Asynkron programmering

- Asynkron programmering tillader flere ting at ske samtidig – uden at bruge tråde
- Anvendes både i browseren og i Node.js

Asynkron funktion

- En asynkron funktion kan bruge en callback til at fortsætte udførelsen, efter at funktionen selv er blevet udført
- Callbacks er isoleret set let at forstå
- Men exceptions og fejl skal håndteres forskelligt
- Og koden bliver især kompliceret, når to eller flere asynkrone funktioner skal udføres sekventielt

eksempel

```
// callback1.js
function langvarig(nr, callback) {
    let type = Math.trunc(Math.random() * 3);
    if (type === 0) throw new Error(`${nr} Slet ikke OK!`);
    setTimeout(afslut, 500);
    function afslut() {
        if (type === 1) callback(false, `${nr} OK!`);
        else if (type === 2) callback(`${nr} Ikke OK!`);
    }
}
```

```
// callback1.js
function langvarig(nr, callback) {...}

try {
    langvarig(1, slut);
    console.log('Efter kald af langvarig()');
}
catch (e) {
    console.log('Exception: ' + e);
}

function slut(fejl, resultat) {
    if (fejl)
        console.log('Fejl: ' + fejl);
    else
        console.log('Resultat: ' + resultat);
}
```

```
// caLLBack2.js
function langvarig(nr, callback) {...}

try {
    langvarig(1, slut1);
}
catch (e) {
    console.log('Exception: ' + e);
}

function slut2(fejl, resultat) {
    if (fejl)
        console.log('Fejl: ' + fejl);
    else
        console.log('Resultat: ' + resultat);
}
```

```
function slut1(fejl, resultat) {
    if (fejl)
        console.log('Fejl: ' + fejl);
    else {
        console.log('Resultat: ' + resultat);
        try {
            langvarig(2, slut2);
        }
        catch (e) {
            console.log('Exception: ' + e);
        }
    }
}
```

Promise

- En funktion, der returnerer en værdi, er lettere at forstå end en funktion, der kalder en callback funktion
- En promise er et objekt, der repræsenterer en asynkron funktions returnerede værdi
- En promise kan være i én af tre tilstande:
  - en starttilstand, hvor værdien endnu ikke er returneret
  - en sluttilstand, hvor værdien er et positivt resultat
  - en sluttilstand, hvor værdien er en fejl eller exception

Promise - fortsat

- En promise kommer i en sluttilstand med kald af resolve(resultat), reject(fejl) eller ved en exception
- En promise kan tilknyttes eventHandlere med metoderne:
  - `then(onResultat)`
  - `catch(onFejl)`
- Metoderne `then()` og `catch()` returnerer eventHandlerens returværdi som en promise
- Hvis returværdien ikke allerede er en promise, genereres den med `Promise.resolve(result)` hhv. `Promise.reject(fejl/exception)`
- Da der returneres en promise, kan metoderne sammenkædes – eks. nogle then(…) efterfulgt af en enkelt catch(…)

```
// promise1.js
function langvarig(nr) {
    return new Promise(function (resolve, reject) {
        let type = Math.trunc(Math.random() * 3);
        if (type === 0) throw new Error(`${nr} Slet ikke OK!`);
        setTimeout(afslut, 500);
        function afslut() {
            if (type === 1) resolve(`${nr} OK!`);
            else if (type === 2) reject(`${nr} Ikke OK!`);
        }
    });
}

langvarig(1)
    .then(resultat => console.log('Resultat: ' + resultat))
    .catch(fejl => console.log('Fejl/exception: ' + fejl));

console.log('Efter kald af langvarig()');
```

```
// promise2.js
function langvarig(nr) {...}

langvarig(1)
    .then(resultat => {
        console.log('Resultat: ' + resultat);
        return langvarig(2);
    })
    .then(resultat => console.log('og ' + resultat))
    .catch(fejl => console.log('Fejl/exception: ' + fejl));
```

Promise.all()

- Flere asynkrone funktioner kan udføres parallelt
- Promise.all([promise, …]) returnerer en promise, hvis resultat er et array med de enkelte promises resultater – i same rækkefølge
- Hvis én promise fejler, returneres i stedet en promise med denne fejl

```
// promiseAll.js
function langvarig(nr) {...}

Promise.all([langvarig(1), langvarig(2)])
    .then(resultater => console.log('Resultater: ' + resultater))
    .catch(fejl => console.log('Fejl/exception: ' + fejl));
```

Promise.allSettled()

- Flere asynkrone funktioner kan udføres parallelt til alle har nået en sluttilstand
- Promise.allSettled([promise, ...]) returnerer en promise, hvis resultat er et array med de enkelte promises resultat, fejl eller exception

```
// promiseAllSettled.js
function langvarig(nr) {...}

Promise.allSettled([langvarig(1), langvarig(2)])
    .then((resultat) => {
        console.log('Resultater, fejl og exceptions:');
        console.log(resultat[0]);
        console.log(resultat[1]);
});
```

Promise.any()

- Som det modsatte af `Promise.all()`
- Promise.any([promise, ...]) returnerer en promise, hvis resultat er et array med de enkelte promises resultat, fejl eller exception

```
// promiseAllSettled.js
function langvarig(nr) {...}

Promise.all([langvarig(1), langvarig(2)])
    .then((resultat) => {
        console.log('Resultater, fejl og exceptions:');
        console.log(resultat); // langvarig 1 eller langvarig 2
});
```

Promise – i asynkrone API'er

- Da callbacks er kompliceret at bruge korrekt, er asynkrone funktioner nu i stort omfang omskrevet til at returnere en promise eller evt. en thenable
- En thenable er et objekt, der ikke er en Promise, men alligevel har en then metode – dvs. det er et promise-like objekt

async funktion

- En kæde af promises/eventHandlere kan dog også være kompliceret at gennemskue
- Som alternativ kan en asynkrone funktion, der returnerer en promise, kaldes med et foranstillet await
- await betyder, at værdien af promisen returneres, når den er klar – og at programmet først derefter fortsætter
- Dermed kan et program med asynkrone funktioner skrives sekventielt
- Det kræver dog, at koden er placeret i en async funktion
  – der i øvrigt returnerer en promise med funktionens returværdi
- Fejl og exceptions skal nu fanges i en try/catch sætning

```
// async2.js
function langvarig(nr) {...}

async function main() {
    try {
        let resultat = await langvarig(1);
        console.log('Resultat: ' + resultat);
        resultat = await langvarig(2);
        console.log('og ' + resultat);
    }
    catch (e) {
        console.log('Fejl/exception: ' + e);
    }
}
main();
```

```
// asyncAll.js
function langvarig(nr) {...}

async function main() {
    try {
        let resultater = await Promise.all([langvarig(1), langvarig(2)]);
        console.log('Resultater: ' + resultater);
    }
    catch (e) {
        console.log('Fejl/exception: ' + e);
    }
}
main();
```

```
// asyncAllSettled.js
function langvarig(nr) {...}

async function main() {
    let resultat = await Promise.allSettled([langvarig(1), langvarig(2)]);
    console.log('Resultater, fejl og exceptions:');
    console.log(resultat[0]);
    console.log(resultat[1]);
}
main();
```