
ANY DOUBTS AFTER THIS COMPLETION OF PRESENTATION U CAN ASK....!
-MOULI RAMADASU

SetTimeout & setInterval ?

- JavaScript allows to invoke the function can executed right now or particular time
- It take a callback function and attached to the timer

```
> function display(){
    var i=100;
    setTimeout(()=>console.log(i),3000);
    console.log("HelloHai")
  }
  display();
  HelloHai
  undefined
  100
>
```

```
> function display(){
    var i=100;
    setInterval(()=>console.log(i),3000);
    console.log("HelloHai")
  }
  display();
  HelloHai
  undefined
  12 100
```

Callback function ?

- Function take argument as another function, and It's returned a function
- When you want one function to execute only after another function has completed its execution, we use callback functions in JavaScript.

```
> function mycal(num1,num2, callback){
      res= num1+num2;
      callback(res)
  }

  function display(sum){
  console.log(sum)
  }

  mycal(10,20,display)


  30
```

Promise ?

- It is an **object** The Promise object supports two properties like **state** and **result**
- Promise. Then () takes two arguments, a callback for success and another one for failure
- Promise has some built-in Method

1.Promise .all()

2.promise.allSettled()

3.promise.any()

4.promise.prototype.catch()

5.promise.prototype.finally()

6.promise.prototype.then()

The **Promise.all()** method takes an iterable of promises as an input and returns a single Promise that resolves to an array of the results of the input promises.

```
var p1 = Promise.resolve('Hai');
var p2 = 'Hello';
var p3 = new Promise((resolve, reject) => {
  setTimeout(() => {
    resolve("How are u");
  }, 100);
});

Promise.all([p1, p2, p3]).then(values => {console.log(values);});
▼ Promise {<pending>} i
  ▶ [[Prototype]]: Promise
    [[PromiseState]]: "fulfilled"
    [[PromiseResult]]: undefined
▶ (3) ['Hai', 'Hello', 'How are u']
```

```
> var p1 = Promise.resolve('Hai');
  var p2 = 'Hello';
  var p3 = new Promise((resolve, reject) => {
    setTimeout(() => {
      reject("How are u");
    }, 100);
  });

  Promise.all([p1, p2, p3]).then(values => {console.log(values);});
◀ ▼ Promise {<pending>} i
    ▶ [[Prototype]]: Promise
      [[PromiseState]]: "rejected"
      [[PromiseResult]]: "How are u"
```

The **Promise.allSettled()** method returns a promise that resolves after all the given promises have either fulfilled or rejected, with an array of objects that each describes the outcome of each promise.

```
> var p1 = Promise.resolve('Hai');
  var p2 = 'Hello';
  var p3 = new Promise((resolve, reject) => {
    setTimeout(() => {
      reject("How are u");
    }, 100);
  });

  Promise.allSettled([p1, p2, p3]).then(result => {console.log(result);});
<· ▶ Promise {<pending>}
  ▼ (3) [{…}, {…}, {…}] i
    ▶ 0: {status: 'fulfilled', value: 'Hai'}
    ▶ 1: {status: 'fulfilled', value: 'Hello'}
    ▶ 2: {status: 'rejected', reason: 'How are u'}
      length: 3
    ▶ [[Prototype]]: Array(0)
```

Promise.any() takes an iterable of Promise objects. It returns a single promise that resolves as soon as any of the promises in the iterable fulfills

```
const promise1 = Promise.reject(0);
const promise2 = new Promise((resolve) => setTimeout(resolve, 1000, "Angular"));
const promise3 = new Promise((resolve) => setTimeout(resolve, 3000, "ReactJs"));

const promises = [promise1, promise2, promise3];

Promise.any(promises).then((value) => console.log(value));
▶ Promise {<pending>}
Angular
```

The **catch()** method returns a Promise and deals with rejected cases only It behaves the same as calling

```
> const promise1 = new Promise((resolve, reject) => {
    throw "Error May beCome";
  });

  promise1.catch((error) => {console.error(error)});
  ▶ Error May beCome
  ▶ Promise {<fulfilled>: undefined}
>
```

The **finally()** method returns a Promise When the promise is settled either fulfilled or rejected, the specified callback function is executed.

```
function checkMail() {
  return new Promise((resolve, reject) => {
    if (0) {
      resolve('Mail has arrived');
    } else {
      reject('Failed to arrive');
    }
  });
}

checkMail()
  .then((mail) => {console.log(mail);})
  .catch((err) => {console.error(err);})
  .finally(() => {console.log('Experiment completed');});
```

```
▶ Failed to arrive
Experiment completed
▶ Promise {<fulfilled>: undefined}
```

The **then()** method returns a Promise It takes up to two arguments callback functions for the success and failure cases of the Promise.

```
function prom1(resolve,reject){
if(true){
      resolve("Angular")
      }
else{
      reject('React-Js')
      }
}

prom= new Promise(prom1)
prom.then(e=>{console.log(e);return "I am Learning "+e}).then(e=>console.log(e))
```

```
Angular
I am Learning Angular
```

async & await ?

- Async make function as promise
- Await it is waiting for the promise response

```
function resolveAfter2Seconds() {
return new Promise(resolve => {
setTimeout(() => {resolve('resolved');}, 2000);
});
}

async function asyncCall() {
console.log('calling');
const result = await resolveAfter2Seconds();
console.log(result);
}
asyncCall();
```

```
calling
▶ Promise {<pending>}
resolved
```

Thank You …. !