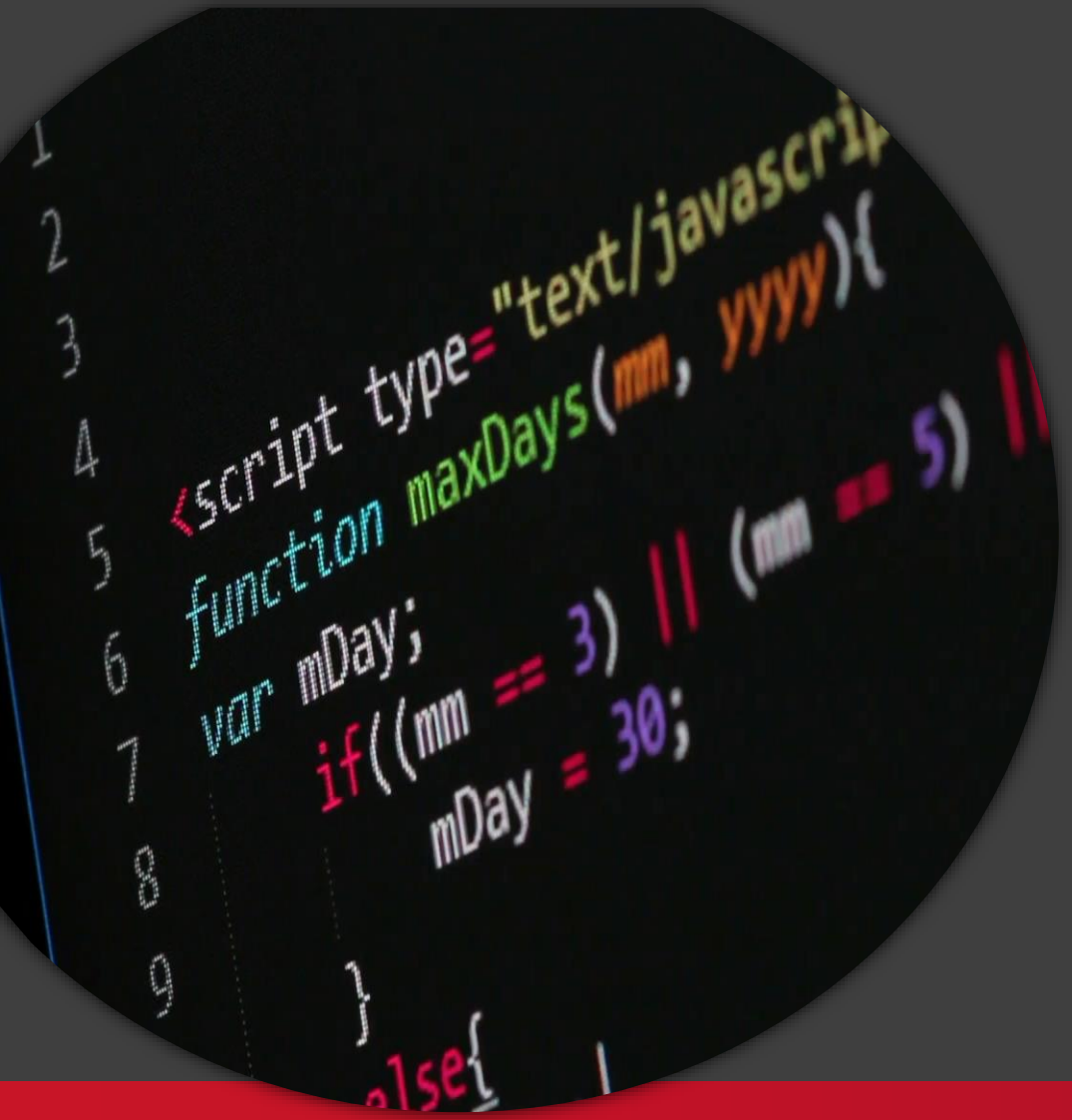


Promise

Kliensalkalmazások

BME AUT
Automatizálási és
Alkalmazott
Informatikai Tanszék

Gincsai Gábor
gincsai@aut.bme.hu

A JavaScript alapvetően aszinkron.

```
function loadScript(src) {
  let script = document.createElement('script');
  script.src = src;
  document.head.append(script);
}

loadScript('/my/script.js'); // sayHello() impl.
sayHello();   // Error: nincs ilyen fv??
```

Callback

```
function loadScript(src, callback) {
  let script = document.createElement('script');
  script.src = src;
  script.onload = () => callback(script);
  document.head.append(script);
}

loadScript('/my/script.js', function() {
    sayHello(); // Működik, a callback betötés után fut.
});
```

Ha sok a callback....

```
loadScript('1.js', function(error, script) {
  if (error) {
    handleError(error);
  } else {
    // ...
    loadScript('2.js', function(error, script) {
      if (error) {
        handleError(error);
      } else {
        // ...
        loadScript('3.js', function(error, script) {
          if (error) {
            handleError(error);
          } else {
            // ...
          }
        });
      }
    })
  }
});
```

Forrás: https://javascript.info/callbacks

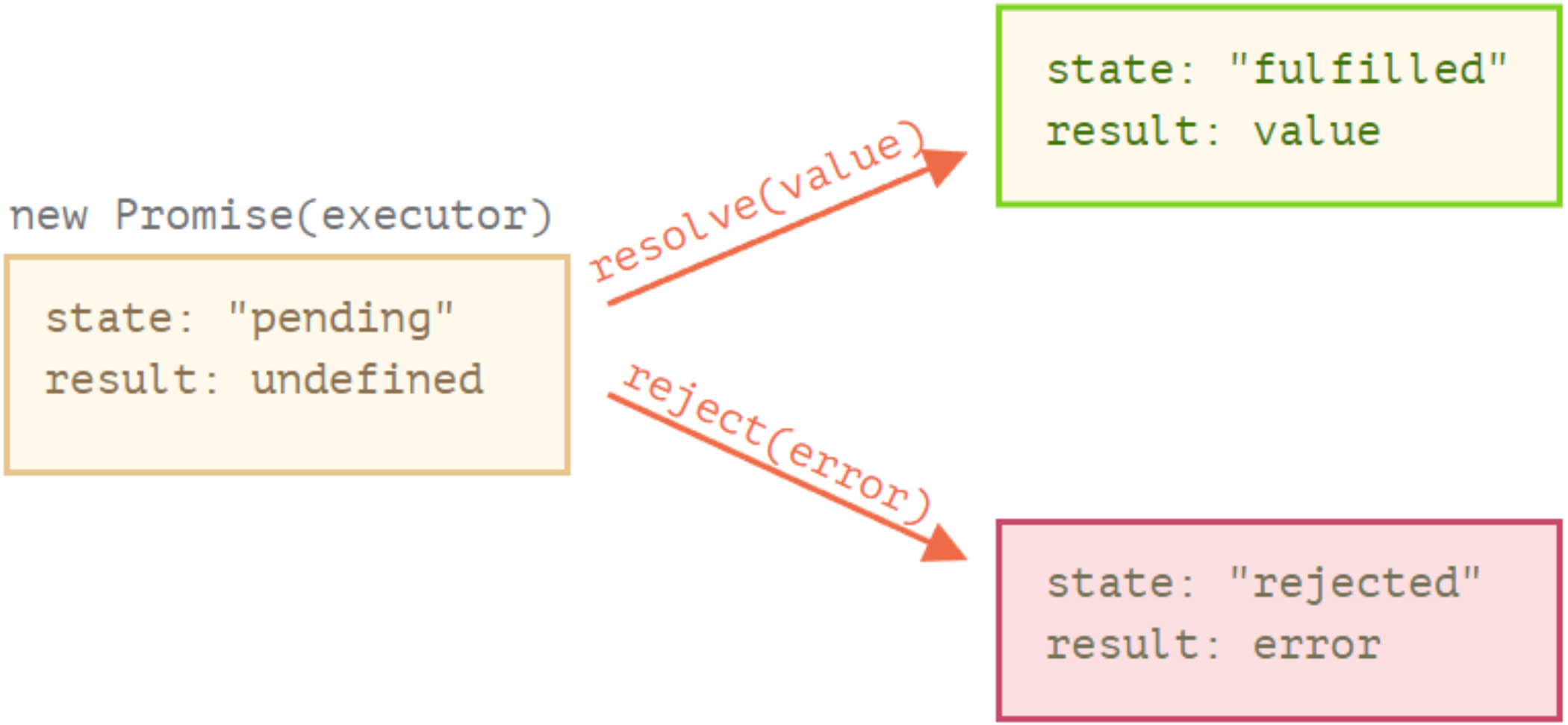
Promise

- A Promise egy olyan objektum, ami majd a jövőben visszaad egy értéket, de nem most.
- E miatt tökéletes aszinkron kérések kezelésére.
- Három állapota van
  - Pending – függőben van
  - Fulfilled – sikeresen lefutott
  - Rejected – hibára futott
- A promise mindig Pendig állapotból indul és Fulfilled vagy Rejected állapotban ér véget.

Promise példa

```
let completed = true;

let learnWeb = new Promise(function (resolve, reject) {
    if (completed) {
        resolve("I have completed learning Web.");
    } else {
        reject("I haven't completed learning Web yet.");
    }
});
```

Promise állapot átmenetei



Promise eredményének feldolgozása

- .then(success, error)
  - Ha lefutott a Promise akkor hívódik meg.
  - Ha sikeresen futott le akkor a success handler hívódik
  - Ha sikertelenül akkor az error handler (opcionális)
- .catch(f)
  - Csak akkor fut le ha a Promise hibával tért vissza
- .finally(f)
  - Minden esetben lefut, ha sikeres ha sikertelen a Promise, de az eseménykezelőben nem tudjuk eldönteni, hogy sikeresen vagy sikertelenül futott le.
  - Tipikusan takarításra használjuk. Pl.: loading indikátor eltűntetése.

Promise példa (then, catch, finally)

```
learnWeb.then(
      result => alert(result);
      error => alert(error);
);

learnWeb.catch(alert);

learnWeb.finally( () => /* Stop loading */ )
```

loadScript példa Promise használatával

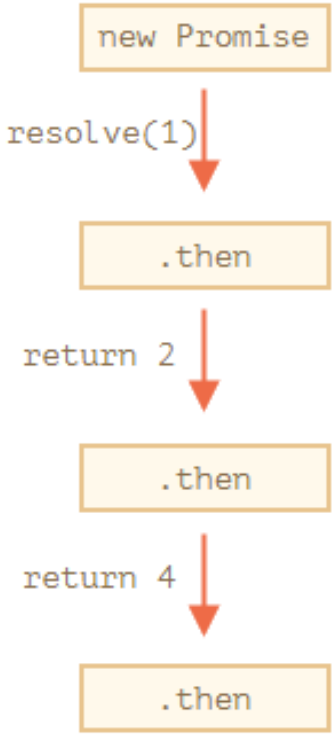
```
function loadScript(src) {
  return new Promise(function(resolve, reject) {
    let script = document.createElement('script');
    script.src = src;

    script.onload = () => resolve(script);
    script.onerror = () => reject(new Error(`Error: ${src}`));

    document.head.append(script);
  });
}
```

loadScript példa Promise használatával

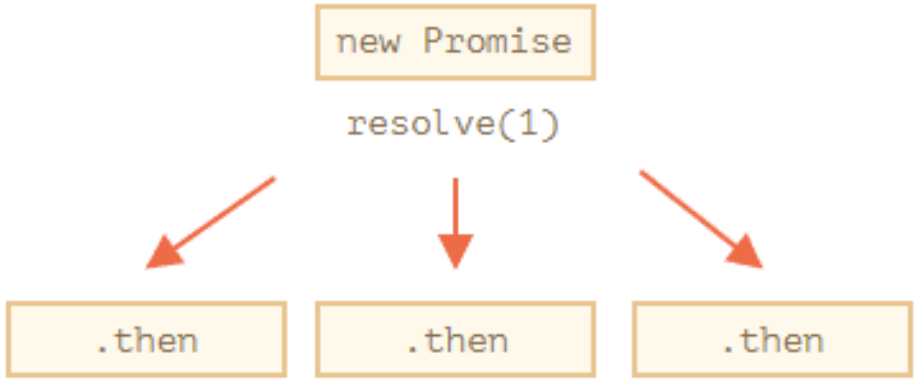
```
let promise = loadScript("https://cdnjs.cloudflare.com/ajax/
                          libs/lodash.js/4.17.11/lodash.js");

promise.then(
  script => alert(`${script.src} is loaded!`),
  error => alert(`Error: ${error.message}`)
);

promise.then(script => alert('Another handler...'));
```

Promise chain

```
promise.then(...).then(...).then(...)
```



A fenti kód nem azonos azzal,
ha a promiseon három thent hívunk.




```
promise.then(...);
promise.then(...);
promise.then(...);
```

Promise API

- Promise.all(promises)
  - Megvárja, hogy az összes Promise befejeződjön.
  - Ha bármelyik hibával ér véget, akkor az egész hibával tér vissza.
- Promise.allSettled(promises)
  - Megvárja, hogy az összes Promise befejeződjön
  - Visszaadja, hogy melyik volt sikeres és melyik hibás.
    - status: „fulfilled” vagy „rejected”
    - value ha sikeres, reason ha sikertelen.
- Promise.race(promises)
  - Csak az első promise-t várja meg és annak eredményét adja vissza.
- Promise.any(promises)
  - Az első sikeresen befejeződött Promise-ra vár.
  - Ha mindegyik sikertelen akkor AggregateErrort ad vissza.

async / await

- Segítségével kényelmesebben kezelhetjük az Promise-okat.
- A függvény előtt lévő async azt jelenti, hogy a függvény egy Promise-sal tér vissza.
  - Ha nem promise-sal térne vissza a függvény, akkor becsomagolja egy promise-ba.

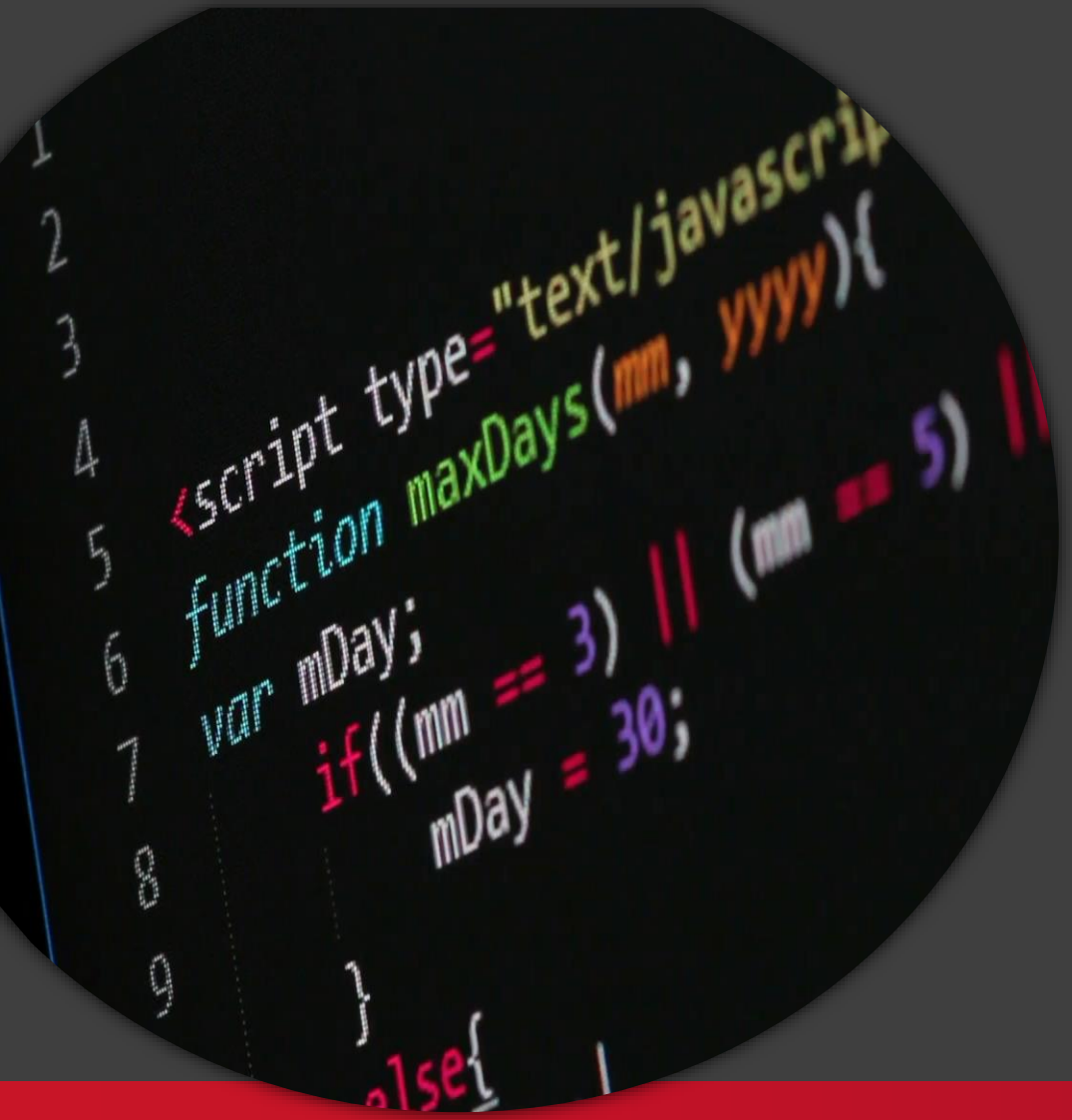
```
async function f() {
  return 1;
}
```

```
async function f() {
  return Promise.resolve(1);
}
```

```
f().then(alert); // 1
```

await

```
async function f() {
  let promise = new Promise((resolve, reject) => {
    setTimeout(() => resolve("Kész!"), 1000)
  });

  let result = await promise; // Vár a promise-ra
  alert(result); // "Kész!"
}
f();
```



Fetch API

Kliensalkalmazások

BME AUT
Automatizálási és
Alkalmazott
Informatikai Tanszék

Gincsai Gábor
gincsai@aut.bme.hu

Fetch API

- A fetch() segítségével hálózati kéréseket küldhetünk a szerver felé.
- Korábban ezeket a kéréseket az XmlHttpRequest segítségével lehetett megoldani,
- A Fetch API a modern böngészőkben megtalálható, polyfill is van a régebbi böngészőkhöz.
- Támogatja a CORS-t tehát tetszőleges szerver felé indíthatunk kéréseket.
- Promise-sal tér vissza.

Fetch API

- A Promise csak akkor reject-elődik ha hálózati hiba van.
- A HTTP státuszkódtól függetlenül sikeresen tér vissza, ha a szerver válaszolt.
  - A válaszban van egy Ok tulajdonság, ami jelzi, hogy a kérés sikeres-e
    - 200-299 közötti státuszkódokra true-ra állítja, egyéb esetben false-ra.
  - Cross-origin sütiket nem fogja elküldeni.

```
let promise = fetch(url, [options])
```

Fetch API

- A böngésző azonnal elindítja a kérést és egy Promise-t ad vissza, amiből majd az eredmény el lehet érni.
- A választ két lépésben lehet kinyerni.
  1. A fetch Promise a beépített Response osztályt adja vissza, amiben a szervertől visszakapott Header-ök találhatók.
     - Status: HTTP státuszkód.
     - Ok: Sikeres-e a kiszolgálás (Státusz kód 200-299)
  2. A response-ból a body-t egy újabb promose-sal kapjuk meg amit pl a json() függvény segítségével lehet elérni.

Fetch API példa

```
let response = await fetch(url);

if (response.ok) { // HTTP-status 200-299
  // Response body kinyerése
  let json = await response.json();
} else {
  alert("HTTP-Error: " + response.status);
}
```

Fetch API - Post

```
let user = { name: 'John', surname: 'Smith' };

let response = await fetch('/article/fetch/post/user', {
    method: 'POST',
    headers: {
      'Content-Type': 'application/json;charset=utf-8'
    },
    body: JSON.stringify(user)
});

let result = await response.json();
alert(result.message);
```