**forEach() method -**

* The forEachforEach() is a built-in array method that lets you loop through each element of an array and perform some action on it.
* It does not return a new array and always returns undefined

**Syntax**

array.forEach(function(element, index, array)

**Parameters**

element → the current element being processed.

index (optional) → the index of the current element.

array (optional) → the entire array on which forEach() was called.

Example:

let numbers = [1, 2, 3, 4];

numbers.forEach((num, index, arr) => {

arr[index] = num \* 2; // updating original array

});

console.log(numbers);

**reduce () method -**

1. This method is used to reduce an array in a single value by executing a callback function on each array element

Syntax –

arrayName.reduce(callback(accumulator,currentValue,index,array),initialValue)

**Parameters:**

* 1. accumulator → stores the running result.
  2. currentValue → the current element being processed.
  3. index (optional) → current element index.
  4. array (optional) → the entire array.
  5. initialValue (optional but recommended) → starting value of the accumulator.

let numbers = [1, 2, 3, 4, 5];

let sum = numbers.reduce((acc, num) => acc + num, 0);

console.log(sum);

|  |  |  |  |
| --- | --- | --- | --- |
| **Feature** | **forEach()** | **map()** | **filter()** |
| **Purpose** | To **execute** a function on each array element | To **transform** each item and return a **new array** | To **filter** items based on a condition and return a **new array** |
| **Returns** | undefined | A **new array** of same length | A **new array** with  **filtered elements** |
| ✅ **Original Array** | Not changed (unless manually modified) | Not changed (returns new one) | Not changed (returns new one) |
| **Use Case** | Performing side effects like logging, updating UI, pushing to another array | Transforming data (e.g., convert strings to uppercase) | Extracting a subset of data (e.g., users who are active) |
| **Chainable** | ❌ No | ✅ Yes | ✅ Yes |
| ✅ **Common Use In** | Loops, console logs, DOM updates | Data transformation | Conditional filtering |

**Introduction to Object**

1. An Object is a real time entity
2. A JavaScript Object is a collection of key:value pairs (properties)
3. **These keys are always in String** (or Symbol) and the value can be anything like **number, string, array, object, function, boolean** etc
4. Objects are used to store structured data and represent the real-world entities like user, products, orders etc.
5. ***Ways to create Objects in JavaScript -***
   1. **Object Literal Method (Most common and recommended)**
   2. Using new Object ()
   3. Using Constructor function (class keyword)
6. **Accessing Object Properties –**

Bracket Notation - [ ]

Dot Notation(**Most Common and recommended**)

**By Object Literal:**

It is the easiest way to create an object.

Object literal is an list of name: value pairs.

**Syntax:**

const obj = {

prop1: value1,

prop2: value2,

prop3: value3,

…,

propN: valueN};

**By Creating an instance of Object Class:**

By using the **new** keyword we can create object.

**Syntax:**

const obj = new Object();

obj.prop1 = value1;

obj.prop2 = value2;

**By using Object Constructor:**

**this** keyword refers to an object(like self in Python).

**Syntax:**

class ClassName {

constructor(parameters) {

this.property1 = parameters;

this.property2 = ...;

}

methodName() {

// method code

}

}

// Create object

let obj = new ClassName(arguments);