**Debouncing - https://www.freecodecamp.org/news/deboucing-in-react-autocomplete-example/#:~:text=Debouncing%20is%20a%20strategy%20used,a%20certain%20period%20of%20time.**

**Hoisting:**

**Everything in js is stored inside execution context- 2 parts - mem execution, code execution**

**Callstack will have global econtext first then ec1,ec2**

**Variable will get stored into execution context at mem execution phASE as x:undfined**

**Whereas func will store in mom execution phase itself with its def as its value**

**Hoising:**

So in case of variable var x => console.log(x) before initialixing is getting undefined

But console.log(func) before initializing func, we get func code.

Bcoz run this code in terms of memory execution and code execution.

        var x=7;

        function getName(){

            console.log('namaste javascript')

        }

        getName();

        console.log(x)

        console.log(getName)

**reason: so funcs , before running/ before execution (2nd phase) only, funcs will get initialized with func code, but vars initialize with undefined-**

**Arrow funcs:**

        getName();

        console.log(x)

        console.log(getName)

        var x=7;

        // function getName(){

        //     console.log('namaste javascript')

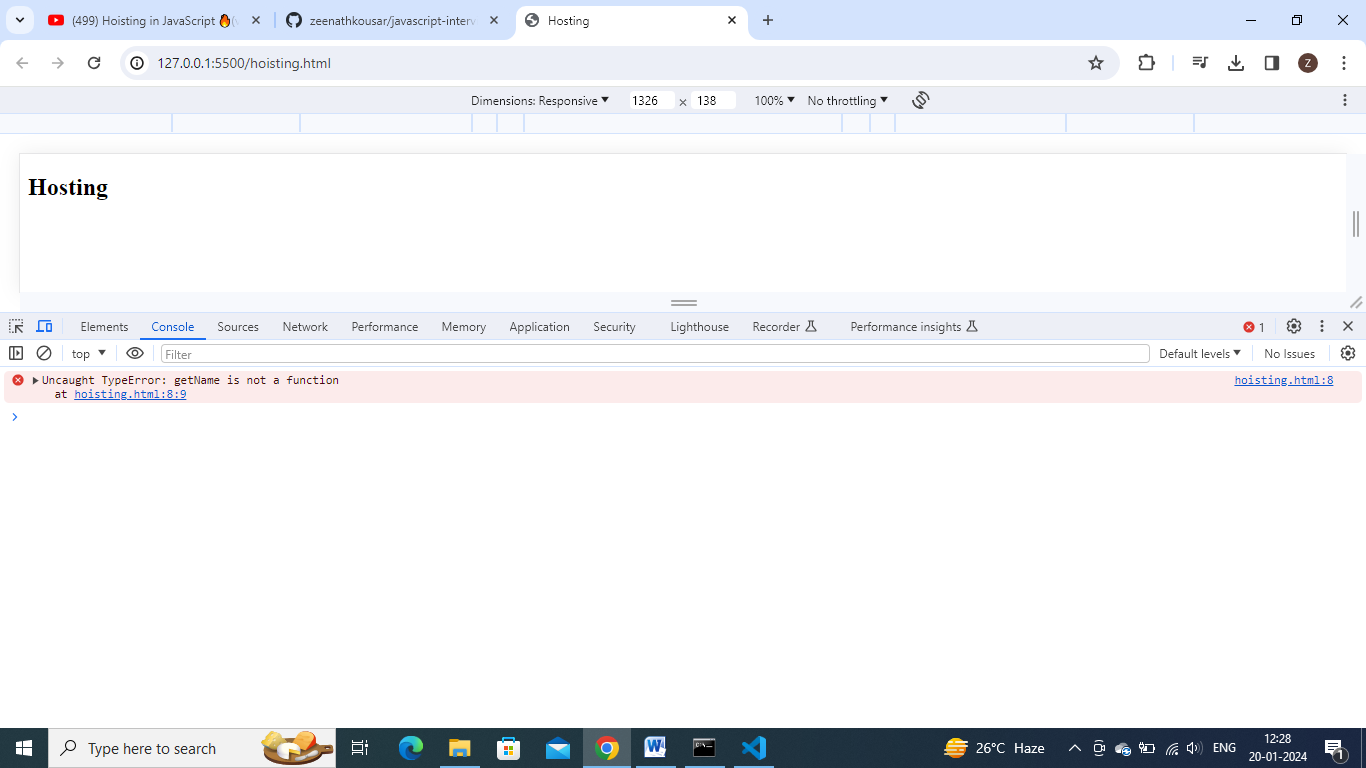
        // }

        var getName=()=>{

            console.log('namaste hello')

        }

**As arrow funcs – means assigned func value to a variable=> err**



**Arrow func behaves like a variable.**

        //another way to define func

        var func2= function getName (){

        }

**In this way also,It work as a variable**

**Var,let,const:**

Var,let,const – 3 ways we can declare variables in js.

Var is exists in js since its invention

Let and const introduced in es6 version to overcome limitations of var.

**Questions based on scope:**

Scope- is a certain region of a program ,where a defined variable exists and can be recognized and beyond that it cannot be recognized.

Types of scope- global , block,functional scopes

Var a=5 ; -> it means it is in global scope

Can be accessed anywhere.

Console.log(a) -> u get 5 as output in console.

So var is a global scope , but let and const are block scope.

// eg: write var inside a  block and try to acccess it outside the block, still it will be accessible.

{

  var a = 5;

}

console.log(a); // get outpt as 5 in console.

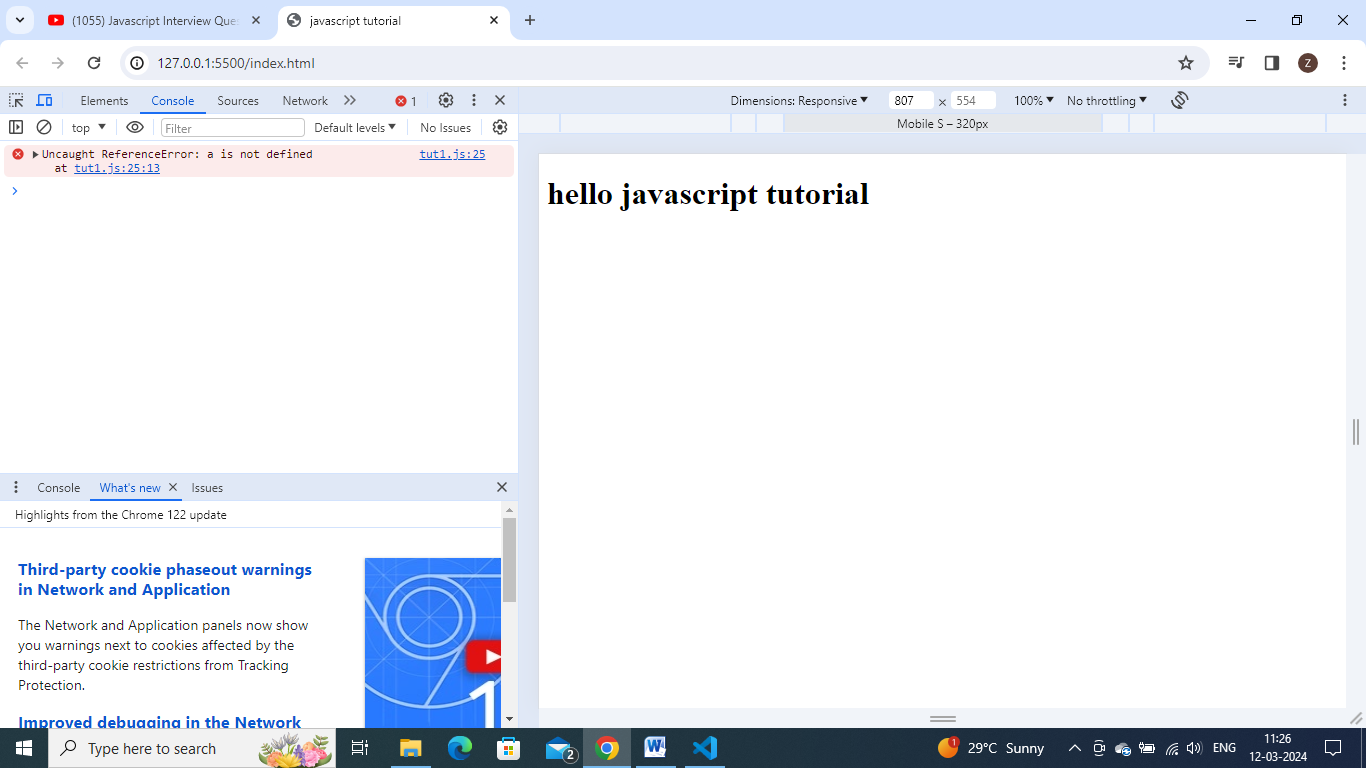
//but chnage it to let , u get reference err- a is not defined.

{

  let a = 5;

}

console.log(a);



So let variable is accessible only inside it block.

//now trying to access it inside that block itself. - u get output as 5

{

  let a = 5;

  console.log(a);

}

//similaly const

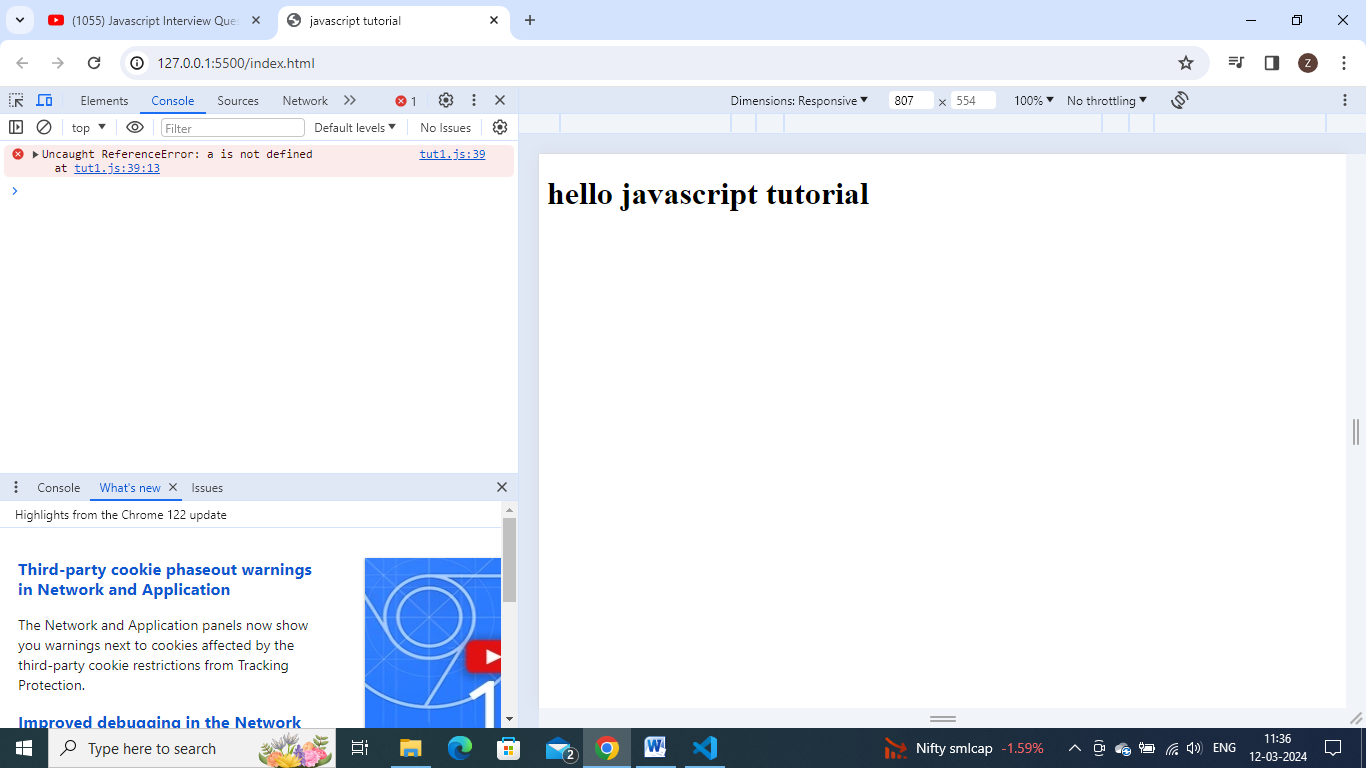
{

  const a = 5;

}

console.log(a);

get ref err as accessing const variable outside scope.



{

  const a = 5;

  console.log(a); //get output as 5 as accessing it inside its block only

}

The introduction of let and const in es6 along with block scoping allows variable shadowing.

**What is variable shadowing.**

//lets create a variable inside a function

function test() {

  let a = "hello";

  console.log(a); //here simply u get ouput as hello

}

test();

**shadowing:**

//lets create a block inside this function  and create a variable with same name inside that block also. and try to access it inside that block.

function test() {

  let a = "hello";

  if (true) {

    let a = "Hi";

    console.log(a); // u get Hi here

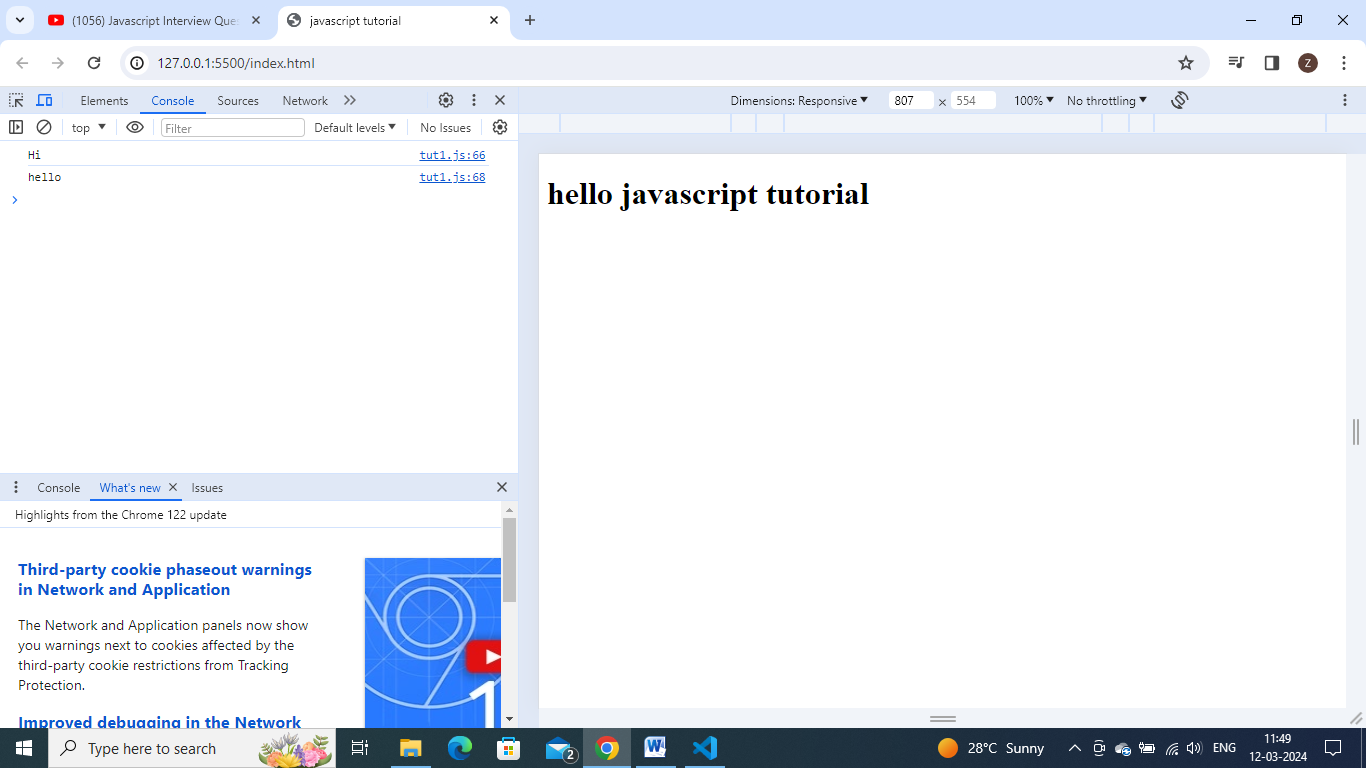
  }

  console.log(a); //here simply u get ouput as hello

}

test();

//the console.log() inside that if block is printing its own value 'HI" means Hi is overriding Hello value. it is nothing but inside a variable is shadowing  the outside value , but outside of block , the outer a is still acceesible., only inside block it is ovarriding.



**Illegal shadowing:**

//so,while shadowing a variable it should not close the boundary of scope.that is we can shadow  var variable using let  but cannot use the opposite.

//if u try to shadow let variable by using var , it is known as illegal shadowing and it gives us the error  that variable is already defined.

**eg**: shadowing var using let - same output as let shadowing let

function test() {

  var a = "hello";

  if (true) {

    let a = "Hi";

    console.log(a); // u get Hi here

  }

  console.log(a); //here simply u get ouput as hello

}

test();

// eg: shadowing let using var -

function test() {

  let a = "hello";

  if (true) {

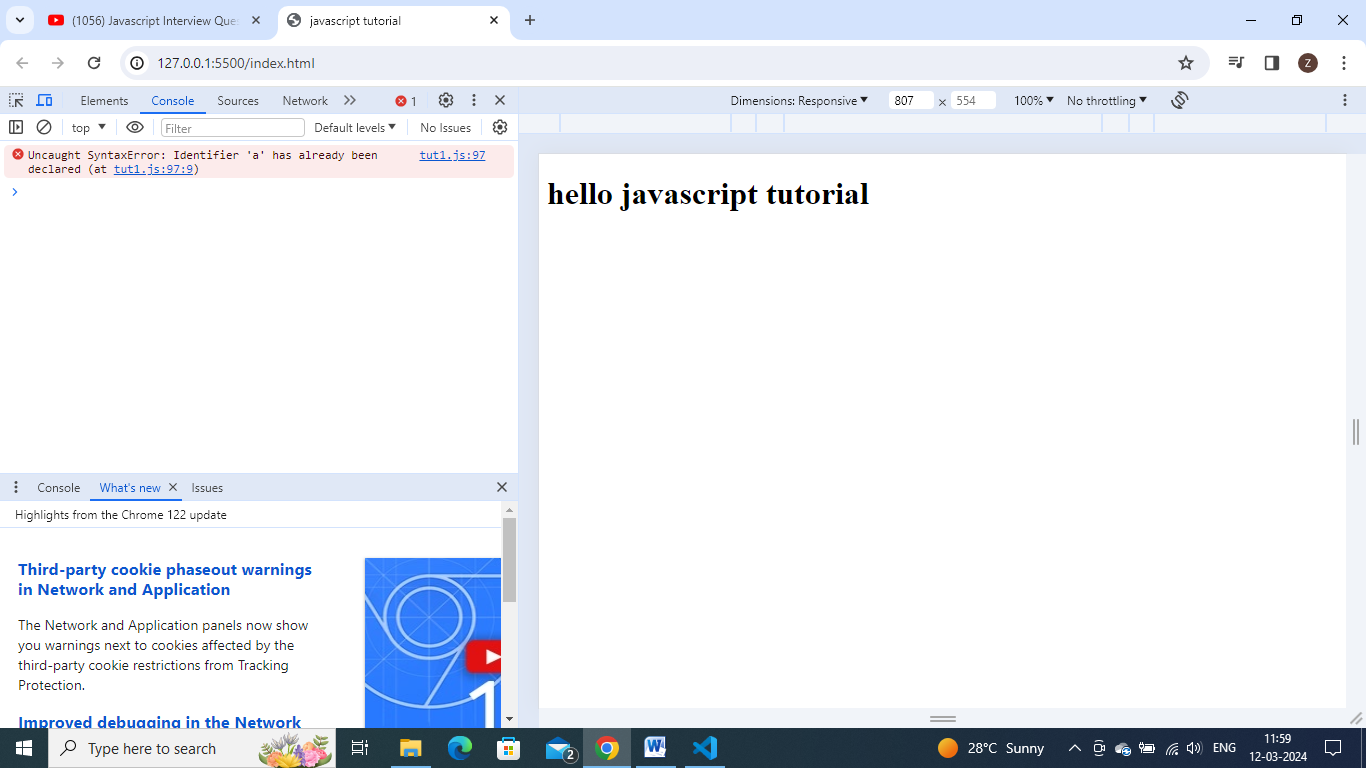
    var a = "Hi"; //we get error here as a is already been declared.

    console.log(a);

  }

  console.log(a); }

test();



This above case is known as illegal shadowing.

**Type2 : Declaration Questions:**

// if we do var=a; then again doing var=a , then its absolutely fine. - no error

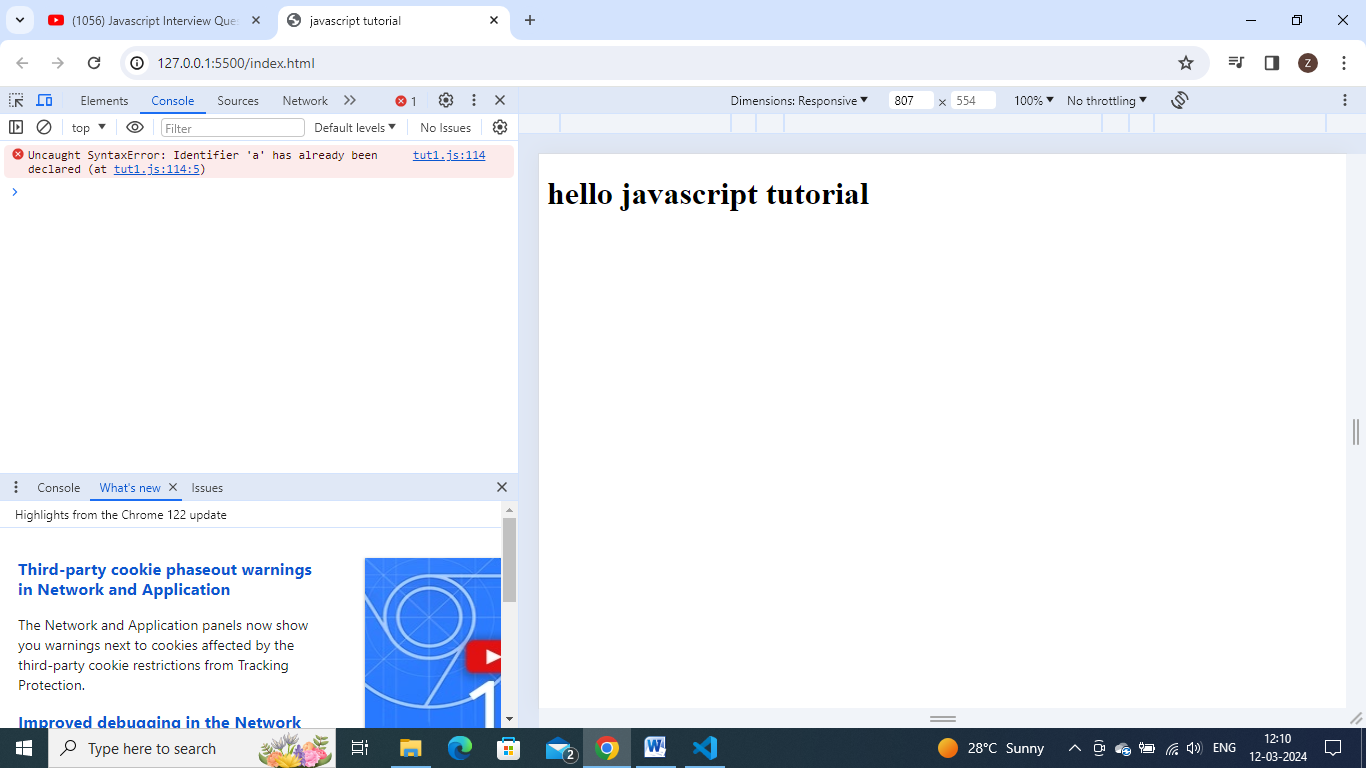
var a;

var a;  //- no err, we can redeclare var it as many times we want.

// but try to redeclare let , it gives  err as 'a' has already been declared.

let a;

let a;

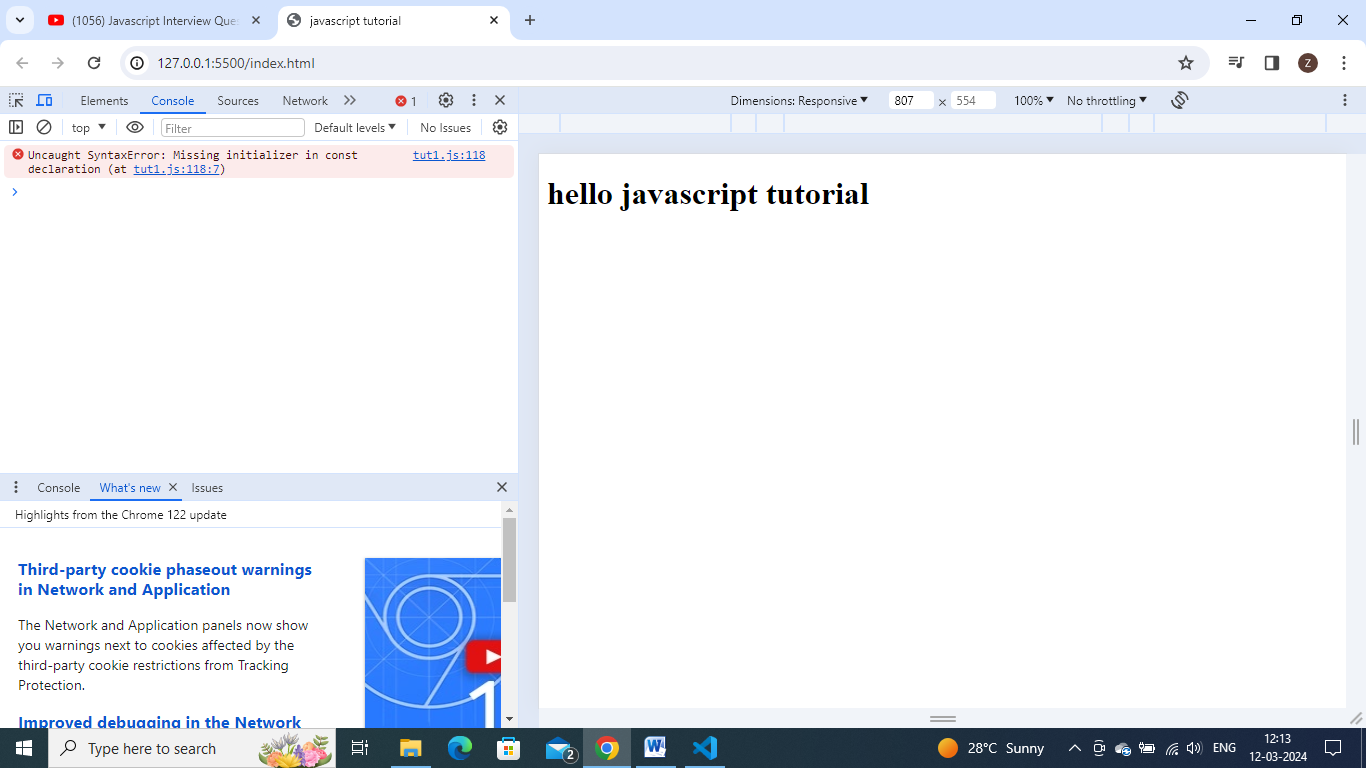


So we cannot redeclare a variable using let.

Const:

//const  variable, we cannot just declare, we have to initialize it with some value while declaring itself.- otherwise get err as missing initializer.

const a;



//while redeclaring const also gets err - const and let cant be redeclared

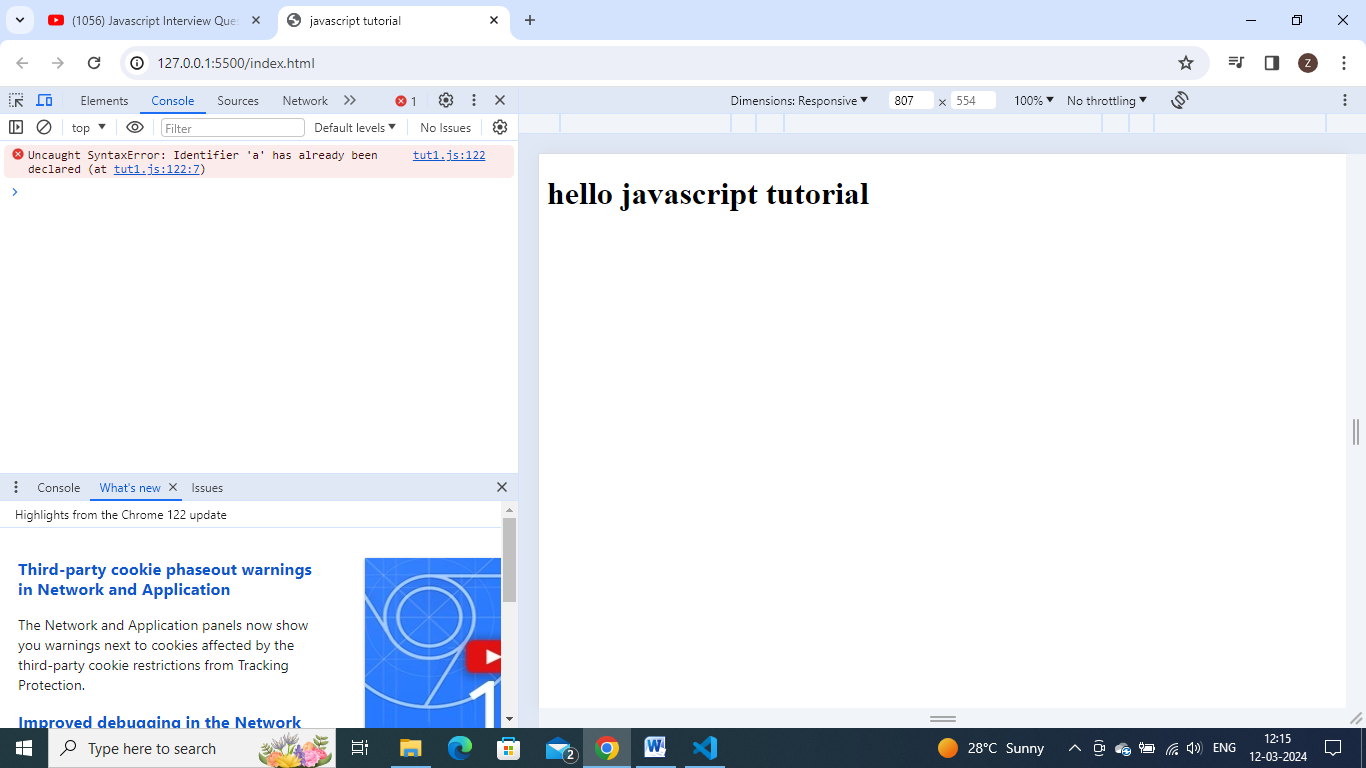
const a;

const a;

//while redeclaring const also gets err - const and let cant be redeclared

const a = 10;

const a = 11;



// // but if we do like this then its fine as we already discusssed this as it comes under shadowing

let a;

{

    let a;

}

**Type 3 questions : Declaration without Initializaiton.**

// Type 3 questions : Declaration without Initialization

//var  - declaring without initialization is totally fine

var a; //fine

var a = 10; //  initialization and declaration in one line is also fine.

//with let also it is completely fine

let a;

let a = 10;

//but with const - we have to initialize it when declaring only. otherwise we get initialization err at declaration.

const a; // missing initializer in  const declaration

we have to initialize it with some value ,while declaring itself as const a=10;

**type 4: Re initialization Questions:**

// var - fine no err - we can do it

var a=5;

a=6;

//let -  fine no err - we can do it

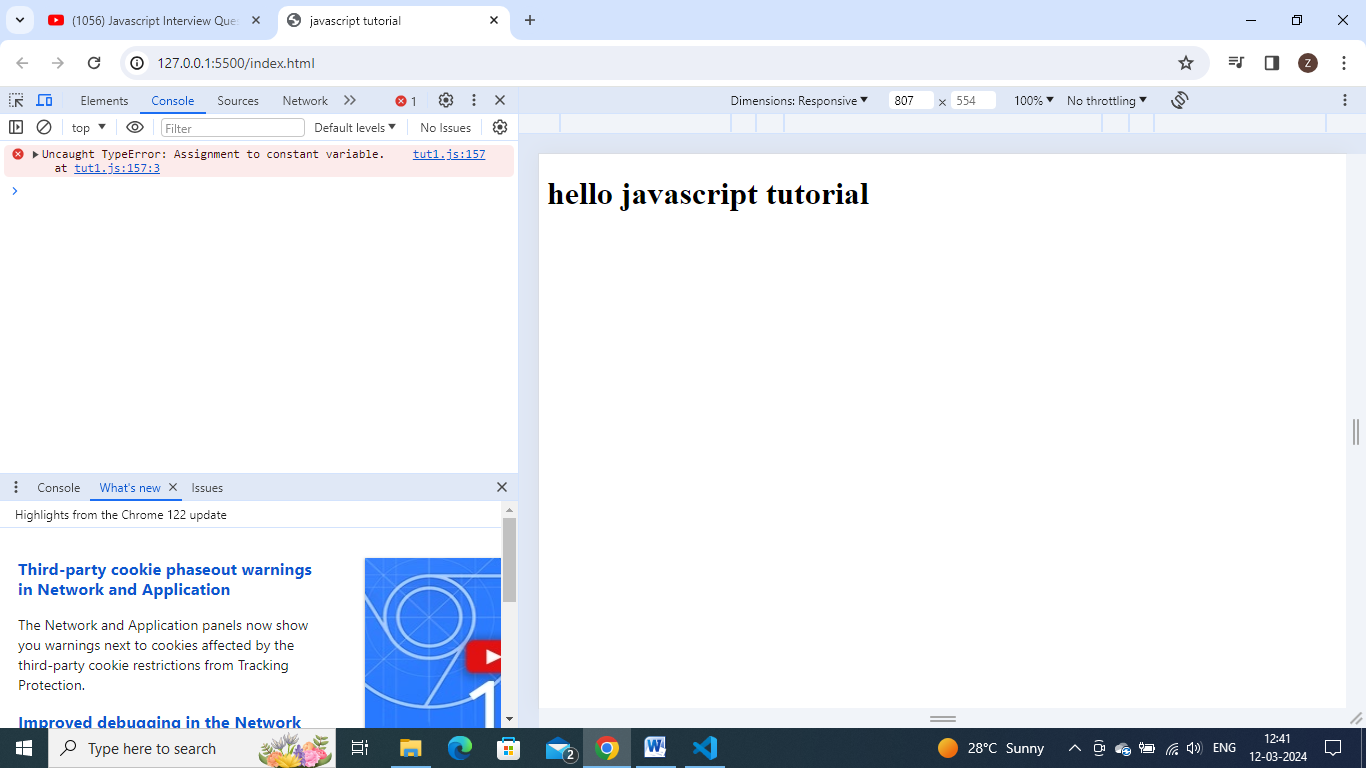
let a=5;

a=6;

//const - err cant change the value of const - so cant reinitialize

const a = 5;

a = 5; //assignment to a const vaiable err



//so var and let can be updated but const value cant be updated.

**Type 5 Questions – Hoisting – very important topic.**

Before understanding hoisting lets check how execution context works.

//let say we have a function  as below

let a = 10;

function multiply(x) {

  return x \* 10;

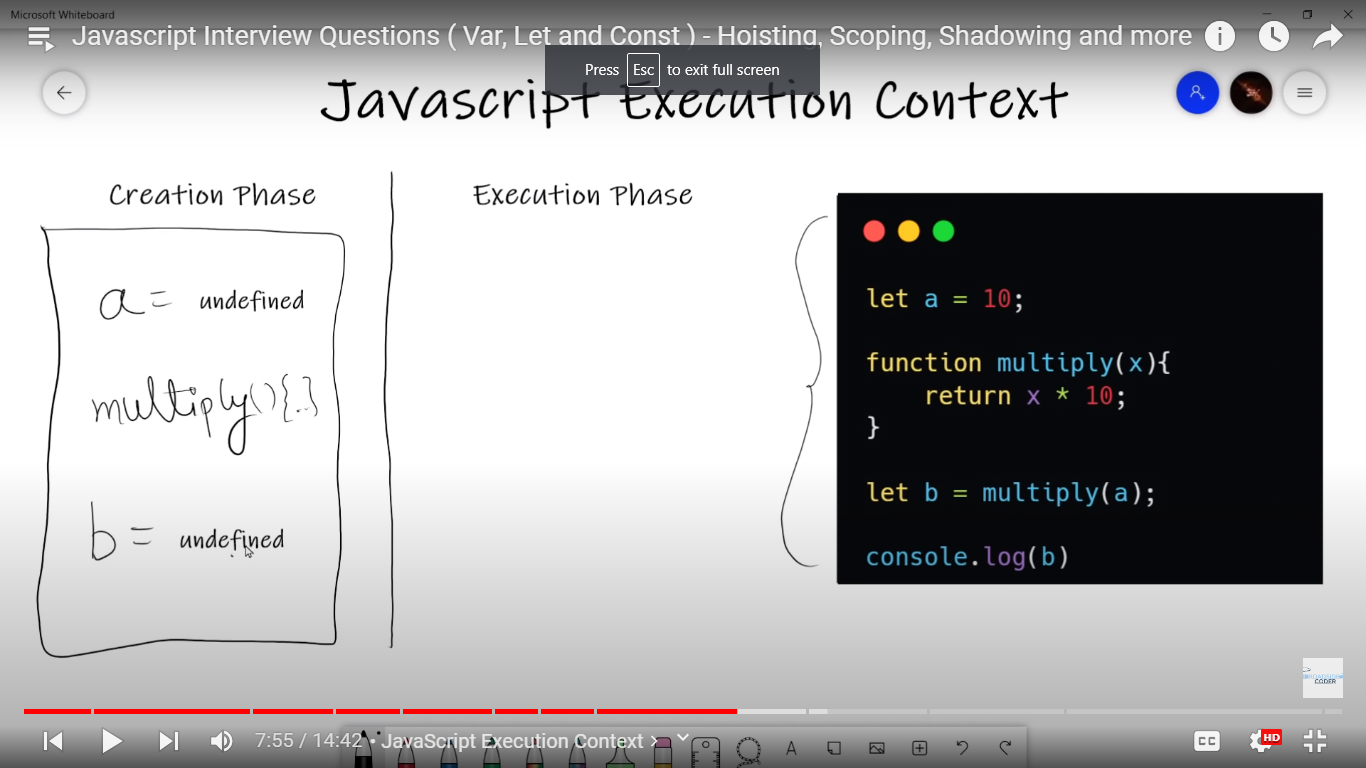
}

let b = multiply(a);

console.log(b);

//to execute this there are 2 phases- creation phase and execution phase.

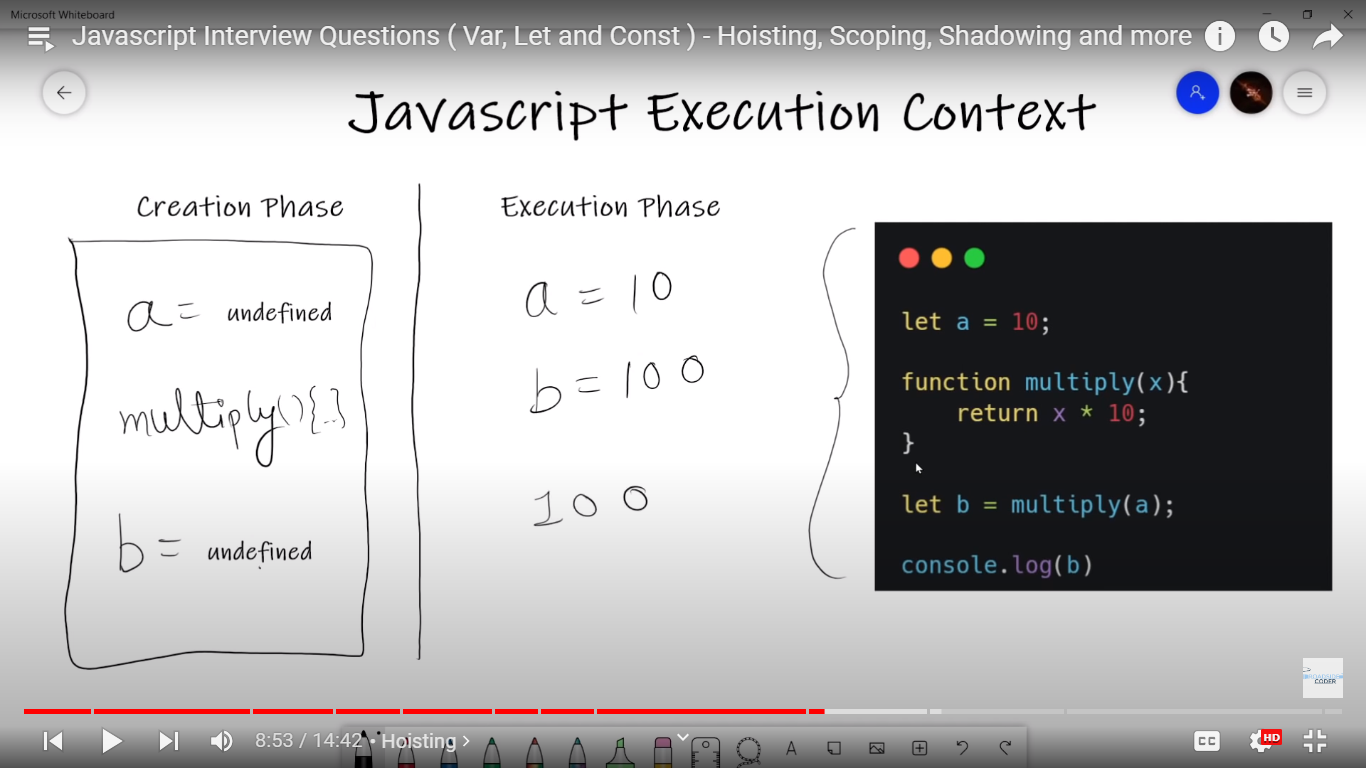
In creationphase – 3 things happens- it creates a window / global obj…..the next step is it setup memory heap for storing variables and func references and next step is it will initialize those functions and variable declarations with undefined.



During execution phase- javascript engine executes the code line by line assigning values to variables and execute function calls.

Also for every new function created , js engine will create new execution context all together.

Let because of this undefined assigning only hoisting concept came, we see it detailly later



Js also uses **callstack** – a mechanism to keep track of all function calls.

**Hoisting:**

So during creation phase, js engine moves ur variables and function declarations to top of ur code.this is known as hoisting.

console.log(count); //we get undefined , we didnt get err as declaration var count - is moved to top

var count = 1;

in creation phase – count = undefined

//this is how js execution context works, it declares all variables and functions at the top during creation phase , annd in execution phase, it checks whether this variable exist in creation phase or not, obviously it was present but value was undeined so got undefined.

Js will look the above code as this:

var count;

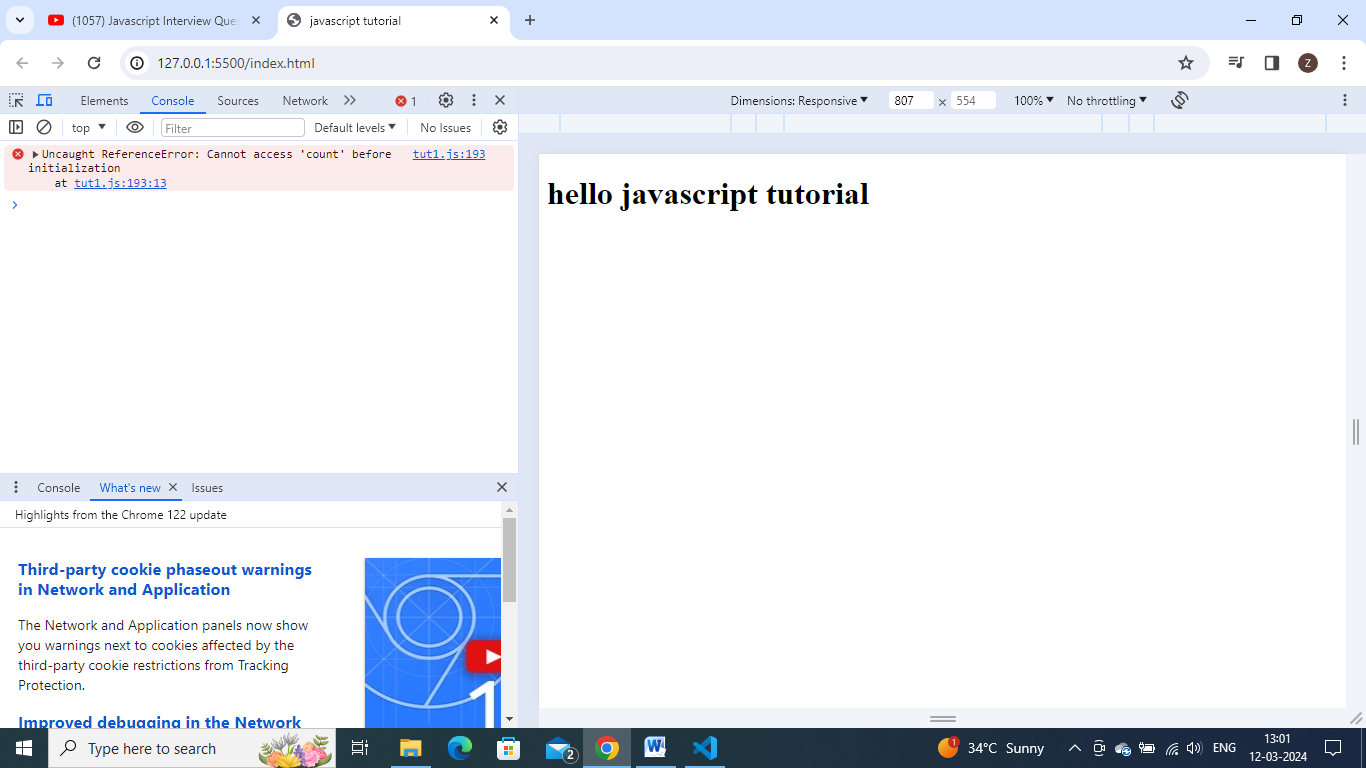
console.log(count);

count =1;

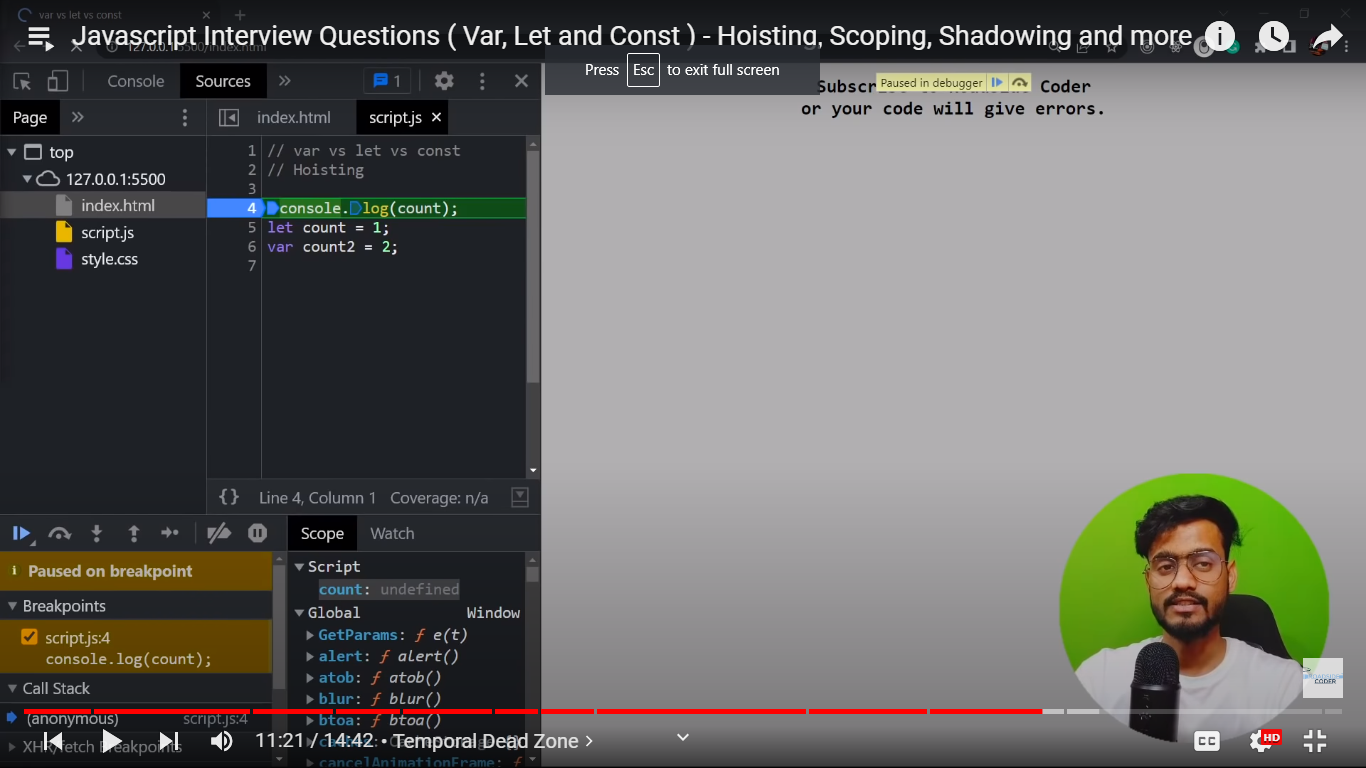
//trying with let

console.log(count);

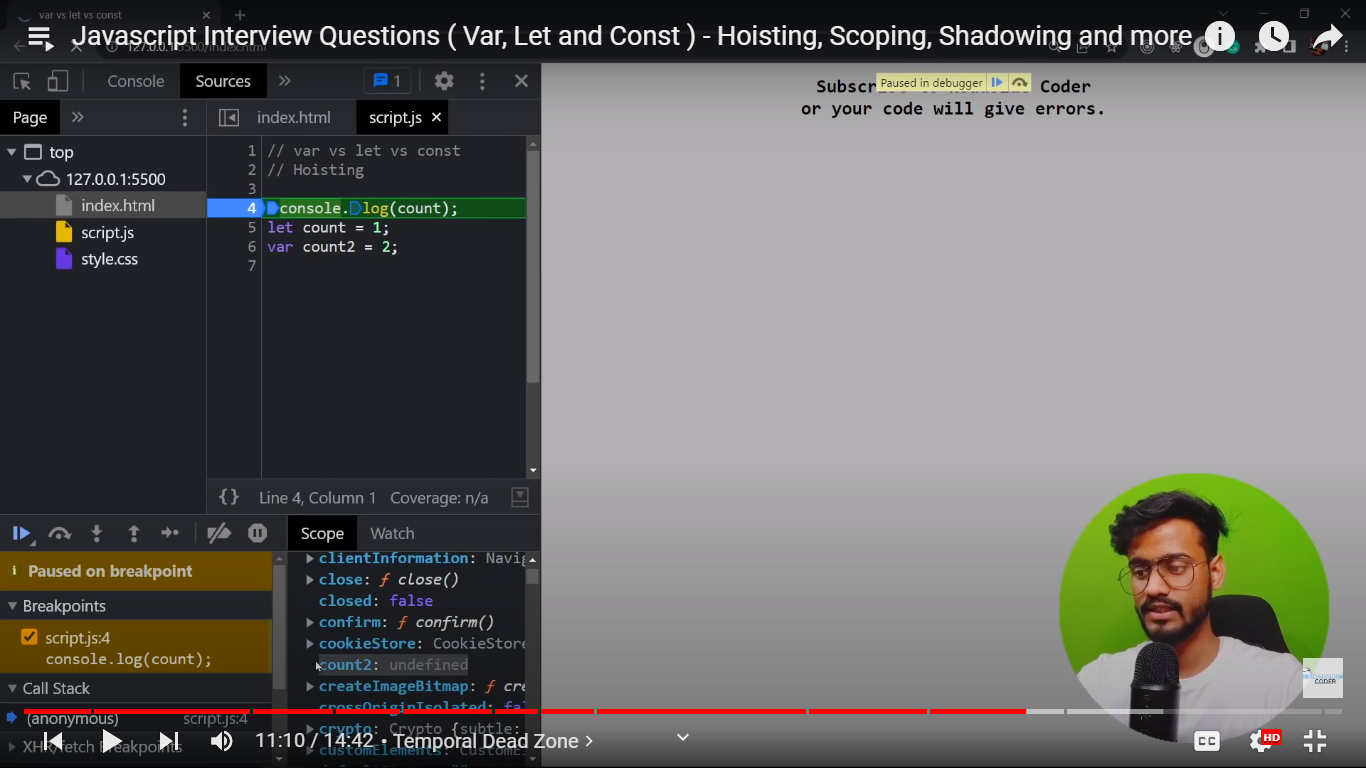
let count = 1;



As we got err – cannot access count before initialization – in this way let is overcome the limitations of var.



The variable declared with let =>count variable also got initialized with undefined but not in global space, its in some other space.



Wheareas , variable count2 is undefined in global scope.

So let variable is in separate scripts space. It is known as temporal dead zone.

**Temporal Dead Zone: is the time between the declaration and initialization of let and const variaables.**

// question- functional hoisting

function abc() {

  console.log(a); // we get undefined - bcoz it has initialized this function but it has not initialized this variable yet.

  //if console.log was after the var a=10 then it will print 10

  var a = 10;

}

abc();

//question2 -

function abc() {

  console.log(a, b, c);

  //we know here a will be undefined but for let and const (b,c) it acts differently.

  //are these two variables also gets hoisted - yes but in temporal dead zone.

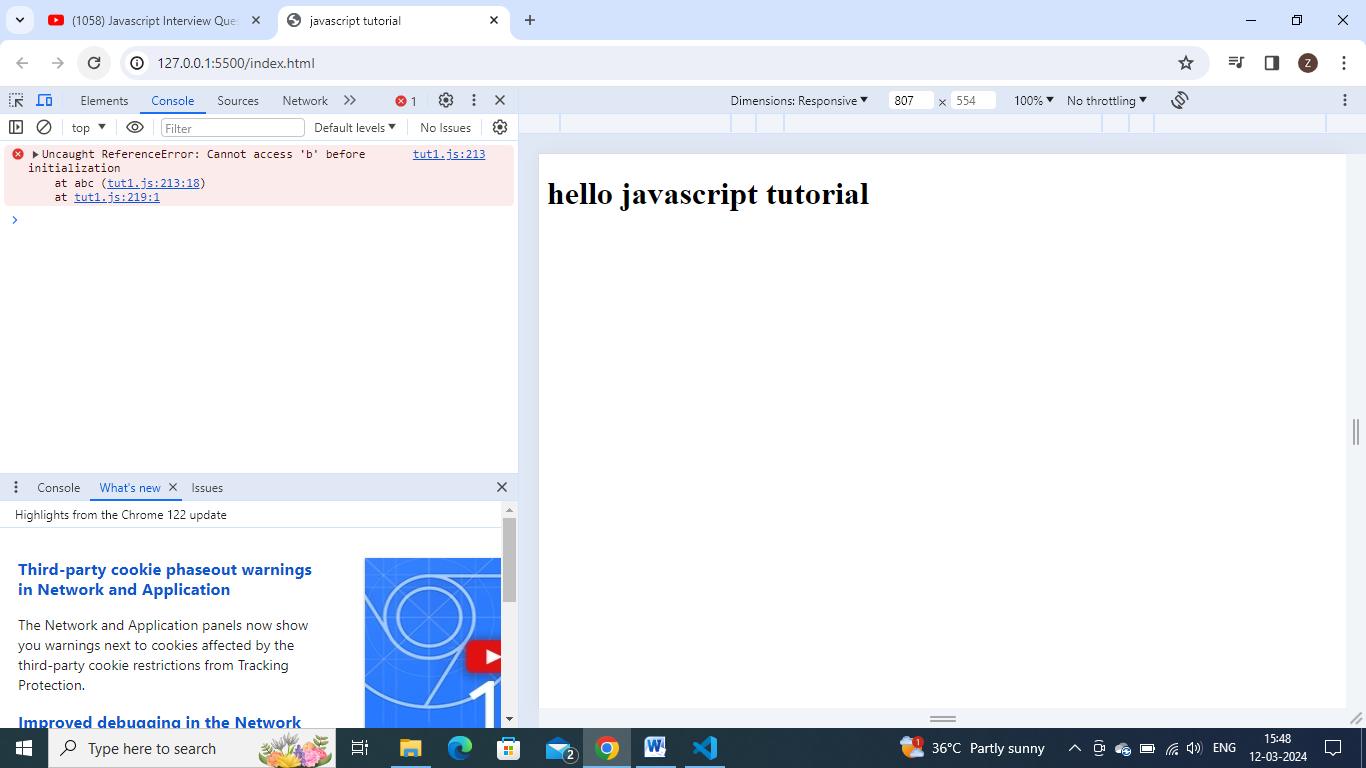
  const c = 30;

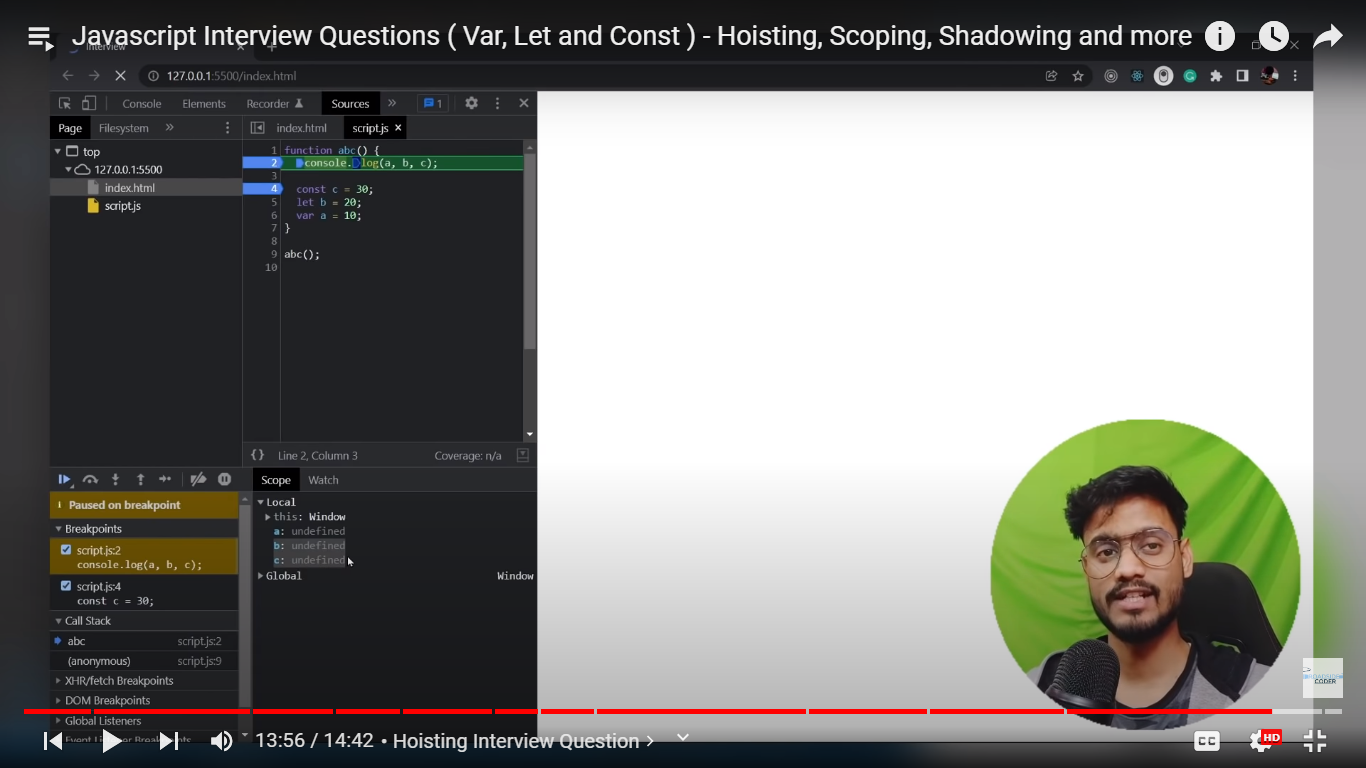
  let b = 20;

  var a = 10;

}

abc();





**Here these two variables b and c gets**  initialized in temporal dead zone.- term to describe A STATE WHERE variables are in the scope but they are not yet declared.

**Map filter reduce:**

//map, filter ,reduce are array methods , usually used to iterate a array , perform transformation/computation.

//each may or may not return a new array based on computation.

//what is map()?

// the map () method is used to create a new aray from existing one by applying a function to each elements of first array.

const nums = [1, 2, 3, 4];

//syntax : map(currentelement,index,originalarray)

const multiplyThree = nums.map((num, i, nums) => {

  return num \* 3; //it takes each element from array , multiply by 3 and return completely new array multiplied with 3.

});

console.log(multiplyThree); //output - [3,6,9,12]

//adding index

const multiplyThree = nums.map((num, i, nums) => {

  return num \* 3 + i; //it takes each element from array , multiply by 3 and return completely new array multiplied with 3.

});

console.log(multiplyThree); //output - [3,7,11,15]

what is filter()?:

filter method will takes each element from array and apply a condition on each element-> if consition returns true, the element gets pushed into output array. If condition returns false, element will not get pushed into conditional array.

Filter the data which fulfills certain criteria.

const nums=[1,2,3,4];

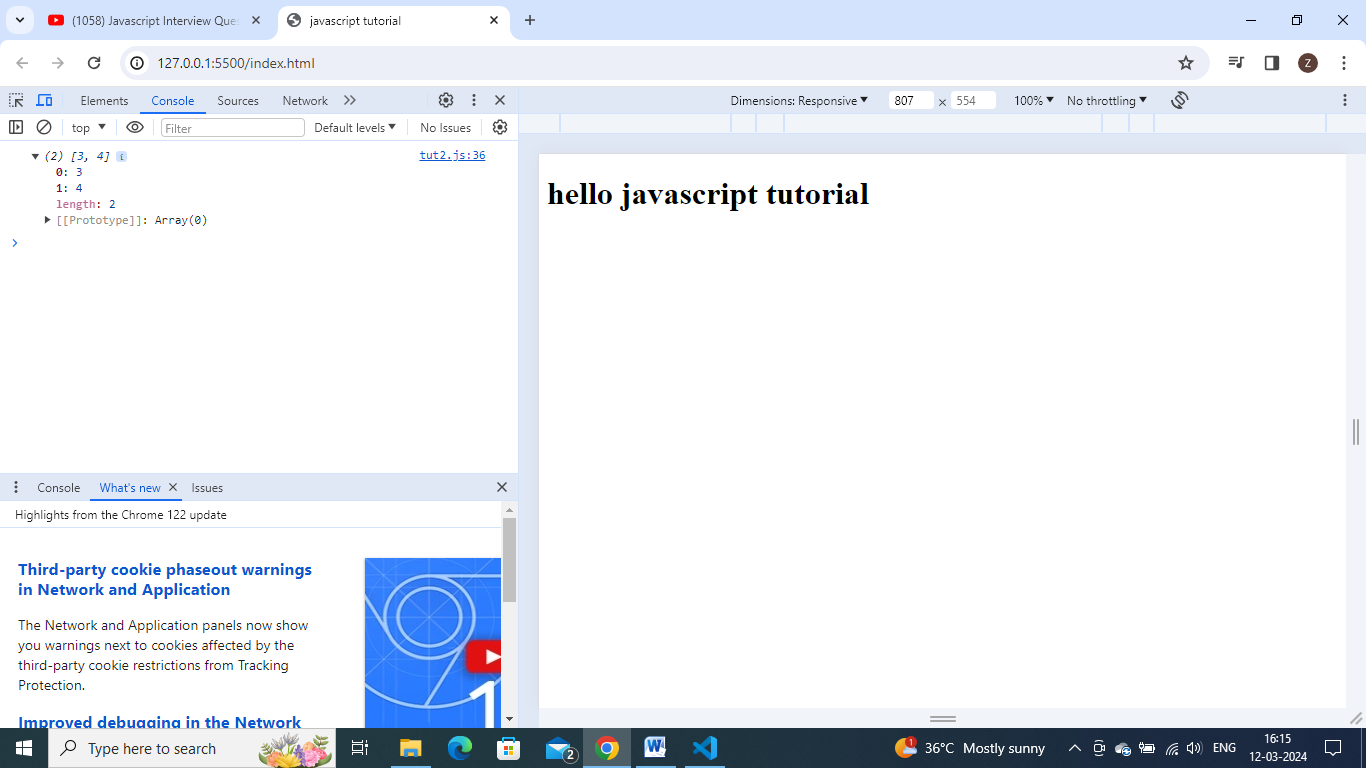
// syntax : filter(currentnum,index,originalarray) - here i dont need of index and original array  so writing only num

const moreThanTwo=nums.filter((num)=>{

    return num>2;

})

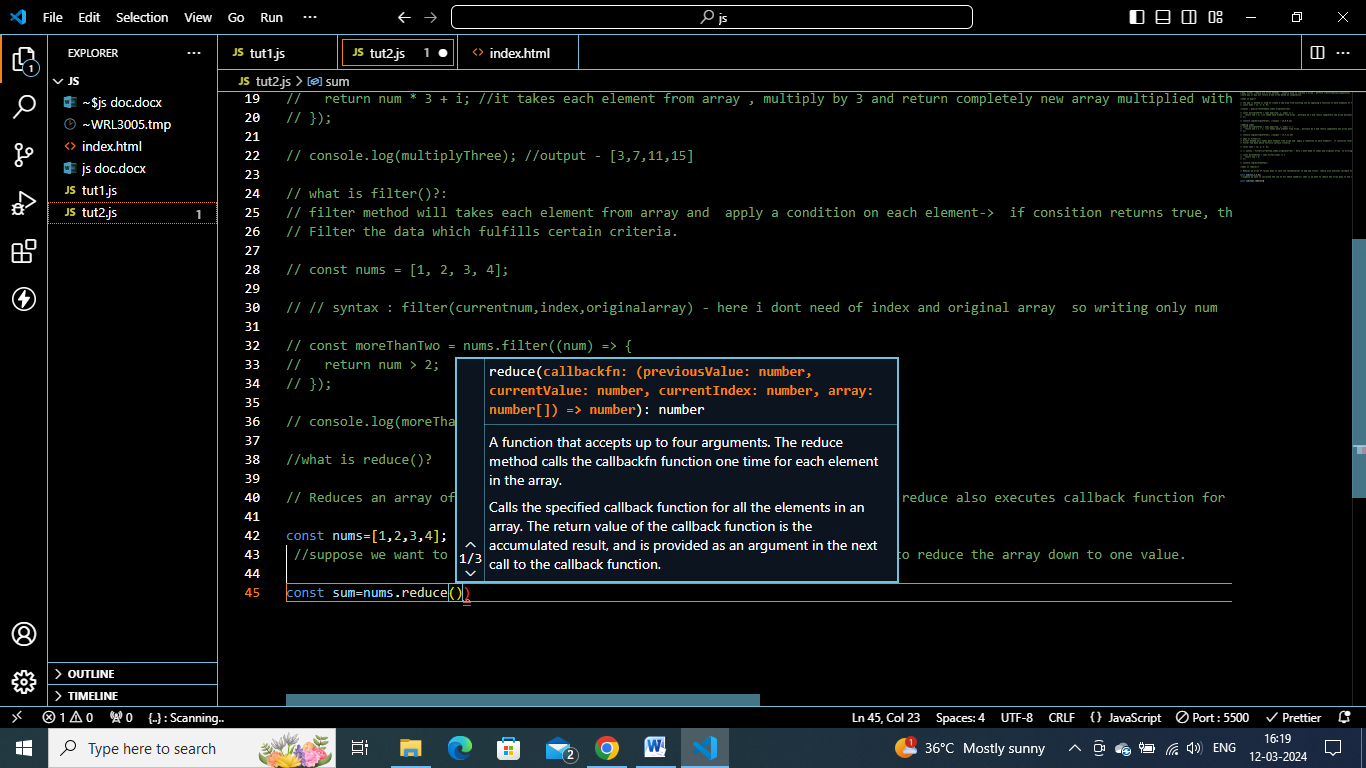
console.log(moreThanTwo)



See we got only 3 and 4, as they are greater than 2.

What is reduce()?

Reduces an array of values down to just one value…similar to map and filter, reduce also executes callback function for each element of array.



It takes one callback fun and an initial value.

const sum=nums.reduce(()=>{},0)

let initial value be 0.

This callback func has different parameters –acc,currvalue,index and arr

Accumulator is the result of previous computation.

Currvalue=>present element of array.

const nums = [1, 2, 3, 4];

//suppose we want to calculate the sum of all these numbers=> that is we want to reduce the array down to one value.

const sum = nums.reduce((acc, curr, i, arr) => {

  return acc + curr;

  //initially acc will  be 0 => 0+1;

//   then acc becomes 1 =>1+2 => 3;

//then acc becomes 3 =>  3+ 3 =>6

//THEN 6+4 =10

}, 0);

console.log(sum) //10

Let variables also gets hoisted but in temporal deadd zone.

**Map ,filter,reduce: from Namaste javascript:**

// Map(),filter() , reduce() are higher order functions.

// Higher Ordder functions :In Javascript, functions can be assigned to variables in the same way that strings or arrays can. They can be passed into other functions as parameters or returned from them as well. A “higher-order function” is a function that accepts functions as parameters and/or returns a function.

Higherorderfunc- take one or more functions as arguments or return a function

Map() function is basically used to transform an array.

Transforming array is nothing but doubling each and every element of array, tripling each and every element of array., or to perform any such operation on each and every element of array and getting a new array out of it.

// eg:  let we have an array

const arr=[5,1,3,2,6]

// double : [10,2,6,4,12]

//triple: [15,3,9,6,18]

//Binary: ["101","1","11","10","110"]

So to transform whole array – in such a situation we can use map() function.

Implementing it in code.

const output=arr.map()

inside this , we need to pass a function, which tells u that wt transformation do we need.(double,triple or binary or any).

Lets create function code for double .

function double(x){

   return  x\*2;

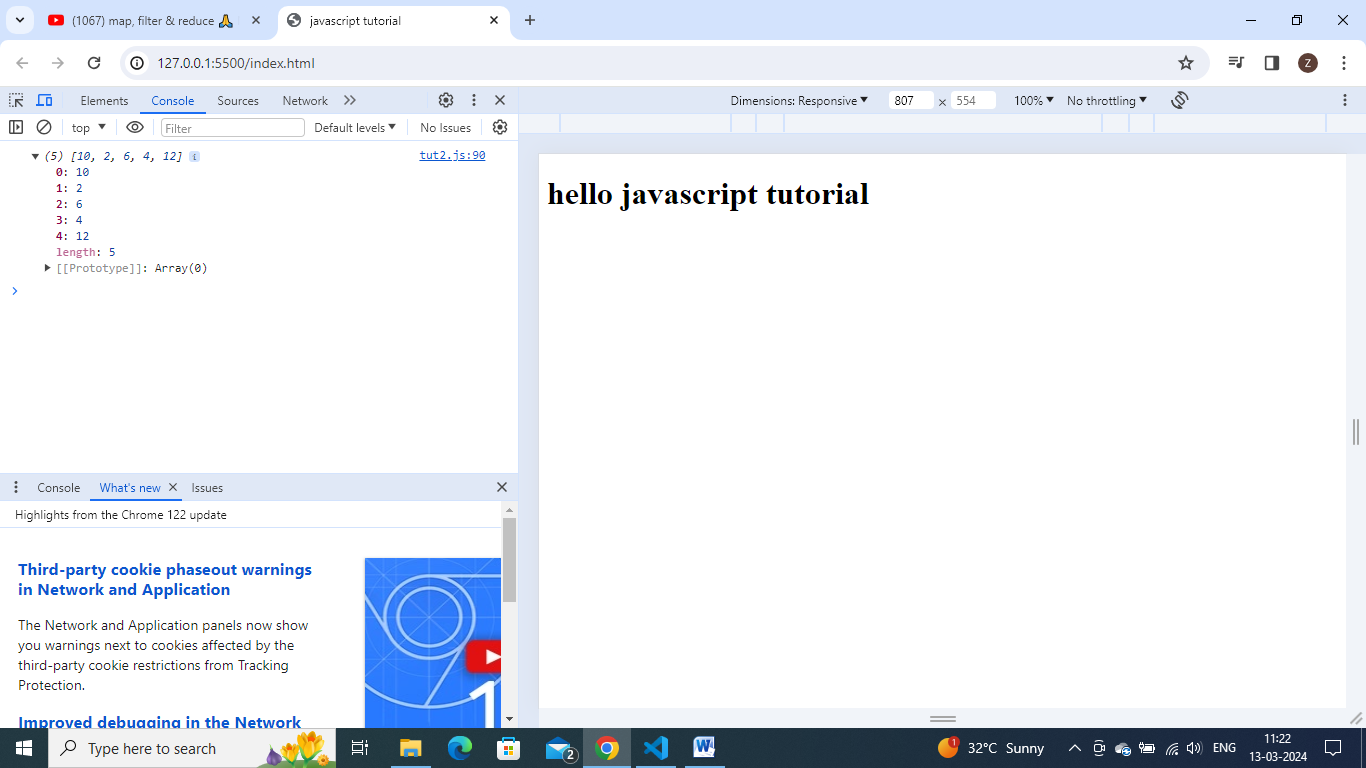
}

const output=arr.map(double)

it will run this function for each and every element of array and create a new arr out of it.

U can do console.log(output) – to see new array.

console.log(output);



Suppose u want to triple here, just instead of double function , u can pass triple function.

function double(x) {

  return x \* 2;

}

function triple(x) {

  return x \* 3;

}

const output = arr.map(triple);

console.log(output)

Lets now create a transformation logic for binary operation.

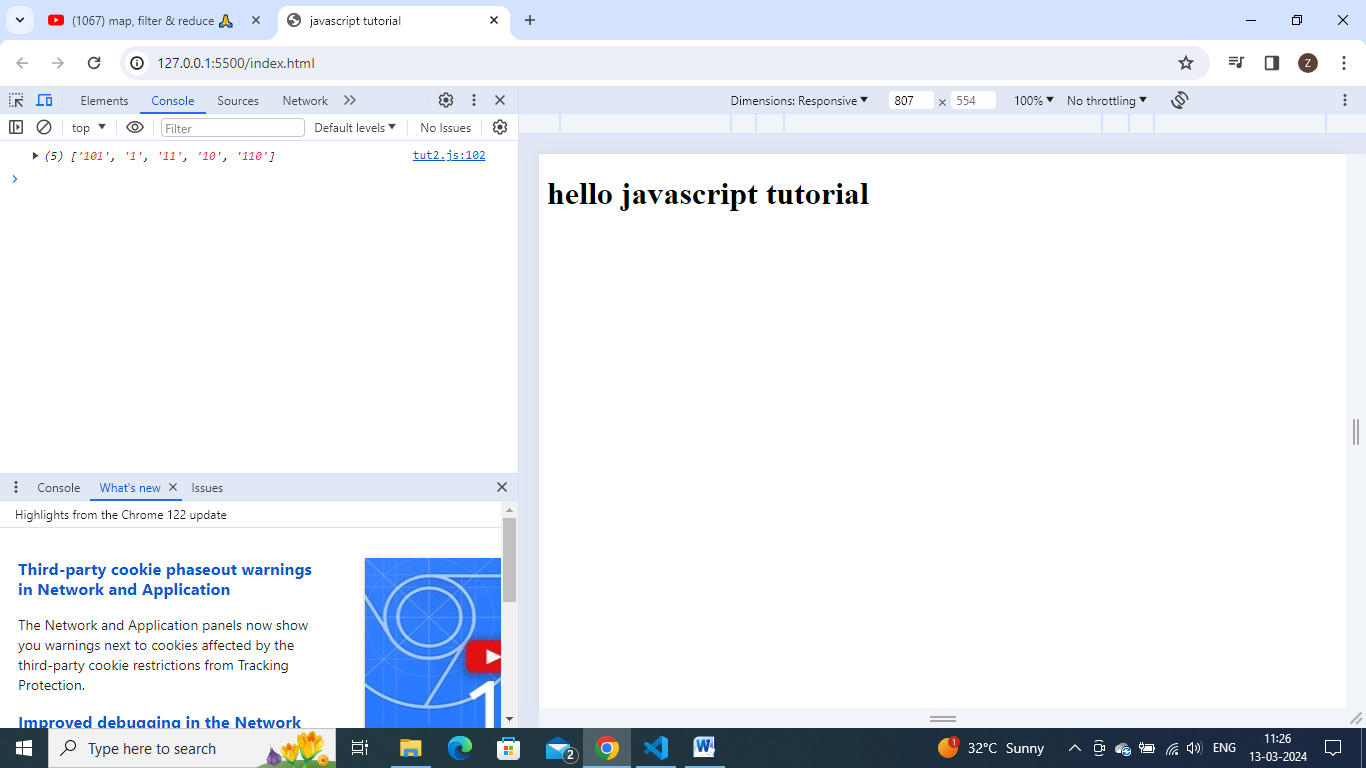
function binary(x) {

  return x.toString(2);

}

const output = arr.map(binary);

console.log(output);



The other valid syntaxes are:

const output = arr.map(function binary(x) { //keeping function directly as parameter

      return x.toString(2);

    }

    );

This is nothing but a higher order function.

const output = arr.map((x) => {

  //can pass arrow functions /anonymous functions - can remove return keyword as we have only single line

  return x.toString(2);

});

const output = arr.map((x) => x.toString(2));

//can pass arrow functions /anonymous functions - can remove return keyword as we have only single line

// so basically map() - is used for mapping each and every element of an array with a logic function.

**Filter:**

// filter is basically used to filter the values inside an array

//suppose we have an araray, and we want to filter some values from array which are odd inside it.

// eg: filtering values from array - which are greater than 4

//  or filtering all even numbers from an array.

//or filtering all the numbers divisible by 5

//so we are taking an array, and filtering out only some elements from array which satisfy some logic.

// eg: implementing code for finding odd values from Array.

const arr = [5, 1, 3, 2, 6];

//filter odd values

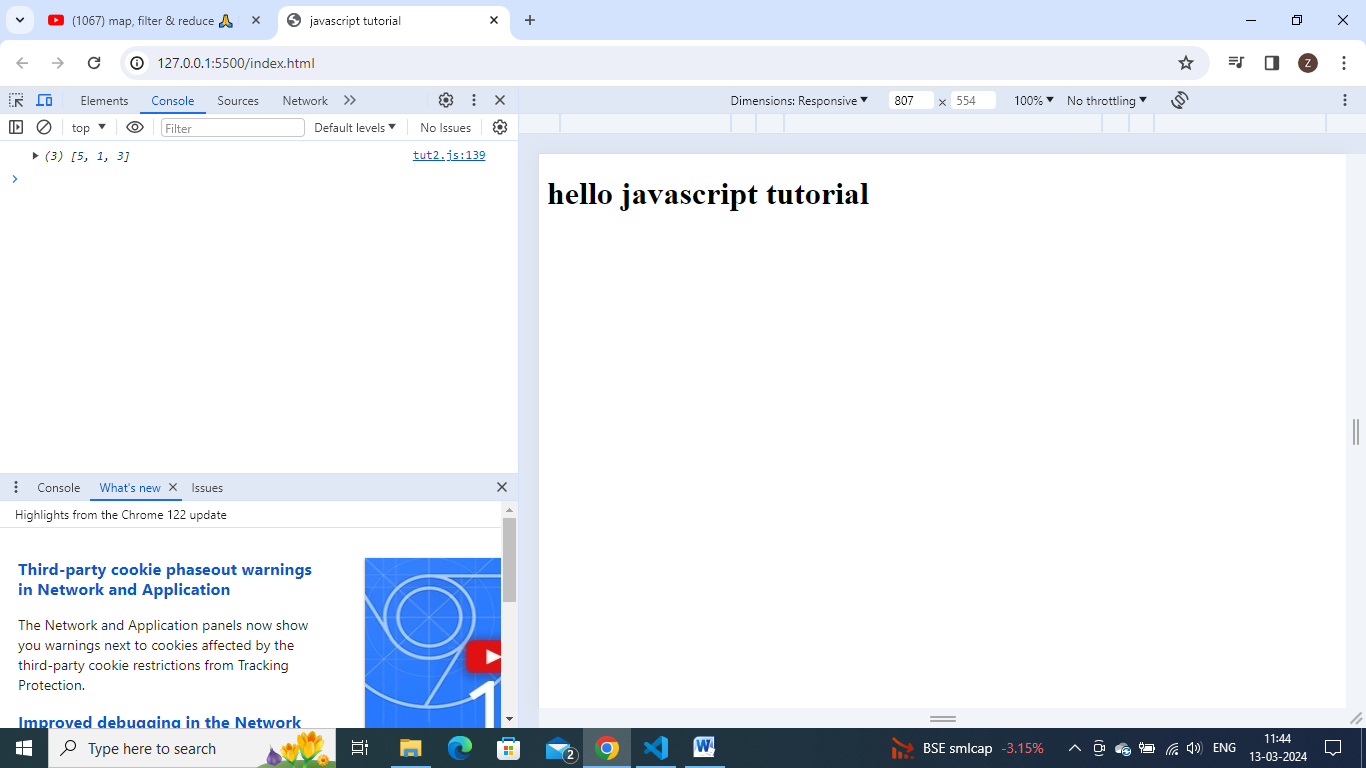
function isOdd(x) {

  return x % 2;

}

const output = arr.filter(isOdd);

console.log(output)



// eg: implenting for filtering even values

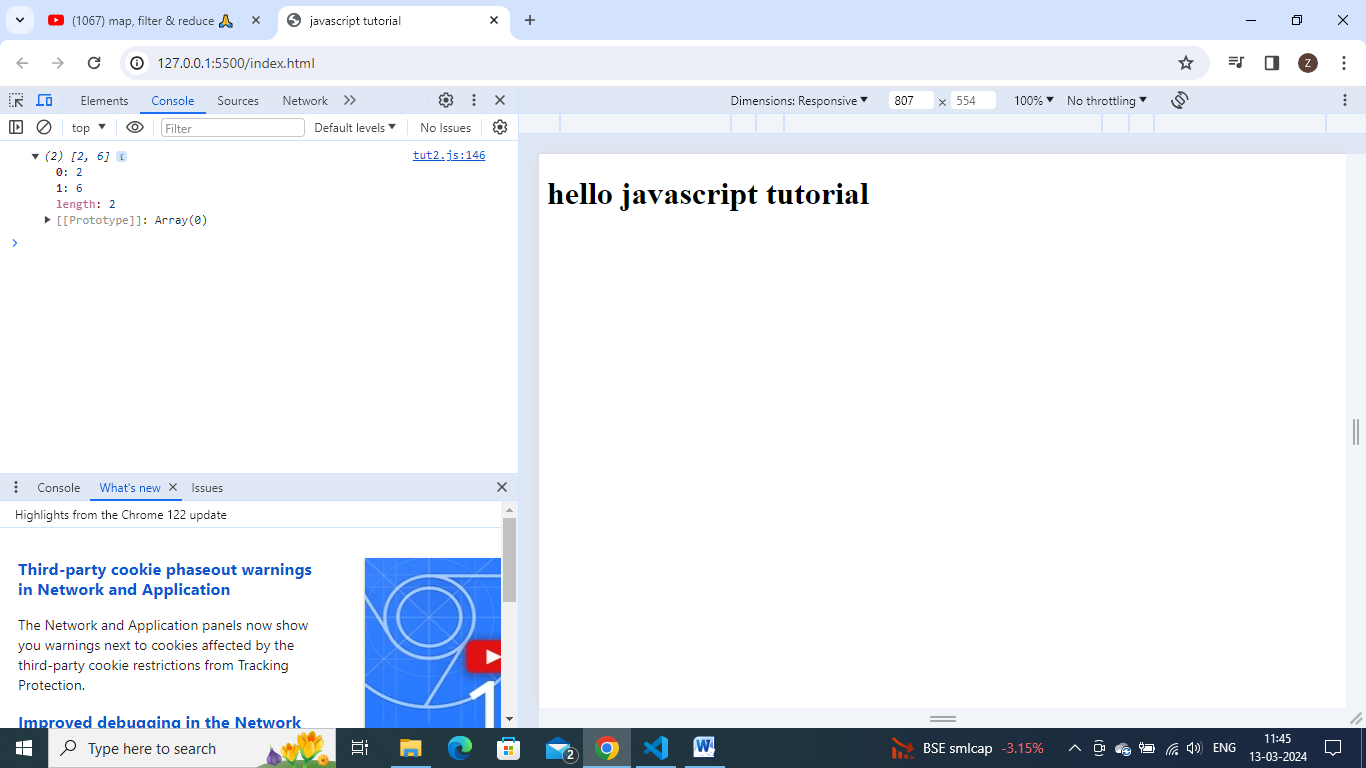
function isEven(x){

    return x%2==0;

}

const output = arr.filter(isEven);

console.log(output)



// eg: finding all values greater than 4

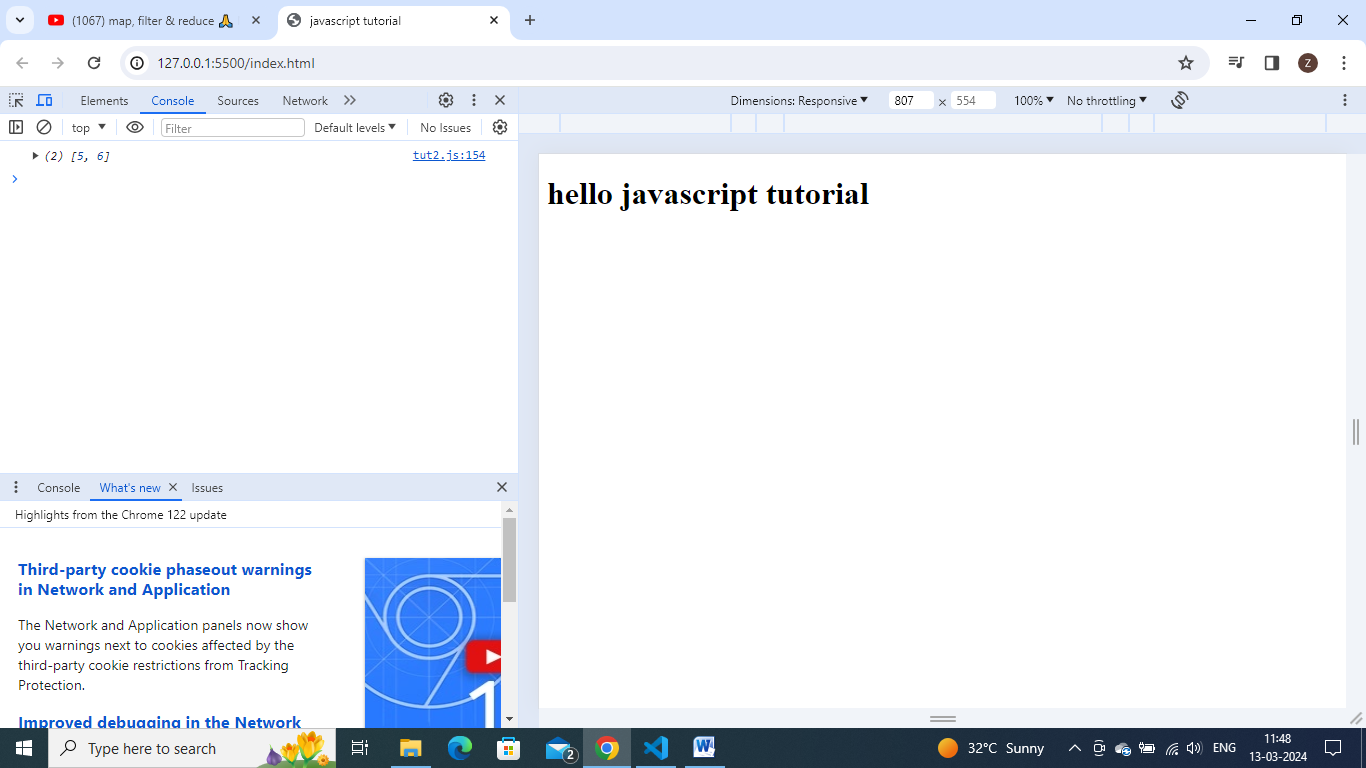
function isGre(x) {

  return x > 4;

}

const output = arr.filter(isGre);

console.log(output);



//similarly we can directly pass function to this filter method and also as arrow functions. these syntaxes also valid for this filter method.

const output = arr.filter(function isGre(x) { //valid

  return x > 4;

}

);

const output = arr.filter((x)=> x>4); //valid

**reduce():**

// Reduce method: it is not reducing anything

so its functionality is not related to the method name

so reduce is used in such a case, where u take all elements of an array and came out with only single value as a result out of it.

eg: taking an arr and iterate each and every element and find out the sum of all elemnets

// eg: or iterating whole array and finding max/largest number out of it.

// eg: finding sum without using filter

const arr = [5, 1, 3, 2, 6];

function findSum(arr){

    let sum=0;

    for( let i=0; i<arr.length;i++){

        sum=sum+ arr[i];

    }

    return sum;

}

Output: 17

// converting this func into reduce function

const output=arr.reduce(fucntion())

the inner function will takes 2 parameters –accumulator and current

const output=arr.reduce(fucntion(acc,curr){}) //the inside function will take 2 parameters- accumulator and current

this reduce func will iterate each value of an array and ‘curr’(2nd parameter) represents the current element. ‘acc’(1st parameter ) is basically used to accumulate the values/accumulate the result wt we get at each step.

If we compare reduce method with traditional function .

Curr is nothing but arr[i] and acc is sum. => so here , we can do as , acc= acc+curr; and then return acc;

const output=arr.reduce(fucntion(acc,curr){

    acc=acc+curr;

    return acc;

})

But here we need to give some initial value also for accumulator . -> that we can provide as second arg to reduce function.

2nd parameter- any initial value that u pass to ACC variable.

const output=arr.reduce(function(acc,curr){

    acc=acc+curr;

    return acc;

},0)

console.log(output)

output- 17

eg:finding maximum value from an array

// eg:finding maximum value from an array

//own tried logic

const arr = [5, 1, 3, 2, 6];

function findMax(arr) {

  let max = arr[0];

  for (let i = 0; i < arr.length; i++) {

    if (arr[i] > max) {

      max = arr[i];

    }

  }

  return max;

}

console.log(findMax(arr)); //output:6

// otherr logic

const arr = [5, 1, 3, 2, 6];

function findMax(arr) {

  let max = 0;

  for (let i = 0; i < arr.length; i++) {

    if (arr[i] > max) {

      max = arr[i];

    }

  }

  return max;

}

console.log(findMax(arr));

//writing it using reduce method ,lets try without using reference of above function

const arr = [5, 1, 3, 2, 6];

const output = arr.reduce(function (max, curr) {

  //here acc can be max, and curr can be arr[i];

  //initial value of acc=0; //y zero, array willbe non empty and positive value, so least possible positive value is 0

  if (curr > max) {

    max = curr;

  }

  return max;

}, 0);

console.log(output); //6

//lets take a complicated array.

const users=[

    {firstName:"akshay",lastname:"saini",age:26},

    {firstName:"donald",lastname:"trump",age:75},

    {firstName:"elon",lastname:"musk",age:50},

    {firstName:"deepika",lastname:"padukone",age:26}

];

This array has 3 properties firstname,lastname and age.

Let assume that this is the data which we are getting from some api.a realworld eg

Let say finding a list of full names from this obj. [“akshay saini”,”Donald trump”,”elon musk”,”deepika padukone”]

?Guess wt function will be used here.- we will use map here.

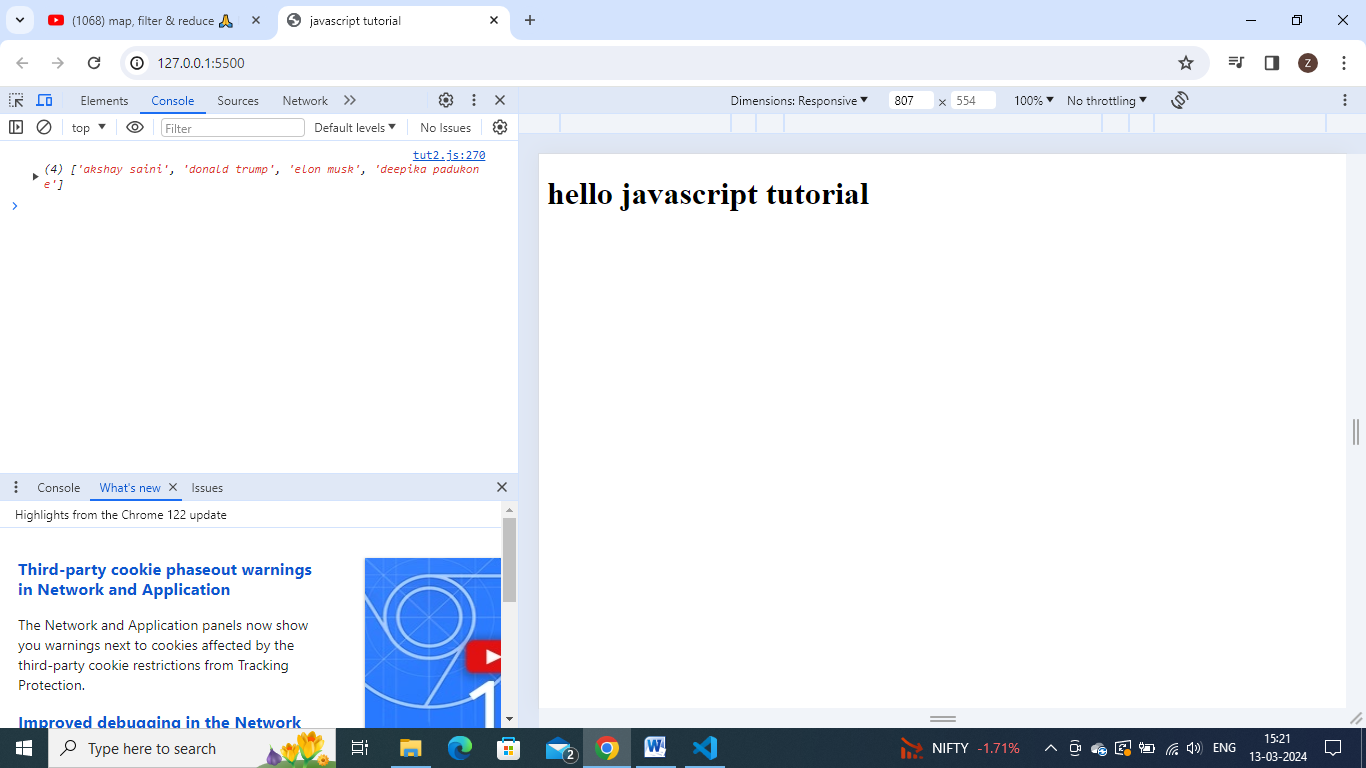
We will use map() function here.

// const output=users.map(x=>) //here x repesent the first item of array -  { firstName: "akshay", lastname: "saini", age: 26 }

const output = users.map((x) => x.firstName +" " + x.lastname);

console.log(output);

\



Eg: finding no of people with particular same age.

For this finding different kinds of ages available and then finding how many people having age 26 inside this.

{26 age:2 , 75 :1 ,50:1}

Like this if we have huge list of users, and finding no of users of particular age.

{26 age:2 , 75 :1 ,50:1} we have to make a list like this, should we use map,filter,reduce.

We use reduce – bcoz from whole array – we need find a single value digit . we need result as single object - {26 age:2 , 75 :1 ,50:1} with different unique values with count of individual values. As we want to reduce our array to one particular value i.e. an obj.

const output=users.reduce(function (acc,curr){

},initial value)

Over here, our acc is the obj {26 age:2 , 75 :1 ,50:1} . so initial value will be an empty obj.

//curr points to each obj at a time.

const output = users.reduce(function (acc, curr) {

  if (acc[curr.age]) {

    acc[curr.age] = ++acc[curr.age];

  } else {

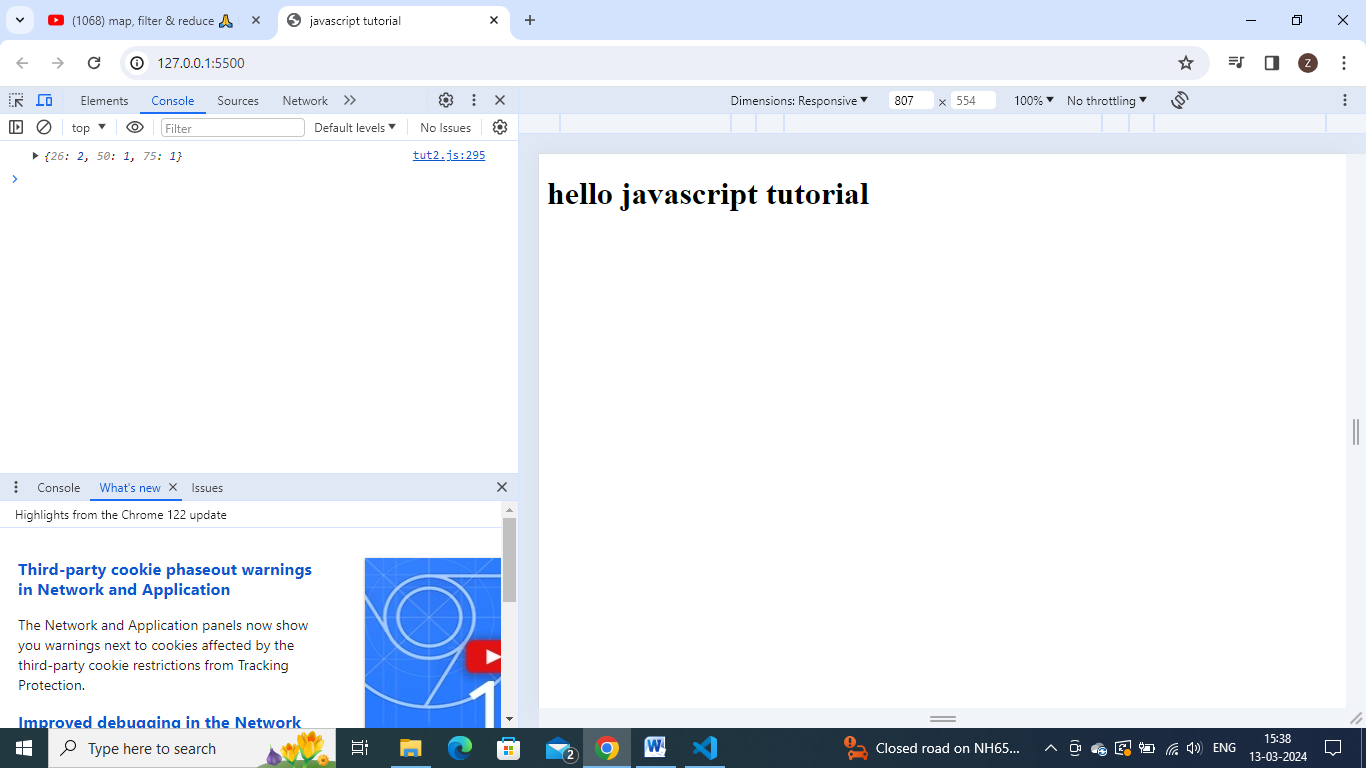
    acc[curr.age] = 1;

  }

  return acc;

}, {});

console.log(output);



Eg: finding firstnameof all users whose age is less than 30;

const output = users.filter(function (x) {

  let names = [];

  // if (x.age < 30) {

  //   names = x.firstName;

  //   console.log(names);

  // }

  // return names;

});

console.log(output);

const output = users.filter(function (x) {

  return x.age < 30;

});

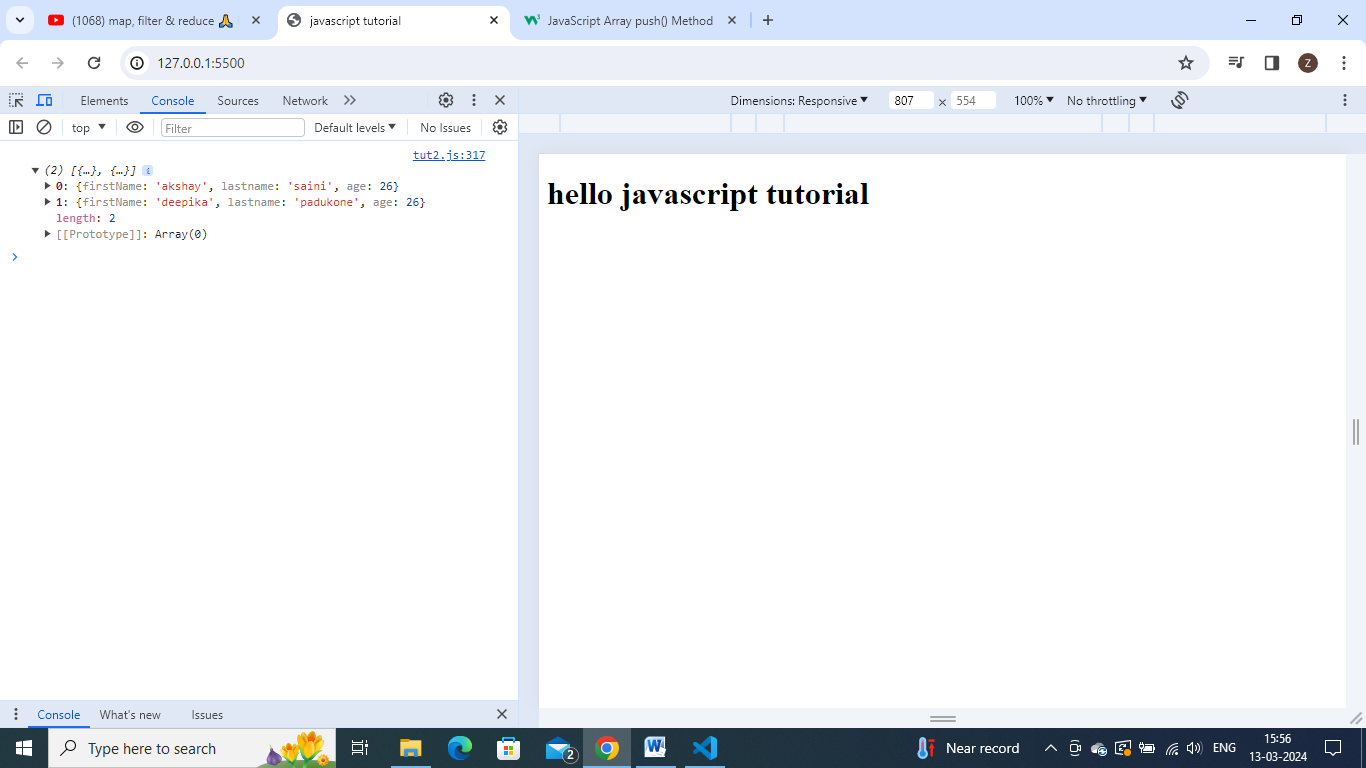
for (i = 0; i < output.length; i++) {

  console.log(output[i].firstName);

}

const output=users.filter(x=> x.age <30);

console.log(output)

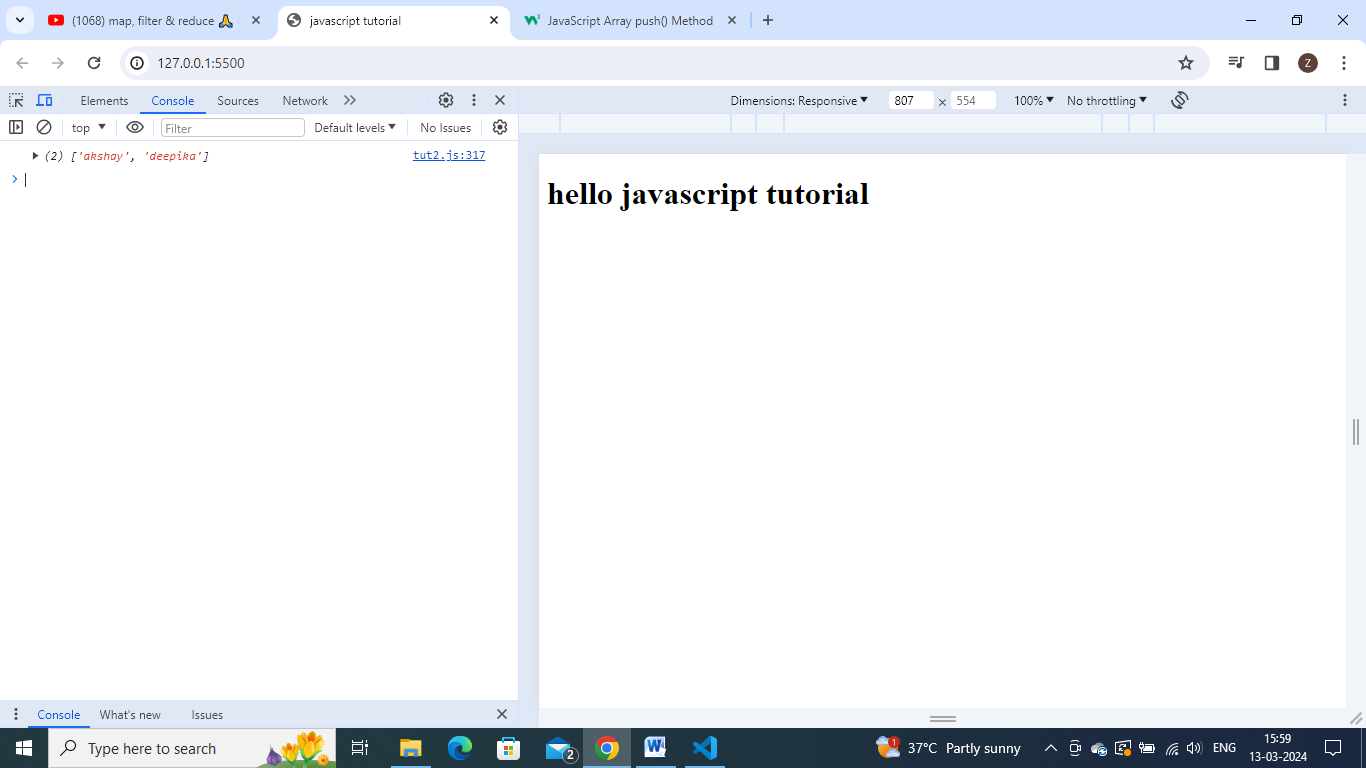


//suppose we want output as ["akshay","deepika"]

U can chain map,filter ,reduce , u can run map also on the above filter function.

Map will run on output of it.

const output = users.filter((x) => x.age < 30).map((x) => x.firstName);



We got the desired output.

Instead of filter and map, u can use reduce also.

const output = users.reduce(function (acc, curr) {

  if (curr.age < 30) {

    acc.push(curr.firstName);

  }

  return acc;

}, []);

console.log(output);

