**Topics**: let and const hoisting , let and const – temporal dead zone

Yes let and const declarations get hoisted , but they gets hoisted very differently than var hoisting.

Or u can say that they will be in temporal dead zone for time being.

//hoisting in var

console.log(b); //we know that we can access even before its initialization.- in other languages we cannot access b - without declaring and initialization.

//but in js, the memory for these variables and functions will get assigned even before executing a single line of code.

// so by accessing b-- we get undefined here.this is known as wt hoisting

let a = 10;

var b = 20;

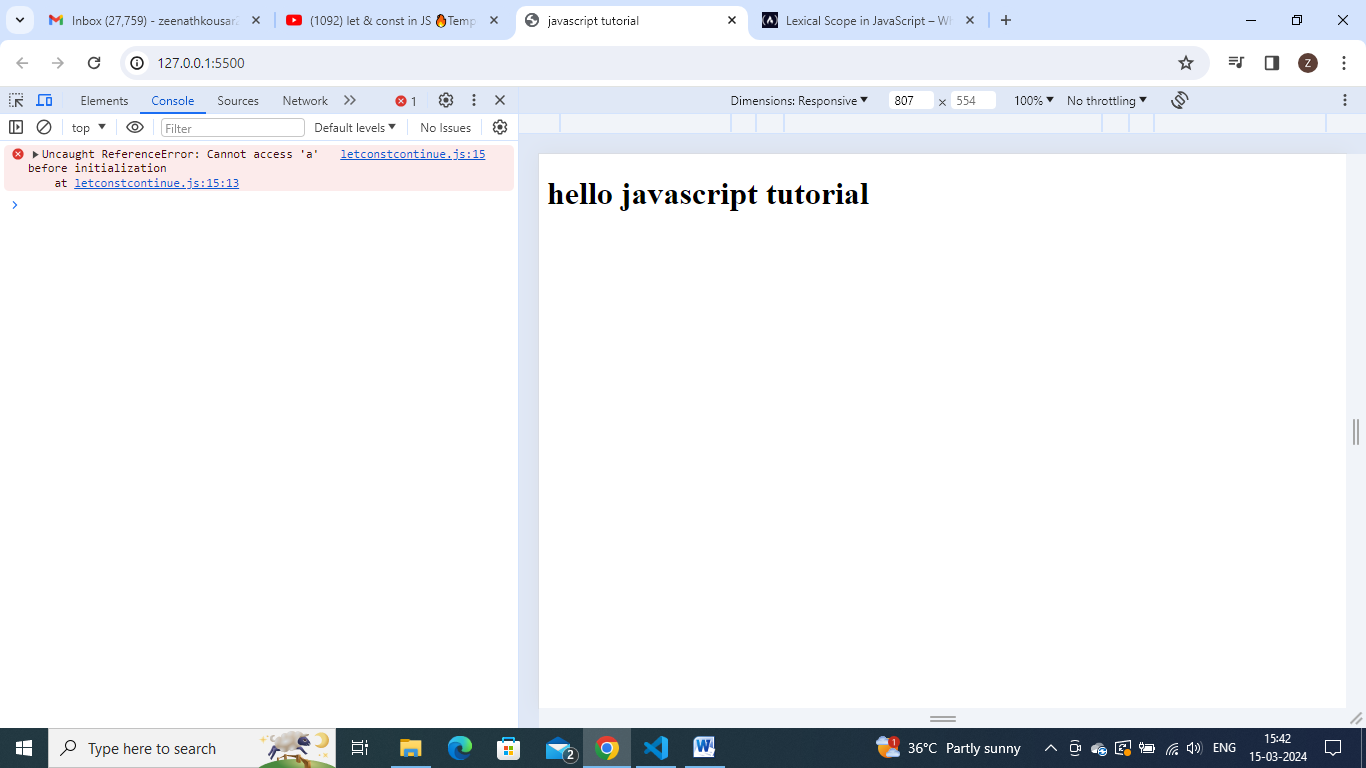
//hositing in let

// but as a variable (let) is also getting hosited. try to access it before its initialization.

console.log(a); //but in case of let- we get err- cannot access 'a' before initialization

let a = 10;

var b = 20;



//thhis err tell us that we can access that variable'a' only after it gets initialize/assiogned anyn  value to it.

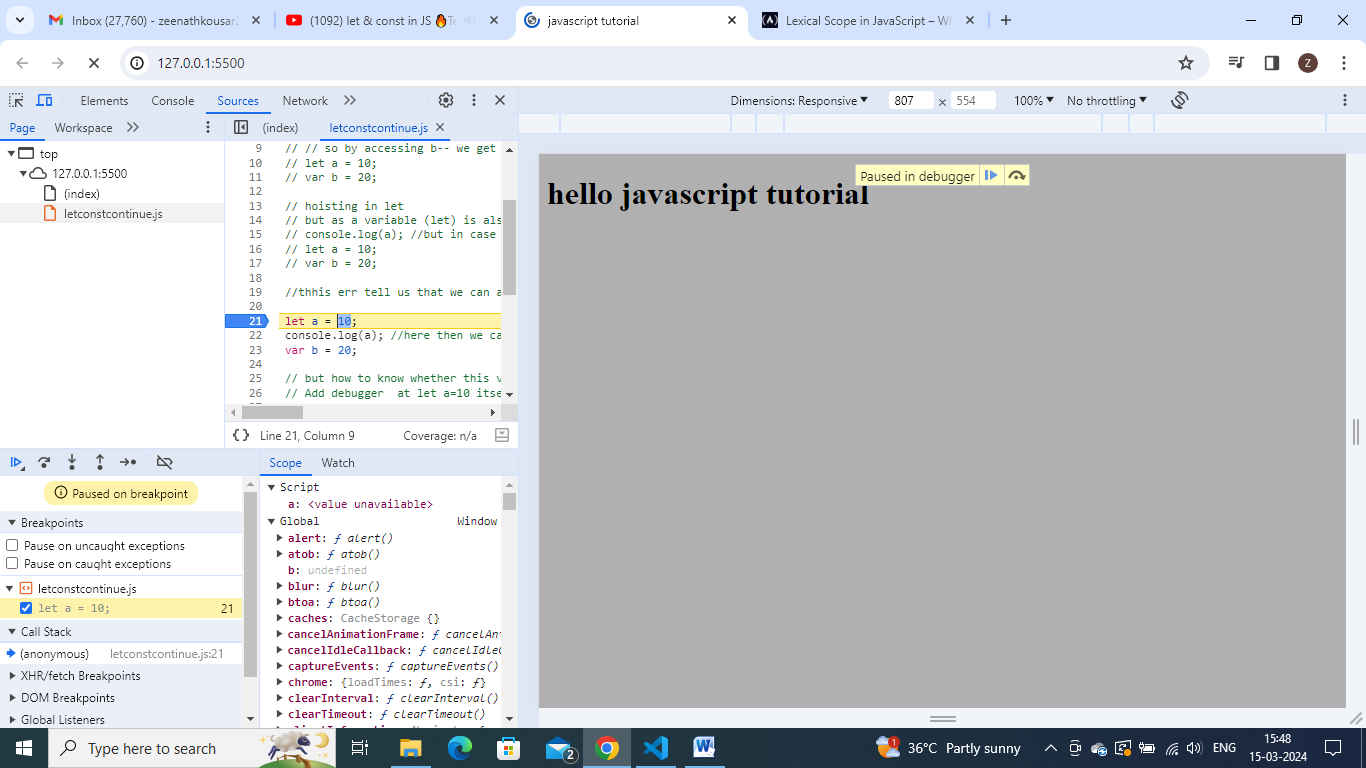
let a = 10;

console.log(a) //here then we can access it - get output as 10

var b = 20;

but how to know whether this variable is getting hoisted or not.

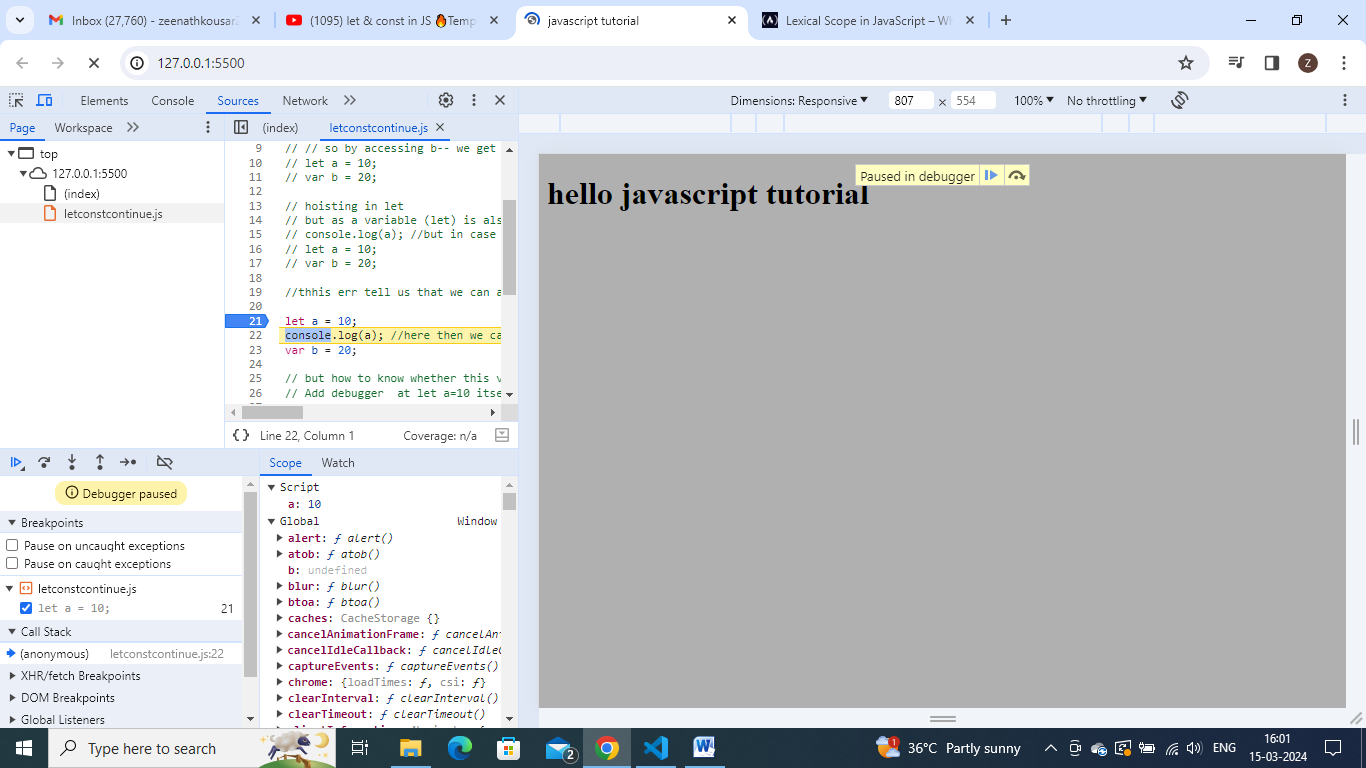
Add debugger at let a=10 itself in above code and check.



We seen that even before executing a single line of code, we observe that js has allocated a memory to a.so we have this a in scope (script) and we have b within scope(global).but values for them is undefined.in case of var variabke (b) it is in global scope but let variable ‘a’ is in script space(some other space).

So var variable gets initialized with undefined and got attached with global object, but in case of let and const they are also allocated memory.- that is nothing but hoisting, but they are stored in a different mem space than global.so we cannot access these let and const values before putting some value into it.

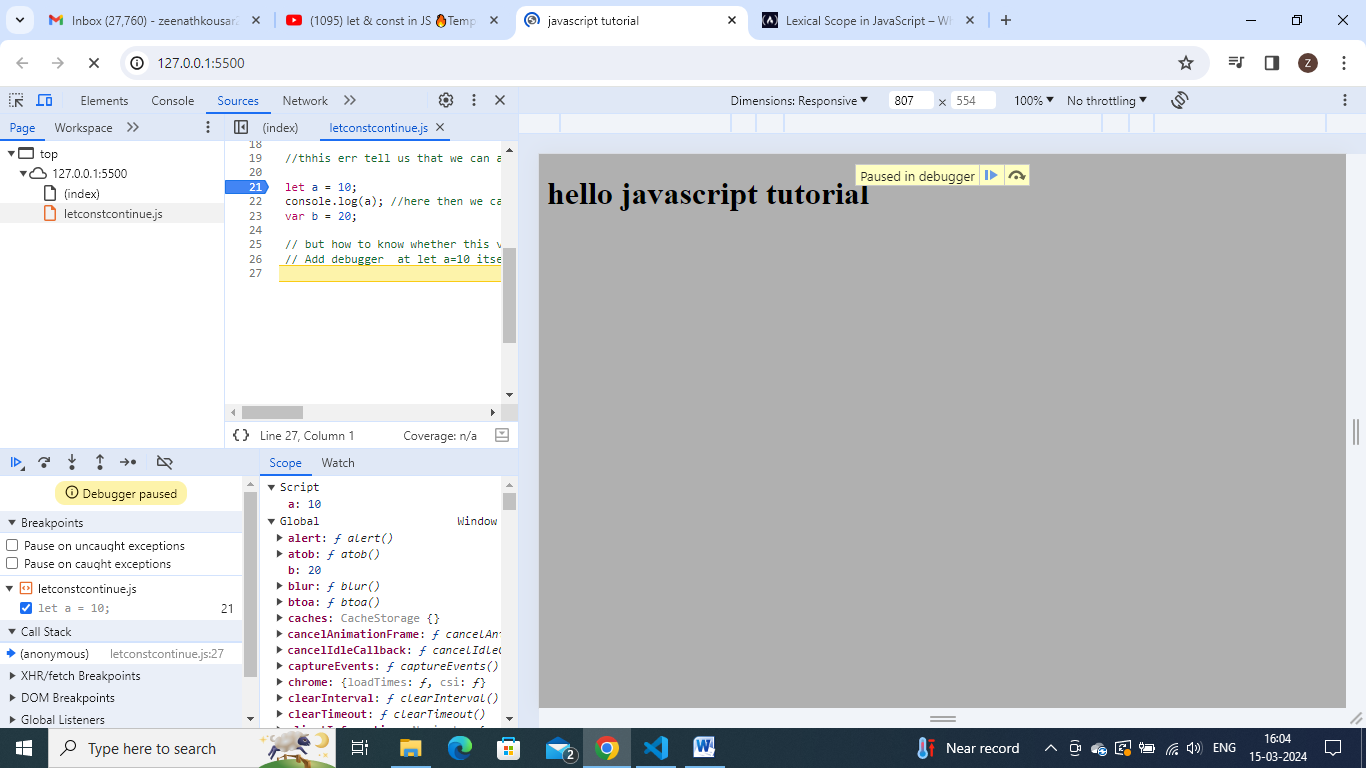
Move to next line in debugger and check.



See now u get value 10 to identifier ‘a’. but it is in separate mem space now it is available for access.

But see ‘b’ is undefined still

Go to next line in debugger – go to next line of var b=20 – then we fount b is 20.



**Temporal dead zone:** is the time since let variable was hoisted till it gets value.

console.log(a); //at this line a:undefined in separate script space, but not initialized .this phase is known as temporal dead zone.

let a = 10; //hree, temporal dead zone ends.here at this line  a is getting initialized - so till this time from console.log(a) , it is known as temporal dead zone

var b = 20;

//so time from hoising of let/const variable to till it gets assigned some value is known as hoising.

// //so time from hoising of let/const variable to till it gets assigned some value is known as hoising.

// so whenever u try to access let/const variable in temporal dead zone, it will give reference error.

//reference err- explanation in depth.

//reference err- explanation in depth.

// doing printing of a variable which is not exisiting anywhere in our pgm - do console.log(x)- see wt erru get

console.log(x);

let a = 10;

var b = 100;

//we get err as x is not defined - so when js tries to find this x valuein current scope - it is not found, no reference anywhere so.

// but do console.log(b) - we get undefined bcoz memory is allocated to b but it is not yet initialized

// but in case of let(prinitng a )/- it throws a ref err

Window obj(global obj):  
//window obj (global obj) - relation of global obj with var,let,const

//window obj is same global obj which we seen while debugging

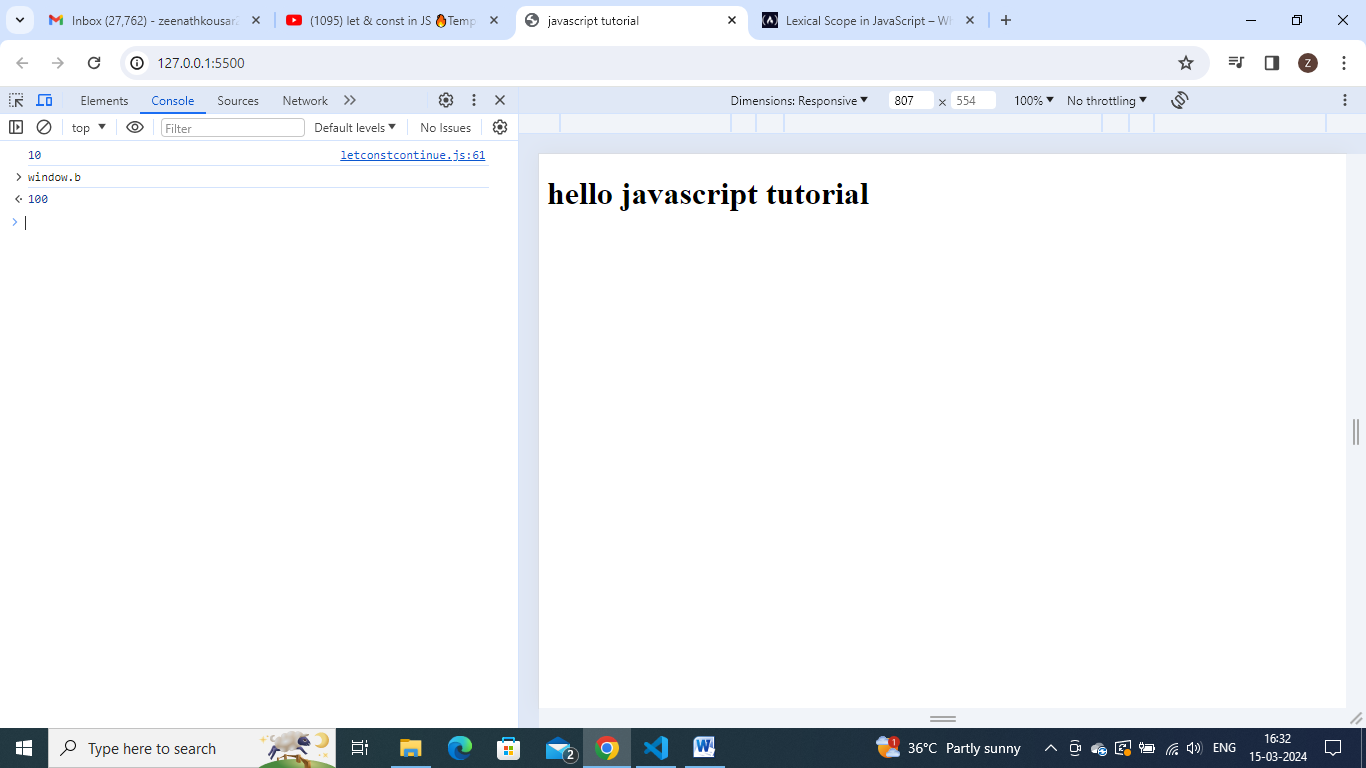
//we observed in debugging that var declarations were get attached with this window obj

let a = 10;

console.log(a);

var b = 100;

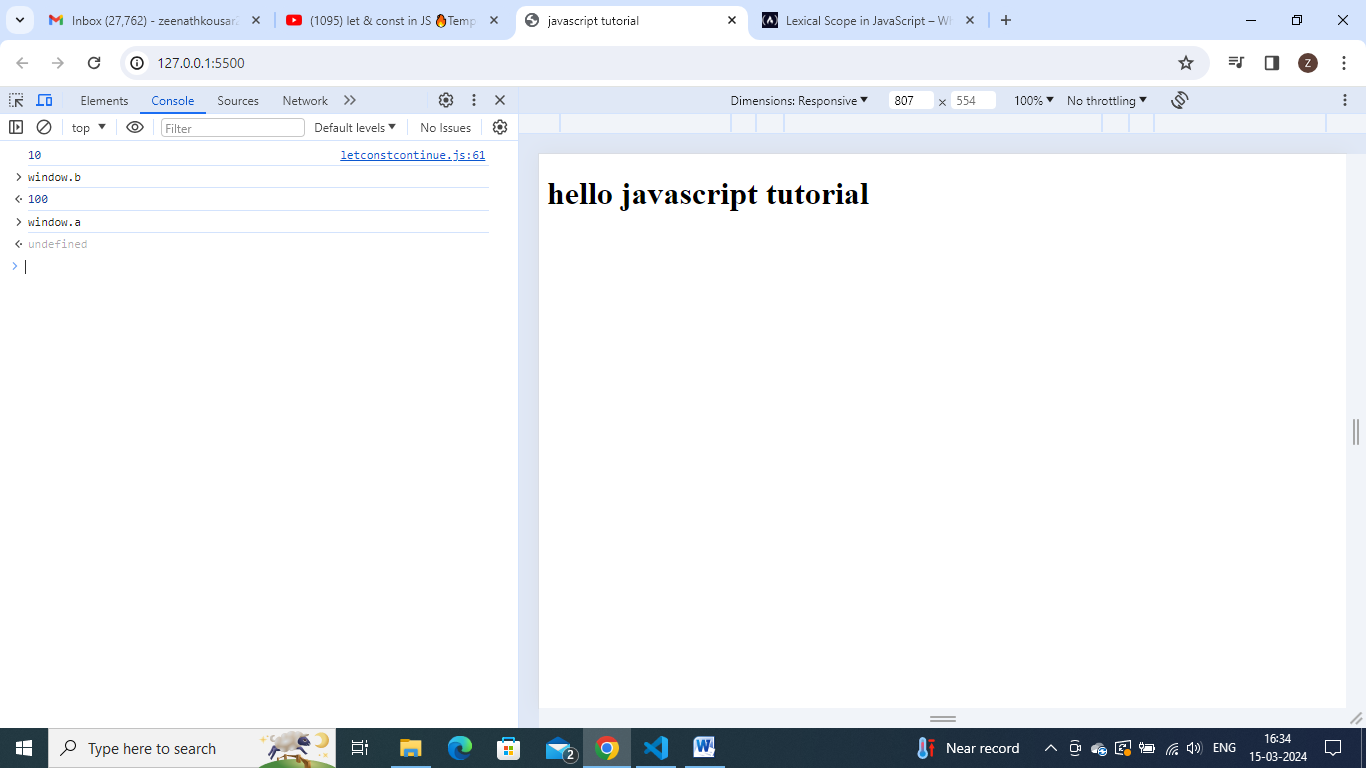
//so if i try to access window.b  - we get 100



//so if i try to access window.b  - we get 100

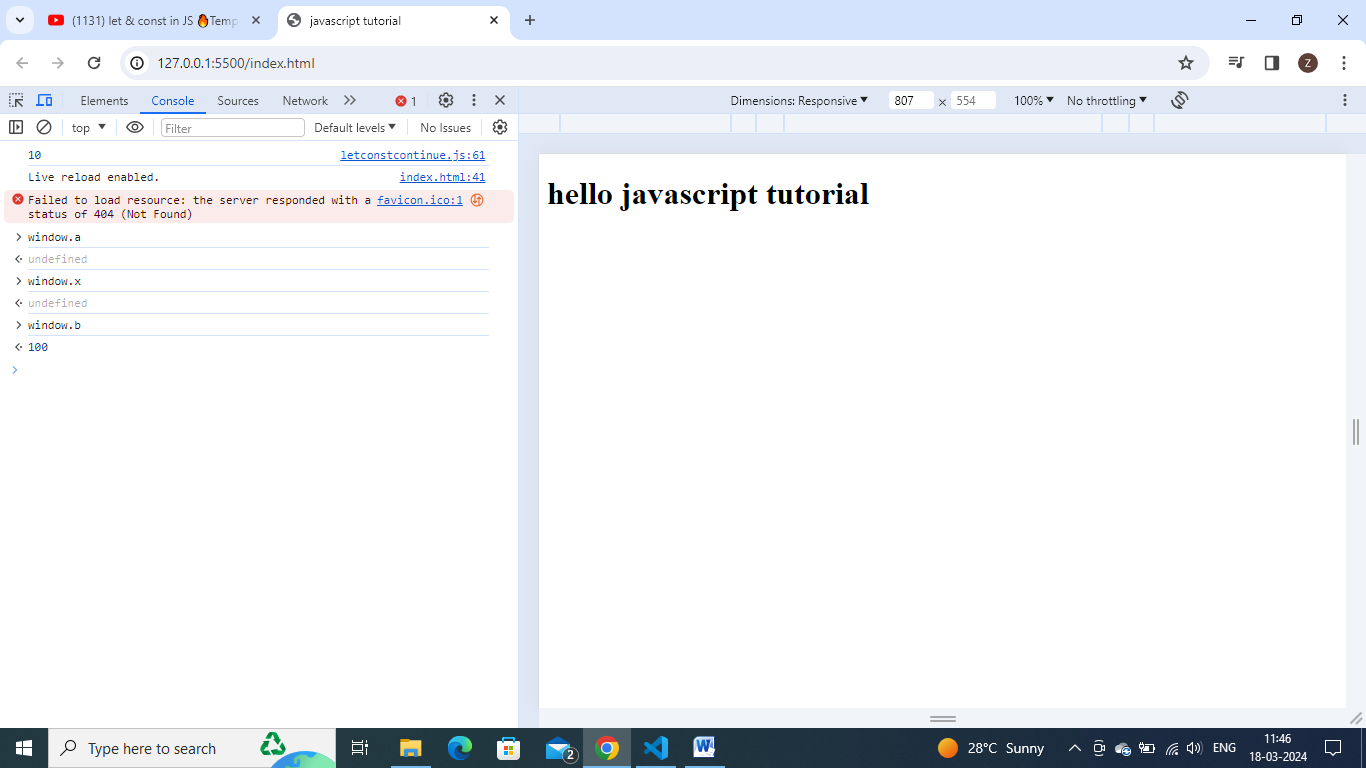
//but in case of a- as a is not getting allocated in global space, so it will not get attached to window obj.they gets allocated in separate space(reserverd for let and const variables).

//so if i do window.a- we get undefined



So here ‘a’ is behaving same as a variable which is not present

Eg: do window.x - as x variable is not present anywhere in the whole pgm.



And at global level ,this will refer to global object.

So this.b- we can access b as 100 value

But doing this.a- get undefined.

//so its not present in window obj

//window – means global obj, in browsers – global obj means window

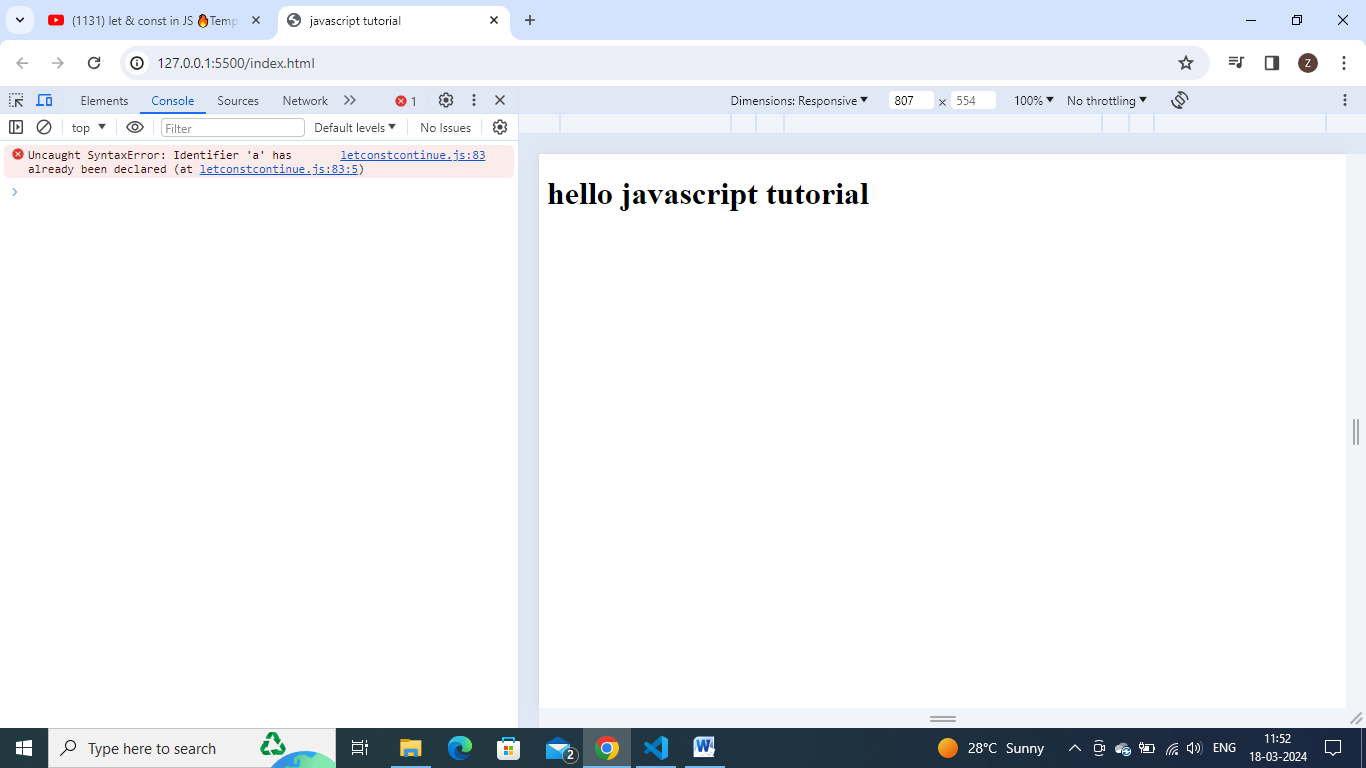
We notice that let is more strict then var.

There is one more strictness of redeclaration in let,

let a = 10;

let a = 100;

//it will throw an err- as  identifier 'a' has already been declared.



//it will throw an err- as  identifier 'a' has already been declared. this time it is a syntax err- no code will be run -> it wont execute singlee line of code , if it sees  redeclaration /duplicate declaration of let- it wont even console.log(a) it,

// eg:

console.log("kjchc") //it doesnt even execute this line

let a = 10;

let a = 100;

//it directly says its not possible, its completely syntax error.

//try to redecalre let variable using var.- we get same syntax error

console.log("kjchc"); //it doesnt even execute this line

let a = 10;

var a = 100;

//u cannot use same variable in same scope again

Remember we can do this in case of var.

// Remember we can do this in case of var. - it doesnot give any err

console.log("kjchc");

let a = 10;

var b = 100;

var b=1000

//lets see in case of const - it behaves very much similar to this let,it is even more strict than let. it behaves same in hositing like storing in separate mem space, we cannot access.it also goes to temporal dead zone .

// let a = 10;

// const b = 1000;

//add debugger at let  let a=10 line. - a and b both get stored in diff mem script and get assigned with undefiend..

//but in wt case const is more strict than let ? - see in case of let - u can just declare variable without initialization . u can do initialization after wards anywhere in the pgm

let a; //valid

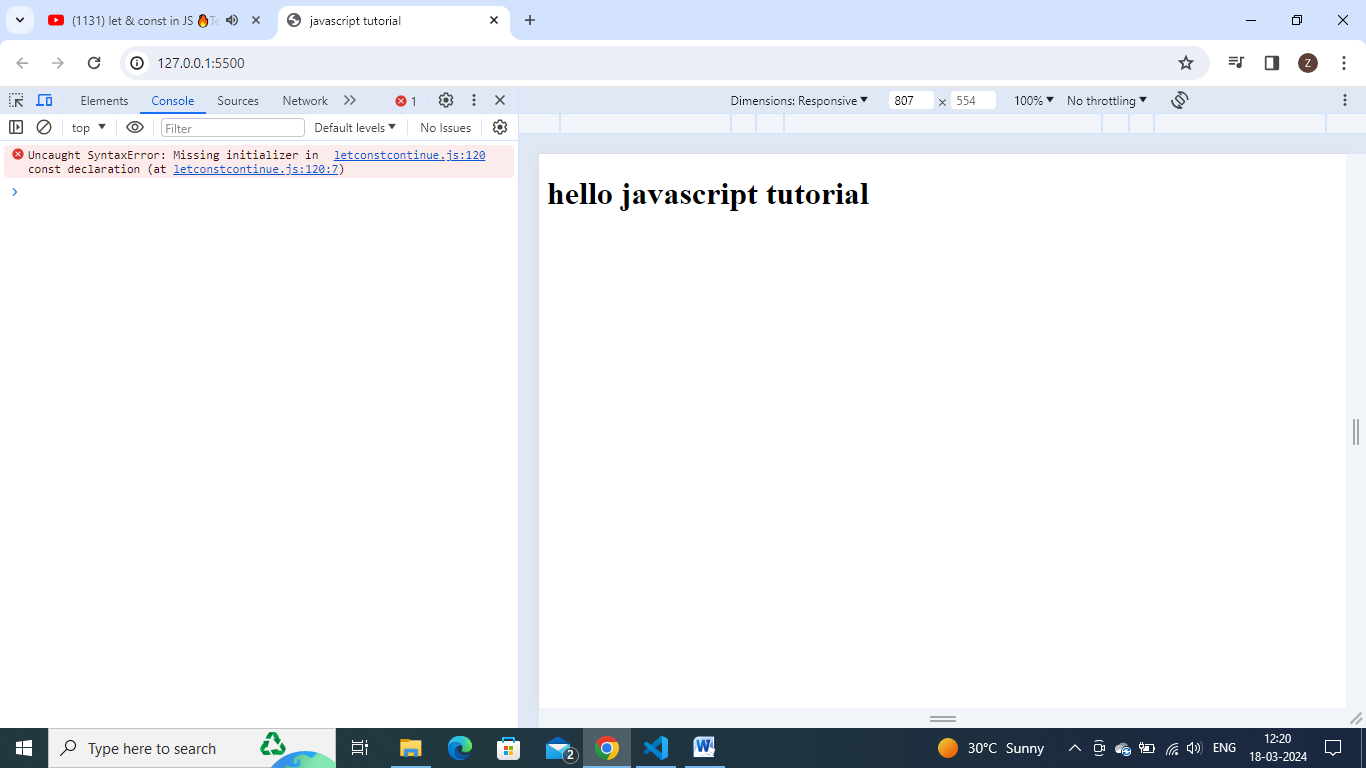
a = 10; //valid

const b = 1000; //valid

console.log(a); //we dont get even this console.log as the pgm has syntax err - it wont execute a single line

const c; //invlaid- err  syntax err- missing initializer in const declaration

c=90;

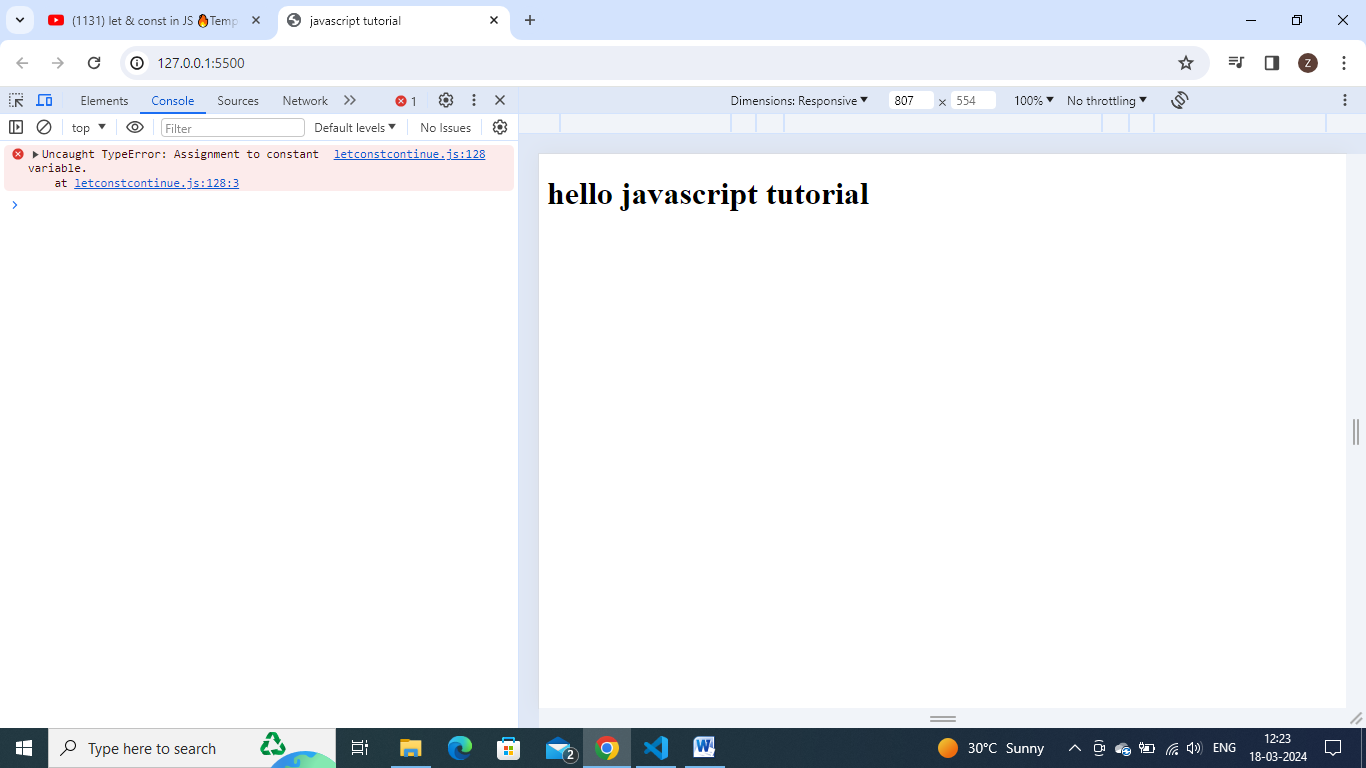


//so const varialbes should be initialize at time of  declaration only

**Type error in const:**

const b=1000;

 b=10000; //Type err-assignment to constant variable.



**Differenece between syntax err vs type err vs reference err:**

Type err- assignment to constant variable- means we are trying to assign some other value to a const variable which already initliazed. Y it is a type error- as const is a constant datatype – we are violating its rule of datatype.

Syntax- err- if we do only declaration and no initialization- we get syntax err- missing initializer in const declaration – so it is violating const syntax of declaration. Const b;

Reference err- when js engine tries to find out a specific variable inside a memory space ,and u cannot access it , then we get ref err

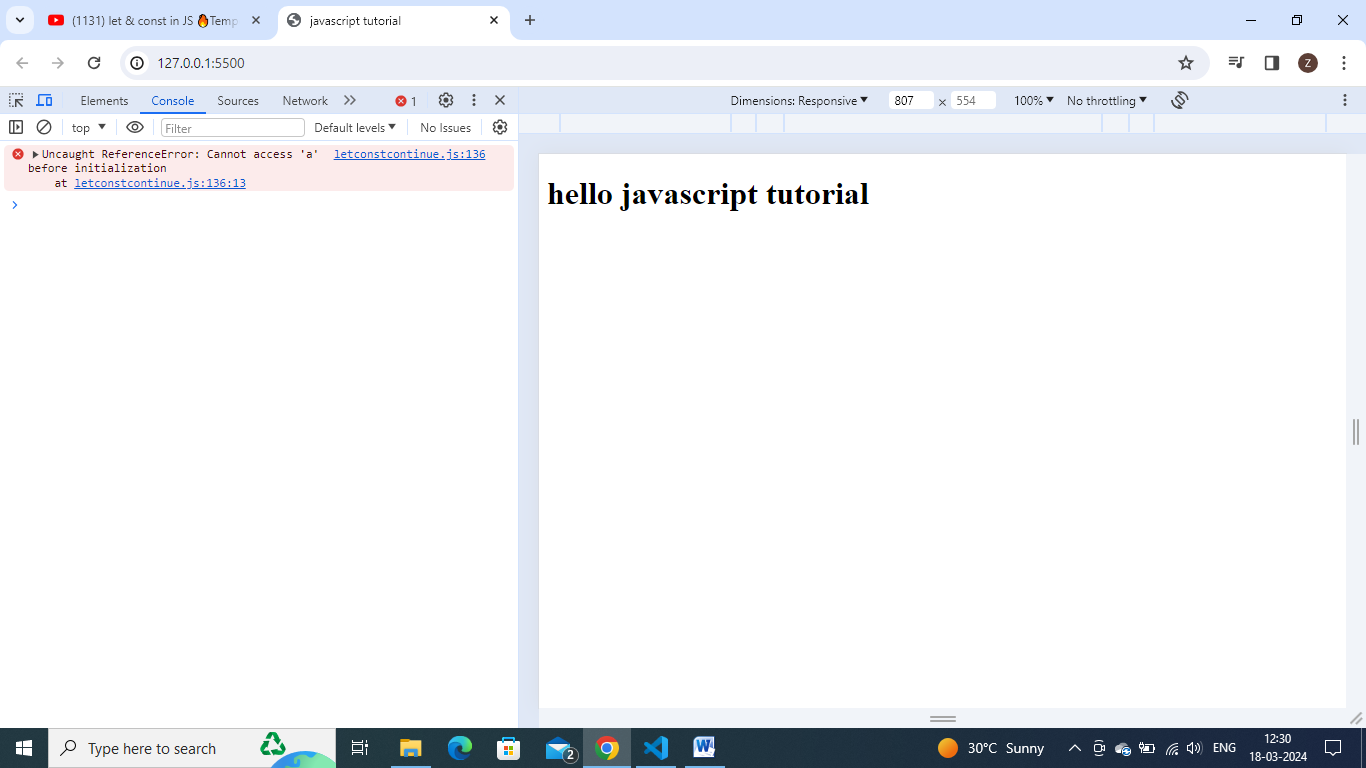
Eg: suppose I try to access let variable ‘a’ before its initialization then we get reference err

console.log(a)

let a=1990;

console.log(a) //here we get reference err- cannot access 'a' before initialization . here a was in the temporal dead zone

let a=1990;



// eg: one more example of reference err -

console.log(y); //consoling a variable which is not defined   , gets ref err as y is not defined.  as y is never placed in memory- u cannot access y

let a = 10;

receommended to use const and let depending on requirement to not get unnecessary errors.

To avoi this temporal dead zone errors- its always recommended to keep all ur declarations and initializations tokeep on top of scope.so as soon as ur code starts run, it hits the initialization part first. Then u get to logic. – this is also known as we are shrinking the temporal deadzone window to zero.