Resurse suplimentare
pe care recomand să le parcurgi

Declararea funcțiilor

În JavaScript, există mai multe moduri de a defini funcții:

1. Declararea unei funcții (Function Declaration)

```
function greet(name) {
  return 'Salut, ' + name + '!'
}

// Apelarea funcției
console.log(greet('Ana')) // Afișează: Salut, Ana!
```

2. Expresii de funcții (Function Expression)

```
const greet = function (name) {
  return 'Salut, ' + name + '!'
}

console.log(greet('Mihai')) // Afișează: Salut, Mihai!
```

3. Funcții Arrow (ES6)

```
const greet = (name) => {
  return 'Salut, ' + name + '!'
}

// Sau forma simplificată, pentru funcții cu o singură expresie:
const shortGreet = (name) => 'Salut, ' + name + '!'

console.log(shortGreet('Elena')) // Afișează: Salut, Elena!
```

Parametri și argumente

Parametrii sunt variabilele declarate în definiția funcției, iar argumentele sunt valorile pe care le pasezi funcției când o apelezi.

```
// name și age sunt parametri
function describePerson(name, age) {
  return `${name} are ${age} ani.`
}

// "Maria" și 25 sunt argumente
console.log(describePerson('Maria', 25)) // Afișează: Maria are 25 ani.
```

Parametri impliciți (Default Parameters)

Poți seta valori implicite pentru parametri, care se vor folosi dacă nu sunt furnizate argumente.

```
function greet(name = 'vizitator') {
  return 'Salut, ' + name + '!'
}

console.log(greet()) // Afișează: Salut, vizitator!
console.log(greet('Ion')) // Afișează: Salut, Ion!
```

Rest Parameters

Folosind sintaxa `...` (rest operator), poți colecta mai multe argumente într-un array.

```
function sumNumbers(...numbers) {
  let total = 0
  for (let number of numbers) {
    total += number
  }
  return total
}

console.log(sumNumbers(1, 2, 3, 4, 5)) // Afișează: 15
```

Returnarea valorilor

Funcțiile pot returna valori folosind instrucțiunea `return`.

```
function add(a, b) {
  return a + b
}

const sum = add(5, 3)
console.log(sum) // Afișează: 8
```

Câteva aspecte importante despre `return`:

- Orice cod după `return` nu va fi executat
- Dacă nu există o instrucțiune `return`, funcția returnează implicit `undefined`
- O funcție poate returna orice tip de date, inclusiv obiecte, array-uri sau alte funcții

```
function createGreeting(name) {
  // Funcția returnează un obiect
  return {
    message: `Salut, ${name}!`,
    timestamp: new Date(),
  }
}

console.log(createGreeting('Alex').message) // Afișează: Salut, Alex!
```

Scope-ul variabilelor în funcții

Scope-ul se referă la contextul în care variabilele sunt accesibile.

Variabile locale

Variabilele declarate în interiorul unei funcții (cu `let`, `const` sau `var`) sunt vizibile doar în interiorul acelei funcții.

```
function scopeExample() {
  const localMessage = 'Aceasta este o variabilă locală'
  console.log(localMessage) // Funcționează
}

scopeExample()
// console.log(localMessage); // Eroare: localMessage is not defined
```

Variabile globale

Variabilele declarate în afara oricărei funcții sunt variabile globale și pot fi accesate din orice parte a codului.

```
const globalMessage = 'Aceasta este o variabilă globală'

function displayMessage() {
  console.log(globalMessage) // Funcționează
}

displayMessage()
console.log(globalMessage) // Funcționează și aici
```

Lexical Scope și Closure

JavaScript utilizează un concept numit "lexical scope", ceea ce înseamnă că funcțiile interioare au acces la variabilele din funcțiile exterioare.

```
function outer() {
  const message = 'Salut!'

  function inner() {
    console.log(message) // Poate accesa message din funcția outer
  }

  inner() // Apelează funcția interioară
}

outer() // Afișează: Salut!
```

Un closure se formează atunci când o funcție interioară este returnată din funcția exterioară și încă are acces la variabilele funcției exterioare, chiar după ce aceasta s-a încheiat.

```
function functionCreator() {
  const greeting = 'Salut, '

  return function (name) {
    return greeting + name
  }
}

const generatedFunction = functionCreator()
console.log(generatedFunction('Maria')) // Afișează: Salut, Maria
```

Funcții anonime

Funcțiile anonime sunt funcții fără nume, adesea folosite ca argumente pentru alte funcții sau în expresii de funcții.

```
// Funcție anonimă ca argument pentru setTimeout
setTimeout(function () {
  console.log('Acest mesaj apare după 2 secunde')
}, 2000)

// Funcție anonimă arrow ca argument
;[1, 2, 3].map((x) => x * 2) // Returnează [2, 4, 6]
```

Funcții ca valori și callback-uri

În JavaScript, funcțiile sunt valori de prim rang (first-class citizens), ceea ce înseamnă că pot fi:

- Atribuite variabilelor
- Pasate ca argumente altor funcții
- Returnate din alte funcții

Callback-urile sunt funcții pasate ca argumente altor funcții, care vor fi executate mai târziu.

```
function processData(data, callback) {
  // Procesează datele...
  const result = data.toUpperCase()

  // Apelează callback-ul cu rezultatul
  callback(result)
}

// Folosirea funcției cu un callback
processData('text de procesat', function (result) {
  console.log('Rezultatul procesării:', result)
})
```

Funcții Immediately Invoked (IIFE)

IIFE (Immediately Invoked Function Expression) este o funcție care se auto-execută imediat după ce este definită.

```
;(function () {
  const secret = 'Nu poți accesa această variabilă din exterior'
  console.log('Funcția IIFE s-a executat!')
})()

// secret nu este accesibilă aici
```

Acestea sunt utile pentru a crea un scope izolat și a evita poluarea namespace-ului global.

Recursivitatea

Recursivitatea este procesul prin care o funcție se apelează pe sine. Este utilă pentru a rezolva probleme care pot fi descompuse în subprobleme similare.

```javascript
// Calculul factorial folosind recursivitate
function factorial(n) {
  // Cazul de bază
  if (n <= 1) {
    return 1
  }
  // Apel recursiv
  return n * factorial(n - 1)
}

console.log(factorial(5)) // Afișează: 120 (5! = 5 * 4 * 3 * 2 * 1)
```

Metode de obiect

Atunci când o funcție este o proprietate a unui obiect, aceasta se numește metodă.

```javascript
const person = {
  name: 'Ana',
  age: 30,
  introduce: function () {
    return `Mă numesc ${this.name} și am ${this.age} ani.`
  },
}

console.log(person.introduce()) // Afișează: Mă numesc Ana și am 30 ani.
```

În acest context, cuvântul cheie `this` se referă la obiectul care "deține" metoda.