Resurse suplimentare
pe care recomand să le parcurgi

Ce este codul sincron vs. asincron?

Codul Sincron

În execuția **sincronă**, instrucțiunile sunt executate una după alta, în ordine. Fiecare instrucțiune așteaptă finalizarea celei anterioare înainte de a se executa.

```
console.log('Prima instrucțiune')
console.log('A doua instrucțiune')
console.log('A treia instrucțiune')
```

Rezultatul este previzibil:

```
Prima instrucțiune
A doua instrucțiune
A treia instrucțiune
```

Codul Asincron

În execuția **asincronă**, anumite operațiuni pot fi "amânate" pentru a fi executate mai târziu, permițând programului să continue execuția altor instrucțiuni.

```
console.log('Înainte de setTimeout')

setTimeout(() => {
  console.log('În interiorul setTimeout (după 2 secunde)')
}, 2000)

console.log('După setTimeout')
```

Rezultatul este:

```
Înainte de setTimeout
După setTimeout
În interiorul setTimeout (după 2 secunde)
```

Event Loop: Inima JavaScript-ului Asincron

Pentru a înțelege cum funcționează asincronitatea în JavaScript, trebuie să cunoaștem conceptul de **Event Loop**.

JavaScript are:

- Un **Call Stack** (stivă de apeluri) - unde sunt executate funcțiile sincrone
- Un **Task Queue** (coadă de sarcini) - unde sunt plasate callback-urile evenimentelor asincrone
- **Web APIs** (furnizate de browser) - care gestionează operațiunile asincrone

Procesul:

1. Funcțiile sincrone sunt adăugate și executate în Call Stack
2. Operațiunile asincrone sunt delegate către Web APIs
3. Când sunt gata, Web APIs le plasează în Task Queue
4. Event Loop verifică dacă Call Stack-ul este gol și, dacă da, mută prima sarcină din Task Queue în Call Stack pentru execuție

Callbacks: Prima soluție pentru asincronicitate

Un **callback** este o funcție care este pasată ca argument altei funcții și este executată după ce acea funcție își termină execuția.

Exemplu simplu de callback

```
function greetUser(name, callback) {
  console.log(`Salut, ${name}!`)
  callback()
}

function showAdditionalMessage() {
  console.log('Bine ai venit la cursul nostru de JavaScript!')
}

greetUser('Ana', showAdditionalMessage)
```

Output:

```
Salut, Ana!
Bine ai venit la cursul nostru de JavaScript!
```

Callbacks pentru operațiuni asincrone

Iată un exemplu cu `setTimeout`:

```
function downloadData(url, callback) {
  console.log(`Începe descărcarea de la ${url}`)

  // Simulăm o cerere asincronă cu setTimeout
  setTimeout(() => {
    const data = { users: ['Ion', 'Maria', 'Alex'] }
    console.log('Date descărcate cu succes')
    callback(data)
  }, 2000)
}

function processData(data) {
  console.log('Procesare date...')
  console.log(`Am primit ${data.users.length} utilizatori.`)
  console.log(`Utilizatori: ${data.users.join(', ')}`)
}

downloadData('<https://api.exemplu.ro/utilizatori>', processData)
console.log('Se continuă execuția programului...')
```

Output:

```
Începe descărcarea de la <https://api.exemplu.ro/utilizatori>
Se continuă execuția programului...
Date descărcate cu succes
Procesare date...
Am primit 3 utilizatori.
Utilizatori: Ion, Maria, Alex
```

Callback Hell: Problema înlănțuirii callback-urilor

Callback-urile sunt simple, dar când avem nevoie să executăm mai multe operațiuni asincrone în secvență, codul poate deveni greu de citit și întreținut - o situație cunoscută drept **"Callback Hell"** sau **"Pyramid of Doom"**:

```
downloadUsers(
  (users) => {
    downloadProducts(
      (products) => {
        downloadOrders(
          (orders) => {
            downloadReviews(
              (reviews) => {
                // Procesează toate datele
                // Codul este foarte indentat și greu de urmărit
              },
              (reviewError) => {
                console.error('Eroare la descărcarea recenziilor:', reviewError)
              }
            )
          },
          (orderError) => {
            console.error('Eroare la descărcarea comenzilor:', orderError)
          }
        )
      },
      (productError) => {
        console.error('Eroare la descărcarea produselor:', productError)
      }
    )
  },
  (userError) => {
    console.error('Eroare la descărcarea utilizatorilor:', userError)
  }
)
```

Gestionarea erorilor cu callbacks

Gestionarea erorilor în callback-uri se face de obicei folosind modelul **error-first callback**:

```javascript
function readFile(path, callback) {
  // Simularea unei operațiuni asincrone de citire fișier
  setTimeout(() => {
    const random = Math.random()

    if (random > 0.3) {
      // Operațiune reușită
      const content = 'Acesta este conținutul fișierului.'
      callback(null, content)
    } else {
      // Eroare
      callback(new Error('Nu s-a putut citi fișierul'), null)
    }
  }, 1000)
}

readFile('fisier.txt', (error, content) => {
  if (error) {
    console.error('A apărut o eroare:', error.message)
    return
  }

  console.log('Conținutul fișierului:', content)
})
```