Resurse suplimentare
pe care recomand să le parcurgi

Obiectele sunt perfecte pentru a reprezenta entități din lumea reală, cu proprietățile și comportamentele lor.

```
// Un obiect simplu
const person = {
  name: 'Ana Popescu',
  age: 28,
  profession: 'Programator',
}
```

De ce folosim obiecte?

- **Organizarea datelor conexe** - Gruparea informațiilor legate de o entitate
- **Modelarea relațiilor** - Reprezentarea relațiilor dintre diferite entități
- **Abstractizarea complexității** - Ascunderea detaliilor de implementare
- **Encapsularea** - Combinarea datelor și funcționalităților asociate

Crearea obiectelor

Există mai multe modalități de a crea obiecte în JavaScript:

1. Sintaxa literală de obiect

Cea mai comună și concisă metodă de a crea un obiect.

```
const student = {
  name: 'Ion Ionescu',
  age: 22,
  faculty: 'Informatică',
  year: 3,
}
```

2. Folosind constructorul Object()

```
const student = new Object()
student.name = 'Ion Ionescu'
student.age = 22
student.faculty = 'Informatică'
student.year = 3
```

3. Folosind funcții constructor

```
function Student(name, age, faculty, year) {
  this.name = name
  this.age = age
  this.faculty = faculty
  this.year = year
}

const student1 = new Student('Ion Ionescu', 22, 'Informatică', 3)
const student2 = new Student('Maria Popescu', 20, 'Economie', 2)
```

4. Folosind Object.create()

```
const personPrototype = {
  greet: function () {
    return `Salut, numele meu este ${this.name}!`
  },
}

const student = Object.create(personPrototype)
student.name = 'Ion Ionescu'
student.age = 22

console.log(student.greet()) // "Salut, numele meu este Ion Ionescu!"
```

Accesarea proprietăților unui obiect

Există două moduri principale de a accesa proprietățile unui obiect:

1. Notația cu punct

```
const person = {
  name: 'Ana Popescu',
  age: 28,
}

console.log(person.name) // "Ana Popescu"
console.log(person.age) // 28
```

2. Notația cu paranteze pătrate

```
console.log(person['name']) // "Ana Popescu"
console.log(person['age']) // 28
```

Notația cu paranteze pătrate este necesară când:

- Numele proprietății este stocat într-o variabilă
- Numele proprietății conține caractere speciale sau spații
- Numele proprietății începe cu un număr

```
const property = 'name'
console.log(person[property]) // "Ana Popescu"

const extendedPerson = {
  'full name': 'Ana Maria Popescu',
  'birth-date': '15-03-1995',
}

console.log(extendedPerson['full name']) // "Ana Maria Popescu"
```

Modificarea obiectelor

Obiectele în JavaScript sunt mutabile, ceea ce înseamnă că le putem modifica după creare.

Adăugarea sau actualizarea proprietăților

```
const person = {
  name: 'Ana Popescu',
}

// Adăugarea unei proprietăți noi
person.age = 28
person['profession'] = 'Programator'

// Actualizarea unei proprietăți existente
person.name = 'Ana Maria Popescu'

console.log(person)
// { name: "Ana Maria Popescu", age: 28, profession: "Programator" }
```

Ștergerea proprietăților

Folosim operatorul `delete` pentru a elimina o proprietate.

```
const person = {
  name: 'Ana Popescu',
  age: 28,
  profession: 'Programator',
}

delete person.profession
console.log(person) // { name: "Ana Popescu", age: 28 }
```

Metode de obiect

Când o proprietate a unui obiect este o funcție, aceasta se numește **metodă**.

```
const person = {
  name: 'Ana Popescu',
  age: 28,
  // Metodă definită în obiect
  greet: function () {
    return `Salut, numele meu este ${this.name} și am ${this.age} ani.`
  },
  // Sintaxă prescurtată pentru metode (ES6)
  introduce() {
    return `Sunt ${this.name}, un programator de ${this.age} ani.`
  },
}

console.log(person.greet()) // "Salut, numele meu este Ana Popescu și am 28 ani."
console.log(person.introduce()) // "Sunt Ana Popescu, un programator de 28 ani."
```

În cadrul unei metode, cuvântul cheie `this` se referă la obiectul pe care îl "posedă" metoda.

Obiecte imbricate

Obiectele pot conține alte obiecte ca proprietăți, creând structuri de date complexe și ierarhice.

```
const person = {
  name: 'Ana Popescu',
  age: 28,
  address: {
    street: 'Strada Principală',
    number: 42,
    city: 'București',
    country: 'România',
  },
  contacts: {
    email: 'ana.popescu@exemplu.ro',
    phone: '0700123456',
  },
}

// Accesarea proprietăților imbricate
console.log(person.address.city) // "București"
console.log(person['contacts']['email']) // "ana.popescu@exemplu.ro"
```

Verificări și operațiuni pe obiecte

Verificarea existenței unei proprietăți

Există mai multe moduri de a verifica dacă un obiect are o anumită proprietate:

```
const person = {
  name: 'Ana Popescu',
  age: 28,
  profession: 'Programator',
}

// 1. Folosind operatorul in
console.log('name' in person) // true
console.log('address' in person) // false

// 2. Folosind metoda hasOwnProperty
console.log(person.hasOwnProperty('name')) // true
console.log(person.hasOwnProperty('toString')) // false (moștenită, nu proprie)

// 3. Verificare directă (atenție la valorile falsy)
console.log(person.name !== undefined) // true
console.log(person.address !== undefined) // false
```

Enumerarea proprietăților unui obiect

Există mai multe moduri de a parcurge proprietățile unui obiect:

```
const person = {
  name: 'Ana Popescu',
  age: 28,
  profession: 'Programator',
}

// 1. Bucla for...in (parcurge toate proprietățile enumerabile, inclusiv cele moștenite)
for (let property in person) {
  console.log(`${property}: ${person[property]}`)
}

// 2. Folosind Object.keys() (doar proprietățile proprii și enumerabile)
const keys = Object.keys(person)
console.log(keys) // ["name", "age", "profession"]

keys.forEach((key) => {
  console.log(`${key}: ${person[key]}`)
})

// 3. Folosind Object.values() (doar valorile proprietăților proprii și enumerabile)
const values = Object.values(person)
console.log(values) // ["Ana Popescu", 28, "Programator"]

// 4. Folosind Object.entries() (perechi [cheie, valoare] pentru proprietățile proprii ș
i enumerabile)
const entries = Object.entries(person)
console.log(entries) // [["name", "Ana Popescu"], ["age", 28], ["profession", "Programato
r"]]

entries.forEach(([key, value]) => {
  console.log(`${key}: ${value}`)
})
```

Proprietăți computate și shorthand

ES6 a introdus sintaxe îmbunătățite pentru lucrul cu obiecte:

Proprietăți computate

Putem folosi expresii în interiorul parantezelor pătrate pentru a defini numele proprietăților.

```
const propertyName = 'skill'
const propertyValue = 'JavaScript'

const developer = {
  name: 'Ana',
  [propertyName]: propertyValue,
  [`${propertyName}Level`]: 'Advanced',
}

console.log(developer) // { name: "Ana", skill: "JavaScript", skillLevel: "Advanced" }
```

Sintaxa prescurtată pentru proprietăți (Property Shorthand)

Când numele variabilei este identic cu numele proprietății, putem folosi o sintaxă prescurtată.

```
const name = 'Ana'
const age = 28
const profession = 'Programator'

// În loc de:
// const person = { name: name, age: age, profession: profession }

// Putem scrie:
const person = { name, age, profession }

console.log(person) // { name: "Ana", age: 28, profession: "Programator" }
```

Copierea și unirea obiectelor

Copierea superficială (Shallow Copy)

Există mai multe moduri de a crea o copie superficială a unui obiect:

```
const original = { name: 'Ana', skills: ['JavaScript', 'HTML', 'CSS'] }

// 1. Folosind Object.assign()
const copy1 = Object.assign({}, original)

// 2. Folosind operatorul spread (ES6)
const copy2 = { ...original }

// Modificarea copiei nu afectează originalul pentru proprietățile simple
copy1.name = 'Maria'
console.log(original.name) // "Ana"
console.log(copy1.name) // "Maria"

// DAR, modificarea proprietăților de referință (array-uri, obiecte) afectează și origina
lul
copy1.skills.push('React')
console.log(original.skills) // ["JavaScript", "HTML", "CSS", "React"]
console.log(copy1.skills) // ["JavaScript", "HTML", "CSS", "React"]
```

Copierea profundă (Deep Copy)

Pentru a copia un obiect inclusiv toate obiectele imbricate:

```
// Metodă simplă pentru obiecte serializabile (fără funcții, etc.)
const original = {
  name: 'Ana',
  address: { city: 'București', country: 'România' },
  skills: ['JavaScript', 'React'],
}

const deepCopy = JSON.parse(JSON.stringify(original))

// Modificarea proprietăților imbricate nu mai afectează originalul
deepCopy.address.city = 'Cluj'
deepCopy.skills.push('Node.js')

console.log(original.address.city) // "București"
console.log(original.skills) // ["JavaScript", "React"]
```

Unirea obiectelor (Merging)

```
const person = { name: 'Ana', age: 28 }
const job = { profession: 'Programator', experience: '5 ani' }

// Unirea a două sau mai multe obiecte
const personWithJob = { ...person, ...job }
console.log(personWithJob)
// { name: "Ana", age: 28, profession: "Programator", experience: "5 ani" }

// Proprietățile dublate sunt suprascrise de ultimul obiect
const person1 = { name: 'Ana', age: 28 }
const person2 = { name: 'Maria', city: 'București' }

const mergedPerson = { ...person1, ...person2 }
console.log(mergedPerson)
// { name: "Maria", age: 28, city: "București" }
```

Destructurarea obiectelor

Destructurarea este o caracteristică ES6 care permite extragerea proprietăților dintr-un obiect în variabile separate.

```javascript
const person = {
  name: 'Ana Popescu',
  age: 28,
  profession: 'Programator',
  address: {
    city: 'București',
    country: 'România',
  },
}

// Destructurare de bază
const { name, age } = person
console.log(name) // "Ana Popescu"
console.log(age) // 28

// Redenumirea variabilelor
const { name: fullName, profession: job } = person
console.log(fullName) // "Ana Popescu"
console.log(job) // "Programator"

// Destructurare cu valori implicite
const { salary = 'Confidențial' } = person
console.log(salary) // "Confidențial" (proprietatea nu există, se folosește valoarea impl
icită)

// Destructurare imbricate
const {
  address: { city, country },
} = person
console.log(city) // "București"
console.log(country) // "România"

// Destructurarea restului proprietăților (rest operator)
const { name, age, ...rest } = person
console.log(rest) // { profession: "Programator", address: { city: "București", country:
"România" } }
```

Obiecte nemodificabile

JavaScript oferă mai multe moduri de a restricționa modificările asupra obiectelor:

Object.freeze()

Împiedică adăugarea, ștergerea sau modificarea proprietăților unui obiect (cea mai strictă metodă).

```
const user = { name: 'Ana', age: 28 }
Object.freeze(user)

// Încercările de modificare vor eșua (în mod silențios în non-strict mode)
user.age = 29
user.email = 'ana@exemplu.ro'
delete user.name

console.log(user) // { name: "Ana", age: 28 } - nicio modificare
console.log(Object.isFrozen(user)) // true
```

Object.seal()

Împiedică adăugarea sau ștergerea proprietăților, dar permite modificarea celor existente.

```
const user = { name: 'Ana', age: 28 }
Object.seal(user)

// Modificarea proprietăților existente funcționează
user.age = 29

// Adăugarea sau ștergerea proprietăților va eșua
user.email = 'ana@exemplu.ro'
delete user.name

console.log(user) // { name: "Ana", age: 29 }
console.log(Object.isSealed(user)) // true
```

Object.preventExtensions()

Împiedică adăugarea de noi proprietăți, dar permite modificarea sau ștergerea celor existente.

```
const user = { name: 'Ana', age: 28 }
Object.preventExtensions(user)

// Modificarea și ștergerea funcționează
user.age = 29
delete user.name

// Adăugarea eșuează
user.email = 'ana@exemplu.ro'

console.log(user) // { age: 29 }
console.log(Object.isExtensible(user)) // false
```

Getteri și setteri

Getters și setters sunt metode speciale care permit controlul asupra accesului la proprietățile unui obiect.

```
const person = {
  firstName: 'Ana',
  lastName: 'Popescu',

  // Getter - se comportă ca o proprietate când este accesat
  get fullName() {
    return `${this.firstName} ${this.lastName}`
  },

  // Setter - se comportă ca o proprietate când este setat
  set fullName(value) {
    const parts = value.split(' ')
    this.firstName = parts[0]
    this.lastName = parts[1]
  },
}

// Folosirea getterului
console.log(person.fullName) // "Ana Popescu"

// Folosirea setterului
person.fullName = 'Maria Ionescu'
console.log(person.firstName) // "Maria"
console.log(person.lastName) // "Ionescu"
```

Metode statice pentru Object

JavaScript oferă metode utile atașate direct constructorului Object:

```
// Object.assign() - combină obiecte
const target = { a: 1 }
const source = { b: 2, c: 3 }
const result = Object.assign(target, source)
console.log(result) // { a: 1, b: 2, c: 3 }
console.log(target) // { a: 1, b: 2, c: 3 } - obiectul target este modificat!

// Object.keys() - returnează un array cu cheile enumerabile
const keys = Object.keys({ name: 'Ana', age: 28 })
console.log(keys) // ["name", "age"]

// Object.values() - returnează un array cu valorile proprietăților enumerabile
const values = Object.values({ name: 'Ana', age: 28 })
console.log(values) // ["Ana", 28]

// Object.entries() - returnează un array de array-uri [cheie, valoare]
const entries = Object.entries({ name: 'Ana', age: 28 })
console.log(entries) // [["name", "Ana"], ["age", 28]]

// Object.fromEntries() - transformă un array de array-uri [cheie, valoare] într-un obiec
t
const object = Object.fromEntries([
  ['name', 'Ana'],
  ['age', 28],
])
console.log(object) // { name: "Ana", age: 28 }
```