

- Rename the `starter.js` file to your student ID and add the `.js` extension. For example, if your student ID is 123123123, rename the file to `123123123.js`. This renamed file will be where you write your code.
- Make sure to replace assignment3.html line 17 "starter.js" with your script name (`123123123.js`)

```
15
16      <!-- Your JavaScript file -->
17      <script src="starter.js"></script>
18
```

3. **Open Your Student JavaScript File:**

- Open your renamed JavaScript file (e.g., `123123123.js`) using a text editor of your choice.

4. **Complete Each Part:**

   - Inside your JavaScript file, you will find five functions named `Part 1 to Part 5`. Your task is to complete the code inside these functions for each part of the assignment, following the specific instructions provided for each part.

5. **Use the Return Statement:**

   - Except for the first function part, ensure you use the return statement within each function to return the expected values per the instructions.

6. **Evaluate the Result:**

   - To evaluate your code, open the `assignment3.html` file in a web browser. The tests will assess the correctness of your solutions for each part.

7. **Review the Browser Output:**

   - After completing each part, review the output displayed in your web browser. The browser will show whether your code passes the tests for each part.

8. **Repeat Steps 4-7 for Each Part:**

   - Complete and test each part of the assignment one by one.

9. **Submission:**

   - Once you have completed all parts and tested your code successfully, create a folder named `assignment03` under `comp1234/assignments`.
   - Upload your JavaScript file (e.g., `123123123.js`) and the `assignment3.html` file to the `assignment03` folder.
   - Make sure to comment "Your local test file" section and uncomment "Remote test file." (leaving the local test in this file will result in a zero grade.)

```
23    <!-- Your local test file
24    <script src="sampleTest.js"></script>
25    -->
26    <!-- Remote test file -->
27    <script src="https://comp1234.gblearn.com/winter2024/as3/winter2024-testa3.js"></script>
28
```

Due Date: Monday, April 07, 2025 @ 11:59 PM

10. Online Submission on GBLearn:

- Login to `GBLearn` using your GBLearn `username and password`.
- If you have not selected your lab instructor, please go to your profile and select your lab instructor before submission.
- Select `COMP1234` and "`Assignment 03`" to submit your assignment on GBLearn (you must upload your `studentID.js` file).
- Ensure you can access the assignment files and folder on your GBLearn via a browser at `https://username.gblearn.com/comp1234/assignments/assignment03/assignment3.html`.

11. You will receive a zero mark if one of the following occurs:

- You place your assignment files and directory in the wrong path.
- You forgot to upload your code to your GBLearn account.
- You forgot to submit your assignment via my.gblearn.com.

Important Notice on Code Similarity and Academic Integrity:

We utilize a similarity detection tool to scrutinize all code submissions for this assignment. Detected similarities will be considered plagiarism, leading to a zero grade for the involved students and a formal incident report. **Using the code provided by tutors or professors in your solution will be considered plagiarism.**

Upholding academic integrity is crucial for several reasons:

- **Promotes Original Thought**: It encourages personal engagement and creativity, which is vital for developing learning and problem-solving skills.
- **Ensures Equal Assessment**: Maintain a fair evaluation process where students are graded on merit and effort.
- **Builds Trust**: Supports a trustworthy academic environment where work accurately reflects student abilities.
- **Prepares for Professionalism**: Fosters ethical behaviour and responsibility, qualities essential in the professional world.

Due Date: Monday, April 07, 2025 @ 11:59 PM

Tips for completing this assignment successfully.

1. **Understand the Assignment Requirements**: Carefully read the instructions to ensure you fully understand what is expected in each part of the assignment.
2. **Plan Your Approach**: Before jumping into coding, take some time to plan your approach to each part of the assignment. Break down the tasks into smaller, manageable steps.
3. **Start Early**: Begin working on the assignment as soon as possible. This gives you ample time to tackle challenges and seek help if needed.
4. **Utilize MobiHelp Tutoring Services**:
    - Visit MobiHelp and log in using your `studentID@georgebrown.ca` and password.
    - Schedule a tutoring session, choosing between online or in-person options based on your preference and availability.
    - Prepare specific questions or areas where you need guidance to make the tutoring session as productive as possible.
5. **Implement the Solution**:
    - Apply the concepts discussed in class and the hints provided to start coding your solution.
    - Remember, tutors from MobiHelp will guide you through the problem-solving process but will not solve the problem for you.
6. **Test Your Code**: Thoroughly test your solution for each part of the assignment. Ensure your code meets the requirements and handles edge cases.
7. **Seek Feedback**: If possible, seek feedback on your approach and code from peers, tutors at MobiHelp, or during office hours with your instructor.
8. **Revise and Improve**: Based on feedback and further insights you might gain as you work through the assignment, revise and improve your code.
9. **Reflect on the Learning Process**: After completing the assignment, take some time to reflect on what you learned and how you can apply these insights to future tasks.

Following the instructions below and completing each part of the assignment within your JavaScript file, you will practice essential JavaScript concepts and demonstrate your problem-solving skills.

Good luck with the assignment, and enjoy the learning process!

Due Date: Monday, April 07, 2025 @ 11:59 PM

Part One: Displaying and Encoding Information

Objective: In this part of the assignment, you will practice initializing JavaScript variables with specific values and dynamically updating webpage content. You'll also apply basic text encoding and text formatting to differentiate between labels and values. This foundational skill set is crucial for manipulating and presenting data in your scripts.

Instructions:

1. **Function Overview**: Your studentID.js file provides a function skeleton for part 1. Use JavaScript to display your student ID, full name, professor's name, and lab session day in the browser window. Employ the `document.write` method for this purpose. Each piece of information should have a bold label (e.g., "Student ID:", "Full Name:", etc.), but the actual values should be displayed in regular font weight.

```
1  function part1(){
2      // Start your code here for first part. There is no need to return any value.
3
4
5  }
```

2. **JavaScript File Setup**: Open your renamed JavaScript file (`123123123.js`, replace it with your actual student ID) in a text editor of your choice.

3. **Variable Declaration and Initialization**: Declare and initialize variables in your JavaScript file with the provided values:
   - `Student ID`: [Your Student ID]
   - `Full Name`: [Your Full Name]
   - `Lab Professor`: [Professor's Name]
   - `Lab Session Day`: [Day of the Week]
   - `Semester`:[Current Semester]

4. **Formatting and Positioning**: Display the student ID and full name on one line and the professor's name and lab session day on the following line. Use inline CSS to format the labels in bold and position the content at the bottom right corner of the page. Remember, you cannot modify `assignment3.html`, so all styling must be done within your JavaScript code using `document.write`.

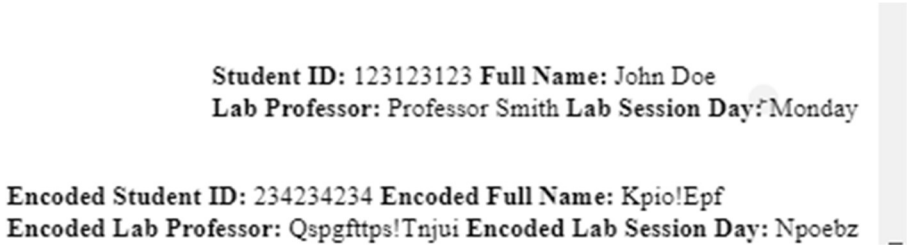
5. **Encoding**: Next, encode the same information set by shifting each character by 1 in the character set (e.g., 'A' becomes 'B', '1' becomes '2', etc.). Display this encoded information below the plain text, applying the same formatting rules with bold labels and non-bold values. Ensure this encoded information is also positioned at the bottom right corner of the screen.

   Show the student ID and full name in one line and the professor's name and lab day on the second line. `Apply inline CSS to` ensure the content is positioned at the right bottom of the page. (**You cannot modify assignment3.html**)

Testing Your Code:

1. Open the `assignment3.html` file in a web browser.
2. Review the browser's output to check if your code displays your student information, including the lab session day, correctly in a bold format.
3. If your code is correct, it will display your student information and the lab session day as specified.
4. Keep your browser console open to view the results and verify that your code is error-free.
5. Proceed to the next part once you have completed and tested this part successfully.

Output:



Hint for Part One:

- Begin by declaring and initializing variables for each piece of information you need to display. Think about what types of data you're dealing with and how you might represent them in JavaScript.
- Consider how you can use the **document.write** method to output HTML content to your webpage. Remember, this method can accept HTML strings, allowing you to incorporate HTML tags and attributes directly into your JavaScript code.

- Inline CSS can be applied directly within HTML elements using the **style** attribute. Explore how you can use this to position your content (using properties like **position**, **bottom**, and **left**) and to style text (e.g., making labels bold).
- To ensure the correct information is displayed in the proper format and position, carefully construct your HTML strings with the necessary CSS styling and concatenate your variables at the appropriate places.

- **Encoding Strategy**:
  - Encoding involves transforming the original characters into different characters based on a specific rule, such as shifting character codes.
  - Utilize **String.charCodeAt(index)** to get the character code for each character in your strings. This function is critical to understanding a character's numerical representation.
  - After altering the character codes to encode your information, turn these codes back into characters with **String.fromCharCode(...)**. This step is crucial for converting your encoded numeric values back into a readable string.

- **Iterating Over Strings**: Loop through each character in your strings to apply the encoding rule. JavaScript allows you to treat strings like arrays for this purpose, accessing each character by its index.

Part Two: Comparing Numbers with a Twist – Mark 18

Objective: This part of the assignment focuses on applying conditional statements in JavaScript with an added layer of complexity. By manipulating input parameters before comparison, you will deepen your understanding of basic arithmetic operations and conditional logic in JavaScript, all within the context of a function (part 2) designed for automated unit testing.

Instructions for the `part2` Function:

1. **Function Overview**: You are provided with a function skeleton for `part 2` in your `studentID.js` file. This function receives two numerical arguments, `num1` and `num2`.

Your task is to manipulate these numbers according to a specific rule, then compare the results and assign a return value based on this comparison.

2. **Code Implementation**:
   - Directly after the comment `// Your code should start here`.

```
function part2(num1, num2) {
    // Assign the return value to a variable named _return
    let _return = '';
    // Your code should start here


    /* Your code ends here.
       Don't add or change anything after this line.*/
    return _return;
}
```

   Implement the following steps within the `part2` function:

   a. **Twist**: Before comparing `num1` and `num2`, add 5 to `num1` and subtract 3 from `num2`. This introduces a "twist" to the original values, altering the comparison dynamics.

   b. **Comparison Logic**: Compare the modified `num1` and `num2` values according to these conditions:
   - If `num1` is now less than `num2`, assign -1 to `_return`.
   - If `num1` is equal to `num2`, assign 0 to `_return`.
   - If `num1` is greater than `num2`, assign 1 to `_return`.

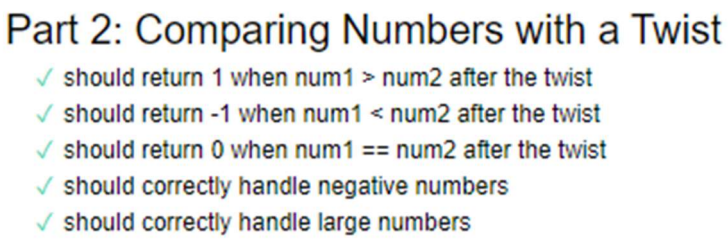
3. **Return Value**: Ensure that the variable `_return` holds the result of your comparison logic. The function will return this variable, allowing for automated unit testing of your solution.

Testing Your Code:

1. **Testing and Debugging**: After implementing your solution, test its correctness using the `assignment3.html` file, which includes unit tests for the `part2` function. These tests will automatically verify if your function returns the correct values for various input pairs.

2. **Review and Adjust**: Observe the test outcomes in your browser's console. Success messages will confirm correct implementation, while failures should prompt you to review and adjust your code as needed.

Expected Output:



Hint for Part Two:

- Reflect on how to apply a "twist" to your input numbers before comparing them. If the twist involves adding or subtracting a constant value to your numbers, think about how to perform these operations to ensure they affect the comparison as intended.
- Familiarize yourself with writing conditional statements (**if**, **else if**, **else**) in JavaScript. These will be crucial for comparing the adjusted values of **num1** and **num2** and determining the correct return value based on their relationship.
- The return values based on the comparison outcomes (-1, 0, 1) must be assigned to a variable. Consider how you can use a single variable to store your result throughout the conditional logic.
- Testing edge cases, such as negative numbers or large values, can help ensure your logic is robust. Consider how the "twist" might impact these scenarios differently and ensure your comparison logic accounts for these cases.

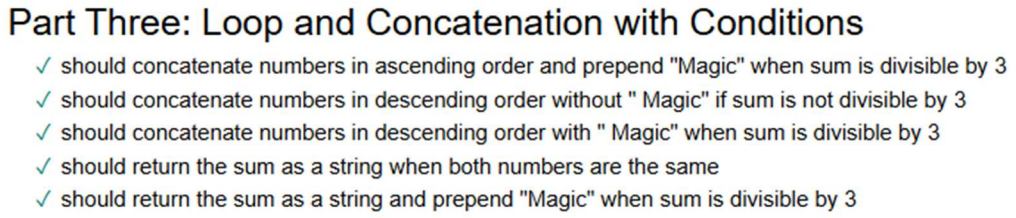
Part Three: Loop and Concatenation

Objective: In this part of the assignment, you will practice working with loops and concatenating strings in JavaScript. This skill is essential for tasks that involve repetitive operations on strings.

Instructions:

1. Open your `.js` file using a text editor.
2. Inside the `.js` file, you should add code to implement the following logic using JavaScript:
    - You will receive two integer values, `num1` and `num2`, as function parameters when the `part3(num1, num2)` function is called.
    - Your task is to check the values of `num1` and `num2`.
        1. If `num1` is less than `num2`, concatenate all the integers from `num1` to `num2` (inclusive) into a single string.
        2. If `num1` is greater than `num2`, concatenate the integers from `num1` to `num2` (inclusive) into a single string in reverse order (from high to low).
        3. If `num1` and `num2` are equal, such as both being 3, the result should be the sum of these two numbers as a string.
        4. Additionally, if the sum of all numbers concatenated is divisible by 3, prepend "Magic" to the result string.
    - For example, if `num1` is 2 and `num2` is 5, the resulting string should be "2345."
    - If `num1` is 5 and `num2` is 2, the resulting string should be "5432."
    - If `num1` and `num2` are equal, such as both being 3, the result should be "6" (the sum of 3 and 3) as a string.
    - If `num1` = 3 and `num2` = 30 sum is 33 _return should be Magic33 (no space between number and word Magic)
    - You must use a loop to achieve this concatenation or sum.
3. Save your changes to the `.js` file.
4. Open the `assignment3.html` file in a web browser to test your code.
5. Review the browser's output to check if your code correctly performs the concatenation or sum based on the provided values of `num1` and `num2`.
6. If your code is correct, it will display a success message for this part.
7. Keep your browser console open to view the results and verify that your code functions as expected.

Output:



Hint for Part Three:

- Think about how you can use a loop to traverse the array or range of numbers from **num1** to **num2** (and vice versa). Pay special attention to how loops can run both forwards and backwards.
- Consider determining whether a number is part of an ascending or descending sequence by comparing **num1** and **num2**.
- For concatenating numbers or calculating their sum, remember how string concatenation works in JavaScript and how you can use a loop to build up a string or sum incrementally.
- Explore the condition that requires a "Magic" modification. How can you check if a condition is met and alter your output accordingly?

Part Four: Advanced Array Index and Value Analysis – Mark 24

Objective: Enhance your skills in handling arrays and conditional statements in JavaScript. You will verify the validity of an array index, perform arithmetic operations based on the value at the index, and apply logic to handle special cases.

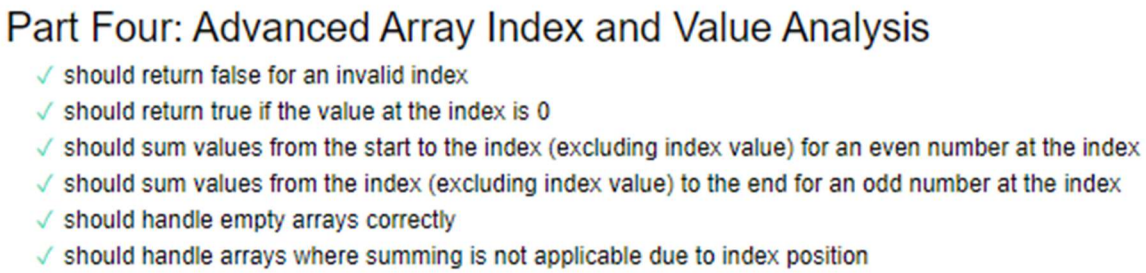
Instructions:

1. Open your `.js` file using a text editor.
2. Implementing the Logic:
    - Inside your `.js` file, add code to the `part4(array_index, array)` function. This function will be called with two parameters: `array_index`, an integer indicating an index in the array, and an `array` containing arbitrary-length integers.
    - **Index Validity Check**: Begin by checking if `array_index` is valid within the provided array. A valid index is greater than or equal to 0 and less than the array's length.
        1. If `array_index` is valid, proceed to the following steps to determine what value `_return` should hold based on additional conditions.
        2. If `array_index` is invalid (outside the array's boundaries), set `_return` to `false`.
    - **Conditional Summation**:
        1. If the value at `array_index` is **even**, sum the values from the beginning of the array to just before `array_index` (exclude the value at `array_index` itself). Set `_return` to this sum.

    2. If the value at `array_index` is **odd**, sum the values from just after `array_index` to the end of the array (exclude the value at `array_index` itself). Set `_return` to this sum.
    3. **Special Case**: If the **value** at `array_index` is **zero**, set `_return` to `true`, overriding the previous conditions.
  - Return the `_return` variable at the end of the function with the result of your logic.
3. **Save Your Work**: After implementing the function, save your changes to the `.js` file.
4. **Testing**: To test your function, open the `assignment3.html` file in a web browser. This HTML file should include unit tests or a testing mechanism that will call `part 4` with various inputs to evaluate its correctness.
5. **Review and Debug**:
  - Review the browser's output to verify that your function correctly handles all the conditions outlined, especially the validity of `array_index`, the conditional summing based on the even or odd value at `array_index`, and the special case when the value is zero.
  - If your implementation is correct, you should see a success message for this part of the assignment. Keep the browser console open to help identify and fix any issues in your code.
6. **Adjustments**: You may need to adjust your code based on the test results. Ensure all scenarios are correctly handled and re-test as necessary.

Note: These instructions are designed to help you add more sophisticated logic to your JavaScript functions, enhance your ability to work with arrays, conditional statements, and arithmetic operations, and write testable and maintainable code.

Output:



Hint for Part Four:

- Begin by understanding how to check if an index is within the bounds of an array. What makes an index valid or invalid?

- To tackle the problem of summing values up to a certain index, consider using a loop to iterate through a portion of an array. Remember that the range of your loop can be controlled by its starting and ending conditions.
- To check if a value is even or odd, recall how the modulo operator (%) can determine if a number is divisible by 2.
- Consider edge cases, such as an empty array or when special conditions about specific values (like 0) are involved. How will these cases affect your logic?

Part Five: Array Rearrangement and Summation with Parity – Mark 30

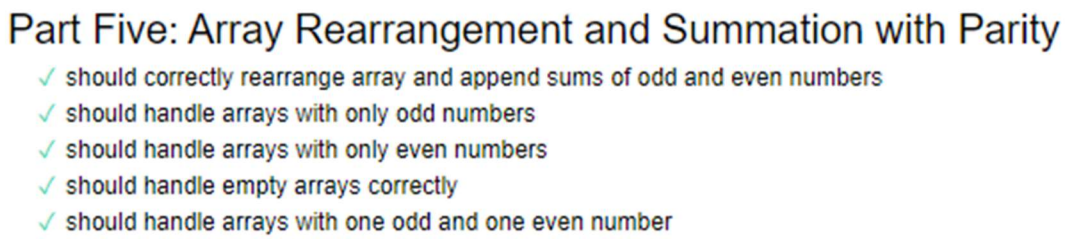
Objective: This part of the assignment enhances your proficiency with arrays, loops, and conditional logic in JavaScript. You will rearrange an array so that all odd numbers appear in the first half, all even numbers in the second half, and the sums of odd and even numbers are placed at the last two indexes of the array, respectively.

Instructions:

1. Open your `.js` file using a text editor.
2. **Implement the `part5` Function**:
    - The function `part5(array)` will be provided with a single parameter: `array`, which is an array of integers of arbitrary length.
    - Your task is to first loop through the `array` and segregate the odd and even numbers, keeping track of their sums separately.
    - Rearrange the array so that all odd numbers are placed in the first half and even numbers in the second half. Unlike the initial task, do not convert the sums into strings.
    - After segregation, append the sum of all odd numbers at the second-to-last index and the sum of all even numbers at the last index of the array.
    - **Note**: Adjust the original array length to accommodate the two sum values.
3. **Save Your Changes**: After coding the required logic, save the modifications to your `.js` file.
4. **Test Your Function**:
    - Open `assignment3.html` in a web browser to test the functionality of your updated `part5` function.
    - This HTML file should include scripts to run unit tests against your function, verifying that the array is correctly rearranged and the sums are accurately calculated and placed.

5. **Review and Debug**:
    - Check the browser's output to ensure your function behaves as expected, rearranging the array and appending the sums correctly.
    - Keep the browser console open to observe any error messages or warnings that can help you debug your code.
6. **Verification**:
    - Verify that the rearranged array maintains the correct order of odd and even numbers and that the sums of odd and even numbers are correctly appended.
    - Ensure your code handles edge cases, such as arrays with only odd or even numbers, and test with empty arrays to confirm robustness.

Output:

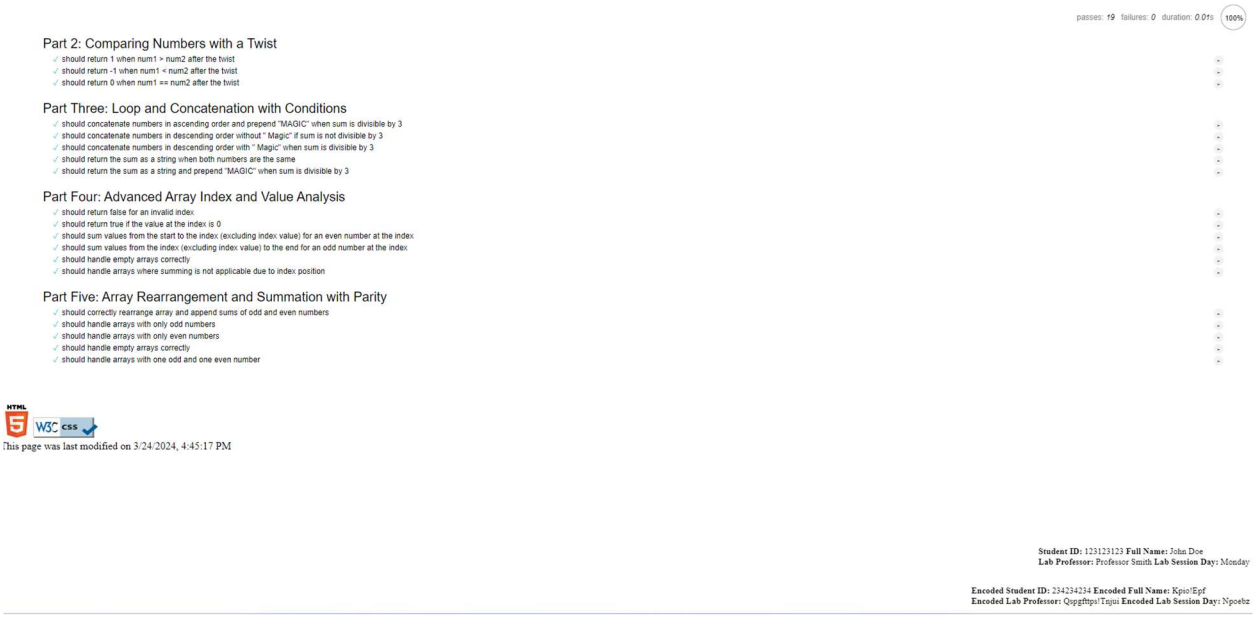


Hint for Part Five:

- First, consider how you might separate odd and even numbers. Can you think of a way to traverse the array once and categorize each element?
- When rearranging elements, you might find it helpful to create temporary arrays or lists to hold odd and even numbers separately before combining them. The built-in function **concat** can also be used.
- The requirement to place the sums of odd and even numbers at the end of the array requires additional calculations. How can you keep track of these sums as you categorize the numbers?
- Lastly, consider how the final array should be structured. After placing all odd and even numbers, how will you ensure the sums find their way to the correct positions? Remember, the length of your final array will differ from the original.

Sample Output:

Due Date: Monday, April 07, 2025 @ 11:59 PM



Reflection

In addition to the tasks outlined in the previous description, you must create a file called "assignment3-reflection.html" in the comp1234/assignments/assignment03 directory. Write about what you learned while completing this assignment. Some questions you may want to consider answering include:

- What challenges did you encounter while creating this assignment?
- How did you approach troubleshooting and debugging any of the code you wrote?
- What did you learn about the relationship between HTML, CSS and Javascript in web development projects?
- What did you find most challenging about the assignment, and how did you overcome it?

This reflection should be at least 250 words and written in proper English

Due Date: Monday, April 07, 2025 @ 11:59 PM

Submit AI Usage Declaration:

1. Download the AI Usage Declaration from D2L, complete it with details on any AI tools you used, and include your reflection in the declaration. (You are still required to submit this document even if you have not used the AI. In that case, at the beginning of the file, write, "I did not use AI to complete this assignment."

2. Convert the completed declaration to a PDF (aud.pdf).

3. ***Upload the aud.pdf to the Assignment03 folder*** and create a link to the document in your reflection HTML file.

Important Notes:

1. **AI Usage Declaration:** The declaration form requires you to detail any AI tools used, their purpose (e.g., coding assistance, idea generation), and how you refined the AI-generated content. **Failure to upload aud.pdf to the correct folder will result in a zero for the assignment**.

2. **Similarity Check:** An automated tool will check All code submissions for similarity. If any identical code is found across submissions, it will result in a grade of zero for all involved students. ***Ensure that your work is original and reflects your effort.***

Rubric: The total mark for each part is calculated based on the number of tests for each part. Example 18 / 6 (tests) each test worth 3 marks.

Criteria	**Marks**
Part One: Initializing Variables	
Four Variable Declaration and Assignment	2
Inline CSS Application (bottom, left, bold label)	3
Encode information	3
Correct Information Display	2
Total for Part One	*10*

Due Date: Monday, April 07, 2025 @ 11:59 PM

Part Two: Comparing Numbers with a Twist	
num1 > num2 (After Adjustment): Returns 1	3.6
num1 < num2 (After Adjustment): Returns -1	3.6
num1 == num2: Returns 0	3.6
Handle negative numbers	3.6
Handle large numbers	3.6
Total for Part Two	*18*
Part Three: Loop and Concatenation with Conditions	
Ascending Sequence for num1 < num2 with Magic	3.6
Descending Sequence for num1 > num2 with Magic	3.6
Descending Sequence for num1 > num2 without Magic	3.6
Correct Handling of num1 == num2 (Sum as string)	3.6
"Magic" Condition Applied Appropriately	3.6
Total for Part Three	*18*
Part Four: Advanced Array Index and Value Analysis	
Valid Index within Array Bounds	4
Sum Calculation for Even Values Appropriately	4
Sum Calculation for Odd Values Appropriately	4
Handling Special Case for Index Value 0 Correctly	4

Handle empty arrays	4
Handle non-sum arrays	4
Total for Part Four	*24*
Part Five: Array Rearrangement and Summation with Parity	
Correct Rearrangement of Odd and Even Numbers Correct Calculation and Placement of Sums	6
Handling Special Cases (Only Odd)	6
Handling Special Cases (Only Even)	6
Handling Special Cases (Empty Array)	6
handle arrays with one odd and one even number	6
Total for Part Five	*30*
Overall Quality & Organization	
Code Organization and Structure	-10
Code Comments and Readability	-10
Grand Total	100