Դաս 10: Callback հիմունքներ և ասինխրոնություն

Callback ֆունկցիան ֆունկցիա է, որը փոխանցվում է որպես պարամետր այլ ֆունկցիայի, և կանչվում է որևէ գործողության ավարտից հետո:

```
function downloadFile(fileName, callback) {
  console.log(`Ներբեռնում եմ ${fileName} ֆայլը...`);
  setTimeout(() => {
    console.log(`${fileName} ֆայլը ներբեռնված է`);
    callback(); // Կանչ callback-ը ներբեռնումից հետո
  }, 2000);
}

function processFile() {
  console.log("Ֆայլի մշակումը սկսվեց:");
}

downloadFile("example.txt", processFile);
```

Առաջին հայացքից կարող է թվալ, թե նման մոտեցումը բավականաչափ պարզ է, սակայն եթե **callback** ֆունկցիաների քանակը մեկը մյուսի մեջ շատանում են, ապա կոդը դառնում է քաոսային:

```
function step1(callback) {
  setTimeout(() => {
    console.log("Քայլ 1");
    callback();
  }, 1000);
}

function step2(callback) {
```

```
  setTimeout(() => {
    console.log("Քայլ 2");
    callback();
  }, 1000);
}

function step3(callback) {
  setTimeout(() => {
    console.log("Քայլ 3");
    callback();
  }, 1000);
}

step1(() => {
  step2(() => {
    step3(() => {
      console.log("Բոլոր քայլերը ավարտված են:");
    });
  });
});
```

Նման խնդիրների լուծման համար կարող ենք օգտագործել **Promise**-ներ՝

Promise-ը օբյեկտ է, որը ներկայացնում է գործողություն, որը կամ կավարտվի հաջողությամբ, կամ կձախողի ապագայում: **Promise**-ները լուծում են **քառսային կոդը** խնդիրը:

Promise-ը ստեղծվում է **new Promise** կոնստրուկտորի միջոցով: Այս կոնստրուկտորը որպես արգումենտ ստանում է **executor** կոչվող ֆունկցիա, որն ինքն իր հերթին ստանում է երկու արգումենտ՝

resolve - կանչվում է, երբ գործողությունը հաջողությամբ ավարտվում է:

reject - կանչվում է, երբ գործողությունը ձախողվում է:

```
const myPromise = new Promise((resolve, reject) => {
    // Օրինակ ասինխրոն գործողություն
    let success = true;
// Կայացած տվյալ, որը կարող է փոփոխվել

    if (success) {
        resolve("Գործողությունը հաջողվեց!");
    } else {
        reject("Գործողությունը ձախողվեց...");
    }
});
```

Ինչու են պետք resolve և reject-ը

resolve(value)

Կանչվում է, երբ գործողությունը հաջողությամբ ավարտվում է: Որպես արգումենտ այն ստանում է արժեքը (օրինակ՝ տվյալներ սերվերից), որը հետագայում կօգտագործվի:

reject(reason)

Կանչվում է, երբ գործողությունը ձախողվում է: Որպես արգումենտ այն փոխանցում է սխալի մասին տեղեկություն (օրինակ՝ սխալի հաղորդագրություն կամ օբյեկտ):

Promise-ի վերամշակում

Promise-ը կարող է ունենալ երեք հիմնական վիճակ.

Pending (**սպասող**) — Սկզբնական վիճակ:

Fulfilled (**կատարված**) — Երբ կանչվում է **resolve():**

Rejected (**մերժված**) — Երբ կանչվում է **reject():**

Օգտագործվում են **.then** և **.catch** մեթոդները՝ համապատասխանաբար հաջող և սխալի դեպքում:

```
myPromise
  .then((result) => {
    console.log("Հաջողության արդյունք:", result);
  })
```

```
  .catch((error) => {
    console.error("Սխալի հաղորդագրություն:", error);
  });
```

Օրինակ՝ պատկերացնենք, որ ասինխրոն գործողությունը ստուգում է ֆայլի առկայությունը:

```
const checkFile = new Promise((resolve, reject) => {
  let fileExists = false; // Փորձեք փոխել true և false

  if (fileExists) {
    resolve("Ֆայլը հայտնաբերվեց:");
  } else {
    reject("Ֆայլը բացակայում է:");
  }
});

checkFile
  .then((message) => {
    console.log("Հաջողություն:", message);
  })
  .catch((error) => {
    console.error("Սխալ:", error);
  });
```

Բացի այդ կարող ենք օգտագործել **.finally()** մեթոդը, որը կատարում է նույն ֆունկցիոնալը ինչպես **switch/case**-ում:
Promise-ները կարելի է կանչել շղթայաբար՝

```
function step1() {
  return new Promise((resolve) => {
    setTimeout(() => {
      console.log("Քայլ 1");
      resolve();
    }, 1000);
```

```
  });
}

function step2() {
  return new Promise((resolve) => {
    setTimeout(() => {
      console.log("Քայլ 2");
      resolve();
    }, 1000);
  });
}

function step3() {
  return new Promise((resolve) => {
    setTimeout(() => {
      console.log("Քայլ 3");
      resolve();
    }, 1000);
  });
}

step1()
.then(step2)
.then(step3)
.then(() => console.log("Բոլոր քայլերը ավարտված են:"));
```

async/await-ը **JavaScript**-ում **Promise**-ների հետ աշխատելու ավելի պարզ և հարմարավետ եղանակ է, որը հեշտացնում է **asyncronous** (անսինխրոն) կոդի ընթերցումը ու վերլուծությունը: Այս մեթոդը թույլ է տալիս գրել կոդ, որը նման է սինխրոն կոդի կառուցվածքին, բայց իրականում դեռ աշխատում է **asyncronously:**

async ֆունկցիա

async բառը ֆունկցիայի առաջ ավելացնելու դեպքում, այն ավտոմատ կերպով վերածվում է **Promise**-ի՝ անկախ նրանից՝ ֆունկցիան վերադարձնում է արժեք, թե ոչ: Եթե **async** ֆունկցիան վերադարձնում է արժեք, ապա այդ արժեքը պահվում է **Promise**-ի մեջ:

```
async function example() {
  return "Արդյունք";
// Սա համարվում է Promise.resolve('Արդյունք')
}

example().then(console.log); // 'Արդյունք'
```

Այս կոդում **async** ֆունկցիան վերադարձնում է **'Արդյունք'**, սակայն այն վերածվում է **Promise**-ով: **then()** մեթոդը սպասում է **Promise**-ի կատարման ավարտին և առաջացնում է արդյունքը:

await բառը օգտագործվում է **async** ֆունկցիաների մեջ, որպեսզի սպասի **Promise**-ի ավարտին: Երբ օգտագործվում է **await**, **JavaScript**-ը ժամանակավորապես կասեցնում է կոդի կատարումը՝ մինչև **Promise**-ը չավարտվի:

```
function delay(ms) {
   return new Promise((resolve) => setTimeout(resolve,
ms)); // 2 վայրկյան սպասում
}

async function main() {
  console.log("Սկիզբ");
  await delay(2000); // Սպասում է 2 վայրկյան
  console.log("Վախտս");
}

main();
```

Այս կոդում **await delay(2000)** կասեցնում է **main** ֆունկցիայի կատարումը **2** վայրկյանով: Այսինքն՝ **'Սկիզբ'** տեքստը կհայտնվի կոնսոլում, ապա միայն 2 վարկյան անց կհայտնվի **'Ավարտ'**-ը:

Try/Catch async/await-ի հետ

Async/await-ի գործառույթների դեպքում հեշտ է աշխատել **try/catch** բլոկների հետ՝ սխալների ուղղման համար: Եթե **async** ֆունկցիայի ընթացքում որևէ սխալ է տեղի ունենում, ապա այն կարող է բռնվել **catch** բլոկով:

```
async function fetchData() {
  try {
    const data = await Promise.reject("Սխալ");
// Իմիտացնում է սխալ
    console.log(data); // Այս մասը չի կատարվի
  } catch (error) {
    console.error("Սխալ՝", error); // Կտպի 'Սխալ՝ Սխալ'
  }
}

fetchData();
```

Այս կոդում, **await**-ը սպասում է այնքան ժամանակ, քանի դեռ **Promise**-ից չի ստացել պատասխան: Այս օրինակում մենք փոխանցել ենք **Promise.reject()**-ին: Այն արտադրել է սխալ, որը բռնվել է **catch** բլոկով: Սա թույլ է տալիս հեշտությամբ կառավարել սխալները և ճիշտ վերաբերվել նրանց:

Async/Await-ի առավելությունները

- ☐ **Կոդի պարզություն՝ async/await**-ը կոդը դարձնում է շատ ավելի ընթեռնելի և նման է սինխրոն կոդին:

- ☐ **Սխալների հեշտ կառավարման հնարավորություն՝** **try/catch** բլոկներով հեշտ է սխալներ բռնել ու կառավարել:
- ☐ **Կոնկրետ արդյունքների սպասում՝** await-ը թույլ է տալիս գրել կոդ, որը սպասում է միանգամից արդյունքին, առանց անհանգստանալու, թե երբ կամ ինչպես կավարտվի **Promise**-ը: