JavaScript Arrays, Objects, Map, and Set

Training Material (Part 1 of 9)

---

1. ARRAYS

What are Arrays?

Arrays are ordered collections of values that can store multiple items in a single variable. Each item has an index (position) starting from 0.

Why use Arrays?

- Store multiple related values together
- Organize data in a sequential manner
- Easy to iterate through collections
- Built-in methods for manipulation

How to use Arrays?

Creating Arrays

```
javascript

// Method 1: Array literal
const fruits = ['apple', 'banana', 'orange'];

// Method 2: Array constructor
const numbers = new Array(1, 2, 3, 4, 5);

// Method 3: Empty array
const empty = [];
```

Accessing Array Elements

```
javascript
```

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

console.log(fruits[0]);  // 'apple'
console.log(fruits[1]);  // 'banana'
console.log(fruits.length);  // 3
```

Common Array Operations

```javascript
const colors = ['red', 'blue'];

// Adding elements
colors.push('green');       // Add to end: ['red', 'blue', 'green']
colors.unshift('yellow');   // Add to start: ['yellow', 'red', 'blue', 'green']

// Removing elements
colors.pop();               // Remove from end: ['yellow', 'red', 'blue']
colors.shift();             // Remove from start: ['red', 'blue']

// Finding elements
const index = colors.indexOf('blue');  // 1
const exists = colors.includes('red'); // true
```

Visual Diagram: Array Structure

```
Array: fruits = ['apple', 'banana', 'orange']

Index:    0         1          2
       ┌─────────┬──────────┬──────────┐
Value: │ 'apple' │ 'banana' │ 'orange' │
       └─────────┴──────────┴──────────┘
```

---

2. OBJECTS

What are Objects?

Objects are collections of key-value pairs that store related data and functionality together. Keys are strings (or Symbols), and values can be any data type.

Why use Objects?

- Group related data and functions
- Represent real-world entities
- More descriptive than arrays (named properties)
- Fundamental to JavaScript programming

How to use Objects?

Creating Objects

```javascript
// Method 1: Object literal (most common)
const person = {
  name: 'John',
  age: 30,
  city: 'New York'
};

// Method 2: Object constructor
const car = new Object();
car.brand = 'Toyota';
car.model = 'Camry';

// Method 3: Using Object.create()
const employee = Object.create(person);
employee.jobTitle = 'Developer';
```

Accessing Object Properties

```javascript
```

```
const person = {
  name: 'John',
  age: 30,
  city: 'New York'
};

// Dot notation
console.log(person.name);   // 'John'

// Bracket notation
console.log(person['age']); // 30

// Adding new properties
person.email = 'john@example.com';

// Deleting properties
delete person.city;
```

Object Methods

```javascript
const calculator = {
  value: 0,
  add: function(num) {
    this.value += num;
    return this;
  },
  subtract: function(num) {
    this.value -= num;
    return this;
  },
  getValue: function() {
    return this.value;
  }
};

calculator.add(10).subtract(3).getValue(); // 7
```

Visual Diagram: Object Structure

```
Object: person = { name: 'John', age: 30, city: 'New York' }

┌─────────────────────────────────┐
│       person Object         │
├──────────────┬──────────────────┤
│ Key       │ Value        │
├──────────────┼──────────────────┤
│ name       │ 'John'        │
│ age       │ 30          │
│ city       │ 'New York'       │
└──────────────┴──────────────────┘
```

---

3. MAP

What is Map?

Map is a collection of key-value pairs where keys can be of any data type (not just strings). It maintains insertion order and provides better performance for frequent additions/deletions.

Why use Map?

- Keys can be any data type (objects, functions, primitives)
- Maintains insertion order
- Better performance for large datasets
- Built-in size property
- Easy iteration

How to use Map?

Creating and Using Maps

```javascript
```

```
// Creating a Map
const userRoles = new Map();

// Setting values
userRoles.set('john@example.com', 'admin');
userRoles.set('jane@example.com', 'editor');
userRoles.set('bob@example.com', 'viewer');

// Getting values
console.log(userRoles.get('john@example.com')); // 'admin'

// Checking existence
console.log(userRoles.has('jane@example.com')); // true

// Size
console.log(userRoles.size); // 3

// Deleting
userRoles.delete('bob@example.com');

// Clearing all
userRoles.clear();
```

Map with Different Key Types

```javascript
const mixedMap = new Map();

// Object as key
const objKey = { id: 1 };
mixedMap.set(objKey, 'Object value');

// Function as key
const funcKey = function() {};
mixedMap.set(funcKey, 'Function value');

// Number as key
mixedMap.set(42, 'Number value');

console.log(mixedMap.get(objKey));  // 'Object value'
console.log(mixedMap.get(42));      // 'Number value'
```

Iterating over Maps

javascript

```javascript
const fruits = new Map([
  ['apple', 5],
  ['banana', 3],
  ['orange', 7]
]);

// forEach
fruits.forEach((value, key) => {
  console.log(`${key}: ${value}`);
});

// for...of with entries
for (const [key, value] of fruits.entries()) {
  console.log(`${key}: ${value}`);
}

// Getting keys
for (const key of fruits.keys()) {
  console.log(key);
}

// Getting values
for (const value of fruits.values()) {
  console.log(value);
}
```

Visual Diagram: Map Structure

```
Map: userRoles

┌─────────────────────────────────────┐
│         Map Structure           │
├───────────────────┬─────────────────┤
│ Key (any type)    │ Value         │
├───────────────────┼─────────────────┤
│ 'john@example.com'│ 'admin'       │
│ 'jane@example.com'│ 'editor'      │
│ 'bob@example.com' │ 'viewer'      │
└───────────────────┴─────────────────┘

Size: 3
Maintains insertion order: Yes
```

---

4. SET

What is Set?

Set is a collection of unique values. Each value can only occur once in a Set. Values can be of any data type.

Why use Set?

- Automatically removes duplicates
- Fast lookup for checking existence
- Useful for mathematical operations (union, intersection)
- Maintains insertion order

How to use Set?

Creating and Using Sets

```javascript
```

```
// Creating a Set
const uniqueNumbers = new Set();

// Adding values
uniqueNumbers.add(1);
uniqueNumbers.add(2);
uniqueNumbers.add(3);
uniqueNumbers.add(2); // Duplicate, won't be added

console.log(uniqueNumbers.size); // 3

// Checking existence
console.log(uniqueNumbers.has(2)); // true

// Deleting
uniqueNumbers.delete(1);

// Clearing all
uniqueNumbers.clear();
```

Creating Set from Array

```javascript
// Remove duplicates from array
const numbers = [1, 2, 3, 2, 4, 1, 5];
const uniqueNumbers = new Set(numbers);

console.log(uniqueNumbers); // Set { 1, 2, 3, 4, 5 }

// Convert back to array
const uniqueArray = [...uniqueNumbers];
console.log(uniqueArray); // [1, 2, 3, 4, 5]
```

Iterating over Sets

```javascript
```

```
const colors = new Set(['red', 'blue', 'green']);

// forEach
colors.forEach(color => {
  console.log(color);
});

// for...of
for (const color of colors) {
  console.log(color);
}
```

Practical Set Operations

```javascript
// Union (combining two sets)
const setA = new Set([1, 2, 3]);
const setB = new Set([3, 4, 5]);
const union = new Set([...setA, ...setB]);
console.log(union); // Set { 1, 2, 3, 4, 5 }

// Intersection (common elements)
const intersection = new Set(
  [...setA].filter(x => setB.has(x))
);
console.log(intersection); // Set { 3 }

// Difference (elements in A but not in B)
const difference = new Set(
  [...setA].filter(x => !setB.has(x))
);
console.log(difference); // Set { 1, 2 }
```

Visual Diagram: Set Structure

```
Set: uniqueNumbers = Set { 1, 2, 3, 4, 5 }

┌─────────────────────────┐
│   Set Structure    │
│ (Only Unique Values)   │
├─────────────────────────┤
│ 1          │
│ 2          │
│ 3          │
│ 4          │
│ 5          │
└─────────────────────────┘

Size: 5
Duplicates: Automatically removed
Order: Insertion order maintained
```

COMPARISON TABLE

Feature	**Array**	**Object**	**Map**	**Set**
Key Type	Numeric index	String/Symbol	Any type	N/A (values only)
Ordered	Yes	No guarantee	Yes	Yes
Duplicates	Allowed	Keys unique	Keys unique	Values unique
Size Property	.length	Manual count	.size	.size
Iteration	Easy	Requires methods	Easy	Easy
Use Case	Ordered lists	Entity representation	Key-value with any keys	Unique values

---

COMPLETE EXAMPLE: User Management System

```
html
```

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>User Management System</title>
  <style>
    body {
      font-family: Arial, sans-serif;
      max-width: 800px;
      margin: 50px auto;
      padding: 20px;
      background: #f5f5f5;
    }
    .container {
      background: white;
      padding: 30px;
      border-radius: 10px;
      box-shadow: 0 2px 10px rgba(0,0,0,0.1);
    }
    h1 {
      color: #333;
      text-align: center;
    }
    .output {
      background: #f9f9f9;
      padding: 15px;
      border-radius: 5px;
      margin-top: 20px;
      border-left: 4px solid #4CAF50;
    }
    .output h3 {
      margin-top: 0;
      color: #4CAF50;
    }
    button {
      background: #4CAF50;
      color: white;
      padding: 10px 20px;
      border: none;
      border-radius: 5px;
      cursor: pointer;
      margin: 5px;
    }
    button:hover {
      background: #45a049;
    }
  </style>
</head>
<body>
  <div class="container">
    <h1>User Management System</h1>
    <p>Demonstrating Arrays, Objects, Map, and Set</p>

    <button onclick="demonstrateAll()">Run Demonstration</button>

    <div class="output" id="output">
      <h3>Output:</h3>
      <pre id="results"></pre>
    </div>
  </div>

  <script>
    function demonstrateAll() {
      let output = '';

      // 1. ARRAY - Store user IDs
      output += '=== ARRAYS ===\n';
      const userIds = [101, 102, 103, 104];
      output += `User IDs: ${userIds}\n`;
      userIds.push(105);
      output += `After adding user: ${userIds}\n`;
      output += `Total users: ${userIds.length}\n\n`;

      // 2. OBJECT - Store user details
      output += '=== OBJECTS ===\n';
      const user = {
        id: 101,
        name: 'Alice Johnson',
        email: 'alice@example.com',
        role: 'admin',
        active: true
      };
      output += `User: ${user.name}\n`;
      output += `Email: ${user.email}\n`;
      output += `Role: ${user.role}\n\n`;
```

```
// 3. MAP - Store user sessions (user ID -> session data)
output += '=== MAP ===\n';
const sessions = new Map();
sessions.set(101, { loginTime: '2024-01-15 10:30', ip: '192.168.1.1' });
sessions.set(102, { loginTime: '2024-01-15 11:45', ip: '192.168.1.2' });
sessions.set(103, { loginTime: '2024-01-15 12:15', ip: '192.168.1.3' });

output += `Active sessions: ${sessions.size}\n`;
const session101 = sessions.get(101);
output += `User 101 logged in at: ${session101.loginTime}\n\n`;

// 4. SET - Store unique tags/skills
output += '=== SET ===\n';
const userSkills = new Set();
userSkills.add('JavaScript');
userSkills.add('React');
userSkills.add('Node.js');
userSkills.add('JavaScript'); // Duplicate, won't be added

output += `Unique skills: ${[...userSkills].join(', ')}\n`;
output += `Total unique skills: ${userSkills.size}\n\n`;

// 5. COMBINING ALL - Complete user database
output += '=== COMPLETE EXAMPLE ===\n';
const database = {
  users: [
    { id: 101, name: 'Alice', tags: ['admin', 'developer'] },
    { id: 102, name: 'Bob', tags: ['developer'] },
    { id: 103, name: 'Charlie', tags: ['designer', 'developer'] }
  ],
  sessions: new Map(),
  uniqueTags: new Set()
};

// Collect all unique tags
database.users.forEach(user => {
  user.tags.forEach(tag => database.uniqueTags.add(tag));
});

output += `Total users: ${database.users.length}\n`;
output += `All tags: ${[...database.uniqueTags].join(', ')}\n`;
output += `Developers: ${database.users.filter(u => u.tags.includes('developer')).length}\n`;

document.getElementById('results').textContent = output;
```

```
    }
  </script>
</body>
</html>
```

PRACTICE EXERCISES

Exercise 1: Array Manipulation

Create an array of student names. Add a new student, remove the first student, and check if a specific student exists.

Exercise 2: Object Creation

Create an object representing a book with properties: title, author, year, and a method to display book info.

Exercise 3: Map Operations

Create a Map to store product names as keys and their prices as values. Add, retrieve, and delete products.

Exercise 4: Set for Uniqueness

Given an array with duplicate values, use a Set to remove duplicates and return a new array with unique values.

KEY TAKEAWAYS

1. **Arrays**: Use for ordered collections with numeric indices
2. **Objects**: Use for structured data with named properties
3. **Map**: Use when you need non-string keys or frequent additions/deletions
4. **Set**: Use when you need to store unique values only

Next Topic: String and Array Methods