Gmail Bikash Adhikari <virtual.bikash@gmail.com>

# closure in JavaScript

1 message

**Bikash** <bikash.caim@gmail.com> Sat, Mar 8, 2025 at 4:30 PM
To: virtual.bikash@gmail.com

In JavaScript, a closure is a fundamental concept that allows a function to access variables from its outer (enclosing) function's scope, even after the outer function has finished executing. Here's a breakdown:

### Understanding Closures

- **Lexical Scoping:**
  - Closures are closely tied to lexical scoping. This means that a function's scope is determined by where it's defined in the code, not where it's called.
- **The Closure Mechanism:**
  - When a function is created inside another function, the inner function forms a closure. This closure allows the inner function to "remember" the variables from the outer function's scope.
  - Even after the outer function has completed its execution, the inner function retains access to those variables.

### Why Closures Are Important

- **Data Encapsulation:**
  - Closures enable you to create private variables and functions, hiding them from the global scope. This helps in organizing your code and preventing unintended modifications.
- **State Preservation:**
  - Closures allow functions to maintain state between invocations. This is useful for creating functions that "remember" previous values.
- **Asynchronous JavaScript:**
  - Closures play a vital role in asynchronous programming, such as handling callbacks and event listeners. They ensure that functions have access to the necessary variables when they are executed later.
- **Function Factories:**
  - Closures allows for the creation of function factories. Functions that create other specialized functions.

### Example

JavaScript

```
function outerFunction(outerVariable) {
  return function innerFunction(innerVariable) {
```

```
    console.log('Outer Variable: ' + outerVariable);
    console.log('Inner Variable: ' + innerVariable);
  };
}

const myFunction = outerFunction('Hello');
myFunction('World'); // Output: Outer Variable: Hello, Inner Variable: World
```

In this example:

- `innerFunction` forms a closure, allowing it to access `outerVariable` even after `outerFunction` has finished executing.

**Key Considerations**

- **Memory Usage:**
  - Closures can potentially lead to increased memory usage if they hold references to large objects or variables that are no longer needed.
- **Potential for memory leaks:**
  - Care should be taken to avoid unneeded references that could keep objects in memory.

I hope this explanation helps!

**Sources**
1. https://docs.pingcode.com/ask/ask-ask/165972.html
2. https://github.com/AkhmalN/javascript-dasar