ARRAY CONTINUED...

find()

find(callbackFn): This built-in method is designed to return the first element in an array that satisfies a given condition. It iterates through the array and invokes the callback function for each element. If the callback function returns true for a particular element, that element is immediately returned. Otherwise, the iteration continues until the end of the array is reached, and find returns undefined.

Use Cases:
- Finding the first occurrence of a certain condition in an array.
- Searching for an object in an array based on a specific property value.

Syntax

array.find(function(currentValue, index, arr), thisValue)

function(currentValue, index, arr): A function to be called for each element in the array. It takes three arguments:

- **currentValue:** The current element being processed in the array.
- **index (Optional)**: The index of the current element being processed in the array.
- **arr (Optional):** The array find was called upon.
- **thisValue (Optional):** An object to which the keyword this can refer in the callback function. If omitted, undefined is used as the value of this.

e.g.-1

```
<script>
  const numbers = [1, 2, 3, 4, 5];
  function isGreaterThanTwo(currentValue,index,arr) {
    console.log(currentValue + " " + index + " " + arr)
    return currentValue > 2;
  }
  const found = numbers.find(isGreaterThanTwo);
  console.log(found); // Output: 3
</script>
```

e.g. – 2

```
<script>
  const persons = [
    {
      name: 'Florin',
      age: 25
    },
    {
      name: 'Ivan',
      age: 20
    },
    {
      name: 'Liam',
      age: 18
    }
  ];

  function findFlorin(currentValue, index, arr) {
```

```
    return currentValue.name === 'Florin';

  }
  const res = persons.find(findFlorin); // return object
  console.log(res);
  console.log(res.name);
  console.log(res.age)


  const age = persons.find(findFlorin).age;
  console.log(age); //return age
</script>
```

findIndex(callbackFn)

Like find, this method also iterates through the array using a callback function. However, instead of returning the element itself, it returns the index of the first element that meets the condition. If no element matches, it returns -1.

e.g.-1

```
<script>
  const numbers = [1, 2, 3, 4, 5];
  isGreaterThanTwo = (currentValue,index,arr) =>{
    console.log(currentValue + " " + index + " " + arr)
    return currentValue > 2;
  }
  const found = numbers.findIndex(isGreaterThanTwo);
  console.log(found); // 2
</script>
```

forEach()

Use Cases:
Iterating over an array and performing an action for each element, such as logging it, manipulating it, or performing calculations based on its value.

The forEach() method in JavaScript is used to iterate over elements of an array. It executes a provided callback function once for each element in the array, in ascending order. The forEach() method does not return anything (i.e., it returns undefined), and it does not mutate the original array.

Syntax

```
array.forEach(callback(currentValue [, index [, array]])[, thisArg])
```

- **array:** The array you want to iterate over.
- **callback:** A function that will be executed once for each element in the array. This function is called with up to three arguments:
- **currentValue:** The current element being processed in the array.
- **index (optional):** The index of the current element being processed in the array.
- **array (optional):** The array forEach() was called upon.
- The callback function is mandatory.
- **thisArg (optional):** An optional object to be used as this when executing the callback function.

e.g.-1

```
<script>
   const numbers = [10, 23, 43, 425, 3];
   numbers.forEach(consoleItem);

   function consoleItem(item, index, arr) {
      console.log(item + " " + index + " " + arr );
   }
</script>
```

e.g.

```
 <script>
   const numbers = [1, 2, 3, 4, 5, 10, 15, 25];
   // let sum = 0;
   // for ( i = 0 ; i < numbers.length; i++ ){
   //  sum = sum + numbers[i];
   // }

   let sum = 0;
   for (item of numbers){
    sum = sum + item ;
   }

   // let sum = 0;
   // numbers.forEach((item) => {
   //  sum += item;
   // });
   // console.log(sum);
 </script>
```

e.g.

```
<script>
   const letters = ['a', 'b', 'a', 'b', 'c', 'd', 'a'];
   let count = {};
   letters.forEach((item) => {
      if (count[item]) {
         count[item]++;
      } else {
         count[item] = 1;
      }
   });
   console.log(count);
</script>
```

Includes()

The includes() method in JavaScript is used to determine whether an array contains a specified element. It returns true if the array contains the element, and false otherwise.

Syntax

array.includes(searchElement[, fromIndex])

- array: The array you want to search within.

- searchElement: The element you want to search for within the array.
- fromIndex (optional): The position in the array at which to begin the search. If omitted, the search starts at index 0.

Return value:

- true if the array contains the specified element, false otherwise.

e.g.

```
<script>
  const names = ['Florin', 'Ivan', 'Liam'];
  const res = names.includes('Ivan');
  console.log(res);
</script>
```

Slice()

The slice() method in JavaScript is used to extract a section of an array and return a new array containing the extracted elements. It doesn't modify the original array but instead returns a shallow copy of a portion of the array into a new array object.

Syntax:

array.slice([start[, end]])

- array: The array you want to extract a portion from.
- start (optional): The index at which to begin extraction. If omitted, slice() starts from index 0. If negative, it specifies an offset from the end of the array.
- end (optional): The index at which to stop extraction. slice() extracts up to but not including end. If omitted, slice() extracts through the end of the array. If negative, it specifies an offset from the end of the array.

e.g.-1

```
<script>
    const numbers = [1, 2, 3, 4, 5];

    const slice2 = numbers.slice(1);
    console.log(slice2);

    const slice3 = numbers.slice(1, 3);
    console.log(slice3);
</script>
```

e.g.-1

```
<script>
    const numbers = [1, 2, 3, 4, 5];
    const last3 = numbers.slice(-3);
    console.log(last3);

    const last = numbers.slice(-4, -2);
    console.log(last);
</script>
```

Splice()

The splice() method in JavaScript is used to change the contents of an array by removing or replacing existing elements and/or adding new elements in place. It modifies the original array and returns an array containing the deleted elements, if any.

Syntax:

array.splice(start[, deleteCount[, item1[, item2[, ...]]]])

- array: The array you want to modify.
- start: The index at which to start modifying the array. If negative, it specifies an offset from the end of the array. If start is greater than the length of the array, it will be set to the length of the array.
- deleteCount (optional): The number of elements to remove starting from the start index. If omitted or if greater than the number of elements remaining in the array, all elements from start to the end of the array will be deleted.
- item1, item2, etc. (optional): Elements to add to the array, starting from the start index. If you don't specify any elements, only elements will be removed without adding new ones.

e.g.-1

```
<script>
  const numbers = [1, 2, 3, 4, 5];
  numbers.splice(2, 3);
  console.log(numbers);
</script>
```

e.g.-2

```
<script>
  const numbers = [1, 2, 3, 4, 5];
  const deleted = numbers.splice(2, 3);
  console.log(deleted);
</script>
```

e.g.-3

```
<script>
  const numbers = [1, 2, 3, 4, 5];
  const deleted = numbers.splice(2, 3, 6, 7);
  console.log(numbers);
  console.log(deleted);
</script>
```

Sort()

The sort() method in JavaScript is used to sort the elements of an array in place and returns the sorted array. By default, the elements are sorted in ascending order based on their string Unicode code points.

Syntax:

array.sort([compareFunction])

- **array:** The array to be sorted.
- **compareFunction (optional):** An optional function that defines the sort order. If omitted, the array elements are sorted based on their string Unicode code points.

Behavior:

- The sort() method sorts the elements of an array in place and returns the sorted array.
- By default, elements are sorted based on their string Unicode code points. This means that the elements are converted to strings and then compared based on their Unicode code points.
- If compareFunction is provided, the elements are sorted according to the return value of the compareFunction. If a and b are two elements being compared, the function should return:
- A negative value if a should come before b.
- Zero if a and b are considered equal, and their order does not change.
- A positive value if a should come after b.
- If compareFunction is not provided or is undefined, the elements are sorted based on their string representations as mentioned earlier.

e.g.-1

```
<script>
 const names = ["Florin", "Liam", "Jai", "Ivan"];
 names.sort();
 console.log(names);
</script>
```

e.g.-2

```
<script>
   const products = [
      {
         name: 'laptop',
         price: 1500
      },
      {
         name: 'desktop',
         price: 1000
      },
      {
         name: 'phone',
         price: 5001
      }
   ]
   function sortByPrice(a,b){
      // console.log(a.price + " " + a.name )
      // console.log(b.price + " " + b.name )
      return a.price - b.price;
   }
   products.sort(sortByPrice);
   console.log(products);
</script>
```

fill()

The fill() method in JavaScript is used to fill all the elements of an array with a static value. It changes the elements of the original array without creating a new array.
Syntax:
array.fill(value[, start[, end]])

- array: The array to be filled.
- value: The value to fill the array with.
- start (optional): The index at which to start filling the array. If omitted, start defaults to 0.
- end (optional): The index at which to stop filling the array (exclusive). If omitted, end defaults to array.length.

e.g.

```
<script>
  const numbers = [1, 2, 3, 4, 5];
  const num2 = numbers.fill(0);
  console.log(numbers);
  console.log(num2);
</script>
```

e.g.

```
<script>
  const numbers = [1, 2, 3, 4, 5];
  const num2 = numbers.fill(0, 1, 4);
  console.log(numbers);
</script>
```

isArray()

The Array.isArray() method in JavaScript is a utility function used to determine whether a given value is an array. It returns true if the value is an array, and false otherwise.
Syntax:
Array.isArray(value)
e.g.

```
<script>
  const names = ["Florin", "Ivan", "Liam"];
  const str = "Hello world";
  const number = 17;
  console.log(Array.isArray(str));
</script>
```

Every()

The every() method in JavaScript is used to test whether all elements in an array pass a provided condition. It returns true if the provided condition is satisfied for every element in the array; otherwise, it returns false.

Use Cases:

- Checking if all elements in an array satisfy a specific condition.
- Implementing validation logic on array elements.

Syntax:
array.every(callback(currentValue [, index [, array]])[, thisArg])

- array: The array to be tested.
- callback: A function to test each element of the array. It takes up to three arguments:
- currentValue: The current element being processed in the array.
- index (optional): The index of the current element being processed in the array.
- array (optional): The array every() was called upon.
- The callback function must return a truthy or falsy value.
- thisArg (optional): An optional object to be used as this when executing the callback function.

e.g.- 1

```
<script>
   const numbers = [1, 2, 3, 4, 5];
   function isPositive(currentValue, index, arr) {
      return currentValue > 3;
   }
   const res = numbers.every(isPositive);
   console.log(res);
</script>
```

e.g.-2

```
<script>
   const persons = [
   {
   name: 'Florin'
   },
   {
   name: 'Ivan'
   },
   {
   name: 'Liam'
   },
   {
    name :'shivangi',
    Surname: 'Jai'
   }
   ];
   function personTest(person,index,arr){
      return person.name !== undefined
   }
   const res = persons.every( personTest);

   // const res = persons.every( (person,index,arr) => person.name !== undefined);
   console.log(res);
 </script>
```

flat()

The flat() method in JavaScript is used to flatten an array, meaning it converts a multidimensional array into a one-dimensional array by concatenating all nested arrays within it. This method provides a convenient way to work with nested arrays and simplifies array manipulation tasks.

Syntax:

array.flat([depth])

- **array:** The array to be flattened.
- **depth (Optional):** Specifies the depth level of flattening. It determines how deep nested arrays should be flattened. The default value is 1.

e.g.

```
<script>
  const arr = [1, [2, [3, [4]]]];
  // [1, 2, 3, 4];
  const res = arr.flat(Infinity);
  console.log(res);
</script>
```

Note:

You can use Infinity as the depth parameter to completely flatten the array, regardless of its nesting depth:

copywithin()

The copyWithin() method in JavaScript is used to shallow copy a portion of an array to another location in the same array, overwriting its existing elements. This method modifies the original array and returns a reference to the modified array. It's a versatile tool for manipulating arrays efficiently.

Syntax

array.copyWithin(target, start, end)

- target: The index at which to start copying elements to. It represents the beginning of the copied sequence in the target array.
- start: (Optional) The index at which to start copying elements from. It represents the beginning of the sequence to be copied.
- end: (Optional) The index at which to stop copying elements from. It represents the end of the sequence to be copied (excluding this index).

  e.g.-1

  ```
  <script>
    const nums = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9];
    nums.copyWithin(1);
    console.log(nums);
  </script>
  ```

  e.g.-2

  ```
  <script>
    const nums = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9];
    nums.copyWithin(0, 3, 5);
    console.log(nums);
  </script>
  ```

some()

The some() method in JavaScript is used to test whether at least one element in an array passes the test implemented by the provided function. It returns true if at least one element in the array satisfies the condition specified by the callback function, otherwise, it returns false. It's a convenient way to check if any element in an array meets a certain criteria without having to iterate manually.

Syntax

array.some(callback[, thisArg])

- callback: A function that is called for each element in the array. It takes three arguments:
  - element: The current element being processed in the array.
  - index (Optional): The index of the current element being processed.
  - array (Optional): The array on which some() was called.
  - The callback function should return a truthy value to indicate that the condition is met for the current element.
- thisArg (Optional): An object to use as this when executing the callback function.

e.g.-1

```
<script>
 const numbers = [1, 5, 3, 4, 6];
 function greaterThanFour(currentValue, index, arr) {
  return currentValue > 4;
 }
 const res = numbers.some(greaterThanFour);
 console.log(res);
</script>
```

e.g.-2

```
<script>
 const persons = [
  {
   name: "Florin",
   age: 15,
  },
  {
   name: "Ivan",
   age: 19,
  },
  {
   name: "Liam",
   age: 16,
  },
 ];
 function isAdult(currentValue, index, arr) {
  return currentValue.age >= 18;
 }
 const res = persons.some(isAdult);
 console.log(res);
</script>
```

join()

The join() method in JavaScript is used to join all elements of an array into a single string. It allows you to specify a separator that is inserted between each pair of adjacent elements in the resulting string. If no separator is provided, a comma (,) is used as the default separator. This method does not modify the original array; instead, it returns a new string representing the joined elements.

Syntax

array.join(separator)

- separator (Optional): The string used to separate each pair of adjacent elements in the array when they are joined together. If omitted, a comma (,) is used as the default separator.

e.g.-1

```
<script>
 const countries = ["Romania", "USA", "India"];
 const res = countries.join("-");
 console.log(res);
</script>
```

e.g.-2

```
<script>
 const countries = ["Romania"];
 const res = countries.join("-");
 console.log(res);
</script>
```

map()

The map() method in JavaScript is used to iterate over an array and transform each element of the array using a provided callback function. It returns a new array containing the results of applying the callback function to each element of the original array. This method does not modify the original array; instead, it returns a new array with the transformed elements.

Syntax:

- array.map(callback(currentValue[, index[, array]])[, thisArg])
  - callback: A function that is called for each element in the array. It takes three arguments:
    - currentValue: The current element being processed in the array.
    - index (Optional): The index of the current element being processed.
    - array (Optional): The array on which map() was called.
    - The callback function should return the value that will be placed in the new array at the current index.
- thisArg (Optional): An object to use as this when executing the callback function.

e.g. -1

```
<script>
 const numbers = [1, 2, 3, 4, 5];

 function double(currentValue, index, arr)
 {
    return currentValue * 2;
 }
 doubleArray = numbers.map(double);
 console.log(numbers);
```

```
        console.log(doubleArray);

      </script>
```

e.g.- 2

```
<script>
 const products = [
 {
 name: 'laptop',
 price: 1000,
 count: 5
 },
 {
 name: 'desktop',
 price: 1500,
 count: 2
 },
 {
 name: 'phone',
 price: 500,
 count: 10
 }
 ];

 function test(currentValue, index, arr){
  return {
            'product price' : currentValue.price,
            'product count' : currentValue.count
        };
 }
 const totalProductsValue = products.map(test)

//
// const totalProductsValue = products.map(item => {
//    return{
//      'product price' : item.price,
//      'product count' : item.count
//    }
// });
 console.log(totalProductsValue);
</script>
```

filter()

The filter() method in JavaScript is used to create a new array containing only the elements of the original array that satisfy a specified condition. It iterates over each element in the array and applies a callback function to determine whether the element should be included in the filtered array. If the callback function returns true for an element, that element is included in the filtered array; otherwise, it is excluded. This method does not modify the original array; instead, it returns a new array containing the filtered elements.

Syntax:

- array.filter(callback(element[, index[, array]])[, thisArg])
    - callback: A function that is called for each element in the array. It takes three arguments:
    - element: The current element being processed in the array.
    - index (Optional): The index of the current element being processed.
    - array (Optional): The array on which filter() was called.
- The callback function should return true if the element should be included in the filtered array, or false otherwise.
- thisArg (Optional): An object to use as this when executing the callback function.

e.g.

```
<script>
 const numbers = [1, 2, 3, 4, 5, 6];
 function isEven(value) {
  return value % 2 === 0;
 }
 const even = numbers.filter(isEven);
 console.l0og(even);
</script>
```

e.g.

```
<script>
 const people = [
 {
 name: 'Florin', age: 26
 },
 {
 name: 'Ivan',
 age: 18
 },
 {
 name: 'Jai',
 age: 15
 }
 ];
 function isAdult (currentValue, index, arr) {
  return currentValue.age >= 18 ;
 };
 const adults = people.filter(isAdult)
 // const adults = people.filter(person => person.age >= 18);
 console.log(adults);
</script>
```

reduce()

The reduce() method in JavaScript is used to reduce the elements of an array to a single value, applying a specified callback function to each element of the array. It iterates over each element in the array and accumulates a single result by repeatedly calling the callback function, which takes four arguments:

- Accumulator: The accumulated result of the reduction.

- Current Value: The current element being processed in the array.
- Index (Optional): The index of the current element being processed.
- Array (Optional): The array on which reduce() was called.
- The callback function should return the updated value of the accumulator after each iteration. The final value of the accumulator is returned as the result of the reduce() method.

Syntax:
array.reduce(callback(accumulator, currentValue[, index[, array]])[, initialValue])

- callback: A function that is called for each element in the array. It takes four arguments:
  - accumulator: The accumulated result of the reduction.
  - currentValue: The current element being processed in the array.
  - index (Optional): The index of the current element being processed.
  - array (Optional): The array on which reduce() was called.
  - The callback function should return the updated value of the accumulator after each iteration.
- initialValue (Optional): A value to use as the initial value of the accumulator. If not provided, the first element of the array will be used as the initial value, and iteration will start from the second element.

e.g.

```
<script>
 const numbers = [1, 2, 3, 4, 5];
 function sum(accumulator, value) {
  return accumulator + value;
 }
 const total = numbers.reduce(sum);

 console.log(total);
</script>
```

MULTIDIMENSIONAL ARRAY

Multidimensional arrays, also known as nested arrays, are arrays that contain other arrays as elements. This concept extends the idea of arrays beyond one-dimensional structures, allowing for the creation of arrays with multiple dimensions. In JavaScript, while arrays are typically one-dimensional, you can create multidimensional arrays by nesting arrays within arrays.

Create a Multidimensional Array

e.g.1

```
<script>
 let studentsData = [
  ['Jack', 24],
  ['Sara', 23],
  ['Peter', 24]
 ];

 console.log(studentsData)
 console.log(studentsData[0])
 console.log(studentsData[0][1])
```

ACCESSING THE DOUBLE DIMENSIONAL ARRAY : You can access the elements of a multidimensional array using indices (0, 1, 2 …)

```
let x = [
        ['Jack', 24],
        ['Sara', 23],
        ['Peter', 24]
];
```

- e.g.1
  // access the first item
  console.log(x[0]); // ["Jack", 24]

- e.g.
  // access the first item of the first inner array
  console.log(x[0][0]); // Jack

- e.g.
  // access the second item of the third inner array
  console.log(x[2][1]); // 24

ADD AN ELEMENT TO A MULTIDIMENSIONAL ARRAY

You can use the Array's **push() method** or an indexing notation to add elements to a multidimensional array.

Adding Element to the Outer Array

```
<script>
 let studentsData = [
   ["Jack", 24],
   ["Sara", 23],
 ];
 studentsData.push(["Peter", 24]);
 console.log(studentsData);
</script>
```

ADDING ELEMENT TO THE INNER ARRAY

// using index notation

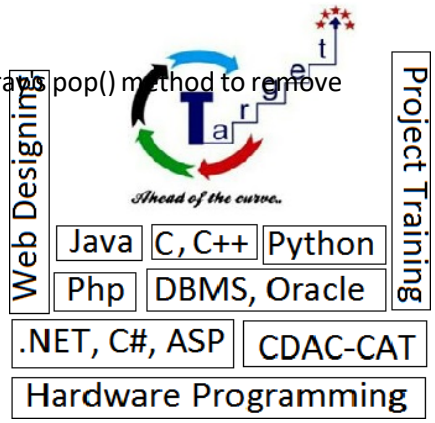
e.g.

```
<script>
 let studentsData = [
   ["Jack", 24],
   ["Sara", 23],
 ];
 studentsData[1][2] = "hello";
 console.log(studentsData); // [["Jack", 24], ["Sara", 23, "hello"]]
```

e.g.
using push()

```
<script>
  let studentsData = [['Jack', 24], ['Sara', 23],];
  studentsData[1].push('hello');
  console.log(studentsData); // [["Jack", 24], ["Sara", 23, "hello"]]
</script>
```

Remove an Element from a Multidimensional Array : You can use the Array's pop() method to remove the element from a multidimensional array.



e.g.

```
<script>
  // remove the array element from outer array
  let studentsData = [
    ["Jack", 24],
    ["Sara", 23],
  ];
  studentsData.pop();
  console.log(studentsData); // [["Jack", 24]]
</script>
```

Remove Element from Inner Array

e.g.

```
<script>
  let studentsData = [
    ["Jack", 24],
    ["Sara", 23],
  ];
  studentsData[1].pop();
  console.log(studentsData); // [["Jack", 24], ["Sara"]]
</script>
```

Iterating over Multidimensional Array

e.g.

```
<script>
  let studentsData = [
    ["Jack", 24],
    ["Sara", 23],
  ];
  for (let i = 0; i < studentsData.length; i++) {
    for (let j = 0; j < studentsData[i].length; j++) {
    console.log(studentsData[i][j]);
    }
  }
</script>
```

e.g.

```
<script>
  let studentsData = [
    ["Jack", 24],
    ["Sara", 23],
  ];

  for (let subarray of studentsData) {
    for (let value of subarray) {
    console.log(value);
    }
  }
/script>
```

e.g.

```
<script>
  let studentsData = [
    ["Jack", 24],
    ["Sara", 23],
  ];
studentsData.forEach((student) => {
    student.forEach((data) => {
       console.log(data);
    });
});
</script>
```

Array.from()

Array.from() is a static method in JavaScript that creates a new array instance from an array-like or iterable object. It provides a convenient way to convert array-like objects (objects with a length property and indexed elements) or iterable objects (objects that implement the iterable protocol, such as Map, Set, or String) into arrays. This method allows you to work with these objects as regular arrays, enabling array manipulation operations.

Syntax:

Array.from(arrayLike[, mapFn[, thisArg]])

- arrayLike: The array-like or iterable object to convert to an array.
- mapFn (Optional): A mapping function to call on each element of the array-like object.
- thisArg (Optional): The value to use as this when executing the mapping function.

e.g.-1

```
<script>
  const str = "1234567";
  // [1, 2, 3, 4, 5, 6, 7]
  const res = Array.from(str);
  console.log(res);
</script>
```

e.g.-2

```
<script>
  const str = "1234567";
  // [1, 2, 3, 4, 5, 6, 7]
  const res = Array.from(str, Number);
  console.log(res);
</script>
```

e.g.-3

```
<script>
  const numbers = [1, 2, 3, 3, 2, 1, 4, 4, 3, 2, 1, 5];
  const res = Array.from(new Set (numbers));
  //const res = new Set(numbers);
  console.log(res);
</script>
```

Akhilesh Kumar Gupta
(Technical Training specialist)
TechCarrel LLP