```javascript
//*****************CLOSURES LEXICAL SCOPE******** */

//Lexical Scope

// Global scope in here
var foo = function () {
  // local scope in here
};

function subscribe() {
  var name = "Roadside Coder"; // name is a local variable created by foo
  function displayName() {
    //<----- A Closure // displayName() is the inner function
    alert(name); // variable used which is declared in the parent function
  }
  displayName();
}
subscribe();
//MDN Definitions closures
//Closures
function makeFunc() {
  var name = "Roadside Coder";
  function displayName(num) {
    console.log(name, num);
  }
  return displayName;
}

var myFunc = makeFunc();
myFunc();
makeFunc()(5); //Also do this

//Closures have three scopes :
// Local Scope (Own scope)
// Outer Functions Scope
// Global Scope
//Examples

//O/P based
let count = 0;
(function immediate() {
  if (count === 0) {
    let count = 1;
    console.log(count); // Output ?
  }
  console.log(count); // Output?
})();
```

```javascript
//Closure question
function createBase(baseNumber) {
  return function (N) {
    // we are referencing baseNumber here even though it was declared
    // outside of this function. Closures allow us to do this in JavaScript
    return baseNumber + N;
  };
}

var addSix = createBase(6);
addSix(10);
addSix(21);

//Reduce time Optimise our code 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); // this takes 0.25 ms
console.timeEnd("6");
console.time("12");
closure(12); // this takes 0.025ms
console.timeEnd("12");

//Block Scope and Function scope
for (var i = 0; i < 3; i++) {
  setTimeout(function log() {
    console.log(i); // What is logged?
  }, 1000);
}
for (var i = 0; i < 3; i++) {
  function inner(i) {
    setTimeout(function log() {
      console.log(i); // What is logged?
    }, 1000);
  }
  inner(i);
}
//Closure implement a counter
```

```javascript
function counter() {
  var _counter = 0;
  // return an object with several functions that allow you
  // to modify the private _counter variable
  return {
    add: function (increment) {
      _counter += increment;
    },
    retrieve: function () {
      return "The counter is currently at: " + _counter;
    },
  };
}
// error if we try to access the private variable like below// _counter;//
usage of our counter function
var c = counter();
c.add(5);
c.add(9);
// now we can access the private variable in the following way
c.retrieve(); // => The counter is currently at: 14
//MODULE PATTERN
var Module = (function () {
  function privateMethod() {
    // do something
  }

  return {
    publicMethod: function () {
      // can call privateMethod();
    },
  };
})();
Module.publicMethod(); // works
Module.privateMethod(); // Uncaught ReferenceError: privateMethod is not
defined
var Module = (function () {
  function _privateMethod() {
    // do something
  }
  function publicMethod() {
    // do something
  }
  return {
    publicMethod: publicMethod,
  };
})();
let view;
function likeTheVideo() {
  let called = 0;

  return function () {
    if (called > 0) {
      console.log("Already Subscribed");
    } else {
      view = "Roadside Coder";
      console.log("Subscribe to", view);
      called++;
    }
  };
}

let isSubscribe = likeTheVideo();
isSubscribe(); // Subscribe to Roadside Coder
isSubscribe(); // Already Subscribed

//Once function
function once(func, context) {
  let ran;
  return function () {
    if (func) {
      ran = func.apply(context || this, arguments);
      func = null;
    }
    return ran;
  };
}
// Usage
const hello = once(() => console.log("hello"));
hello();
hello();
//Polyfill Memoize
function myMemoize(func) {
  const res = {};
  return function (...args) {
    const argsCache = JSON.stringify(args);
    if (!res[argsCache]) {
      res[argsCache] = func(...args);
      //func.call(context||this,...args)
    }
    return res[argsIndex];
  };
}

const clumsysquare = (num) => {
  for (let i = 1; i <= 100000000; i++) {}
  return num * 2;
```

```
};
console.time("First call");
console.log(clumsysquare(9467));
console.timeEnd("First call");
// use the same value two times
console.time("Second call");
console.log(clumsysquare(9467));
console.timeEnd("Second call");

//Difference between scope and closure
```