Strings in JavaScript - Documentation

1. Introduction

In JavaScript, strings are sequences of characters used to represent text. Strings can be enclosed within:

- Single quotes ' '
- Double quotes " "
- Backticks ` ` (template literals)

Strings are primitive and immutable by default, ensuring safety and predictability in your code.

---

2. Primitive vs Non-Primitive in JavaScript

Primitive (Non-Reference) Types:

- Hold simple, single values (e.g., strings, numbers, booleans).
- Stored directly in the call stack.
- Immutable: Once created, the value cannot be changed.
- When assigned to another variable, a new copy is created.

Example:

```
let a = "hello";
let b = a;
b = "bye";
console.log(a); // "hello"
console.log(b); // "bye"
```

Explanation:

Assigning b = a creates a copy of the value. Changing b does not affect a due to immutability.

Non-Primitive (Reference) Types:

- Hold collections of values (e.g., objects, arrays, functions).
- Stored in heap memory, and variables store references (memory addresses) to the data.

- Mutable: The contents can be changed after creation.
- When assigned to another variable, both variables point to the same object in memory.

Example:

```
let a = { name: "sri" };
let b = a;
b.name = "srinivas";
console.log(a.name); // "srinivas"
console.log(b.name); // "srinivas"
```

Explanation:

Both a and b reference the same object, so changes through one reflect in the other.

---

3. Are Strings Always Primitive?

Strings are primitive and immutable unless created using the String constructor:

```
let str = new String("hello");
```

Using new String() creates a String object, making it a non-primitive (reference type).

---

4. Immutability and Mutability Explained

Immutability:

- Value cannot change after it is created.
- Modifying a variable creates a new copy instead of altering the original.
- Example: Primitive strings.

Mutability:

- Value can change after it is created.
- Modifications affect all references to that variable.
- Example: Objects and arrays.

---

5. Real-Life Analogies

Primitive:

Copying your friend's note and editing it does not change your friend's note.

Non-Primitive:

Changing your permanent address automatically updates your reference address in official documents.

---

6. Technical Insights

Primitives:

- Stored in the call stack.
- Assigning to another variable creates a new copy.

Non-Primitives:

- Stored in heap memory.
- Variables hold references to data.
- Assigning to another variable shares the same reference.

---

7. Summary Table

Aspect | Primitive Strings | Non-Primitive Strings

-------|-------------------|----------------------

Type | Primitive | Reference (Object)

Mutability | Immutable | Mutable

Storage | Call stack | Heap

Assignment | Creates a copy | Shares reference

Example | let a = "hello"; | let a = new String("hello");

---

8. Conclusion

- Strings in JavaScript are primitive and immutable by default.

- Understanding the difference between primitive and non-primitive types helps avoid bugs related to unintended mutations.

- Use primitive strings for simplicity and performance unless object wrappers are needed.