JS Promises

- It is a good way to handle asynchronous operations.
- It is used to find out if asynchronous operation is succesfully completed or not.
- 3 States → Pending → Fulfilled → Rejected

Name	:	Description
Pending	~	Process is not complete
Fulfilled	~	operation is successful
Rejected	~	Process ends (if error occurs)

- For eg: When request data from server by Promise will be Pending State
  When data arrives succesfully → Fulfilled state
  If error occur → Rejected.
- Create Promise by Promise constructor. It also takes function as an argument → also accepts 2 fun.
  - resolve
  - reject

```
let promise = new Promise(function(resolve, reject){
    //do something
});
```

- Promise return success → resolve()
  Error → reject()

Eg 1) Program with Promise

```
<Script>
    const count = true;
    let promi = new Promise(function(resolve, reject){
        // Producing code
        if(count){
            resolve("There is a count value");
        }else{
            reject("There is no count value");
        }
    });
```

document.querySelector("p").innerHTML = promi;

o/p :: [object Promise]

Console o/p :: ▶ Promise {<fulfilled>: 'This is a count value'}

2) Promise Chaining

Promises → Useful → Handle → More than one → Asynchronous task → one after another → for this → we use → Promise chaining

3) then()

Used with callBack → when Promise (success) fulfilled or resolved.

Syntax

```
promiseObject.then(onFullfilled, onRejected)
```

then() → Chain functions to Promise

→ called when Promise is resolved successfully.

→ Can chain multiple then() with promise.

```
<script>
  let promi = new Promise(function(resolve, reject){
      resolve("Promise resolved <br/>");
  });
  promi
    .then(function SuccessValue(result){
      document.write(result)
    })
    .then(function SuccessValue(){
      document.write("...");
    })
</script>
```

3) catch() → Used with callback when promise is rejected or if error occurs.

```
<script>
    let promi = new Promise (function (resolve, reject) {
        reject ("Promise rejected");
    });
    //execute when promise is resolved successfully
    promi
        .then (function successValue (result) {
            document.write(result);
        })
    //execute if there is an error
        .catch (function errorValue (result) {
            document.write (result);
        })
</script>
```

o/p

Promise rejected

4) Promise Versus CallBack

Promise —similar→ CallBack function

diff → Error Handling → Promise (easy) → callback → difficult

Promise	CallBack

```
api().then(function(result){
    return api2();
}).then(function(result2){
    return api3();
}).then(function(result3){
    //do work
}).catch(function(error){
    // handle
});
```

```
api(function(res){
    api2(function(res2){
        api3(function(res3){
            //do work
            if(error){
                //do
            }
            else{
                //do
            }
        });
    });
});
```

5) finally

You can also use finally() with Promises.

Eg:
```
<script>
let promi = new Promise(resolve, reject) {
    reject("Promise Rejected <br/>");
  });
  promi
    .finally(
        function greet() {doc.write("code exe!...")};
    )
</script>
```

o/p :- Code Executed!...

← work.

Pg (29)

Is Promise: Object → contain both [Producing code to calls → consuming code

Syntax:

```
let {promise-name} = new Promise(function(myResolve, myReject) {

// Producing code (May take time)
myResolve(); // when success
myReject(); // when error
});

// Consuming code (Must wait → fulfilled Promise)
{promise-name}.then( function(value) {/* If success */},
function(error) {/* If error */}
);
```

MAY 2020

Su	Mo	Tu	We	Th	Fr	Sa
					1	2
3	4	5	6	7	8	9
10	11	12	13	14	15	16
17	18	19	20	21	22	23
24	25	26	27	28	29	30

13 7

Promise Object Properties

* JS Promise → can be → Pending → Rejected
  → Fulfilled

* Support 2 properties → State
  → result
  → Cannot access
  → must use Promise handle methods → properties

When Promise obj is	Result is
Pending (working)	Undefined
Fulfilled	Value
Rejected	Error Object

Eg:- O/p :- OK

```
<body>
  <p id="p1"></p>
  <script>
    function display(some){
      document.getElementById("p1").innerHTML = some;
    }
  let prom = new Promise(function(myResolve, myReject){
    let x=0;
    //some code (try to change x to 5)
    // x=10;
    if(x==0){
      myResolve("OK");
    }else{
      myReject("Error");
    }
  });
  prom.then(
      function(value){display(value);},
      function(error){display(error);}
  );
  </script>
</body>
```

```
<body>
    <p id="p"></p>
<script>
    function display(some) {
        document.getElementById("p").innerHTML = some;
    }
let prom = new Promise(function(myResolve, myReject){
    let req = new XMLHttpRequest();
    req.open("GET", "promise.html");
    req.onload = function(){
        if(req.status == 200){
            myResolve(req.response);
        }else{
            myReject("File not found");
        }
    };
    req.send();
});
prom.then(
    function(value){display(value);},
    function(error){display(error);}
);
</script>
</body>
```

asynch

await

promise.html

```
<body>
<p>Promise is successful</p>
</body>
```

o/p

Promise is Success

Promise [Pg 136 - 138]

Promise chaining

* Allows Developer to sequence multiple Async ops together.
* Each op in chain returns promise
  - Promise result passed as i/p to next Promise (sequence)
* Methods of then() used with callback when promise succesfully fulfilled or resolved

Syntax:

```
promiseObject.then(
    onFulfilled,
    onRejected);
```

Used to chain functions to Promise.

Eg: 1) Check Payment Status ⇒ Place order ⇒ Show order status

```
checkout().then(data => { return getPayment(data); })
  .then(response => { return placeorder(data); })
  .then(response => { ShowTheorderStatus(); })
  .catch(error => {
      // handle error occur
  });
```

Eg 2)

```
let countval = new Promise (function (resolve, reject) {
  resolve ("Promise Resolved") ;
}) ;

countval .then (function succ (result) {
    console . log (result) ;
  })
  .then (function succ () {
      console . log ("Promise Chaining like this") ;
  }) ;
```

→ 1

→ 2

Eg3) catch() → used with callback when Promise (Rejected or Error occurrence)

```
let countval = new Promise (function (resolve, reject) {
  reject ("Promise Rejected") ;
}) ;

countval .then (function sucval (result) {
    console . log (result) ;
  })
  .catch (function errval (result) {
      console . log (result) ;
  }
}
```

O/P

"Promise Rejected"