37)Array Basics

If you access an item by its index and it does not exist you are just going to get undefined back. You won’t get any error.

38)manuplating arrays with methods

Arrays have ton of methods. See docs – google array mdn and open first link.

Now see pdf. When ever we add some element and remove some element, that method along with updating our array, also returns us added or removed item.

Push – add item to last

Pop – remove item from last

Shift – remove item from beginning

Unshift – add item to starting

One method that allow us to manupilate array from middle is splice. See pdf.

It can be used to add, delete,update array items. So it can perform all 3 operations depending upon number and type of arguments you pass to it.

const notes = [1,2,3,4,5];

notes.splice(3,0,30);

console.log(notes);

output-

**[ 1, 2, 3, 30, 4, 5 ]**

it means starting from index 3, do not remove any elment but add a new element with value 30 at index 3.

const notes = [1,2,3,4,5];

notes.splice(3,2);

console.log(notes);

output-

**[ 1, 2, 3 ]**

it means remove 2 elements from array staring from index 3.

we can also change array items using splice, so we delete array item and provide its replacement. Code-

const notes = ['Note1', 'Note2','Note3'];

notes.splice(1,1,'New Note');

console.log(notes);

output-

**[ 'Note1', 'New Note', 'Note3' ]**

We can also replace more than one item. Code-

const notes = ['Note1', 'Note2','Note3'];

notes.splice(1, 2, 'NewNotes1','NewNotes2','NewNotes3');

console.log(notes);

it means remove 2 elements starting from index 1 and then add elements starting from index 1. Here we have specified more elements than we have removed.

39)Looping over arrays

Here we saw forEach method on arrays.

const notes = ['Note1', 'Note2', 'Note3'];

notes.forEach((item, index) => {

console.log(item);

console.log(index);

});

41)Searching Arrays: Part 1

const notes = ['Note1', 'Note2', 'Note3'];

console.log(notes.indexOf('Note2'));

we get the index of item. If our item is not there then get -1. However if we replace string with objects, like this-

const notes = [{

title: 'My next trip',

body: 'I would like to go to spain'

}, {

title: 'Habbits to work on',

body: 'Exercise. Eating a bot better'

}, {

title: 'Office Modification',

body: 'Get a new seat'

}, {}];

console.log(notes);

console.log(notes.indexOf({}));

we get -1. Why this? We have {}(emty object) in our array. Then why we got -1. It is because indexof uses **===** to compare value. now this code-

let obj1 = {};

let obj2 = {};

console.log(obj1 === obj2);

we get false as output. Because objects points to location in memory. 2 objects will not be equal if they have same set of property-values. They will be equal if they point to same location in memory. Like this-

let obj1 = {};

let obj2 = obj1;

console.log(obj1 === obj2);

here objects are equal because we made them point to same location in memory. Even now we get true-

let obj1 = {};

let obj2 = obj1;

obj2.name = 'Sumeet';

console.log(obj1 === obj2);

this is because they still point to same location we just added a name property on obj2.

Now we cannot use indexOf method for array of objects. For such use cases we have got a lot of methods designed for this useCase. We can use **findIndex()** method. Here is how this method works-

const notes = [{

title: 'My next trip',

body: 'I would like to go to spain'

}, {

title: 'Habbits to work on',

body: 'Exercise. Eating a bot better'

}, {

title: 'Office Modification',

body: 'Get a new seat'

}, {}];

let item = notes.findIndex((note, index) => {

console.log(note);

return note.title === 'Habbits to work on';

});

console.log('This is it');

console.log(item);

To this method we pass a function, this function is executed for each item in array until condition that this function returns become true. This function has 2 arguments, item of array and index of that item. This function returns the index when this condition becomes true.

42)Searching Array: Part 2

We have another method called **find**. It works in same way as findIndex() works. Syntax is also same. Only difference is in **findIndex()** ,we return index when match was found. But in case of find, array item for which we got match is returned. All other things are same.

One more thing arrays are also passed by refrence.it means if we pass array to a function and inside that function we make changes on that array, then our original array is also modified.

43)Filtering Arrays

Filtering means we take existing array and we create new array with some of items based off of whatever filter is. Lets say in our notes application, we want to get all notes in which we have some keyword that we want to search. Here we will get more than one matches. We will use filter method. Code-

const notes = [{

title: 'My next trip',

body: 'I would like to go to spain'

}, {

title: 'Habbits to work on',

body: 'Exercise. Eating a bot better'

}, {

title: 'Office Modification',

body: 'Get a new seat'

}];

const filterNotes = notes.filter((note,index) => {

const isTitleMatch = note.title.toLowerCase().includes('office');

const isBodyMatch = note.body.toLowerCase().includes('office');

return isTitleMatch || isBodyMatch;

});

console.log(filterNotes);

output –

**[ { title: 'Office Modification', body: 'Get a new seat' } ]**

Filter method returns an array. When we call filter method on array we pass it a function. that function has 2 arguments array item and index of array. This function is executed for each item of array. If that function returns true then that particular item for which it ran returns true. If it returns false then that item is not added in returning array.

44)Sorting Array

See documentation of sort function on array. This function takes compareFunction as argument. This function argument to sort is optional.

For array of simple data types, sort function sorts them alphabetically.

Code-

var months = ['March', 'Jan', 'Feb', 'Dec'];

months.sort();

console.log(months);

output –

**[ 'Dec', 'Feb', 'Jan', 'March' ]**

So if we do not pass compare function the array is sorted according to each character's [Unicode](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Values,_variables,_and_literals#Unicode) code point value, according to the string conversion of each element.

Here we have not passed any argument to this function. so it sorted string of array. Now if we have array of objects then we need to pass compare function to sort. This function defines the criteria on which we want to sort array.

So compareFunction defines whether a item should come before or after another item. Unlike other methods beings passed to arraymethods, Compare function is called once for 2 array items. Typically we call first a and second one b. depending on which of these values should come first, we can return 3 values from our compare function. if a should come first than b then we return -1. If b should come first then we return 1. If order does not need to be changed then we return 0.