15/5/21

* Array - Iteration →

→ Either we can use the normal loops i.e., for, while, do-while.

→ for arrays we also have → foreach

Ex → let myArr = [1, 2, 4];
myArr.forEach (function(item) {
console.log(item) });

and we have for map.

```
myArr.map((item) => {
        console.log(item)})
```

→ Suppose we want to multiply every element of an array by 2 what we can do is →

① forEach

```
marks.forEach((individual) => {
```

↓ in here we require one more Parameter.

→ marks.forEach((individual, index) => {

```
    marks[index] = marks[index] * 2;
})
console.log(marks);
```

② map → we simply return the value without needing extra Parameter.

```
marks = marks.map((individual) => {
    return individual * 2;
})
console.log(marks);
```

→ map by default creates a new array and when we return it the new array is having all the elements doubled in it apart from the original array.

→ Whereas forEach doesn't returns anything if you do so it will show you undefined.

* Array Methods →

(1) Concat → It concatenate the array.

(2) join → It joins the elements with delimiter passed.

(3) Push → Pushes the element to the last of array.

(4) Pop → Pops / removes the last element from the array.

(5) shift → first element is removed.

(6) unshift → Elements are added in the beginning.

(7) slice → removes the elements from the array acc. to the index provided.

(8) splice → removes and add elements till the count given.

(9) reverse → reverses the array.

(10) sort → sorts the array in increasing order by default.

Ex → (1) Concat → let myArr = [1, 2, 3];
myArr.concat(5, 6, 7);

O/P → [1, 2, 3, 5, 6, 7]

(2) Join → let myArr = ["Hi", "How", "Are", "You"];
myArr.join("_")

O/P → Hi-How-Are-You.

(3) Ex → Push, Pop, shift, unshift...

let myArr = [1, 2, 3]

Push → myArr.push(7)

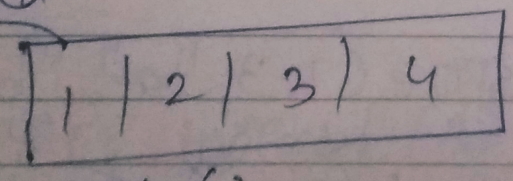
O/P → [1, 2, 3, 7]

Pop → myArr.pop()

O/P → [1, 2, 3]

} Stack

shift → Removed



myArr.shift()

unshift → myArr.unshift('4', '5')

O/P → [4, 5, 2, 3, 4]

} Queue

* To → insert beginning (unshift), deletion end (pop)
and
insert end (push), deletion beginning (shift).

Ex →

① Slice → let myArr = [1, 2, 3, 4, 5, 6, 7]
(indexes: 0 1 2 3 4 5 6)

myArr.slice(1, 3)
↳ not included
only 1, 2 will be include the last index is not calculated.

Console.log(myArr)

o/p → [1, 4, 5, 6, 7];

② Splice → myArr = [1, 2, 3, 4, 5, 6, 3] (index; count = elements from index)

myArr.splice(1, 3, "a", "b", "c", "d");
(1 = starting index, 3 = count)

o/p → [1, a, b, c, d, 5, 6, 3];

③ reverse → Arr = [1, 2, 3, 4]

Arr.reverse()

o/p → [4, 3, 2, 1]

(4) Arr = [10, 14, 12, 1, 4]

Arr.sort()

O/P → [1, 4, 10, 12, 14];

(5) IndexOf → gives me the first index acc. to the occurence of the element.

whereas,

(6) lastIndexOf → gives me the last index acc. to the occurence of an element.

Ex → myArr = [1, 2, 3, 4, 5, 3]
0 1 2 3 4 5

So, in here we have 3 at index '2' and '5'

* So, myArr.indexOf(3)

O/P → 2

* and

myArr.indexO myArr.lastIndexOf(3)

O/P → 5

(7) filter → It filters the elements from an array acc. to the given condition

```
newArr = myArr.filter(item => {
        return (condition)
})
console.log(newArr);
```

(8) Some ~~& every~~ → It returns us - 'true' or 'false'

If condition put in filter is even fullfilled by a single element then it will return "True" otherwise "False".

(9) Every ← ~~Even~~ Check whether every element is fulfilling the condition or not.

If not "False" is returned.

Ex-1

```
myArr.some(item => {
    return (    )
})
```

2) myArr.every(item => {
    return (condition);
})

* Maps → (key, value) pair.

Ex → 'emailid' , 'Address'
↓ ↓
key value.

* Sets → All the items are unique in set.

[Key = value.]

→ In here, we don't give the key explicitly, whatever value will be given will also be the key.

→ Both will be same.