Javascript-Scope and hoisting

Objective

In this exercise, you will learn about **hoisting** and **scope** in JavaScript. Hoisting refers to the behavior where variables and function declarations are moved to the top of their containing scope during the compilation phase. Scope refers to the accessibility of variables, functions, and objects in different parts of the code.

Understanding the Code

You are provided with a blank `index.js` file. Your task is to fill in the file by completing the following steps.

Here's what you need to do:

Step 1: Hoisting with `var`

- **Task**: Declare a variable `myVar` using `var` and assign it the value `'Hello, world!'`. Print the value of `myVar` before its assignment.

- **Expected Behavior**: Before assignment, `myVar` should print `undefined` because `var` declarations are hoisted but not initialized.

Step 2: Hoisting with `let`

- **Task**: Declare a variable `myLet` using `let` with value 'Hello, let!'. Attempt to print the value of `myLet` **before** its initialization.

- **Expected Behavior**: Trying to access `myLet` before initialization will throw a `ReferenceError` due to **hoisting** with `let`. The error message should be logged as `Cannot access 'myLet' before initialization`. Handle the `ReferenceError` by catching it

Step 3: Hoisting with `const`

- **Task**: Declare a variable `myConst` using `const` with value as 'Hello, const!'. Attempt to print the value of `myConst` **before** its initialization.

- **Expected Behavior**: Trying to access `myConst` before initialization will throw a `ReferenceError` because `const` also behaves like `let` in terms of hoisting but does not

allow access before initialization. Log the error message: `Cannot access 'myConst' before initialization`. Handle the `ReferenceError` by catching it.

Step 4: Function Scope with `var`

- **Task**: Create a function called `testVarScope` that declares a variable `varInFunction` using `var` with value 'Inside function scope'. Print the value of `varInFunction` inside the function.

- **Expected Behavior**: Inside the function, `varInFunction` should print `'Inside function scope'`.

- Invoke `testVarScope`.

Step 5: Block Scope with `let` and `const`

- **Task**: Create a function called `testLetConstScope` that declares two variables: `letInFunction` using `let` with value 'Inside function scope with let' and `constInFunction` using `const` with value 'Inside function scope with const'. Print both variables inside the function.

- **Expected Behavior**: Inside the function, both variables should print their respective values: `'Inside function scope with let'` and `'Inside function scope with const'`.

Mandatory Assessment Guidelines:

1. **All actions like build, compile, running application, running test cases will be through Command Terminal.**
2. **To open the command terminal the test takers, need to go to Application menu (Three horizontal lines at left top) -> Terminal ->New Terminal.**
3. **This editor Auto Saves the code.**
4. **If you want to exit(logout) and continue the coding later anytime (using Save & Exit option on Assessment Landing Page) then you need to use CTRL+Shift+B-command compulsorily on code IDE. This will push or save the updated contents in the internal git/repository. Else the code will not be available in the next login.**

5. **These are time bound assessments the timer would stop if you logout and while logging in back using the same credentials the timer would resume from the same time it was stopped from the previous logout.**
6. **This is a web-based application, to run the application on a browser, use the internal browser in the workspace. Click on the second last option on the left panel of IDE, you can find Browser Preview, where you can launch the application.**

   Note: The application will not run in the local browser
7. **You can follow series of command to setup Angular environment once you are in your project-name folder:**
   a. **npm install -> Will install all dependencies -> takes 10 to 15 min.**
   b. **node src/index.js -> To compile and run the index.js file.**
   c. **node src/test/custom-grader.js -> to run all test cases. It is mandatory to run this command before submission of workspace -> takes 5 to 6 min.**
8. **Once you are done with development and ready with submission, you may navigate to the previous tab and submit the workspace. It is mandatory to click on "Submit Assessment" after you are done with code.**
9. **You need to use CTRL+Shift+B - command compulsorily on code IDE, before final submission as well. This will push or save the updated contents in the internal git/repository, and will be used to evaluate the code quality.**