JavaScript Data Structures

This guide covers essential JavaScript data structures that help you organize and manipulate data effectively.

Table of Contents

Strings

Strings are sequences of characters used to represent text.

Creating Strings

```
let greeting = "Hello World";
let name = 'Alice';
let message = `Welcome, ${name}!`; // Template literal
```

Common String Properties and Methods

Length Property

```
let text = "JavaScript";
console.log(text.length); // 10
```

String Methods

```
let sentence = "Learning JavaScript is Fun";

// Case conversion
console.log(sentence.toLowerCase()); // "learning javascript is fun"
console.log(sentence.toUpperCase()); // "LEARNING JAVASCRIPT IS FUN"

// Searching
console.log(sentence.indexOf("Script"));     // 10
console.log(sentence.includes("JavaScript")); // true
console.log(sentence.startsWith("Learning")); // true
console.log(sentence.endsWith("Fun"));       // true
```

```
// Extracting parts
console.log(sentence.slice(0, 8));          // "Learning"
console.log(sentence.substring(9, 19));      // "JavaScript"

// Splitting and joining
let words = sentence.split(" ");             // ["Learning", "JavaScript", "is",
"Fun"]
let rejoined = words.join("-");              // "Learning-JavaScript-is-Fun"

// Replacing
let newSentence = sentence.replace("Fun", "Awesome");
console.log(newSentence); // "Learning JavaScript is Awesome"

// Trimming whitespace
let messy = "  trim me  ";
console.log(messy.trim()); // "trim me"
```

Template Literals (Template Strings)

```
let firstName = "John";
let lastName = "Doe";
let age = 30;

// Multi-line strings and variable interpolation
let profile = `
Name: ${firstName} ${lastName}
Age: ${age}
Status: ${age >= 18 ? 'Adult' : 'Minor'}
`;
console.log(profile);
```

Arrays

Arrays are ordered lists of values, perfect for storing multiple related items.

Creating Arrays

```
let fruits = ["apple", "banana", "orange"];
let numbers = [1, 2, 3, 4, 5];
let mixed = ["hello", 42, true, null];
let empty = [];
```

Array Properties and Methods

Length and Access

```
let colors = ["red", "green", "blue"];
console.log(colors.length);  // 3
console.log(colors[0]);      // "red" (first element)
console.log(colors[2]);      // "blue" (third element)
colors[1] = "yellow";        // Change "green" to "yellow"
```

Adding and Removing Elements

```
let animals = ["cat", "dog"];

// Adding elements
animals.push("bird");           // Add to end: ["cat", "dog", "bird"]
animals.unshift("fish");        // Add to beginning: ["fish", "cat", "dog",
"bird"]

// Removing elements
let lastAnimal = animals.pop();     // Remove from end: "bird"
let firstAnimal = animals.shift();  // Remove from beginning: "fish"
console.log(animals); // ["cat", "dog"]

// Add/remove at specific position
animals.splice(1, 0, "rabbit");    // Add "rabbit" at index 1
animals.splice(2, 1);              // Remove 1 element at index 2
```

Array Iteration Methods

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

// forEach - execute function for each element
numbers.forEach((num, index) => {
    console.log(`Index ${index}: ${num}`);
});

// map - create new array with transformed elements
let doubled = numbers.map(num => num * 2);
console.log(doubled); // [2, 4, 6, 8, 10]

// filter - create new array with elements that pass test
let evens = numbers.filter(num => num % 2 === 0);
console.log(evens); // [2, 4]

// find - find first element that passes test
let found = numbers.find(num => num > 3);
console.log(found); // 4

// reduce - reduce array to single value
let sum = numbers.reduce((total, num) => total + num, 0);
console.log(sum); // 15
```

Other Useful Array Methods

```
let arr1 = [1, 2, 3];
let arr2 = [4, 5, 6];

// Combining arrays
let combined = arr1.concat(arr2);     // [1, 2, 3, 4, 5, 6]
let spread = [...arr1, ...arr2];      // Same result using spread operator

// Searching
console.log(arr1.includes(2));       // true
console.log(arr1.indexOf(3));        // 2

// Sorting
let fruits = ["banana", "apple", "cherry"];
fruits.sort();                       // ["apple", "banana", "cherry"]

let nums = [3, 1, 4, 1, 5];
nums.sort((a, b) => a - b);          // [1, 1, 3, 4, 5] (ascending)
```

Objects

Objects are collections of key-value pairs, perfect for representing real-world entities.

Creating Objects

```
// Object literal syntax
let person = {
    firstName: "John",
    lastName: "Doe",
    age: 30,
    isEmployed: true
};

// Constructor syntax
let car = new Object();
car.make = "Toyota";
car.model = "Camry";
car.year = 2023;
```

Accessing and Modifying Object Properties

```
let student = {
    name: "Alice",
    grade: "A",
    subjects: ["Math", "Science", "English"]
};

// Dot notation
console.log(student.name);          // "Alice"
student.age = 20;                   // Add new property

// Bracket notation (useful for dynamic property names)
console.log(student["grade"]);      // "A"
let prop = "subjects";
console.log(student[prop]);         // ["Math", "Science", "English"]

// Delete properties
delete student.grade;
```

Object Methods

```
let calculator = {
    result: 0,
    add: function(num) {
        this.result += num;
        return this;
    },
    multiply: function(num) {
        this.result *= num;
        return this;
    },
    getResult: function() {
        return this.result;
    }
};

// Method chaining
calculator.add(5).multiply(3);
console.log(calculator.getResult()); // 15
```

Working with Objects

```
let user = {
    username: "alice123",
    email: "alice@email.com",
    preferences: {
        theme: "dark",
        language: "en"
    }
};

// Object methods
console.log(Object.keys(user));       // ["username", "email", "preferences"]
console.log(Object.values(user));    // ["alice123", "alice@email.com", {...}]
console.log(Object.entries(user));   // [["username", "alice123"], ...]

// Check if property exists
console.log("email" in user);        // true
console.log(user.hasOwnProperty("username")); // true

// Nested property access
console.log(user.preferences.theme); // "dark"
```

Dates

The Date object represents dates and times in JavaScript.

Creating Dates

```
let now = new Date();                        // Current date and time
let specific = new Date("2023-12-25");       // Christmas 2023
let withTime = new Date("2023-12-25 10:30:00");
let fromNumbers = new Date(2023, 11, 25, 10, 30, 0); // Month is 0-indexed!
```

Date Methods

```
let birthday = new Date("1995-06-15");

// Getting date components
console.log(birthday.getFullYear());    // 1995
console.log(birthday.getMonth());       // 5 (June - 0-indexed)
console.log(birthday.getDate());        // 15
console.log(birthday.getDay());         // Day of week (0=Sunday, 1=Monday, etc.)

// Getting time components
console.log(birthday.getHours());       // Hours (0-23)
console.log(birthday.getMinutes());     // Minutes (0-59)
console.log(birthday.getSeconds());     // Seconds (0-59)

// Setting date components
birthday.setFullYear(1996);
birthday.setMonth(5); // June
birthday.setDate(20);
```

Date Formatting and Calculations

```
let today = new Date();

// Formatting
```

```
console.log(today.toString());          // Full date string
console.log(today.toDateString());      // Date portion only
console.log(today.toTimeString());      // Time portion only
console.log(today.toISOString());       // ISO format

// Calculations
let futureDate = new Date();
futureDate.setDate(today.getDate() + 30); // 30 days from now

// Time difference
let timeDiff = futureDate - today;      // Difference in milliseconds
let daysDiff = timeDiff / (1000 * 60 * 60 * 24); // Convert to days
```

Sets

Sets are collections of unique values - no duplicates allowed.

Creating and Using Sets

```
let numbers = new Set([1, 2, 3, 3, 4, 4, 5]); // Duplicates removed
console.log(numbers); // Set {1, 2, 3, 4, 5}

let colors = new Set();
colors.add("red");
colors.add("blue");
colors.add("red");    // Won't be added again
console.log(colors.size); // 2
```

Set Methods

```
let fruits = new Set(["apple", "banana", "orange"]);

// Adding and removing
fruits.add("grape");
fruits.delete("banana");

// Checking existence
console.log(fruits.has("apple"));    // true
console.log(fruits.has("banana"));   // false

// Iteration
fruits.forEach(fruit => console.log(fruit));

// Convert to array
let fruitsArray = Array.from(fruits);
// or
let fruitsArray2 = [...fruits];
```

```
// Clear all elements
fruits.clear();
```

Practical Set Examples

```
// Remove duplicates from array
let numbersWithDuplicates = [1, 2, 2, 3, 3, 3, 4];
let uniqueNumbers = [...new Set(numbersWithDuplicates)];
console.log(uniqueNumbers); // [1, 2, 3, 4]

// Find unique characters in string
let text = "hello world";
let uniqueChars = new Set(text.split(""));
console.log(uniqueChars); // Set with unique characters
```

Maps

Maps are collections of key-value pairs where keys can be any data type.

Creating and Using Maps

```
let userRoles = new Map();
userRoles.set("alice", "admin");
userRoles.set("bob", "user");
userRoles.set("charlie", "moderator");

// Or create with initial data
let productPrices = new Map([
    ["apple", 1.50],
    ["banana", 0.75],
    ["orange", 2.00]
]);
```

Map Methods

```
let inventory = new Map();

// Adding and updating
inventory.set("laptops", 15);
inventory.set("phones", 32);
inventory.set("tablets", 8);

// Getting values
console.log(inventory.get("laptops"));     // 15
console.log(inventory.get("cameras"));     // undefined
```

```
// Checking existence
console.log(inventory.has("phones"));      // true

// Size and deletion
console.log(inventory.size);               // 3
inventory.delete("tablets");
console.log(inventory.size);               // 2

// Clear all
inventory.clear();
```

Map Iteration

```
let scores = new Map([
    ["Alice", 95],
    ["Bob", 87],
    ["Charlie", 92]
]);

// Iterate over keys
for (let name of scores.keys()) {
    console.log(name);
}

// Iterate over values
for (let score of scores.values()) {
    console.log(score);
}

// Iterate over entries
for (let [name, score] of scores.entries()) {
    console.log(`${name}: ${score}`);
}

// Using forEach
scores.forEach((score, name) => {
    console.log(`${name} scored ${score}`);
});
```

Maps vs Objects

```
// Objects have string/symbol keys only
let obj = {
    "1": "string key",
    1: "number becomes string"
};

// Maps can have any type as key
let map = new Map();
```

```
map.set(1, "number key");
map.set("1", "string key");
map.set(true, "boolean key");
map.set({id: 1}, "object key");

console.log(map.size); // 4 - all different keys
```

Practice Exercises

Exercise 1: String Manipulation

Create a function that takes a sentence and returns it with:

- First letter of each word capitalized
- All extra spaces removed

```
function formatSentence(sentence) {
    // Your code here
}

console.log(formatSentence("  hello   world  ")); // "Hello World"
```

Exercise 2: Array Operations

Create a function that takes an array of numbers and returns an object with:

- sum: total of all numbers
- average: average of all numbers
- max: largest number
- min: smallest number

```
function analyzeNumbers(numbers) {
    // Your code here
}

console.log(analyzeNumbers([1, 2, 3, 4, 5]));
// Should return: {sum: 15, average: 3, max: 5, min: 1}
```

Exercise 3: Object Management

Create a simple inventory management system:

```
let inventory = {
    items: [],

    addItem: function(name, quantity, price) {
```

```javascript
            // Your code here
        },

        findItem: function(name) {
            // Your code here
        },

        updateQuantity: function(name, newQuantity) {
            // Your code here
        },

        getTotalValue: function() {
            // Your code here
        }
    };
```

Exercise 4: Date Calculator

Create a function that calculates age in years given a birth date:

```javascript
function calculateAge(birthDate) {
    // Your code here
}

console.log(calculateAge("1995-06-15")); // Should return current age
```

Exercise 5: Set and Map Practice

Create a function that analyzes text and returns:

- Unique words (using Set)
- Word frequency (using Map)

```javascript
function analyzeText(text) {
    // Your code here
    // Return object with uniqueWords (Set) and wordFrequency (Map)
}

let result = analyzeText("hello world hello");
console.log(result.uniqueWords);      // Set {"hello", "world"}
console.log(result.wordFrequency);    // Map {"hello" => 2, "world" => 1}
```

Key Takeaways

1. **Strings** are powerful for text manipulation with many built-in methods
2. **Arrays** are perfect for ordered collections with rich iteration methods
3. **Objects** represent entities with properties and methods

4. **Dates** handle all time-related operations and calculations
5. **Sets** ensure unique values and provide fast lookup
6. **Maps** offer flexible key-value storage with any data type as keys

Best Practices

- Use template literals for complex string formatting
- Prefer array methods like `map()`, `filter()`, and `reduce()` for functional programming
- Use meaningful property names in objects
- Always handle potential `undefined` values when accessing object properties
- Use Sets for unique collections and Maps when you need non-string keys
- Be careful with Date month indexing (0-based)

Next Steps

With these data structures mastered, you're ready to tackle more advanced JavaScript topics like:

- Destructuring assignment
- Promises and async programming
- Classes and prototypes
- Error handling
- DOM manipulation

Keep practicing and experimenting! 🎯