

# Data Structures: Arrays and Objects (Part 2)

**Relevel**  
by Unacademy



## Pre-Requisites



Basic knowledge on JS Arrays.



Basic knowledge on JS Objects.

# Introduction

As we would be concentrating on hands-on problem solving in this session, please revise the concepts covered in the last 2 sessions:



Functions



Basic Problem Solving



Arrays



Objects

## List of Concepts Involved



Object Functions



Higher-Order Arrays Methods



Freeze, seal, isSealed and is  
functions



map, filter, reduce, forEach, slice,  
sort etc.

## List of Concepts Involved



Array Functions



Destructuring of objects and Array



Rest Operator in Object



Exercises

# Objects in Real World



- In the real world any non-living entity can be referred to as an **Object**.
- A car, bike, smartphone, laptop, chair, table and any simplest thing you can think of is basically an object.
- These objects have some property and functionality. Consider a smartphone.
- Its properties
  - Colour
  - Size of the screen
  - Storage
  - Camera
  - Battery

# Object Function



- `object.freeze()`

The **`Object.freeze()`** method **freezes** an object for preventing the changes. A frozen object can no longer be changed; freezing an object prevents new properties from being added to it, existing properties from being removed, prevents changing the enumerability, configurability, or writability of existing properties, and prevents the values of existing properties from being changed.

```
const obj = {  
  prop: 'relevel'  
};  
  
Object.freeze(obj);
```

```
obj.prop = 'relevels';  
// Throws an error in strict mode  
  
console.log(obj.prop);  
// expected output: relevel
```



## object.seal()

The `Object.seal()` method seals an object, preventing new properties from being added to it and marking all existing properties as non-configurable. Values of present properties can still be changed as long as they are writable.

```
const object1 = {  
  property1: 'relevel'  
};  
  
Object.seal(object1);  
object1.property1 = 'relevels';  
console.log(object1.property1);  
// expected output: relevels
```

```
        delete object1.property1; // cannot  
delete when sealed  
        console.log(object1.property1);  
        // expected output: relevels
```

## object.isSealed()

The `Object.isSealed()` method determines if an object is sealed.

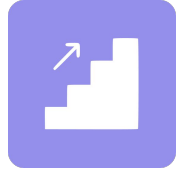
```
const object1 = {  
  property1: 42  
};  
  
console.log(Object.isSealed(object1));  
// expected output: false  
  
Object.seal(object1);  
  
console.log(Object.isSealed(object1));  
// expected output: true
```

## object.is()

The object.is() method is used to compare two values and return true if both are same

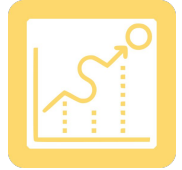
```
Object.is('relevel', 'relevel');  
// true
```

# Higher Order Array Methods



- Higher order functions operate on other functions, either by receiving them as arguments or by returning them. It is a function that accepts a function as a parameter or returns a function as the output.
- In this article, I am going to show you the most commonly used such methods when it comes to arrays.

## Map function



- This is one of the simplest functions you are going to use while working with Arrays. It forms a new array by calling the function passed into it as an argument, on each and every element of the Array. It will map each of the return values of the callback and create a new array.
- The callback passed to the `map()` function can accept any of the three arguments: item, index, array.

## Example

Given an array of integers, create a new array that will have double of each element in the first array and then log it in the console.

Solution:

```
var input = [2, 6, 7, 15];
```

```
var output = input.map(n => n * 2);
```

```
console.log(var); // console: [4, 12, 14, 30]
```

## Example

Given an array of nouns in singular form, create a new array that stores the plural noun of each word in the first array, and log it to the console (assume that the singular nouns can be made plural by adding a 's').

Solution:

```
var input = [ 'pencil', 'kite', 'code' ];  
var output = input.map(word => word + 's');  
console.log(output); // console: ['pencils', 'kites', 'codes']
```



## filter function



- The `filter()` function is used while creating a search bar with a given list of items. The `filter()` method also creates a new array by executing the passed callback on every element of the array, and keeps it in the resulting array IF and ONLY IF the element passes the Boolean test returned by the callback.
- The callback passed into the `filter()` method accepts any of the three arguments: item, index and array; same as the `map()` method.

## Example

Given an array of objects with the city name its corresponding population, create an array with objects from the first array if the population of that particular city is  $\geq 5$  lac.

Solution:

```
var inputArr = [  
  { "name": "Shanghai", "population": 2430000 },  
  { "name": "Los Angeles", "population": 372621 },  
  { "name": "New Delhi", "population": 2180000 },  
  { "name": "Mumbai", "population": 1840000 },  
  { "name": "Chicago", "population": 265598 },  
  { "name": "Houston", "population": 2100263 },  
];
```

```
var outputArr = inputArr.filter( (city) => city.population >= 500000);  
  
console.log(outputArr);  
  
// console: [{ name: "Shanghai", population: 24300000}, { name: "New  
Delhi", population: 21800000}, { name: "Mumbai", population:  
18400000}]
```

## reduce function

- The `reduce()` function creates a new array, executing the callback passed into it on each and every element, and outputs a single value. It does something on every element and keeps a record of the calculations in an accumulator variable and when no more elements are left, it returns the accumulator.
- The `reduce()` function itself takes two inputs:
  - (a) the reducer function also called callback function and
  - (b) a starting point or `initialValue` which is optional.
- The reducer function or the callback accepts 4 arguments: accumulator, `currentItem`, index, array.
- If the optional `initialValue` is given, the accumulator at the first iteration will be equal to the `initialValue` and the `currentItem` will be equal to the first element in the array. Otherwise, the accumulator would be equal to the first item in the input array, and the `currentItem` will be equal to the second item in the array.

## Example

1. Given an array of numbers, find the sum of elements of the array.

Solution:

```
var input = [1, 2, 3, 4];  
var sum = input.reduce((acc, curr) => acc + curr);  
console.log(sum); // console: 10
```

2. Given an array of numbers, find the sum of every element in the array, starting with 5, and log the result to the console

Solution:

```
var input = [1, 2, 3];
```

```
var sum = input.reduce((acc, curr) => acc + curr, 5);
```

```
console.log(sum); // console: 11
```

Note: Here, we are passing the optional initialValue parameter to the reduce() function, saying that we want to start with 5 and do whatever we want inside the callback.

## Example

Find the average of given number in an array.

Solution:

```
var input = [1, 2, 6, 455];
```

```
var average = (input.reduce((acc, curr) => acc + curr)) / input.length;
```

```
console.log(average); // console: 116
```

forEach funtion

The forEach function is similar to the

```
for(let i = 0; i < array.length, i++){
```

```
}
```

syntax.

It loops through the array and runs the given callback for each of the elements of the array.

The callback function passed to the `forEach` function can accept the `currentItem`, `index`, `array`.



## Foreach Function

Example:

Given an array of numbers, log every number to the console.

Solution:

```
var arr = [1, 2, 3, 4, 5, 6];  
arr.forEach(val => console.log(val));
```

console output:

1

2

3

4

5

6

The big difference between the map and forEach method is that the map method creates a new array, "mapping" the return value of the callback and create a new array, while the forEach method just iterates over the array.

## Sort function

Sort function is used to sort the array based on the callback function condition and it should sort

ascending order by default

Sorting of an array of numbers

Example

```
// Sorting of an array of numbers  
  
let numbers = [4, 12, 2, 45, 36, 16, 700]
```

```
// Sorting from Lowest to highest

let lowestToHighest numbers.sort ((a, b) => a - b);

//Output: [2,4,12,16,36,45,700]

//Sorting from Highest to lowest

let highestTOLowest numbers.sort ( (a, b) => b-a);

//Output: [700,45,36,16,12,4,2]
```

## Reverse function



As name indicate it will reverse the array  
How to reverse number in javascript

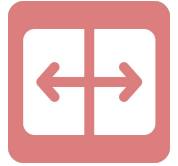
```
const numbers = [9,8,7,6,5,4,3,2,1,0]

const reverseNumbers = numbers.reverse ();

console.log (reverseNumbers);

//Output: [0,1,2,3,4,5,6,7,8,9]
```

## Split function



split function is used to split the string and form an array based on the delimiter. It will accept a delimiter as an argument and it will split the array based on the delimiter. If we pass ';' as an delimiter then the split will happen when the string contains and form an new array.

```
const str = "relevel"  
const splitStr = str.split ("")  
console.log (splitStr) // ['r', 'e', 'l', 'e', 'v', 'e', 'l']
```

## Slice function



The `slice()` method returns a copy of a portion of an array into a new array selected from start to end (end not included) where start and end represent the index of items in that array. The original array will not be modified.

```
const animals = ['ant', 'bison', 'camel', 'duck', 'elephant'];
console.log (animals.slice (2));
// expected output: Array ["camel", "duck", "elephant"]
console.log (animals.slice (2, 4));
// expected output: Array ["camel", "duck"]
```

## Functions on arrays:

There are many Functions used in JScript, these are the few of them along with actions performed by those functions.





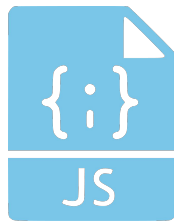
Functions	Action Performed by function
asList()	This function returns a fixed-size list backed by the specified Arrays
binarySearch()	This function Searches for the definite element in the Array with the help of the Binary Search Algorithm
binarySearch(array, fromIndex, toIndex, key, Comparator)	This function Searches a range of the definite Array for the specified Object using the Binary Search Algorithm

<code>deepEquals(Object[] a1, Object[] a2)</code>	This function returns true if the two specified arrays are totally equal to each other.
<code>deepHashCode(Object[] a)</code>	This function returns a string representation of the “deep contents” of the specified Arrays.
<code>deepEquals(Object[] a1, Object[] a2)</code>	This function returns true if the two specified arrays are totally equal to each other.
<code>deepHashCode(Object[] a)</code>	This function returns a string representation of the “deep contents” of the specified Arrays.

<code>setAll(originalArray, functionalGenerator)</code>	setAll is used to make Sets every element of the specified Array using the generator function provided.
<code>parallelSetAll(originalArray, functionalGenerator)</code>	This function sets all the elements of this Array in parallel, using the provided generator function.
<code>hashCode(originalArray)</code>	Returns an integer hashCode of any particular array instance.
<code>copyOf(originalArray, newLength)</code>	This function Copies the identified range of the indicated Array into new Arrays.

## Destructuring in Javascript:

Destructuring in JavaScript is an expression that makes it possible to unpack values from arrays, or properties from objects. We can extract data from arrays and objects and assign them to distinct variables. The value that should be unpacked from the sourced variable is defined on the left-hand side.



Destructuring assignment Using Functions:

```
function getGame(){  
    let games = ["Cricket", "Football" , "Hockey", "Golf"];  
    return games;  
}  
  
let [game_1, game_2] = getGame();  
  
console.log(game_1); //"Cricket"  
console.log(game_2); //"Football"
```

Using Default Values:

```
let [game_1 = "Basketball", game_2 = "Golf"] = ["Cricket"];  
  
console.log(game_1); //"Cricket"  
console.log(game_2); //"Golf"
```

## Object Destructuring

```
let newAvenger = {realName: "Tony Stark", city: "California", heroName: "Iron Man"};

let {realName, city, heroName} = newAvenger;

console.log(realName); //"Tony Stark"
console.log(city); //"California"
console.log(heroName); //"Iron Man"
```

## Declaring the variables before destructuring assignment

```
let newAvenger = {realName: "Tony Stark", city: "California", heroName: "Iron Man"};  
let realName, city, heroName;  
  
({realName, city, heroName} = newAvenger);  
  
console.log(realName); // "Tony Stark"  
console.log(city); // "California"  
console.log(heroName); // "Iron Man"
```



## Combination of Array and Object Destructuring:

```
let newAvenger = {realName: "Tony Stark", city: ["California", "Malibu"], heroName: "Iron Man"};

let {realName: foo, city: bar} = newAvenger;

console.log(foo); // "Tony Stark"
console.log(bar); // ["California", "Malibu"]
```

## Object Destructuring in Nested Objects:

```
let newAvenger = {
  realName: "Tony Stark",
  location: {
    country: "USA",
    city: "California"
  },
  heroName: "Iron Man"
};

let {
  realName: foo,
  location: {
    country: bar,
    city: x
  },
} = newAvenger;

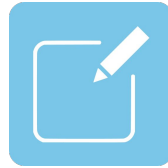
console.log(foo); // "Tony Stark"
console.log(bar); // "USA"
```

## Rest in Object Destructuring:

```
let newAvenger = {  
  realName: "Tony Stark", country: "USA", city: ["California", "Malibu"], heroName: "Iron Man"  
};  
  
let {realName, country, ...restOfDetails } = newAvenger;  
  
console.log(realName); // "Tony Stark"  
console.log(restOfDetails); // { city: [ 'California', 'Malibu' ], heroName: 'Iron Man' }
```

## Rest Parameters:

```
function getProduct(...input){  
    let product = 1;  
    for(let item of input){  
        product *= item;  
    }  
    return product;  
}  
  
console.log(getProduct(1, 2)); // 2  
console.log(getProduct(1, 2, 3, 4)); // 24
```



## Exercises

Write a code to rotate the array in clock-wise direction by one position.

Example:

Input:

```
a = [9, 8, 7, 6, 4, 2, 1, 3];
```

```
n = a.length;
```

Output:

```
[3, 9, 8, 7, 6, 4, 2, 1]
```

You can run the code live and test by clicking [here](#).

## Code:

```
function rotateClockwise(a, n){  
    let lastIdx = n-1;  
    let lastEl = a[lastIdx];  
  
    while(lastIdx > 0){  
        a[lastIdx] = a[lastIdx - 1];  
        lastIdx--;  
    }  
    a[0]= lastEl;  
  
    return a;  
}
```

```
}
```

```
let a = [9, 8, 7, 6, 4, 2, 1, 3];
```

```
let len = a.length;
```

```
console.log(a);
```

```
console.log(rotateClockwise(a, len));
```



Write a method to rearrange an array such that even positioned elements are greater than odd positioned elements.

Example:

Input:

```
a = [1, 3, 2, 2, 5];
```

```
n = a.length;
```

Output:

```
[1, 5, 2, 3, 2]
```

You can run the code live and test by clicking [here](#).

```
function rearrangeArray(a, n) {  
    // Sort the array  
    a.sort();  
    let result = [];  
    let p = 0, q = n - 1;  
    for (let i = 0; i < n; i++) {  
        // Assign even indexes with maximum elements  
        if ((i + 1) % 2 == 0)  
            result[i] = a[q--];  
    }  
}
```

```
// Assign odd indexes with remaining elements
else
    result[i] = a[p++];
}
return result;
}

let a = [ 1, 3, 2, 2, 5 ];
let n = a.length;
console.log(rearrangeArray(a, n));
```

5. Ravi having a set of data and Raju having a set of data and they want to compare both are same or not. Write a program to compare two objects without functions

Code:

Checkout to code [link](#) for reference.

```
// Define the first object
let obj1 = {
  name: "John",
  age: 23,
  degree: "CS",
};

// Define the second object
let obj2 = {
  age: 23,
  degree: "CS",
};
```

```
// Define the function check
function check(obj1, obj2) {
  // Iterate the obj2 using for..in
  for (key in obj2) {
    // Check if both objects do
    // not have the equal values
    // of same key
    if (obj1[key] !== obj2[key]) {
      return false;
    }
  }
  return true;
}
```



**MCQS**

1. Which of the below code will give us the length of an array arr?

- A. `arr.length`
- B. `arr.len`
- C. `length(arr)`
- D. None of the above.



2. What is the functionality of the push method in an array?

- A. Insert another element in the array
- B. increases the length of array by one
- C. returns the deleted element
- D. deletes the first element of array

3. What is the output of the following code?

```
var input = [1,2,4,5,6, 7];  
input.filter(x => x>3);  
console.log(input);
```

- A. 1,2,3,4,5,6
- B. 4,5,6,7
- C. 1,2,3
- D. Error

4. What will be the result of below code?

```
var input = ['h','e','l','l','o'];  
input.join('');
```

- A. 'hello'
- B. hello
- C. 'helo'
- D. None of the above

5. What is the output of the following code?

```
var a = [1,2,4,5,6, 7];  
console.log(a.splice(0,4));
```

- A. [1,2, 4, 5]
- B. [5,6]
- C. [2,4]
- D. [1,2,4]

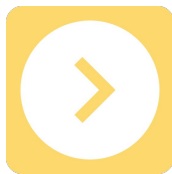
## Practice problem



1. Write a method that accepts an array of integers and updates every element with multiplication of previous and next elements.
1. Move all negative numbers to the end and positive to the beginning of the array without using any predefined sorting method.

## Upcoming Session

Practice problems on Arrays and Objects



**THANK YOU**