**Arrays in JavaScript**

* Introduction

Adding new elements

Finding elements

Removing elements

Splitting elements

Combining elements

* Adding elements

const number = [3,4];

// adding element into array in last

number.push (5,6)

// adding element into array in start

number.unshift (1, 2);

// adding element into array in middle

number.splice (2 , 0 , 'a', 'b');

console.log(number);

* Finding elements (primitives)

  const number = [1,2,3,4,5];

    console.log(number.indexOf(6));

    console.log(number.indexOf(2));

    console.log(number.lastIndexOf(5));

console.log(number.indexOf(2) !== -1);  // old method

  console.log(number.includes(4));  // new method

* Finding elements (reference types)

  const courses = [

        {id : 1, name : 'a' },

        {id : 2, name : 'b' },

    ];

//   const course =  courses.find(function(course) {

//         return course.name === 'a';

// });

const course =  courses.findIndex(function(course) {

    return course.name === 'b';

});

 console.log(course);

* Arrow Function

    const courses = [

        {id : 1, name : 'a' },

        {id : 2, name : 'b' },

    ];

const course =  courses.findIndex (course => course.name === 'b');

 console.log(course);

* Removing elements

   const numbers = [1,2,3,4,5];

    //  remove element from end

    // const last =  numbers.pop();

    // remove element form start

    // const start = numbers.shift();

    // remove number in the middle

    numbers.splice (2,2);

    console.log(numbers);

* Emptying an Array

let numbers = [1,2,3,4,5];

let another = numbers;

// solution 1

// numbers = [];

// solution 2

numbers.length = 0 ;              // solution 2 is most efficient

// solution 3

// numbers.splice (0 , numbers.length);

// solution 4

// while(numbers.length > 0)

//     numbers.pop();

//

console.log(numbers);

console.log(another);

* Combining and slicing Arrays

// const first = [1,2,3];

const first = [{id:1}];

const second = [4,5,6];

// combining an array

   const combine =  first.concat(second);

   first[0].id = 10;

//  slicing an array

//   const slice = combine.slice(2,4);  // first way

//   const slice = combine.slice(3);    // second way

     const slice = combine.slice();   // third way

  console.log(combine);

  console.log(slice);

* The spread Operator

    // spread operator is more efficient than concat method

    const first  = [1,2,3];

    const second = [4,5,6];

    const combined = [...first , 'a', ...second , 'b'];

    const copy = [...combined];

    console.log(combined);

    console.log(copy);

* Iterating an array

// first way

 const numbers = [1,2,3,7]

 for (let number of numbers)

    console.log(number);

//  second way

//   numbers.forEach (function (number) {

//     console.log(number);

//   });

//  more simple way

numbers.forEach ( (number, index) => console.log(index, number));

* joining array

 const number = [1,2,3,4];

   const joined =  number.join (',');  // convert array into string

   console.log(joined);

//

const message = 'this is a book';

const parts = message.split (' ');     // convert string into array

console.log(parts);

//

const combine = parts.join('-');

console.log(combine);

* sorting array

const couse  = [

    {id:1 , name : 'Node'},

    {id:2 , name : 'Javascript'}

];

 couse.sort (function (a,b) {

    // a < b => -1

    // a > b => 1

    // a === b => 0

    const nameA = a.name.toUpperCase();

    const nameB = b.name.toUpperCase();

    if (nameA < nameB) return -1;

    if (nameA > nameB) return 1;

    return 0;

 });

 console.log(couse);

* Testing the element of an array

const number = [1,2,-1,3,4];

//  every

// some

     const allpositve = number.some(function(value){

        return value >= 0;

     });

     console.log(allpositve);

* Filtering an array

const numbers = [1,2,3,-1,-2,7];

const filter = numbers.filter(n => n >= 0);

console.log(filter);

* Mapping an array

A Map holds key-value pairs where the keys can be any datatype.

A Map remembers the original insertion order of the keys.

const numbers = [1,2,3,-1,-2,7];

// const filter = numbers.filter(n => n >= 0);

// const item = filter.map( n => '<li>' + n + '</li>');

// const html =  '<ul>' + item.join('') + '/ul';

const items = numbers

.filter (n => n >= 0)

.map (n => ({value : n}))

.filter( obj => obj.value > 1)

.map (obj => obj.value)

console.log(items);

* Reducing an array

const number = [1,-1,2,3];

const sum = number.reduce ((accmulator , currentnumber) => accmulator + currentnumber);

console.log(sum);

Exercise 1 array from range

const numbers = arrayFromRange (1,4);

console.log (numbers);

function arrayFromRange (min , max) {

    let output =[];

    for (let i = min ; i <= max ; i++) {

    output.push(i);

    }

    return output;

};

* Exercise 2 includes

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

console.log(includes(numbers , 1));

function includes (array , searchNumber) {

  for (let element of array)

    if( element === searchNumber)

        return true;

    return false;

}

* Exercise 3 Except

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

const output = except (numbers , [1,4,5])

console.log(output);

function except (array , excluded) {

    const output = [];

    for (let element of array)

        if (!excluded.includes(element))

    output.push(element);

return output;

}

* Exercise 4 moving an element

const number = [1,2,3,4];

const output = move (number , 1, 3);

console.log(output);

function move (array, index , ofset) {

    const positon = index + ofset;

    if (index > array.length || index < 0) {

        console.log ( 'invalid ofset');

    return;

    }

    const output = [...array];

    const element = output.splice(index , 1)[0];

    output.splice (positon , 0 , element);

    return output;

}

* Exercise 5 count occurrences

 const  number = [1,2,3,4,5,5];

 const count = countOccrrence (number , 5);

console.log(count);

function countOccrrence ( array , searchelement) {

return array.reduce((accumalator , current) => {

    const occrence = (current === searchelement)

    console.log(accumalator , current , searchelement)

    return accumalator + occrence;

},0)

}

* Exercise 6 get max