**Name:** Mounika Gadibavi

**EMP ID:** 003PPD

**1.Array.prototype.at():**

The **at()** method takes an integer value and returns the item at that index, allowing for positive and negative integers. Negative integers count back from the last item in the array.

EX:

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta http-equiv="X-UA-Compatible" content="IE=edge">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Variables</title>

    <script>

        // // console.log(user); //???

        // console.log(user in window ? "exists" : "do not exist"); // ??

        // var user = "mouni";

        var arr1 = [3,6,9];

       // var arr2 = [2,4,6];

        //console.log(arr1)

       // console.log(arr2)

        //arr3 = arr1.concat(arr2);

        console.log(arr1.at(1));

    </script>

</head>

<body>

</body>

</html>

1. **Array.prototype.concat():**

The **concat()** method can be called on any existing array and allows you to concatenate or join multiple arrays together by appending passed arrays or values to the end of the array. For example, you could do the following:

EX:

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta http-equiv="X-UA-Compatible" content="IE=edge">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Variables</title>

    <script>

        // // console.log(user); //???

        // console.log(user in window ? "exists" : "do not exist"); // ??

        // var user = "mouni";

        var arr1 = [3,6,9];

        var arr2 = [2,4,6];

        console.log(arr1)

        console.log(arr2)

        arr3 = arr1.concat(arr2);

        console.log(arr3);

    </script>

</head>

<body>

</body>

</html>

1. **Array.prototype.copyWithin():**

The **copyWithin** () method shallow copies part of an array to another location in the same array and returns it, without modifying its size.

EX:

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta http-equiv="X-UA-Compatible" content="IE=edge">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Variables</title>

    <script>

        const fruits = ["Banana", "Orange", "Apple", "Mango", "Kiwi", "Papaya"];

        console.log(fruits.at(2));

        console.log(fruits.copyWithin(2,1,2));

        </script>

</head>

<body>

</body>

</html>

1. **Array.prototype.entries():**

**The** entries() is another one of the quicker, smaller Array methods as it takes in no arguments and returns us one thing, an 'Array Iterator' object.

EX:

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta http-equiv="X-UA-Compatible" content="IE=edge">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Variables</title>

    <script>

        const fruits = ["Banana", "Orange", "Apple", "Mango"];

        const f = fruits.entries();

        for (let x of f) {

        console.log (x);

        }

    </script>

</head>

<body>

</body>

</html>

1. **Array.prototype.every():**

The **every()** method tests whether all elements in the array pass the test implemented by the provided function. It returns a Boolean value.

The **every()** method executes a function for each array element.

The **every()** method returns **true** if the function returns true for all elements.

The **every()** method returns **false** if the function returns false for one element.

The **every()** method does not execute the function for empty elements.

The **every()** method does not change the original array.

EX:

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta http-equiv="X-UA-Compatible" content="IE=edge">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Variables</title>

<script>

        const ages = [32, 33, 17, 40];

        const age = ages.every(checkAge);

        console.log(age);

        function checkAge(age) {

            return age > 18;

        }

    </script>

</head>

<body>

</body>

</html>

1. **Array.prototype.fill()**

The **fill()** method fills specified elements in an array with a value.

The **fill()** method overwrites the original array.

Start and end position can be specified. If not, all elements will be filled.

EX:

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta http-equiv="X-UA-Compatible" content="IE=edge">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Variables</title>

    <script>

      const fruits = ["Banana", "Orange", "Apple", "Mango"];

     const fillFruits =  fruits.fill("Kiwi",2,4);

     console.log(fillFruits);

    </script>

</head>

<body>

</body>

</html>

1. **Array.prototype.filter()**

The filter() method creates a new array filled with elements that pass a test provided by a function.

The filter() method does not execute the function for empty elements.

The filter() method does not change the original array.

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta http-equiv="X-UA-Compatible" content="IE=edge">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Variables</title>

<script>

        const ages = [32, 33, 16, 40];

        const ageFilter = ages.filter(checkAdult);

        console.log(ageFilter);

        function checkAdult(age) {

            return age >= 18;

        }

    </script>

</head>

<body>

</body>

</html>

**8.Array.prototype.find()**

The find() method returns the value of the first element that passes a test.

The find() method executes a function for each array element.

The find() method returns undefined if no elements are found.

The find() method does not execute the function for empty elements.

The find() method does not change the original array.

EX:

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta http-equiv="X-UA-Compatible" content="IE=edge">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Variables</title>

<script>

        const ages = [3, 10, 18, 20];

        console.log(ages.find(checkAge));

        function checkAge(age) {

            return age > 18;

        }

    </script>

</head>

<body>

</body>

</html>

**9.** **Array.prototype.findIndex()**

The findIndex() method executes a function for each array element.

The findIndex() method returns the index (position) of the first element that passes a test.

The findIndex() method returns -1 if no match is found.

The findIndex() method does not execute the function for empty array elements.

The findIndex() method does not change the original array.

EX:

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta http-equiv="X-UA-Compatible" content="IE=edge">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Variables</title>

<script>

      const ages = [3, 10, 18, 20];

      const findage =  ages.findIndex(checkAge);

    console.log(findage);

function checkAge(age) {

  return age > 18;

}

    </script>

</head>

<body>

</body>

</html>

**10.** **Array.prototype.flat()**

Array.prototype.flat() method is used to flatten an array recursively up to a specified depth. It doesn’t manipulate the original array but creates a new flattened array.

EX:

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta http-equiv="X-UA-Compatible" content="IE=edge">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Variables</title>

    <script>

        const arr1 = [0, 1, 2, [3, 4]];

        console.log(arr1.flat());

        const arr2 = [0, 1, 2, [[[3, 4]]]];

        console.log(arr2.flat(2));

    </script>

</head>

<body>

</body>

</html>

**11. Array.prototype.flatMap():**

FlatMap() is an inbuilt function in JavaScript which is used to flatten the input array element into a new array. This method first of all map every element with the help of mapping function, then flattens the input array element into a new array.

EX:

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta http-equiv="X-UA-Compatible" content="IE=edge">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Variables</title>

    <script>

        let arr1 = [1, 2, 3, 4];

        const value = arr1.flatMap(x => [x \* 2]);

        console.log(value);

// only one level is flattened

const value2 = arr1.flatMap(x => [[x \* 2]]);

console.log(value2);

</script>

</head>

<body>

</body>

</html>

**12. Array.prototype.forEach():**

The forEach() method calls a function for each element in an array.

The forEach() method is not executed for empty elements.

EX:

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta http-equiv="X-UA-Compatible" content="IE=edge">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Variables</title>

    <script>

        let text = "";

        const fruits = ["apple", "orange", "cherry"];

        const value = fruits.forEach(myFunction);

        console.log(text);

        function myFunction(item, index) {

            text += index + ":" + item + "\n";

}

    </script>

</head>

<body>

</body>

</html>

**13. Array.prototype.includes():**

The includes() method returns true if an array contains a specified value.

The includes() method returns false if the value is not found.

The includes() method is case sensitive.

EX:

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta http-equiv="X-UA-Compatible" content="IE=edge">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Variables</title>

    <script>

        const fruits = ["Banana", "Orange", "Apple", "Mango"];

        const value = fruits.includes("Orange");

        console.log(value);

    </script>

</head>

<body>

</body>

</html>

**14. Array.prototype.indexOf():**

The indexOf() method returns the first index (position) of a specified value.

The indexOf() method returns -1 if the value is not found.

The indexOf() method starts at a specified index and searches from left to right.

By default the search starts at the first element and ends at the last.

Negative start values counts from the last element (but still searches from left to right).

EX:

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta http-equiv="X-UA-Compatible" content="IE=edge">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Variables</title>

    <script>

        const fruits = ["Banana", "Orange", "Apple", "Mango"];

        let index = fruits.indexOf("Banana");

        const value = index;

        console.log(value);

    </script>

</head>

<body>

</body>

</html>

**15. Array.prototype.join():**

The join() method returns an array as a string.

The join() method does not change the original array.

Any separator can be specified. The default is comma (,).

EX:

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta http-equiv="X-UA-Compatible" content="IE=edge">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Variables</title>

    <script>

        const fruits = ["Banana", "Orange", "Apple", "Mango"];

        let text = fruits.join(" and ");

        const value = text;

        console.log(value);

    </script>

</head>

<body>

</body>

</html>

**16. Array.prototype.keys():**

The keys() method returns an Array Iterator object with the keys of an array.

The keys() method does not change the original array.

EX:

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta http-equiv="X-UA-Compatible" content="IE=edge">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Variables</title>

    <script>

        const fruits = ["Banana", "Orange", "Apple", "Mango"];

        const keys = fruits.keys();

        let text = "";

        for (let x of keys) {

        text += x + "\n";

}

        console.log(text);

    </script>

</head>

<body>

</body>

</html>

**17. Array.prototype.lastIndexOf():**

The lastIndexOf() method returns the last index (position) of a specified value.

The lastIndexOf() method returns -1 if the value is not found.

The lastIndexOf() starts at a specified index and searches from right to left.

By defalt the search starts at the last element and ends at the first.

Negative start values counts from the last element (but still searches from right to left).

EX:

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta http-equiv="X-UA-Compatible" content="IE=edge">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Variables</title>

    <script>

        const fruits = ["Orange","Apple","Mango","Apple","Banana","Apple"];

        let index = fruits.lastIndexOf("Apple");

        const value = index;

        console.log(value);

    </script>

</head>

<body>

</body>

</html>

**18. Array.prototype.map():**

map() creates a new array from calling a function for every array element.

map() calls a function once for each element in an array.

map() does not execute the function for empty elements.

map() does not change the original array.

EX:

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta http-equiv="X-UA-Compatible" content="IE=edge">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Variables</title>

    <script>

        const numbers = [4, 9, 16, 25];

        const value = numbers.map(Math.sqrt);

        console.log(value);

    </script>

</head>

<body>

</body>

</html>

**19. Array.prototype.pop():**

The pop() method removes (pops) **the last element** of an array.

The pop() method changes the original array.

The pop() method returns the removed element.

EX:

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta http-equiv="X-UA-Compatible" content="IE=edge">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Variables</title>

    <script>

        const fruits = ["Mango", "Orange", "Apple", "Banana"];

        fruits.pop();

        const value = fruits;

        console.log(value);

    </script>

</head>

<body>

</body>

</html>

**20. Array.prototype.push():**

The push() method adds new items **to the end** of an array.

The push() method changes the length of the array.

The push() method returns the new length.

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta http-equiv="X-UA-Compatible" content="IE=edge">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Variables</title>

    <script>

         const fruits = ["Banana", "Orange", "Apple", "Mango"];

         fruits.push("Kiwi", "Lemon");

         console.log(fruits);

    </script>

</head>

<body>

</body>

</html>

**21. Array.prototype.reduce():**

The reduce() method executes a reducer function for array element.

The reduce() method returns a single value: the function's accumulated result.

The reduce() method does not execute the function for empty array elements.

The reduce() method does not change the original array.

EX:

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta http-equiv="X-UA-Compatible" content="IE=edge">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Variables</title>

    <script>

        const numbers = [15.5, 2.3, 1.1, 4.7];

        const value = numbers.reduce(getSum, 0);

        console.log(value);

        function getSum(total, num) {

        return total + Math.round(num);

        }

    </script>

</head>

<body>

</body>

</html>

**22. Array.prototype.reduceRight():**

The reduceRight() method executes a reducer function for each array element.

The reduceRight() method works from right to left.

The reduceRight() method returns a single value: the function's accumulated result.

The reduceRight() method does not execute the function for empty elements.

EX:

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta http-equiv="X-UA-Compatible" content="IE=edge">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Variables</title>

    <script>

        const numbers = [175, 50, 25];

        const value = numbers.reduceRight(myFunc);

        console.log(value);

        function myFunc(total, num) {

            return total - num;

        }

    </script>

</head>

<body>

</body>

</html>

**23. Array.prototype.reverse()**

The reverse() method reverses the order of the elements in an array.

The reverse() method overwrites the original array.

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta http-equiv="X-UA-Compatible" content="IE=edge">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Variables</title>

    <script>

        const fruits = ["Banana", "Orange", "Apple", "Mango"];

        const value = fruits.reverse();

        console.log(value);

    </script>

</head>

<body>

</body>

</html>

**24. Array.prototype.shift():**

The shift() method removes **the first item** of an array.

The shift() method changes the original array.

The shift() method returns the shifted element.

EX:

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta http-equiv="X-UA-Compatible" content="IE=edge">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Variables</title>

    <script>

        const fruits = ["Banana", "Orange", "Apple", "Mango"];

        fruits.shift();

        console.log(fruits);

        console.log(fruits.shift());

    </script>

</head>

<body>

</body>

</html>

**25.** **Array.prototype.slice():**

The slice() method returns selected elements in an array, as a new array.

The slice() method selects from a given start, up to a (not inclusive) given end.

The slice() method does not change the original array.

EX:

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta http-equiv="X-UA-Compatible" content="IE=edge">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Variables</title>

    <script>

        const fruits = ["Banana", "Orange", "Lemon", "Apple", "Mango"];

            const myBest = fruits.slice(1, 4);

            console.log(myBest);

    </script>

</head>

<body>

</body>

</html>

**26.** **Array.prototype.some()**

The some() method checks if any array elements pass a test (provided as a callback function).

The some() method executes the callback function once for each array element.

The some() method returns true (and stops) if the function returns true for one of the array elements.

The some() method returns false if the function returns false for all of the array elements.

The some() method does not execute the function for empty array elements.

The some() method does not change the original array.

EX:

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta http-equiv="X-UA-Compatible" content="IE=edge">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Variables</title>

    <script>

        const ages = [3, 10, 18, 20];

        const value = ages.some(checkAdult);

        console.log(value);

        function checkAdult(age) {

            return age > 18;

        }

    </script>

</head>

<body>

</body>

</html>

**27.** **Array.prototype.sort():**

The sort() sorts the elements of an array.

The sort() overwrites the original array.

The sort() sorts the elements as strings in alphabetical and ascending order.

EX:

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta http-equiv="X-UA-Compatible" content="IE=edge">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Variables</title>

    <script>

        const fruits = ["Banana", "Orange", "Apple", "Mango"];

        console.log( fruits.sort());

    </script>

</head>

<body>

</body>

</html>

**28. Array.prototype.splice():**

The splice() method adds and/or removes array elements.

The splice() method overwrites the original array.

EX:

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta http-equiv="X-UA-Compatible" content="IE=edge">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Variables</title>

    <script>

        const fruits = ["Banana", "Orange", "Apple", "Mango"];

// At position 2, add 2 elements:

        fruits.splice(2, 0, "Lemon", "Kiwi");

        const value = fruits;

        console.log(value);

// At position 2, remove 2 items:

        fruits.splice(2, 2);

        const value2 = fruits;

        console.log(value2);

</script>

</head>

<body>

</body>

</html>

**29. Array.prototype.toLocaleString():**

The **toLocaleString()** method returns a string representing the elements of the array. The elements are converted to Strings using their toLocaleString methods and these Strings are separated by a locale-specific String (such as a comma ",").

EX:

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta http-equiv="X-UA-Compatible" content="IE=edge">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Variables</title>

    <script>

    const array1 = [1, 'a', new Date('21 Dec 1997 14:12:00 UTC')];

    const localeString = array1.toLocaleString('en', { timeZone: 'UTC' });

    console.log(localeString);

</script>

</head>

<body>

</body>

</html>

**30.** **Array.prototype.toString():**

The toString() method returns a string with array values separated by commas.

The toString() method does not change the original array.

EX:

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta http-equiv="X-UA-Compatible" content="IE=edge">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Variables</title>

    <script>

        const fruits = ["Banana", "Orange", "Apple", "Mango"];

        let text = fruits.toString();

        console.log(text);

    </script>

</head>

<body>

</body>

</html>

**31.** **Array.prototype.unshift():**

The unshift() method adds new elements to **the beginning** of an array.

The unshift() method overwrites the original array.

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta http-equiv="X-UA-Compatible" content="IE=edge">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Variables</title>

    <script>

       const fruits = ["Banana", "Orange", "Apple", "Mango"];

        fruits.unshift("Lemon", "Pineapple");

        console.log(fruits);

    </script>

</head>

<body>

</body>

</html>

**32.** **Array.prototype.values():**

The valueOf() method returns the array itself.

The valueOf() method does not change the original array.

fruits.valueOf() returns the same as fruits.

EX:

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta http-equiv="X-UA-Compatible" content="IE=edge">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Variables</title>

    <script>

      const fruits = ["Banana", "Orange", "Apple", "Mango"];

      const myArray = fruits.valueOf();

      console.log(myArray);

    </script>

</head>

<body>

</body>

</html>