Resurse suplimentare
pe care recomand să le parcurgi

ES6 Classes

Classes oferă o sintaxă sugar peste prototipuri.

```
class Person {
  constructor(name, age) {
    this.name = name
    this.age = age
  }

  greet() {
    return `Salut, sunt ${this.name}`
  }

  static fromObject({ name, age }) {
    return new Person(name, age)
  }
}

const ana = new Person('Ana', 26)
console.log(ana.greet()) // "Salut, sunt Ana"
```

Moștenire (Inheritance)

```
class Student extends Person {
  constructor(name, age, university) {
    super(name, age)
    this.university = university
  }

  info() {
    return `${this.name} studiază la ${this.university}`
  }
}

const ion = new Student('Ion', 21, 'UPB')
```

Classes vs Function Constructors

Aspect	Function Constructor	Class Syntax
Sintaxă	Funcție + prototip	`class`
`super`	Manual (`call`)	`super()`
Hoisting	Parțial	Nu
Metode implicite	Nu	Da (`constructor`)

Classes în React (legacy)

```
class Counter extends React.Component {
  state = { count: 0 }

  increment = () => {
    this.setState(({ count }) => ({ count: count + 1 }))
  }

  render() {
    return (
      <button onClick={this.increment}>Clicked {this.state.count} times</button>
    )
  }
}
```

JavaScript Modules

Modulele izolează codul și facilitează *code splitting*. Browserele moderne suportă nativ ES Modules (`<script type="module">`).

Exporturi

```
// math.js
export const PI = 3.14
export function add(a, b) {
  return a + b
}

// utils.js
export default function capitalize(str) {
  return str.charAt(0).toUpperCase() + str.slice(1)
}
```

Importuri

```
import { PI, add } from './math.js'
import capitalize from './utils.js'

console.log(add(PI, 2))
console.log(capitalize('react'))
```

Renaming & Namespace Imports

```
import { add as sum } from './math.js'
import * as math from './math.js'
```

Dynamic Imports (Code Splitting)

```
button.addEventListener('click', async () => {
  const { default: confetti } = await import(
    '<https://cdn.skypack.dev/canvas-confetti>'
  )
  confetti()
})
```

În React, bundlere precum Webpack și Vite transformă modulele și realizează tree-shaking pentru eliminarea codului neutilizat.

Closures și Lexical Scoping

Un closure se formează când o funcție „ține minte” variabile din scope-ul în care a fost creată.

```
function createCounter() {
  let count = 0
  return () => ++count
}

const counter = createCounter()
counter() // 1
counter() // 2
```

Use Case: Custom React Hook (conceptual)

```
function usePrevious(value) {
  const ref = useRef()
  useEffect(() => {
    ref.current = value
  })
  return ref.current // păstrează valoarea anterioară
}
```

IIFE (Immediately Invoked Function Expression)

```
const module = (() => {
  const secret = '🔒'
  return {
    reveal: () => secret,
  }
})()

console.log(module.reveal())
```