Strings & Template Literals

Topics

---

1. Strings in JavaScript

1.1 What is a String?

In JavaScript, a string is a sequence of characters enclosed in single (`'...'`), double (`"..."`), or backticks (`` `...` ``). Strings are used to represent text and can include letters, numbers, symbols, and even spaces.

Example:

```
let singleQuoteString = 'Hello, World!'; let doubleQuoteString = "Hello, World!"; let backtickString = `Hello, World!`;
```

1.2 Real-World Analogy

Think of a string as a necklace made up of different beads. Each bead represents a character, and the entire necklace is the string. Just as you can create different patterns with beads, you can create different sequences with characters in a string.

1.3 Properties of Strings in JavaScript

- **Immutability:** Strings in JavaScript are immutable, meaning once a string is created, it cannot be changed. Any operations that seem to modify a string will actually create a new string.

  Example:

  ```
  let str = "Hello"; str[0] = "Y"; // This won't change the string console.log(str); // Output: "Hello"
  ```

- **Length:** The `length` property returns the number of characters in a string.

  Example:

  ```
  let greeting = "Hello, World!"; console.log(greeting.length); // Output: 13
  ```

- **Indexing:** You can access individual characters in a string using bracket notation, with the index starting from 0.

  Example:

  ```
  let str = "JavaScript"; console.log(str[0]); // Output: "J" console.log(str[4]); // Output: "S"
  ```

1.4 Common String Methods

JavaScript provides a variety of methods for working with strings.

- `toUpperCase()` and `toLowerCase()`
  Convert the entire string to uppercase or lowercase.

  Example:

```
let str = "Hello, World!"; console.log(str.toUpperCase()); // Output: "HELLO, WORLD!" console.log(str.toLowerCase()); // Output: "hello, world!"
```

- `indexOf()`
  Returns the index of the first occurrence of a specified value in a string.

  Example:

```
let str = "Hello, World!"; console.log(str.indexOf("World")); // Output: 7
```

- `slice()`
  Extracts a part of a string and returns it as a new string.

  Example:

```
let str = "JavaScript"; // Basic usage to extract a substring console.log(str.slice(0, 4)); // Output: "Java" // Using only the start index console.log(str.slice(4)); // Output: "Script" // Using negative start index console.log(str.slice(-6)); // Output: "Script" // Using both start and end indices with negative values console.log(str.slice(-10, -6)); // Output: "Java" // Using a start index with a negative end index console.log(str.slice(0, -6)); // Output: "Java"
```

- `substring()`
  Similar to `slice()` but does not accept negative indices.

  Example:

```
let str = "JavaScript"; console.log(str.substring(4, 10)); // Output: "Script"
```

- `replace()`
  Replaces a specified value with another value in a string.

  Example:

```
let str = "Hello, World!"; let newStr = str.replace("World", "JavaScri
pt"); console.log(newStr); // Output: "Hello, JavaScript!"
```

- `split()`
  Splits a string into an array of substrings based on a specified delimiter.

  Example:

```
let str = "apple,banana,cherry"; let fruits = str.split(","); console.
log(fruits); // Output: ["apple", "banana", "cherry"]
```

Exercise 1: String Manipulation

1. Create a string variable with your full name.
2. Convert the string to uppercase.
3. Extract and log your first name using `slice()` or `substring()`.
4. Replace your first name with "Hello" in the string.

---

2. Template Literals

2.1 Introduction to Template Literals

Template literals, introduced in ES6, are an improved way to work with strings in JavaScript. They are enclosed in backticks (`` `...` ``) and allow for easier string interpolation and multi-line strings.

Example:

```
let name = "Alice"; let greeting = `Hello, ${name}!`; console.log(greetin
g); // Output: "Hello, Alice!"
```

2.2 Real-World Analogy

Imagine you're writing a letter and you have placeholders for the recipient's name and address. Template literals work like those placeholders, allowing you to easily insert variables into your strings.

2.3 String Interpolation

One of the key features of template literals is string interpolation, which allows you to embed variables and expressions directly into a string.

Example:

```
let item = "book"; let price = 19.99; let message = `The price of the ${item} is $${price}.`; console.log(message); // Output: "The price of the book is $19.99."
```

2.4 Multi-line Strings

Template literals make it easy to create multi-line strings without using escape characters.

Example:

```
let multiLineStr = `This is a multi-line string in JavaScript.`; console.log(multiLineStr);
```

2.5 Expressions in Template Literals

You can also embed any valid JavaScript expression inside a template literal, including function calls and mathematical operations.

Example:

```
let a = 5; let b = 10; let result = `The sum of ${a} and ${b} is ${a + b}.`; console.log(result); // Output: "The sum of 5 and 10 is 15."
```

Exercise 2: Template Literals Practice

1. Create a template literal that includes your name and age.
2. Use string interpolation to create a sentence that includes a calculation (e.g., your age next year).

3. Write a multi-line template literal describing your favorite hobby or activity.

---

3. Unique String Properties in JavaScript

3.1 `length` Property

The `length` property is unique to strings in JavaScript, allowing you to quickly determine the number of characters in a string.

Example:

```
let str = "JavaScript"; console.log(str.length); // Output: 10
```

3.2 Immutable Nature

JavaScript strings are immutable, meaning you cannot change the individual characters of a string once it is created. Any operation that appears to modify a string actually returns a new string.

Example:

```
let str = "hello"; str[0] = "H"; // This won't work console.log(str); // Output: "hello"
```

3.3 Automatic Type Conversion

JavaScript automatically converts other types to strings when needed, making it easy to concatenate strings with numbers or other data types.

Example:

```
let num = 10; let str = "The number is " + num; console.log(str); // Output: "The number is 10"
```

Exercise 3: Working with String Properties

1. Create a string variable and find its length using the `length` property.
2. Attempt to change a character in the string. What happens?

3. Concatenate a string with a number using the `+` operator.

Real-World Example: Creating a Dynamic Welcome Message

Imagine you are developing a web application that displays a personalized welcome message to users when they log in. You can use strings and template literals to create this message dynamically.

Example:

```
function welcomeUser(username) { let currentDate = new Date().toLocaleDat
eString(); let welcomeMessage = `Welcome back, ${username}! Today's date
```