AlgoTutor

JavaScript

Most Important Interview Questions (2024)

+91-7260058093
www.algotutor.io

1. What Features Have Been Introduced In The ES6 Version?d Time

- Let And Const Keywords.
- Arrow Functions.
- Multi-Line Strings.
- The Destructuring Assignment.
- Enhanced Object Literals.
- Promises.

2. What Is The Main Difference Between Var, Const, And Let?

- Variables Declared With Let And Const Are Block-Scoped; Variables Declared With Var Are Globally-Scoped Or Function-Scoped.
- Var Variables Can Be Updated And Re-Declared Within Its Scope; Let Variables Can Be Updated But Not Re-Declared; Const Variables Can Neither Be Updated Nor Re-Declared.
- Var Can Be Hoisted To The Top Of Their Scope. Where Var Variables Are Initialised As Undefined, Let And Const Variables Are Not Initialised (Temporary Dead Zone, TDZ).
- While Var And Let Can Be Declared Without Being Initialised, Const Must Be Initialised During Declaration.

3. What Are Promises And Async-Await?

- Promises Is A Way To Enable Asynchronous Programming In JavaScript. In More General Termis, Promise Means That A Program Calls A Function With The Expectation That It Will Return The Result That The Calling Program Can Use In Further Computation.
- Async-Await Also Helps In Asynchronous Programming. It Is Syntactic Sugar For Promises. Async-Await Has A Simple Syntax And Is Easy To Maintain Lots Of Asynchronous Calls In A Single Function. Also, Async-Wait Prevents Callback Hell.

4. Why Is JavaScript Single-Threaded?

JavaScript Is A Single-Threaded Language Because While Running Code On A Single Thread, It Can Be Really Easy To Implement As We Don't Have To Deal With The Complicated Scenarios That Arise In The Multi-Threaded Environment Like A Deadlock.

5. How Does JavaScript Maintain Concurrency?

- Event Loop.
- Micro & Macro Queue.
- CallBack.
- Thread Pool & Clustering (Multi-Threading).

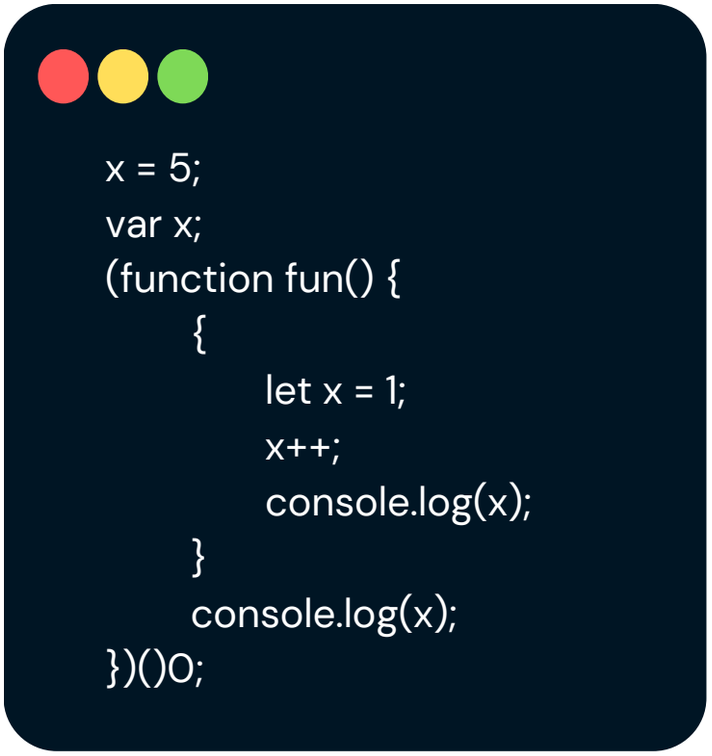
6. What Is A Callback And How Does It Work Behind The Scene?

- The Callback Is Possible Only Because JavaScript Supports The First-Class Function.
- A Function That Has Passed As An Argument To Another Function Or It Can Be Executed In That Other Function Is Called A Callback.
- In Node.Js, It Consists Of 4 Default Threads That Are Responsible For Maintaining The Main Stack And Other Queues, Most Asynchronous Functions Call Other Asynchronous Functions And Then Call The Callback. You Can Think Of It As A Chain Of Functions And Callbacks. All Asynchronous And 10 Operations Are Directly Not Handled By The Main Thread, All Callbacks And Asynchronous Calls Have Been Handled By The Other Queue Which Presents In The JS Engine.

7. How Many Ways Do We Have For Declaring A Function And How Are They Different From Each Other?

- A Function Declaration Has Made Of A Function Keyword, Followed By An Obligatory Function Name, A List Of Parameters In A Pair Of Parenthesis.
- Shorthand Method Definition Can Be Possible To Use In A Method Declaration On Object Literals And ES2015 Classes.
- An Arrow Function Is Defined Using A Pair Of Parenthesis That Contains The List Of Parameters. Followed By A Fat Arrow => And A Pair Of Curly Braces That Delimits The Body Statements.
- In A Function Expression, You Assign A Function To A Variable.
- A Function Can Be Dynamically Created Using The Function Constructor, But It Suffers From Security And Performance Issues And Is Not Advisable To Use.

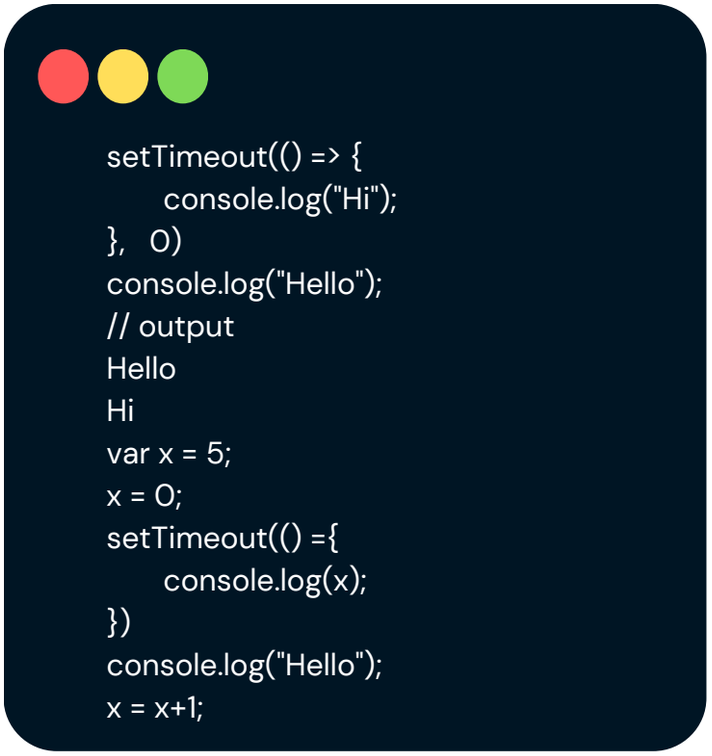
8. What Will Be The Output Of The Code Below?



```
x = 5;
var x;
(function fun() {
        {
                let x = 1;
                x++;
                console.log(x);
        }
        console.log(x);
})()0;
```

```
// output
2
5
```

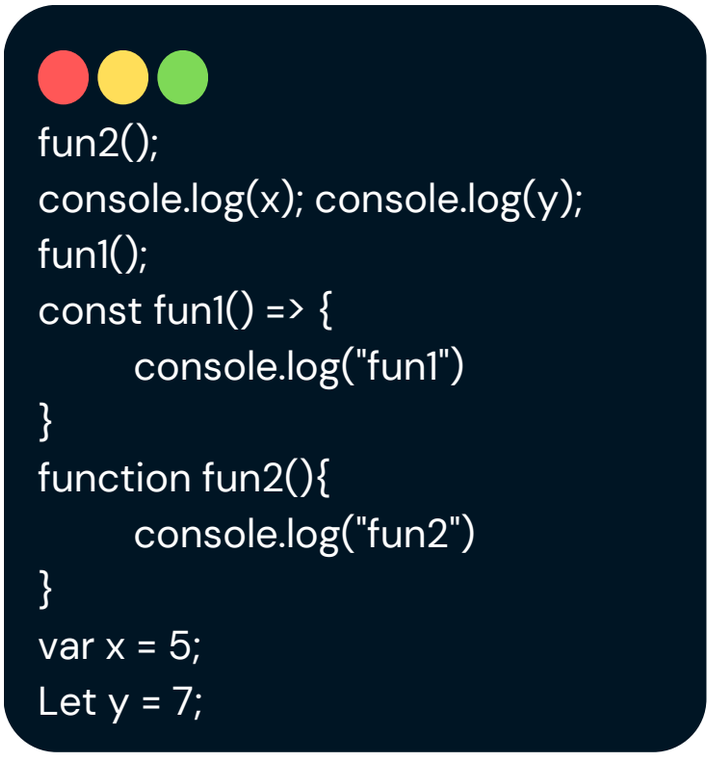
9. What Will Be The Output Of The Code Below?



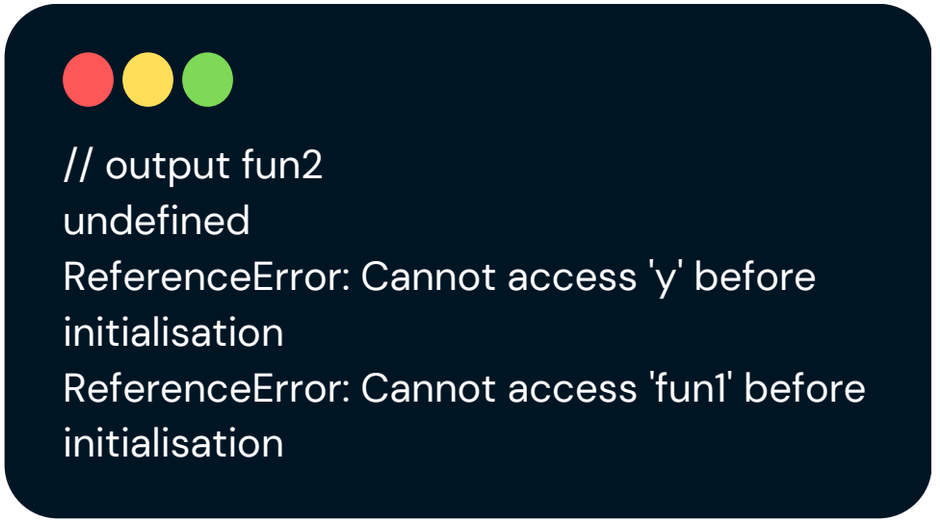
```
setTimeout(() => {
      console.log("Hi");
},   0)
console.log("Hello");
// output
Hello
Hi
var x = 5;
x = 0;
setTimeout(() ={
      console.log(x);
})
console.log("Hello");
x = x+1;
```

```
// output
Hello
1
```

10. What Will Be The Output Of The Code Below?



```
fun2();
console.log(x); console.log(y);
fun1();
const fun1() => {
        console.log("fun1")
}
function fun2(){
        console.log("fun2")
}
var x = 5;
Let y = 7;
```



```
// output fun2
undefined
ReferenceError: Cannot access 'y' before
initialisation
ReferenceError: Cannot access 'fun1' before
initialisation
```