

# Data Structure: Arrays and Objects (Part-1)

**Relevel**  
by Unacademy



# List of Concepts Involved

- Arrays
  - Array Types
  - Array Declaration
  - Array Creation
  - Array Operations
  - Array Basic Problems
- Objects
  - Objects Creation
  - Access Objects
  - Objects Looping
  - Objects Basic Problems

## What are Arrays

Arrays are a type of data structure which are used to store different elements sequentially. In JavaScript, arrays can be stored in a single variable and can be accessed and manipulated using that single variable.

## What are Objects

Object in Javascript is an entity with properties and methods associated with it. These objects have properties in the form of "key: value" pairs; here, the key is a string which can also be referred to as property name, and value can be anything (e.g., string, number, null, Array, function, etc.). Everything in Javascript is an object excluding the primitive data types, including number, string, boolean, null, and undefined.

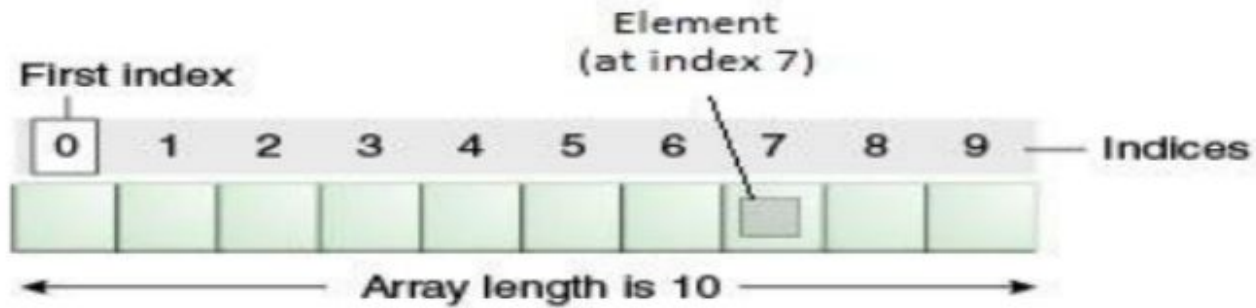
# Arrays

- An array can be described as a group or collection of items or elements stored inside a memory
- The main function of Array is to store a collection of homogenous data of the same type. For example, integers, string, floating numbers, etc
- In an array, all items are kept in a single memory region sequentially.
- Arrays provides a variable-size list data structure, allowing elements to be removed or added in an array. We can add or delete elements based on our requirement
- An array is index-based, with the first element, or the first element in the Array, stored at the 0th index, the second element at the 1st index, the third element at the 2nd index, and so on.
- An array can be used to store primitive values or objects in JavaScript.
- We can create both single-dimensional and multidimensional arrays.

# What are the Advantages of Array?

- Arrays represent numerous data items of a similar type using a single variable.
- In arrays, the elements can be accessed in any order by using the index number.
- Arrays allocate memory in contiguous memory locations for every element present in an array. Henceforth there is no possibility of additional memory being allocated in the case of arrays. This saves memory spillage or overflow of memory and helps to regulate memory usage in any database.
- Arrays can be used to implement other data structures like Stack, Queue, Tree, Linked lists, Graph, etc.
- Size Limit: In the Array, usually we can only store elements of a fixed size. It does not expand dynamically. Due to this, we need to initialize size while declaring arrays itself. This problem is not present in JavaScript. It handles it automatically and size can be dynamically allocated

## Representation of an array



## Scenario without Array

In the below example, we are using five different variables to save our elements one by one.

```
const city1 = 'London';  
const city2 = 'Mumbai';  
const city3 = 'Chennai';  
const city4 = 'Bangalore';  
const city5 = 'Kerala';  
  
// To print all the elements to the console  
  
console.log(city1); // London  
console.log(city2); // Mumbai  
console.log(city3); // Chennai  
console.log(city4); // Bangalore  
console.log(city5); // Kerala
```

## Scenario using Array

Here in this example, we will write the same code as above using an Array.

```
// To create an array of cities to store values  
  
const cities = ['London', 'Mumbai', 'Chennai', 'Bangalore', 'Kerala']  
  
// To print all the elements entered in a array to the console  
  
for (let i = 0; i < cities.length; i++) {  
    console.log (cities [i]);  
}
```

London

Mumbai

Chennai

Bangalore

Kerala



- Without array each value are stored in different variables which result in different memory location, whereas using array the values are stored in contiguous memory allocation
- For developers we can easily maintain code and data manipulation which is good for larger data. Without an array we need to create 100 variables for 100 values but in an array we can insert the values in the single array.
- When we have to create a database with complex and multiple variables to store, it will be difficult to control a database without using an array.
- Using Arrays in the database makes it easier to manage and control the database in the programming language; we can easily create an array and enter or remove values in it.

## Array Declaration and Creation

1. An array can be created using array literal or new keyword in JavaScript

2. Array literal syntax -

```
var stringArray = ["one", "two", "three"]
```

```
var numericArray = [1,2,3,4,5]
```

1. Array using new keyword -

```
var numericArray = new Array(3);
```

```
var numericArray = new Array("cat", "dog");
```

## Single dimensional Array

When we have elements stored in a single dimension sequentially, they are termed as single dimensional arrays.

We can declare and allocate memory to a single dimensional array using a single variable in JavaScript.

Here is an example,

```
var colors = [" Blue "," Green "," Yellow "];  
  
// To print the elements in the console:  
  
console.log(colors);  
  
["red", "blue", "green"]
```

## Syntax for declaring an single dimension array -

```
var <name_of_array> = [element_0, element_1, element_2, element_3, ...  
elementN];
```

*// To create an Single-Dimension array:*

```
const myArray = [ 'h', 'e', 'l', 'l' , 'o' ];
```

*// first element*

```
console.log(myArray[0]); // to print "h"
```

*// second element*

```
console.log(myArray[1]); // to print "e"
```

*// third element*

```
console.log(myArray[2]); // to print "l"
```

*// fourth element*

```
console.log(myArray[3]); // to print "l"
```

*// fifth element*

```
console.log (myArray[4]); // to print  
"o"
```

Output:

```
"h"
```

```
"e"
```

```
"l"
```

```
"l"
```

```
"o"
```

## Multi dimensional Array

- A Multidimensional Array can be described as an array that consists of two or more than two-dimensions
- A multidimensional array is also termed as Array of arrays.
- To build a two-dimensional array, wrap each Array in its own pair of "[]" square brackets.
- In other words, each element of a multidimensional array is an array in and of itself.

## Example for Multi Dimensional Array :

/example to create multi-dimensional Array

```
var items = [  
  [2, 31],  
  [5, 6],  
  [7, 8]  
];  
  
console.log(items[0][0]); // 1  
console.log(items[0][1]); // 2  
console.log(items[1][0]); // 3  
console.log(items[1][1]); // 4  
console.log(items);
```

Output:

```
2,  
3,  
5,  
6  
[[2, 31, [5, 6], [7, 8]]
```

## Common Operations

Length of array

```
const cities = ["London", "Mumbai"," Bangalore"," Kerala"]  
let length =cities.length  
console.log(length) // print 4
```

Note: Array index always starts from 0, which means the first element is stored at index 0 and the last element will be stored at index length-1.

Looping through Array

There are many ways to iterate over the Array. The most common ways of looping through arrays in Javascript are:

### For Loop

```
// Creating an Array  
  
const cities = ['Mumbai', 'Delhi', 'Kolkata', 'Lucknow', 'Chennai'];  
  
// Using loop to iterate over array  
  
for (let i = 0; i < cities.length; i++)
```

```
// Printing array elements using index  
console.log (cities [i]);  
  
// Mumbai  
  
// Delhi  
  
// Kolkata  
  
// Lucknow  
  
// Chennai
```

## Common Operations

for...of loop

This loop helps in iterating over iterable objects like String, Array and so on.

Syntax -

```
for (variable of iterableObject ) {  
  
statement;
```



Example -

```
const obj = [10, 20, 30];  
for (const val of obj) {  
  console.log(val);  
}
```

Output -

10

20

30

# 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 can be -

- Color
  - Size of the screen
  - Storage
  - Camera
  - Battery
- Its functionalities can be -
    - Calling
    - Running applications
    - Browsing
    - Taking pictures/videos

In the same way objects are used in Javascript or in programming in general to represent it as a real-world object. These objects also possess some properties and functionalities more commonly referred to as methods/functions.

## Create Objects

- **Object Literal:** The easiest and the most common way of creating an object is using Object Literals.

*// Creating object using Object literal*

```
const laptop = {  
  make: 'Dell',  
  model: 'Alienware',  
  memory: ['SSD', 'HDD'],  
  cores: 8,  
  memorySize: [256, 512],  
};
```

- **Object.create method:** There is an inbuilt function for creating an object in javascript

```
// Creating object using create method
```

```
const laptop = {  
  make: 'Dell',  
  model: 'Alienware',  
  memory: ['SSD', 'HDD'],  
  cores: 8,  
  memorySize: [256, 512],  
};
```

```
const laptopObj = Object.create(laptop);
```

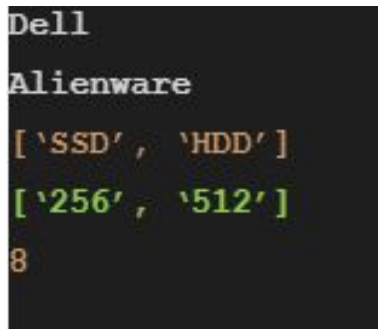
## Access properties of objects

The most popular way to access an object is using the '.' (dot) operator.

```
const laptop = {  
  make: 'Dell',  
  model: 'Alienware',  
  memory: ['SSD', 'HDD'],  
  cores: 8,  
  memorySize: [256, 512],  
};
```

Output:

```
console.log(laptop.make);  
console.log(laptop.model);  
console.log(laptop.memory);  
console.log(laptop.memorySize);  
console.log(laptop.cores);
```



```
Dell  
Alienware  
[ 'SSD', 'HDD' ]  
[ '256', '512' ]  
8
```

## Access properties of objects

Note: We need to make sure that along with `object[]` we are using “ ” to access the property.

Example. `laptop[“model”]`

```
const laptop = { make: 'Dell',  
  model: 'Alienware',  
  memory: ['SSD', 'HDD'],  
  cores: 8,  
  memorySize: [256, 512],  
};
```

```
console.log("Laptop specifications:");  
console.log(laptop);
```

Output:

*Laptop Specifications*


```
{  
  make: 'Dell',  
  model: 'Alienware',  
  memory: ['SSD', 'HDD'],  
  cores: 8,  
  memorySize: [ 256, 512 ]  
}
```

## Access properties of objects

Objects can also have methods as their properties:

```
const laptop = {  
  make: 'Dell',  
  
  model: 'Alienware',  
  
  memory: ['SSD', 'HDD'],  
  
  cores: 8,  
  
  memorySize: [256, 512],  
  
  getModel : function(){  
  
    return this.model;  
  
  }  
};  
  
console.log(laptop.getModel());
```

Output:

A screenshot of a terminal window with a black background and green text. The text 'Alienware' is displayed, representing the output of the JavaScript code's console.log statement.

Alienware

## Looping through an object

### 1. Using a for...in a loop

The most common and easy way to implement loop through an object's properties is by using the **for...in** statement:

Here is an example that implements for...in loop over an object:

```
const array_for_userdata = {  
  name: 'Ben Accord',  
  email: 'ben.english@example.com',  
  age: 25,  
  dob: '08/12/1996',  
  active: true  
};  
// iterate over the user object  
  
for (const key in user)  
{  
  console.log(`${key}: ${array_for_userdata[key]}`);  
}
```



```
// name: John Doe  
  
// email: ben.english@example.com  
  
// age: 25  
  
// dob: 08/12/1995  
  
// active: true
```

## Looping through an object

### 2. Object.keys () method

The Object.keys() method makes it easier to loop over objects.

Object.keys() method takes the object that we want to loop over as an argument and returns the elements in an array that contains all properties names (or keys).

```
const array_name_for_courses = {  
  java: 15,  
  javascript: 78,  
  nodejs: 38,  
  php: 96 };
```

*// to implement conversion object to key's array*

```
const keys = Object.keys(array_name_for_courses);
```

*// to print all keys*

```
console.log(keys);
```

*// [ 'java', 'javascript', 'nodejs', 'php' ]*

*// iterate over object*

```
keys.forEach((key, index) => {  
  console.log(`${key}: ${array_name_for_courses[key]}`);  
});
```

*Output:*

*// java: 15*

*// javascript: 78*

*// nodejs: 38*

*// php: 96*

## Looping through an object

### 3. Object.entries () method

This is the third method known as Object.entries() another method that can be used for traversing an array.

Object.entries() gives outputs of an array of arrays that consists of each inner array having two elements. The first element is considered being the property and the second element is the value.

```
const array_animals = {  
  lion: 1,  
  giraffe: 2,  
  tiger: 3,  
  elephant: 4  
};
```

Output:

```
// [ [ 'lion', 1 ],  
// [ 'giraffe', 2 ],  
// [ 'tiger', 3 ],  
// [ 'elephant', 4 ] ]
```

```
const entries = Object.entries(array_animals);  
console.log(entries);
```

## Looping through an object

### 4. Object.values () method

Looping through The Object.values() method works directly opposite to that of Object.key(). Object.values() method functions by returning the values of all properties in the object as an array.

We can also then loop through the values of the array by using any of the array looping methods.

Example for Object.values() method:

```
const array_of_animals = {  
  lion: 1,  
  horse: 2,  
  giraffe: 3,  
  elephant: 4  
};  
// iterate over object values  
Object.values(array_of_animals).forEach(val => console.log(val));
```

Output:

```
// 1  
// 2  
// 3  
// 4
```

## Looping through an object

### 5. Object.assign method

Object.assign method is used to assign one or more source objects and to form a new object.

```
// clone object using assign method
```

```
const laptop = {  
  make: 'Dell',  
  model: 'Alienware',  
  memory: ['SSD', 'HDD'],  
  cores: 8,  
  memorySize: [256, 512],  
};
```

```
const laptopObjCopy = Object.assign({}, laptop);
```

# JSON JavaScript Object Notation

JSON stands for JavaScript Object Notation. It is basically a widely accepted format to exchange data between various application including client-server applications as well and also a great alternative to XML. The file containing JSON objects is saved with the extension .json.

```
{  
  "car": "Audi",  
  "model": "Q7",  
  "launchYear": 2021,  
  "price": 5000000  
}
```

```
{  
  "Title": "The Cuckoo's Calling"  
  "Author": "Robert Galbraith",  
  "Genre": "classic crime novel",  
  "Detail": {  
    "Publisher": "Little Brown"  
    "Publication_Year": 2013,  
    "ISBN-13": 9781408704004,  
    "Language": "English",  
    "Pages": 494  
  }  
  "Price": [  
    {  
      "type": "Hardcover",  
      "price": 16.65,  
    }  
    {  
      "type": "Kindle Edition",  
      "price": 7.03,  
    }  
  ]  
}
```

Diagram illustrating the structure of a JSON object and its nested elements:

- Object Starts**: Indicated by a red arrow pointing to the opening curly brace `{` at the top.
- Object Starts**: Indicated by a red arrow pointing to the opening curly brace `{` for the `"Detail"` object.
- Value string**: Indicated by a blue arrow pointing to the string value `"Little Brown"`.
- Value number**: Indicated by a yellow arrow pointing to the number value `2013`.
- Object ends**: Indicated by a red arrow pointing to the closing curly brace `}` for the `"Detail"` object.
- Array starts**: Indicated by a green arrow pointing to the opening square bracket `[` for the `"Price"` array.
- Object Starts**: Indicated by a red arrow pointing to the opening curly brace `{` for the first object in the `"Price"` array.
- Object ends**: Indicated by a red arrow pointing to the closing curly brace `}` for the first object in the `"Price"` array.
- Object Starts**: Indicated by a red arrow pointing to the opening curly brace `{` for the second object in the `"Price"` array.
- Object ends**: Indicated by a red arrow pointing to the closing curly brace `}` for the second object in the `"Price"` array.
- Array ends**: Indicated by a green arrow pointing to the closing square bracket `]` for the `"Price"` array.
- Object ends**: Indicated by a red arrow pointing to the closing curly brace `}` for the main object.

## Accessing properties of JSON objects

JSON objects can be stored in Javascript objects and can be accessed similarly as we do for Javascript objects using the ' (dot) operator and object[" "].

**// JSON Objects**

```
const data = {  
  "car": "Audi",  
  
  "models": ["Q7","Q5"],  
  "launchYear": 2021,  
  
  "price": [5000000, 3500000]  
};  
  
console.log(data);  
console.log(data.car);  
  
console.log(data.models[0]);
```



# Difference between JSON Object and Javascript Object:

Through JSON Object and Javascript Object have a close resemblance as both are key:value pairs there are few things we need to note

- Property name/key are always in double quotes " " in JSON.
- The keys are any valid string, but the JSON values can only be one of the six data types (strings, numbers, objects, arrays, boolean, null). Unlike in Javascript Objects, the values can be anything, as we saw earlier.

## Application of JSON

- Helps you to transfer data from a server to client
- Sample JSON file format helps in transmitting and serializing all types of structured data.
- Allows you to perform asynchronous data calls without the need to do a page refresh
- It is widely used for JavaScript-based applications, which includes browser extension and websites.
- You can transmit data between the server and web application using JSON.
- It is used for writing JavaScript-based applications that include browser add-ons.
- Web services and Restful APIs use the JSON format to get public data.

# Problems

## Remove the duplicate element in the array

Input: `const arr = ['a', 'b', 'c', 'a', 'c', 'd', 'a'];`

Output: `[ 'a', 'b', 'c', 'd' ]`

### Steps:

- 1) Create a object to store the array values and their occurrence counts
- 2) Iterate through array and create array values as object key and there occurrence as object value
- 3) Use the `Object.keys()` function to get the keys as an array

Solution Link -> <https://jsfiddle.net/hkxjy28e/>

```
const removeDuplicate = (arr) => {  
  const obj = {};  
  arr.map(item => {  
    if (obj[item])  
      obj[item] += 1;  
    else  
      obj[item] = 0;  
  })  
  return Object.keys(obj);  
}  
  
const arr = ['a', 'b', 'c', 'a', 'c', 'd', 'a'];  
console.log(removeDuplicate(arr));
```

## Write a program to print object different values based on the keys

### Explanation for the above program:

Here we have used an object called car and define the below property

- a) Color – black
- b) Speed – 120Kmph
- c) Brand – Audi
- d) Start and stop as function

calling the property and functionality by framing the meaning full sentence

Using `Object.entries(object)` to get the object individually from the array and iterate using for loop

Output:

My car is Audi and color is Black

My car is BMW and color is Red

## Javascript Code:

Code: -

<https://jsfiddle.net/Lecmy9fz/>

```
const car = [{
  color: 'Black',
  speed: '120Kmph',
  brand: 'Audi',
  start: function () {
    console.log('Car started');
  },
  stop: function () {
    console.log('Car stopped');
  },
},
{
  color: 'Red',
  speed: '100Kmph',
  brand: 'BMW',
  start: function () {
    console.log('Car started');
  },
  stop: function () {
    console.log('Car stopped');
  },
},
].]

for ([key, value] of Object.entries(car)){
  console.log(`My car is ${value.brand} and color is ${value.color}`)
}
```

## Write a program to find the Frequency of all characters in the given string

Input: OCCURRENCE

Output:

O occurring 1 times  
C occurring 3 times  
U occurring 1 times  
R occurring 2 times  
E occurring 2 times  
N occurring 1 times

Explanation:

1. split the string using split() function and it will give the array of letters
2. declare an empty object
3. Iterate the array and add the letter in the object as key and assign 0 as value if the key is not present in the object
4. If the letter is already present as key in the object then add the values with 1
5. Finally iterate the object and log the occurrence

Code: <https://jsfiddle.net/Lkhscuar/>

```
function getOccurrence (str) {  
  const splitStr = str.split("");  
  const occurrenceObj = {};  
  splitStr.map(item => {  
    if (occurrenceObj[item]) {  
      occurrenceObj[item] += 1;  
    }  
    else{  
      occurrenceObj[item] = 1;  
    }  
  })  
  for (let [value, index] of Object.entries(occurrenceObj)){  
    console.log(`${value} occurring ${occurrenceObj[value]} times`);  
  }  
}  
  
getOccurrence("OCCURRENCE")
```

Q 1 – From the below code what is the output

```
const car = {  
  color: 'Black',  
  speed: '120Kmph',  
  brand: 'Audi',  
  start: function () {  
    console.log('Car started');  
  },  
  stop: function () {  
    console.log('Car stopped');  
  },  
}  
const newCar = car;  
newCar.brand = 'BMW';  
console.log(car.brand);
```

- A – Audi
- B – BMW
- C – undefined
- D - Error



Q 2 – JSON stand for

- A – Java Symbol Object Notation
- B – Java Script Object Notation
- C – Java Script Object Nation
- D - None of the above.

Q 3 – InBuilt function to get the keys of the object

- A – Object.keys(obj)
- B – Object.values(obj)
- C – Object.entries(obj)
- D - None of the above.

Q 4 - Which of the following options are correct for accessing the object in Javascript?

```
Const car = { brand: 'Audi' }
```

A- car.brand

B- brand

C- car['brand']

D- All the above

Q 5 – Which of the below function is used to convert string to JSON?

A – JSON.stringify(string)

B – JSON.parse(string)

C – JSON(string)

D – JSON.jsonify(string)

# Practice problem

1. Program to demonstrate destructuring in nested objects
2. Program to clone the object and change the property and then iterate the array of objects using inbuilt functions and to console the object property and frame a meaning full sentence.

**Thank you**