JavaScript ohjelmointi
Asynkronisuus

MARGIT TENNOSAARIN MATERIAALISTA
LAURA JÄRVISEN MUOKKAAMA

Mikä asynkronisuus?

- Asynkroninen JavaScript-koodi tarkoittaa sitä, että koodin osia suoritetaan taustalla ilman, että koko ohjelma “jää odottamaan” niiden valmistumista.
- Tämä on tärkeä sellaisten prosessien yhteydessä, jotka vievät aikaa esimerkiksi ajastimet ja verkkopyynnöt.
- **Estää käyttöliittymän jumittumisen** ja useamman tehtävän suorittamisen samaan aikaan.

Lupaukset
Promises

Promise

- Promise on olio, joka edustaa asynkronisen toiminnon lopullista valmistumista (onnistuminen tai epäonnistuminen).
- Toimintoja liitetään tapahtumaan lupauksen jälkeen.
- Promisella on kolme tilaa:
  - Pending eli kesken
  - Fulfilled eli toteutettu
  - Rejected eli epäonnistunut

Promisen käyttö

- Lupauksen jälkeinen toiminto eli funktio voidaan liittää promiselle kolmella tapaa:
  - .then() onnistumiseen
  - .catch() epäonnistumiseen
  - .finally() joka suoritetaan aina joka tapauksessa

```
fetch("https://api.example.com")
  .then((response) => response.json()) // Käsittelee onnistumisen
  .catch((error) => console.error(error)) // Käsittelee virheen
  .finally(() => console.log("Toiminto valmis")); // Suoritetaan aina
```

Async / Await

Uudempi tapa käsitellä lupauksia

- Async/Await on modernimpi tapa käsitellä promiseja:
  - **async**-avainsana funktiota määritellessä tekee siitä asynkronisen ja palauttaa automaattisesti promisen.
  - **await**-avainsana keskeyttää funktion suorittamisen, kunnes promise on ratkaistu.
- Koodirivit etenevät ylhäältä alas ilman .then()-ketjuja.
- **try...catch** on suoraan käytössä, eikä virheitä tarvitse käsitellä erikseen .catch()-ketjussa.

Miten?

- Ensin määritellään, mitä funktion halutaan yrittävän tehdä eli **try**.
- **await** lisätään kohtiin, jotka saattavat viedä aikaa.
- **catch** nappaa mahdolliset virheet.

```
async function fetchData() {
  try {
    let response = await fetch("https://api.example.com/data");
    let data = await response.json();
    console.log(data);
  } catch (error) {
    console.error("Virhe:", error);
  }
}
```

Kumpaa käyttää?

- **Promises:** hyvä yksinkertaisiin, erillisiin asynkronisiin kutsuihin tai tilanteisiin, joissa halutaan ketjuttaa useita toimintoja
- **Async/await:** parempi, kun haluat luettavamman tavan käsitellä monimutkaisempia prosesseja, kuten sisäkkäisiä lupauksia tai laajaa virheenkäsittelyä.

Callbackit

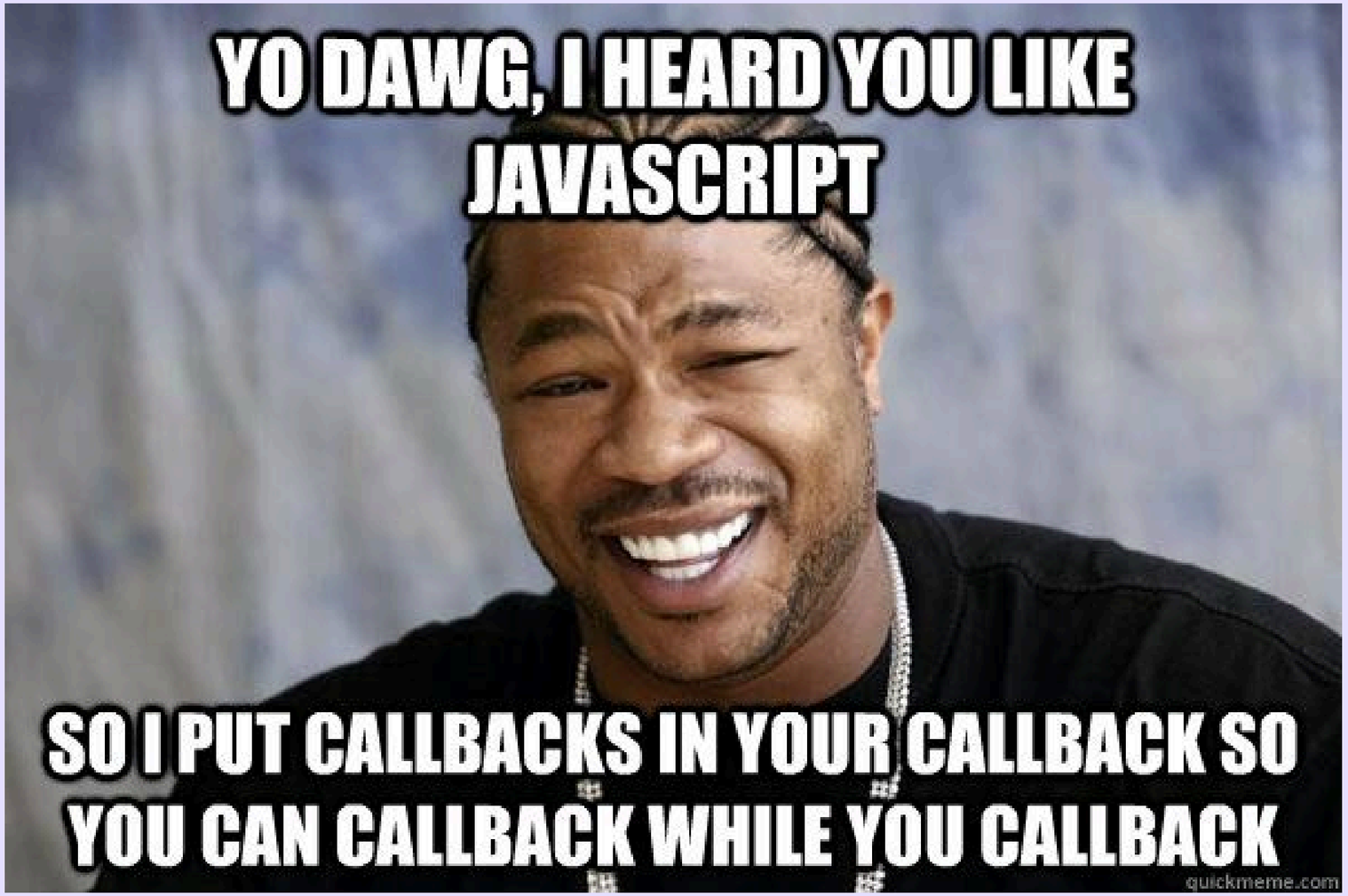
Vanha tapa asynkronisuuteen

- Callback on funktio, joka siirretään toiselle funktiolle argumenttina

```
function fetchData(callback) {
  setTimeout(() => {
    callback("Data haettu");
  }, 1000);
}

fetchData((data) => console.log(data));
```

Callbackit aiheuttavat ongelmia



Callback helvetti

Tapahtuu, kun palautefunktioita laitetaan sisäkkäin mikä tekee koodista vaikealukuisen ja vaikeasti ylläpidettävän. Tätä tapahtuu, kun useat asynkroniset toiminnot ovat riippuvaisia edellisistä.

```
getData(function (a) {
  getMoreData(a, function (b) {
    getMoreData(b, function (c) {
      getMoreData(c, function (d) {
        console.log(d);
      });
    });
  });
});
```

Pura ongelma:

Lupaukset:

```
getData()
  .then((result) => getMoreData(result))
  .then((result) => getMoreData(result))
  .then((result) => console.log(result))
  .catch((error) => console.error(error));
```

Async/Await

```
async function processData() {
  try {
    let result = await getData();
    result = await getMoreData(result);
    console.log(result);
  } catch (error) {
    console.error(error);
  }
}

processData();
```

Datan hakeminen

- Ehkä yleisin asynkroninen toiminto on datan haku API-rajapinnoista.
- **fetch** on moderni selainrajapinta HTTP-pyyntöjen tekemiseen ja vastauksen käsittelyyn. Se palauttaa aina Promise-olion, jonka avulla voit käsitellä vastauksen joko lupauksilla eli .then()/.catch()-ketjuilla tai async/awaitilla
- Lisää fetchin käytöstä Githubissa

Harjoitellaan

Löydät tehtäviä asynkronisuuden harjoitteluun viikon 9 kansiosta

https://github.com/bc-web-ohjelmistokehitys/WP25K-JS

Huom! Opettele itse - älä pyydä tekoälyä ratkaisemaan tehtävää puolestasi.