Array Manipulation Functions

1. length Property

- **It is used to know the length of array.**
- The length property returns the length of a string (number of characters).
- The length of an empty string is 0.

Example:-

```
let arr = [10,20,30,40,50,60,70,80,90,100];

console.log( arr.length );     //10

console.log( arr[0],arr[4],arr[9],arr[10],arr[-1] );
             //10     50    100     undefined     //undefined

arr.length = 5;

console.log(arr);                    //[ 10, 20, 30, 40, 50 ]

console.log(arr[0],arr[4],arr[5]);   //10 50 undefined

arr.push(60);

console.log(arr);                 //[ 10, 20, 30, 40, 50, 60 ]
```

2.delete property

- The delete property deletes the particular element in an array.
- But memory never release the element and also not deleted the index.
- When the element is deleted but memory won't release instead of that it shows an empty set.
- The `delete` keyword deletes both the value of the property and the property itself.
- After deletion, the property cannot be used before it is added back again.

- The `delete` operator is designed to be used on object properties. It has no effect on variables or functions.
- The `delete` operator should not be used on predefined JavaScript object properties. It can crash your application.

Examples:-

Ex-1.

```
let arr = [10,20,30,40,50,60,70,80,90,100];

console.log( arr.length );      //10

delete arr[4];

console.log(arr.length);                //10



console.log(arr) //[ 10, 20, 30, 40, <1 empty item>, 60, 70, 80, 90, 100 ]

console.log(arr[0],arr[4],arr[9]);     //10   undefined  100
```

Ex-2:-

```
let arr = [1,2,3,4,5];
   console.log(arr);           //[ 1, 2, 3, 4, 5 ]

   arr[0] = "ReactJS";
   arr[1] = "React Native";
   arr[2] = "VueJS";
   arr[3] = "MongoDB";
   arr[4] = "Angular";

console.log(arr);        //[ 'ReactJS', 'React Native', 'VueJS', 'MongoDB', 'Angular' ]
```

3.map()

- **it is used to manipulate the each and every array element**
- **map () function will return "new array".**
- **map () function is "immutable".**

Ex-1:-

```
let arr1 = [1,2,3,4,5];
let arr2 = arr1.map((element, index)=>{
    return element*100;
});
console.log(arr1);              //[ 1, 2, 3, 4, 5 ]
console.log(arr2);              //[ 100, 200, 300, 400, 500 ]
```

Ex-2:-

```
let arr = [10,20,30,40,50];
   console.log(
       arr.map((element,index)=>{
           return element*10;
       }).filter((element,index)=>{
           return element>=300;
       })
   );    //[ 300, 400, 500 ]
```

Ex-3:-

```
let arr = [10,20,30,40,50];
   console.log(
       arr.map((element,index)=>{
           return [element];
       })
   );     //[ [ 10 ], [ 20 ], [ 30 ], [ 40 ], [ 50 ] ]
```

Ex-4:-

```
let arr1 = [1,2,3,4,5];
let arr2 = ["one","two","three","four","five"];
```

```
console.log(
    arr1.map((element,index)=>{
         return [element,arr2[index]];
     })
);
```

4. filter()

- **filter() function creates the new array based on condion**
- **filter() function also introduced in ES6**
- **filter() function also immutable**

Ex:-

```
let arr1 = [10,20,30,40,50];

let arr2 = arr1.filter((element,index)=>{
    return element>=30;
});

console.log(arr1);          //[ 10, 20, 30, 40, 50 ]
console.log(arr2);          //[ 30, 40, 50 ]
```

5. reduce( )

- find the sum of array elements from left to right.

Ex-1:-

```
console.log(
    [1,2,3,4,5].reduce((firstValue,nextValue)=>{
        return firstValue+nextValue;
    })
);    //15
```

Ex-2:-

```
 console.log(
        [1,2,3,4,5].map((element,index)=>{
            return element*10;
        }).filter((element,index)=>{
            return element<=50;
        }).reduce((firstValue,nextValue)=>{
            return firstValue+nextValue;
        })
    );      //150
```

6. reduceRight( )

- it is used to find the summation from right to left.

Ex-1:-

```
console.log(
    ["reactjs","to","welcome"].reduceRight((firstValue,nextValue)=>{
        return firstValue+" "+nextValue;
    })
);
```

7. push( )

- push( ) is used to add the element add the end.

Ex-1:-

```
let arr = [20,30,40];
console.log(arr);          //[ 20, 30, 40 ]

arr.push(50);
console.log(arr);         //[ 20, 30, 40, 50 ]
```

8. unshift( )

- unshift() is used to add the element into an array at the beginning.

Ex-1:-

```
 let arr = [20,30,40];
console.log(arr);          //[ 20, 30, 40 ]
arr.unshift(10);
console.log(arr);        //[ 10, 20, 30, 40, 50 ]
```

9. pop( )

- pop( ) is used to delete the element at the end of an array.

Ex-1:-

```
let arr = [20,30,40];
console.log(arr);          //[ 20, 30, 40 ]
arr.pop();
console.log(arr);        //[ 10, 20, 30, 40 ]
```

10. shift( )

- shift( ) is used to delete the particular element from the beginning of an array.

Ex-1:-

```
let arr = [20,30,40];
console.log(arr);          //[ 20, 30, 40 ]
arr.shift();
console.log(arr);        //[ 20, 30, 40 ]
```

11. findIndex( )

- it is used to know the index of a particular element

Ex-1:-

```
let arr = [10,20,30,40,50,60,70,80,90,100];
console.log(
    arr.findIndex((element,index)=>{
        return element==30;
    })
);
```

Ex-2:-

```
let arr1 = [{"sub":"ReactJS"},
            {"sub":"Angular"},
            {"sub":"NodeJS"},
            {"sub":"MongoDB"},
            {"sub":"VueJS"}];
console.log(
    arr1.findIndex((element,index)=>{
        return element.sub == "ReactJS";
    })
)
```

12. splice( )

- splice() is used to add or remove the elements from array

Ex-1:-

```
let arr1 = [10,20,30,40,50,60,70,80,90,100];

arr1.splice(4,2);
console.log(arr1);      //[10,20,30,40,70,80,90,100]

arr1.splice(4,1);
console.log(arr1);      //[10,20,30,40,80,90,100]

arr1.splice(5,2);
console.log(arr1);      //[ 10, 20, 30, 40, 80 ]
```

```
arr1.splice(1,1);
console.log(arr1);      //[ 10, 30, 40, 80 ]

arr1.splice(1,0,20);
console.log(arr1);      //[ 10, 20, 30, 40, 80 ]

arr1.splice(4,0,50,60,70);
console.log(arr1);      //[10,20,30,40,50,60,70,80]

arr1.splice(8,0,90,100);
console.log(arr1);      // //[10,20,30,40,70,80,90,100]
```

Ex-2:-

```
let arr2 = [10,20,30,40,50,60,70,80,90,100];
arr2.splice(arr2.findIndex((element,index)=>{
    return element == 50;
}),2);
console.log(arr2);
```

Ex-3:-

```
arr2.splice(arr2.findIndex((element,index)=>{
    return element == 90;
}),1);
console.log(arr2);
```

Ex-4:-

```
arr2.splice(arr2.findIndex((element,index)=>{
    return element == 70;
}),0,50,60)
console.log(arr2);
```

Ex-5:-

```
arr2.splice(arr2.findIndex((element,index)=>{
    return element == 100;
}),0,90);
console.log(arr2);
```

Ex-6:-

```
let arr3 = [{"sub":"Angular10"},
            {"sub":"AngularJS"},
            {"sub":"ReactJS"},
            {"sub":"NodeJS"},
            {"sub":"VueJS"}];
arr3.splice(arr3.findIndex((element,index)=>{
    return element.sub === "AngularJS";
}),1);
console.log(arr3);
```

Ex-7:-

```
arr3.splice(arr3.findIndex((element,index)=>{
    return element.sub === "NodeJS";
}),1);
console.log(arr3);
```

13. slice( )

- to get particular elements , then we will use slice()

Ex-1:-

```
let arr1 = [10,20,30,40,50,60,70,80,90,100];
console.log( arr1.slice(4,6) );       //[ 50, 60 ]
console.log(arr1);
console.log( arr1.slice(3,9) );       //[ 40, 50, 60, 70, 80,
 90]
console.log( arr1.slice(8) );        //[ 90, 100 ]
console.log( arr1.slice(7) );        //[ 80, 90, 100 ]
console.log( arr1.slice(7,-1) );     //[ 80, 90 ]
console.log( arr1.slice(7,-3) );     //[]
```

14. find( )

- find( ) is used to checks the particular element in array if it is present then return that element else return undefined.

Ex-1:-

```
let arr1 = ["Angular","ReactJS","NodeJS"];
console.log(
    arr1.find((element,index)=>{
        return element === "React"
    })
);    //undefined
```

Ex-2:-

```
let arr1 = ["Angular","ReactJS","NodeJS"];
console.log(
    arr1.find((element,index)=>{
        return element === "ReactJs"
    })
);    //ReactJs
```

15. includes( )

- includes( ) checks the element in array if the element is present then return true else return false.

Ex-1:-

```
let arr1 = ["Angular","ReactJS","NodeJS"];

console.log( arr1.includes("ReactJS") );     //true
```

16. some( )

- some( ) helps to check the condition in the array if it the condition is true the return true else return false.

Ex-1:-

```
let arr1 = [10,20,30,40,50];

console.log(
    arr1.some((element,index)=>{
        return element>50;
    })
);     //true
```

17. every( )

- if all elements satisfies the condition then it will return true otherwise false

Ex-1:-

```
let arr = [10,20,30,40,50];
console.log(
    arr.every((element,index)=>{
        return element<=50;
    })
);     //true
```

18. fill( )

- It is used to replace existed elements with new elements.

Ex-1:-

```
let arr = [10,20,30,40,50];
console.log( arr.fill(100) );        //[ 100, 100, 100, 100, 100 ]
console.log( arr.fill(200,2) );      //[ 100, 100, 200, 200, 200 ]
console.log( arr.fill(300,1,3) );    //[ 100, 300, 300, 200, 200 ]
```

19. sort( )

- it is used to sort the elements either ascending order or Descending order.

Ex-1:-

```
let arr = [10,50,20,40,30];
console.log(
    arr.sort((arg1,arg2)=>{
        return arg1-arg2;
    })[1]
);     //[ 10, 20, 30, 40, 50 ]
//output : 20 (second min element)
```

Ex-2:-

```
console.log(
    arr.sort((arg1,arg2)=>{
        return arg2-arg1;
    })[1]
);    //output: 40 (second max element)
```

20. indexOf( )

- This function won't return indexes to repeated elements.

Ex-1:-

```
let arr = [10,20,10,30,10,20,40];
arr.forEach((element,index)=>{
    console.log( arr.indexOf(element) );
});     //0 1 0 3 0 1 6
```

Ex-2:-

```
let arr = ["Angular","React","Angular","NodeJS","Angular"];
arr.forEach((element,index)=>{
    console.log( arr.indexOf(element) );
});    //0 1 0 3 0
```

Ex-3:-

```
let arr = [10,20,30,10,40,20,30];
console.log(
    arr.filter((element,index)=>{
        return arr.indexOf(element) == index;
    })
);     [ 10, 20, 30, 40 ]
```

21. flat( )

- The **`flat()`** method creates a new array with all sub-array elements concatenated into it recursively up to the specified depth.

Ex-1:-

```
let arr = [10,[20],30,[40],[50],60];
console.log(arr.flat(1));    //[ 10, 20, 30, 40, 50, 60 ]
```

Ex-2:-

```
let arr = [10,[[20]],[[30]]];
console.log(arr);                    //[ 10, [ [ 20 ] ], [ [ 30 ] ] ]
console.log( arr.flat(Infinity) );
```

Ex-3:-

```
let arr = [10,[[20]],[[[[[[30]]]]]],[30]];
console.log( arr.flat(Infinity).reduce((firstValue,nextValue)=>{
             return firstValue+nextValue;
}) );    //90
```

22. flatMap( )

- The **`flatMap()`** method first maps each element using a mapping function, then flattens the result into a new array. It is identical to a `map()` followed by a `flat()` of depth 1,

but `flatMap()` is often quite useful, as merging both into one method is slightly more efficient.

Ex-1:-

```
let arr1 = [1,3,5,7,9];
let arr2 = [2,4,6,8,10];

console.log(
    arr1.flatMap((element,index)=>{
        return [element,arr2[index]]
    })
);
```

23. reverse( )

- it is used to reverse the array elements

Ex-1:-

```
let arr = [10,20,30,40,50];
console.log( arr.reverse() );           //[ 50, 40, 30, 20, 10 ]
```

Ex-2:-

```
console.log( ["Angular","NodeJS","ReactJS"].reverse() );

//[ 'ReactJS', 'NodeJS', 'Angular' ]
```

24. join( )

- The `join()` method returns the array as a string.
- The elements will be separated by a specified separator. The default separator is comma (,).

Ex-1:-

```
console.log( Array.from("hello").reverse().join("") );

              //olleh
```

25. repeat( )

- it is used to repeat the string.

Ex-1:-

```
    let arr = "Angular";
    console.log( arr.repeat(5) );
```

26. copyWithIn( )

- to shift the array elements.

Ex-1:-

```
let arr = [10,20,30,40,50,60,70,80,90,100];
arr.copyWithin(5);
console.log(arr);
```

Ex-2:-

```
let arr1 = ["Angular","ReactJS","NodeJS"];
arr1.copyWithin(2);
console.log(arr1);      //[ 'Angular', 'ReactJS', 'Angular' ]
```

Ex-3:-

```
let arr2 = [10,20,30,40,50,60,70,80,90,100];
arr2.copyWithin(-3);
console.log(arr2);          //[10,20,30,40,50,60,70,10,20,30]
```

Ex-4:-

```
let arr3 = [100,200,300,400,500];
arr3.copyWithin(-3);
console.log(arr3);        //[ 100, 200, 100, 200, 300 ]
```

Ex-5:-

```
let arr4 = [10,20,30,40,50,60,70,80,90,100];
arr4.copyWithin(3,6);
console.log(arr4);              //[10,20,30,70,80,90,100,80,90,100]
```

Ex-6:-

```
let arr5 = ["Angular","NodeJS","ReactJS","VueJS","MongoDB","React Native"];
arr5.copyWithin(2,4);

console.log(arr5);  //["Angular","NodeJS","MongoDB","React Native","
MongoDB","React Native"]
```

Ex-7:-

```
let arr6 = [10,20,30,40,50,60,70,80,90,100];
arr6.copyWithin(2,4,7);
console.log(arr6);           //[10,20,50,60,70,60,70,80,90,100]
```

27. lastIndexOf( )

- The lastIndexOf() method returns the position of the last occurrence of a specified value in a string.
- The string is searched from the end to the beginning, but returns the index starting at the beginning, at position 0.
- This method returns -1 if the value to search for never occurs.

Ex-1:-

```
let arr1 = [10,20,10,20,10,20];
console.log( arr1.lastIndexOf(10) );    //4
console.log( arr1.lastIndexOf(20) );    //5
console.log( arr1.lastIndexOf(10,4) );  //4
```

```
console.log( arr1.lastIndexOf(20,0) );  //-1
console.log( arr1.lastIndexOf(20,1));   //1
```

28. Array.form( )

- The **Array**. from() function is an inbuilt function in **JavaScript** which creates a new **array** instance from a given **array**. In case of a string, every alphabet of the string is converted to an element of the new **array** instance and in case of integer values, new **array** instance simple take the elements of the given **array**.
- The **`Array.from()`** method creates a new, shallow-copied `Array` instance from an array-like or iterable object.

29. tostring( )

- it is used to convert array to equalent string.

Ex-1:-

```
let arr = ['h','e','l','l','o'];
console.log( arr.toString() );       //h,e,l,l,o
console.log( arr.join("") );         //hello
```

Ex-2:-

```
console.log( ['h','e','l','l','o']
             .toString()
             .replace(/,/g,"") );    //hello
```

30. replace( )

- It is used to replace the particular portion of string with required string.

Ex-1:-

```
let str = "h,e,l,l,o";
console.log( str.replace(",","") );
console.log( str.replace(/,/g,"") );     //hello
```

31. trim( )

- it is used to remove the white spaces at begining and end of string.

Ex-1:-

```
let str = " hello ";
console.log( str.length );          //7
console.log(str.trim().length);     //5
```

32. trimStart( )

- It is used to remove white spaces at beginning of string.

Ex-1:-

```
let str = " hello ";
console.log( str.length );
console.log(str.trimStart().length);//6
```

33. trimEnd( )

- it is used to remove the white spaces at the end of string.

Ex-1:-

```
let str = " hello ";
console.log( str.length );          //7
console.log(str.trimEnd().length);  //6
```

34. padStart ( )

- The **`padStart()`** method pads the current string with another string (multiple times, if needed) until the resulting string reaches the given length. The padding is applied from the start of the current string.
- padStart() is a string method that is used to pad the start of a string with a specific string to a certain length. This type of padding is sometimes called *left pad* or *lpad*. Because the padStart() method is a method of the String object, it must be invoked through a particular instance of the String class.

Ex-1:-

```
let str = "hello";
console.log( str.padStart(20,"reactjs") );
```

35. padEnd ( )

- The **`padEnd()`** method pads the current string with a given string (repeated, if needed) so that the resulting string reaches a given length. The padding is applied from the end of the current string.
- padEnd() is a string method that is used to pad the end of a string with a specific string to a certain length. This type of padding is sometimes called *right pad* or *rpad*. Because the padEnd() method is a method of the String object, it must be invoked through a particular instance of the String class.

Ex-1:-

```
let str = "hello";
console.log( str.padEnd(10,"hello") );     //hellohello
```

36. substr ( )

- The substr() method extracts parts of a string, beginning at the character at the specified position, and returns the specified number of characters.
- To extract characters from the end of the string, use a negative start number (This does not work in IE 8 and earlier).
- The substr() method does not change the original string.

Ex-1:-

```
let str = "welcome to reactjs";
console.log( str.substr(0,7) );      //welcome
console.log( str.substr(8,2) );      //to
console.log( str.substr(11) );       //reactjs
```

37. split( )

- The split() method is used to split a string into an array of substrings, and returns the new array.
- If an empty string ("") is used as the separator, the string is split between each character.
- The split() method does not change the original string.

Ex-1:-

```
let str = "welcome to reactjs";
console.log( str.split(" ") );     //[ 'welcome', 'to', 'reactjs' ]
```

38. substring( )

- The substring() method extracts the characters from a string, between two specified indexes, and returns the new sub string.

Ex-1:-

```
let str = "welcome to reactjs";
console.log( str.substring(0,7) );     //welcome
console.log( str.substring(8,10) );    //to
console.log( str.substring(11,16) );    //react
console.log( str.substring(16,19) );    //js
```

39. forEach( )

- The `forEach()` method calls a function once for each element in an array.
- `forEach()` calls a provided *`callback`* function once for each element in an array in ascending order.
- **arr.forEach()** function calls the provided function once for each element of the array.

Ex-1:-

```
let arr=[1,3,7];
let num=0;
arr.forEach((element,index)=>{
        console.log(element);  //1 3 7
            num++
})
    console.log(num); //3
```

40. for……of( )

- The **`for…of` statement** creates a loop iterating over iterable objects, including: built-in `String`, `Array`, array-like objects (e.g., `arguments` or `NodeList`), `TypedArray`, `Map`, `Set`.
- `for/of` lets you loop over data structures that are iterable such as Arrays, Strings, Maps, NodeLists, and more.

Ex-1:-

```
const array1 = ['a', 'b', 'c'];

for (const element of array1) {
  console.log(element);          //a  b  c
```

```
}
```

41. for……in( )

- The for/in statement loops through the properties of an object.
- The **`for...in` statement** iterates over all enumerable properties of an object that are keyed by strings (ignoring ones keyed by Symbols), including inherited enumerable properties.

Ex-1:-

```
let object = { a: 1, b: 2, c: 3 };

for (let property in object) {
  console.log(`${property}: ${object[property]}`);

                          //a: 1
                          // b: 2
                          // c: 3
}
```

Some Important Questions in Array for Interview

1.What is immutability & Mutability?

creating multiple copies, without midifying source copy called as immutablity.
we will create immutablity by using "..."(spread) operator

"..."(spread) introduced in ES6

```
let arr1 = [10,20,30];
let arr2 = [...arr1];
arr1.push(40);
arr2.push(50);
console.log(arr1);          //[ 10, 20, 30, 40 ]
console.log(arr2);          //[ 10, 20, 30, 50 ]
```

- modifying original copy called as mutability

```
let arr1 = [10,20,30];
let arr2 = arr1;
arr1.push(40);
arr2.push(50);
console.log(arr1);          //[ 10, 20, 30, 40, 50 ]
console.log(arr2);          //[ 10, 20, 30, 40, 50 ]
```

2.What is Map()?

3.what is the use of Array.From()?

4.what is join()?

5.What is the difference between substr() and SubString()?

6.what is the difference between tostring() and join?

Created By Kishan