"working with arrays"

① Slice

- it slice the array from the starting index to the last index-1 provided as params of the func
- It also doesn't affect the original array.

*in this case it will return an array of the characters at indicies from one to two, the three will not be included.

```
const arr = ["a", "b", "c", "d"];
console.log(arr.slice(1, 3));
```

② Splice :—

- It is used to get a sub array of the original array but in this case the Reflection will be mutated to the original one.
- It accepts two parameters the first for the starting index and the second indicated the num of character to get from the current index.

*in this case, I will remove the character at index one from original array.

```
const arr = ["a", "b", "c", "d"];
//console.log(arr.slice(1, 3));

arr.splice(1, 1);
console.log(arr);
```

③ Reverse :— used to Reverse the original array and its reflection is mutated

③ For Each method :→

- For each accepts a CallBack function which will be called at every movement.
- The For Each is the caller of the call back function not you ?

```
for (const [i, movement] of movements.entries()) {
  if (movement > 0) {
    console.log(`Movement ${i + 1}: You deposited $
    {movement}`);
  } else {
    console.log(`Movement ${i + 1}: You withdrew $
    {Math.abs(movement)}`);
  }
}

console.log('---- FOREACH ----');
movements.forEach(function (mov, i, arr) {
  if (mov > 0) {
    console.log(`Movement ${i + 1}: You deposited $
    {mov}`);
  } else {
    console.log(`Movement ${i + 1}: You withdrew $
    {Math.abs(mov)}`);
  }
});
```

- The CallBack Function accepts arguments of (elemt, index, total array) In order.

④ For each with maps and Sets :→

Same For arrays

in arrays the arguments are (elemt, index, array) ← array

(Value, key, map) ← maps and sets

```
currencies.forEach((value, key, map) => {
  console.log(`value at key ${key} equal to ${value}`);
});

const newSet = new Set(["USD", "USD", "EGP"]);
newSet.forEach((value, key, map) => {
  console.log(`value at key ${key} equal to ${value}`);
});
```

```
containerMovements.insertAdjacentHTML
("afterbegin", htmlDisplay);
```

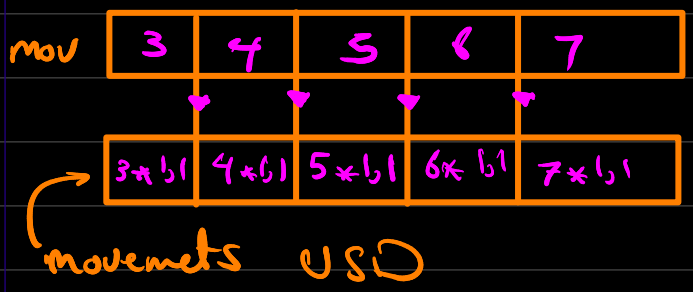
if I selected an html and saved it to variable and want to display its Content.

* after begin makes every elemt received will be at the top of the Container, and the oldest ones goes down.

Data Transformation :→

① map :→ It is used to make operation at every index of the array and returns the answer to a new array.

- There is a difference between For each and map as Foreach doesn't returns any thing and simply perform operation other wise map performs operation and return the result to a new array.




```
const eurToUsd = 1.1;

const movementsUSD = movements.map(function(mov) {
  return mov * eurToUsd;
})
```

② filter :→

It is used to loop over an iterable and returns the elemets that satisfies a Certain Condition.

- note that map and filter returns new array as it doesn't mutate the original ones.

deposits will only contain movemets > 0.

```
const deposits = movements.filter(function (mov) {
  return mov > 0;
});
console.log(movements);
console.log(deposits);
```

3 Reduce :→

- used to make the whole array tends to one value.
- In this case It gets the sum

```
const balance = movements.reduce(function (acc, cur, i, arr) {
  console.log(`Iteration ${i}: ${acc}`);
  return acc + cur;
}, 0);
console.log(balance);
```

- acc keeps track of sum and initialized with zero and cur which refers to current index.

4 find :→

as usual it loops throughout iterable and returns the first element that satisfies the condition.

```
const account = accounts.find(acc => acc.owner ===
'Jessica Davis');
console.log(account);
```

5 Some and every :→

```
const anyDeposits = movements.some(mov => mov > 0);
console.log(anyDeposits);
```

it will return true if any element matches the condition.

it is similar to includes but the difference is that includes accepts a value not a call back func and condition.

otherwise :→ every will return true if and only if all the elements matches the specified condition.

Flat and flatMap: →

```
const arr = [[1, 2, 3], [4, 5, 6], 7, 8];
console.log(arr.flat());
```

↳ Flat is used to Convert nested arrays to be in the same outer level.

* The output will be [1, 2, 3, 4, 5, 6, 7, 8]

. Flat also accepts a number which indicates the no of deeper levels you wanna make them Flat.

. If I don't specify any number, It is 1 by default.

↳ flatMap == map then flat like that →

```
const balanceFinal = accounts
  .map((acc) => acc.movements)
  .flat()
  .reduce((acc, curr) => curr + acc, 0);
console.log(balanceFinal);

=

const balanceFinal_01 = accounts
  .flatMap((acc) => acc.movements)
  .reduce((acc, curr) => curr + acc, 0);
```

. notice that flatMap goes one level deeper

. if you wanna more than one level you must use the first way and specify the no of levels you wanna go deeper.

* Sort accepts a callback func
if the res > 0 → swap
res < 0 → keep

↳ we can do it like that:→

```
movements.sort((a, b) => a - b);
```

↳ if a > b ⇒ a - b ⇒ (+) ⇒ swap
a < b ⇒ a - b ⇒ (-) → keep as it is

```
// return < 0, A, B (keep order)
// return > 0, B, A (switch order)
movements.sort((a, b) => {
  if (a > b) return 1;
  if (b > a) return -1;
});
console.log(movements);
```

ASC

* Create and fill arrays:→

```
Const x = new Array(7);
```

↳ This will create an empty array which has size of seven.

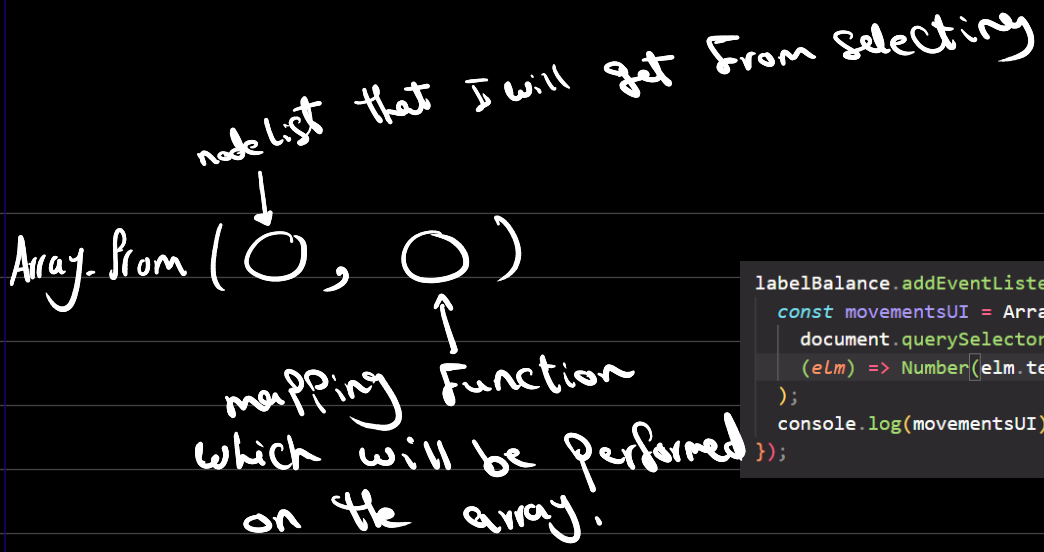
```
x.fill(1, 2, 5)
```

number to fill the array with
starting index
ending index

* from method:→

```
// Array.from
const y = Array.from({ length: 7 }, () => 1);
console.log(y);

const z = Array.from({ length: 7 }, (cur, i) => i + 1)
;
console.log(z);
```




```
labelBalance.addEventListener('click', () => {
  const movementsUI = Array.from(
    document.querySelectorAll('.movements__value'),
    (elm) => Number(elm.textContent.replace('€', ''))
  );
  console.log(movementsUI);
});
```

note that:- QuerySelectorAll doesnt return Pure array but it returns Something called nodelist which can be converted to actual array using _from_