Closures

What is a closure in JavaScript?

A. A function combined with its lexical environment.

B. A global variable

C. A private object

D. A constructor function

Why are closures useful?

A. To execute asynchronous code

B. To create private variables.

C. To manage multiple DOM events

D. To avoid callbacks

What is the output of the following code?

```
function outer() {
    let count = 0;
    return function inner() {
        count++;
        console.log(count);
    };
}
const counter = outer();
counter();
counter();
```

A. 1, 1

B. 1, 2.

C. 0, 1

D. undefined, undefined

What does a closure in JavaScript allow?

A. Access to variables within a function after the function execution has completed.

B. Execution of a function without defining it

C. Copying variables into a new object

D. None of the above

Which of the following will output 5?

```
function outer() {
    let count = 5;
        return function() {
        console.log(count);
    };
}
const inner = outer();
inner();
```

A. 5.

B. Undefined

C. Error

D. Nothing

In closures, where are the variables stored?

A. Call stack

B. Memory heap

C. Global scope

D. Closure scope.

What is the result of the following code?

```
function makeMultiplier(multiplier) {
    return function(x) {
        return x * multiplier;
    };
}
const double = makeMultiplier(2);
console.log(double(5));
```

A. 2

B. 10.

C. 5

D. Error

Event Bubbling and Capturing

What is event bubbling?

A. Events start from the innermost element and propagate outward.

B. Events start from the outermost element and propagate inward

C. Events are stopped at the first handler

D. None of the above

How can you stop event bubbling in JavaScript?

A. event.stopPropagation().

B. event.preventDefault()

C. event.returnValue = false

D. stopEvent()

What is the order of execution in the "capturing" phase?

A. Outermost to innermost element

B. Innermost to outermost element

C. Random order

D. Synchronous execution

What happens if both capturing and bubbling phases are handled for the same event?

A. Only the capturing phase is triggered

B. Only the bubbling phase is triggered

C. Both phases are executed in order.

D. The event is canceled

In event bubbling, which element's event is captured first?

A. Target element

B. Innermost child.

C. Outermost parent

D. None of the above

Which method can be used to stop event propagation during bubbling?

A. stopBubbling()

B. preventDefault()

C. stopPropagation().

D. None of the above

In event capturing, the order of event handling starts from:

A. Target to root

B. Parent to child

C. Root to target.

D. None of the above

Which JavaScript method adds both capturing and bubbling event listeners?

A. addEventListener(type, listener)

B. addEventListener(type, listener, useCapture).

C. attachEvent(type, listener)

D. None of the above

THIS Keyword

In the global scope, what does this refer to in JavaScript?

A. The window object.

B. Undefined

C. Null

D. The current function

What is the value of this inside a regular function?

A. The object that called the function

B. The global object

C. Undefined in strict mode

D. All of the above.

What happens when you use this inside an arrow function?

A. It takes the value of this from the enclosing lexical scope.

B. It creates its own this context

C. It throws an error

D. None of the above

What will this refer to in the following code snippet?

```
const obj = {
   method: function() {
      console.log(this);
   }
};
obj.method();
```

A. Global object

B. Undefined

C. obj.

D. None of the above

In a simple function, what does this refer to in non-strict mode?

A. Global object.

B. Current object

C. Function itself

D. Undefined

In an arrow function, what does this refer to?

A. Global object

B. The object calling the function

C. Lexical scope.

D. Undefined

What does this refer to in event handlers?

```
document.getElementById('btn').addEventListener('click',function()
{
console.log(this);
      });
```

A. The button element.

B. The window object

C. Undefined

D. Global object

Call, Apply, and Bind Functions

What does the call() method do in JavaScript?

A. Creates a new function

B. Executes a function with a given this value and arguments provided individually.

C. Executes a function with arguments as an array

D. Binds this value for later execution

How is apply() different from call()?

A. It cannot take arguments

B. It takes arguments as an array.

C. It modifies the function's prototype

D. It returns a promise

What does bind() return?

A. A new function.

B. A reference to the original function

C. The this object

D. A boolean value

What is the output of the following code?

```
const obj = { num: 10 };
function add(a, b) {
    return this.num + a + b;
}
const result = add.call(obj, 20, 30);
console.log(result);
```

A. undefined

B. 50

C. 60.

D. 40

What is the difference between call and apply?

A. call accepts arguments as an array, and apply takes them separately

B. call binds a function to an object, while apply doesn't

C. apply accepts arguments as an array, and call takes them separately.

D. There's no difference

What does bind return?

A. A reference to the function

B. A new function bound to a specific object.

C. The value of this in the function

D. Undefined

What will this code output?

```
const person = {
firstName: "John",
lastName: "Doe",
};
function greet(greeting) {
   console.log(greeting + " " + this.firstName + " " this.lastName);
}
greet.call(person, "Hello");
```

A. Hello undefined undefined

B. Hello John Doe.

C. Hello

D. Undefined

What will this code output?

```
const obj = {
num1: 10,
num2: 20,
};
function addNumbers(a, b) {
   return this.num1 + this.num2 + a + b;
}
const result = addNumbers.apply(obj, [30, 40]);
console.log(result);
```

A. 100.

B. 60

C. 10

D. Undefined