ES6 Map

Why we use Map if Object is there ?

1. Map remembers the order for insertion and maintain that order unlike in object
2. Map is iterable while objects are not. In objects w have to use something like Object.keys() or Object.values()
3. There are several inbuild method in Map which comes handy
4. We can find length of Map easily while we have to find it manually in object

New features in ES6

* Let and Const keywords
* Arrow functions
* Multi-line strings
* Destructing assignment
* Enhanced Object Literals
* Promises

Promises and async-wait

* Its as way to enable asynchronous programming in JavaScript
* 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?**

* Documentation, why JavaScript is a single-threaded language — [link](https://www.geeksforgeeks.org/why-javascript-is-a-single-thread-language-that-can-be-non-blocking/).

**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 IO 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);  
})();  
// output  
2  
5

**9. What will be the output of the code below and why?**

setTimeout(() => {  
 console.log("Hi");  
}, 0)  
console.log("Hello");// output  
Hello   
Hivar 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 and why?**

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