Resurse suplimentare
pe care recomand să le parcurgi

Ce sunt array-urile?

Array-urile (tablourile) sunt structuri de date care te ajută să stochezi și să organizezi multiple valori într-o singură variabilă. În JavaScript, array-urile sunt obiecte speciale care oferă metode pentru a le manipula și pot conține orice tip de date (numere, șiruri, booleene, obiecte, funcții sau chiar alte array-uri).

```
// Un array simplu cu numere
const numbers = [1, 2, 3, 4, 5]

// Un array cu tipuri diferite de date
const mixed = [42, 'text', true, { name: 'Ana' }, [10, 20]]
```

Crearea array-urilor

Există mai multe moduri de a crea array-uri în JavaScript:

1. Folosind notația cu paranteze pătrate (literal array)

```
const fruits = ['măr', 'banană', 'portocală']
```

2. Folosind constructorul Array()

```
const vegetables = new Array('morcov', 'broccoli', 'roșie')
```

3. Crearea unui array gol și popularea lui ulterior

```
const animals = []
animals[0] = 'câine'
animals[1] = 'pisică'
animals[2] = 'hamster'
```

4. Crearea unui array cu Array.of() (ES6)

```
const numbers = Array.of(1, 2, 3, 4, 5)
```

5. Crearea unui array cu Array.from() (ES6)

```
// Crearea unui array din string
const characters = Array.from('JavaScript')
console.log(characters) // ['J', 'a', 'v', 'a', 'S', 'c', 'r', 'i', 'p', 't']

// Crearea unui array cu mapare
const squaredNumbers = Array.from([1, 2, 3, 4], (x) => x * x)
console.log(squaredNumbers) // [1, 4, 9, 16]
```

Accesarea elementelor

Elementele dintr-un array sunt indexate numeric, începând de la 0.

```
const fruits = ['măr', 'banană', 'portocală', 'struguri']

// Accesarea elementelor folosind indexul
console.log(fruits[0]) // "măr"
console.log(fruits[2]) // "portocală"

// Accesarea ultimului element
console.log(fruits[fruits.length - 1]) // "struguri"

// Folosind at() pentru indexare inclusiv negativă (ES2022)
console.log(fruits.at(-1)) // "struguri" (ultimul element)
console.log(fruits.at(-2)) // "portocală" (penultimul element)
```

Proprietăți și verificări de bază

Lungimea array-ului

Proprietatea `length` returnează numărul de elemente dintr-un array.

```
const fruits = ['măr', 'banană', 'portocală']
console.log(fruits.length) // 3
```

Verificarea dacă o variabilă este array

```
const numbers = [1, 2, 3]
console.log(Array.isArray(numbers)) // true
console.log(Array.isArray('text')) // false
```

Modificarea array-urilor

Array-urile în JavaScript sunt mutabile, adică pot fi modificate după creare.

Adăugarea de elemente

```
const fruits = ['măr', 'banană']

// Adăugare la sfârșit
fruits.push('portocală')
console.log(fruits) // ["măr", "banană", "portocală"]

// Adăugare la început
fruits.unshift('struguri')
console.log(fruits) // ["struguri", "măr", "banană", "portocală"]

// Adăugare la un index specific
fruits[4] = 'kiwi'
console.log(fruits) // ["struguri", "măr", "banană", "portocală", "kiwi"]
```

Eliminarea elementelor

```
const fruits = ['struguri', 'măr', 'banană', 'portocală', 'kiwi']

// Eliminare de la sfârșit
const last = fruits.pop()
console.log(last) // "kiwi"
console.log(fruits) // ["struguri", "măr", "banană", "portocală"]

// Eliminare de la început
const first = fruits.shift()
console.log(first) // "struguri"
console.log(fruits) // ["măr", "banană", "portocală"]

// Eliminarea elementelor de la un index specific
// splice(indexStart, câteElemente, element1, element2, ...)
const removedElements = fruits.splice(1, 1)
console.log(removedElements) // ["banană"]
console.log(fruits) // ["măr", "portocală"]
```

Înlocuirea elementelor

```
const fruits = ['măr', 'banană', 'portocală']

// Înlocuire directă
fruits[1] = 'pere'
console.log(fruits) // ["măr", "pere", "portocală"]

// Înlocuire cu splice
fruits.splice(0, 1, 'ananas')
console.log(fruits) // ["ananas", "pere", "portocală"]
```

Metode de parcurgere a array-urilor

JavaScript oferă mai multe moduri de a parcurge array-urile:

Bucla for clasică

```
const numbers = [10, 20, 30, 40, 50]

for (let i = 0; i < numbers.length; i++) {
  console.log(numbers[i])
}
```

Bucla for...of (ES6)

```
const numbers = [10, 20, 30, 40, 50]

for (const number of numbers) {
  console.log(number)
}
```

Metoda forEach()

```
const numbers = [10, 20, 30, 40, 50]

numbers.forEach(function (number, index) {
  console.log(`Numărul la indexul ${index} este: ${number}`)
})

// Cu arrow function
numbers.forEach((number, index) =>
  console.log(`Numărul la indexul ${index} este: ${number}`)
)
```

Metode de manipulare a array-urilor

JavaScript oferă numeroase metode pentru manipularea array-urilor. Acestea pot fi împărțite în două categorii:

- Metode care modifică array-ul original (mutative)
- Metode care returnează un nou array (non-mutative)

Concatenarea array-urilor

```
const fruits = ['măr', 'banană']
const vegetables = ['morcov', 'broccoli']

// Metoda concat() (non-mutativă)
const foods = fruits.concat(vegetables)
console.log(foods) // ["măr", "banană", "morcov", "broccoli"]

// Operatorul spread (ES6) (non-mutativ)
const foodsSpread = [...fruits, ...vegetables]
console.log(foodsSpread) // ["măr", "banană", "morcov", "broccoli"]
```

Extragerea părților din array

```
const numbers = [10, 20, 30, 40, 50]

// slice(start, end) - returnează o porțiune din array (non-mutativă)
const numbersSlice = numbers.slice(1, 4)
console.log(numbersSlice) // [20, 30, 40]
console.log(numbers) // [10, 20, 30, 40, 50] (array-ul original rămâne neschimbat)
```

Transformarea array-urilor

map() - Creează un nou array aplicând o funcție fiecărui element

```
const numbers = [1, 2, 3, 4, 5]

// Dublează fiecare număr
const doubledNumbers = numbers.map(function (number) {
  return number * 2
})

// Sau cu arrow function
const doubledNumbersArrow = numbers.map((number) => number * 2)

console.log(doubledNumbersArrow) // [2, 4, 6, 8, 10]
```

filter() - Creează un nou array cu elementele care trec un test

```
const numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

// Filtrează numerele pare
const evenNumbers = numbers.filter(function (number) {
  return number % 2 === 0
})

// Sau cu arrow function
const evenNumbersArrow = numbers.filter((number) => number % 2 === 0)

console.log(evenNumbersArrow) // [2, 4, 6, 8, 10]
```

reduce() - Reduce array-ul la o singură valoare

```
const numbers = [1, 2, 3, 4, 5]

// Calculează suma elementelor
const sum = numbers.reduce(function (accumulator, currentValue) {
  return accumulator + currentValue
}, 0)

// Sau cu arrow function
const sumArrow = numbers.reduce((acc, val) => acc + val, 0)

console.log(sum) // 15
```

sort() - Sortează elementele array-ului (mutativă)

```
const fruits = ['portocală', 'măr', 'banană', 'kiwi']

// Sortare alfabetică
fruits.sort()
console.log(fruits) // ["banană", "kiwi", "măr", "portocală"]

// Sortarea numerelor necesită o funcție de comparare
const numbers = [40, 100, 1, 5, 25, 10]
numbers.sort((a, b) => a - b) // Sortare crescătoare
console.log(numbers) // [1, 5, 10, 25, 40, 100]

numbers.sort((a, b) => b - a) // Sortare descrescătoare
console.log(numbers) // [100, 40, 25, 10, 5, 1]
```

Metode de căutare în array-uri

indexOf() și lastIndexOf() - Găsește indexul unui element

```
const fruits = ['măr', 'banană', 'portocală', 'măr', 'kiwi']

console.log(fruits.indexOf('măr')) // 0 (primul index unde apare "măr")
console.log(fruits.lastIndexOf('măr')) // 3 (ultimul index unde apare "măr")
console.log(fruits.indexOf('ananas')) // -1 (elementul nu există în array)
```

includes() - Verifică dacă array-ul conține un anumit element

```
const fruits = ['măr', 'banană', 'portocală']

console.log(fruits.includes('banană')) // true
console.log(fruits.includes('ananas')) // false
```

find() și findIndex() - Găsește primul element care satisface o condiție

```
const persons = [
  { name: 'Ana', age: 25 },
  { name: 'Mihai', age: 30 },
  { name: 'Elena', age: 20 },
]

// Găsește prima persoană cu vârsta peste 25
const foundPerson = persons.find((person) => person.age > 25)
console.log(foundPerson) // { name: "Mihai", age: 30 }

// Găsește indexul primei persoane cu vârsta peste 25
const indexPerson = persons.findIndex((person) => person.age > 25)
console.log(indexPerson) // 1
```

some() și every() - Verifică dacă unele/toate elementele satisfac o condiție

```
const numbers = [1, 2, 3, 4, 5]

// Verifică dacă există cel puțin un număr mai mare decât 3
const existsGreaterThanThree = numbers.some((number) => number > 3)
console.log(existsGreaterThanThree) // true

// Verifică dacă toate numerele sunt pozitive
const allPositive = numbers.every((number) => number > 0)
console.log(allPositive) // true
```

Array-uri multidimensionale

Array-urile în JavaScript pot fi și multidimensionale (array-uri de array-uri).

```
// Un array bidimensional (matrice)
const matrix = [
  [1, 2, 3],
  [4, 5, 6],
  [7, 8, 9],
]

// Accesarea elementelor
console.log(matrix[1][2]) // 6 (al treilea element din al doilea array)

// Parcurgerea unei matrici
for (let i = 0; i < matrix.length; i++) {
  for (let j = 0; j < matrix[i].length; j++) {
    console.log(matrix[i][j])
  }
}
```

Metode moderne de array din ES6+

Array Destructuring

```
const colors = ['red', 'green', 'blue']

// Destructurare de bază
const [first, second, third] = colors
console.log(first) // "red"
console.log(second) // "green"

// Ignorarea unor elemente
const [first2, , third2] = colors
console.log(third2) // "blue"

// Cu valori implicite
const [a, b, c, d = 'yellow'] = colors
console.log(d) // "yellow"

// Rest pattern
const [first, ...rest] = colors
console.log(rest) // ["green", "blue"]
```

Array.from() - Convertirea array-like objects în array-uri adevărate

```
// Convertirea unui string în array
const letters = Array.from('Hello')
console.log(letters) // ["H", "e", "l", "l", "o"]

// Cu o funcție de mapare
const numbers = Array.from('12345', (n) => parseInt(n))
console.log(numbers) // [1, 2, 3, 4, 5]
```

Array.flat() și Array.flatMap() (ES2019)

```
// Aplatizarea unui array imbricat
const nestedArray = [1, 2, [3, 4, [5, 6]]]
console.log(nestedArray.flat()) // [1, 2, 3, 4, [5, 6]]
console.log(nestedArray.flat(2)) // [1, 2, 3, 4, 5, 6]

// flatMap() - combină map() și flat()
const words = ['Hello', 'world']
const letters = words.flatMap((word) => word.split(''))
console.log(letters) // ["H", "e", "l", "l", "o", "w", "o", "r", "l", "d"]
```

Performanța și cele mai bune practici

Optimizarea operațiunilor cu array-uri

1. **Predefiniți lungimea array-ului** când știți câte elemente va conține:

```
// Mai eficient pentru array-uri mari
const arr = new Array(1000)
```

2. **Evitați modificarea unui array în timpul parcurgerii** pentru a preveni comportamente neașteptate.
3. **Folosiți metode specializate în loc de bucle** când este posibil, pentru cod mai clar și mai concis.
4. **Preferați metode non-mutative** pentru a evita efectele secundare nedorite.

Performanța diferitelor metode

- **push()** și **pop()** (operațiuni la sfârșit) sunt mai rapide decât **unshift()** și **shift()** (operațiuni la început)
- Pentru iterație, **forEach()** poate fi mai lent decât bucla **for** clasică în unele cazuri
- **map()**, **filter()** și **reduce()** sunt opțiuni excelente pentru manipularea datelor, dar pot fi mai lente pentru array-uri foarte mari