JavaScript Challenge 6 - Shape Drawing Tool Exercise

Introduction

In this exercise, you will be working on implementing the JavaScript functionality for a shape drawing tool. This is an interactive application that allows users to draw shapes on a canvas by selecting a specific shape tool and clicking on the drawing area.

You will be provided with the HTML and CSS code for the layout and styling of the tool.

Video Demonstration:

Click here to view the video demonstration.

Main Focus: This exercise is specifically designed to practice implementing closures to add event listeners.

Understanding Closures

Before we dive into the exercise, it's essential to have a basic understanding of closures in JavaScript. A closure is a function bundled together with references to its surrounding state, or the lexical environment within which it was created. This allows the closure to remember this state even after the parent function has completed execution.

In the context of event listeners, closures can be incredibly beneficial. They allow you to encapsulate specific behavior or information related to an event, maintaining this information for future use.

Canvas API

In this exercise you will work with the Canvas API. For further reading on the Canvas API, you can refer to the following articles:

- MDN: Canvas API
- W3Schools: HTML Canvas Reference

Exercise Instructions

Step 1: Analyze the Provided Code

Review the HTML and CSS code, and familiarize yourself with the layout and elements. Watch the video demonstration to understand the expected functionality.

Step 2: Define Variables

2.1 Select the Tools and Canvas

- Use `querySelectorAll` to select all elements with the class "tool." This will give you a NodeList of the tool elements.
- Use `getElementById` to select the canvas element with the ID "drawingArea."
- Store these selections in variables `tools` and `canvas`, respectively.

2.2 Get the Canvas Rendering Context

- Use the `getContext` method on the canvas element to get the 2D rendering context. Store this in a variable called `ctx`.

Step 3: Implement Shape Drawing with Closures

3.1 Define a Draw Shape Function

- Create a function called `drawShape` that accepts a `shape` parameter (either 'rectangle' or 'circle').
- Inside this function, return another function (closure) that takes an `event` parameter.

3.2 Draw Specific Shapes Using Canvas API

- In the closure, begin the drawing path by calling `ctx.beginPath()`.
- Use an if-else statement to check the value of the `shape` parameter:
  - If it's 'rectangle', use `ctx.rect` to draw a rectangle at the mouse click position.
  - If it's 'circle', use `ctx.arc` to draw a circle at the mouse click position.
- Call `ctx.stroke()` to apply the drawing to the canvas.

Step 4: Attach Event Listeners to Tools

4.1 Iterate Over the Tools

- Use `forEach` to iterate over the `tools` NodeList.
- Inside the loop, use the `dataset` property to get the value of the `data-tool` attribute (either 'rectangle' or 'circle'). Store this in a variable called `shape`.

4.2 Add Click Event Listener to Each Tool

- Still inside the loop, add a click event listener to the current tool element.
- In the event listener callback, add another click event listener to the canvas, calling the closure from `drawShape` and passing the specific `shape`. Use the `{ once: true }` option to ensure that the listener is removed after being called once.

Step 5: Test Your Application

5.1 Manual Testing

- Click on the tool elements and then click on the canvas. Verify that the shapes are drawn correctly.
- Test with both 'rectangle' and 'circle' shapes.

5.2 Debugging

- If anything doesn't work as expected, use console logs to debug your code and ensure that variables and functions are behaving as intended.

Optional Step: Add Enhancements

Feel free to add enhancements, such as additional shapes, colors, or other interactive features, to make the tool more robust.

Conclusion

This exercise is designed to provide hands-on practice with closures and event listeners in JavaScript. By completing this exercise, you will gain a deeper understanding of how closures work and how they can be used effectively in real-world applications. Make sure to refer to the linked articles if you need additional clarification on closures.

Good luck, and happy coding!