STRING

A string is a data type used to represent text rather than numeric data. It's called a "string" because it's essentially a string of characters, where a character is a single unit of text, such as a letter, digit, punctuation mark, or whitespace.

Strings are for **storing text**

Strings are written **with quotes (single / double)**

Here are some key points about strings:

1**. Immutable**: JavaScript, strings are immutable, meaning once they are created, their value cannot be changed. Any operation that appears to modify a string actually creates a new string with the modified value.

2**. Encapsulation**: Strings are typically enclosed within quotation marks, either single (' ') or double (" "). This encapsulation helps the programming language distinguish between the actual text and other parts of the code.

3. **Character Encoding**: Internally, strings are represented using character encoding, which assigns a unique numerical value to each character. The most common encoding schemes are ASCII (American Standard Code for Information Interchange) and Unicode, which supports a much wider range of characters, including various alphabets, symbols, and emojis.

4. **Length**: The length of a string is the number of characters it contains. This can vary depending on the language and the encoding used. In JavaScript, you can get the length of a string using the `length` property.

5**. Operations**: Strings support various operations, such as concatenation (joining two or more strings together), substring extraction (getting a portion of a string), searching for substrings, converting case (to uppercase or lowercase), and more.

6. **Escaping Characters**: certain characters have special meanings within strings. To include these characters as part of the string itself, you often need to escape them using special sequences, such as `\n` for a newline or `\t` for a tab.

Strings are fundamental in programming and are used extensively in tasks ranging from simple text manipulation to more complex tasks like parsing data, pattern matching, and working with file input/output. Understanding how to work with strings effectively is a crucial skill for any programmer.

Declaration: You can declare a string by enclosing text within either single (' ') or double (" ") quotation marks. JavaScript allows flexibility in choosing between single or double quotes, but it's essential to be consistent.

```
let singleQuotedString = 'This is a single-quoted string.';
let doubleQuotedString = "This is a double-quoted string.";
```

Concatenation: You can concatenate (combine) strings using the `+` operator or the `concat()` method.

```
let greeting = "Hello";
let name = "John";
let message = greeting + ", " + name + "!"; // Using the + operator
let anotherMessage = greeting.concat(", ", name, "!"); // Using the concat() method
```

String Length: You can find the length of a string using the `length` property.

```
let text = "Hello, world!";
let length = text.length; // length will be 13
```

Accessing Characters: You can access individual characters in a string using bracket notation with the character's index.

```
let str = "Hello";
let firstChar = str[0]; // 'H'
```

String charAt():

Returns the character at a specified index in a string.

```
let str = "Hello, world!";
console.log(str.charAt(1)); // Output: 'e'
```

Explanation: The `charAt()` method returns the character at the specified index in the string. Indexing starts from 0.

String charCodeAt():

Returns the Unicode value of the character at a specified index in a string.

```
let str = "Hello";
console.log(str.charCodeAt(0)); // Output: 72
```

Explanation: This method returns the Unicode value (a numeric value) of the character at the specified index in the string.

String at():

Returns the character at a specified index in a string.

```
let str = "Hello, world!";
console.log(str.at(1)); // Output: 'e'
```

Explanation: Similar to `charAt()`, `at()` returns the character at the specified index in the string.

String [ ]:

Provides a way to access the character at a specified index in a string.

```
let str = "Hello, world!";
console.log(str[1]); // Output: 'e'
```

Explanation: This is another way to access characters in a string using bracket notation.

String slice():

Extracts a section of a string and returns it as a new string.

```
let str = "Hello, world!";
console.log(str.slice(7, 12)); // Output: 'world'
```

Explanation: The `slice()` method extracts a portion of the string between the specified start and end indices.

String substring():

Extracts the characters from a string between two specified indices and returns the new substring.

```
let str = "Hello, world!";
console.log(str.substring(7, 12)); // Output: 'world'
```

Explanation: Similar to `slice()`, `substring()` extracts a portion of the string between the specified start and end indices.

String substr():

Extracts a specified number of characters from a string, starting at a specified index.

```
let str = "Hello, world!";
console.log(str.substr(7, 5)); // Output: 'world'
```

Explanation: The `substr()` method extracts the specified number of characters starting from the specified index.

String toUpperCase():

Converts a string to uppercase letters.

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

Explanation: This method converts all the characters in a string to uppercase.

String toLowerCase():

Converts a string to lowercase letters.

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

Explanation: This method converts all the characters in a string to lowercase.

String concat():

Concatenates two or more strings and returns a new string.

```
let str1 = "Hello";
let str2 = "World";
console.log(str1.concat(", ", str2)); // Output: 'Hello, World'
```

Explanation: The `concat()` method joins two or more strings together.

String trim():

Removes whitespace from both ends of a string.

```
let str = "   Hello, world!   ";
console.log(str.trim()); // Output: 'Hello, world!'
```

Explanation: This method removes whitespace (spaces, tabs, and newlines) from the beginning and end of a string.

String trimStart():

Removes whitespace from the beginning of a string.

```
let str = "   Hello, world!   ";
console.log(str.trimStart()); // Output: 'Hello, world!   '
```

Explanation: Similar to `trim()`, `trimStart()` removes whitespace only from the beginning of a string.

String trimEnd():

Removes whitespace from the end of a string.

```
let str = "   Hello, world!   ";
console.log(str.trimEnd()); // Output: '   Hello, world!'
```

Explanation: Similar to `trim()`, `trimEnd()` removes whitespace only from the end of a string.

String padStart():

Pads a string with another string until the resulting string reaches the specified length.

```
let str = "5";
console.log(str.padStart(3, '0')); // Output: '005'
```

Explanation: This method pads the current string with another string (in this case, '0') until the resulting string reaches the specified length.

String padEnd():

Pads a string with another string until the resulting string reaches the specified length.

```
let str = "5";
console.log(str.padEnd(3, '0')); // Output: '500'
```

Explanation: Similar to `padStart()`, `padEnd()` pads the current string with another string until the resulting string reaches the specified length, but it adds padding to the end.

String repeat():

Returns a new string containing the specified number of copies of the string on which it was called.

```
let str = "Hello";
console.log(str.repeat(3)); // Output: 'HelloHelloHello'
```

Explanation: This method creates and returns a new string by concatenating the specified number of copies of the string on which it was called.

String replace():

Searches a string for a specified value or regular expression and returns a new string where the specified values are replaced.

```
let str = "Hello, world!";
console.log(str.replace("world", "John")); // Output: 'Hello, John!'
```

Explanation: This method searches a string for a specified value or regular expression and returns a new string where the specified values are replaced with another string.

String replaceAll():

Searches a string for all occurrences of a specified value or regular expression and returns a new string where all occurrences are replaced.

```
let str = "Hello, world! Hello, universe!";
console.log(str.replaceAll("Hello", "Hi")); // Output: 'Hi, world! Hi, universe!'
```

Explanation: Similar to `replace()`, `replaceAll()` replaces all occurrences of a specified value or regular expression with another string.

String split():

Splits a string into an array of substrings based on a specified separator.

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

Explanation: In this example, the split() method splits the original string str into an array of substrings using the comma , as the separator. Each substring between the commas becomes an element in the resulting array.

String indexOf():

Returns the index within the calling String object of the first occurrence of the specified value, starting the search at fromIndex. Returns -1 if the value is not found.

```
let str = "Hello, world!";
console.log(str.indexOf("o")); // Output: 4
```

```
let str = "Hello, world!";
console.log(str.indexOf("o", 5)); // Output: 7
```

String lastIndexOf():

Returns the index within the calling String object of the last occurrence of the specified value. Returns -1 if the value is not found.

```
let str = "Hello, world!";
console.log(str.lastIndexOf("o")); // Output: 8
```

```
let str = "Hello, world!";
console.log(str.lastIndexOf("o", 7)); // Output: 4
```

String search():

Executes a search for a match between a regular expression and this String object.

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

String match():

Retrieves the result of matching a string against a regular expression.

```
let str = "The rain in Spain falls mainly in the plain";
console.log(str.match(/ain/g)); // Output: [ 'ain', 'ain', 'ain' ]
```

String matchAll():

Returns an iterator of all results matching a string against a regular expression, including capturing groups.

```
let str = "The rain in Spain falls mainly in the plain";
let regex = /ain/g;
let matches = str.matchAll(regex);
for (const match of matches) {
    console.log(match);
}
```

String includes():

```
let str = "Hello, world!";
console.log(str.includes("world")); // Output: true
```

```
let str = "Hello, world!";
console.log(str.includes("world", 7)); // Output: false
```

Determines whether one string may be found within another string, returning true or false as appropriate.

String startsWith():

Determines whether a string begins with the characters of a specified string, returning true or false as appropriate.

```
let str = "Hello, world!";
console.log(str.startsWith("Hello")); // Output: true
```

String endsWith():

Determines whether a string ends with the characters of a specified string, returning true or false as appropriate.

```
let str = "Hello, world!";
console.log(str.endsWith("world!")); // Output: true
```

Template Literals: ES6 introduced template literals, which are enclosed by backticks (` `) and allow embedding expressions and multiline strings.

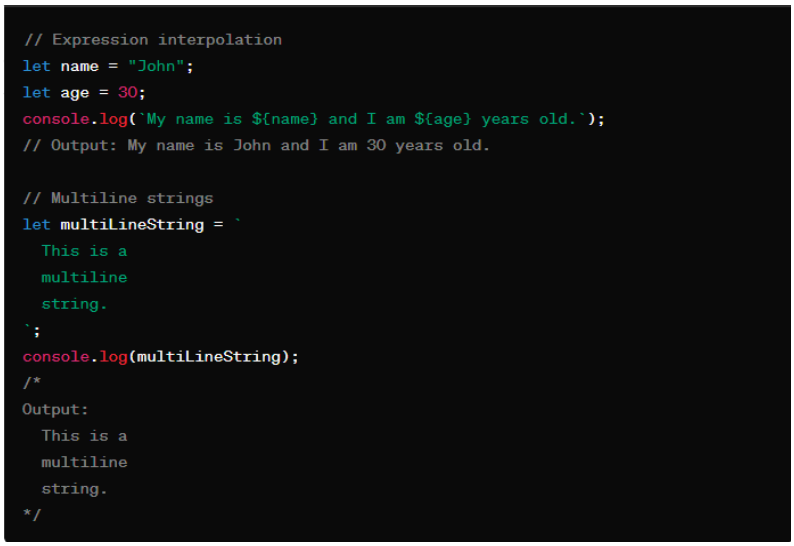
Template literals, introduced in ECMAScript 6 (ES6), provide a more convenient and flexible way to work with strings in JavaScript. They are enclosed by backticks (\`\`) rather than single or double quotes and allow for easy embedding of expressions and multiline strings.

Expression Interpolation:

Template literals allow you to embed expressions directly into the string by wrapping them in `${}`. The expression within `${}` is evaluated and its result is inserted into the string.

Multiline Strings:

With template literals, multiline strings can be created without the need for escape characters like `\n`. You can simply write multiline text as it appears, making code more readable.



```
// Expression interpolation
let name = "John";
let age = 30;
console.log(`My name is ${name} and I am ${age} years old.`);
// Output: My name is John and I am 30 years old.

// Multiline strings
let multiLineString = `
  This is a
  multiline
  string.
`;
console.log(multiLineString);
/*
Output:
  This is a
  multiline
  string.
*/
```

In the first example, `${name}` and `${age}` are expressions that are evaluated and inserted into the string. This feature makes it easy to include dynamic values within strings.

In the second example, the string spans multiple lines without the need for escape characters. This improves code readability, especially for longer strings or when formatting text blocks.

template literals provide a cleaner and more expressive way to work with strings in JavaScript, particularly in scenarios where dynamic content or multiline text is involved.