Closures

a closure gives you access to an outer function's scope from an inner function. In JavaScript, closures are created every time a function is created, at function creation time.(**A closure is a function enclosed with references to the variables in its outer scope**)

```
function makeFunc() {
  const name = "Mozilla";
  function displayName() {
    console.log(name);
  }
  return displayName;
}

const myFunc = makeFunc();
myFunc();// output: Mozilla

//we can also call it like this
makeFunc()();// we can pass parameter in here like makeFunc()(5)
```

Closure scope chain

Every closure has three scopes:

- Local scope (Own scope)
- Enclosing scope (can be block, function, or module scope)
- Global scope
-

```
// global scope
const e = 10;
function sum(a) {
  return function (b) {
    return function (c) {
      // outer functions scope
      return function (d) {
        // local scope
        return a + b + c + d + e;
      };
    };
  };
}

console.log(sum(1)(2)(3)(4)); // 20
```

You can also write without anonymous functions:

```
// global scope
const e = 10;
function sum(a) {
  return function sum2(b) {
    return function sum3(c) {
      // outer functions scope
      return function sum4(d) {
        // local scope
        return a + b + c + d + e;
      };
    };
  };
}

const sum2 = sum(1);
const sum3 = sum2(2);
const sum4 = sum3(3);
const result = sum4(4);
console.log(result); // 20
```

Lexical Scope

A lexical scope in javascript means that a variable defined outside a function can be accessible inside of another function defined after a variable declaration but the opposite is not true

```
// Lexical Scope

var username = "RoadsideCoder";

//global scope

function local(){
console.log(username);
}

local(); // output: RoadsideCoder

/................................./
```

```
function local(){
  var username = "RoadsideCoder";
}
console.log(username);
local(); // output: Uncaught ReferenceError: username is not defined
```

Question 01: What will be logged to console?

```
let count = 0;
(function printCount() {
    if (count === 0) {
        let count = 1; //shadowing
        console.log(count);// output: 1
    }
    console.log(count);//output: 0
})()
```

Question 02: write a function that would allow you to do this

```
function createBase(num) {
    return function (innerNum) {
        console.log(innerNum + num);
    };
}
var addSix = createBase(6);
addSix(10); // returns 16
addSix(21); //returns 27
```

Question 03: Time Optimization

Before using Closure:

```
function find(index) {
    let a = [];
    for (let i = 0; i < 1000000; i++) {
        a[i] = i * i;
    }
    console.log(a[index])
}

console.time('6');
find(6);
console.timeEnd('6');//6: 21ms - timer ended
console.time('12');
```

```
find(12);

console.timeEnd('12');//12: 11ms - timer ended
```

After using closure:

```
function find() {
    let a = [];
    for (let i = 0; i < 1000000; i++) {
        a[i] = i * i;
    }

    return function(index){
        console.log(a[index])
    }

}

const closure = find()
console.time('6');
closure(6);
console.timeEnd('6');//6: 2ms - timer ended
console.time('12');
closure(12);

console.timeEnd('12');//12: 0ms - timer ended
```

Block scope and setTimeout

```
for (var i = 0; i < 3; i++) {
    function inner(i) {
        setTimeout(function log() {
            console.log(i);
        }, i * 1000);

    }
    inner(i);
}//output: 0,1,2
//using closure var will not only print 3 three times
```

question 05: How would you use a closure to create a private counter?

```
function counter(){
  var _counter = 0;
function add(increment){
 _counter += increment;
}
function retrive(){
 return "Counter = " + _counter;
}

return { add, retrive}
}
const c = counter()
c.add(5);
c.add(10);
```

Question 06: What is Module Pattern

```
var Module = (function () {

    function privateMethod() {
        console.log('Private');
    }
    return {
        publicMethod: function () {
            console.log('public')
        }
    }
})()
Module.publicMethod();//output: public
Module.privateMethod();//output: Uncaught TypeError: Module.privateMethod is not a function
```

Question 07: Make this run only once

```
let view;
function likeTheVideo() {
    let called = 0;
    return function () {
        if (called > 0) {
            console.log('Already called')
        }
        else {
            view = "Lazy Coder";
            console.log('Wake up');
            called++;
        }
    }
}

let checkalive = likeTheVideo();//create local scope and it will reference to
//the same called variable(called)
checkalive();
checkalive();
checkalive();
checkalive();
```