Closures:

- **Definition**: A closure is a function that remembers the variables around it, even after the outer function has finished running.
- **Why Useful**: Closures let you keep some data private and only accessible within the function.
- **Access**: Closures allow a function to access variables from an outer function after the outer function has finished.
- **Where Stored**: Variables in closures are stored in a special area called the closure scope.
- **Explanation**: Closures can keep variables safe and only let specific functions use them.

Event Bubbling and Capturing:

- **Event Bubbling**: When an event happens on an element, it starts at the target element and then bubbles up to its parent elements.
- **Stop Event Bubbling**: Use `event.stopPropagation()` to stop the event from moving up to the parent elements.
- **Capturing Phase**: The event starts from the top element (root) and goes down to the target element.
- **Both Phases**: If both capturing and bubbling are used, the event first goes through capturing and then bubbling.
- **Innermost Element**: In event bubbling, the innermost element's event is handled first.
- **Stop Propagation**: Use `stopPropagation()` to stop the event from bubbling up.
- **Order in Capturing**: During event capturing, the event goes from the top element to the target element.
- **Add Event Listeners**: Use `addEventListener(type, listener, useCapture)` to add event listeners for both capturing and bubbling.
- **Difference**: Event bubbling moves the event from the innermost element to the outermost element. Event capturing moves the event from the outermost element to the innermost element.
- **Prevent Default**: Use `event.stopPropagation()` to stop event bubbling and `event.preventDefault()` to stop the default action.

THIS Keyword:

- **Global Scope**: In the global scope, `this` refers to the global object (`window` in browsers).
- **Regular Function**: Inside a regular function, `this` refers to the object that called the function or the global object.
- **Arrow Function**: In an arrow function, `this` refers to the surrounding (lexical) scope.
- **Non-Strict Mode**: In non-strict mode, `this` in a regular function refers to the global object.
- **Lexical Scope**: In an arrow function, `this` takes the value from the surrounding context.
- **Constructor Function**: In a constructor function, `this` refers to the newly created instance.
- **Class Method**: In a class method, `this` refers to the instance of the class.

- **Callback Function**: When a method is used as a callback, this may refer to the global object or be undefined in strict mode unless explicitly bound.

Call, Apply, and Bind Functions:

- **Call Method**: Executes a function with a specific this value and arguments provided one by one.
- **Apply Method**: Executes a function with a specific this value and arguments provided as an array.
- **Bind Method**: Returns a new function with a specific this value.
- **Difference**: apply accepts arguments as an array, call takes them separately.
- **New Function**: bind returns a new function bound to a specific object.
- **Add Numbers Example**:
- **Partially Applied Function**: bind can create a function with preset arguments.
- **Borrow Methods**: call and apply can borrow methods from other objects by setting this explicitly.