UNIT 5 – JAVASCRIPT OBJECTS

1. Array Object

- In JavaScript, the Array object is a global object that is used to store multiple values in a single variable.
- It is one of the most commonly used data structures in JavaScript.
- Arrays in JavaScript can hold elements of any data type, and they use numeric indices to access and manipulate those elements.
- Here are some key details about the Array object in JavaScript along with examples

1.1 Creating an Array:

You can create an array using square brackets '**[ ]**' and initializing it with values.

```
var fruits = ["Apple", "Banana", "Orange"];
```

1.2 Accessing Array Elements:

Array elements are accessed using their index, which starts from 0.

```
console.log(fruits[0]); // Output: "Apple"
console.log(fruits[1]); // Output: "Banana"
```

1.3 Modifying Array Elements:

You can modify array elements by assigning new values to specific indices.

```
fruits[1] = "Grapes";
console.log(fruits); // Output: ["Apple", "Grapes", "Orange"]
```

1.4 Array Length:

The **length** property of an array returns the number of elements it contains.

```
console.log(fruits.length); // Output: 3
```

1.5 Adding and Removing Elements:

JavaScript provides several methods to add and remove elements from an array, such as **push()**, **pop()**, **shift()**, and **unshift()**.

```
fruits.push("Mango"); // Adds an element to the end of the array
console.log(fruits); // Output: ["Apple", "Grapes", "Orange", "Mango"]
fruits.pop(); // Removes the last element from the array
console.log(fruits); // Output: ["Apple", "Grapes", "Orange"]
```

1.6 Iterating Over an Array:

You can loop through the elements of an array using various looping constructs like **for** loop, **forEach()**, **for...of** loop, etc.

```
for (var i = 0; i < fruits.length; i++) {
    console.log(fruits[i]);
}
// Using forEach()
fruits.forEach(function(fruit) {
    console.log(fruit);
});
```

1.7 Array Methods:

Method	Description
concat()	Joins two or more arrays and returns a result.
indexOf()	Searches an element of an array and returns its position.
find()	Returns the first value of an array element that passes a test.
findIndex()	Returns the first index of an array element that passes a test.
forEach()	Calls a function for each element.
includes()	Checks if an array contains a specified element.
sort()	Sorts the elements alphabetically in strings and in ascending order.
slice()	Selects the part of an array and returns the new array.
splice()	Removes or replaces existing elements and/or adds new elements.

➢ Concat()

The concat() method returns a new array by merging two or more values/arrays.

concat() Syntax

The syntax of the concat() method is:

```
arr.concat(value1, value2, ..., valueN)
```

Here, arr is an array.

concat() Parameters

The concat() method takes in an arbitrary number of arrays and/or values as arguments.

concat() Return Value

- Returns a newly created array after merging all arrays/values passed in the argument.

The concat() method first creates a new array with the elements of the object on which the method is called. It then sequentially adds arguments or the elements of arguments (for arrays).

Example

```
var languages1 = ["JavaScript", "Python", "Java"];
var languages2 = ["C", "C++"];
var new_arr = languages1.concat(languages2);
console.log(new_arr);
var new_arr1 = languages2.concat("Lua", languages1);
console.log(new_arr1);
```

Output

```
[ 'JavaScript', 'Python', 'Java', 'C', 'C++' ]
[ 'C', 'C++', 'Lua', 'JavaScript', 'Python', 'Java' ]
```

➢ indexOf()

The indexOf() method returns the first index of occurance of an array element, or **-1** if it is not found.

indexOf() Syntax

The syntax of the indexOf() method is:

```
arr.indexOf(searchElement, fromIndex)
```

Here, arr is an array.

indexOf() Parameters

The indexOf() method takes in:

- searchElement - The element to locate in the array.
- fromIndex (optional) - The index to start the search at. By default, it is **0**.

indexOf() Return Value

Returns the first index of the element in the array if it is present at least once.

Returns -1 if the element is not found in the array.

Example

```
var priceList = [10, 8, 2, 31, 10, 1, 65];
var index1 = priceList.indexOf(31);
console.log(index1); // 3
var index2 = priceList.indexOf(10);
console.log(index2); // 0
var index3 = priceList.indexOf(10, 1);
console.log(index3); // 4
var index4 = priceList.indexOf(69.5);
console.log(index4); // -1
```

Output

```
3
0
4
-1
```

➢ find()

The find() method returns the value of the first array element that satisfies the provided test function.

find() Syntax

The syntax of the find() method is:

```
arr.find(callback(element, index, arr),thisArg)
```

Here, arr is an array.

find() Parameters

The find() method takes in:

- callback - Function to execute on each element of the array. It takes in:
  - element - The current element of array.
- thisArg (optional) - Object to use as this inside callback.

find() Return Value

Returns the value of the first element in the array that satisfies the given function.

Returns undefined if none of the elements satisfy the function.

Example

```
let numbers = [1, 3, 4, 9, 8];
function isEven(element) {
  return element % 2 == 0;
}
let evenNumber = numbers.find(isEven);
console.log(evenNumber);
```

Output

```
4
```

➢ findIndex()

The findIndex() method returns the index of the first array element that satisfies the provided test function or else returns -1.

findIndex() Syntax

The syntax of the findIndex() method is:

```
arr.findIndex(callback(element, index, arr),thisArg)
```

Here, arr is an array.

findIndex() Parameters

The findIndex() method can take **two** parameters:

- callback - Function to execute on each element of the array. It takes in:
  - element - The current element of array.
- thisArg (optional) - Object to use as this inside callback.

findIndex() Return Value

- Returns the **index** of the **first element** in the array that satisfies the given function.
- Returns **-1** if none of the elements satisfy the function.

Example

```
function isEven(element) {
  return element % 2 == 0;
}
let numbers = [1, 45, 8, 98, 7];
let firstEven = numbers.findIndex(isEven);
console.log(firstEven
```

Output

```
2
```

➢ forEach()

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

forEach() Syntax

The syntax of the forEach() method is:

```
arr.forEach(callback(currentValue), thisArg)
```

Here, arr is an array.

forEach() Parameters

The forEach() method takes in:

- callback - The callback function to execute on every array element. It takes in:
  - currentValue - The current element being passed from the array.
- thisArg (optional) - Value to use as this when executing callback. By default, it is undefined.

Example

```
let numbers = [1, 3, 4, 9, 8];
function computeSquare(element) {
  console.log(element * element);
}
numbers.forEach(computeSquare
```

Output

```
1
9
16
81
64
```

➢ includes()

The includes() method checks if an array contains a specified element or not.

includes() Syntax

The syntax of the includes() method is:

```
arr.includes(valueToFind, fromIndex)
```

Here, arr is an array.

includes() Parameters

The includes() method can take **two** parameters:

- searchValue- The value to search for.
- fromIndex (optional) - The position in the array at which to begin the search. By default, it is **0**.

includes() Return Value

The includes() method returns:

- true if searchValue is found anywhere within the array
- false if searchValue is not found anywhere within the array

Example

```
let languages = ["JavaScript", "Java", "C"];
let check = languages.includes("Java");
console.log("Output\n"+check);
```

Output

```
Output
true
```

➢ sort()

The sort() method sorts the items of an array in a specific order (ascending or descending).

sort() Syntax

The syntax of the sort() method is:

```
arr.sort(compareFunction)
```

Here, arr is an array.

sort() Parameters

The sort() method takes in:

- compareFunction (optional) - It is used to define a custom sort order.

sort() Return Value

- Returns the array after sorting the elements of the array in place (meaning that it changes the original array and no copy is made).
-

Example

```
let city = ["Shahada", "Nanadurbar", "Dhule", "Shirpur"];
let sortedArray = city.sort();
console.log(sortedArray);
```

Output

```
[ 'Dhule', 'Nanadurbar', 'Shahada', 'Shirpur' ]
```

➢ slice()

The slice() method returns a shallow copy of a portion of an array into a new array object.

slice() Syntax

The syntax of the slice() method is:

```
arr.slice(start, end)
```

Here, arr is an array.

slice() Parameters

The slice() method takes in:

- start (optional) - Starting index of the selection. If not provided, the selection starts at start **0**.
- end (optional) - Ending index of the selection (exclusive). If not provided, the selection ends at the index of the last element.

Example

```
let numbers = [2, 3, 5, 7, 11, 13, 17];
let newArray = numbers.slice(3, 6);
console.log(newArray);
```

Output

```
[ 7, 11, 13 ]
```

➢ splice()

The splice() method modifies an array (adds, removes or replaces elements).

splice() Syntax

The syntax of the splice() method is:

```
arr.splice(start, deleteCount, item1, ..., itemN)
```

Here, arr is an array.

splice() Parameters

The splice() method takes in:

- start - The index from where the array is changed.
- deleteCount (optional) - The number of items to remove from start.

- item1, ..., itemN (optional) - The elements to add to the start index. If not specified, splice() will only remove elements from the array.

Example

```
let prime_numbers = [2, 3, 5, 7, 9, 11];
let removedElement = prime_numbers.splice(4, 1, 13);
console.log(removedElement);
console.log(prime_numbers);
```

Output

```
[ 9 ]
[ 2, 3, 5, 7, 13, 11 ]
```

---

2. Date Object

- In JavaScript, the **Date** object is used to work with dates and times.
- It allows you to create, manipulate, and format dates and times according to various formats and time zones.
- Here are some details about the **Date** object along with examples:

2.1 Creating a Date Object:

You can create a **Date** object using the **new** keyword followed by the **Date()** constructor.

```
var currentDate = new Date();
```

2.2 Date Constructor with Parameters:

You can also create a **Date** object by passing parameters representing year, month, day, hour, minute, second, and millisecond.

```
var customDate = new Date(2024, 2, 8, 12, 30, 0, 0); // March 8, 2024, 12:30:00
```

2.3 Getting Current Date and Time:

The **Date** object without any parameters represents the current date and time.

```
var currentDate = new Date();
console.log(currentDate); // Output: current date and time
```

2.4 Getting Specific Components of a Date:

You can get individual components of a date such as year, month, day, hour, minute, second, and millisecond.

```
var year = currentDate.getFullYear();
var month = currentDate.getMonth(); // Note: Months are zero-based (0 for January, 1 for February, etc.)
var day = currentDate.getDate();
var hours = currentDate.getHours();
var minutes = currentDate.getMinutes();
```

2.5 Formatting Date and Time:

You can format dates and times using various methods such as **toDateString()**, **toLocaleDateString()**, **toTimeString()**, **toLocaleTimeString()**, etc.

```
console.log(currentDate.toDateString()); // Output: "Tue Mar 08 2024"
console.log(currentDate.toLocaleDateString()); // Output: "3/8/2024"
```

2.6 Working with Timezones:

JavaScript's **Date** object works with dates and times in the local timezone of the user's browser. However, you can also work with dates and times in different timezones using libraries like **moment.js** or by manually adjusting the time.

```
var utcDate = currentDate.toUTCString(); // Convert to UTC timezone
```

2.7 Manipulating Dates:

The **Date** object provides methods for manipulating dates, such as **setFullYear()**, **setMonth()**, **setDate()**, **setHours()**, etc.

```
currentDate.setDate(currentDate.getDate() + 7); // Add 7 days to the current date
```

2.8 Calculating Date Differences:

You can calculate the difference between two dates using subtraction and various methods like **getTime()** to get milliseconds since January 1, 1970.

```
var difference = date2.getTime() - date1.getTime(); // Difference in milliseconds
```

The **Date** object in JavaScript provides a rich set of functionalities for working with dates and times, allowing developers to handle various date-related operations in their applications.

---

3. Math Object

- In JavaScript, the **Math** object provides a set of properties and methods for mathematical operations.
- It includes constants like **PI** and methods for functions such as rounding, logarithms, trigonometry, and more.
- Here are some details about the **Math** object along with examples:

3.1 Constants:

The **Math** object includes several mathematical constants, such as **Math.PI** for the value of π (pi).

```
console.log(Math.PI); // Output: 3.141592653589793
```

3.2 Basic Mathematical Operations:

JavaScript's **Math** object provides methods for basic mathematical operations like **Math.abs()**, **Math.round()**, **Math.floor()**, **Math.ceil()**, **Math.min()**, and **Math.max()**.

```
console.log(Math.abs(-5)); // Output: 5
console.log(Math.round(4.7)); // Output: 5
console.log(Math.floor(4.7)); // Output: 4
console.log(Math.ceil(4.2)); // Output: 5
console.log(Math.min(10, 5, 8)); // Output: 5
console.log(Math.max(10, 5, 8)); // Output: 10
```

3.3 Exponential and Logarithmic Functions:

The **Math** object provides methods for exponentiation and logarithmic functions, such as **Math.pow()**, **Math.exp()**, **Math.log()**, **Math.log10()**, and **Math.sqrt()**.

```
console.log(Math.pow(2, 3)); // Output: 8 (2 raised to the power of 3)
console.log(Math.exp(1)); // Output: 2.718281828459045 (e raised to the power of 1)
console.log(Math.log(10)); // Output: 2.302585092994046 (natural logarithm of 10)
console.log(Math.sqrt(25)); // Output: 5 (square root of 25)
```

3.4 Trigonometric Functions:

The **Math** object provides trigonometric functions like **Math.sin()**, **Math.cos()**, **Math.tan()**, **Math.asin()**, **Math.acos()**, and **Math.atan()**.

```
console.log(Math.sin(Math.PI / 2)); // Output: 1 (sine of π/2)
console.log(Math.cos(0)); // Output: 1 (cosine of 0)
console.log(Math.tan(Math.PI / 4)); // Output: 1 (tangent of π/4)
```

3.5 Random Number Generation:

You can generate random numbers using the **Math.random()** method, which returns a random floating-point number between 0 (inclusive) and 1 (exclusive).

```
var randomNum = Math.random();
console.log(randomNum); // Output: Random number between 0 and 1
```

3.6 Degrees to Radians Conversion:

The **Math** object provides a method **Math.radians()** to convert degrees to radians.

```
function degToRad(degrees) {
    return degrees * (Math.PI / 180);
}
console.log(degToRad(180)); // Output: 3.141592653589793 (π radians)
```

The **Math** object in JavaScript is a powerful tool for performing various mathematical operations and calculations in your JavaScript applications.

4. Form Object

- In JavaScript, the HTML **<form>** element represents a collection of form elements that can be submitted to a server for processing.
- When working with forms in JavaScript, you can access and manipulate form elements and their properties using the **form** object.
- Here are some details about the **form** object along with examples:

4.1 Accessing a Form:

You can access a form using its **id** attribute and the **document.getElementById()** method.

HTML

```
<form id="myForm">
    <!-- Form elements go here -->
</form>
```

JavaScript

```
var form = document.getElementById('myForm');
```

4.2 Accessing Form Elements:

Once you have a reference to the form, you can access its elements using their **name** or **id** attributes.

HTML

```
<input type="text" name="username" id="username">
```

JavaScript

```
var usernameInput = form.username;
```

4.3 Accessing Form Properties:

You can access various properties of the form, such as **action**, **method**, **encoding**, **target**, **elements**, etc.

```
console.log(form.action);
console.log(form.method);
```

4.4 Accessing Form Controls:

The **elements** property of the form object provides access to all form controls as an array-like object.

```
var formControls = form.elements;
```

4.5 Submitting a Form:

You can submit a form programmatically using the **submit()** method of the form object.

```
form.submit();
```

4.6 Resetting a Form:

The **reset()** method of the form object resets all form controls to their default values.

```
form.reset();
```

4.7 Validating Form Input:

You can validate form input using JavaScript by accessing form elements and checking their values.

```
var username = form.username.value;
if (username === '') {
    alert('Please enter a username.');
}
```

4.8 Preventing Form Submission:

You can prevent the default form submission behavior using the **preventDefault()** method of the event object.

```
form.addEventListener('submit', function(event) {
    event.preventDefault();
    // Custom form submission logic
});
```

4.9 Dynamic Form Creation:

You can dynamically create form elements and add them to the DOM using JavaScript.

```
var newInput = document.createElement('input');
newInput.type = 'text';
newInput.name = 'newField';
form.appendChild(newInput);
```

4.10 Form Events:

You can attach event listeners to form elements to handle events such as **submit**, **reset**, **change**, **input**, etc.

```
form.addEventListener('submit', function(event) {
    // Handle form submission
});
```

The **form** object in JavaScript provides a convenient interface for working with HTML forms and their elements. You can use it to access, manipulate, and validate form data in your web applications.