* **Proprietati la array:**

.**length** – contine numarul de elemente din array

* Metode la obiectul array:

- **.push(el1, el2, ...)** – adauga elemente la final de array si returneaza noul length la array

- **shift**() – sterge elementul la inceput d array

- **.unshift(el1, el2, ...)** – adauga elemente la inceput de array si returneaza noul length la array

- .**pop()** – sterge ultimul element din array si il returneaza

- **.indexOf(element) –** returneaza indexul la element sau -1 daca nu il gaseste

- **.includes(element)** – true daca element e in array si false daca nu

- **splice(start)** – sterge totul de la start, inclusiv start

- **splice(start, nr)** – sterge un numar de nr elemente incepand cu start, inclusiv start

const *array* = [1,2,3,4,5];  
*array*.splice(1,2);  
*console*.log(*array*);

[1, 4, 5]

- **splice(start, nr, el1, el2, ...)** – sterge un numar de nr elemente incepand cu start si apoi adauga elementele el1 si el2 incepand cu start. **Afecteaza array original!**

const *array* = ['element1','element2','element3','element4','element5'];  
*array*.splice(1,2, 'new1','new2');  
*console*.log(*array*);

[ 'element1', 'new1', 'new2', 'element4', 'element5' ]

putem folosi 0 ca nr pentru a nu sterge nici unul si pur si simplu sa inseram incepand cu un index

const *array* = ['element1','element2','element3','element4','element5'];  
*array*.splice(1,0, 'new1','new2');  
*console*.log(*array*);

[

'element1',

'new1',

'new2',

'element2',

'element3',

'element4',

'element5'

]

* **slice(from, to)** – returneaza un nou array ce contine elementele de la from pana la to, from inclusive,to exclusive

Daca nu folosim from, se pune 0 automat, adica slice(5) inseaman de la 0 la 5, 5 exclusiv

Putem folosi si indexi negativi, gen slice(1,-2) – de la 1 pana la penultimul element, penultimul exclusiv. Decim -1 e ultimul, -2 penultimul si tot asa

* **slice()** – face o copie la array, caci ia de la primul pana la ultimul index si copie totul
* **some(functie)** – returneaza true daca cel putin un element din array corespunde conditiei din functia predicat

const *array* = [10,20,30,40,51]  
  
*console*.log(*array*.some((el) => el % 2 !== 0))

true

* **reduce((total, current, index, array) => { }, initial)**

- primul argument e o functie, ce ia total, adica rezultatul carat de la o reducere la alta si current care e elementul current

- initial e optional. Daca nu e oferit, se iau primele 2 elemente pentru a face un total, apoi total e tarait la fiecare operatie, si se ia cate un element doar. Daca e oferit, nu se iau primele 2 elemente ca prima operatie, ci primul si acest initial ca total

const *array* = [10,20,30,40,50]  
  
*console*.log(*array*.reduce((total, el) => total+el))

150

intai 10 + 20

apoi 30 + 30

apoi 60 + 40

apoi 100 + 50

const *array* = [10,20,30,40,50]  
  
*console*.log(*array*.reduce((total, el) => total+el,1000))

1150

Putem pune ca element de start si un obiect!

* **at(index) –** spre deosebire de [], putem folosi indexi negativi
* **concat(arr)** – creaza un nou array ce contine elementele cei curent + cele a lui arr puse la urma

const *array1* = [10,20,30,40,50]  
const *array2* = [60,70,80];  
  
*console*.log(*array1*.concat(*array2*));

* **every(predicate)** – verifica daca toate elementele corespund la predicate

const *array1* = [10,20,30,40,50]  
  
*console*.log(*array1*.every((el) => el<100));

* **fill(value, start, end)** – inlocuieste value incepand de la start in array
* const *array1* = [10,20,30,40,50]  
  *array1*.fill(0,2);  
    
  *console*.log(*array1*);

[ 10, 20, 0, 0, 0 ]

* **filter((element, index, array) => )** – creaza un nou array doar cu valorile care corespund filtrului

const *array1* = [10,25,30,45, 60, 81]  
  
*console*.log(*array1*.filter((el) => el % 2 === 0));

* **find(lambda)** – gaseste primul element care corespunde expresiei lambda

const *array1* = [11,25,30,45, 60, 81]  
  
*console*.log(*array1*.find((el) => el % 5 === 0));

* **findIndex(lambda)** – returneaza primul index al elementului ce corespunde prediacutului
* **findLastIndex(lambda)**
* **flat(level)** – transforma un array cu mai multe arrays in el in array cu elemente, fara arrays in el. Level arata cat de adanc sa faca concatenarea. Gen, 1 va face ca sa se concateneze [] cu [], 2 [] cu [, [], ] si tot asa

const *array1* = [1,2, [3, 4, [5, 6 , 7]]]  
  
  
*console*.log(*array1*.flat());

[ 1, 2, 3, 4, [ 5, 6, 7 ] ]

const *array1* = [1,2, [3, 4, [5, 6 , 7]]]  
  
  
*console*.log(*array1*.flat(2));

[

1, 2, 3, 4,

5, 6, 7

]

default e 1!

* **flatMap(lambda)** – intai face map la toate elementele si apoi la noul array face flat. Merge doar la 1 level deep! Nu putem seta mai multe
* **forEach(element, index, array)**
* **join(delimiter)** – creaza un string cu elementele din array, ele fiind delimitate prin delimiter

const *array1* = [1,2, [3, 4, [5, 6 , 7]]]  
  
  
*console*.log(*array1*.join("|"));

* **reverse()** – afecteaza array original

const *array1* = [1,2,3,4,5,6,7,8,9]  
*array1*.reverse();  
  
*console*.log(*array1*);

* **sort(comparator)** – sorteaza pe baza la un comparator function. Exact ca la Java. Sorteaza array original si returneaza o referinta la el
* const *array1* = ['aa','gfgfg','d','dfdfd','dfd'];  
    
  *console*.log(*array1*.sort((el1, el2) => {  
   if(el1.length > el2.length)  
   return 1;  
   else if(el1.length < el2.length)  
   return -1;  
   else return 0;  
  }));

Daca nu oferim un comparator, sortarea se face mereu ca string, chiar daca punem numere!

1 inseamna ca vrem ca elementele sa fie schimbate cu locul

-1 inseamna ca vrem sa ramana asa cum sunt, la locul lor

Nu e obligatoriu sa fie 1 si -1. orice e mai mare ca 0 e ascenden, mai mic e descendent

*console*.log([5,3,2,1,4].sort((a,b) => a - b))

si asa deci va merge

* **map(function(element, index, array){ })** – returneaza unul nou, nu il afecteaza pe cel curent. Va trimite la functie, pentru fiecare element, valoarea sa, indexul si intregul array

**forEach()**

* **forEach()** nu primeste de la array ca argument doar itemul curent
* **forEach** primeste si indexul si intregul array, adica asa:

**forEach(element, index, array)**

const *array* = [10,20,30,40]  
  
*array*.forEach((element, index, array) => *console*.log(element, index, array))

10 0 [ 10, 20, 30, 40 ]

20 1 [ 10, 20, 30, 40 ]

30 2 [ 10, 20, 30, 40 ]

40 3 [ 10, 20, 30, 40 ]

* In cazul la map **forEach(value, key, map)**

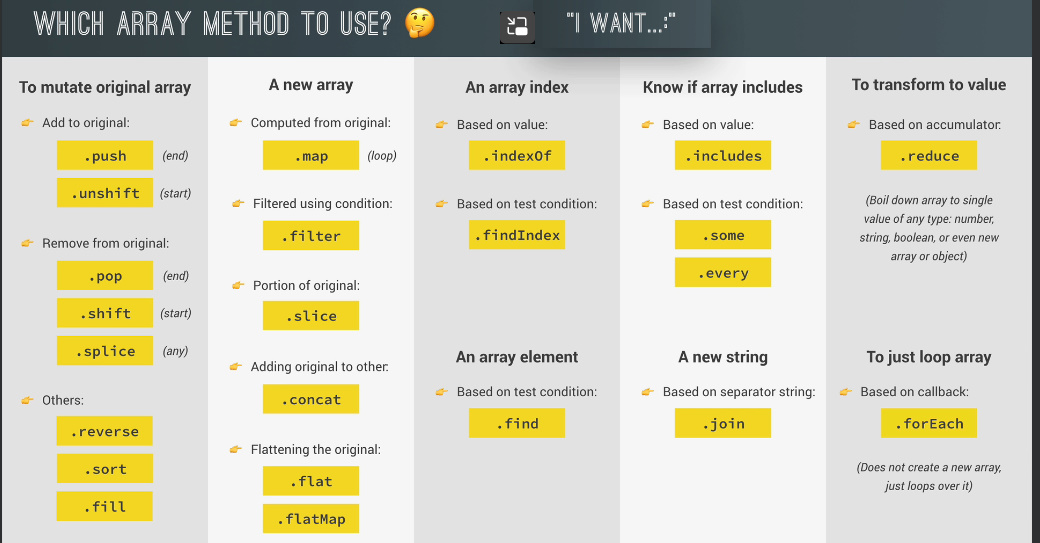
const *map* = new *Map*([['A',1],['B',2],['C',3]]);  
  
*map*.forEach((value, key, map)=> *console*.log(key, value, map))

A 1 Map(3) { 'A' => 1, 'B' => 2, 'C' => 3 }

B 2 Map(3) { 'A' => 1, 'B' => 2, 'C' => 3 }

C 3 Map(3) { 'A' => 1, 'B' => 2, 'C' => 3 }

* **La set forEach(value, key, set) dar value e identic cu key**



**element.insertAdjacentHTML(position, html)**

* position – arata unde anume urmeaza sa fie pus codul html fata de elementul selectat cu querySelect()
* Avem positiile:

<!-- beforebegin -->

<p>

<!-- afterbegin -->

foo

<!-- beforeend -->

</p>

<!-- afterend -->

presupunem ca am facut **querySelect(‘p’).insertAdjacentHTML()**

- beforebegin – e pus inaintea elementului selectat, adica inaintea lui p

- afterbegin – e pus ca prim element in elementul selectat

- beforend – e pus la final de element

- afterend – e pus dupa element

**innerHTML**

* Orice element selectat din DOM, de ex cu querySelector(), are un field innerHTML care nu contine doar textul, ci prorpiu zis intregul code, cu toate elemente din el, de ex daca mai sunt alte elemente, ca <p>, <a> etc. ele toate sunt continute in acest field
* Putem face asa pentru a curata un bloc selectat:

**document.querySelector(‘.test’).innerHTML = ‘’;**

**Problema cu submit in form**

* Daca avem o forma si un buton submit, problema e ca in mod default, HTML face reload la pagina dupa ce se apasa submit button din forma
* Asta e problema, caci ceea ce face codul JS e afisat doar pe o fractiune de secunda, pana pagina sa faca reload, si apoi pagina face reload si tot ce a facut JS dispare
* Pentru a rezolva problema, e necesar sa dezactivam forma de a trimite datele undeva, ca sa nu mai dea asa reload la pagina cu **preventDefault()**
* *btnLogin*.addEventListener('click', (el) => el.preventDefault())

**Moduri de a crea un array**

* []
* new Array()
* Atentie! new Array(5) nu va crea un array cu elementul 5 in el! Va crea un array cu 5 elemente goale
* *console*.log(new *Array*(5))

[ <5 empty items> ]

* Tot ce putem face cu un asa array e sa apelam metoda fill()
* Asa nu e prea clar ce face, de aceea, mai bine folosim asta:

const *array* = *Array*.from({length: 7}, () => 1);  
*console*.log(*array*);

[

1, 1, 1, 1,

1, 1, 1

]

const *array* = *Array*.from({length: 7}, (element, index) => index + 1);  
*console*.log(*array*);

[

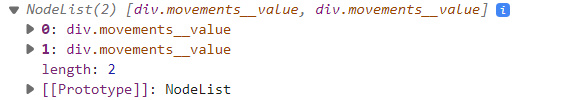
1, 2, 3, 4,

5, 6, 7

]

**NodeList**

* **NodeList** – collection of document nodes
* Problema cu NodeList e ca nu au multe metode pe care le au array, ca map(), filter() etc.
* Un NodeList e returnat mereu cand folosim document.
* *console*.log(*document*.querySelectorAll('.movements\_\_value'))



* Putem sa le transformam in array cu metoda from():
* *console*.log(*Array*.from(*document*.querySelectorAll('.movements\_\_value')))
* From ofera si un al 2 argument, si anume metoda de mapare:

**Array.from(NodeList, (el) => {})**