PER SCHOLAS

ALAB 308.1.1:

Data Manipulation with JavaScript

Version 1.0, 10/11/23
Click here to open in a separate window.

Introduction

This assignment will walk you through a few small activities to familiarize you with data manipulation and processing using JavaScript. By becoming more familiar with the logic behind programming decisions, you will be able to create more efficient solutions in the future.

Objectives

- Create variable declarations using both let and const.
- Apply different types of data literals.
- Use arithmetic operators to manipulate data.
- Use comparison operators to compare data.
- Perform string concatenations.
- Implement escape sequences in strings for special characters.
- Use template literals for string interpolation and multi-line strings.
- Create effective documentation through the use of comments.

Submission

Submit your completed lab using the **Start Assignment** button on the assignment page in Canvas.

Your submission should include:

- A GitHub link to your completed project repository.

Instructions

Initialize a new git repository in a local project folder and create a JavaScript file (.js) to contain your code. Within your code editor of choice (we recommend Visual Studio Code), complete the following activities.

Use comments within your file to separate the code for each activity. Also use comments throughout the code to increase readability and explain your process where necessary. You can also use comments to make notes for yourself if you cannot figure something out and need to return to it at a later time.

Commit frequently! Every time something works, you should commit it. Remember, you can always go back to a previous commit if something breaks. As an optional goal, create a separate branch for each of the activities and merge it into your main branch when you are finished with that activity.

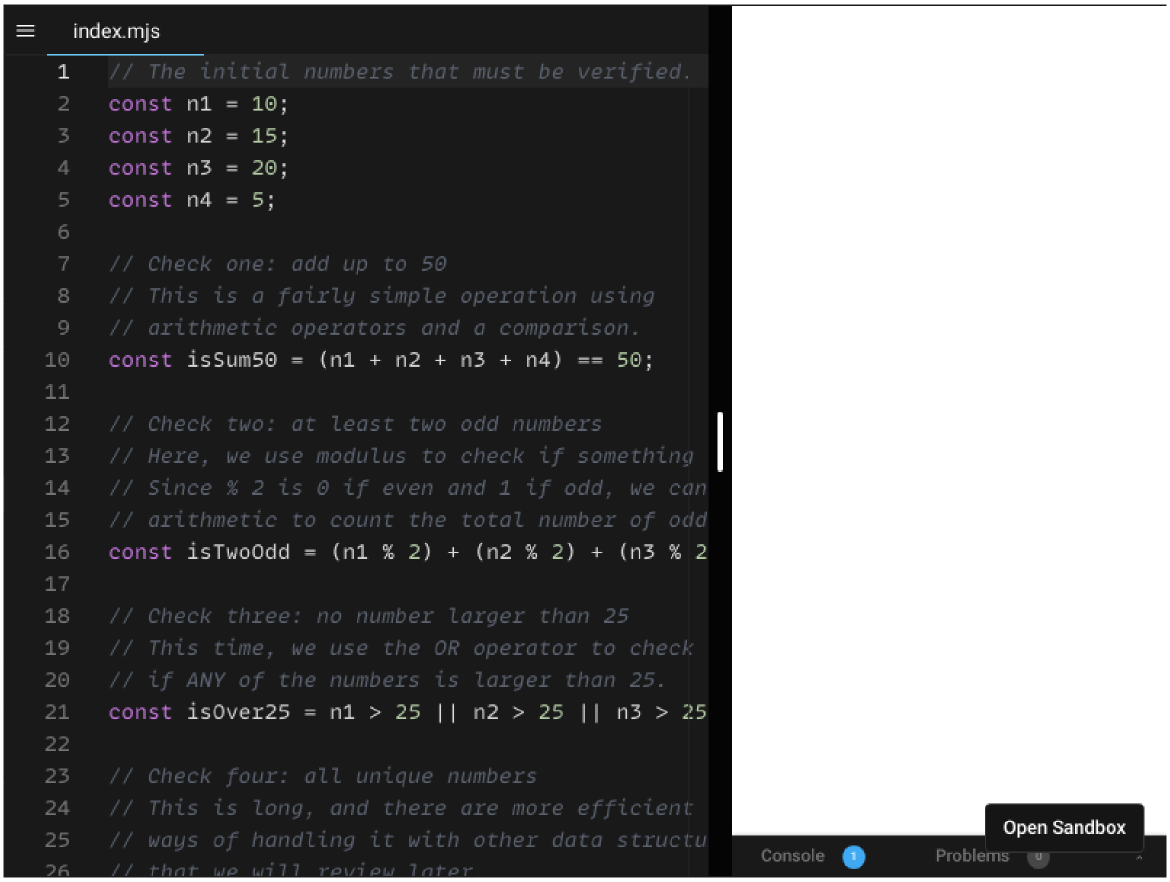
Part 1: Math Problems

Math is hard, which is why we have the computer solve the math problems for us. Often in programming, you will need to determine the logic behind an operation, and then tell the computer how to do it. Once you have established that logic, it can be reused. This is how we handle repetitive tasks.

For example, consider the following problem:

We have four numbers that need to add up to 50. At least two of these numbers must be odd. No number can be larger than 25, and each number must be unique.

How can we verify this? Check out the following CodeSandbox for an example. You can change the values of each of the initial numbers, n1 through n4, to see how the code responds in the console.



```
// The initial numbers that must be verified.
const n1 = 10;
const n2 = 15;
const n3 = 20;
const n4 = 5;

// Check one: add up to 50
// This is a fairly simple operation using
// arithmetic operators and a comparison.
const isSum50 = (n1 + n2 + n3 + n4) == 50;

// Check two: at least two odd numbers
// Here, we use modulus to check if something
// Since % 2 is 0 if even and 1 if odd, we can
// arithmetic to count the total number of odd
const isTwoOdd = (n1 % 2) + (n2 % 2) + (n3 % 2

// Check three: no number larger than 25
// This time, we use the OR operator to check
// if ANY of the numbers is larger than 25.
const isOver25 = n1 > 25 || n2 > 25 || n3 > 25

// Check four: all unique numbers
// This is long, and there are more efficient
// ways of handling it with other data structu
// that we will review later
```

Copy the code from the editor into your own file to use as a starting point. Make sure to commit!

Now that we have verified that these numbers meet the criteria outlined, we can continue to use them in other ways, depending on what the program is attempting to accomplish. For our purposes, we are going to explore other comparisons and operations.

Implement the following:

- Check if all numbers are divisible by 5. Cache the result in a variable.
- Check if the first number is larger than the last. Cache the result in a variable.
- Accomplish the following arithmetic chain:
  - Subtract the first number from the second number.
  - Multiply the result by the third number.

    - Find the remainder of dividing the result by the fourth number.
- Change the way that isOver25 calculates so that we do not need to use the NOT operator (!) in other logic comparisons. Rename the variable as appropriate.

These are all arbitrary problems, but they demonstrate the simplicity of working with arithmetic and comparisons in programming. No matter the complexity of the task in front of you, you will always use these core operators to accomplish it.

Before continuing, go back through your code and add console log statements for each check. Use string concatenation or template literals to format the outputs in a reasonable way.

For example, instead of console.log(isValid), we could write console.log (`The four numbers are valid according to the provided criteria: ${isValid}.`) or something similar.

Part 2: Practical Math

Let's look at a more practical scenario.

You are planning a cross-country road trip!

- The distance of the trip, in total, is 1,500 miles.
- Your car's fuel efficiency is as follows:
    - At 55 miles per hour, you get 30 miles per gallon.
    - At 60 miles per hour, you get 28 miles per gallon.
    - At 75 miles per hour, you get 23 miles per gallon.
- You have a fuel budget of $175.
- The average cost of fuel is $3 per gallon.

Set up a program to answer the following questions:

- How many gallons of fuel will you need for the entire trip?
- Will your budget be enough to cover the fuel expense?
- How long will the trip take, in hours?

Compare the results when traveling at an average of 55, 60, and 75 miles per hour. Which makes the most sense for the trip?

Log the results of your calculations using string concatenation or template literals.

Part 3: Future Exploration

In all of the problems above, there are two tools that would make it much easier to implement the logic we need and to reuse the code to test with different values:

- Control flow, which is how we conditionally determine what a program does next. For example, if we exceed our fuel budget, perhaps the program could automatically change the speed it is testing until it finds the optimal speed.
- Functions, which are reusable blocks of code, allow us to run pieces of code multiple times without rewriting the code or changing the value of variables in the code itself, like we needed to do above.

We will explore control flow in the next lesson, and functions toward the end of the module. Until then, continue exploring with the variety of operators and data types available in JavaScript!