Resurse suplimentare
pe care recomand să le parcurgi

Structuri Condiționale

Instrucțiunea if

Structura `if` execută un bloc de cod dacă o condiție specificată este evaluată ca adevărată.

```
let age = 18

if (age >= 18) {
  console.log('Ești major!')
}
```

Instrucțiunea if-else

Adaugă un bloc alternativ care se execută când condiția este falsă.

```
let age = 16

if (age >= 18) {
  console.log('Ești major!')
} else {
  console.log('Ești minor!')
}
```

Instrucțiunea if-else if-else

Pentru verificarea mai multor condiții în secvență.

```
let grade = 8

if (grade >= 9) {
  console.log('Excelent!')
} else if (grade >= 7) {
  console.log('Foarte bine!')
} else if (grade >= 5) {
  console.log('Suficient.')
} else {
  console.log('Insuficient.')
}
```

Operatorul ternar (condiție ? expr1 : expr2)

O formă condensată a instrucțiunii if-else, utilă pentru atribuiri simple.

```
let age = 20
let status = age >= 18 ? 'adult' : 'minor'
console.log(status) // "adult"

// Operatori ternari înlănțuiți
let message =
  age < 13 ? 'copil' : age < 18 ? 'adolescent' : age < 65 ? 'adult' : 'senior'
```

Instrucțiunea switch

Utilă când trebuie să comparăm o valoare cu mai multe cazuri posibile.

```
let day = 3
let dayName

switch (day) {
  case 1:
    dayName = 'Luni'
    break
  case 2:
    dayName = 'Marți'
    break
  case 3:
    dayName = 'Miercuri'
    break
  case 4:
    dayName = 'Joi'
    break
  case 5:
    dayName = 'Vineri'
    break
  case 6:
    dayName = 'Sâmbătă'
    break
  case 7:
    dayName = 'Duminică'
    break
  default:
    dayName = 'Zi invalidă'
}

console.log(dayName) // "Miercuri"
```

Dacă uiți să adaugi `break` după fiecare caz, execuția va continua până la următorul `break` sau până la sfârșitul instrucțiunii `switch`. Acest comportament poate fi folosit intenționat pentru a grupa cazuri:

```
let day = 6
let dayType

switch (day) {
  case 1:
  case 2:
  case 3:
  case 4:
  case 5:
    dayType = 'Zi lucrătoare'
    break
  case 6:
  case 7:
    dayType = 'Weekend'
    break
  default:
    dayType = 'Zi invalidă'
}

console.log(dayType) // "Weekend"
```

Structuri Repetitive (Bucle)

Bucla for

Execută un bloc de cod de un număr specific de ori.

```
// Structura for: for (inițializare; condiție; incrementare)
for (let i = 0; i < 5; i++) {
  console.log(`Iterația ${i}`)
}
```

Bucla while

Execută un bloc de cod atât timp cât o condiție specificată este adevărată.

```
let i = 0
while (i < 5) {
  console.log(`Iterația ${i}`)
  i++
}
```

Bucla do-while

Similar cu `while`, dar garantează că blocul de cod se execută cel puțin o dată, deoarece condiția este verificată după execuție.

```
let i = 0
do {
  console.log(`Iterația ${i}`)
  i++
} while (i < 5)
```

Diferența între `while` și `do-while` devine evidentă când condiția este falsă de la început:

```
let i = 10

// Nu execută nimic (condiția este falsă inițial)
while (i < 5) {
  console.log('Acest cod nu se execută niciodată')
}

// Execută o dată, apoi condiția este falsă
do {
  console.log('Acest cod se execută o dată') // Se afișează
} while (i < 5)
```

Bucla for...in

Iterează prin proprietățile enumerabile ale unui obiect.

```
const person = {
  name: 'Ana',
  age: 25,
  occupation: 'Programator',
}

for (let property in person) {
  console.log(`${property}: ${person[property]}`)
}
// Afișează:
// name: Ana
// age: 25
// occupation: Programator
```

Bucla for...of (ES6)

Iterează prin elementele unei colecții iterabile (arrays, strings, etc).

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

for (let number of numbers) {
  console.log(number)
}
// Afișează numerele: 10, 20, 30, 40, 50

const word = 'JavaScript'
for (let letter of word) {
  console.log(letter)
}
// Afișează fiecare literă: J, a, v, a, S, c, r, i, p, t
```

Controlul Fluxului în Bucle

break

Întrerupe complet execuția buclei.

```
for (let i = 0; i < 10; i++) {
  if (i === 5) {
    break // Iese din buclă când i ajunge la 5
  }
  console.log(i)
}
// Afișează: 0, 1, 2, 3, 4
```

continue

Sare peste iterația curentă și continuă cu următoarea.

```
for (let i = 0; i < 10; i++) {
  if (i % 2 === 0) {
    continue // Sare peste numerele pare
  }
  console.log(i)
}
// Afișează: 1, 3, 5, 7, 9
```

Instrucțiuni Label

Deși mai rar folosite, instrucțiunile label permit controlul fluxului în bucle imbricate.

```
outerLoop: for (let i = 0; i < 3; i++) {
  for (let j = 0; j < 3; j++) {
    if (i === 1 && j === 1) {
      break outerLoop // Iese din ambele bucle
    }
    console.log(`i=${i}, j=${j}`)
  }
}
```

Algoritmi Simpli Folosind Structuri de Control

Calculul Factorial

```
function factorial(n) {
  let result = 1
  for (let i = 2; i <= n; i++) {
    result *= i
  }
  return result
}

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

Verificarea Numerelor Prime

```
function isPrime(number) {
  if (number <= 1) return false
  if (number <= 3) return true

  if (number % 2 === 0 || number % 3 === 0) return false

  for (let i = 5; i * i <= number; i += 6) {
    if (number % i === 0 || number % (i + 2) === 0) {
      return false
    }
  }

  return true
}

console.log(isPrime(17)) // true
console.log(isPrime(20)) // false
```

Generarea Șirului Fibonacci

```
function fibonacci(n) {
  const sequence = [0, 1]

  for (let i = 2; i < n; i++) {
    sequence[i] = sequence[i - 1] + sequence[i - 2]
  }

  return sequence
}

console.log(fibonacci(10)) // [0, 1, 1, 2, 3, 5, 8, 13, 21, 34]
```