Resurse suplimentare
pe care recomand să le parcurgi

Arrow Functions

Arrow functions (funcțiile săgeată) au fost introduse în ES6 și oferă o sintaxă mai concisă pentru scrierea funcțiilor. Ele sunt extrem de populare în React pentru event handlers și callback-uri.

Sintaxa tradițională vs Arrow Functions

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

// Arrow function echivalentă
const add = (a, b) => {
  return a + b
}

// Arrow function cu sintaxă concisă (implicit return)
const add = (a, b) => a + b
```

Variante de sintaxă pentru Arrow Functions

1. Cu mai mulți parametri

```
// Funcție tradițională
function greet(firstName, lastName) {
  return `Salut, ${firstName} ${lastName}!`
}

// Arrow function
const greet = (firstName, lastName) => `Salut, ${firstName} ${lastName}!`
```

2. Cu un singur parametru

```
// Parantezele sunt opționale pentru un singur parametru
const square = (x) => x * x
const double = (x) => x * 2

// Echivalent cu:
const square = (x) => x * x
const double = (x) => x * 2
```

3. Fără parametri

```
// Parantezele sunt obligatorii când nu există parametri
const sayHello = () => console.log('Hello!')
const getRandomNumber = () => Math.random()
```

4. Cu bloc de cod

```
const processData = (data) => {
  const processed = data.map((item) => item * 2)
  const filtered = processed.filter((item) => item > 10)
  return filtered
}
```

Diferențe importante față de funcțiile tradiționale

1. Binding-ul lui `this`

Aceasta este diferența cea mai importantă și frecvent întâlnită:

```
// Exemplu cu funcții tradiționale
const person = {
  name: 'Ana',
  hobbies: ['citit', 'înot', 'alergat'],

  // Funcție tradițională
  showHobbies: function () {
    this.hobbies.forEach(function (hobby) {
      console.log(`${this.name} îi place să facă ${hobby}`)
      // Eroare! `this` nu se referă la obiectul `person`
    })
  },
}

// Soluția cu arrow functions
const person = {
  name: 'Ana',
  hobbies: ['citit', 'înot', 'alergat'],

  showHobbies: function () {
    this.hobbies.forEach((hobby) => {
      console.log(`${this.name} îi place să facă ${hobby}`)
      // Funcționează! Arrow function păstrează `this` din context
    })
  },
}
```

2. Nu au propriul `arguments` object

```
// Funcție tradițională
function traditionalFunction() {
  console.log(arguments) // Disponibil
}

// Arrow function
const arrowFunction = () => {
  console.log(arguments) // Eroare! Nu este disponibil
}

// Soluția pentru arrow functions - folosește rest parameters
const arrowFunction = (...args) => {
  console.log(args) // Funcționează
}
```

Utilizarea în React

Arrow functions sunt extrem de utile în React:

```jsx
// Event handlers
const Button = () => {
  const handleClick = () => {
    console.log('Button clicked!')
  }

  return <button onClick={handleClick}>Click me</button>
}

// Callback-uri pentru array methods
const UserList = ({ users }) => {
  return (
    <ul>
      {users.map((user) => (
        <li key={user.id}>{user.name}</li>
      ))}
    </ul>
  )
}
```

Template Literals

Template literals (template strings) permit crearea de string-uri cu sintaxă îmbunătățită, incluzând interpolarea variabilelor și string-uri multi-linie.

Sintaxa de bază

```jsx
// Sintaxa tradițională
const name = 'Maria'
const age = 25
const message = 'Salut, sunt ' + name + ' și am ' + age + ' de ani.'

// Template literals
const message = `Salut, sunt ${name} și am ${age} de ani.`
```

Caracteristici principale

1. Interpolarea expresiilor

```
const a = 10
const b = 20

// Poți include orice expresie JavaScript
const result = `Suma dintre ${a} și ${b} este ${a + b}`
console.log(result) // "Suma dintre 10 și 20 este 30"

// Chiar și apeluri de funcții
const formatName = (name) => name.toUpperCase()
const greeting = `Salut, ${formatName('ana')}!`
console.log(greeting) // "Salut, ANA!"
```

2. String-uri multi-linie

```
// Sintaxa tradițională (neindemânatică)
const html = '<div>' + '<h1>Titlu</h1>' + '<p>Paragraf</p>' + '</div>'

// Template literals
const html = `
  <div>
    <h1>Titlu</h1>
    <p>Paragraf</p>
  </div>
`
```

3. Template literals în React

```
// Clase CSS dinamice
const Button = ({ variant, disabled }) => {
  const className = `btn btn-${variant} ${disabled ? 'btn-disabled' : ''}`

  return <button className={className}>Click me</button>
}

// Stiluri inline
const Alert = ({ type, message }) => {
  const style = {
    backgroundColor: type === 'error' ? '#ff6b6b' : '#51cf66',
    color: 'white',
    padding: '10px',
  }

  return <div style={style}>{`${type.toUpperCase()}: ${message}`}</div>
}
```

Tagged Template Literals (concepte avansate)

```
// Funcție tag personalizată
function highlight(strings, ...values) {
  return strings.reduce((result, string, i) => {
    const value = values[i] ? `<mark>${values[i]}</mark>` : ''
    return result + string + value
  }, '')
}

const name = 'JavaScript'
const result = highlight`Învăț ${name} pentru React!`
console.log(result) // "Învăț <mark>JavaScript</mark> pentru React!"
```

Combinarea Arrow Functions cu Template Literals

```
// Exemplu practic: formatarea datelor pentru UI
const users = [
  { id: 1, firstName: 'Ana', lastName: 'Popescu', age: 25 },
  { id: 2, firstName: 'Ion', lastName: 'Ionescu', age: 30 },
]

// Crearea de mesaje formatate
const formatUser = (user) =>
  `${user.firstName} ${user.lastName} (${user.age} ani)`

const userMessages = users.map((user) => formatUser(user))
console.log(userMessages)
// ["Ana Popescu (25 ani)", "Ion Ionescu (30 ani)"]

// Sau direct cu template literals
const createUserCard = (user) => `
  <div class="user-card">
    <h3>${user.firstName} ${user.lastName}</h3>
    <p>Vârsta: ${user.age} ani</p>
  </div>
`
```