ARRAY (JS)

```
const   arr  = [0, 1, 2, 3, 4, 5, 6]
const   arr2 = ["Kapil", "Rahul", "Jignesh"]
const   arr3 = new Array(1, 2, 3, 4)
```

→ JS arrays are resizable and can contain mix of different data type
→ JS arrays are not associative arrays.
  * array elements cannot be accessed using arbitrary strings as index.
  eg :- arr["Kapil"]; ✗
  arr[0]; ✓
→ JS arrays are zero-indexed.
→ JS arrays copy operations create "shallow copies".

* Shallow Copy :- A shallow copy of an object is a copy whose properties share the same references as of the source object from which the copy was made.

for shallow copy only the top-level properties are copied, not the values of nested object.

- Reassigning the top level properties of the copy does not affect the source object.
- Reassigning nested object properties of the copy does affect the source object.

NOTE :- example given in slice() method. (Example - 2)

Kapil Yadav

* Deep Copy :- Copy whose properties do not share the same references.

→ In JS, built in methods like (spread syntax, concat(), slice(), from() and Object.assign() creates shallow copies.

Way to create deep copy :-

→ One way is, if it can be serialized, we can use JSON.stringify() to convert the object to a JSON string, and then JSON.parse() to convert the string back into a completely new JS object

ex :-

```
const name = ["Kapil", {details : [ 26 , "Delhi", "Single"]}];

const namecopy = JSON.parse(JSON.stringify(name));
```

Array, String Methods

1.) push():- Insert element at the end of Array.

```
arr.push (10);
arr.push (20);
```

2.) pop():- pop out (delete) last element from array.

3.) unshift():- adds the specified elements to the beginning of an array and returns the new length of the array.

```
const arr = [1, 2, 3];
console.log(arr.unshift( 4,5, 6 )); //6 (length of the array)
console.log (arr); // [4,5, 6, 1, 2,3]
```

4.) shift():- removes the first element from array.

```
const arr = [1, 2, 3]
arr.shift( ); // [2, 3]
```

5.) includes() :- checks whether an array includes a certain value or not.

```
const arr = [1, 2, 3,4,5]
console.log ( arr. includes (2)); // Output :- true
```

6.) indexOf () :- Returns the first index, at which given element is present, if not returns -1.

```
          0  1  2
const arr = [1, 2, 1]
console.log (arr.indexOf(1)); // 0
console.log (arr.indexOf(1, 2)) // 2
console.log (arr.indexOf (3)) // -1
```

Syntax :- indexOf (searchElement)
indexOf (searchElement, fromIndex);

NOTE :- fromIndex is optional, and there can be multiple cases.

Case 1:) if fromIndex < 0, then fromIndex + array.length is used.

```
arr = [1, 2, 3]
      -3 -2 -1
```

Case 2:) if fromIndex < (- Arr.length), then 0 is used.

```
          0  1  2
arr = [1, 2, 3]
      -3 -2 -1
```

```
consol.log (arr.indexOf(2,-5)) // 1
                ↳ start from 0.
```

Case 3:) if from >= arr.length, return -1.

7.) join( ) : creates and returns a new string by concatenating all elements of array, separated by commas or specified separator.

Syntax :
```
join()
join(separator);
```

```
const arr = [1,2,3];
const newArr = arr.join();
const newArr2 = arr.join('-');   → separator
consol.log(newArr);  // 1,2,3
consol.log(newArr2); //  1-2-3   (typeof - String)
```

8.) slice( ) : returns a shallow copy of a portion of an array into a new array object selected from start index to end index (end not included).

Syntax :
```
slice()
slice(start)
slice(start, end)
```

* slice(): will create a shallow copy of all the elements of array.

* slice(start): will create a shallow copy from start index till end of the array.

* slice(start, end): will create a shallow copy from start index till end (excluded) of the array.

```
                    0   1   2   3   4
const arr  = [ 1, 2, 3, 4, 5 ]  → excluded
const arr2 = arr.slice(1, 4); // [2, 3, 4]
const arr3 = arr.slice(1);    // [2, 3, 4, 5]
const arr4 = arr.slice();     // [1, 2, 3, 4, 5]
```

```
consok.log(arr); // [1, 2, 3, 4, 5] → original arr remains same.
```

Example 2:-

```
const obj = {
    age: 20
}

const arr = [obj, "kapil"]
const arr2 = arr.slice();
consok.log(arr);  // [{age: 20}, "kapil"]
consok.log(arr2); // [{age: 20}, "kapil"]

obj.age = 40

console.log(arr);  // [{age: 40}, "kapil"]
console.log(arr2); // [{age: 40}, "kapil"]
```

→ obj is nested property, so changes will affect in all versions of referenced/shallow copied of it.

```
arr[1] = "Rahul"
console.log(arr);  // [{age: 40}, "Rahul"]
console.log(arr2); // [{age: 40}, "kapil"]
```

since it is top level property, so changes will not affect in other version of array (shallowed copy or vice versa).

9.) splice() ; splice() changes the contents of an array by removing or replacing existing elements and/or add new elements inplace.

Syntax: splice(start)
splice(start, deleteCount)
splice(start, deleteCount, itemtobeInserted/s)

→ It returns the array containing deleted elements.

```
const days = ["Sunday", "Monday", "Tuesday", "Wednesday"]
                    → start from 0 index
const arr = days.splice(0, 1, "Kapil");  → insert Kapil
                          ↳ Delete 1 element
console.log(days); // ["Kapil", "Monday", "Tuesday", "Wednesday"]
console.log(arr); // ["Sunday"]
```

10. concat( ); used to merge two or more arrays. It returns the new array.

```
const months = ["Jan", "Feb", "March"]
Const months2 = ["April", "May", "June"]

months.push(months2);
console.log(months);
// ["Jan", "Feb", "March", ["April", "May", "June"]]
```

NOTE:- array can store arrays, object inside array.

Using concat inplace of push:

```
months.concat(months2);
console.log(months2);
//["Jan", "Feb", "March", "April", "May", "June"]
```

11. Spread operator:- (...) syntax allows an iterable (array/string), to be expanded in places where 0 or more arguments are expected.

```
months3 = ["July", "August", "September"]
months4 = ["October", "November", "December"]
const all-months = [...months, ...months2, ...months3, ...months4]
```

12. flat() : returns a new array with all sub-array elements concatenated into it.

```
const arr = [1, 2, 3, 4, 5, [6, 7, [8, [9, 10]]]]
arr.flat(depth);
```

① ② ③ ④ → Depth

13. Replace() : replace method of string values returns a new string with one, some or all matches of a pattern replaced by a replacement.

ex →

```
const url = "www.google.com/#india"
const newUrl = url.replace("#", '');
```

14. length : To find the no. of elements in array.

```
const name = ["Kapil", "Rahul", "Vineet"]
name.length;
```

15.) toString(): returns a string representing the specified array and its elements.

ex:- const arr = [1, 2, 'k', 'a', 3];
console.log(arr.toString()); // "1,2,a,1a"

16.) fill(): changes all elements within a range of indices in an array to a static value. It returns the modified array.

syntax:

fill(value)
fill(value, start)
fill(value, start, end) — end is excluded

ex:-

const arr = [1, 2, 3, 4]
console.log(arr.fill(0, 2, 4)); // [1, 2, 0, 0]
console.log(arr.fill(5, 1)); // [1, 5, 5, 5]
console.log(arr.fill(6)); // [6, 6, 6, 6]

17. some(): it tests whether at least one element in the array passes the tests implemented by the provided function. It returns boolean value.

ex:- function isEquals10(element, index, array)
returns element === 10;
}
c.l([2,5, 10, 6,8].some(isEquals10)); //true

18. forEach() :- executes a provided function once for each array element.

syntax:- forEach(callbackFn)
element index array

ex- const arr = ['a', 'b', 'c']
arr.forEach((element) => c.log(element));

19. sort() : to sort the array elements after converting it into strings, then comparing UTF-16 code units values. (lexicographically).

→ It is inplace operation

→ To sort without modifying original array, use toSorted().

Syntax :- sort()
sort(compareFn)

example :-
```
const arr = [1, 30, 4, 21, 100000]
arr.sort();
c.log(arr); // [1, 100000, 21, 30, 4]
```

20. map() : Creates a new array populated with results of calling a provided function on every element in the calling array.

```
const arr = [1,2,3,4,5]
const map1 = arr.map((x) => x*2);
c.log(map1); //[2, 4, 6, 8, 10]
```

21. filter() : creates a shallow copy of a portion of a given array, filtered down to just the elements from the given array that pass the test implemented by the provided function.

Syntax :- filter(callbackfunction)
filter(callbackfunction, thisArg)

ex :- const words = ['kapil', 'Rahul', 'Ravi']
const result = words.filter((word) => word.length>4);

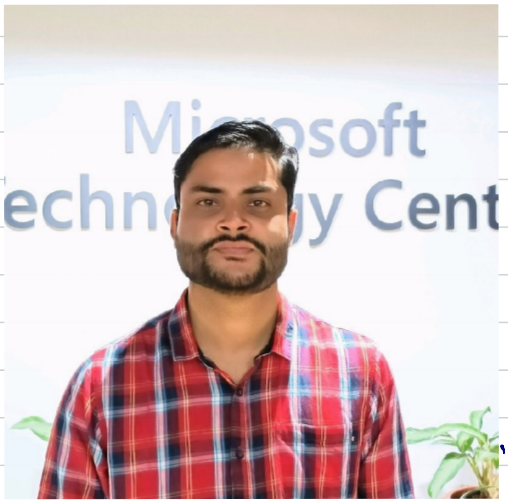
22. Reduce() : It is a function which takes all the values of an array and gives a single output of it. It reduces the array to give a single output.

Syntax : reduce(callbackfn)
reduce(callbackfn, initialValue)

example :- const arr = [1,2,3,4];
const initialvalue = 0;

const sumwithInitial = arr.reduce((accumulator, currentvalue) => accumulator + currentValue, initialValue,);

→ initialValue is optional, if it is given, accumulator will initialized with it, otherwise Accumulator will initialized with first array element.



https://www.linkedin.com/in/kapilyadav22/