Resurse suplimentare
pe care recomand să le parcurgi

Soluția pentru Partea 1: Calculator de Statistici

```javascript
// Array de numere pentru analiză
const numbers = [12, 7, 19, 23, 8, 10, 17, 22, 14, 9]

// Funcția pentru calcularea mediei
function calculateAverage(numbers) {
  const sum = numbers.reduce((total, number) => total + number, 0)
  return sum / numbers.length
}

// Funcția pentru găsirea maximului
function findMax(numbers) {
  return Math.max(...numbers)
  // Alternativ: return numbers.reduce((max, num) => num > max ? num : max, numbers[0])
}

// Funcția pentru găsirea minimului
function findMin(numbers) {
  return Math.min(...numbers)
  // Alternativ: return numbers.reduce((min, num) => num < min ? num : min, numbers[0])
}

// Funcția pentru calcularea sumei
function calculateSum(numbers) {
  return numbers.reduce((total, number) => total + number, 0)
}

// Funcția pentru filtrarea numerelor pare
function filterEven(numbers) {
  return numbers.filter((number) => number % 2 === 0)
}

// Afișează rezultatele
console.log('Statistici pentru array-ul:', numbers)
console.log('Media:', calculateAverage(numbers))
console.log('Maximul:', findMax(numbers))
console.log('Minimul:', findMin(numbers))
console.log('Suma:', calculateSum(numbers))
console.log('Numerele pare:', filterEven(numbers))
```

Soluția pentru Partea 2: Gestiune de Inventar

```javascript
// Inventarul inițial de produse
const inventory = [
  { id: 1, name: 'Laptop', price: 2500, quantity: 5 },
  { id: 2, name: 'Telefon', price: 1200, quantity: 10 },
  { id: 3, name: 'Tabletă', price: 800, quantity: 8 },
]

function addProduct(inventory, product) {
  // Verificăm dacă produsul cu acest ID există deja
  const existingProduct = inventory.find((p) => p.id === product.id)
  if (existingProduct) {
    throw new Error(`Produsul cu ID ${product.id} există deja în inventar`)
  }
  inventory.push(product)
  return inventory
}

function findProduct(inventory, id) {
  const product = inventory.find((p) => p.id === id)
  if (!product) {
    return null // sau putem arunca o eroare
  }
  return product
}

function updateQuantity(inventory, id, newQuantity) {
  const product = findProduct(inventory, id)
  if (!product) {
    throw new Error(`Produsul cu ID ${id} nu a fost găsit`)
  }
  product.quantity = newQuantity
  return inventory
}

function calculateTotalValue(inventory) {
  return inventory.reduce(
    (total, product) => total + product.price * product.quantity,
    0
  )
}

function deleteProduct(inventory, id) {
  const productIndex = inventory.findIndex((p) => p.id === id)
  if (productIndex === -1) {
    throw new Error(`Produsul cu ID ${id} nu a fost găsit`)
  }
  inventory.splice(productIndex, 1)
  return inventory
}

// Testare funcții
console.log('Inventarul inițial:', inventory)

// Adaugă un produs nou
const newProduct = { id: 4, name: 'Monitor', price: 700, quantity: 3 }
addProduct(inventory, newProduct)
console.log('Inventar după adăugare:', inventory)

// Găsește un produs
console.log('Produsul cu ID 2:', findProduct(inventory, 2))

// Actualizează cantitatea
updateQuantity(inventory, 3, 12)
console.log('Inventar după actualizare:', inventory)

// Calculează valoarea totală
console.log('Valoarea totală a inventarului:', calculateTotalValue(inventory))

// Șterge un produs
deleteProduct(inventory, 1)
console.log('Inventar după ștergere:', inventory)
```

Soluția pentru Partea 3: Transformare de Date cu Array Methods

```
// Date despre studenți
const students = [
  { name: 'Ana Ionescu', age: 21, average: 9.5 },
  { name: 'Mihai Popescu', age: 22, average: 8.3 },
  { name: 'Elena Dumitrescu', age: 20, average: 7.8 },
  { name: 'Andrei Stanescu', age: 23, average: 6.4 },
  { name: 'Maria Constantinescu', age: 21, average: 9.1 },
  { name: 'Ion Vasilescu', age: 22, average: 5.2 },
  { name: 'Ioana Munteanu', age: 20, average: 4.9 },
]

// 1. Filtrează studenții care au note de trecere (peste 5)
const passingStudents = students.filter((student) => student.average > 5)

// 2. Sortează studenții după medie, în ordine descrescătoare
const sortedStudents = [...students].sort((a, b) => b.average - a.average)

// 3. Transformă array-ul pentru a include doar numele și media
const nameAndAverage = students.map((student) => ({
  name: student.name,
  average: student.average,
}))

// 4. Calculează media generală a tuturor studenților
const overallAverage =
  students.reduce((total, student) => total + student.average, 0) /
  students.length

// Afișarea rezultatelor
console.log('Studenți cu note de trecere:', passingStudents)
console.log('Studenți sortați după medie:', sortedStudents)
console.log('Nume și medie:', nameAndAverage)
console.log('Media generală:', overallAverage.toFixed(2))
```

Soluția pentru Partea 4: Gestionarea Erorilor în Aplicații

```
// Clasa personalizată pentru erori de validare
class ValidationError extends Error {
  constructor(message, field) {
    super(message)
    this.name = 'ValidationError'
    this.field = field
  }
}

// Funcția de validare
function validateUser(user) {
  // Validare nume
  if (!user.name || user.name.length < 3) {
    throw new ValidationError(
      'Numele trebuie să aibă cel puțin 3 caractere',
      'name'
    )
  }

  // Validare email
  if (!user.email || !user.email.includes('@')) {
    throw new ValidationError(
      "Email-ul trebuie să conțină caracterul '@'",
      'email'
    )
  }

  // Validare vârstă
  if (typeof user.age !== 'number') {
    throw new ValidationError('Vârsta trebuie să fie un număr', 'age')
  }

  if (user.age < 18 || user.age > 120) {
    throw new ValidationError('Vârsta trebuie să fie între 18 și 120', 'age')
  }

  return true
}

// Teste pentru funcția de validare
const users = [
  { name: 'Ana Popescu', email: 'ana@example.com', age: 25 },
  { name: 'Io', email: 'ion-example.com', age: 17 },
  { name: 'Maria Ionescu', email: 'maria@example.com', age: 130 },
  { name: 'Gheorghe Popa', email: 'gheorghe@example.com', age: 'treizeci' },
]

// Verifică fiecare utilizator
users.forEach((user, index) => {
  console.log(`\\nVerificare utilizator ${index + 1}:`, user)
  try {
    validateUser(user)
    console.log('✅ Utilizator valid!')
  } catch (error) {
    if (error instanceof ValidationError) {
      console.log(
        `❌ Eroare de validare pentru câmpul ${error.field}: ${error.message}`
      )
    } else {
      console.log(`❌ Eroare neașteptată: ${error.message}`)
    }
  }
})
```

Soluția pentru Partea 5: Aplicație de Gestionare a Sarcinilor

```javascript
// Manager de sarcini
const taskManager = {
  tasks: [],
  lastId: 0,

  // Adaugă o sarcină nouă
  addTask(title, description, priority) {
    if (!title || title.trim() === '') {
      throw new Error('Titlul sarcinii nu poate fi gol')
    }

    if (priority < 1 || priority > 3 || !Number.isInteger(priority)) {
      throw new Error('Prioritatea trebuie să fie un număr întreg între 1 și 3')
    }

    const id = ++this.lastId

    const newTask = {
      id,
      title,
      description,
      priority,
      completed: false,
      dateCreated: new Date(),
    }

    this.tasks.push(newTask)
    return newTask
  },

  // Marchează o sarcină ca finalizată
  completeTask(id) {
    const task = this.tasks.find((s) => s.id === id)
    if (!task) {
      throw new Error(`Sarcina cu ID ${id} nu a fost găsită`)
    }

    task.completed = true
    return task
  },

  // Șterge o sarcină
  deleteTask(id) {
    const index = this.tasks.findIndex((s) => s.id === id)
    if (index === -1) {
      throw new Error(`Sarcina cu ID ${id} nu a fost găsită`)
    }

    const deletedTask = this.tasks[index]
    this.tasks.splice(index, 1)
    return deletedTask
  },

  // Filtrează sarcinile după starea lor
  filterTasks(completed = true) {
    return this.tasks.filter((task) => task.completed === completed)
  },

  // Sortează sarcinile după prioritate (1 = mare, 3 = mică)
  sortByPriority() {
    return [...this.tasks].sort((a, b) => a.priority - b.priority)
  },

  // Calculează procentul de finalizare
  completionPercentage() {
    if (this.tasks.length === 0) return 0

    const completedTasks = this.tasks.filter((task) => task.completed)
    return Math.round((completedTasks.length / this.tasks.length) * 100)
  },
}

// Testare funcționalități
try {
  // Adăugare sarcini
  taskManager.addTask(
    'Învață JavaScript',
    'Studiază funcțiile, array-urile și obiectele',
    1
  )
  taskManager.addTask('Cumpărături', 'Lapte, pâine, ouă', 2)
  taskManager.addTask('Plimbare', '30 minute în parc', 3)

  console.log('Sarcini inițiale:', taskManager.tasks)

  // Finalizare sarcină
  taskManager.completeTask(2)
  console.log('După finalizare:', taskManager.tasks)

  // Sortare după prioritate
  const sortedTasks = taskManager.sortByPriority()
  console.log('Sarcini sortate după prioritate:', sortedTasks)

  // Filtrare sarcini finalizate
  const completedTasks = taskManager.filterTasks(true)
  console.log('Sarcini finalizate:', completedTasks)

  // Procentaj finalizare
  console.log(
    'Procent de finalizare:',
    taskManager.completionPercentage() + '%'
  )
} catch (error) {
  console.error('A apărut o eroare:', error.message)
}
```