: JAVASCRIPT :

DATE

ARRAYS :

- Arrays methods in detail :

1. toString () :

The toString () returns a string with array values seperated by commas.
And it does not change the original array.

SYNTAX :

```
array.toString ()
```

Example :

```
let bikes = ["Yamaha", "Pulsar", "Hero", "BBT"];
console.log (bikes.toString ());
// "Yamaha, Pulsar, Hero, BBT"
```

11. join () :

The join () returns a new string by concatenating all of the elements in an array. seperated by commas or a specified seperator string.

SYNTAX :

```
array.join (separator);
```

Example :

```
let bikes = ["Yamaha", "Pulsar", "Hero", "TVS"];
console.log (bikes.join ());
// "Yamaha,Pulsar,Hero,TVS"
console.log (bikes.join (" "));
// "Yamaha Pulsar Hero Tvs"
console.log (bikes.join ("-"));
// "Yamaha-Pulsar-Hero-TVS"
```

DATE

III. pop():

The pop() method remove the last element of an array. and return the remove element. And this method changes the original array.

SYNTAX:

```
array.pop();
```

Example:

```
let bikes =["Yamaha", "Pulsar", "Hero'];
console.log (bikes.pop()); // TVS Hero.
console.log (bikes);
// ["Yamaha", "Pulsar"]
```

IV. push():

The push() adds new items to the end of an array. and its changes the length of the array.

returns the new length.

SYNTAX:

```
array.push (item1 , item2 ..... itemx);
```

Example:

```
let bikes = ["Yamaha", "Pulsar"];
console.log (bikes.push ("Hero", "TVS"));
// Hero , TVS → Pulsar.
console.log (bikes);
// ['Yamaha', 'Hero', 'TVS'];
```

V. shift():

The shift() moves the first element and return it.

SYNTAX:

```
array.shift();
```

Example:

```
let bikes = ["Yamaha", "Pulsar"];
console.log (bikes.shift()); // Yamaha
console.log (bikes);
// ['Pulsar'].
```

classmate

PAGE

DATE

VI. unshift():

The unshift() method adds element to beginning & returns new length of array.

SYNTAX:

```
array.unshift(item1, item2, ..... itemX);
```

Example:

```
let bikes = ["Yamaha", "Pulsar"];
console.log(bikes.unshift("Duke")); // Duke
console.log(bikes);
// ['Duke', 'Yamaha', 'Pulsar']
```

VII. delete (operator):

Array elements can be deleted using the Javascript operator delete.

Using delete leaves undefined holes in the array.

SYNTAX:

```
delete array[index];
```

Example:

```
let fruits = ["Apple", "Banana", "Mango"];
delete fruits[1]; // Banana
console.log(fruits);
// ['Apple', <1 empty item>, 'Mango'].
```

VIII. Concat():

The concat() method concatenates (joins) two or more arrays. returns a new array, containing the joined array.

This method does not change the existing arrays.

SYNTAX:

```
array1.concat(array2, array3 .... arrayX);
```

Example:

```
let arr1 = ["1", 2, 3];
let arr2 = [4, 5, 6];
```

```
let arr_new = arr1.concat(arr2);
console.log(arr_new);
// ["1", 2, 3, 4, 5, 6]
```

IX. sort():

The sort() sorts the elements of an array and **overwrite** the original array. sorts the elements as string in **alphabetical** & **ascending** order.

SYNTAX:

```
array.sort(compareFunction)
```

Example:

```
let arr = [15, 9, 2000];
let str = ["Trimbak", "Pathale"];
arr.sort();
console.log(arr); // [9, 15, 2000]
str.sort();
console.log(str); // ["Pathale", "Trimbak"]
```

X. splice():

The splice() method adds and/or removes array elements.

The splice() method overwrites the original array.

SYNTAX:

```
array.splice(index, howmany, item1,
                item2 ..... itemx);
```

Example:

```
let arr = [99, 32, 23, 43, 53, 100];
arr.splice(2, 3, "a", "b", "c");
console.log(arr);
// [99, 32, "a", "b", "c", 100];
```

DATE

XI. slice():

The slice() slice out a piece from an array. it create a new array.

SYNTAX:

```
array.slice(start, end);
```

Example:

```
let numbers = [1,2,3,4,5,6];
let num2 = numbers.slice(1,4);
console.log(num2); // [2,3,4]
```

XII. reverse():

The reverse() method reverse the order of the elements in an array.

This method overwrites the original array.

SYNTAX:

```
array.reverse();
```

Example:

```
let numbers = ["1",2,3,4,5,6];
let string = ["A","B","C","D"];
numbers.reverse();
console.log(numbers);
// [6,5,4,3,2,"1"]
string.reverse();
console.log(string); // ["D","C","B","A"]
```

XIII. isArray():

The isArray() method return true if an object is an array. otherwise false.

check if an object is an array.

SYNTAX:

```
Array.isArray(obj);
```

Example:

```
let numbers = [1,2,3,4,5];
let string = "codeBreaker";
```

DATE

```
console.log (Array.isArray (numbers));
// true
console.log (Array.isArray (string));
// false
```

XIV. indexOf():

The indexOf() method returns the first index (position) of a specified value.
returns -1 if the value is not found.
and it searches from left to right.

Negative start values counts from the last element (but still searches from left to right).

SYNTAX:

```
array.indexOf (item, start)
```

Example:

```
let elements = ["laptop", "headset"];
console.log (elements.indexOf ["headset"]);
// wrong way

console.log (elements.indexOf ("headset", 0)); // 1.
console.log (elements.indexOf ("headset", 2)); // -1.
```

XV. lastIndexOf():

The lastIndexOf() method returns the last index (position) of a specified value, return -1 if the value is not found.
start at a specified index and searches from right to left.

Negative start values counts from the last element (but still searches from right to left).

DATE

SYNTAX:

```
array.lastIndexOf(item, start);
```

Example:

```
let elements = ["Laptop", "Mobile", "Headset"];
console.log(elements.indexOf("Mobile", 0));
// 1
// indexOf(): left to right.
console.log(elements.lastIndexOf("Mobile", 3));
// 1
// lastIndexOf(): right to left.
```

XVI. Find():

The find() method returns the first element in the provided array that satisfied the provided testing function.

If no values satisfy the testing function undefined is returned.

SYNTAX:

```
find((element) => { /* ..... */ });
// arrow function.
```

Elements:

```
const arr = [1, 2, 3, 4, 5];
const found = arr.find(element) =>
                element > 10);
console.log(found); // undefined

const found = arr.find(element) =>
                element > 4);
console.log(found); // 5.
```

DATE

XVII. findIndex():

The findIndex() method returns the index of the first element in an array that satisfy the provided testing function. If no elements satisfy the testing function, then -1 is returned.

SYNTAX:

```
array.findIndex(function(currValue, index, arr), thisValue);
```

Example:

```
let arr = [5, 12, 8, 130, 44];
const isLargest = (element) => element > 13;
console.log(arr.findIndex(isLargest));
// Expected output: 3 (index)
// 130 is large number.
```

XVIII. includes():

The includes() method returns true if an array contains a specified value. (case sensitive).

If the value is not found return false.

SYNTAX:

```
array.includes(element, start);
```

Example:

```
const num = [1, 2, 3];
console.log(num.includes(2));
// true
const string = ["Cat", "Dog", "Elephant"];
console.log(string.includes("Dog"));
// true.
```

DATE

XIX. entries():

The entries() method returns an Array Iterator object with key / value pairs. And this method does not change the original array.

Example:

```
const days= ["sun", "Mon", "Tue", "wed", "Thu",
             "Fri", "sat"];
const day = days.entries();
For(let x of day) {
    console.log (x+"\n");
}
// 0, sun
// 1, Mon
// 2, Tue  & so on.
```

XX. every():

The every() method tests whether all the elements in the array pass the test implementation by the provided function. It returns a Boolean value.

SYNTAX:

```
every(element) => { /* code */ });
```

Example:

```
const arr = [1, 30, 39, 29, 10, 13];
const is_Below = (currValue) => currValue<40;
console.log (arr.every (is_Below));
// true
const isBelow_2 = (currValue) => currValue <30;
console.log (arr.every (isBelow_2));
// true.
```

DATE

XXI. some():

The some() method tests whether at least one element in the array passes the test implemented by the provided function.

It returns true if, in the array, it finds an element for which the provided function return true; otherwise it return false. It doesn't modify the array.

Example:

```
const ages = [3,10,18,20];
ages.some(checkAdult);
function checkAdult(age) {
    return age > 18;
}
// true
```

SYNTAX:

```
array.some(function(value, index, arr), this)
```

XXII. fill():

The fill() method fill specified element in array with a value. method overwrite the original array.

Start and end position can be specified if not all elements will be filled.

SYNTAX:

```
array.fill(value, start, end);
```

DATE

Example:

```
const fruits = ["Apple", "Banana", "orange"];
fruits.fill("kiwi");
// ["kiwi", "kiwi", "kiwi"]

fruits.fill("kiwi", 2, 3);
// ["Apple", "Banana", "orange", "kiwi"]
```

XXIII. copyWithin():

The copyWithin() method copies array elements to another position in the array. and this method overwrites the existing values.

The copy within() method do not add item to the array.

SYNTAX:

```
array.copyWithin(target, start, end);
```

Example:

```
const fruit = ["Banana", "Apple", "Kiwi", "Mango"];
// copy the first two array elements to the
   last two array elements.
fruits.copyWithin(2, 0);   // target, start.

// copy the first two array element to the
   third & fourth position.

const arr = ["Banana", "Mango", "orange",
      "Apple", "kiwi", "Papaya"];

console.log(arr.copyWithin(2, 0, 2));
// ["Banana", "Mango", "Banana", "~~ora~~ Mango",
   "Kiwi", "Papaya"]
```

classmate

PAGE

DATE

XXIV. valueOf():

The valueOF() method returns the array itself and this method does not change the original array.

SYNTAX:

```
array.valueOf();
```

Example:

```
const Fruits = ["Banana", "Mango", "Kiwi"];
const myArr = Fruits.valueOf();
// ["Banana", "Mango", "Kiwi"].

const myArr = Fruits;
// ["Banana", "Mango", "Kiwi"];
```

XXV. forEach():

The forEach() method executes a provided function once for each array element. And this method is not executed for empty elements.

SYNTAX:

```
array.forEach(function(currValue, index, arr), thisValue);
```

Example:

```
// multiply each element
const number = [65, 44, 12, 4];
numbers.forEach(myFunction);

function myFunction(item, index, arr) {
    arr[index] = item * 10;
}
// 650, 440, 120, 40.
```

DATE

XXVI. filter():

The filter method create a new array filled with elements that pass a test provided by a function.

And this method does not change the original array.

SYNTAX:

```
array.filter(function(currValue, index, arr),
                        thisValue);
```

Example:

```
const age = [32, 33, 16, 40];
const result = ages.filter(checkAdult);
function checkAdult(age) {
    return age > 18;
}
// 32, 33, 40.
```

XXVII. reduce():

The reduce() method executes a reducer function for array element and this method return a single value. the function's accumulated result.

SYNTAX:

```
array.filter(function(currValue, index, arr),
                        thisValue);
```

Example:

```
const number = [175, 50, 25];
number.reduce(myFunc);
function myFunc(total, num) {
    return total - num;
}   // 24.
```

DATE

XXVIII. reduceRight():

The reduce() method execute reduced Function for array element.

Works from right to left

SYNTAX:

```
array.reduceRight(function(total, curVal
        curIndex, arr), initialValue);
```

Example:

```
const number = [175, 50, 25];
number.reduceRight(myFunc);
Function myFunc(total, num) {
    return total - num;
}
// -200
```

XXIX. from():

The Array.from() method returns an array from any object with a length property.

And this method returns an array from any iterable object.

SYNTAX:

```
Array.from(object, mapFunction, thisValu
```

Example:

```
// Create an array from a string
console.log(Array.from("Code"));
// Array ["C", "O", "d", "e"];
```