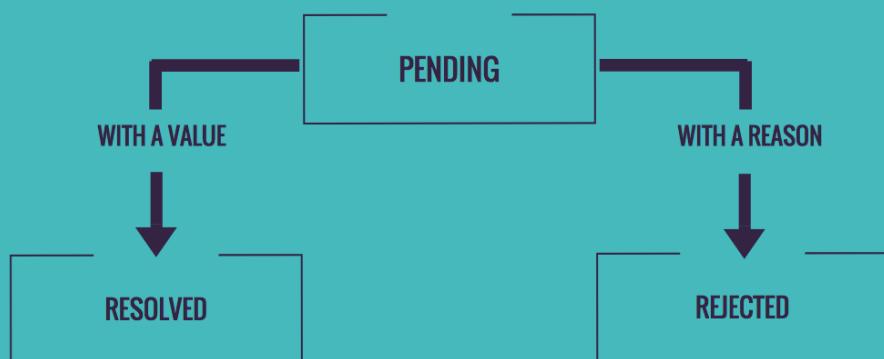


# PROMISES

JAVASCRIPT



# Javascript

## Fulfilled төлөвтэй Promise-ийг барьж авах

Promise дотроо нэг үйлдэл хийчихээд тэрнээс гарсан **амжилттай** хариуг нь өөр зүйлд эсвэл цааш ашиглах шаардлагатай болдог энэ үед then гэсэн функцийг үргэлжлүүлэн дуудаж ашигладаг.

```
myPromise.then(result => {  
});
```

```
1 const myPromise = new Promise((resolve, reject) => {  
2   // responseFromServer is set to false to represent an  
3   // unsuccessful response from a server  
4   let responseFromServer = false;  
5  
6   if(responseFromServer) {  
7     resolve("We got the data");  
8   } else {  
9     reject("Data not received");  
10  }  
11};  
12 myPromise.then(result => {  
13   console.log(result);  
14});  
15 myPromise.catch(error => {  
16   console.log(error);  
17});
```

# Javascript

## Rejected төлөвтэй Promise-ийг барьж авах

Promise дотроо нэг үйлдэл хийчихээд тэрнээс гарсан **амжилтгүй/алдаатай** хариуг нь өөр зүйлд эсвэл цааш ашиглах шаардлагатай болдог энэ үед catch гэсэн функцийг үргэлжлүүлэн дуудаж ашигладаг.

```
myPromise.catch(error => {  
});
```

```
1 const myPromise = new Promise((resolve, reject) => {  
2   // responseFromServer is set to false to represent an  
3   // unsuccessful response from a server  
4   let responseFromServer = false;  
5  
6   if(responseFromServer) {  
7     resolve("We got the data");  
8   } else {  
9     reject("Data not received");  
10  }  
11};  
12  
13 myPromise.then(result => {  
14   console.log(result);  
15});  
16 myPromise.catch(error => {  
17   console.log(error);  
18});
```

## Javascript Promise

Promise буюу амлалт нь яг л утга шигээ үйлдэл гүйцэтгэдэг. Энийг дор тайлбарлах болно. Promise нь бүтээгч(constructor) функц бөгөөд энэ нь юу гэсэн үг вэ гэхээр new гэдэг түлхүүр үг ашиглан бичиж байж энэ нь объект үүсгэдэг. Бусад тохиолдолд хоосон байгаад алдаа заагаад байдаг. Зэрэг гүйцэтгэгдэж(async) байгаа үйлдэл дээр эсвэл гүйцэтгэх хугацаа нь тодорхойгүй серверийн хүсэлттэй ажиллахдаа promise-г ашиглана. Функц зарлахдаа заавал (resolve, reject) параметрүүдийг хүлээн авна.

```
const myPromise = new Promise((resolve, reject) => {  
});
```

Promise өөрөө дотроо

- pending
- fulfilled
- rejected гэсэн Зтөлөвтэй байдаг. Дээрх зарлсан myPromise маань default дээрээ pending гэсэн төлөвтэй байна гэсэн үг. Доорх тохиолдолд амжилттай болон амжилтгүй гэсэн төлөвтэй болно.

```
const myPromise = new Promise((resolve, reject) => {  
  if(condition here) {  
    resolve("Promise was fulfilled");  
  } else {  
    reject("Promise was rejected");  
  }  
});
```

## Javascript Promise

```
const myPromise = new Promise((resolve, reject) => {
  if(condition here) {
    resolve("Promise was fulfilled");
  } else {
    reject("Promise was rejected");
  }
});
```

```
myPromise.then(result => {
});
```

```
myPromise.catch(error => {
});
```

Promise зэрэг хийгдэж байгаа үйлдлийг заавал гүйцээнэ дарааллаар нь хийнэ гэсэн амлалтыг өгч ажилдаг үйлдэл юм.

## Javascript

### Promising XMLHttpRequest

```
function get(url) {
  // Return a new promise.
  return new Promise(function(resolve, reject) {
    // Do the usual XHR stuff
    var req = new XMLHttpRequest();
    req.open('GET', url);

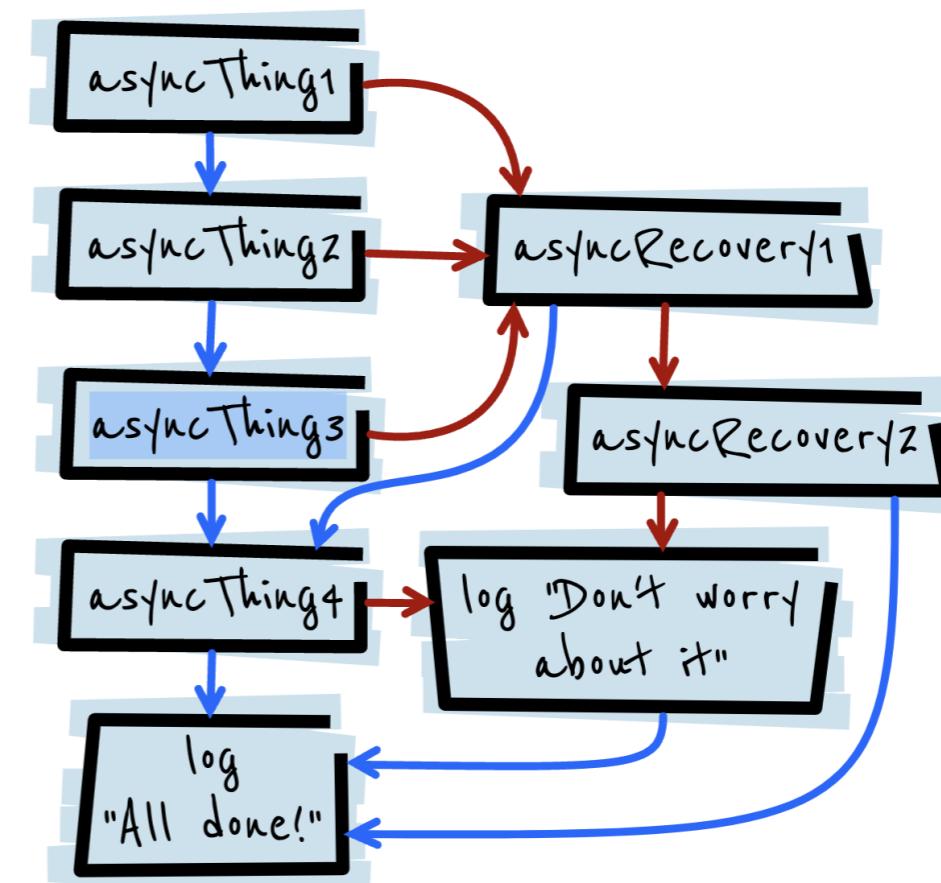
    req.onload = function() {
      // This is called even on 404 etc
      // so check the status
      if (req.status == 200) {
        // Resolve the promise with the response text
        resolve(req.response);
      } else {
        // Otherwise reject with the status text
        // which will hopefully be a meaningful error
        reject(Error(req.statusText));
      }
    };

    // Handle network errors
    req.onerror = function() {
      reject(Error("Network Error"));
    };

    // Make the request
    req.send();
  });
}
```

```
get('story.json').then(function(response) {
  console.log("Success!", response);
}, function(error) {
  console.error("Failed!", error);
})
```

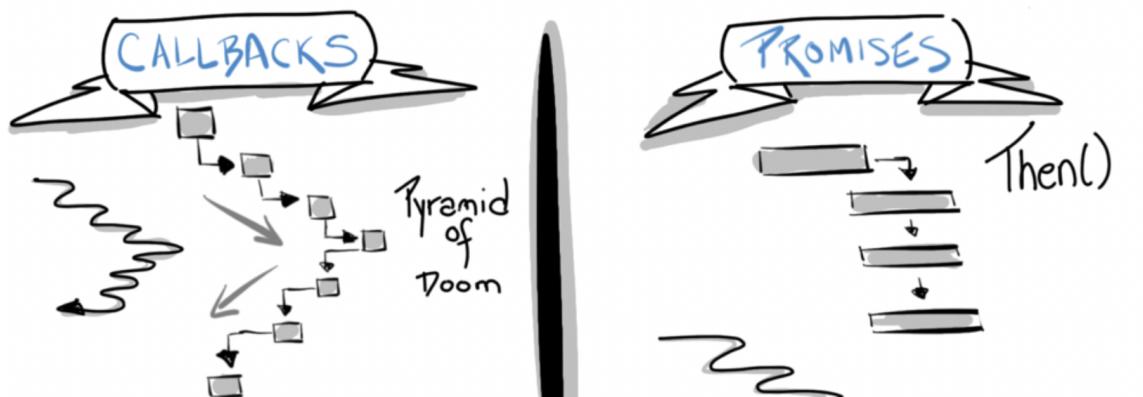
AJAX хүсэлттэй хамтран promise-ийг ашиглах нь илүү давуу талтай бөгөөд тухайн хүсэлтийг яваад хариу иртэл нь хүлээгээд үйлдлээ гүйцээгээд амлсан бол амлалтаа биелүүлж үргэлжлүүлдэг.



# Javascript

## Promise yy? Callback yy?

Callback Promise 2 нь ерөнхийдөө ижил асуудал болох `async`-ийн асуудлыг шийдэх гэж гарч ирсэн бөгөөд `promise` нь сүүлд гарсан арай орчин үеийн хувилбар гэж ойлгож болно. Ижил зориулалттай ч бичиглэлээсээ шалтгаалаад өөр өөр зүйл ашиглагддаг учраас аль нэгийг нь сонгон ашигладаг. Гэхдээ ер нь хоёуланг нь хамтад нь бичих гээд хэрэггүй гэдгийг зөвлөж байна. Бичиглэлийн хувьд `promise` нь илүү цэгцтэй томоохон асуудалд эвтэйхэн шийдэл болж чадна. Маш хурдан жижиг хэмжээний хүсэлтэд `callback`-ийг дуудахад буруудахгүй.



# Javascript

## Promise-тэй холбоотой дасгалууд

```
let done = true

const isItDoneYet = new Promise((resolve, reject) => {
  if (done) {
    const workDone = 'Here is the thing I built'
    resolve(workDone)
  } else {
    const why = 'Still working on something else'
    reject(why)
  }
})

const checkIfItsDone = () => {
  isItDoneYet
    .then(ok => {
      console.log(ok)
    })
    .catch(err => {
      console.error(err)
    })
}

checkIfItsDone()
```

# Javascript

## Promise-тэй холбоотой дасгалууд

### Console-той ажиллах нь

Доорх кодыг editor дээрээ хуулж тавиад ажиллуулаад promise-ийн өмнөх дараах утгууд хэрхэн црж байгааг хараарай.

```
<script>
console.log('Hello i am starting');
let value_checker = "";

const isItDoneYet = new Promise((resolve, reject) => {
  const api_url = "https://reqres.in/api/users?page=2";
  const req = new XMLHttpRequest();
  req.open('GET', api_url);
  req.onload = function() {
    if (req.status == 200) {
      resolve(req.response);
    }
    else {
      reject(Error(req.statusText));
    }
  };
  req.send();
});

const checkIfItsDone = () => {
  isItDoneYet
    .then(ok => {
      value_checker = ok;
      console.log('after promise its value_checker= ' + value_checker)
      console.log(ok)
    })
    .catch(err => {
      console.log('Aldaa=>');
      console.error(err)
    })
}

checkIfItsDone();
console.log('its value_checker= ' + value_checker)

</script>
```