1. Arrays

An array in Javascript is a data structure used to store multiple values in a single variable. Arrays are ordered, indexed (starting from 0), and can hold values of any data type (numbers, strings, objects, or even other arrays).

Example of an Array:

```
let fruits = ["Apple", "Banana", "Mango"];
console.log(fruits[0]) ; // Output: "Apple"
```

2. Array Methods

Javascript provides several built-in methods to manipulate arrays. Below are some of the most commonly used methods:

a. map()

The map() method creates a new array by applying a function to each element of the original array.

```
let numbers = [1, 2, 3, 4];
let squaredNumbers = numbers.map(num => num * num);
console.log(squaredNumbers) ; // Output: [1, 4, 9, 16]
```

b. filter()

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

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

```
let evenNumbers = numbers.filter(num => num % 2 === 0);
console.log(evenNumbers) ; // Output: [2, 4]
```

c. reduce()

The reduce() method reduces the array to a single value by executing a reducer function on each element.

```
let numbers = [1, 2, 3, 4];
let sum = numbers.reduce((accumulator, currentValue) => accumulator +
currentValue, 0);
console.log(sum) ; // Output: 10
```

d. sort()

The sort() method sorts the elements of an array in place and returns the sorted array. By default, it sorts elements as strings.

```
let fruits = ["Banana", "Apple", "Mango"];
fruits.sort();
console.log(fruits) ; // Output: ["Apple", "Banana", "Mango"]
```

For Numeric

```
let numbers = [40, 100, 1, 5, 25];
numbers.sort((a, b) => a - b);
console.log(numbers) ; // Output: [1, 5, 25, 40, 100]
```

3. Set & Map Objects

a. Set

A Set is a collection of unique values. It does not allow duplicate values.

```
let myset = new Set();
myset.add(1);
myset.add(2);
myset.add(2); // Duplicate value, will not be added
console.log(myset); // Output: Set { 1, 2 }
```

b. Map

A Map is a collection of key-value pairs where keys can be of any type.

```
let myMap = new Map();
myMap.set("name", "John");
myMap.set("age", 30);
console.log(myMap.get("name")) ; // Output: "John"
```

c. WeakMap & Weakset

WeakMap: A WeakMap is a collection of key-value pairs where the keys must be objects, and the values can be of any type. The keys in a WeakMap are weakly referenced, meaning they can be garbage collected if there are no other references to them.

```
// Create a WeakMap
let weakMap = new WeakMap();

// Create objects to use as keys
```

```
let key1 = { id: 1 };
let key2 = { id: 2 };

// Add key-value pairs to the WeakMap
weakMap.set(key1, "Value for key1");
weakMap.set(key2, "Value for key2");

// Retrieve values using keys
console.log(weakMap.get(key1)); // Output: "Value for key1"
console.log(weakMap.get(key2)); // Output: "Value for key2"

// Check if a key exists in the WeakMap
console.log(weakMap.has(key1)); // Output: true

// Delete a key-value pair
weakMap.delete(key1);
console.log(weakMap.has(key1)); // Output: false

// Garbage collection example
key2 = null; // Remove the reference to key2
// After some time, key2 will be garbage collected, and its entry will be
removed from the WeakMap
```

WeakSet: A WeakSet is a collection of objects where each object can occur only once. Like WeakMap, the objects in a WeakSet are weakly referenced, meaning they can be garbage collected if there are no other references to them.

```
// Create a WeakSet
let weakSet = new WeakSet();

// Create objects to add to the WeakSet
let obj1 = { name: "Alice" };
let obj2 = { name: "Bob" };

// Add objects to the WeakSet
weakSet.add(obj1);
weakSet.add(obj2);

// Check if an object exists in the WeakSet
console.log(weakSet.has(obj1)); // Output: true
console.log(weakSet.has(obj2)); // Output: true

// Delete an object from the WeakSet
weakSet.delete(obj1);
console.log(weakSet.has(obj1)); // Output: false

// Garbage collection example
obj2 = null; // Remove the reference to obj2
// After some time, obj2 will be garbage collected, and it will be removed from the WeakSet
```

d. flatMap()

The flatMap() method first maps each element using a mapping function, then flattens the result into a new array.

```
let numbers = [1, 2, 3];
let result = numbers.flatMap(num => [num, num * 2]);
console.log(result); // Output: [1, 2, 2, 4, 3, 6]
```

4. Strings

A string is a sequence of characters used to represent text. Strings are immutable, meaning once created, they cannot be changed.

Example of a String:

```
let greeting = "Hello, World!";
console.log(greeting); // Output: "Hello, World!"
```

5. String Methods

a. split()

The split() method splits a string into an array of substrings based on a specified separator.

```
let sentence = "Hello World";
let words = sentence.split(" ");
console.log(words); // Output: ["Hello", "World"]
```

b. slice()

The slice() method extracts a section of a string and returns it as a new string.

```
let text = "Hello, World!";
```

```
let slicedText = text.slice(0, 5);
console.log(slicedText); // Output: "Hello"
```

c. substring()

The substring() method extracts characters between two indices.

```
let text = "Hello, World!";
let subText = text.substring(7, 12);
console.log(subText); // Output: "World"
```

d. indexOf()

The indexOf() method returns the index of the first occurrence of a specified value in a string.

```
let text = "Hello, World!";
let index = text.indexOf("World");
console.log(index); // Output: 7
```

6. Array Destructuring

Array destructuring allows you to unpack values from arrays into distinct variables.

Example:

```
let numbers = [1, 2, 3];
let [a, b, c] = numbers;
console.log(a); // Output: 1
console.log(b); // Output: 2
```

```
console.log( c ); // Output: 3
```

You can also skip elements or use the rest operator:

```
let numbers = [1, 2, 3, 4, 5];
let [a, , b, ...rest] = numbers;
console.log(a); // Output: 1
console.log(b); // Output: 3
console.log(rest); // Output: [4, 5]
```