Here are **30 new JavaScript code challenges** focusing on **Advanced JavaScript topics** such as **Closures**, **Modules**, and how to **import/export** in JavaScript:

## Closures (الإغلاق)

### *1. Closure Example with Counter*

- Write a function that returns another function that acts as a counter. Every time the returned function is called, it increments the counter by 1.

### *2. Simple Closure with Private Variable*

- Create a closure that simulates private variables. The outer function should initialize a private variable, and the inner function should allow modifying that private variable.

### *3. Use Closure to Create a Function That Adds a Number*

- Write a function `createAdder()` that takes a number as input and returns another function that adds that number to any given input.

### *4. Callback with Closure*

- Write a function `delayedGreeting()` that takes a message and a delay (in milliseconds), and uses a closure to display the message after the delay.

### *5. Function with Closure Storing Values*

- Create a closure where the outer function keeps track of a list of values, and the inner function allows adding new values to that list.

### *6. Using Closure for Memoization*

- Write a function that calculates the Fibonacci series. Use a closure to memoize the results for faster calculations.

### *7. Create a Closure for User Authentication*

- Write a function `authenticate()` that uses a closure to store the password. It should return another function that checks if the input password matches the stored password.

### *8. Closure for Setting Multiple Properties*

- Write a function `createPerson()` that accepts a name and age, and returns an object with methods to get and set both name and age using closures.

### *9. Closure for Counting Function Calls*

- Write a function `callCounter()` that keeps track of how many times a function has been called, using a closure.

### *10. Closure for Event Handlers*

- Write a function `buttonClickHandler()` that creates a closure to keep track of the number of times a button has been clicked.

## Modules (الوحدات)

### *11. Create and Import a Simple Module*

- Create a module `math.js` that exports a function to add two numbers, and import it into another file to use that function.

### *12. Export and Import Multiple Functions*

- Create a module `utils.js` that exports multiple utility functions, such as `sum()`, `multiply()`, and `divide()`. Import and use them in another file.

### *13. Default Export in Modules*

- Write a module that exports a default function called `greet()`, which accepts a name and returns a greeting message. Import it in another file and call it.

### *14. Module with Object Export*

- Create a module that exports an object representing a book, containing properties like `title`, `author`, and `price`. Import and display these properties in another file.

### *15. Create a Module with a Constant*

- Write a module that exports a constant `PI` and import it in another file to calculate the circumference of a circle with a given radius.

### *16. Dynamic Imports in JavaScript*

- Use dynamic `import()` to load a module only when it's needed. Demonstrate how it works by importing a `helper.js` module inside an event handler.

### *17. Module with Named Exports*

- Create a module `shapes.js` that has named exports for `Circle` and `Square` classes. Import them in another file and create instances of each.

### *18. Create a Simple Calculator Module*

- Write a module `calculator.js` that exports functions like `add()`, `subtract()`, `multiply()`, and `divide()`. Import and use them to perform calculations.

### *19. Use Module with `export` and `import` to Split Code*

- Split a large program into multiple files, where each file contains different functionalities. Use `export` and `import` to tie them together.

### *20. Module to Export a Method Inside an Object*

- Create an object with multiple methods and export it as a module. Import the object and invoke the methods.

## Advanced Concepts in Modules

### *21. Using Module to Encapsulate Variables*

- Create a module that encapsulates a counter variable and exposes only a method to increment the counter. Import this module and use it to manipulate the counter.

### *22. ES6 Modules vs CommonJS*

- Write a CommonJS module and convert it to an ES6 module. Compare the differences and import both into a main JavaScript file.

### *23. Re-exporting Modules*

- Create a module `shapes.js` that imports from `circle.js` and `square.js`, then re-exports the functions from both modules.

### *24. Module with Lazy Loading*

- Implement lazy loading for a module in JavaScript using `import()` and only load the module when needed, for example, on a button click.

### *25. Circular Dependencies in Modules*

- Create two modules where each imports the other. Handle circular dependencies and resolve the issue without causing errors.

### *26. Mixing Named and Default Exports*

- Write a module that combines both named exports and a default export. Import both types of exports in another file.

### *27. Encapsulating Configuration in a Module*

- Write a configuration module that exports settings like `url`, `timeout`, and `retryCount`. Import and use these settings in an application module.

### *28. Async Functions in Modules*

- Create a module that includes an `async` function that fetches data from an API. Import this function into another file and handle the promise.

### *29. Named Imports with Aliases*

- Create a module with multiple exports and import them in another file using aliases. For example, import `add` as `sum` and `subtract` as `difference`.

### *30. Module to Handle User Input*

- Create a module `input.js` that exports a function to capture user input (such as a form field). Import this module into your app and handle the input accordingly.

---

These challenges will help you dive deeper into **Advanced JavaScript** topics such as **Closures**, **Modules**, **Async Programming**, and using **import/export** statements to manage modular JavaScript code effectively.