 **Exercise – Promise (Ajax, fetch, Axios, await/async)

Exercise (Promise - 1)**A JS promise can be in 3 states ’*unresolved*’ (pending), ’*resolved*’ and ’*rejected*’. The promise will be in the initial state ’*unresolved*’ during the execution and it changes the state to ’*resolved*’ by calling the *resolve( )* function (the function given by 1st parameter to promise()) or to the state: ’*rejected*’ by calling the *reject( )* function (the function given by 2nd parameter for promise()).

When you create a new promise and assign it to a variable, it will automatically get 2 properties: *then* and *catch*. *Then* and *catch* can be given callback-functions which are automatically called when the promise switches state from ’*unresolved’* to ’*resolved’* (then-callbacks) and from ’*unresolved’* to ’*rejected’* (catch-callbacks)
 **Given the following code:**

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 the code to the console in chromes devTool. Run the code and observe what happens!? Clear the console and try also with reject() instead of resolve().

b) Simulate making an asynchronous call to a webservice by inserting a setTimeout() calls that calls the resolve() after 3000ms.

**Exercise (Promise – 2, Ajax)**Ajax (**A**synchronous **J**avascript **A**nd **X**ml) was the traditional method of doing asynchronous updates of a Wep-page. It involves the following steps:

1. An event occurs on the page (e.g. a click-event on a button)
2. An XMLHttpRequest object is created by the JS-engine.
3. The XMLHttpRequest object sends a Request to the server (e.g. a web service)
4. The Server sends back a Response to the web-page
5. The Response is received by the JS-engine
6. The JS-engine invoke the Callback-function (that typically update the web-page)

In the example below, a *Promise* is created that uses the *XMLHttpRequest* (Ajax) to make an asynchronous call to a web-service. Notice, the first argument to *open()* is the type of the call (here a GET call), and the second argument is the url. The *onload* property is assigned to the callback-function that is called when the Responce is ready. The *onerror* property is assigned to the callback-function that is called, if the call to the webservice is rejected or fails. The *Send()* makes the asynchronous call to the 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);

});

1. Copy/past the code into the Console of the devTool in Chrome. Run the code and observe what happens!?

**Exercise (Promise – 3, fetch)**With ES2015 came a new and simpler way to make asynchronous calls.
Promise became a part of the standard (ie the Promise function goes native in the JS-engine) and at the same time came the fetch() function.

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

fetch(url)

 .then(data => console.log(data));

Note, with fetch we don’t get the data back immediately, but only an object representing the Response from the server. That’s why we need to call json() to get our data:

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

fetch(url)

 .then(responce => responce.json())

 .then(data => console.log(data));

1. Copy/past the code into the Console of the devTool in Chrome. Run the code and observe what happens!?
2. Add the catch:

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

And try the following:
1) replace ’posts’ with ’posts12345’
2) replace ’typicode.com’ with ’typicon.dk’
See what’s happen (maybe that’s why we recommend using Axios instead ☺)

**Exercise (Promise – 4, fetch)**a) Try using fetch() to retrieve information about Formula 1 drivers from: (<https://ergast.com/api/f1/2019/drivers.json>)

**Exercise (Promise – 5, Axios with Promise)**a) Try using Axios/Promise to retrieve the information about the Formular 1 drivers.

**Exercise (Promise – 6, Axios with async/await)**a) Try using Axios with ES2017 async/await to retrieve the information about the Formular 1 drivers.

**Exercise (Promise – 7, Axios with async/await)**a) Try using Axios with ES2017 async/await await to retrieve information from TheMovieDb