**Exercise 1: Counter with Closures**

Create a counter function that returns an object containing two methods: increment and getValue. The increment method should increase the value of a counter, and getValue should return the current value of the counter.

**Exercise 2: Customizable Greeting**

Write a function createGreeter that takes a greeting message (like "Hello") and returns a function that takes a name and prints a greeting using the message.

**Exercise 3: Once Function**

Write a function once that takes another function as an argument and returns a new function that can only be called once. Subsequent calls should do nothing.

**Exercise 4: Multiply with Closures**

Create a function multiply that takes a number and returns a function that multiplies any given number by the first number.

**Exercise 5: Private Variables with Closures**

Write a function bankAccount that returns an object with two methods: deposit and withdraw. Use closures to keep the balance private.

**Exercise 6: Task Manager with Closures**

Create a taskManager function that allows you to manage tasks. It should return an object with methods to:

* addTask(task): Add a task.
* removeTask(task): Remove a task by its name.
* getAllTasks(): Return all the tasks.
* clearTasks(): Clear all tasks.

The tasks should be private and only accessible via the provided methods.

**Exercise 7: Timer with Start, Stop, and Reset**

Create a timer function that tracks the elapsed time. The timer should:

* Start counting time when start() is called.
* Stop counting when stop() is called.
* Reset the time to 0 when reset() is called.
* Return the current time with getTime().

Use closures to manage the timer's state (like the elapsed time and interval ID).

**Exercise 8: Secure Bank Account with Deposit, Withdraw, and Statement**

Write a bankAccount function that:

* Allows the user to deposit money.
* Allows the user to withdraw money.
* Keeps track of transaction history.
* Provides a statement of all transactions.

Each transaction should be logged privately, and the statement should return a list of all transactions along with the account balance.

**Exercise 9: Voting System with Multiple Candidates**

Create a votingSystem function that allows users to vote for candidates. It should:

* Allow the addition of new candidates.
* Allow voting for any candidate.
* Show the results with the vote counts of all candidates.

Use closures to maintain the vote count for each candidate.

**Exercise 10: Event Subscription System**

Create an eventManager function that allows event subscription and notification. It should:

* Allow subscription to events with on(eventName, callback).
* Trigger events using emit(eventName, data).
* Keep track of event subscribers privately.

**Exercise 11: To-Do List with Categories**

Create a toDoList function that manages tasks in multiple categories. Each category has its own set of tasks. The system should support the following:

* addCategory(name): Adds a new category.
* removeCategory(name): Removes a category.
* addTask(category, task): Adds a task to a specific category.
* removeTask(category, task): Removes a task from a specific category.
* getTasks(category): Returns all tasks for a specific category.
* clearTasks(category): Clears all tasks in a category.
* getAllCategories(): Returns all category names.
* getAllTasks(): Returns all tasks across all categories.

Each method will work with closures to keep tasks and categories private.

**Exercise 12: Library System with Books and Authors**

Create a library function that manages books and authors. The system should:

* addAuthor(name): Add an author to the system.
* addBook(author, book): Add a book to a specific author.
* removeBook(author, book): Remove a book from an author.
* removeAuthor(name): Remove an author and all their books.
* getAuthorBooks(author): Get all books for a specific author.
* getAllAuthors(): Get a list of all authors.
* getAllBooks(): Get a list of all books across all authors.
* clearLibrary(): Remove all authors and books.

**Exercise 13: Shopping Cart with Discounts**

Create a shoppingCart function that manages a cart and applies discounts. It should:

* addItem(name, price): Add an item to the cart.
* removeItem(name): Remove an item from the cart.
* applyDiscount(code, percentage): Apply a discount code.
* removeDiscount(): Remove any applied discount.
* getTotal(): Return the total price of all items in the cart after discount.
* getItems(): Return all items in the cart.
* clearCart(): Clear the cart.
* getDiscount(): Return the current discount information.

**Exercise 14: Online Store with Inventory Management, Cart, and Promotions**

Create an online store system with the following features:

* **Inventory**:
  + addItem(name, price, stock): Add an item to the inventory.
  + removeItem(name): Remove an item from the inventory.
  + updateStock(name, stock): Update the stock quantity of an item.
  + getInventory(): Return all items in the inventory.
* **Shopping Cart**:
  + addToCart(name, quantity): Add an item to the cart if it exists in inventory and has enough stock.
  + removeFromCart(name): Remove an item from the cart.
  + getCartTotal(): Return the total price of items in the cart.
  + clearCart(): Clear the cart.
* **Promotions**:
  + applyPromo(code, percentage): Apply a promotion code for a percentage discount.
  + removePromo(): Remove the promotion code.
  + getTotalWithPromo(): Get the cart total after applying the promo.
* **Checkout**:
  + checkout(): Complete the purchase by reducing stock for each item in the car

**Exercise 15: Task Manager with User Access Control**

Create a taskManager function that manages tasks for multiple users. It should handle:

* **User Management**:
  + addUser(name): Add a user to the system.
  + removeUser(name): Remove a user and all their tasks.
  + getUsers(): Return a list of all users.
* **Task Management**:
  + addTask(user, task): Add a task for a specific user.
  + removeTask(user, task): Remove a task for a specific user.
  + getUserTasks(user): Get all tasks for a specific user.
  + clearUserTasks(user): Clear all tasks for a specific user.
* **Access Control**:
  + grantAccess(user, task): Grant access to a task for another user.
  + revokeAccess(user, task): Revoke access to a task from another user.
  + getSharedTasks(user): Return all tasks that the user has access to (granted by others).