Seminarul 3

Arrays and higher order array methods

Metodele **Higher Order Array** sunt funcții în limbajul JavaScript care pot fi aplicate pe un array și care primesc ca argument una sau mai multe funcții sau returnează o funcție. Aceste metode sunt esențiale pentru programarea funcțională și sunt utile pentru a **efectua operații complexe** pe **elementele unui array**, cum ar fi **filtrarea**, **maparea** și **reducerea**, folosind **funcții callback** sau **expresii lambda**.

Criteriul de căutare în funcția de filtrare JavaScript este transmis folosind o funcție de apel invers (**callbackfn**). Această funcție de apel invers (callback) este executată pentru **fiecare element** din array.

❖ Filter

Metoda **filter()** a instanțelor de array creează o **copie superficială** a unei părți dintr-un array dat, filtrând-o pentru a include doar elementele din array-ul dat care trec de testul implementat de funcția furnizată.

În principiu, dacă funcția de apel invers (callback) returnează **true**, elementul curent va fi în array-ul rezultat. Dacă returnează **false**, nu va fi în acesta.

```
let filtered = [12, 5, 8, 130, 44].filter(el => el > 10);  //expected output: [12, 130, 44]
```

De reținut

1. Funcția de filtrare a array-ului **nu modifică array-ul**. Aduceți-vă aminte **să salvați array-ul filtrat** în cazul în care aveți în plan să-l utilizați ulterior.
2. Funcția **nu va fi executată** în cazul în care **array-ul este gol**.

Pentru mai multe detalii legate de metoda filter: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/filter

❖ Map

Metoda **map()** a instanțelor de Array creează un nou array populat cu rezultatele apelării unei funcții furnizate pentru fiecare element din array-ul apelat.

Totodată, aceasta este o **metodă iterativă** (primește o funcție de apel invers (callback) ca argument). Ea apelează o funcție de apel invers (**callbackFn**) furnizată o singură dată pentru fiecare element dintr-un array și construiește un nou array din rezultate.

```
let mapped = [12, 5, 8, 130, 44].map((x) => x * 2); // expected output: [24, 10, 16, 260, 88]
```

De reținut

1. Funcția de mapare a array-ului **creează un nou array**. Aduceți-vă aminte **să salvați array-ul mapat** în cazul în care aveți în plan să-l utilizați ulterior.

Pentru mai multe detalii legate de metoda map: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/map

❖ Reduce

Metoda **reduce()** a instanțelor de array execută o funcție de apel invers furnizată de utilizator pe fiecare element al array-ului, în ordine, trecându-i valoarea returnată din calculul elementului precedent. Rezultatul final al rulării reducer-ului asupra tuturor elementelor din array este **o singură valoare**.

```
const reduced = [12, 5, 8, 130, 44].reduce((acc, el) => acc + el, inValue); // expected output: 199
```

De reținut

1. Prima dată când este rulat callback-ul, nu există "valoare returnată a calculului anterior". Dacă este furnizată, o valoare inițială poate fi folosită în locul ei.
2. Dacă nu este furnizată o valoare inițială, **primul element** din array de **la indexul 0** o să fie folosit ca și valoare inițială, iar iterația începe de la **următorul element** (indexul 1 în loc de indexul 0).
3. Generează **eroare** dacă array-ul este gol și dacă nu este declarată o valoare inițială.

Pentru mai multe detalii legate de metoda reduce: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/reduce

Ex 1. Folosiți metodele map și filter pentru a procesa un array de numere reprezentând ani de naștere pentru a obține toate vârstele peste 18 ani.

const birthYears = [1990, 2000, 1985, 1995, 2005, 2010];

Ex 2. Implementați o funcție care primește ca parametrii un array de numere și un număr și returnează suma tuturor numerelor din array divizibile cu cel de-al doilea parametru.

Ex 3. Se furnizează un array de obiecte care reprezintă un grup de studenți, fiecare cu un nume și un array de note la teste. Trebuie să găsiți o soluție prin care să utilizați metodele **map**, **filter** și **reduce** pentru a calcula media notelor pentru fiecare student, apoi să returnați un array de obiecte care conține doar studenții care au o medie a notelor mai mare de **90**.

const students = [{ name: "Alice", scores: [90, 85, 92] }, { name: "Bob", scores: [75, 80, 85] }, { name: "Charlie", scores: [90, 95, 85] }, { name: "David", scores: [100, 100, 100] }];

Funcții de formatare a șirurilor de caractere

❖ IndexOf

Metoda **indexOf()** a instanțelor de array returnează primul index la care poate fi găsit un element dat în array, sau -1 dacă acesta nu este prezent.

```
let mapped = [12, 5, 8, 130, 44].indexOf(8); // expected output: 2
```

De reținut

1. Metoda **indexOf()** compară elementul care se dorește a fi căutat cu elementele din array folosind egalitate strictă (aceeași metodă utilizată de operatorul ===)
2. **Valorile NaN nu sunt niciodată considerate egale**, așadar indexOf(NaN) **returnează întotdeauna -1**.

```
const numbers = [1, 2, 3, 4, NaN, 6];

// Căutăm indexul pentru valoarea NaN
const indexOfNaN = numbers.indexOf(NaN);

console.log(indexOfNaN); // Va afișa: -1, deoarece indexOf() nu găsește NaN în mod egal cu el însuși
```

Pentru mai multe detalii legate de metoda indexOf: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/indexOf

❖ Replace

Metoda **replace()** a valorilor de tip șir de caractere returnează un șir de caractere nou cu una, unele sau toate potrivirile unui șablon înlocuite cu o substituție. Șablonul poate fi un șir de caractere sau o expresie regulată, iar substituția poate fi un șir de caractere sau o funcție apelată pentru fiecare potrivire. Dacă șablonul este un șir de caractere, doar prima apariție va fi înlocuită. **Șirul original rămâne neschimbat**.

```
const text = "Hello, my name is [name].";
const replacedText = text.replace("[name]", "Alex");

console.log(replacedText); // Will output: "Hello, my name is Alex."
```

De reținut

1. Această metodă **nu modifică valoarea șirului** pe care este apelată. Ea returnează un șir nou.
2. Un șablon de șir va fi înlocuit **doar o dată**. Pentru a efectua o căutare globală și înlocuire, utilizați o expresie regulată cu flag-ul "g" sau folosiți metoda **replaceAll()** în schimb.

Pentru mai multe detalii legate de metoda replace: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String/replace

Alte metode utile pentru array-uri și obiecte

❖ Find

Metoda **find()** a instanțelor de array returnează primul element din array-ul furnizat care satisface funcția de testare furnizată. Dacă niciun element nu satisface funcția de testare, se returnează **undefined**.

Totodată, este o **metodă iterativă**. Ea apelează o funcție de callback furnizată o singură dată pentru fiecare element dintr-un array în ordine crescătoare a indicilor, până când funcția de callback returnează o valoare evaluată ca adevărat (**truthy**). Apoi, **find()** returnează acel element și se oprește din iterarea prin array. Dacă funcția de callback nu returnează niciodată o valoare evaluată ca adevărat, **find()** returnează **undefined**.

```
const found = [12, 5, 8, 130, 44].find((element) => element > 10); //expected output: 12
```

De reținut

1. În cazul în care **nu găsește** un element care să satisfacă condiția, metoda find o să returneze **undefined**.

Pentru mai multe detalii legate de metoda find: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/find

❖ Every

Metoda **every()** a instanțelor de array verifică dacă toate elementele din array trec testul implementat de funcția furnizată. Ea **returnează o valoare booleană**.

Metoda **every()** este o metodă iterativă. Ea apelează o funcție de **callback** furnizată o singură dată pentru fiecare element dintr-un array, până când funcția de callback returnează o valoare evaluată ca **falsă**. Dacă se găsește un astfel de element, **every() returnează imediat false** și **se oprește** din iterarea prin array. În caz contrar, dacă funcția de callback returnează o valoare evaluată ca adevărată pentru toate elementele, every() returnează **true**.

```
const found = [12, 5, 8, 130, 44].every((element) => element >= 10); // false
```

De reținut

1. Metoda every o să ruleze atâta timp cât condiția **este adevărată** și va returna **true** la sfârșit sau **false** în caz contrar.

Pentru mai multe detalii legate de metoda every: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/every

❖ Some

Metoda **some()** a instanțelor de array testează dacă cel puțin un element din array trece testul implementat de funcția furnizată. Ea returnează **true** dacă, în array, găsește un element pentru care funcția furnizată returnează true; în caz contrar, returnează **false**. Aceasta nu modifică array-ul.

```
const found = [12, 5, 8, 130, 44].some(el => el % 2 === 0);  // expected output: true
```

Pentru mai multe detalii legate de metoda some: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/some

❖ Object.keys

Metoda statică **Object.keys() returnează un array** cu numele proprietăților proprii ale unui obiect dat, care sunt enumerable și reprezintă chei sub formă de șiruri de caractere.

```
const person = {
    firstName: "John",
    lastName: "Doe",
    age: 30,
};

console.log(Object.keys(person)); // expected output: Array ["firstName", "lastName", "age"]
```

Metoda **Object.keys()** returnează un array ale cărui elemente sunt șiruri de caractere corespunzătoare **numelor de proprietăți** string-keyed enumerate găsite direct în obiect. Acest lucru este echivalent cu a itera cu un ciclu **for...in**, cu excepția faptului că un ciclu **for...in** enumerează și proprietățile din lanțul de prototipuri. Ordinea array-ului returnat de Object.keys() este aceeași cu cea furnizată de un ciclu **for...in**.

De reținut

1. Object.keys() returnează numele directe, **fără proprietăți moștenite**, spre deosebire de **for...in**.

Pentru mai multe detalii legate de metoda Object.keys: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object/keys

❖ Object.values

Metoda statică **Object.values()** returnează un array al valorilor proprietăților enumerate, direct definite, ale unui obiect dat, care sunt string-keyed (bazate pe șiruri de caractere).

```
const person = {
    firstName: "John",
    lastName: "Doe",
    age: 30
};

console.log(Object.values(person)); // expected output: ["John", "Doe", 30]
```

Object.values() returnează un array ale cărui elemente sunt **valorile proprietăților** enumerate, direct definite, ale unui obiect, care sunt bazate pe chei sub formă de șiruri de caractere. Acest lucru este echivalent cu a itera cu un ciclu **for...in**, cu excepția faptului că un ciclu **for...in** enumerează și proprietățile din lanțul de prototipuri. Ordinea array-ului returnat de Object.values() este aceeași cu cea furnizată de un ciclu **for...in**.

Pentru mai multe detalii legate de metoda Object.values: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object/values

❖ Object.entries

Metoda statică **Object.entries()** returnează un array al perechilor cheie-valoare ale proprietăților enumerable, direct definite, ale unui obiect dat, care sunt bazate pe chei sub formă de șiruri de caractere.

```
const person = {
    firstName: "John",
    lastName: "Doe",
    age: 30
};

console.log(Object.entries(person));
// expected output: [["firstName","John"],["lastName","Doe"],["age",30]]
```

Pentru mai multe detalii legate de metoda Object.values: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object/entries

Conditional (ternary) operator

Operatorul condițional (ternar) este singurul operator JavaScript care primește **trei operanzi**: **o condiție** urmată de un semn de întrebare (?), apoi **o expresie de executat dacă condiția este evaluată ca adevărată**, urmată de două puncte (:), și în cele din urmă **expresia de executat dacă condiția este evaluată ca falsă**. Acest operator este frecvent utilizat ca alternativă la o instrucțiune if...else.

```
const num = 10;

const message = num % 2 === 0 ? "Numărul este par" : "Numărul este impar";

console.log(message); // expected output: "Numărul este par"
```

Pentru mai multe detalii legate de operatorul condițional (ternar): https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Conditional_operator

Rest parameters

Sintaxa **rest parameter** permite unei funcții să accepte un număr indefinit de argumente ca un array, oferind o modalitate de a reprezenta funcții variadice în JavaScript.

```
function sum(...args) {
    return args.reduce((total, num) => total + num, 0);
}

const result = sum(1, 2, 3, 4, 5);
console.log(result); // expected output: 15
```

Pentru mai multe detalii legate de operatorul condițional (ternar): https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Functions/rest_parameters

Template literals

Template literals sunt literale delimitate de caracterele **backtick** (`), permițând șiruri multiline, interpolare de șiruri cu expresii încorporate și construcții speciale numite șabloane etichetate.

```
`string text ${expression} string text`
```

```
console.log(`string text line 1
string text line 2`);
// "string text line 1
// string text line 2",
```

În plus față de șirurile normale, literalurile șablon pot conține și alte părți numite plasări, care sunt **expresii încorporate** delimitate de un semn de dolar și paranteze drepte: **${expresie}**. Șirurile și plasările sunt transmise unei funcții - fie unei funcții implicite, fie unei funcții pe care o furnizați. Funcția implicită (când nu furnizați una proprie) realizează doar interpolare de șir pentru a efectua substituția plaselor și apoi concatenează părțile într-un singur șir.

Ex 4. Creați o funcție arrow care primește ca și parametrii un text și un cuvânt și se dorește găsirea respectivului cuvânt în cadrul textului. În final, funcția trebuie să returneze un mesaj de tip “template literals” care să conțină cuvântul care s-a dorit a fi căutat și poziția cuvântului doar dacă acesta a fost găsit.

Ex 5. Implementați o funcție care primește un factor și o listă de numere. Funcția trebuie să calculeze suma tuturor numerelor din listă și să o înmulțească cu factorul dat și la sfârșit să returneze rezultatul.