Introduction to Promises (Promises Lec_001) - Milind Mishra

Promises in JavaScript (important for implementations) "Async Programming"

↳ coding practices are so good → other programming languages use the pattern i.e. Java, Dart etc...
(uses promise-based code)

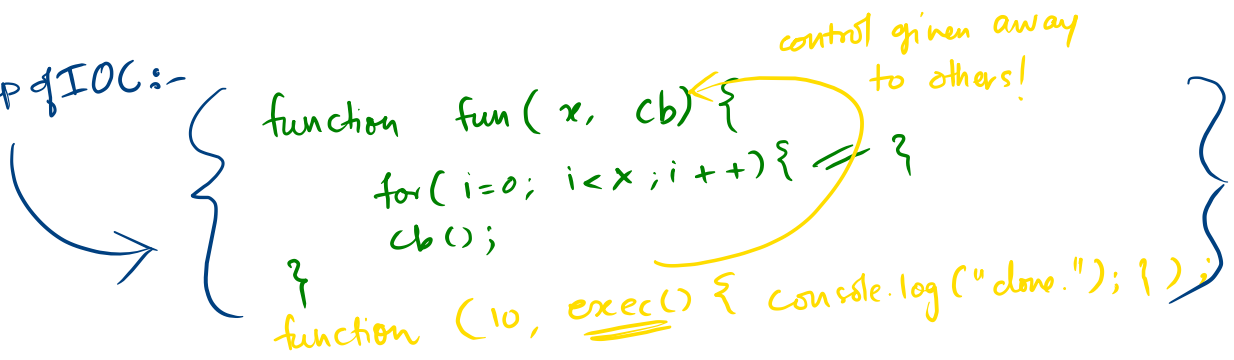
Problems w/ callbacks → inversion of control & "Callback-Hell"
↙ more concerned.
↙ code-readability issue

Promises are nothing but readability enhancers!

Promises
* Readability Enhancers ✓
* Solves Inversion of Control. ✓

↳ In JS, promises are special kind of objects that gets returned, immediately after we call them.

Recap of IOC:-



Promises acts as a placeholder (object) we hope to get back in future.

eg: fetch(url, ...) → downloads (time consuming task.)

↳ runtime feature not core JS.

↳ if "fetch" is written with promises then it will return a promise (acts as a placeholder for the result)

* In these promise objects we can attach the functionality we want to execute once the future task is done.

* once future task gets done, promises automatically attaches/runs the function.

But isn't the same w/ CBs?? technically YES! But IOC. Problem. which is why promises.

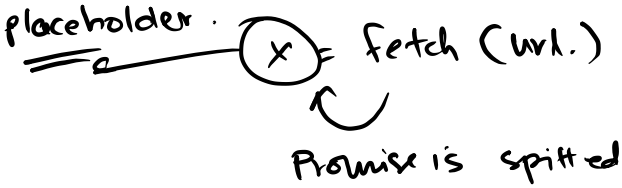
↓

Yes, setTimeout(exec, 1000).

CB. fetch:

```
fetch ( url, function exec() {
    console.log("done");
});
```

But promise based we can attach the f^n we want to be executed latter on can be attached latter on as well as we have the placeholder object in Promise based syntax.



x.then ~~ . . ← can be called & attached logic latter on...

✓ 1) How to create a Promise? → learn to make Promises for others to use!
✓ 2) —— consume a Promise? → Learn to consume Promises made by others!

{Promise} ✓ → can be kept/break. (2 situation)

"maybe fulfill" "maybe not fulfilled"

* How to create a Promise? → reference to Promise object in ECMA Docs → Native to JS.

{Creation of Promise nature} sync. in

official definition, says,

" A Promise is an object that is used as an "Placeholder" for ' eventual results of a deffered (and possibly asynchronous) computation.

Any promise object can be in 3 Mutually Exclusive States: fulfilled, rejected & pending.

3 States :- When we create a new Promise object → default state is PENDING.

1. PENDING : default state
2. FULFILLED : (transitions to when sucessfull)
3. REJECTED : (transitions to when rejected)

Creating a new Promise object :-

constructor → special function using which a new object can be made.

```
new Promise()
```

Keyword

↑ this constructor, expects a callback.

expects 2 parameters ...

```
new Promise(function (resolve, reject) {
    // inside the fn we can write time consuming tasks
    // ex: downloading something, big for loop! etc.. anything you want!
})
```

callback : executor fn → expects 2 params → resolve, reject. ✓

new Promise (f)

functions.

new Promise (resolve , reject)

If none gets called, stays @ PENDING

called when succesfull

The promise goes to FULFILLED state

value gets updated w/ argument of resolve fn

called when failed

REJECTED State

value gets updated w/ argument of reject fn

flow of promise :-

PENDING

FULFILLED

REJECTED

```
function getRandomInt(max) {
    return Math.floor(Math.random() * max);
}

function createPromiseWithLoop() {
    return new Promise(function executor(resolve, reject) {
        for(let i=0; i<1000000000; i++) { }
        let num = getRandomInt(10);
        if(num % 2 == 0) resolve(num);
        else reject(num);
        {
    });
}
```

If random no. is even we, FULFILL.

If random number is odd we, REJECT.

```
function getRandomInt(max) {
    return Math.floor(Math.random() * max);
}

function createPromiseWithLoop() {
    return new Promise(function executor(resolve, reject) {
        for(let i=0; i<1000000000; i++) { }
        let num = getRandomInt(10);
        if(num % 2 == 0) resolve(num);
        else reject(num);
        }
    });
}
```

```
let x = createPromiseWithLoop();
Console.log(x);
```

Promise {<fulfilled>: 8}
[[Prototype]]: Promise
[[PromiseState]]: "fulfilled"
[[PromiseResult]]: 8 ✓ value

forloop is a blocking pc. of code, makes the return wait for it...

```
function getRandomInt(max) {
    return Math.floor(Math.random() * max);
}

function CreatePromiseWithTimeout(){
    return new Promise(function executor(resolve, reject) {
        setTimeout(function() {
            let x = getRandomInt(10);
            if (x %2 ==0) resolve(x);
            else reject(x)
        }, 10000)
    }
}
```

Immediately returns a Promise, Non-Blocking pc. of code.

10 sec. timer Started.

after timer → resolve if even ✓
reject if odd ✓

NOTE: accepts only 1 arg. value.
for multiple pass array. or something.