"Vanlig" kod exekveras synkront

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

async function getResponse () {
  const responsePromise = fetch(request);
  console.log("before await");
  const response = await responsePromise;
  console.log(response);
}
getResponse();

for (let i = 0; i < 100; i++) {
  console.log("test1");
}

console.log("test2");
```

Viktigt att notera:

JS-kod exekveras alltid synkront, förutom vid specifika tillfällen. I denna kurs exekveras all kod synkront förutom när vi använder en *promise*.

Så alla rader som exekveras efter getResponse(); kommer att exekveras synkront (det finns ingen *promise* där).

Hade vi haft en jättelång loop som tog flera minuter att köra så måste den avslutas först, och alla rader under också. Även om responsen har kommit in så kommer webbläsaren att utföra all synkron kod först, innan den går tillbaka till await-instruktionen.

Async-funktioner exekveras synkront och asynkront

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

async function getResponse () {
  const responsePromise = fetch(request);
  console.log("before await");
  const response = await responsePromise;
  console.log(response);
}
getResponse();

for (let i = 0; i < 100; i++) {
  console.log("test1");
}

console.log("test2");
```

Viktigt att notera:

Async-funktionen getResponse exekveras synkront fram tills det att webbläsaren kommer till en await-instruktion.

Hade vi haft en jättelång for-loop i början av getResponse så hade den körts i sin helhet, även om det tog flera minuter. Webbläsaren reagerar inte på användarens handlingar.

När webbläsaren kommer till await-instruktionen så avstannas exekveringen av getResponse, men webbläsaren fortsätter exekvera raderna efter anropet, alltså efter instruktionen getResponse()

Det är därför man säger att koden exekveras asynkront då, för att webbläsaren kan utföra andra instruktioner (och reagera på användarens handlingar).