Data Structures: Arrays and Objects (Part-2)

Relevel
by Unacademy

List of Concepts Involved

- Object Functions
  - freeze, seal, isSealed and is functions
- Higher-Order Arrays Methods
  - map, filter, reduce, forEach, slice, sort etc.
- Array Functions
- Destructuring of objects and Array
- Problems

Objects

- 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

1) 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
- 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

- Prevent new properties from being added to it
- Mark 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.
- Let’s look at most commonly used such methods of 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 1

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 2

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 ex
every element of the array, and keeps it in the resulting array IF and ONLY IF the element passes the Boolean test returned by th

The callback passed into the filter() method accepts any of the three arguments: item, index and array; same as the map() method

Example 1

Given an array of costs of different items, create a new array with the costs from the input array if the cost is <= 50 rupees and pr

Solution:

```
var input = [100, 300, 65, 15, 45, 30, 5];
var output = input.filter(itemCost => itemCost <= 350);
console.log(output) // console: [15, 45, 30, 5];
```

filter function

Example 2

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 >= 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.

3) 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 Function

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.

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

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

```
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 and creates a new array

```
const str = "relevel"
const splitStr = str.split ("")
console.log (splitStr) // ['r', 'e', 1' 'e', 'v', 'e', '1']
```

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"]
```

Destructuring in Javascript:

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

Object Destructuring

Need for Object destructuring:

Without Object destructuring we access the object items as below

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

let realName = avenger.realName;
let city = avenger.city;
let heroName = avenger.heroName;
```

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

Here we are writing the same code to extract object elements.
Object destructuring saves us from this tedious process and is also efficient enough to save our time.
Object destructuring is more or less similar to Array destructuring.

Object Destructuring

Example 1:

Simple Destructuring assignment on Object

```
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
```

Example 2:

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

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

Object Destructuring

Example 3:

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); // Error : "Unexpected token ="
```

Note:

As we can see here there is an unexpected output as
Error : "Unexpected token ="
This is because the syntax that we used for using previously declared variables in Array destructuring is different for object destructuring.

We need to wrap the entire destructuring assignment in round parenthesis ( ). This is because the { } braces on the left-hand side are considered as a block and not object literal. We need to note that the ( ) should be preceded by a semicolon. Otherwise, it may execute a function from the previous line.

Example 4:

```
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
```

Note:
We have the same variable names as declared inside the object.
Let's see what we get if the names are different.

Object Destructuring

Example 5:

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

As we have used a "location" instead of "city," we get undefined as a result.
Object Destructuring using new variable name:
In case we want to use different names, we can do that as below

Example 6:

```
let newAvenger = {realName: 'Tony Stark', city: 'California', heroName: 'Iron Man'};
let realName, location, heroName;
({realName: fooName, city: location, heroName: barName} = newAvenger);
```

```
console.log(fooName); // Tony Stark
console.log(location); // California
console.log(barName); // Iron Man
```

Here we have assigned the values to fooName, location and barName.

Using Default values:

Example 7:

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

let {realName = 'Downey Jr', location = 'Malibu'} = newAvenger;
console.log(realName); // Tony Stark
console.log(location); // Malibu
```

Here the realName was changed to "Tony Stark," but the location had the default value as "Malibu" as there is no corresponding element location in the extracted object.

Using Computed property name

Example 8:

Using Computed property name, we can specify the name of a property via an expression if we put it in square brackets:
Note that the property name should be a string.

```
let prop = "realName";
let {[prop] : foo} = {realName: 'Tony Stark', city: 'California', heroName: 'Iron Man'};
console.log(foo); // Tony Stark
```

Combination of Array and Object Destructuring

We can also use arrays with objects in object destructuring.
Example 9:

```
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

Example 10:

```
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
console.log(x); // California
```

Rest in Object Destructuring

Example 11:

```
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' }
```

The object elements/properties which do not have the corresponding variable names are assigned to the rest of the details.

Exercises

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]

Code: https://jsfiddle.net/q2wx8aoy/4/

```
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));
```

Write a program to compare Objects without using any in-built function

Input - obj1 - {name:”John”, age:23,degree:”CS”}

obj2 - { age:23,degree:”CS”}

Output - true(Since same key have same values)

Code: https://jsfiddle.net/hfcx2a6j/

```
// 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;
}

// Call the function
console.log(check(obj1, obj2));
```

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.

2) 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