Tuwash Notes

21st Feb 2023

DOM + Modern js - Class 4

→ **API :-** (Application programming Interface)

Interface

↳ mediator b/w the two

here API is mediator b/w Frontend & Backend.

Establish the communication b/w two Software Components

Features of Async Code

↳ Clean & concise

↳ Better error handling

↳ easier debugging

↳ H/w

Promise :- → Fulfilled

→ Not Fulfilled

✓ parallely execute in background in javascript we use promise

Async promise

callback function

two parameters

```
let meraPromise = new Promise (
                    function (resolve, reject) {
    console.log ("I am Inside promise");
    resolve (1998);
});

console.log ('Pehla');
```

output

- I am Inside promise
- Pehla

Now Async

```
let mera Promise = new Promise ( function
                    (resolve, reject) {
    SetTimeout ( function () {
        clg ('I am Inside');
    }, 5000);
    resolve (2233);
});
clg ('Pehla');
```

→ explicitly saying to resolve

output

- pehla
- I am Inside

We can also mark reject with an error.

```
reject (new Error ('Error Aaya'))
```

Call back function

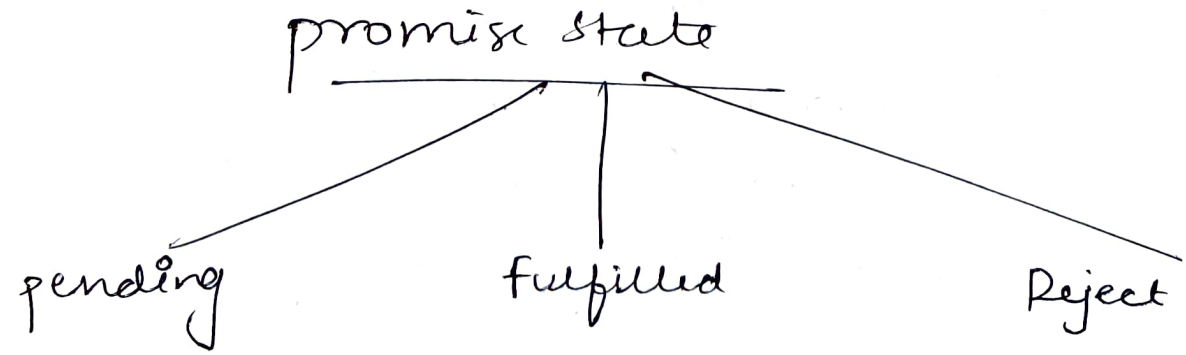
```
Let p = new promise ( _____ )
                    (_ , ___ )
                    {
```

Two parameter (resolve, reject)

if successfully → accepted
if not, error → Rejected.

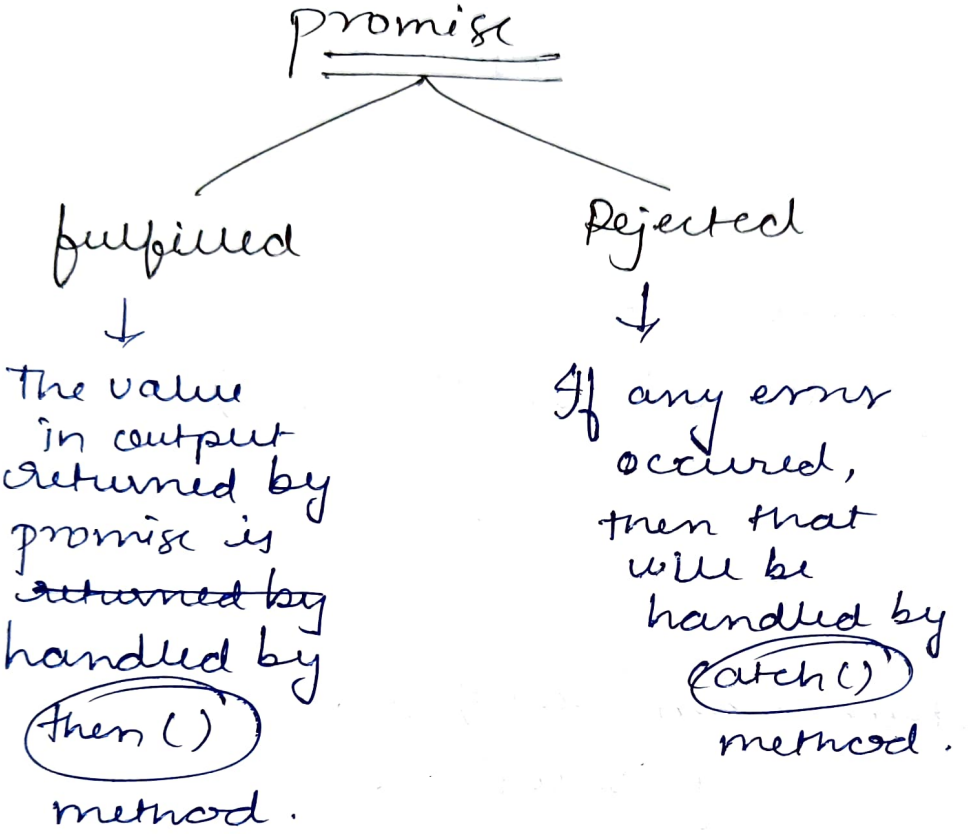
So catch the Error.

```
}
```

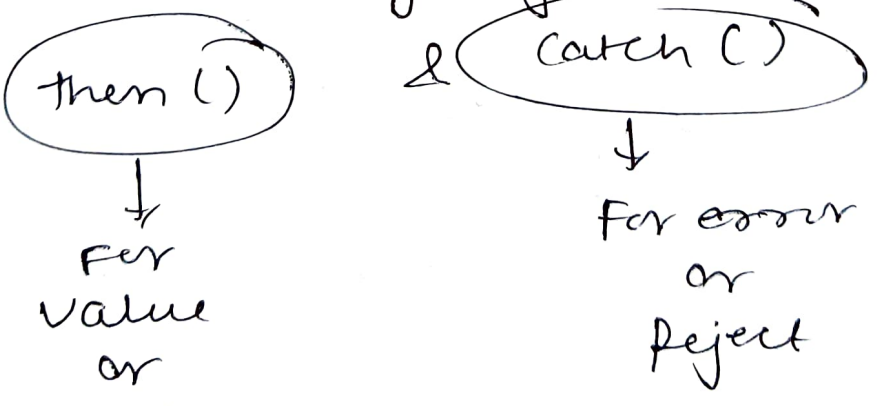


promise:- represents the eventual completion or failure of an asynchronous operation & its resulting value.

✓ parallel execution of code using promise



After the promise is Done, then we execute anything with the help of



```
let newPromise = new Promise (function
                    (resolve, reject) {

setTimeOut (function () {
    clg ('I am inside promise');
}, 5000);

//resolve (12345);
// reject (new Error ('error'))

});

newPromise.then ((value) => {
        clg (value)});
```

↓ will give 12345 output.

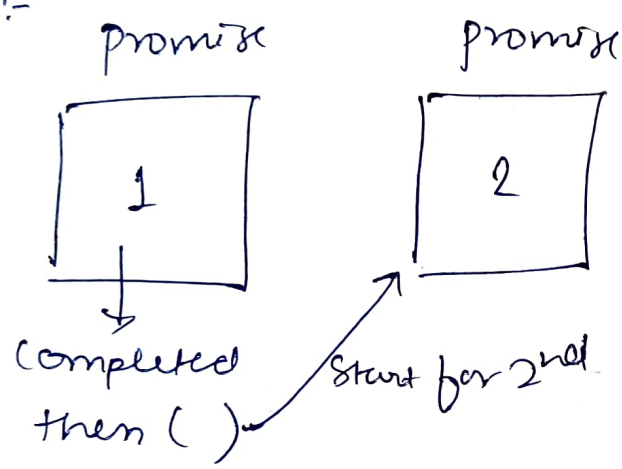
```
newPromise.catch ((error) => {
        clg ('error')});
```

↓ will give 'error' output written. after reject error occurred

✓ We Don't let our Synchronous code wait for Asynchronous, we let Asynchronous work in background parallly, we give it promise for accept & Reject of Asynchronous code

✓ if promise is completed, & then you want to perform any action, then use then() or catch()

eg:-



eg:-

```
let waadaa1 = new Promise (function (resolve,
                                    reject) {
    setTimeout(() => {
        console.log ('SetTimeout1 Started');
    }, 2000); resolve (true);
})

waadaa1. then (() => {
    let waadaa2 = new Promise (function (
                                    resolve, reject) {
        resolve ("waada 2 resolved");
    })
```

Here one more promise with 3000

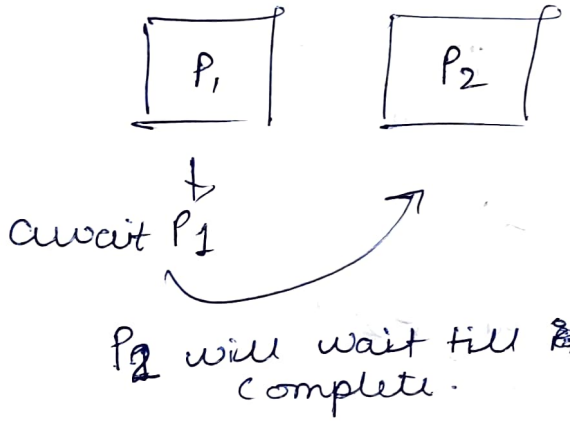
return waadaa!;

}).then((value) => console.log(value));

If we have 50 promises, then 50 then()?

NO

Async-await → Special Syntax used to work with promises



When you want to run your another Async code only when your first Async code is completed, use await

---

To make any code Async

```
async function abcd() {
    return 7;
}
console.log(abcd);
```

async will return promise

```
async function utility() {
  let delhiMausam = new Promise((resolve, reject) => {
      setTimeout(() => {
          resolve("Delhi is hot");
      }, 5000);
  });
  let hydMausam = new Promise((resolve, reject) => {
      setTimeout(() => {
          resolve("Hyderabad is cool");
      }, 6000);
  });
  let dM = await delhiMausam
  let hM = await hydMausam.
  return [dM, hM];
}
```

use await to make it wait else they will run parallelly.

Fetch API

In Network,
sending or retrieving data,
we use fetch API to retrieve
and ~~API~~ to send data.

```
let content = fetch("url....");
```

Syntax to fetch API

API will return → promise.

```
async function utility(){
let content = await fetch("url....");
let output = await content.json();
    console.log(output);
}
utility();
```

JavaScript Object Notation.

data is retrieved here & stored in content & then converted to JSON format.

JSON :- JavaScript Object Notation

i.e. in an object

key : value pair

(get call in API)

Fetch API → get() → retrieve

↓

let a = fetch ("url pass");

a.status
a.ok
a.JSON()
a.text()
} to check.

ex:-
```
let op = a.Json();
console.log(op);
```

✓ Sometimes the API is protected & you have to send the key or your authenticated data (user id), if you want to send then you use 'request header'

fetch (`url`, '[options]')

↓

create object & then add authentication or secret key.

{ header : {
authentication : key;
}
}

Now Sending using fetch API

post → send

→ fetch along with only url is get call

```
fetch('url')
```

→ fetch along with url & options but the object in option is secret key or authentication then also its get call.

```
fetch('url', 'options')
```

Now, In this options only the way we create object, it will be post in which we send data using fetch API;

```
fetch('url', 'options')
                ↳ post
```

```
let options = {
    method : 'post'
    header :
    .
    .
    .
}
```

post call :-

```
async function helper() {
    let options = {
        method : 'POST'
        body : JSON.stringify ({
            title : 'foo',
            body : 'bar',
        }),
        headers : {
            'content-type' : ------';
        },
    };
let content = await fetch ('url', options);
    let response = content.json();
    return response;
}

async function utility() {
    let ans = helper();
    console.log(ans);
}
utility();
```

this object can be copied from internet

we are sending this data in the fetch

url to store in database

an object is sent in url to update Data.

headers is additional information

JSON.Stringify()

↳ converting object Notation to String.

Format conversion ✓

Now,

Closures :-

creating function inside function

```
function abcd ()
    {
        var name = "xyz";
        function displayName()
          {
            console.log(name);
          }
        displayName();
    }
abcd(), abcd();
```

xyz printed

✓ let is a block Scope

if we will use let in place of var then also xyz will be printed

```
let name = "Sher";
function init() {
    let name = "Mozilla";
    function displayName() {
        let name = "Babbar"
        console.log(name);
    }
    displayName();
}
init();
```

→ Babbar will be printed

Variable → local → let → block scope

Variable → Global → var

when the function is completed then the name variable will be destroyed

if you will call

```
let funct = init();
funct();
```

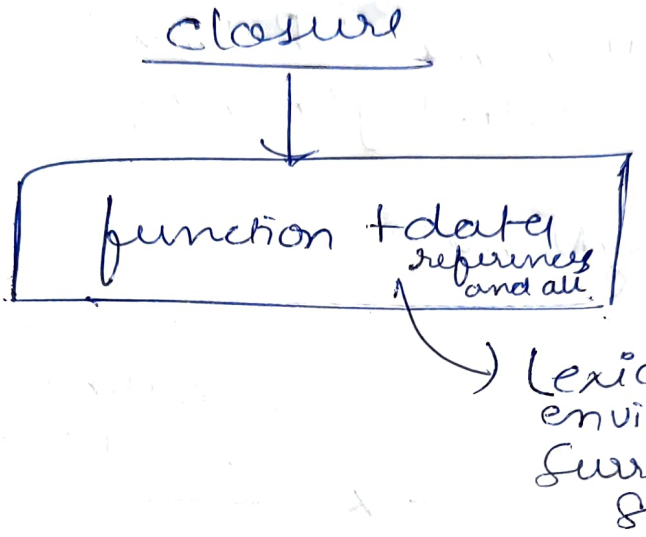
→ here name is destroyed. but output will be 'Mozilla'

↑

(because of closure)

When you create nested function every function has its closure

Closure is Something in which function is Binded with its required data.



with references of data not copy

✓ closure is made for all nested function you create in the form of References ✓

Nested function → Closure

↓

Reference

Not copy