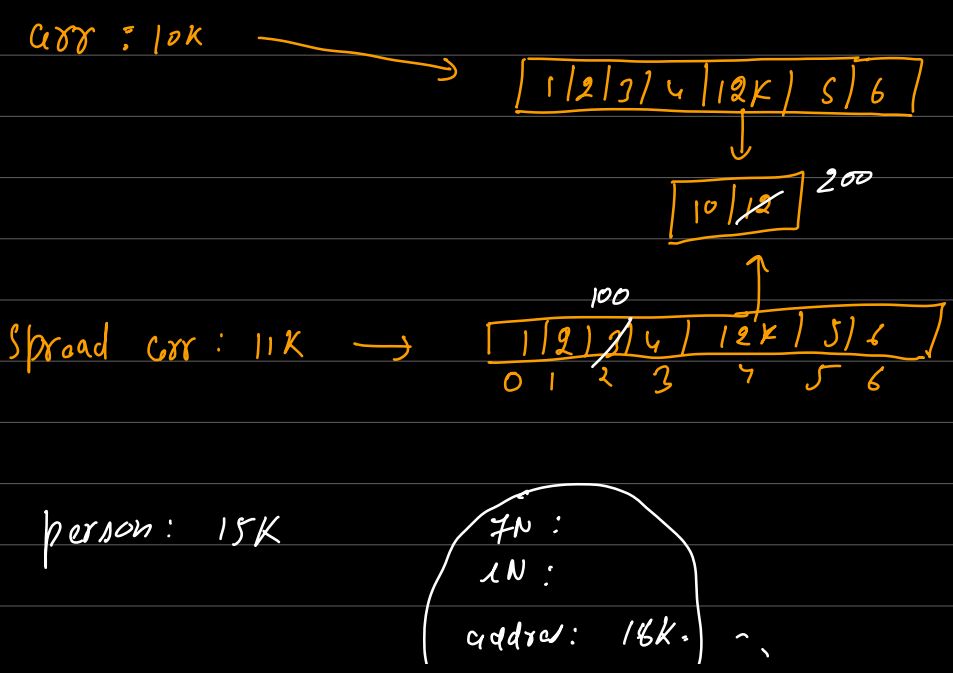
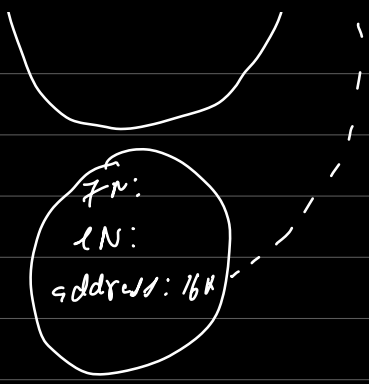
Agenda

1. deep copy and shallow copy
2. polyfill of deep copy
3. flatten an array
4. array and its important function
5. function, IIFE and its use case
6. polyfill of HOF.





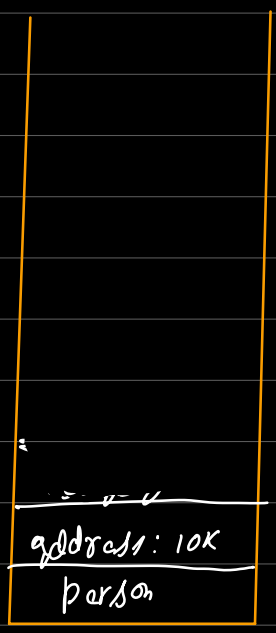

```
function superCloneEffective(input) {
    if(!Array.isArray(input) && typeof input !== "object"){
        return input;   // function or either primitive data type.
    }

    // create new container to clone values.
    let clone = Array.isArray(input) ? [] : {};

    // copy all the keys and values
    for (let key in input) {
        const value = input[key]
        clone[key] = superCloneEffective(value);
    }

    //   retrun the obj
    return clone;
}
```

```
let person = {
    firstName: 'John',
    lastName: 'Doe',
    address: {
        street: 'North 1st street',
        city: 'San Jose',
        state: 'CA',
        country: 'USA'
    },
    friends: ["Steve", "Nikola", "Ray", { name: "Jai", lastName: "Roy" }],
    sayHi: function(){
        console.log("Hi Class!");
    }
};
```



12k: {

firstName: John

lastName: Doe

address: 15k

friends: 16k

sayHi: 17k

}

15K: {

Street: "North 1st street

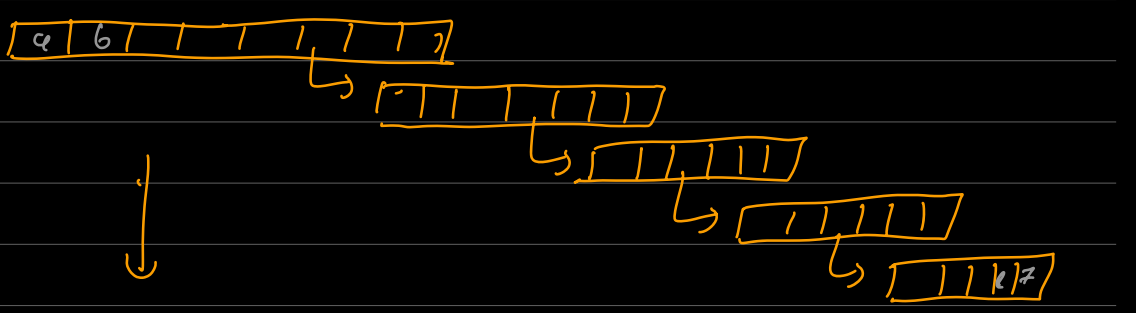
city: "San jose"

State: "CA"

country: "USA"

}

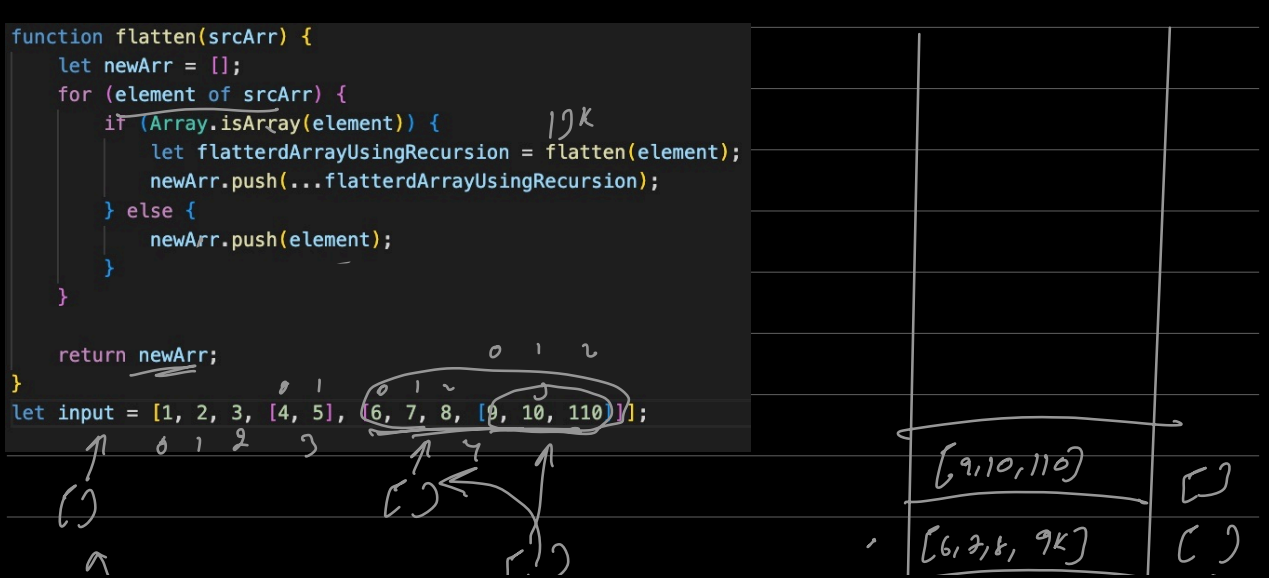
(*) Flattern array



output should be 1D array.

```
function flatten(srcArr) {
    let newArr = [];
    for (element of srcArr) {
        if (Array.isArray(element)) {
            let flatterdArrayUsingRecursion = flatten(element);
            newArr.push(...flatterdArrayUsingRecursion);
        } else {
            newArr.push(element);
        }
    }

    return newArr;
}
let input = [1, 2, 3, [4, 5], [6, 7, 8, [9, 10, 110]]];
```



[6,7,8,9,10,11] . [4,5] [ )

10K [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]

12K: [4, 5)

13K: [6, 7, 8, 9, 10, 11]

14K: [9, 10, 11]

Error:

1	2	3	4	5	6	7	8
0	1	2	3	4	5	6	7

output:

3	4	5

slice (si, ei+1) ≡ slice (2, 4+1)

slice (2, 4+1) ≡ slice (2, 5)

5-2 = 3 (no. of element in b/w 2, 4)

[si, ei]

↳ no. of element in b/w = (ei - si + 1)

= ((ei+1) - si)