Topic: JavaScript

Title: Task 1 - Palindrome Checker

Objective
The objective of this task is to get you more comfortable working with strings, arrays and functions in JavaScript. You are expected to build a simple app that checks if a user-entered string is a palindrome. [A palindrome is a word that is the same backwards and forwards eg racecar, level and dad]. There are a few different ways to achieve this, use the one you understand best. Research will be required.

The requirements of the app are as follows:
1. There should be a UI (User Interface) where the user is going to interact with the app. It should have the following inputs:
    a. A text input to capture what the user enters
    b. A button the user can press when done entering the word, bind this to a function as stated on 2 below.
2. A function that will contain the logic to check if a word is a palindrome, and also return a result to the user (on the browser) letting them know if it is or not
3. Your app should store all the previously checked words with their results in an array. And show this to the user (on the browser - this data does not need to be persistent). Find the best way to lay out your UI to allow for this.

Concepts the task covers:
1. Strings and string methods
2. Arrays and array methods
3. Functions and function calls
4. DOM manipulation (Updating the DOM from JavaScript)

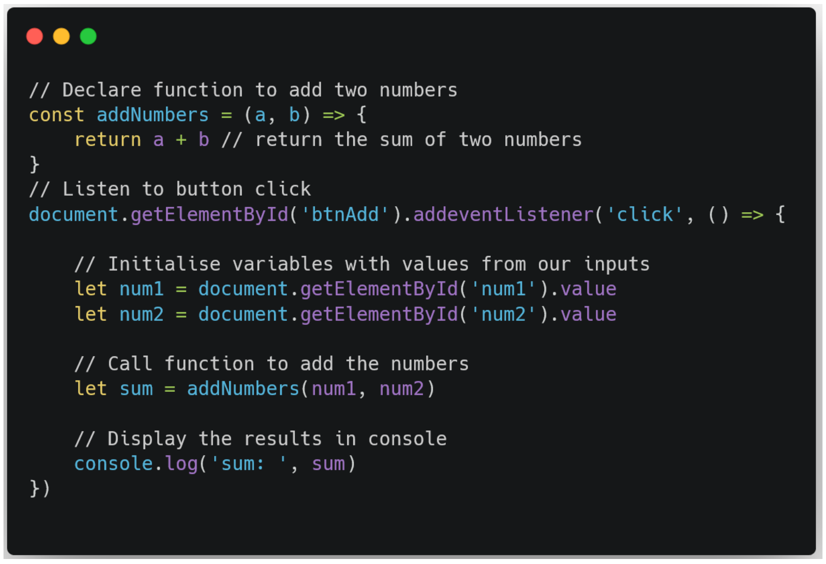
Instructions:
1. Create a JavaScript/HTML project for the task
2. In terms of CSS the portal should be as user friendly as possible so element lay out is important
3. Data is not required to be persistent, it can be refreshed with every reload.
4. You are allowed to use libraries in this project as long as they do not get in the way of you addressing the requirements of the app. Eg Tailwind for CSS
5. You are to submit in the form of links, either a github link or live link of the app. Sending code files will not count as a submission.

Tools used in solution planning:
We will use algorithms and pseudocode, together with flowcharts to plan through the project. There are other processes and tools used when planning but these are the only three we will focus on.

The internet is filled with different explanations, definitions and examples of what algorithms and pseudocode are but from hereon forth we will use the terms according to the explanations below.

Terms:
Algorithm:
A set of step-by-step instructions needed to be executed in order to solve a problem. This term can be used to identify both the plain language planning of the steps needed to solve a problem in programming and the actual steps written in code. We will use the word algorithm to identify the steps written in plain language.

An example of an algorithm in actual code for adding two numbers is



```
// Declare function to add two numbers
const addNumbers = (a, b) => {
    return a + b // return the sum of two numbers
}
// Listen to button click
document.getElementById('btnAdd').addeventListener('click', () => {

    // Initialise variables with values from our inputs
    let num1 = document.getElementById('num1').value
    let num2 = document.getElementById('num2').value

    // Call function to add the numbers
    let sum = addNumbers(num1, num2)

    // Display the results in console
    console.log('sum: ', sum)
})
```

An example of plain language algorithm for the same issue is

1. Display UI with two text inputs for two numbers, and a button
2. Create a function that adds two numbers and returns the sum, call it addNumbers. addNumbers should expect two parameters called “a” and “b”
3. Listen to the button click event
4. When button is clicked
    a. Initialize two variables to hold the values from our text inputs
    b. Declare a variable called sum. Call the function addNumbers and parse it the variables with our values. Set this function call as our value for our sum variable.
    c. Display the result on the console

    d. [Go back to listening to button click event (Not usually required but can be included if it helps with visualising how the program runs)]

5. [When button is not clicked, continue listening to button click event (Also not required, but helps with understanding how events are getting triggered and when the program starts executing specific functions)]

Flowcharts:

Flowcharts provide a graphical representation of the processes that are required to run in the program in order to solve the problem. This is usually drawn up according to the plain language algorithm, it aims to show exactly how the system gets inputs/data from users or other processes, what it does to that input and what outputs are expected. There are shapes reserved for every step eg a diamond for conditions, a rectangle for processes, etc. An example of this is provided with the palindrome planning but explanation of the shapes can be found here. Further reading in addition to the provided link might be required

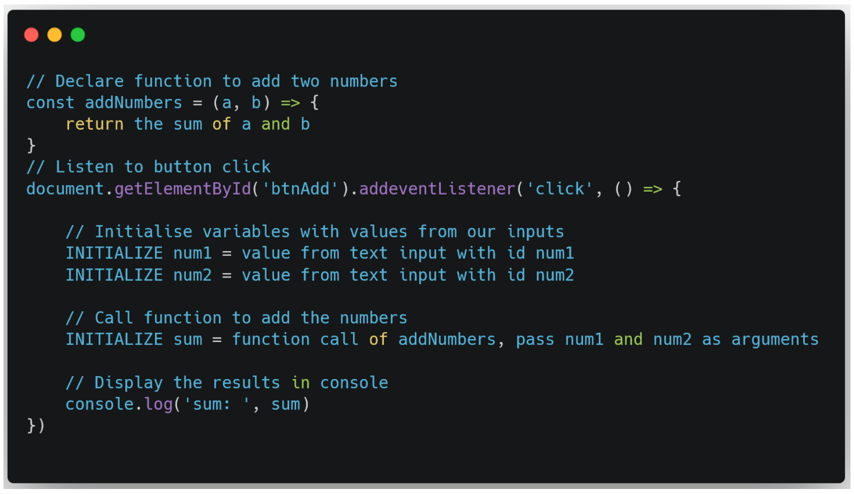
Pseudocode:

Pseudocode is a combination of the words pseudo (fake) and code, because it resembles both code and also contains words from simple language which are not used in programming. It provides a bridge between plain language algorithms, and actual code, in a way self-documenting what the code to be written will actually be doing.

There are no strict rules like with actual code, however there are keywords that people tend to utilise eg FUNCTION or SUBROUTINE to identify functions, and INITIALIZE to identify variable declarations. These words are usually taken from actual programming languages and provide ways to identify common programming concepts such as looping, declarations, conditional statements etc. Pseudocode is usually written after and according to the plain language algorithm. Which then provides a guide for the actual code.

NB: Pseudocode isn't real code so it can't be compiled, interpreted, run, or executed.

An example of this is shown in the below figure.



```
// Declare function to add two numbers
const addNumbers = (a, b) => {
    return the sum of a and b
}
// Listen to button click
document.getElementById('btnAdd').addeventListener('click', () => {

    // Initialise variables with values from our inputs
    INITIALIZE num1 = value from text input with id num1
    INITIALIZE num2 = value from text input with id num2

    // Call function to add the numbers
    INITIALIZE sum = function call of addNumbers, pass num1 and num2 as arguments

    // Display the results in console
    console.log('sum: ', sum)
})
```

Since I can be vague or specific [because pseudocode isn't strict], I chose to be specific on the addEventListener but less specific on the other parts. This is a step developers do for their own benefit and isn't usually required with the planning documentation. But helps lay out the logical steps required to solve the problem in a manner that's simple to read, understand and then translate to actual code. It is a step that is underutilized in many projects, but very beneficial when used.

Planning
Planning the solution to checking if a word is a palindrome. We already have the user requirements as provided above in the project specification. So we only need to continue planning from the given requirements. The next steps developers can go through is

1. Creating a plain language algorithm
2. Creating a flowchart to visually exhibit how the program will work
3. Creating pseudocode based on the flowchart
4. [Using the pseudocode to write the actual working code]

Solution 1:
Algorithm:

1. Display the UI
2. Create a variable called enteredText
3. Create a variable called enteredTextArr
4. Create a variable called reversedTextArr: to store the reversal of enteredTextArr
5. Create a variable called reversedText:
6. Listen to the onclick event of a submit button
7. Receiving the string value entered by user and store it in enteredText
8. Split enteredText into individual letters and store the returned array of characters into enteredTextArr
9. Reverse enteredTextArr and store this reversal in reversedTextArr
10. Combine the individual characters into one word and store it in reversedText
11. Use if statement to check if enteredText is equal to reversedText
12. If they are equal, output (console.log) “It is a palindrome”
    a. Append the enteredText and the result to the DOM
13. If they are not equal, output “It is not a palindrome”
    a. Append the enteredText and the result to the DOM

Solution 2:
High Level Algorithm:

1. Display the UI
2. Declare variables that will be used to store data
3. Listen for the button onclick event. When clicked do 4 till 7
4. Get the user entered text
5. Reverse the text
6. Check if text is equal to reversal of the text (Entered text is the same forward and backwards)
7. Show user the result

Detailed Algorithm:

1. Display the UI
2. Declare variables that will be used
    a. Create a variable called enteredText: (that will store the text entered by user)
    b. Create a variable called enteredTextArr
    c. Create a variable called reversedTextArr: to store the reversal of enteredTextArr

    d. Create a variable called `reversedText`: to store the word that will be formed after joining the individual characters in reversedTextArr

3. Listen for the button onclick event
4. Get the user entered text
    a. Fetch the text in the textbox on the DOM
    b. Save this text in the `enteredText` variable
5. Reverse the text
    a. Split enteredText into individual letters and store the returned array in `enteredTextArr`
    b. Reverse `enteredTextArr` and store this reversal in `reversedTextArr`
    c. Combine the individual characters/items of `reversedTextArr` into one word and store them in `reversedText`
6. Check if text is equal to the reversal of the text
    a. Use if statement to check if enteredText is equal to reversedText
7. Show user the result
    a. Create a variable called `domView`
    b. Get a reference to the DOM element to be used for displaying results and set domView to this ref
    c. Create a variable called `resultView`
    d. Create a new DOM element and set resultView to this new element
    e. Add the result to `resultView`
    f. Append `resultView` to `domView` to display the results (Add `resultView` as a child to `domView` so it's visible on the DOM)

Flowchart:

1. Provided in file "2. Palindrome Checker Flowchart"

Pseudocode:

2. Provided in file "planning (part 3).pseudo" and a refactored version in "refactored_planning (part 4).pseudo".

   The .pseudo file extension indicates that the file contains pseudocode. This allows keywords used in pseudocode to be highlighted just like in real code for easier reading.