Lets discuss it with small example.

// Lets discuss it with small example.

function x() {

  var a = 7;

  function y() {

    console.log(x); //u already know lexical scope- if x is not present in its local space, then it sees in lexical parent

  }

Y()

}

x(); //run it and see it prints 7

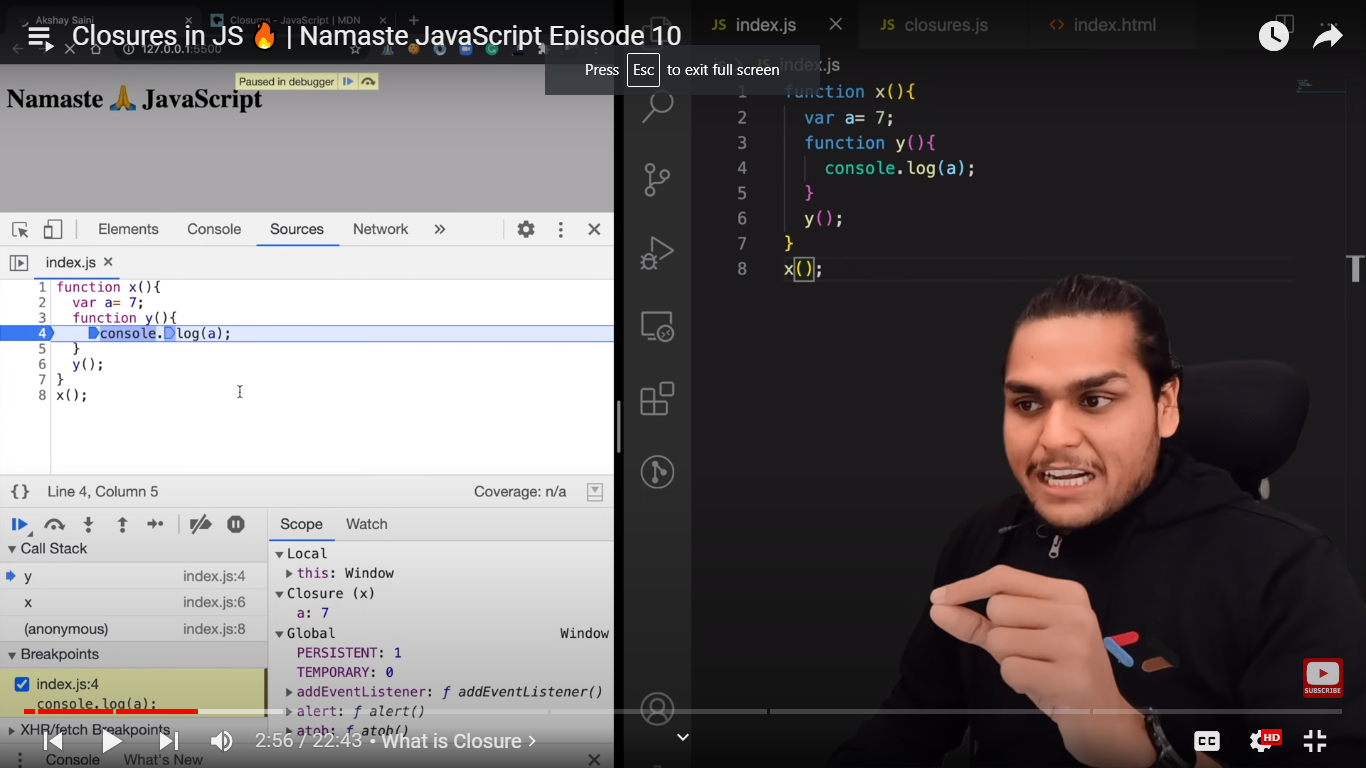
output: 7

//this is wt closure is .

//keep debugger inside the inner function - console.log(x)- u find a separate section in memory as closure(x) : in that variable a:7

//it means - clousure basically means  a function binding together with its lexical environment.

//function along with a lexical scope- together   bundled is known as clousure



Here function y, inside y function it forms a closure with a variable which was part of x lexical scope.

The function y is bind to lexical scope variables of x. it means it forms a clousure , it has access to parents lexical scope,.

A **closure** is the combination of a function bundled together (enclosed) with references to its surrounding state (the **lexical environment**). In other words, 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.

In javascript, it is even valid to assign whole function to a variable.

Eg: var a= function y(){ console.log()}

U can also pass whole function to another function as a parameter

X(function y(){…..})

But many other pgmg languages wont allow it.

Similarly u can return these functions from functions.,so instead of calling y() , we can just return it

function x() {

  var a = 7;

  function y() {

    console.log(x); //u already know lexical scope- if x is not present in its local space, then it sees in lexical parent

  }

  return y;

}

x(); //run it and see it prints 7

here y is nothing but returning whole function.

Lets see wt it returns actually.

//to see wt it is returning lets print it by storing in a variable

function x() {

  var a = 7;

  function y() {

    console.log(x);

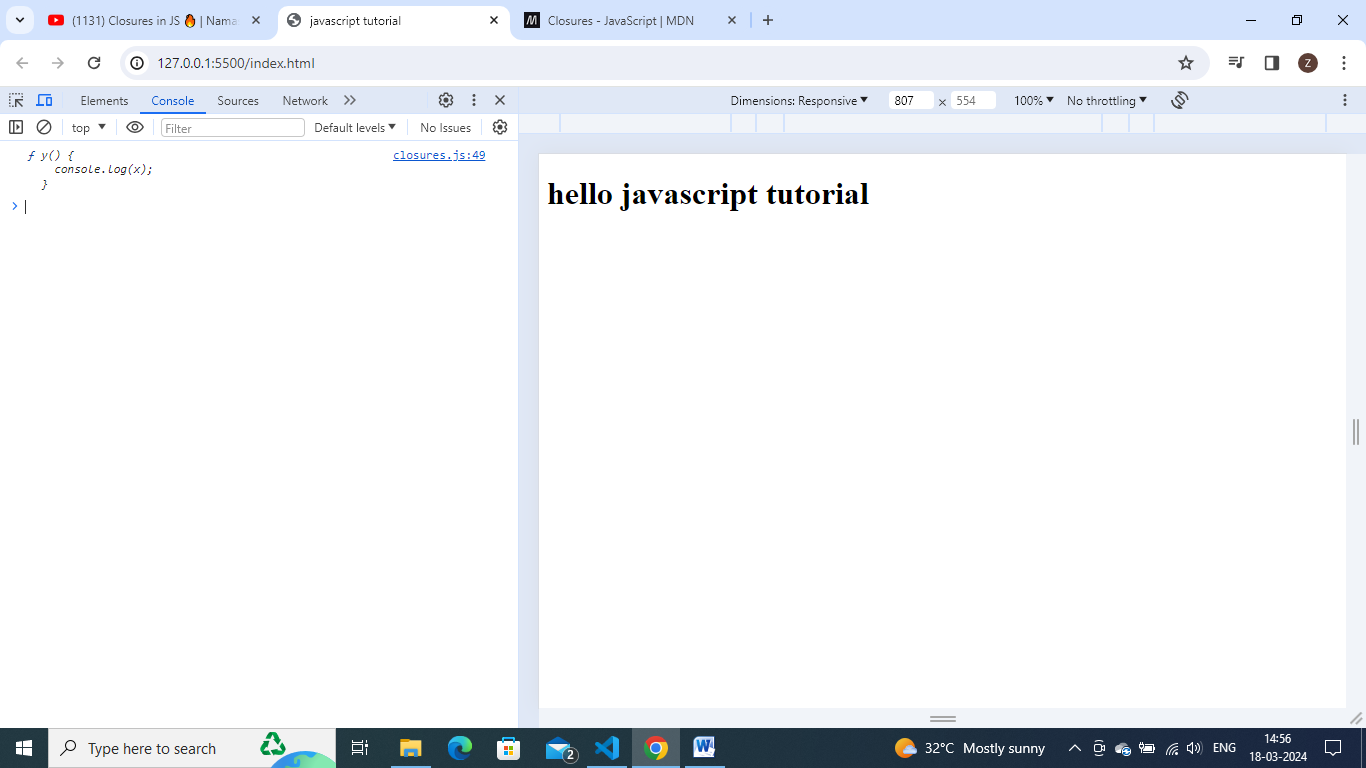
  }

  return y; //returning a func instead of calling it

}

var z = x();

console.log(z); //it gives us whole function y in console as it is



Remember when we called x() function here , wt happenes, it creates a whole execution context , it sets up mme space for variables and fucntions, etc.,

In that line var z=x(), this function got returned to z and now x is not there now in callstack, it is blown off from execution context.. and its variables gone.now z contains this function y. here now we returned y function outside . so it no longer resides in x. then how will it behave outside the scope.now we can use function y outside of x by using reference z.

Let saywe have thousands of code after it and lets call z(), see wt happens

function x() {

  var a = 7;

  function y() {

    console.log(a);

  }

  return y; //returning a func instead of calling it

}

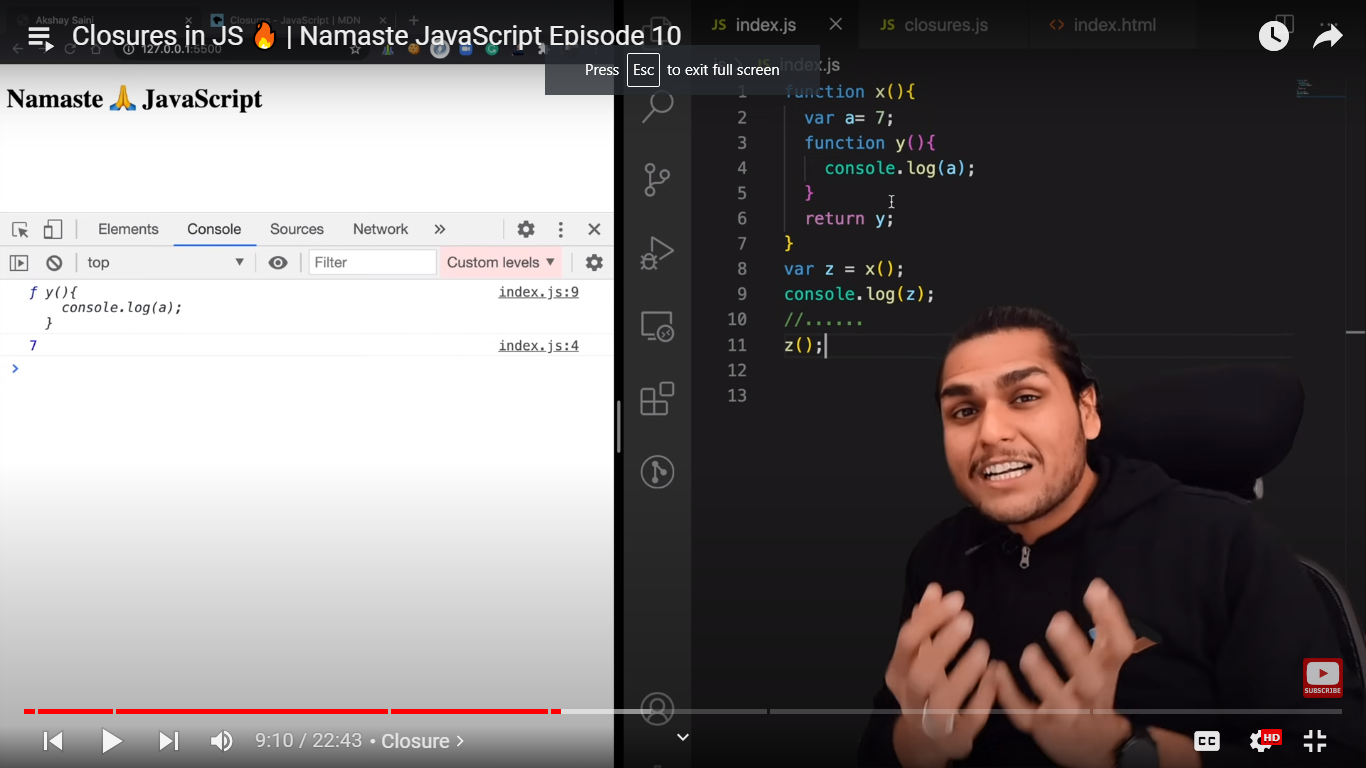
var z = x();

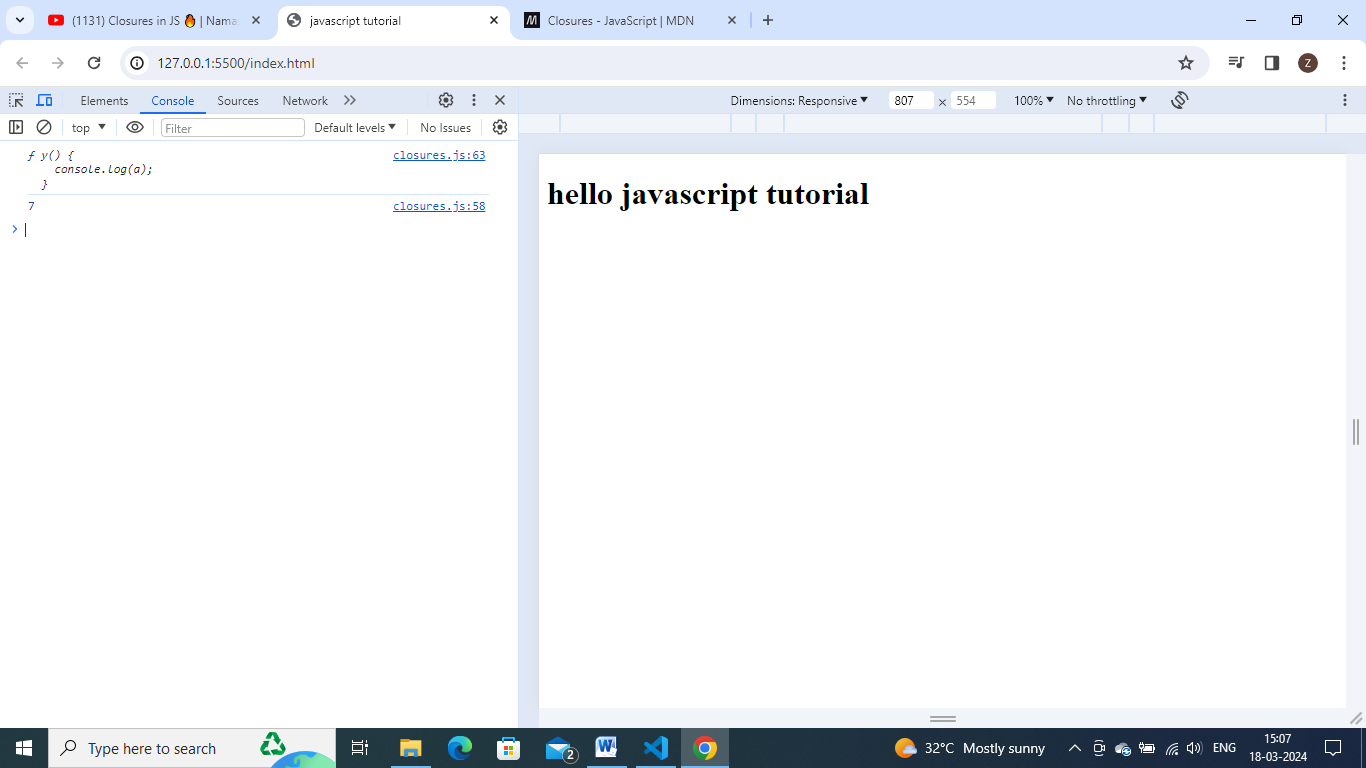
console.log(z); //it gives us whole function y in console as it is

/...//thousands of line of code

z()

it tries to findout ‘a’ to print it. But where is ‘a’. ‘a’ is no longer in global scope now.and x function is also not available now. Wt happens. Wt it prints? Undef,null or other value – it prints 7





Even if it is return from another function it still maintains its lexical scope.

Though x no longer exists – but still y function will remember where it came from, that my parent had some variable a and had some value in it. Binding strong.

So in simple terms, we can say as when we are returning function y then not only that function, but whole closure will gets returned.(closed function along with its lexical scope). It will kept into z. so after some lines of code, if u call z.still it remembers the ref to a.

So in interview – u can say clousure as a function bundled together with its lexical scope.and can explain this eg.

//developers can simply that return stmt like this. - by keeping return keyword before def of y function

function x() {

  var a = 7;

  return function y() {

    console.log(a);

  }

   //returning a func instead of calling it

}

var z = x();

console.log(z); //it gives us whole function y in console as it is

//...//thousands of line of code

z();

**corner questions becoz of clousure:**

//so before returning y ,wt if i change valueof a. which value will it remembers , old a valueor  new a value.

function x() {

  var a = 7;

  function y() {

    console.log(a);

  }

  a = 100;

  return y; //returning a func instead of calling it

}

var z = x();

console.log(z); //it willprint 100

//...//thousands of line of code

z();

here a is not referring to value. When y was in mem=- a value was 7. So a function along with the ref to those variables will be returned.not just the value. The value 7 doesn’t persist, the reference a will persist.now that refrence poin t to 100.

**Cornor point 2:**

//cornor point 2

//lets say i have function z and lets have this whole code inside that fucntion z. and lets say accessing variables of one more parent- grand parent.

function function z() {

  var b = 900;

  function x() {

    var a = 7;

    function y() {

      console.log(a, b); //let say im trying to access here,parent scope and its parents parents scope also lets see wt happens

    }

y()

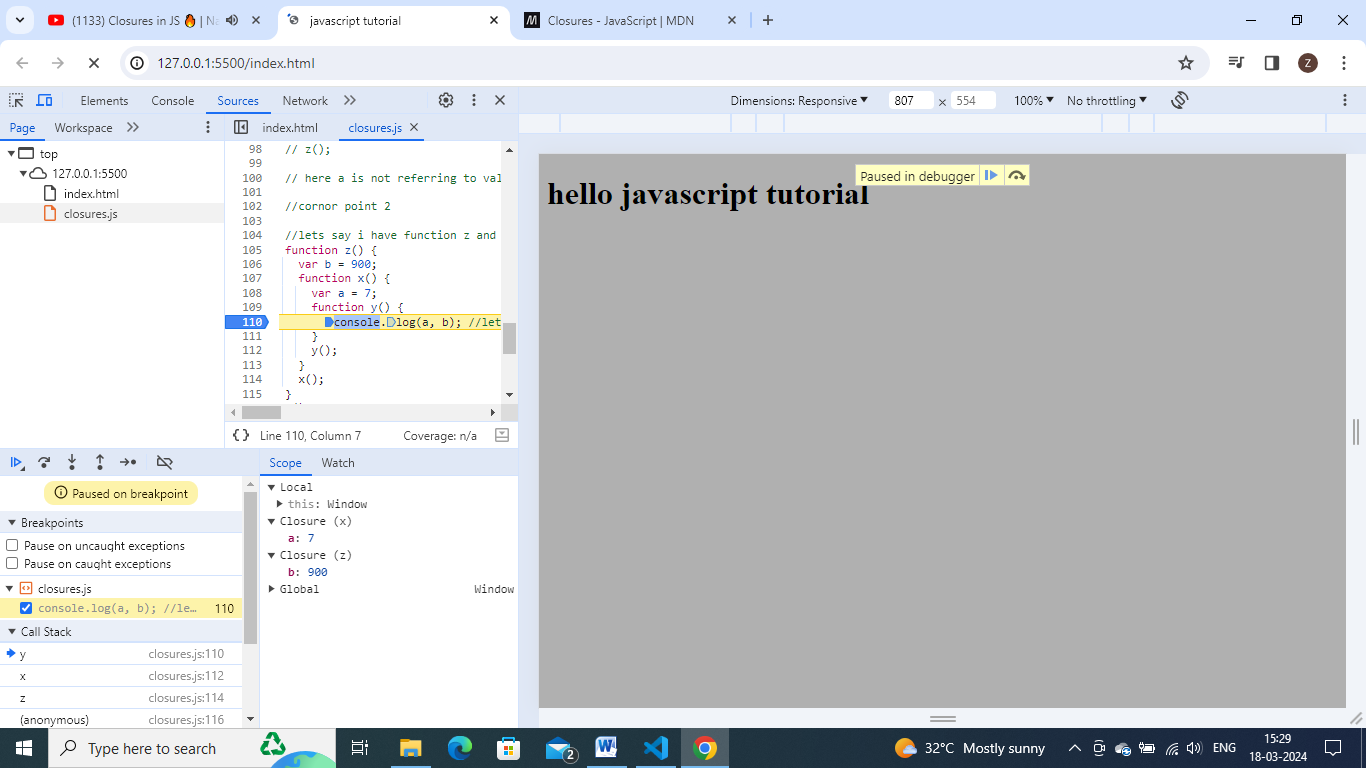
  }

  x()

}

Z();

Keep debugger at line console.log(a,b)-



So y is forming closure along with scope of x and scope of y.

So here it is forming closure with parent variables and also grand parent variables.

So if I return function y to outside, then a and b values persist not gets garbage collected.

**Uses of clousures:**

* Module design pattern
* Currying
* Functions like once
* Memorize
* Maintaining state in async world
* Settimeouts
* Iterators
* And etc.

Currying in js is possible just bcoz of clousure.

U can make a function to run only once by using once function