Promises

Promises are special JS objects.

- how to create these objs
- how to consume
- properties

How promises work behind the scene ??

The promise object we create has 4 major properties

1) status / State
2) value
3) onfulfillment
4) onReject

* Status/ → status shows current promise status
state

1) pending state
2) fulfilled State → Success
3) rejected State. → error

* value → when status of the promise is pending, this value property is undefined. The moment promise is resolved status → fulfilled the value property is updated from undefined to the new value (this value we can consider as the returned value) resolved value)

So the value property acts like a placeholder till the time promise finishes.

* onfulfillment → This is an array, which contains functions that we attach to our promise object. (To a promise object we can attach some func^n using .then() method). When the value property is updated from undefined, to the new value, JS gives chance to those attached func^n one by one with the value property as their argument (if there is no piece of code in the call stack & global code left)

status

value

onfullfilment :

[f, g, h, i]

```
for(i=0; i<10^10; i++)
{
```

```
return new Promise (function(resolve, reject) {

})
```

Promise constructor

this constructor takes callback as argument

```
new Promise ( function ( resolve , reject ) {

        //write here

})
```

→ To create a promise call the promise constructor.

→ the promise constructor takes a callback as an argument.

→ the callback passed inside constructor, expects 2 arguments resolve , reject → func7

→ then inside the callback write your logic

→ if you want to return something on success, then call resolve func^n^ with whatever value you want to return.

Q» When do we consider a promise fulfilled ??

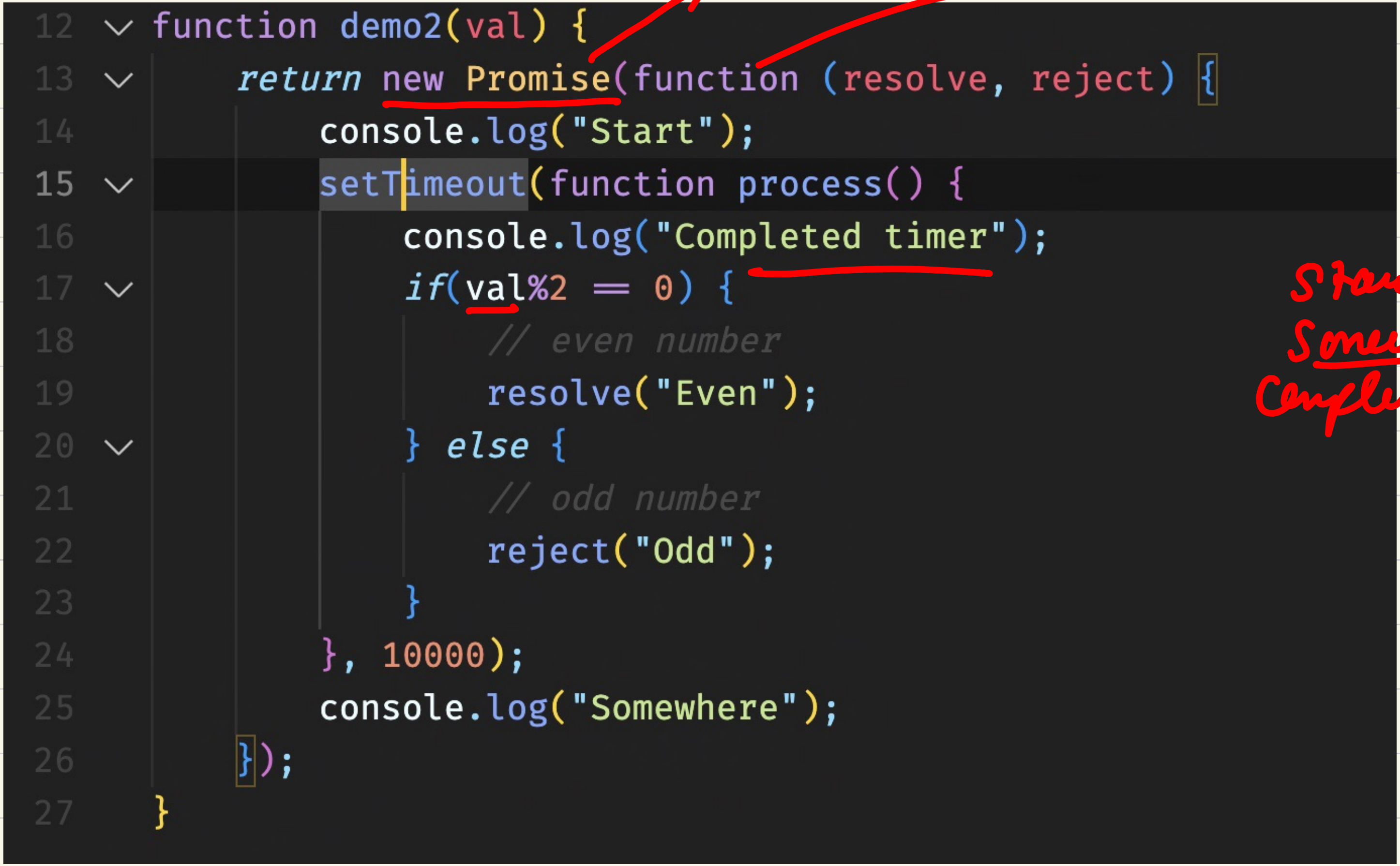
→ if we call resolve ( ) funcⁿ, we consider it fulfilled.

→ we consider it rejected if we call reject ( ) ;

# Creation of a promise obj is sync.

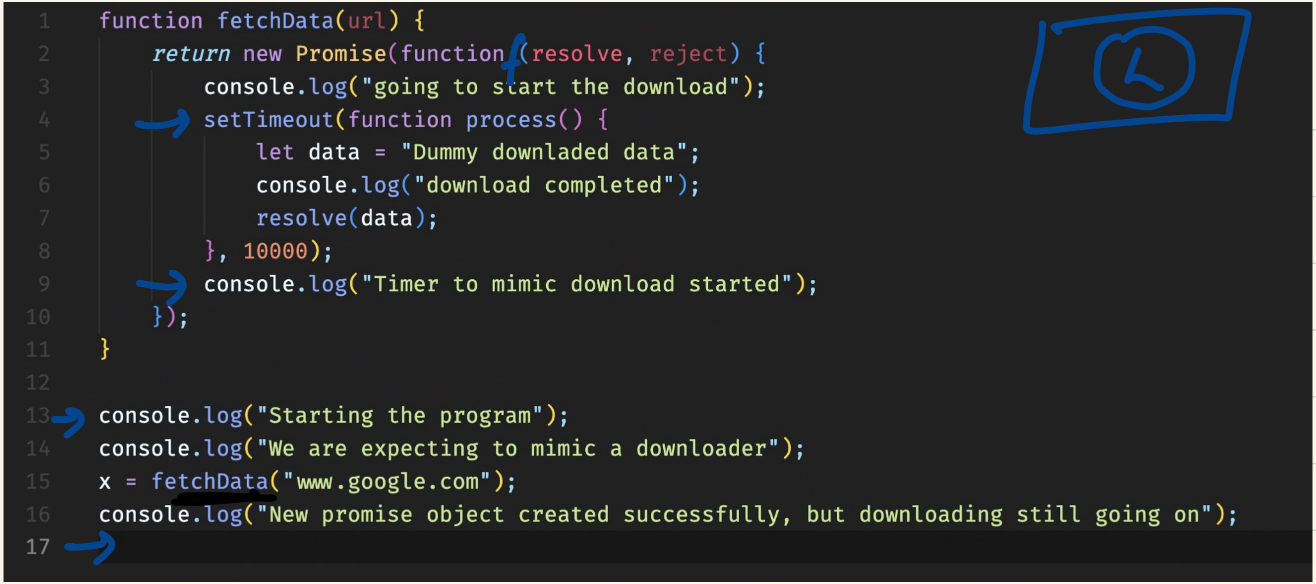
constructor

callback to constructor



```javascript
function demo2(val) {
    return new Promise(function (resolve, reject) {
        console.log("Start");
        setTimeout(function process() {
            console.log("Completed timer");
            if(val%2 == 0) {
                // even number
                resolve("Even");
            } else {
                // odd number
                reject("Odd");
            }
        }, 10000);
        console.log("Somewhere");
    });
}
```

Start
Somewhere
Completed timer



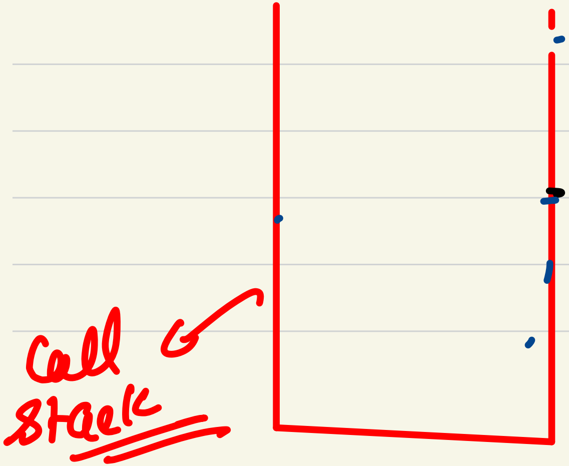
```
function fetchData(url) {
    return new Promise(function (resolve, reject) {
        console.log("going to start the download");
        setTimeout(function process() {
            let data = "Dummy downladed data";
            console.log("download completed");
            resolve(data);
        }, 10000);
        console.log("Timer to mimic download started");
    });
}

console.log("Starting the program");
console.log("We are expecting to mimic a downloader");
x = fetchData("www.google.com");
console.log("New promise object created successfully, but downloading still going on");
```

global

x =

state ~~pending~~ fulfilled

value ~~undefined~~ data



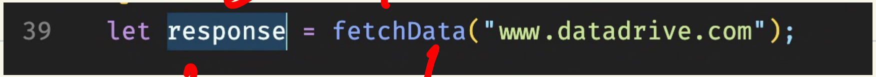
Starting the program
we are expecting ----downloade
going to start download
Timer to mimic download started
New prom obj .... on.
download completed

Consuming a promise

The promise consumption is the main beauty, using which we will avoid inversion of control.

Whenever we call a function, returning a promise, we will get a promise object which is like any JS object that we can store in a variable.

→ Now, the question , will JS wait here ??



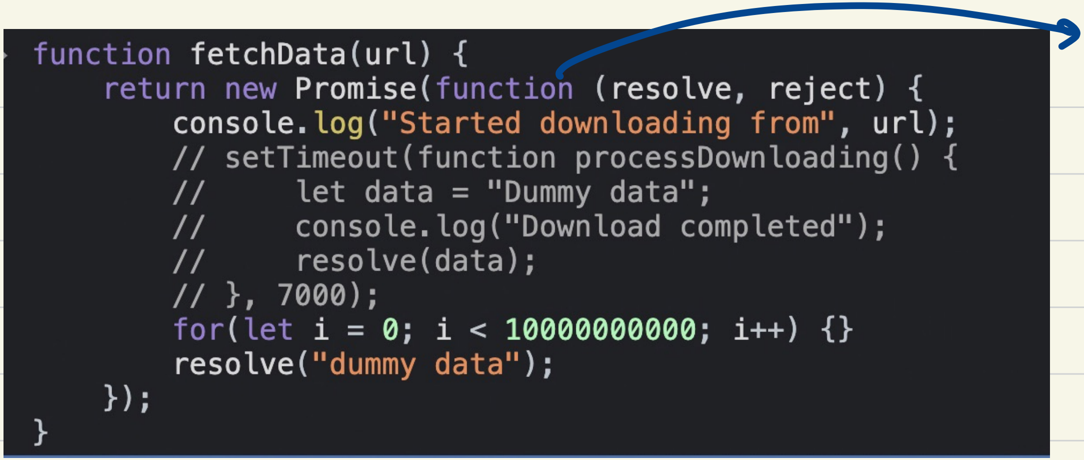
```
let response = fetchData("www.datadrive.com");
```

stores the promise object

function returns a promise object

Q: Will JS wait here for promise to be resolved if it involves any async piece of code ?.

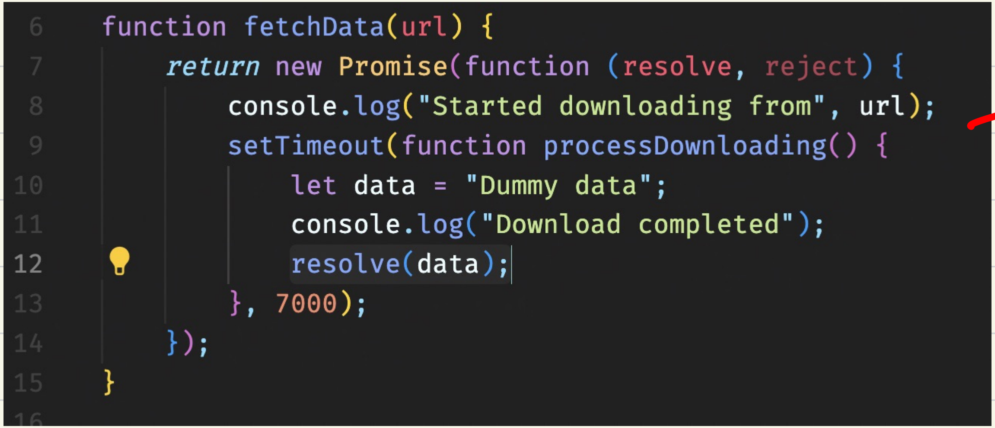
→ if creation of promise involves sync piece of code it will wait, otherwise won't.



```
function fetchData(url) {
    return new Promise(function (resolve, reject) {
        console.log("Started downloading from", url);
        // setTimeout(function processDownloading() {
        //     let data = "Dummy data";
        //     console.log("Download completed");
        //     resolve(data);
        // }, 7000);
        for(let i = 0; i < 10000000000; i++) {}
        resolve("dummy data");
    });
}
```

this callback is hang along sync piece of code, so JS will have to wait for promise object creation.

And just after the for loop, we also resolve the promise so we get a resolved promise.



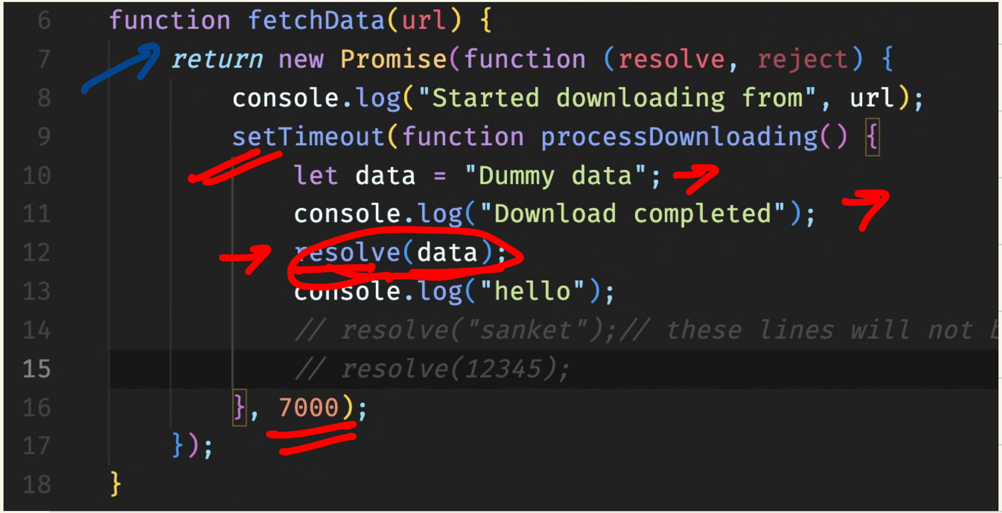
```
function fetchData(url) {
    return new Promise(function (resolve, reject) {
        console.log("Started downloading from", url);
        setTimeout(function processDownloading() {
            let data = "Dummy data";
            console.log("Download completed");
            resolve(data);
        }, 7000);
    });
}
```

→ Promise object will get created easily as there is no blocking piece of code, but initially it will be pending. As the fulfillment happens after a time of 7 sec.

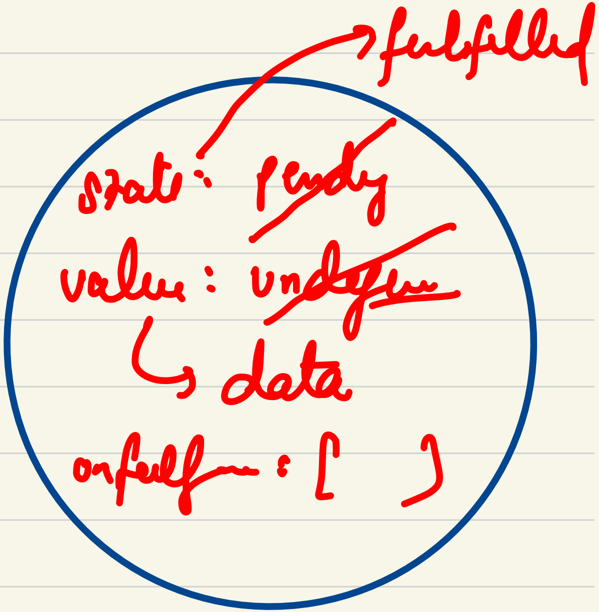
Now technically, when promise gets resolved, we have to execute some functions.

→ We can use .then() function on the promise object, to bind the functions we want to execute once we fulfill a promise.

The .then() function takes function as an argument that we want to execute after promise fulfills, and the argumen function takes value property as parameter.



```
function fetchData(url) {
    return new Promise(function (resolve, reject) {
        console.log("Started downloading from", url);
        setTimeout(function processDownloading() {
            let data = "Dummy data";
            console.log("Download completed");
            resolve(data);
            console.log("hello");
            // resolve("sanket");// these lines will not b
            // resolve(12345);
        }, 7000);
    });
}
```

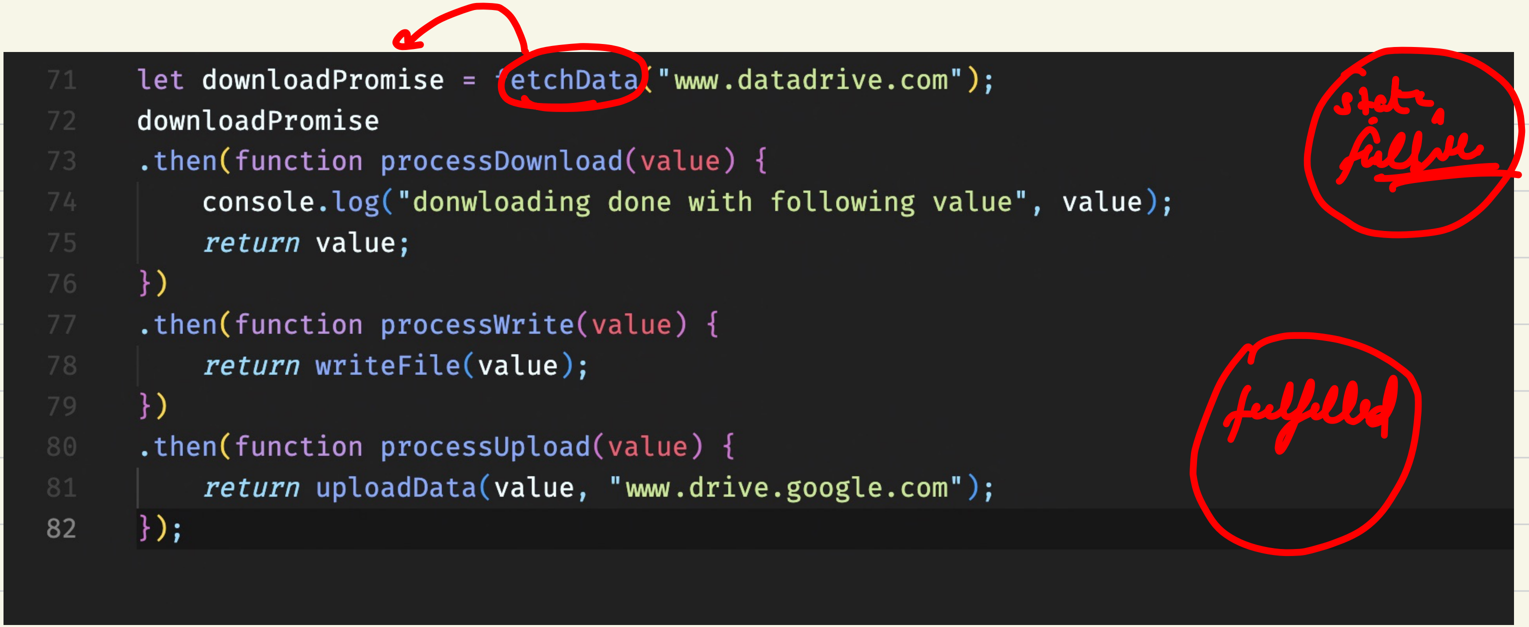


hello

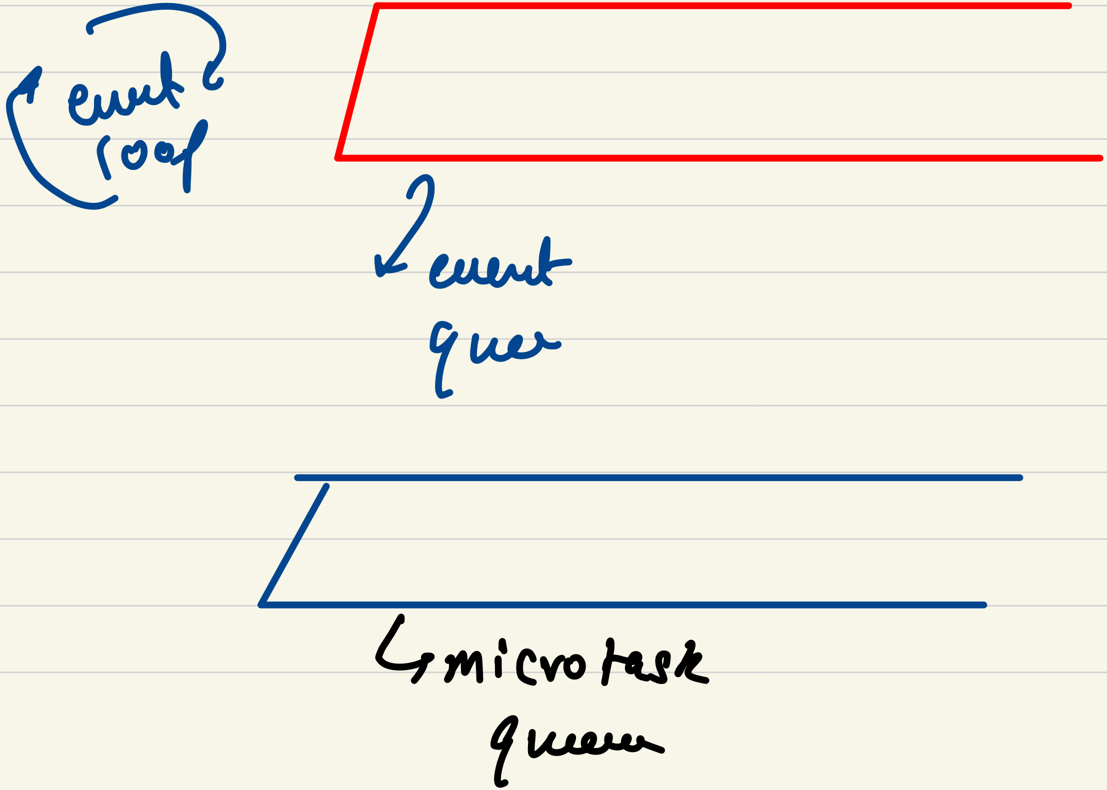
```
download Promise = fetchdata ( "www.google.com");

x = downloadPromise . then ( function f (value) {
        console.log (value)
        return   "Sanket";
  })
```

new Promise

the .then function itself returns a new promise.



```javascript
let downloadPromise = fetchData("www.datadrive.com");
downloadPromise
.then(function processDownload(value) {
    console.log("donwloading done with following value", value);
    return value;
})
.then(function processWrite(value) {
    return writeFile(value);
})
.then(function processUpload(value) {
    return uploadData(value, "www.drive.google.com");
});
```


event loop
event queue
microtask queue