SUNSTONE

JavaScript

LECTURE 11 NOTES

Arrays

- **What are Arrays?**

  - An array is a single variable that is used to store elements of different data types. JavaScript arrays are zero-indexed. The arrays are not associative in nature.
  - Arrays are used when we have a list of items. An array allows you to store several values with the same name and access them by using their index number.
  - It is a container-like object that can hold elements of different types, such as numbers, strings, objects, or even other arrays.
  - Here's an example of how to create an array in JavaScript:

```
// Creating an empty array let myArray = [];

// Creating an array with values let myArray = [1, 2, 3, 4, 5];

// Creating an array with different data types

let mixedArray = [1, 'two', true, { name: 'John' }, null];
```

  - You can access individual elements in an array using square brackets notation, specifying the index of the element you want to access. Note that array indices are zero-based, meaning the first element has an index of 0.

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

console.log(myArray[0]);  // Output: 1

console.log(myArray[2]);  // Output: 3

console.log(myArray[4]);  // Output: 5
```

  - You can modify the value of an element by assigning a new value to a specific index:

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

myArray[2] = 10;console.log(myArray);  // Output: [1, 2, 10, 4,
5]
```

  - Arrays in JavaScript have many built-in methods that allow you to perform various operations, such as adding or removing elements, sorting, searching, and more. Some commonly used methods include push(), pop(), splice(), concat(), slice(), sort(), and forEach(). These methods provide flexibility and utility when working with arrays in JavaScript.

- **Array Methods for Manipulation and Iteration**
  - JavaScript provides several built-in array methods for manipulation and iteration.
  - Here are some commonly used array methods:

1. **push:** Adds one or more elements to the end of an array and returns the new length of the array.

```
const fruits = ['apple', 'banana'];

fruits.push('orange');

console.log(fruits); // ['apple', 'banana', 'orange']
```

2. **pop:** Removes the last element from an array and returns that element.

```
const fruits = ['apple', 'banana', 'orange'];

const removedFruit = fruits.pop();

console.log(fruits); // ['apple', 'banana']

console.log(removedFruit); // 'orange'
```

3. **concat:** Combines two or more arrays and returns a new array.

```
const fruits = ['apple', 'banana'];

const vegetables = ['carrot', 'broccoli'];

const combined = fruits.concat(vegetables);

console.log(combined);
//['apple','banana','carrot',broccoli']
```

4. **slice:** Extracts a portion of an array and returns a new array without modifying the original array.

```
const fruits = ['apple', 'banana', 'orange', 'grape',
'kiwi'];

const slicedFruits = fruits.slice(1, 4);
```

```
console.log(slicedFruits); // ['banana', 'orange', 'grape']
```

5. **forEach:** Executes a provided function once for each array element.

```
const fruits = ['apple', 'banana', 'orange'];

fruits.forEach((fruit) => {

console.log(fruit);

});

// Output:

'apple'

'banana'

'orange'
```

6. **map:** Creates a new array with the results of calling a provided function on every element in the array.

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

const multiplied = numbers.map((num) => num * 2);

console.log(multiplied); // [2, 4, 6, 8, 10]
```

7. **filter:** Creates a new array with all elements that pass a test implemented by the provided function.

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

const evenNumbers = numbers.filter((num) => num % 2 === 0);

console.log(evenNumbers); // [2, 4]
```

8. **reduce:** Applies a function to reduce the array to a single value.

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

const sum = numbers.reduce((acc, num) => acc + num, 0);
```

```
        console.log(sum); // 15
```

- **Types of Arrays**

  JS arrays are divided into two types:

  1. **One-dimensional arrays**
  2. **Multi-dimensional arrays**

1. One-dimensional Array

- A 1D array, also known as a single-dimensional array, is a linear collection of elements stored in a sequence. In JavaScript, you can create a 1D array using array literals or by dynamically populating the array.
- Here's an example of creating and working with a 1D array in JavaScript:

```
// Creating a 1D array

array1D = [1, 2, 3, 4, 5];

// Accessing elements in a 1D array

console.log(array1D[0]);                 // Output: 1

console.log(array1D[2]);                 // Output: 3

// Modifying elements in a 1D array

array1D[3] = 10;

console.log(array1D); // Output: [1, 2, 3, 10, 5]

// Length of a 1D array

console.log(array1D.length);                  // Output: 5
```

- In the above example, we create a 1D array array1D with five elements. We can access individual elements of the array using square brackets notation ([]), where the index starts from 0. The example demonstrates accessing elements at index 0 and 2.
- We can also modify elements in the 1D array by assigning new values to specific indices. In the example, we modify the element at index 3 to be 10.

- The length property of the array gives the number of elements in the 1D array. In the example, array1D.length returns 5.
- You can perform various operations on 1D arrays, such as iterating over elements using loops (for, forEach), applying array methods (map, filter, reduce), and combining multiple arrays using methods like concat

2. Two-dimensional Array

- The most common multi-dimensional array is a two-dimensional array
- A 2D array, also known as a two-dimensional array, is an array of arrays. It represents a grid or a table-like structure with rows and columns. In JavaScript, you can create a 2D array using array literals or dynamically populate the array.
- Here's an example of creating and working with a 2D array in JavaScript:

```
// Creating a 2D array
const array2D = [
  [1, 2, 3],
  [4, 5, 6],
  [7, 8, 9]
];
// Accessing elements in a 2D array
console.log(array2D[0][1]);       // Output: 2
console.log(array2D[2][0]);       // Output: 7
// Modifying elements in a 2D array
array2D[1][2] = 10;
console.log(array2D); // Output:[[1, 2, 3],[4, 5, 10],[7, 8, 9]]
// Length of a 2D array
console.log(array2D.length);           //  Output:  3  (number  of
rows)

console.log(array2D[0].length);    // Output: 3 (number of columns
                                               in the first row)
```

- In the above example, we create a 2D array array2D with three rows and three columns. Each row is an array, and the rows are enclosed within another array.
- To access elements in a 2D array, you use square brackets notation twice: the first index represents the row, and the second index represents the column. In the example, array2D[0][1] returns the element at row 0 (first row) and column 1 (second column), which is 2.
- You can also modify elements in the 2D array by assigning new values to specific indices. In the example, we modify the element at row 1 and column 2 to be 10.
- To get the length of a 2D array, you can use the length property. The array2D.length returns the number of rows, and array2D[0].length returns the number of columns in the first row.
- You can perform various operations on 2D arrays, such as traversing the elements using nested loops, accessing and modifying specific elements, and applying array methods to manipulate the data in the array.

3. Multi-dimensional Array

- In JavaScript, you can create multidimensional arrays, which are arrays with more than two dimensions. While JavaScript natively supports arrays with two dimensions (2D arrays), for higher dimensions, you typically use nested arrays to simulate multidimensional behavior.
- Here's an example of creating and working with a multidimensional array in JavaScript:

```
// Creating a 3D array

const array3D = [

  [ [1, 2, 3],

    [4, 5, 6] ],

  [ [7, 8, 9],
    [10, 11, 12] ]

];

// Accessing elements in a multidimensional array

console.log(array3D[0][1][2]);          // Output: 6

console.log(array3D[1][0][1]);          // Output: 8

// Modifying elements in a multidimensional array

array3D[0][0][2] = 15;

console.log(array3D);

// Output: [[[1, 2, 15], [4, 5, 6]], [[7, 8, 9], [10, 11, 12]]]

// Length of a multidimensional array

console.log(array3D.length);            //Output:2(number  of  2D
arrays)

console.log(array3D[0].length);

// Output: 2 (number of rows in the first 2D array)

console.log(array3D[0][0].length);

// Output: 3 (number of elements in the first row of the first 2D
array)
```

- In the example above, we create a multidimensional array array3D using nested arrays. array3D is a 3D array that contains two 2D arrays, which in turn contain nested arrays representing rows and elements.
- To access elements in the multidimensional array, you use square brackets notation with multiple indices corresponding to each dimension. For example, array3D[0][1][2] returns the element at the first 2D array, second row, and third element, which is 6.
- You can modify elements in the multidimensional array by assigning new values to specific indices. In the example, we modify the element at the first 2D array, first row, and third element to be 15.
- To get the length of the multidimensional array, you can use the length property for each dimension. In the example, array3D.length returns the number of 2D arrays,

array3D[0].length returns the number of rows in the first 2D array, and array3D[0][0].length returns the number of elements in the first row of the first 2D array.

- You can extend this concept to create arrays with higher dimensions by adding more levels of nesting. However, keep in mind that working with highly nested arrays can become complex, and it may be more appropriate to use other data structures or abstractions depending on the specific problem you are solving.

- **Arrays are Objects**

  - Arrays are a special type of object. The **typeof** operator in JavaScript returns 'object' for arrays. But JavaScript arrays are best described as arrays.

  - Arrays use numbers to access their 'elements'. In this example, person[0] returns John:

  - **Array:**

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

    Objects use names to access its 'members'. In this example, person.firstName returns John:

  - **Object:**

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

  - **Array Elements can be Objects**

    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.

    You can have objects in an array. You can have functions in an array. You can have arrays in an array:

    ```
    myArray[0] = Date.now;

    myArray[1] = myFunction;
    ```

```
myArray[2] = myCars;
```

- **Note:**
    - In JavaScript, arrays use numbered indexes.
    - In JavaScript, objects use named indexes.
    - Arrays are a special kind of object, with numbered indexes.
    - JavaScript does not support associative arrays.
    - You should use objects when you want the element names to be strings (text).
    - You should use arrays when you want the element names to be numbers.