Arrays And Data Structure

Week	3

Adib Sakhawat
IUT CSE ‘21

Arrays are fundamental data structures in JavaScript used to store multiple values in a single variable. They are ordered collections that can hold elements of any data type, including numbers, strings, objects, and even other arrays.

Creating Arrays

Arrays can be created using two main methods:

1. Using Square Brackets `[]`

```
// Empty array
let emptyArray = [];

// Array with elements
let fruits = ['Apple', 'Banana', 'Cherry'];
```

2. Using the `Array` Constructor

```
// Empty array
let emptyArray = new Array();

// Array with elements
let numbers = new Array(1, 2, 3, 4, 5);
```

Note: It's generally recommended to use square brackets `[]` for simplicity and readability.

Accessing Elements

Array elements are accessed using zero-based indexing.

```
let fruits = ['Apple', 'Banana', 'Cherry'];

console.log(fruits[0]); // Outputs: Apple
console.log(fruits[1]); // Outputs: Banana
console.log(fruits[2]); // Outputs: Cherry
```

Accessing the Length of an Array

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

Accessing the Last Element

```
console.log(fruits[fruits.length - 1]); // Outputs: Cherry
```

Modifying Elements

You can modify elements directly by accessing them via their index.

```
let fruits = ['Apple', 'Banana', 'Cherry'];

fruits[1] = 'Blueberry';
console.log(fruits); // Outputs: ['Apple', 'Blueberry', 'Ch
erry']
```

Adding elements beyond the current length:

```
fruits[3] = 'Date';
console.log(fruits); // Outputs: ['Apple', 'Blueberry', 'Ch
erry', 'Date']
```

Common Array Methods

push()

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

Syntax:

```
array.push(element1, ..., elementN);
```

Example:

```
let numbers = [1, 2, 3];
numbers.push(4, 5);
console.log(numbers); // Outputs: [1, 2, 3, 4, 5]
```

---

pop()

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

Syntax:

```
let removedElement = array.pop();
```

Example:

```
let numbers = [1, 2, 3];
let lastNumber = numbers.pop();
console.log(numbers);    // Outputs: [1, 2]
console.log(lastNumber); // Outputs: 3
```

---

shift()

Description: Removes the **first** element from an array and returns that element.

Syntax:

```
let removedElement = array.shift();
```

Example:

```
let numbers = [1, 2, 3];
let firstNumber = numbers.shift();
```

```
console.log(numbers);      // Outputs: [2, 3]
console.log(firstNumber); // Outputs: 1
```

unshift()

Description: Adds one or more elements to the **beginning** of an array and returns the new length of the array.

Syntax:

```
array.unshift(element1, ..., elementN);
```

Example:

```
let numbers = [3, 4, 5];
numbers.unshift(1, 2);
console.log(numbers); // Outputs: [1, 2, 3, 4, 5]
```

splice()

Description: Adds and/or removes elements from an array.

Syntax:

```
array.splice(start, deleteCount, item1, ..., itemN);
```

- **start**: The index at which to start changing the array.
- **deleteCount**: The number of elements to remove.
- **item1, ..., itemN**: Elements to add to the array.

Examples:

1. **Removing elements:**

   ```
   let fruits = ['Apple', 'Banana', 'Cherry', 'Date'];
   fruits.splice(1, 2);
   console.log(fruits); // Outputs: ['Apple', 'Date']
   ```

2. **Adding elements:**

```
let fruits = ['Apple', 'Date'];
fruits.splice(1, 0, 'Banana', 'Cherry');
console.log(fruits); // Outputs: ['Apple', 'Banana', 'Ch
erry', 'Date']
```

3. **Replacing elements:**

```
let fruits = ['Apple', 'Banana', 'Cherry'];
fruits.splice(1, 1, 'Blueberry');
console.log(fruits); // Outputs: ['Apple', 'Blueberry',
'Cherry']
```

---

forEach()

Description: Executes a provided function once for each array element.

Syntax:

```
array.forEach(function(element, index, array) {
    // Code to execute for each element
});
```

- **element**: The current element being processed.
- **index**: The index of the current element.
- **array**: The array `forEach` was called upon.

Example:

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

numbers.forEach(function(number, index) {
    console.log(`Index ${index}: ${number}`);
});
```

Output:

```
Index 0: 1
Index 1: 2
Index 2: 3
Index 3: 4
Index 4: 5
```

Using Arrow Functions:

```
numbers.forEach((number, index) => {
    console.log(`Index ${index}: ${number}`);
});
```

Arrays as Other Data Structures

Arrays can be used to implement other data structures like **stacks** and **queues** due to their flexible nature.

Stacks

A **stack** is a Last-In-First-Out (LIFO) data structure where the last element added is the first one removed.

Implementation using Arrays:

- **push()**: To add an element to the top of the stack.
- **pop()**: To remove the top element from the stack.

Example:

```
let stack = [];

// Push elements onto the stack
stack.push(1);
stack.push(2);
stack.push(3);

console.log(stack); // Outputs: [1, 2, 3]

// Pop elements from the stack
let topElement = stack.pop();
```

```
console.log(topElement); // Outputs: 3
console.log(stack);      // Outputs: [1, 2]
```

Queues

A **queue** is a First-In-First-Out (FIFO) data structure where the first element added is the first one removed.

Implementation using Arrays:

- **push()**: To add an element to the end of the queue.
- **shift()**: To remove the first element from the queue.

Example:

```
let queue = [];

// Enqueue elements
queue.push('Task 1');
queue.push('Task 2');
queue.push('Task 3');

console.log(queue); // Outputs: ['Task 1', 'Task 2', 'Task
3']

// Dequeue elements
let firstTask = queue.shift();
console.log(firstTask); // Outputs: 'Task 1'
console.log(queue);     // Outputs: ['Task 2', 'Task 3']
```

Note: Alternatively, you can use **unshift()** and **pop()** for queue operations, depending on which end you want to add or remove elements.

Summary

- **Creating Arrays:**
  - Use square brackets `[]` or the `Array` constructor.
  - Arrays can hold elements of any data type.
- **Accessing and Modifying Elements:**

    - Use zero-based indexing to access elements.
    - Modify elements by assigning a new value to a specific index.
- **Common Array Methods:**
    - **push()**: Add elements to the end.
    - **pop()**: Remove the last element.
    - **shift()**: Remove the first element.
    - **unshift()**: Add elements to the beginning.
    - **splice()**: Add, remove, or replace elements at a specific index.
    - **forEach()**: Execute a function for each element.
- **Arrays as Other Data Structures:**
    - **Stacks (LIFO):** Use `push()` and `pop()`.
    - **Queues (FIFO):** Use `push()` and `shift()`.

Practice Exercises:

1. **Reverse an Array:**

   Write a function that reverses an array without using the built-in `reverse()` method.

   ```
   function reverseArray(arr) {
       let reversed = [];
       for (let i = arr.length - 1; i >= 0; i--) {
           reversed.push(arr[i]);
       }
       return reversed;
   }

   let originalArray = [1, 2, 3, 4, 5];
   console.log(reverseArray(originalArray)); // Outputs:
   [5, 4, 3, 2, 1]
   ```

2. **Filter Even Numbers:**

   Use `forEach()` to create a new array that contains only the even numbers from the original array.

```
let numbers = [1, 2, 3, 4, 5, 6];
let evenNumbers = [];

numbers.forEach(number => {
    if (number % 2 === 0) {
        evenNumbers.push(number);
    }
});

console.log(evenNumbers); // Outputs: [2, 4, 6]
```

Further Reading:

- **Array Methods:**
  - Explore other array methods like `map()`, `filter()`, `reduce()`, `find()`, `includes()`, and `slice()` for more advanced operations.
- **Typed Arrays:**
  - For handling binary data, look into typed arrays like `Uint8Array`, `Float32Array`, etc.
- **Multidimensional Arrays:**
  - Arrays can contain other arrays, allowing you to create multidimensional arrays (matrices).

```
let matrix = [
    [1, 2, 3],
    [4, 5, 6],
    [7, 8, 9]
];

console.log(matrix[0][1]); // Outputs: 2
```