3 måder at kode asynkront på i JS

1. Callbacks
2. Promises
3. Async/Await

2. Promises

```
1   'use strict'
2
3   var promise = new Promise((resolve,reject) => {
4       resolve('Promises are meant to be broken.');
5   });
6
7   console.log(promise);
```

```
'use strict'

var promise = new Promise((resolve,reject) => {
    resolve('Promises are meant to be broken.');
});

promise.then((result) => {
    console.log(result);
}).catch((error) => {
    console.log(error);
})
```

```
'use strict'

var promise1 = new Promise((resolve,reject) => {
    resolve('Inside Promise 1.');
});

var promise2 = new Promise((resolve,reject) => {
    resolve('Inside Promise 2.');
});

Promise.all([promise1,promise2])
    .then((result) => {
        console.log(result);
    })
    .catch((error) => {
        console.log(error);
    })
```

```
'use strict'

// First Promise.
var promise1 = new Promise((resolve,reject) => {

    var a = 15;
    var b = 5;
    var c = a+b;

    // Resolving the sum obtained.
    resolve(c);

});

// Second Promise.
var promise2 = new Promise((resolve,reject) => {

    var x = 5;
    var sum = null;

    // Obtaining the value returned by first promise.
    promise1.then((result) => {
    sum = result+x;

    // Resolving the final sum obtained.
    resolve(sum);

    });

});

// Obtaining the sum returned by second promise.
promise2.then((finalResult) => {
    console.log(finalResult);
}).catch((error) => {
    console.log(error);
})
```

Real world example

```
'use strict'

fetch('https://jsonplaceholder.typicode.com/posts')
    .then(res => {
        res.json()
            .then((data) => {
                console.log(data);
            })
    }).catch((error) => {
        console.log(error);
    })
```

`fetch` returner et `Promise`. Resultatet af dette promise vil vi gerne konvertere til JS Objekt med `res.json()` som igen returnerer endnu et promise.

```
(100) [{…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}
▼, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}
, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}]
 ▶0: {userId: 1, id: 1, title: "sunt aut facere repellat provident occaecati excepturi optio reprehenderit", body: "quia et suscipit↵suscipit recusandae consequuntur …strum rerum est au…
 ▶1: {userId: 1, id: 2, title: "qui est esse", body: "est rerum tempore vitae↵sequi sint nihil reprehend…aperiam non debitis possimus qui neque nisi nulla"}
 ▶2: {userId: 1, id: 3, title: "ea molestias quasi exercitationem repellat qui ipsa sit aut", body: "et iusto sed quo iure↵voluptatem occaecati omnis e…↵molestiae porro eius odio et lab…
 ▶3: {userId: 1, id: 4, title: "eum et est occaecati", body: "ullam et saepe reiciendis voluptatem adipisci↵sit … ipsam iure↵quis sunt voluptatem rerum illo velit"}
 ▶4: {userId: 1, id: 5, title: "nesciunt quas odio", body: "repudiandae veniam quaerat sunt sed↵alias aut fugi…sse voluptatibus quis↵est aut tenetur dolor neque"}
 ▶5: {userId: 1, id: 6, title: "dolorem eum magni eos aperiam quia", body: "ut aspernatur corporis harum nihil quis provident …s↵voluptate dolores velit et doloremque molestiae"}
 ▶6: {userId: 1, id: 7, title: "magnam facilis autem", body: "dolore placeat quibusdam ea quo vitae↵magni quis e…t excepturi ut quia↵sunt ut sequi eos ea sed quas"}
 ▶7: {userId: 1, id: 8, title: "dolorem dolore est ipsam", body: "dignissimos aperiam dolorem qui eum↵facilis quibus…↵ipsam ut commodi dolor voluptatum modi aut vitae"}
 ▶8: {userId: 1, id: 9, title: "nesciunt iure omnis dolorem tempora et accusantium", body: "consectetur animi nesciunt iure dolore↵enim quia a…st aut quod aut provident voluptas autem v…
 ▶9: {userId: 1, id: 10, title: "optio molestias id quia eum", body: "quo et expedita modi cum officia vel magni↵dolorib…it↵quos veniam quod sed accusamus veritatis error"}
 ▶10: {userId: 2, id: 11, title: "et ea vero quia laudantium autem", body: "delectus reiciendis molestiae occaecati non minima…luptates ut commodi qui incidunt↵ut animi commodi"}
 ▶11: {userId: 2, id: 12, title: "in quibusdam tempore odit est dolorem", body: "itaque id aut magnam↵praesentium quia et ea odit e…uia id voluptatem↵incidunt ea est distinctio odio"}
 ▶12: {userId: 2, id: 13, title: "dolorum ut in voluptas mollitia et saepe quo animi", body: "aut dicta possimus sint mollitia voluptas commodi …ssumenda consectetur porro architecto ip…
```

Lad os lave en fejl

```
'use strict'

fetch('https://jsonplaceholdertypicode.com/posts')
    .then(res => {
        res.json()
            .then((data) => {
                console.log(data);
            })
    }).catch((error) => {
        console.log('Checking for error - ', error);
    })
```

```
▶ GET https://jsonplaceholdertypicode.com/posts net::ERR_NAME_NOT_RESOLVED
Checking for error -  TypeError: Failed to fetch
```

3 Async/Await

Vi starter med `async` keyword

```
async function f() {
  return 1;
}
```

Hvilket bare betyder at funktionen altid returnere et Promise.

Lad os bruge funktionen

```
async function f() {
  return 1;
}

f().then(alert); // 1
```

Explicit kunne vi returnere et Promise

```
async function f() {
  return Promise.resolve(1);
}

f().then(alert); // 1
```

Det andet keyword er `await` og det kan kun bruges inde i async funktioner. Det ser således ud:

```
// works only inside async functions
let value = await promise;
```

`await` får javascript til at vente indtil vores `promise` returnere et resultat.

Her er et eksempel, hvor et promise tager 1 sekund at udføre:

```
async function f() {

  let promise = new Promise((resolve, reject) => {
    setTimeout(() => resolve("done!"), 1000)
  });

  let result = await promise; // wait until the promise resolves (*)

  alert(result); // "done!"
}

f();
```

Fejlhåndtering

```
async function f() {

  try {
    let response = await fetch('/no-user-here');
    let user = await response.json();
  } catch(err) {
    // catches errors both in fetch and response.json
    alert(err);
  }
}

f();
```

Promise.all

Lad os antage vi har et API kald der efter noget tid returnerer noget JSON

```
// First promise returns an array after a delay
const getUsers = () => {
  return new Promise((resolve, reject) => {
    return setTimeout(() => resolve([{ id: 'jon' }, { id: 'andrey' }, { id: 'tania' }]), 600)
  })
}
```

Så har vi et andet kald der afhænger af det første kalds resultat

```
// Second promise relies on the result of first promise
const getIdFromUser = users => {
  return new Promise((resolve, reject) => {
    return setTimeout(() => resolve(users.id), 500)
  })
}
```

Og for at blive lidt vild; Et tredie kald der ændrer det andet kald.

```
// Third promise relies on the result of the second promise
const capitalizeIds = id => {
  return new Promise((resolve, reject) => {
    return setTimeout(() => resolve(id.toUpperCase()), 200)
  })
}
```

Løsningen kunne være

```
const runAsyncFunctions = async () => {
  const users = await getUsers()

  for (let user of users) {
    const userId = await getIdFromUser(user)
    console.log(userId)

    const capitalizedId = await capitalizeIds(userId)
    console.log(capitalizedId)
  }

  console.log(users)
}

runAsyncFunctions()
```

Men problemet er at jeg får følgende output:

```
jon
JON
andrey
ANDREY
tania
TANIA
(3) [{…}, {…}, {…}]
```

Brug `Promise.all`for at kalde og anvende resultaterne sekventielt

```
const runAsyncFunctions = async () => {
  const users = await getUsers()

  Promise.all(
    users.map(async user => {
      const userId = await getIdFromUser(user)
      console.log(userId)

      const capitalizedId = await capitalizeIds(userId)
      console.log(capitalizedId)
    })
  )

  console.log(users)
}
```

Hvilket giver følgende output:

```
(3) [{…}, {…}, {…}]
jon
andrey
tania
JON
ANDREY
TANIA
```