// what is a callback function in js

// the fucntion which we pass to another function , that function is known as callback function.

//this call back functions are very powerful in js world.it gives us access to the whole asynchonous world in a synchornous single threaded language.

//js is a single threaded lang- it will do everything in a certain order, but due to callback functions  u can do async things in js

//

// eg:

// function x(y) {}

// x(function y() {});

//here function y is passing as a parameter to another function.

//here, function y is known as callback function.

//so passed y as an argument to x.

//so we are giving access to fucntion y to x

//now its x responsibility to decide when to call function y .

//now lets see how it can be used in asynchronous tasks.

//for that lets take eg of settimeout

//as we know settimeout func will take a callback func and lets prints something in it to visualize

setTimeout(function () {

  console.log("timer");

}, 5000);

function x(y) {

  console.log("x");

  y();

}

x(function y() {

  console.log("y");

});

//now see how js execute it.

//as js is single threaded lang,it means code execute one after another and in order sequence

//so settimeout will take this callback func and store it in separate space.and it will attach a timer to it and store it.

//js wont wait for settimeout to finish . so callback func give us the power of asynchronity

//it doesnot wait for 5000 ms to expire and wt ever needs to be done after 5000 ms ,we are passing that piece of code as a callback function

//it will be later exeucted

//js waits for none . so it moves to next line see funciton def of x     and it will try to call this x functin,it wil pass y as a callback function and do console.log(x);

//till this time this 5000 ms is not expired.

//so it will print x ,then print y then after 5000 ms go to settimeout func

//so output will be

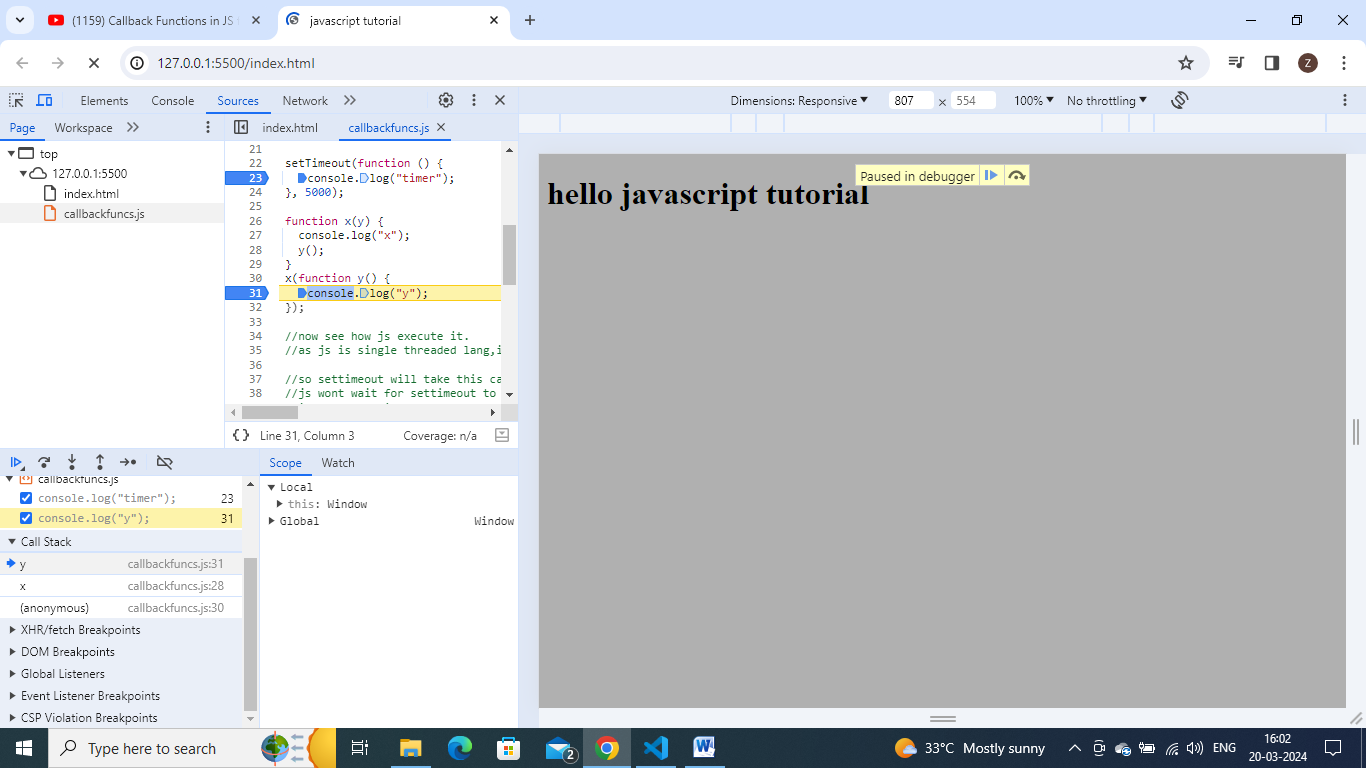
//x y then after 5000 ms gap we get timer

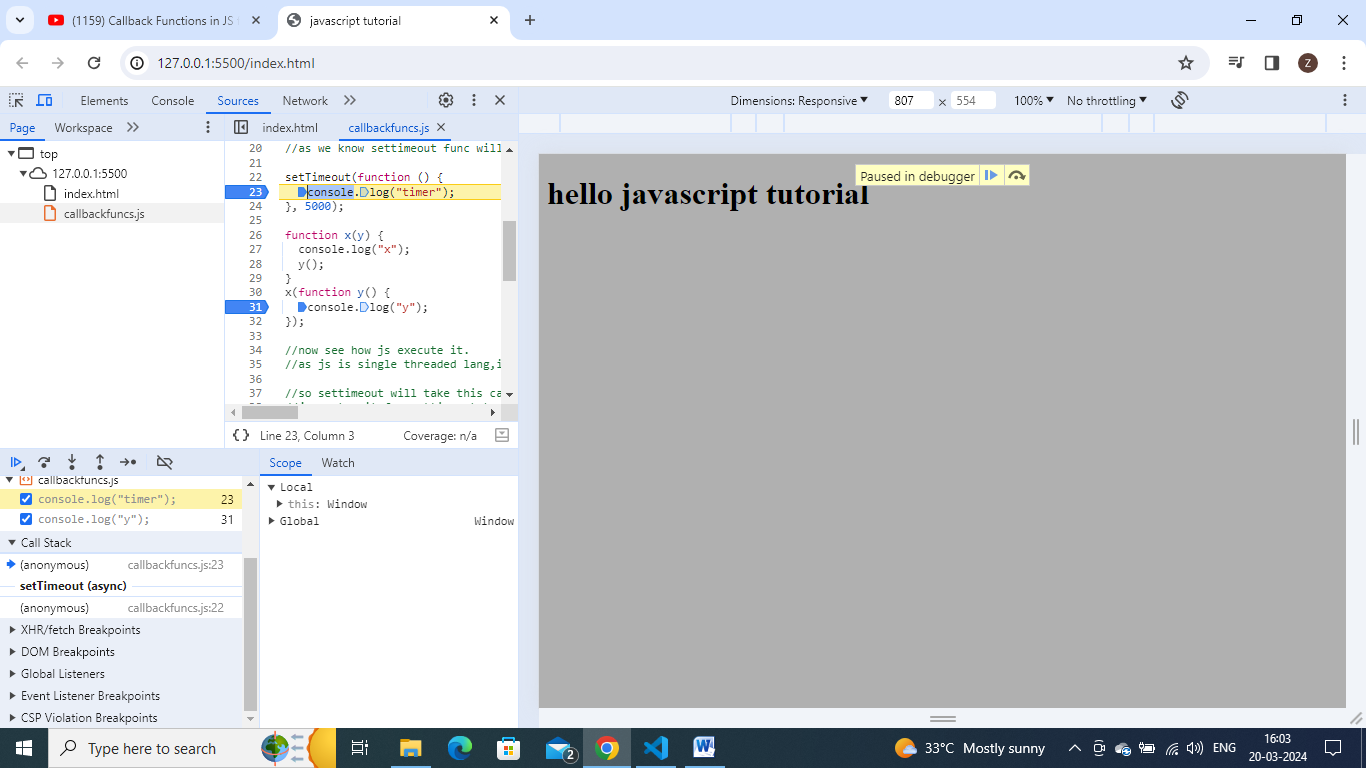
//so settimeout asynchronous operations are not possible , without these callbacks

//we have given this callback func to settimeout to execute after some time

//keep debugger at console.log("y");  and keep another debugger at console.log("timer")  and observe callstack

//so observe callstack -- it will have x,y then  callstack becomes empty, then after 5000  ms , then anonymous fun will get into callstack





Before getting anonymous func into callstack,first becomes empty then aftr 5000 ms , get anonymous fun into callstack.anonymous got into callstack automatically after 5000 ms.

So js has only one callstack, and so u can even call it as the main thread.

Everything which gets executed inside the page, is executed through the callstack only.

So any function, settimeout or function x or function y or callback func inside settimeout , everything will be executed thorugh call stack, so if any operation blocks the callstack, that is known as blocking the main thread .

suppose if ur function x has a very heavy operation that takes 20 – 30 seconds to complete that function, so by that time , bcoz js has one callstack, one main thread,so it wont be any other function in the code, that means everything will be blocked on the code.thats y we say that , we should never block our main thread(Call stack)), so we should always try to use async operations for things which takes time,just like we did here a timer function inside settimeout.

Settimeout will basically takes a callback and will executes this piece of code sometime later and gets out of the callstack.

So if js , doesn’t have this first class fucntions, and we did not have this callback functions and we could not have able to pass like this a function to another function. We could not able to do asynchronous operations.

So, using this web api’s, the settimeout and the callback functions, we can achieve asynchronous opeartions.

Where does this settimeout takes ur function and where it keeps it and how it attaches the timer and get back to callstack- this all we cover in next video- event loops

Lets see now event listeners with dev tools.

Lets create a button on index.html.

    <button id="clickMe">Click Me</button>

Lets attach a click handler to it.

document.getElementById("clickMe").addEventListener("click",function(){

})

Here the function is a callback function.

Here, js will pickup this button and add a event listener to it.when event occurs it will call this callback function.

This is again a callback function which is stored somewhere and that will automatically comes into callstack.

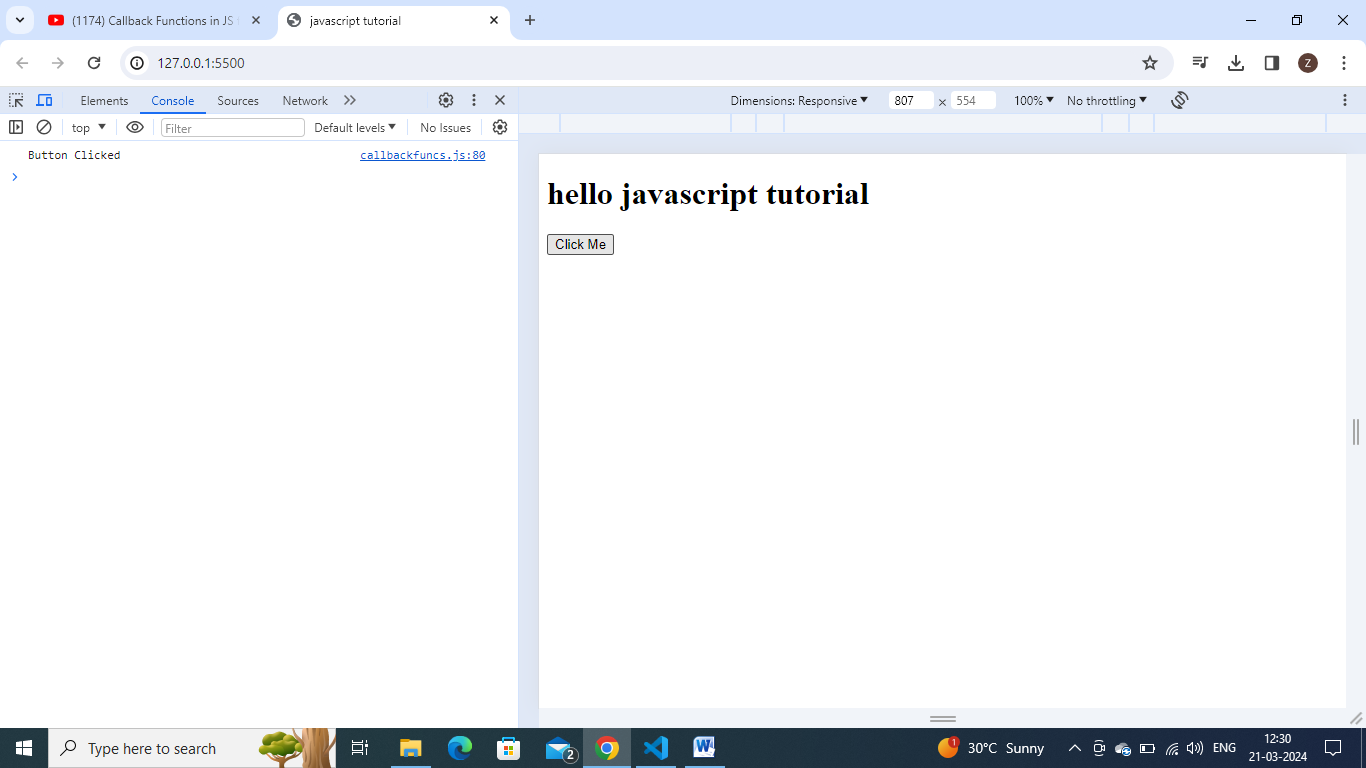
document.getElementById("clickMe").addEventListener("click", function xyz() {

  console.log("Button Clicked");

});

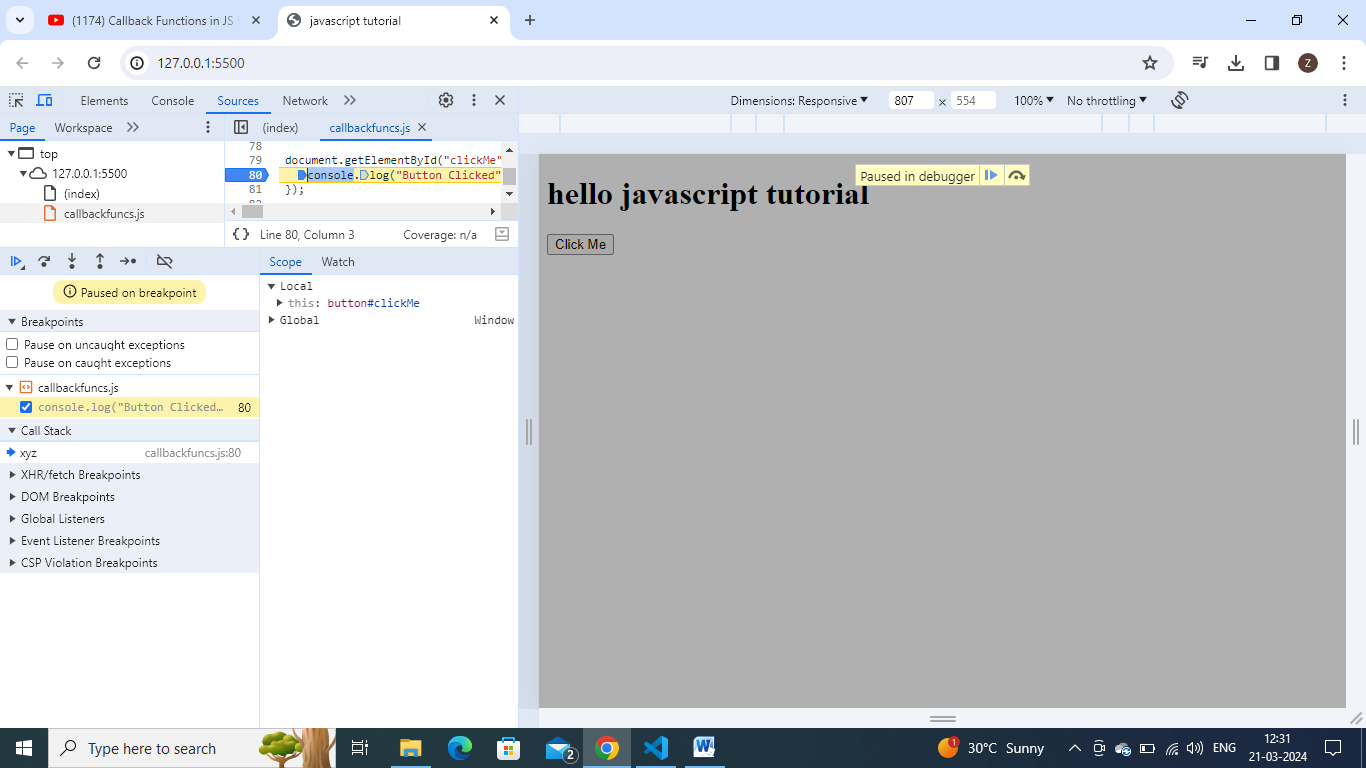
So now we attached this event handler to this button,

Now see by clicking button , u get in console- button clicked.



Now keep debugger inside the callback func,at console.log stmt

After keeping debugger, click button, it will automatically comes into callstack.



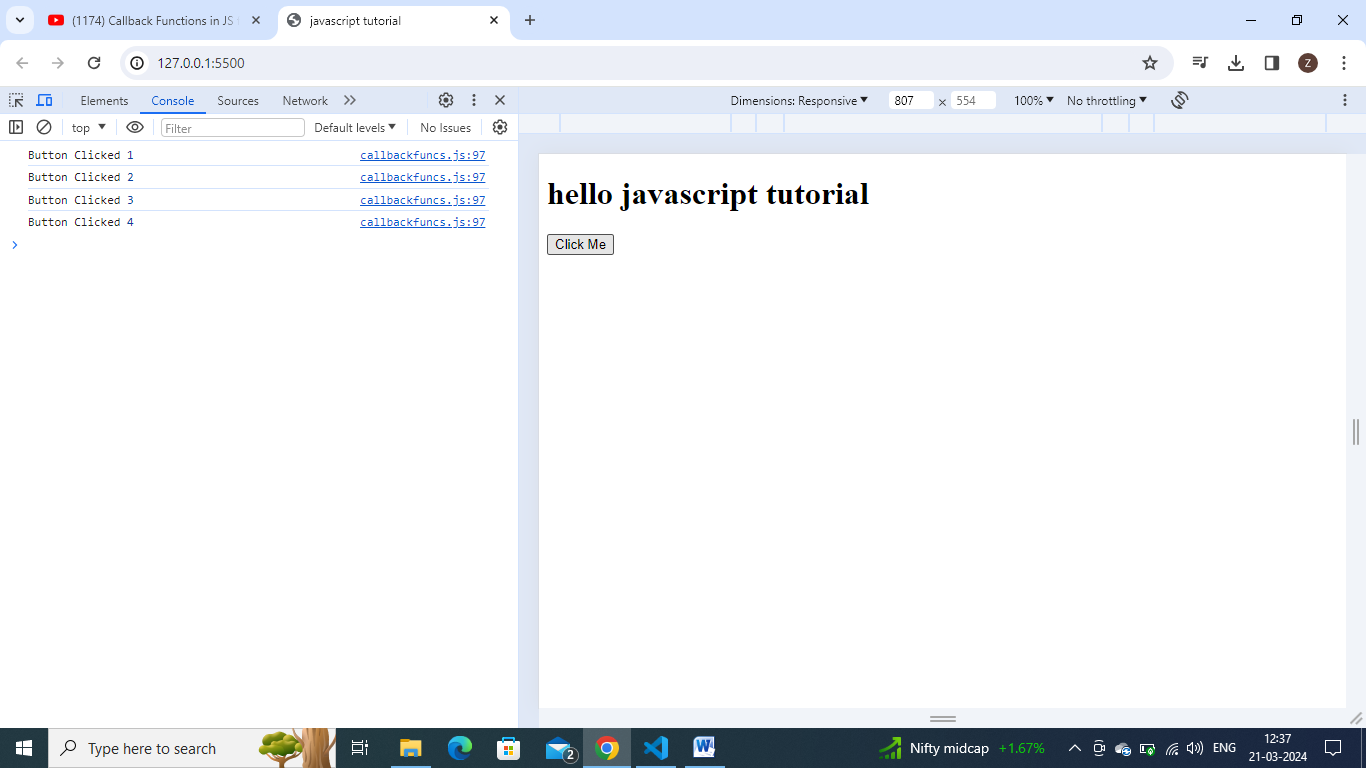
So whenever, we click the button, the function xyz will be pushed into call stack and gets executed.

This is how it actually works.

Now lets show closures demo , along with event listeners.

Suppose if I have to count, how many times, the button got clicked and print that count value.

One way is to use a global variable count and increment it.



let count = 0;

document.getElementById("clickMe").addEventListener("click", function xyz() {

  console.log("Button Clicked", ++count);

});

// when u run this code- u get as buton cliked 1,2..

But using a global variable is not a good solution,so how can we make a closure and secure count, so that this count is not modified by anyother thing inside program.

So to create a closure wrap all this inside a function.

function attachEventListeners() {

  let count = 0;

  document.getElementById("clickMe").addEventListener("click", function xyz() {

    console.log("Button Clicked", ++count);

  });

}

// it forms a closure and attaches a event listener to this click me button

//now this call backfuncion is forming a closure with its outer scope.

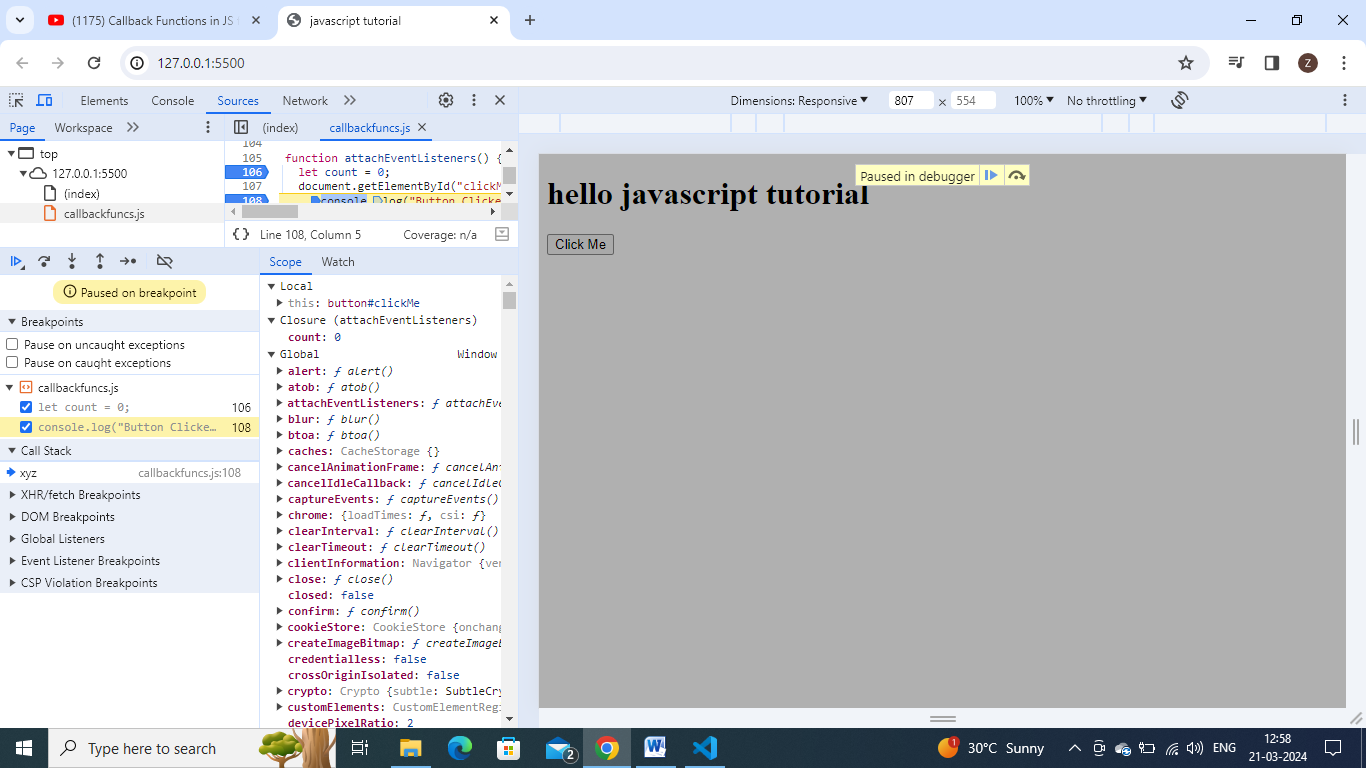
attachEventListeners(); //calling that function

//when js run this code, this callbackfunction forms a closure with count .

// it remembers where the count is present , let execute and see

//keep debugger at line console.log() and carefully observe callstack.

//we found that xyz function will be in callstack

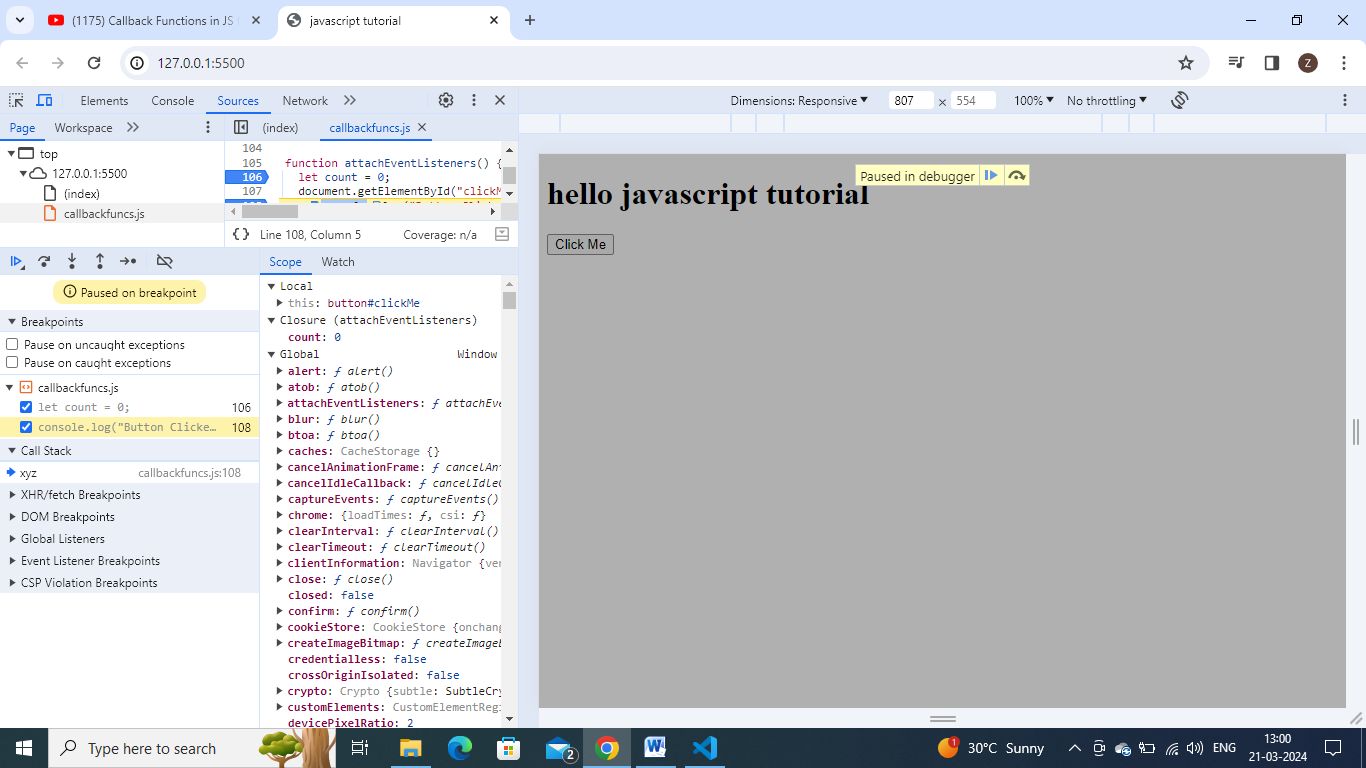


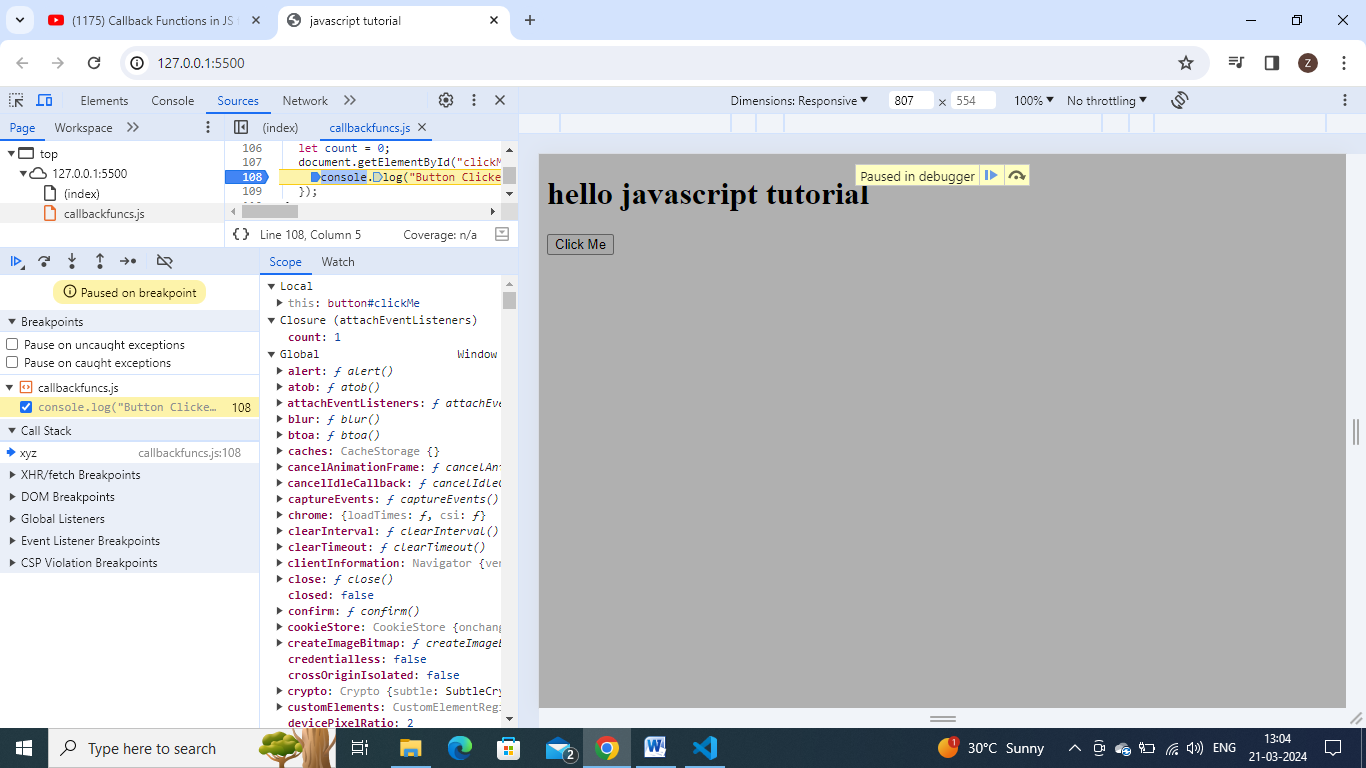
//keep debugger at line console.log() and carefully observe callstack.

//we found that xyz function will be in callstack.

//keep debugger and click the button and see callstack - will have xyz() function in it,

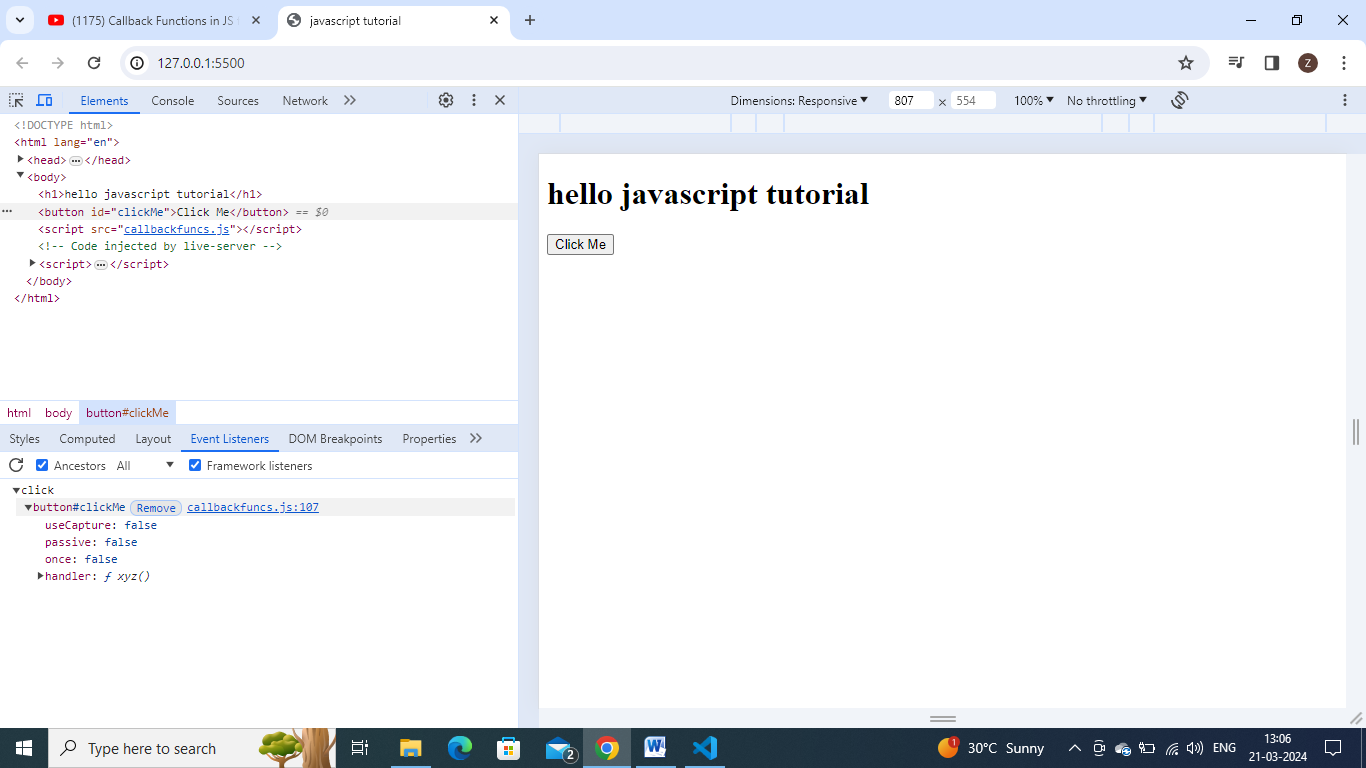
Observe in scope- there is closure mem location, there it is showing count





And whenever, u click this button, it remembers the count and increments and remembers that where count is stored.

Lets see one more in dev tools, if u go to elements, refershthe page and u see a tab inside it, event listeners



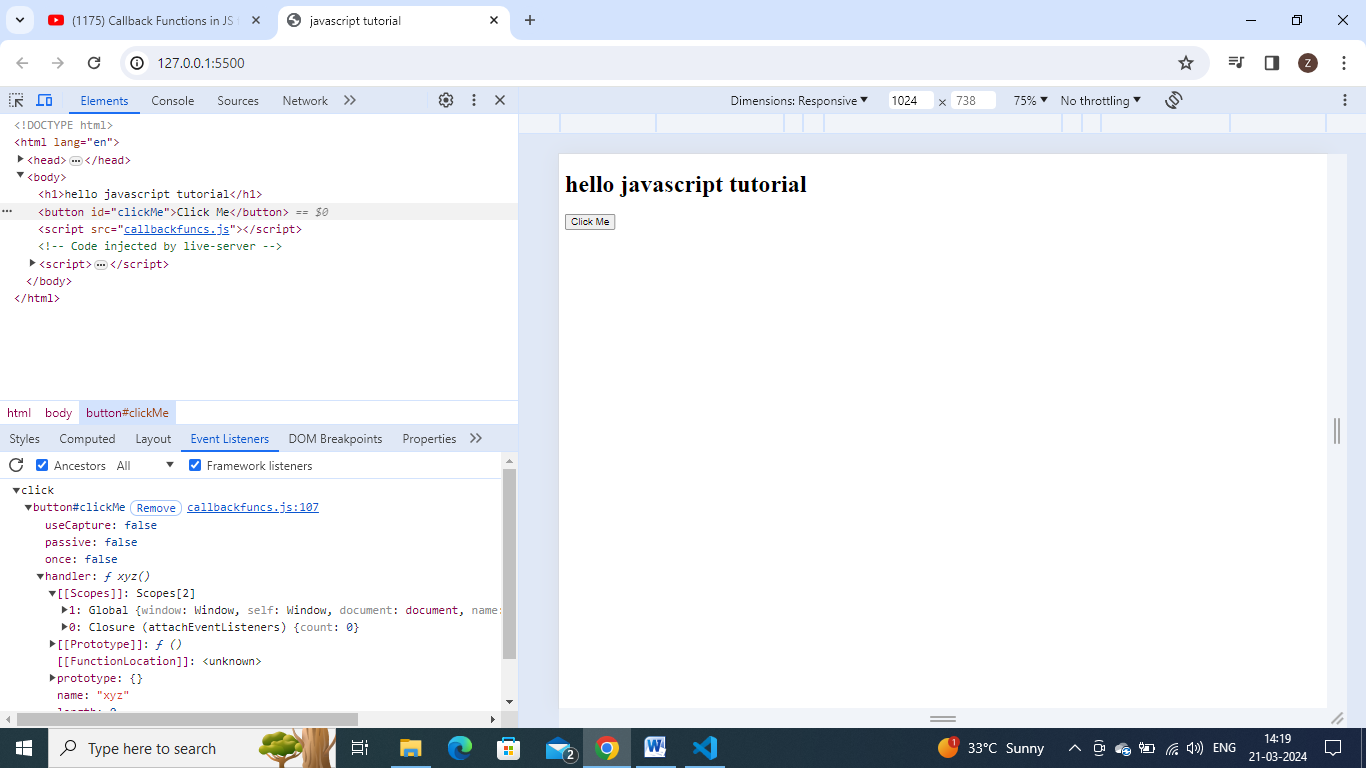
Here we see a click event listener is attached and it has a handler func- callback func same xyz function.

If we go inside this handler, we can see the scope of this handler function

This scope is the same scope which the function carries, same lexical scope.

Inside this scope, we will have its grand parent environment(global) and parent env(closure ),

Everything is inside the scope of callback function.whenever, this callback function is executed, has a scope attached to it.



**Garbage collection and remove event listeners:**

Event listeners are heavy, it means it takes memory,

So whenever , u attach any event listener,it forms a closure with this count var and when the callstack is empty,but also see it is not freeing memory for event listeners for count var, see in event listener tab

This is y it is heavy, so we remove event listeners when we wont use them.

If we have many buttons and have many event listeners, our page goes slow, so many closures sitting in memory, they store their scopes and callback functions hold these scopes, so it is better to free up them.

If I remove this event listener then all the variables, which are held by this closure will be garbage collected.