Let take a eg of function.

//created a func with 1000 milli sec=>i sec timer

function x() {

  var i = 1;

  setTimeout(() => {

    console.log(i);

  }, 1000);

}

x();

//it pritns value of i after one sec.

function x() {

  var i = 1;

  setTimeout(() => {

    console.log(i);

  }, 1000);

  console.log("namaste js");

}

x();

//the above code will first prints 'namaste js' then after 1 sec  and print i

How and y it is printing like this means bcoz, this function passed to settimeout forms a closure , it remembers a reference to I and forms a closure.

Wherever this fucn goes , it takes the ref of I along with it.

Settimeout will takes this callback func and stores it in some place. And attach a timer to it.

And js will proceeds with next lines.

Once the timer expires (1000 ms completed) then it takes the func and runs by putting it in callstack.

So js doesn’t wait any time .

**Situation :**

Suppose printing 1,2,3,4,5 after each and every sec.

Most developers do it using for loop as below.

function x() {

  for (var i = 1; i <= 5; i++) {

    setTimeout(() => {

      console.log(i);

    }, i \* 1000);

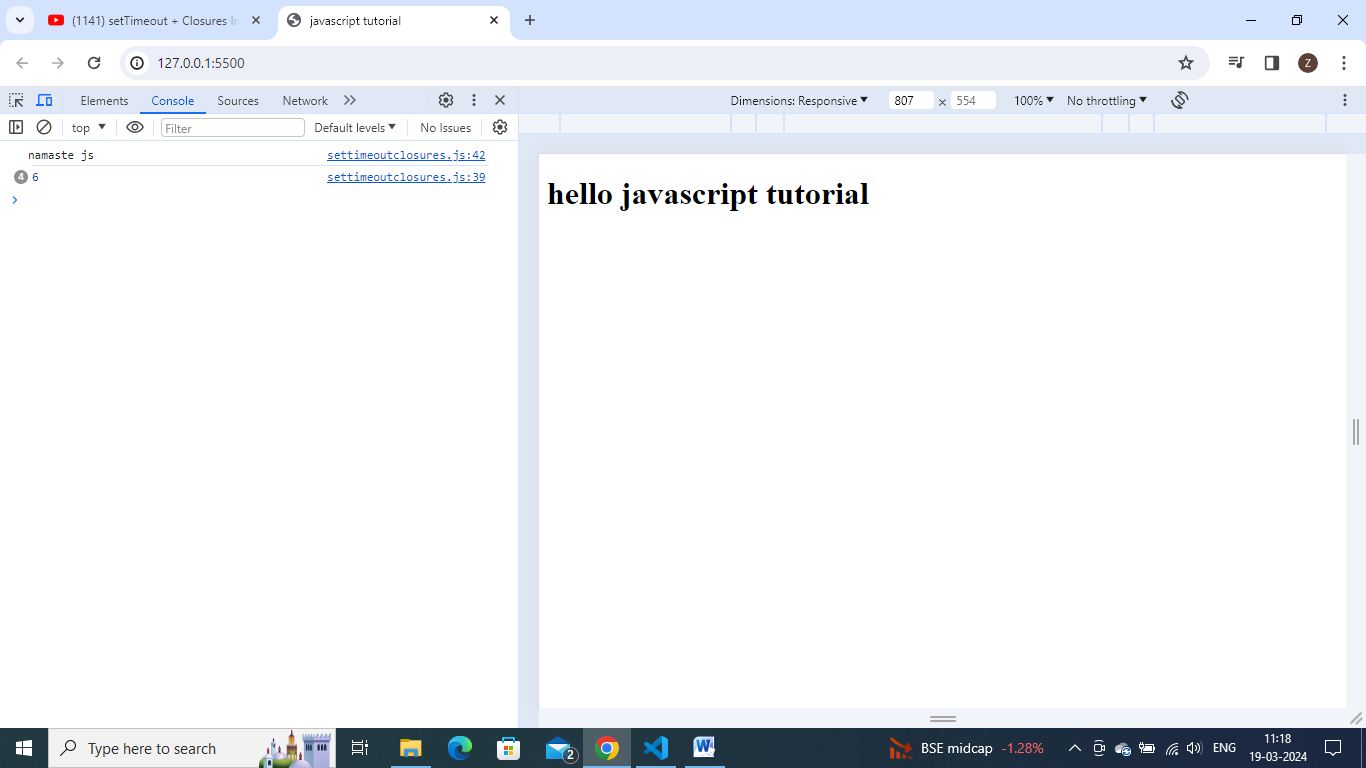
  }

  console.log("namaste js");

}

x();

unfortunately it giving different result,,it is printing continuously 6,6,6,6 after every second y?



It is working this way bcoz of closure.if u understand closure properly then only u understand y it is printing like this 6 here.

Closure is a function along with its lexical environment.

Even when function is taken out from its original scope, still it remember its lexical scope.

When the settimeout takes this function and stores it somewhere,so the function remember the reference to it. It remember reference to I, not value of i.

When loop runs the first time,it make a copy of a function, attaches a timer and also remembers the reference of i.similarly these 5 copy of functions,all of them are pointing to same reference of I, bcoz the environment for all of these are same.

All copy of settimeout callback functions , will refer to same i.

Js doesnot wait for anthing,js will store these all 5 functions and move on , loop doesn’t wait for timer to expire.so before the timer expires only, full loop got run, and value became 6.

So when this callback func runs, till then I value became 6.

Bcoz all of these 5 functions referring same I .

To fix this, we can use let to declare I value.

function x() {

  for (let i = 1; i <= 5; i++) {//used let

    setTimeout(() => {

      console.log(i);

    }, i \* 1000);

  }

  console.log("namaste js");

}

x();

Let has a block scope,so for each and every loop iteration,this I will be a new variable all together.it is a neww copy of i.each time settimeout is run,this callback function has a new copy of I with it.

As let variables are block scope , each and every time, this loop is called,settimeout function is called, and this functions forms a closure with new variable itself.copy of I is new in each iteration.

I++ =>I becomes 2 will be a new variable of closure , and settimeout takes this function and new value of I copy-2 - binds it and save it.

Lly, I =3 save a copy. Like this save 5 copies with new variables.

In simple terms, each and every time this func is called,it is referring to different memory location, which is there individual copy of i.

Let is block scope, is creating new copy of function each time , when this loop is executed.

Q. suppose interviwer ask u to do it by using var variable only.

Here also u can use closures.it is not working with var bcoz this copy of I,refer to same mem location, so somehow , we have to give new copy of I , every time to settimeout and form a closure with it.

So to fix it , by using var only here,we can form a closure , create a new function and enclose this whole settimeout into it.

So we want new copy of I, each and every time this loop is executed.

So wrapping this setttimeout inside another function and some how we have to provide new copy of I.

Try to call that function by passing parameter as I inside loop . – now see it works.

And passing I as a parameter in func definition also.

function x() {

  for (var i = 1; i <= 5; i++) {

    function close(i) {

      setTimeout(() => {

        console.log(i);

      }, i \* 1000);

    }

    close(i);

  }

  console.log("namaste js");

}

x();

the above code is now printing properly, 1,2,3,4,5 after every sec.

bcoz everytime u call this close function with value of I, it creates a new copy of I for itself over here.

As there are multiple I variables in above code, we can differentiate and write like this also.

function x() {

  for (var i = 1; i <= 5; i++) {

    function close(y) {

      setTimeout(() => {

        console.log(y);

      }, y \* 1000);

    }

    close(i);

  }

  console.log("namaste js");

}

x();

here, actually we recognized the problem that it isreferring to same memory space and now by using closures , this close function is created a new copyof x , everytime the settimeout was called.everytime, settimeout func will be stored in a separate mem and attach the timer and it stores and remember a new copy of x. so everytime close () func is called, will have a new copy of i.

js is loosely typed lang - Since JavaScript is a loosely typed language, you are not required to correctly predict the kind of data that will be kept in a variable. Depending on the information you supply to a variable (such as this ' ' or " " to signify string values), JavaScript will automatically type it.12 Dec 2022