Date:20-06-24

JavaScript:-

1. Objects

- The object class represents one of JavaScript's data types. It is used to store various keyed collections and more complex entities. Objects can be created using the Object() constructor.
- we have two types of values used in JavaScript- primitive and reference.
- **Literal form:** The literal form uses the construction of object literals that can be said as a collection of key-valued pairs enclosed within a pair of curly braces.
- **Syntax-**

```
let obj = {
    key1: value1,
    key2: value2,
    ...
};
```

- **Constructed form:** The constructed form uses either an object constructor function or the new keyword to create an empty object and then adds properties to the object one by one.
- **Object constructor function:** In this methodology, the user creates an explicit function to take required values as a parameters and assign them as the properties of the desired object.
- **Syntax-**

```
function obj(value1, value2, ...) {
    this.key1 = value1;
    this.key2 = value2;
    ...
}
```

- **New keyword**: This methodology uses the new keyword in front of any constructor method or any bult-in constructor method(such as Object, Date, String, etc) and creates a new instance of the following object by mounting it on memory.
- **Syntax**-

```
let obj = new Object();
obj.key1 = value1;
obj.key2 = value2;
...
```

- **Difference between using Object Literals and Constructed Form:**
  Both the constructed form and literal form result in creating exactly the same sort of objects i.e. the end result is the same for both methodologies. The only difference between the both is that object literals can take care of several key-value pairs at once and thus is more

convenient while on the other hand with construction-form objects, we must add the properties one-by-one in separate statements.

2. Objects Methods

Javascript Object Methods can be grouped into :

- General Methods
- Property Management Methods
- Object Protection Methods

General methods

- Syntax-

```
// Copies properties from a source object to a target object
Object.assign(target, source)

// Creates an object from an existing object
Object.create(object)

// Returns an array of the key/value pairs of an object
Object.entries(object)

// Creates an object from a list of keys/values
Object.fromEntries()

// Returns an array of the keys of an object
Object.keys(object)

// Returns an array of the property values of an object
Object.values(object)

// Groups object elements according to a function
Object.groupBy(object, callback)
```

Property Management Methods

- **Syntax-**

```
// Adding or changing an object property
Object.defineProperty(object, property, descriptor)

// Adding or changing object properties
Object.defineProperties(object, descriptors)

// Accessing a Property
Object.getOwnPropertyDescriptor(object, property)

// Accessing Properties
Object.getOwnPropertyDescriptors(object)

// Returns all properties as an array
Object.getOwnPropertyNames(object)
```

```
// Accessing the prototype
Object.getPrototypeOf(object)
```

Object Protection Methods

- **Syntax-**

```
        // Prevents re-assignment
const car = {type:"Fiat", model:"500", color:"white"};

// Prevents adding object properties
Object.preventExtensions(object)

// Returns true if properties can be added to an object
Object.isExtensible(object)

// Prevents adding and deleting object properties
Object.seal(object)

// Returns true if object is sealed
Object.isSealed(object)

// Prevents any changes to an object
Object.freeze(object)

// Returns true if object is frozen
Object.isFrozen(object)
```

3. **Arrays**
   1. An array can hold many values under a single name, and you can access the values by referring to an index number.
   2. Syntax-

```
const array_name = [item1, item2, ...];
```

   3. Spaces and line breaks are not important. A declaration can span multiple lines.
   4. Syntax-

```
const cars = [
 "Saab",
 "Volvo",
 "BMW"
];
```

   5. You can also create an array, and then provide the elements:
   6. Syntax-

```
const cars = [];
cars[0]= "Saab";
cars[1]= "Volvo";
cars[2]= "BMW";
```

   7. You access an array element by referring to the **index number**:
      Syntax-

```
const cars = ["Saab", "Volvo", "BMW"];
let car = cars[0];
```

   8. This statement changes the value of the first element.

Syntax-

cars[0] = "Opel";

9. The JavaScript method toString() converts an array to a string of (comma separated) array values.

   Syntax-

   const fruits = ["Banana", "Orange", "Apple", "Mango"];
   document.getElementById("demo").innerHTML = fruits.toString();

10. With JavaScript, the full array can be accessed by referring to the array name:

    Syntax-

    const cars = ["Saab", "Volvo", "BMW"];
    document.getElementById("demo").innerHTML = cars;

11. Arrays are a special type of objects. The typeof operator in JavaScript returns "object" for arrays.
12. Arrays use **numbers** to access its "elements"
13. Syntax-

    const person = ["John", "Doe", 46];

14. Objects use **names** to access its "members". In this example, person.firstName returns John:
15. Syntax-

    const person = {firstName:"John", lastName:"Doe", age:46};

16. JavaScript variables can be objects. Arrays are special kinds of objects.Because of this, you can have variables of different types in the same Array.

17. myArray[0] = Date.now;
    myArray[1] = myFunction;
    myArray[2] = myCars;

4. **Array Properties and Methods**

- The real strength of JavaScript arrays are the built-in array properties and methods:

  cars.length // Returns the number of elements
  cars.sort() // Sorts the array

- **Length Property**: The length property of an array returns the length of an array (the number of array elements).

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

- **Accessing The first array element**:

  const fruits = ["Banana", "Orange", "Apple", "Mango"];
  let fruit = fruits[0];

- **Acccessing the last array element:**

  const fruits = ["Banana", "Orange", "Apple", "Mango"];
  let fruit = fruits[fruits.length - 1];

- **Looping Array Elements**: One way to loop through an array, is using a for loop

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
let fLen = fruits.length;

let text = "<ul>";
for (let i = 0; i < fLen; i++) {
  text += "<li>" + fruits[i] + "</li>";
}
text += "</ul>";
```

- You can also use the Array.forEach() function:

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

let text = "<ul>";
fruits.forEach(myFunction);
text += "</ul>";

function myFunction(value) {
  text += "<li>" + value + "</li>";
}
```

- The easiest way to add a new element to an array is using the push() method:

```
const fruits = ["Banana", "Orange", "Apple"];
fruits.push("Lemon");  // Adds a new element (Lemon) to fruits
```

- New element can also be added to an array using the length property:

```
const fruits = ["Banana", "Orange", "Apple"];
fruits[fruits.length] = "Lemon";  // Adds "Lemon" to fruits
```

5. **Associative Arrays**: Many programming languages support arrays with named indexes.

Arrays with named indexes are called associative arrays (or hashes).

JavaScript does **not** support arrays with named indexes.

In JavaScript, **arrays** always use **numbered indexes**

Synatx-

```
const person = [];

person[0] = "John";
person[1] = "Doe";
person[2] = 46;
person.length;    // Will return 3
person[0];        // Will return "John"
```

6. **Nested Array and Objects:** Values in objects can be arrays, and values in arrays can be objects:

```
const myObj = {
  name: "John",
```

```
 age: 30,
 cars: [
  {name:"Ford", models:["Fiesta", "Focus", "Mustang"]},
  {name:"BMW", models:["320", "X3", "X5"]},
  {name:"Fiat", models:["500", "Panda"]}
 ]
}
```

7. **Basic Array Methods:**

- Array length
- Array toString()
- Array at()
- Array join()
- Array pop()
- Array push()
- Array shift()
- Array unshift()
- Array delete()
- Array concat()
- Array copyWithin()
- Array flat()
- Array splice()
- Array toSpliced()
- Array slice()

Some others methods are:-

- Search Methods
- Sort Methods
- Iteration Methods