**Difference between call, apply & bind:**

The only difference between call & apply is call() method takes the arguments separated by comma while apply() method takes the array of arguments.

var obj = { num: 2 };

var addToThis = function(a) { return this.num + a; }

addToThis.call(obj, args)

addToThis.apply(obj, [args])

.bind allows you to set the "this" value now while allowing you to execute the function in the future, because it returns a new function object

addToThis.bind(obj, args)();

**Hoisting:**

It is JavaScript's default behavior of moving all declarations to the top of the current scope.

let or const are not hoisted!

JavaScript only hoists declarations, not initializations.

"use strict" does not allow hoisting.

var x = 10;

console.log(x, y); // x is 10, y is undefined

var y = 20;

y is undefined because only the declaration (var y), not the initialization (=7) is hoisted to the top.

function declarations are hoisted, while function expressions are not.

var funExp = function() {}

function funcDecl() {}

x = 1 is shorthand for window.x = 1

**Ways to create object in JS:**

1. var car = { model: 'BMW'}

2. function Car(model) { this.model = model; } var c1 = new Car('BMW');

3. Using ES6 class

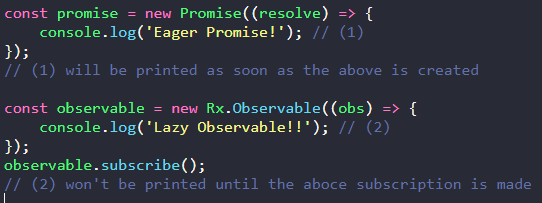
4. Using Object.create

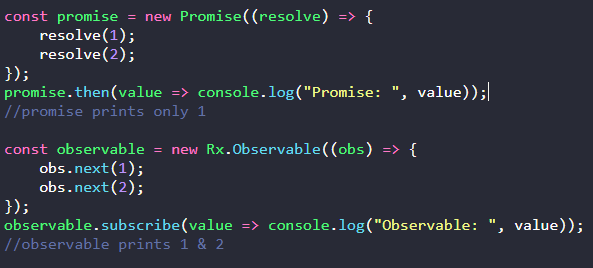
**Promise vs Observable:**

|  |  |  |
| --- | --- | --- |
| nature | Promise | Observable |
| Execution | Eager | Lazy |
| Emission of values | Single(first) | Many |
| Casting | Multicasting(same value for multiple subscribers) | Unicasting |
| Execution of handlers | async | Synchronous/async |
|  | Not cancellable | Cancellable (unsubscribe) |

.then() returns a new Promise while .subscribe() returns a new subscription (which has one method unsubscribe) but not an observable.

That’s why we can chain .then() calls but not .subscribe() calls





# 

**Async/Await:**

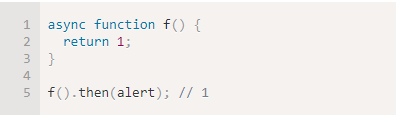
The word “async” before a function means one simple thing: a function always returns a promise. Other values are wrapped in a resolved promise automatically.

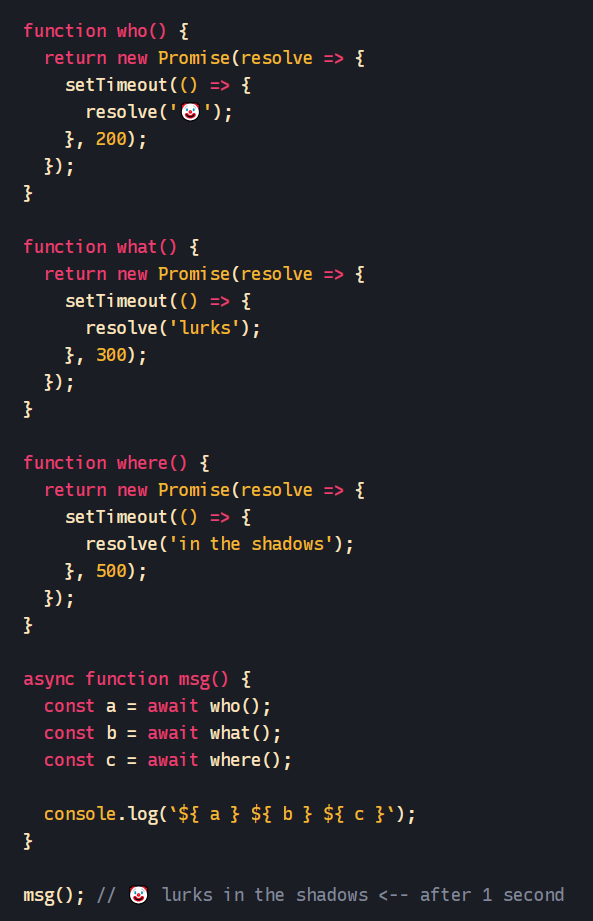
Async functions enable us to write promise based code as if it were synchronous, but without blocking the execution thread. It operates asynchronously via the event-loop.

The keyword await makes JavaScript wait until that promise settles and returns its result.

await can be used only in an async function

It has to be noted that await only makes the async function wait and not the whole program execution.





In the above example each step is done sequentially, with each additional step waiting for the step before to resolve or reject before continuing.

If you instead want the steps to happen in parallel, you can simply use Promise.all() to wait for all the promises to have fulfilled

The Promise.all() method returns a single Promise that resolves when all of the promises passed as an iterable have resolved or when the iterable contains no promises.

It rejects with the reason of the first promise that rejects.

