Closures

A closure is a function that has access to its own scope, the scope of the outer function, and the global scope. This means a closure can remember and access variables from its outer function even after that function has finished executing.

```
function ahello() {
  var a = "variable inside a outer function";

  function ahi() {
    var b = "varaible inside inner function";
    console.log(a);
    console.log(b);

  }
  ahi();
}
ahello();
```

Scope Chaining in JavaScript

Scope in JavaScript refers to the context in which variables, functions, and objects are accessible. JavaScript has three types of scope:

1. **Global Scope:** Variables declared outside of any function or block are in the global scope. They are accessible from anywhere in the code.
2. **Local Scope:** Variables declared within a function or block are in the local scope. They are only accessible within that function or block.

3. **Block Scope:** Variables declared inside a block can't able to access outside of the function

Scope Chain:

- When a variable is accessed, JavaScript looks for it in the current scope.
- If the variable is not found, it looks in the outer scope.
- This process continues until it reaches the global scope.
- If the variable is not found in any scope, it results in a ReferenceError

```
 var globalVar = "I am global";

function outerFunction() {
   var outerVar = "I am outer";

   function innerFunction() {
      var innerVar = "I am inner";
      console.log(innerVar);     // Output: I am inner
      console.log(outerVar);     // Output: I am outer
      console.log(globalVar);    // Output: I am global
   }

   innerFunction();
   console.log(outerVar);        // Output: I am outer
   console.log(globalVar);       // Output: I am global
   // console.log(innerVar);     // Error: innerVar is not defined
}

outerFunction();
console.log(globalVar);          // Output: I am global
// console.log(outerVar);        // Error: outerVar is not defined
// console.log(innerVar);        // Error: innerVar is not defined
```

Lexical Scoping:

- JavaScript uses lexical scoping, meaning that the scope of a variable is determined by its position in the source code.
- Inner functions have access to variables declared in their outer functions (but not vice versa).

Immediately Invoked Function Expression

An IIFE (Immediately Invoked Function Expression) is a JavaScript function that runs as soon as it is defined.

Following shows the syntax

```
(function(){
   //code goes here
})()
```

```
(function(){
   console.log("self invoking function invoked by itself") ;
})();
```