Deep Dive into JavaScript Strings: Memory, Indexing & Methods

Instructor- Pramod Kumar Jena

---

1. What is a String?

In JavaScript, a **string** is a *primitive data type* that represents a sequence of **Unicode characters**.

✅ It's immutable – **once created, it can't be changed.**
✅ It can be enclosed in:

- `'single quotes'`
- `"double quotes"`
- `` `backticks` `` (for template literals)

```
let message = "Hello, World!";
```

---

2. How Strings Are Stored in Memory

Strings are stored in **memory as a series of 16-bit values (UTF-16 encoding)**.

- 🔹 Each character is stored using **Unicode encoding**.
- 🔹 JavaScript **does not store strings as arrays**, but they behave similarly in indexing.

```
let str = "Hello";
```

Internally:

Character	Unicode	Memory Index
H	72	0
e	101	1
l	108	2
l	108	3
o	111	4

➡️ Stored as: `'H' 'e' 'l' 'l' 'o'`
➡️ JavaScript strings are **indexed collections of characters**, but **not arrays.**

---

3. Indexing in JavaScript Strings

Each character in a string is assigned a **zero-based index**.

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

📌 Index starts at `0` and goes till `string.length - 1`.

❗ **Negative indexing doesn't work** in plain JS like in Python.

```
console.log(word[-1]); // undefined ❌
```

To get last character:

```
console.log(word[word.length - 1]); // t ✅
```

✍️ 4. Strings Are Immutable

Once a string is created, it **cannot be changed**. Any string operation creates a **new string** in memory.

```
let greet = "Hello";
greet[0] = "Y";
console.log(greet); // Still "Hello", not "Yello"
```

✅ To “change” a string, you must create a new one.

```
let newGreet = "Y" + greet.slice(1);
console.log(newGreet); // Yello
```

🧵 5. Template Literals (`)

A modern way of handling strings. Supports:

- Multiline strings
- String interpolation

```
let name = "Pramod";
let age = 25;
let sentence = `My name is ${name} and I am ${age} years old.`;
```

👍 Super useful for generating dynamic messages, emails, HTML blocks, etc.

🧰 6. Important String Methods

Some powerful built-in methods:

Method	Description	Example
`length`	Length of string	`"hello".length → 5`

`charAt(i)`	Character at index	`"hello".charAt(1) → e`
`slice(start, end)`	Extract part of string	`"hello".slice(1, 4) → ell`
`substring()`	Similar to slice	`"hello".substring(1, 4) → ell`
`substr()`	Deprecated (use slice)	`"hello".substr(1, 3)`
`includes()`	Checks substring	`"hello".includes("ell") → true`
`startsWith()`	Checks beginning	`"hello".startsWith("he") → true`
`endsWith()`	Checks ending	`"hello".endsWith("o") → true`
`replace()`	Replaces part of string	`"hello".replace("l", "r")`
`split()`	Splits by separator	`"a,b,c".split(",") → [a,b,c]`
`repeat(n)`	Repeats string	`"hi".repeat(3) → "hihihi"`

📦 7. Unicode, Emojis & Special Characters

JavaScript strings are UTF-16 encoded — each character takes 2 bytes.

```
let smile = "😊";
console.log(smile.length); // 2 😮 (surrogate pairs)
```

⚠️ Some emojis and rare characters are encoded as **two UTF-16 code units**.

---

8. Practice Problems

➤ **Task 1:**

Take a name from user via `prompt()` and:

- Print the first and last character.
- Convert name to uppercase and lowercase.

➤ **Task 2:**

Create a template literal that prints:

> Hello, my name is [your name] and I'm learning JavaScript from Pramod sir.

➤ **Task 3:**

Create a function that:

- Takes a sentence
- Returns the number of words using `.split(" ")`

💡 Quick Recap

- Strings are **indexed, immutable, UTF-16 encoded** sequences.
- Index starts at **0**, and there's no native negative indexing.
- Use **template literals** for clean, dynamic strings.
- Strings behave like arrays but aren't actual arrays.
- All string methods return **new strings** — the original stays unchanged.