Opgaver – Promise (Ajax, fetch, Axios, await/async)

Opgave (Promise - 1)

Et JS promise kan være i 3 tilstande '*unresolved*' (pending), '*resolved*' og '*rejected*'. Under afvikling er promise i starttilstanden: '*unresolved*' og det skifter tilstand til: '*resolved*' ved kald af funktionen *resolve( )* (funktionen givet ved 1. parameter til promise()) og tilstanden: '*rejected*' ved kald af funktionen *reject( )*) (funktionen givet ved 2. parameter til promise()).

Når man opretter en ny promise og tildeler det til en variabel, vil denne automatisk få 2 properties: *then* og *catch*. *then* og *catch* kan gives calback-funktioner der automatisk kaldes når promise skifter tilstand fra '*unresolved'* til '*resolved'* (then-callbacks) og fra '*unresolved'* til '*rejected'* (catch-callbacks)

Givet følgende kode:

```
promise = new Promise((resolve, reject)  => {
    resolve(); // reject();
});

promise
    .then(()=>console.log('Im finish'))          //callback, udføres ved
resolved
    .then(()=>console.log('jeg blev også kaldt'))
    .catch(() => console.log('uh oh!!'))         //callback, udføres ved
reject
```

a) Copy/past koden til console i chromes devTool. Kør koden og observer hvad der sker! Clear console og prøv også med reject() i stedet for resolve().

b) Simuler at der laves et asynkront kald til en webservice ved at indsætte et setTimeout() kald, der kalder resolve() efter 3000ms.

Opgave (Promise – 2, Ajax)

Ajax (**A**synchronus **J**avascript **A**nd **X**ml) var den traditionelle metode til lave asynkrone opdateringer af en Wep-page. Der benyttes følgende trin:

1. Der sker en event på siden (fx click-event på en button)
2. Der oprettes et XMLHttpRequest objekt af JS-engine
3. XMLHttpRequest objektet sender et request til server (fx en webservice)
4. Server sender et response tilbage til web-page
5. Response modtages af JS-engine
6. JS-engine kalder Callback-funktionen (der typisk opdatere web-page)

I nedenstående eksempel oprettes et *Promise*, der benytter *XMLHttpRequest* (Ajax) til at lave et asynkront kald til en webservice. Bemærk *open()* der angiver typen af kald (her GET), samt url'en, *onload* propertien der tildeles den callback-funktion der kaldes når responset er klar, *onerror* propertien tildeles den callback-funktion der kaldes, hvis kaldet til webservice afvises/fejler og endeligt *send()* der laver det asynkrone kald til webservice:

```
promise = new Promise((resolve, reject) => {
  const request = new XMLHttpRequest();

  request.open('GET', 'https://api.icndb.com/jokes/random');
  request.onload = () => {
    if (request.status === 200) {
      resolve(request.response); // we got data here, so resolve the
Promise
    } else {
      reject(Error(request.statusText)); // status is not 200 OK, so
reject
    }
  };

  request.onerror = () => {
    reject(Error('Error fetching data.')); // error occurred, reject the
Promise
  };

  request.send(); // send the request
});

console.log('Asynchronous request made.');

promise.then((data) => {
```

```
  console.log('Got data! Promise fulfilled.');
  document.body.textContent = JSON.parse(data).value.joke;
}, (error) => {
  console.log('Promise rejected.');
  console.log(error.message);
});
```

a) Copy/past koden til console i chromes devTool. Kør koden og observer hvad der sker!

Opgave (Promise – 3, fetch)

Med ES2015 kom en ny og mere simple måde at lave asynkrone kald på.
Promise blev en del af standarden (dvs funktionen Promise blev native i JS-engine) og samtidigt kom funktionen fetch().

```
url = "https://jsonplaceholder.typicode.com/posts/";

fetch(url)
  .then(data => console.log(data));
```

Bemærk, vi får ikke umiddelbart data retur med fetch, kun et objekt der repræsentere det response der kommer fra serveren. Derfor skal vi kalde json() for at få vores data:

```
url = "https://jsonplaceholder.typicode.com/posts/";
fetch(url)
  .then(responce => responce.json())
  .then(data => console.log(data));
```

a) Copy/past koden til console i chromes devTool. Kør koden og observer hvad der sker!
b) Tilføj catch:

```
.catch (error => console.log('BAD', error))
```

Og prøv følgende:
1) udskift posts med posts12345
2) udskift typicode.com med typicon.dk
Se hvad der sker (måske det er derfor vi anbefaler brugen af axios i stedet ☺)

Opgave (Promise – 4, fetch)

a) Prøv at anvende fetch() til at hente informationer om formel 1 kører fra:
(https://ergast.com/api/f1/2018/drivers.json)

Opgave (Promise – 5, Axios med Promise)

a) Prøv at anvende Axios/Promise til at hente informationer om formel 1 kørerne.

Opgave (Promise – 6, Axios med async/await)

a) Prøv at anvende Axios med ES2017 async/await til at hente informationer om formel 1 kørerne.

Opgave (Promise – 7, Axios med async/await)

a) Prøv at anvende Axios med ES2017 async/await til at hente informationer fra TheMovieDb