### **JavaScript String**s

A **string** in JavaScript is a sequence of characters used to represent text. It can be a single character, a word, a sentence, or even a whole paragraph.

* **Primitive Type:** Strings are one of JavaScript's primitive data types.
* **Immutable:** Once a string is created, its value *cannot be changed*. Any method that seems to "modify" a string actually returns a *new* string with the changes. The original string remains untouched.
* **Zero-indexed:** Like arrays, characters in a string are accessed by their position (index), starting from 0.

**1. Creating Strings:**

You can create strings using single quotes (' '), double quotes (" "), or backticks (`).

* **Single Quotes:**

let greeting = 'Hello, world!';

* **Double Quotes:**

let name = "Alice";

* **Backticks (Template Literals - ES6+):** These are powerful for embedding variables and multi-line strings.

let firstName = "John";

let lastName = "Doe";

let message = `Hello, ${firstName} ${lastName}! How are you?`; // Embedding variables

console.log(message); // Output: "Hello, John Doe! How are you?"

let multiLine = `This is

a multi-line

string.`;

console.log(multiLine);

**2. Accessing Characters:**

You can access individual characters in a string using bracket notation [] or the charAt() method.

let text = "JavaScript";

console.log(text[0]);       // Output: "J"

console.log(text[4]);       // Output: "S"

console.log(text.charAt(0));  // Output: "J"

console.log(text.charAt(99)); // Output: "" (empty string if index out of bounds)

console.log(text[99]);      // Output: undefined (if index out of bounds)

**3. String Length:**

The length property tells you the number of characters in the string.

let text = "JavaScript";

console.log(text.length); // Output: 10

### **JavaScript String Methods**

String methods are built-in functions that allow you to manipulate, search, and transform strings. Remember, they *return new strings* and do not change the original.

#### **I. Changing Case:**

1. **toLowerCase():**
   * Returns a new string with all characters converted to lowercase.

let original = "Hello World";

let lower = original.toLowerCase();

console.log(lower);    // "hello world"

console.log(original); // "Hello World" (original unchanged)

1. **toUpperCase():**
   * Returns a new string with all characters converted to uppercase.

let original = "Hello World";

let upper = original.toUpperCase();

consol e.log(upper);    // "HELLO WORLD"

console.log(original); // "Hello World" (original unchanged)

#### **II. Searching and Checking:**

1. **indexOf(searchValue, fromIndex):**
   * Returns the index of the *first* occurrence of searchValue within the string.
   * Returns -1 if searchValue is not found.
   * fromIndex (optional): The index to start the search from.

let text = "hello world hello";

console.log(text.indexOf("world")); // 6

console.log(text.indexOf("hello")); // 0

console.log(text.indexOf("hello", 1)); // 12 (starts search from index 1)

console.log(text.indexOf("xyz"));   // -1

1. **lastIndexOf(searchValue, fromIndex):**
   * Returns the index of the *last* occurrence of searchValue.
   * Returns -1 if not found.
   * fromIndex (optional): The index to start the search backwards from.

let text = "hello world hello";

console.log(text.lastIndexOf("hello")); // 12

console.log(text.lastIndexOf("o", 7)); // 6 (searches backwards from index 7)

1. **includes(searchValue, fromIndex):**
   * Checks if a string contains searchValue.
   * Returns true or false.
   * Case-sensitive.

let text = "apple, banana, orange";

console.log(text.includes("banana")); // true

console.log(text.includes("grape"));  // false

console.log(text.includes("Banana")); // false (case-sensitive)

1. **startsWith(searchString, position):**
   * Checks if a string begins with searchString.
   * Returns true or false.
   * position (optional): The position in the string to begin searching.

let text = "Hello world!";

console.log(text.startsWith("Hello")); // true

console.log(text.startsWith("world", 6)); // true (starts check from index 6)

console.log(text.startsWith("hi"));    // false

1. **endsWith(searchString, length):**
   * Checks if a string ends with searchString.
   * Returns true or false.
   * length (optional): Considers the string to be length characters long.

let text = "Hello world!";

console.log(text.endsWith("world!")); // true

console.log(text.endsWith("world", 11)); // true (checks "Hello world" - length 11)

console.log(text.endsWith("?"));      // false

#### **III. Extracting Parts of a String:**

1. **slice(startIndex, endIndex):**
   * Extracts a portion of a string and returns it as a *new string*.
   * startIndex: The index to start extraction (inclusive).
   * endIndex (optional): The index to end extraction (exclusive). If omitted, extracts to the end.
   * Can take negative indices (counts from the end).

let text = "JavaScript";

console.log(text.slice(0, 4));  // "Java"

console.log(text.slice(4));     // "Script"

console.log(text.slice(-6));    // "Script" (starts 6 from the end)

console.log(text.slice(2, -2)); // "vaScri"

1. **substring(startIndex, endIndex):**
   * Similar to slice(), but handles negative indices differently (treats them as 0).
   * If startIndex is greater than endIndex, it swaps them.

Let text = "JavaScript";

console.log(text.substring(0, 4));  // "Java"

console.log(text.substring(4, 0));  // "Java" (swaps 0 and 4)

console.log(text.substring(-5, 5)); // "JavaS" (treats -5 as 0)

1. **substr(startIndex, length) (Deprecated but still common):**
   * Extracts length characters from startIndex.
   * Consider using slice() instead.

let text = "JavaScript";

console.log(text.substr(4, 6)); // "Script" (starts at index 4, takes 6 characters)

#### **IV. Modifying and Replacing:**

1. **replace(searchValue, newValue):**
   * Searches for searchValue and replaces its *first* occurrence with newValue.
   * Returns a *new string*.
   * Can take a string or a regular expression as searchValue. For global replacement, use a regular expression with the g flag.

let text = "Dog bites dog.";

console.log(text.replace("dog", "cat")); // "cat bites dog." (only first "dog" replaced)

// Using a Regular Expression for global replacement (g flag)

let text2 = "The quick brown fox jumps over the lazy fox.";

console.log(text2.replace(/fox/g, "dog")); // "The quick brown dog jumps over the lazy dog."

1. **trim():**
   * Removes whitespace (spaces, tabs, newlines) from *both ends* of a string.
   * Returns a *new string*.

let text = "   Hello World   ";

console.log(`'${text.trim()}'`); // 'Hello World'

1. **trimStart() (or trimLeft()):**
   * Removes whitespace from the *beginning* of a string.

let text = "   Hello World   ";

console.log(`'${text.trimStart()}'`); // 'Hello World   '

1. **trimEnd() (or trimRight()):**
   * Removes whitespace from the *end* of a string.

let text = "   Hello World   ";

console.log(`'${text.trimEnd()}'`); // '   Hello World'

#### **V. Splitting and Repeating:**

1. **split(separator, limit):**
   * Splits a string into an array of substrings based on a separator.
   * Returns a *new array*.
   * separator (optional): The string or regex to use as a delimiter. If omitted, the array will have one element: the original string. If an empty string '' is used, it splits into individual characters.
   * limit (optional): A number specifying a limit on the number of splits.

let sentence = "JavaScript is awesome";

let words = sentence.split(" ");

console.log(words);    // ["JavaScript", "is", "awesome"]

let characters = "hello".split("");

console.log(characters); // ["h", "e", "l", "l", "o"]

let csvData = "apple,banana,orange";

let fruits = csvData.split(",");

console.log(fruits); // ["apple", "banana", "orange"]

1. **repeat(count):**
   * Constructs and returns a new string which contains the specified number of copies of the string on which it was called,1 concatenated together.

let star = "\*";

console.log(star.repeat(5)); // "\*\*\*\*\*"

let pattern = "abc";

console.log(pattern.repeat(3)); // "abcabcabc"

#### **VI. Other Useful Methods:**

1. **padStart(targetLength, padString):**
   * Pads the current string with another string until the resulting string reaches the targetLength. The padding is applied from the *start* (left) of the current string.

let num = "5";

console.log(num.padStart(2, "0")); // "05"

let id = "123";

console.log(id.padStart(5, "X"));  // "XX123"

1. **padEnd(targetLength, padString):**
   * Pads the current string with another string until the resulting string reaches the targetLength. The padding is applied from the *end* (right) of the current string.

let name = "Alice";

console.log(name.padEnd(10, ".")); // "Alice....."