PER SCHOLAS

ALAB 308.5.1:

Creating Reusable Functions

Version 1.0, 10/13/23

Click here to open in a separate window.

Introduction

This assignment walks you through implementing one of the most crucial guidelines in programming, “Don’t Repeat Yourself” (DRY), through the use of functions. Creating reusable functions is one of the most important skills to build upon as you continue to grow as a programmer.

Objectives

- Create reusable blocks of code with functions.
- Implement Array object methods.

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 to contain your code. Complete the activities below.

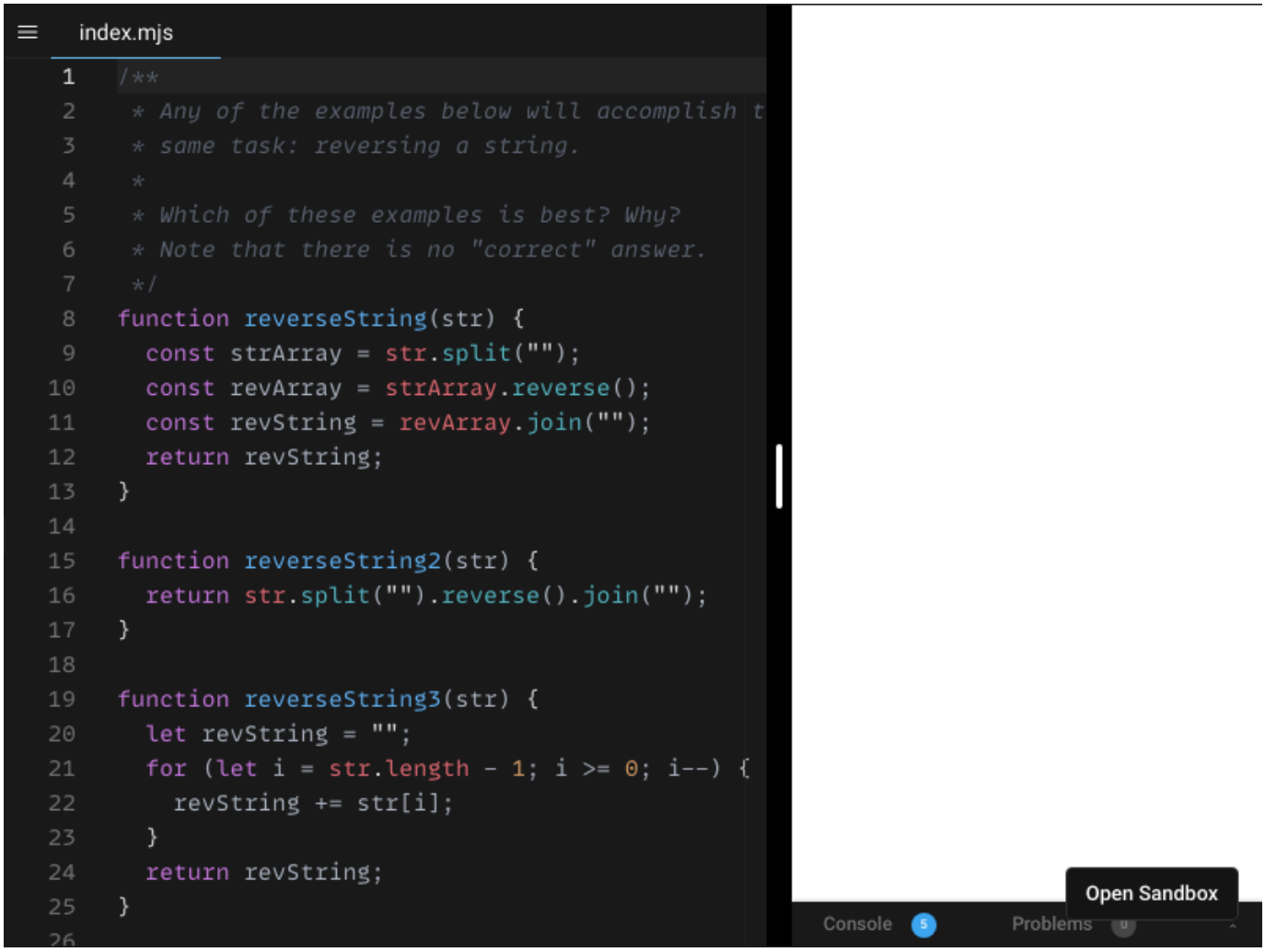
Commit frequently! Every time something works, you should commit it. Remember, you can always go back to a previous commit if something breaks.

Part 1: Thinking Functionally

When coding, it is important to approach your work using small, manageable blocks of code. Some functions may become dozens or hundreds of lines long, but keeping things small and simple will help you scale and maintain your code.

This section will have you build a few simple functions to accomplish arbitrary tasks. When building functions, remember that there are many ways to accomplish a task in programming. Sometimes, the shortest route is the best, and sometimes it is not.

Take the following example, which contains five functions that accomplish the same task. If you were looking at this code for the first time, which would make the most sense to you?



While there is rarely a “correct” answer in programming, it is important to keep your audience (other programmers) in mind. Write functions with descriptive names, and clear inputs and outputs.

With that in mind, write functions that accomplish the following:

- Take an array of numbers and return the sum.
- Take an array of numbers and return the average.
- Take an array of strings and return the longest string.
- Take an array of strings, and a number and return an array of the strings that are longer than the given number.
  - For example, `stringsLongerThan(['say', 'hello', 'in', 'the', 'morning'], 3);` would return `["hello", "morning"]`.
- Take a number, `n`, and print every number between 1 and `n` *without* using loops. Use recursion.

Part 2: Thinking Methodically

When functions are built into objects, like Arrays, they are referred to as “methods” of those objects. Many methods, including Array methods, require “callback functions” to determine their behavior.

For the tasks below, use the following data to test your work:

```
[{ id: "42", name: "Bruce", occupation: "Knight", age: "41" },
 { id: "48", name: "Barry", occupation: "Runner", age: "25" },
 { id: "57", name: "Bob", occupation: "Fry Cook", age: "19" },
 { id: "63", name: "Blaine", occupation: "Quiz Master", age: "58" },
 { id: "7", name: "Bilbo", occupation: "None", age: "111" }]
```

Use callback functions alongside Array methods to accomplish the following:

- Sort the array by age.
- Filter the array to remove entries with an age greater than 50.
- Map the array to change the “occupation” key to “job” and increment every age by 1.
- Use the reduce method to calculate the sum of the ages.
    - Then use the result to calculate the average age.

Part 3: Thinking Critically

It is important to remember that when working with objects in JavaScript, we can either pass those objects into functions **by value** or **by reference**. This important distinction changes the way that functions behave, and can have large impacts on the way a program executes.

For this section, develop functions that accomplish the following:

- Take an object and increment its age field.
- Take an object, make a copy, and increment the age field of the copy. Return the copy.

For each of the functions above, if the object does not yet contain an age field, create one and set it to 0. Also, add (or modify, as appropriate) an updated_at field that stores a Date object with the current time.

Thought experiment: since the Date object is an object, what would happen if we modified it in the copy of the object created in the second function using setTime() or another method? How could we circumvent potentially undesired behavior?

Part 4: Thinking Practically

Practical application of these concepts varies greatly in industry, but the core foundations are the same: functions handle repeated, specialized tasks, and methods are functions attached to specific types of objects.

The Skills-Based Assessment (SBA) for this module will have you work on a real-world example that employs all of the tools you have learned thus far. To prepare for it, revisit your previous work as described below.

Part 5: Thinking Back

Once you have completed the tasks outlined above, take any extra time you may have to revisit your previous JavaScript assignments.

- How many of the scripts could be turned into functions?
- What would the parameters look like? What kind of returns should they have?
- Could you package your code into even smaller blocks, creating helper functions?
- What else could be changed to optimize the code, knowing what you now know?

Explore, experiment, and experience the magic of reusable code!