Async-funktioner returnerar ALLTID en *promise*

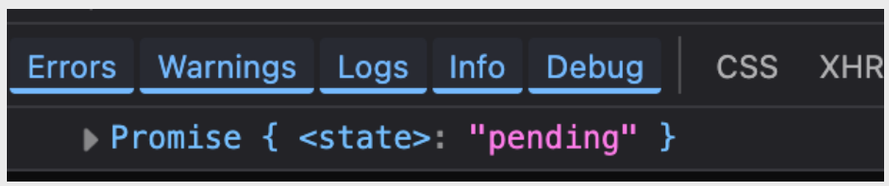
```
const request = new Request("http://localho

async function getResponse () {
  const response = await fetch(request);
  return response;
}
const x = getResponse();
console.log(x);
```

Det ser ut som att koden till vänster kommer att logga response-objektet.

MEN: async-funktioner returnerar ALLTID en *promise*.

Så det som kommer att loggas är:



Async-funktioner returnerar ALLTID en *promise*

```
const request = new Request("http://localho

async function getResponse () {
  const responsePromise = fetch(request);
  const response = await responsePromise;
  return response;
}
const x = getResponse();
console.log(x);
```

Steg för steg:

1) Förfrågan skapas och tilldelas konstanten request
2) Async-funktionen getResponse deklareras
3) getResponse anropas
    1) getResponse returnerar *direkt* en *promise*, som tilldelas konstanten x.
    2) getResponse börjar utföras
        1) fetch anropas
            1) Förfrågan skickas
            2) fetch returnerar en promise, som tilldelas till konstanten responsePromise.
        2) webbläsaren pausar exekveringen av getResponse, eftersom det finns en await.
4) Värdet av x loggas, det är en *promise* med status "väntande" (pending).
5) När responsePromise uppfylls (för att responsen från servern har kommit in) så tilldelas response-objektet till konstanten response.
6) return-instruktionen gör så att *promise*:n som finns i konstanten x löses till response-objektet (eftersom det är det som returneras)
    1) Men: x är fortfarande en promise! Inte response-objektet
    2) Dessutom loggades x tidigare, den loggas inte nu.

Hur når man responsen utanför getResponse?

Alternativ 1: Med then()

```
const request = new Request("http://localho

async function getResponse () {
  const responsePromise = fetch(request);
  const response = await responsePromise;
  return response;
}
const x = getResponse();
x.then(responseHandler);

function responseHandler(response) {
  console.log(response);
}
```

x är ju en *promise*.

Med hjälp av then-metoden kan vi ange en funktion som ska anropas när *promise* uppfylls.

Alltså precis som vi lärde oss i tidigare videos.

Men koden känns svårt att läsa... och det känns konstigt att blanda två olika syntax: async/await och then.

Hur når man responsen utanför getResponse?

Alternativ 2: Med async/await

```
const request = new Request("http://localho

async function getResponse () {
  const responsePromise = fetch(request);
  const response = await responsePromise;
  return response;
}
```

```
async function driverFunction () {
  const x = getResponse();
  const response = await x;
  console.log(response);
}
driverFunction();
```

Vi skapar en s.k. "driver function":

> *en driver function initierar och hanterar exekveringen av andra funktioner, särskilt i asynkrona sammanhang. Den ansvarar för att starta processer, vänta på deras slutförande och hantera resultaten.*

Om vi gör den till en async-funktion så kan vi använda await inuti den, och nu är koden mer läsbar.

Hur når man responsen utanför getResponse?

Alternativ 2: Med async/await

```
const request = new Request("http://localho

async function getResponse () {
  const responsePromise = fetch(request);
  const response = await responsePromise;
  return response;
}

(async function () {
  const x = getResponse();
  const response = await x;
  console.log(response);
})();
```

Driver-funktionen kan också kodas som en anonym funktion som anropas direkt.

Uttrycket som skapar den anonyma funktionen och anropar den direkt kallas "Immediately Invoked Function Expression" (IIFE).

Detta tillvägagångssätt är ganska vanligt och har en del fördelar, som dock ligger utanför kursens ramar.