STRING

String allows us to handle text based information such as word, sentence and paragraph such as “The Target Institute”. **A string can represent an object** as well as the primitive data type. JavaScript automatically converts primitive strings to String objects.

Strings in JavaScript are immutable, meaning that once a string is created, it cannot be changed. However, you can manipulate strings using various methods and operations such as concatenation, substring extraction, finding the length, etc.

There are two ways to create string object

- By string literal
- By using constructor (using new keyword)

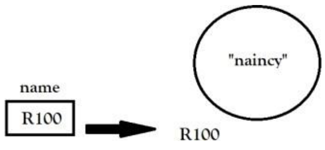
By string literal : The string literal is created using **single and double quotes**.

e.g.

```
var name="naincy";
or
var name= 'naincy';
```

e.g.

```
<html>
<body>
 <script>
 var str = "naincy";
 document.write(str);
</script>
</body>
</html>
```



By using constructor (using new keyword)

e.g.

```
        var name = new String("naincy");
```

e.g.

```
<html>
<body>
  <script>
    var name = new String("naincy");
    document.write(name);
  </script>
</body>
</html>>
```

Length

In JavaScript, the length property of a string returns the number of characters in that string. This property is accessible on any string object.

e.g.

```
<html>
<body>
  <script>
    let name = "naincy";
    console.log(name.length);
  </script>
</body>
</html>
```

trim ()

The trim() method in JavaScript is used to remove whitespace (spaces, tabs, newlines, etc.) from both ends of a string. This method does not modify the original string but returns a new string with the leading and trailing whitespace removed.

e.g.

```
<html>
<body>
  <script>
    let course = "      JavaScript is cool     ";
    console.log(course);
    console.log(course.trim());
  </script>
</body>
</html>
```

toUpperCase() & toLowerCase()

The toUpperCase() and toLowerCase() methods in JavaScript are used to convert a string to either uppercase or lowercase characters, respectively. These methods do not modify the original string but return a new string with the characters converted to the desired case.

e.g.

```
<html>
<body>
  <script>
    let course = "naincy";
    console.log(course.toUpperCase());
    console.log(course.toLowerCase());
  </script>
</body>
</html>
```

Slice()

The slice() method in JavaScript is used to extract a section of a string and return it as a new string without modifying the original string. It takes one or two parameters: the starting index from where to begin extraction and the optional ending index (exclusive) where to end extraction. If the ending index is omitted, slice() extracts to the end of the string.

e.g.-1
```
<html>
<body>
  <script>
   let course = "JavaScript is cool";
   console.log(course.slice(5));
   console.log(course.slice(5,10));
   let x= course.slice(5,10)
   document.writeln(x.toUpperCase());
  </script>
</body>
</html>
```

e.g.2
```
<html>
<body>
  <script>
    var str = "JavaScript is a GREAT Language";
    var a = str.slice(-2);
    document.write(a);
  </script>
  </script>
</body>
</html>
```

charCodeAt()

The charCodeAt() method in JavaScript is used to retrieve the Unicode value (UTF-16 code unit) of the character at a specified index in a string. Unicode is a standardized character encoding system that assigns a unique numeric value to each character in most of the world's writing

e.g.
```
<html>
<body>
  <script>
    var str = "naincy";
    var a = str.charCodeAt(1);
    console.log(a)
  </script>
</body>
</html>
```

String.fromCharCode()

The fromCharCode() method in JavaScript is the counterpart of the charCodeAt() method. While charCodeAt() returns the Unicode value of a character at a specified index in a string, fromCharCode() does the opposite: it takes one or more Unicode values (UTF-16 code units) and returns the corresponding string.
Syntax :
String.fromCharCode(charCode1, charCode2, …)

e.g.

```
let str = String.fromCharCode(72, 101, 108, 108, 111);
console.log(str); // Output: "Hello"
```

concat()

The concat() method in JavaScript is used to concatenate one or more strings together, creating and returning a new string. It does not modify the original strings; instead, it returns a new string containing the concatenated values of the strings provided as arguments.

Syntax:

```
string.concat(string1, string2, ..., stringN)
```

e.g.

```
<html>
<body>
  <script>
    var str = "JavaScript is a GREAT Language";
    var str2 = "Hello";
    var str3 = "World";
    var a = str.concat(str2, str3);
    document.write(a);
  </script>
</body>
</html>
```

Split()

The split() method in JavaScript is used to split a string into an array of substrings based on a specified separator and returns the array. This method does not modify the original string.

Syntax:

```
string.split(separator,  limit)
```

- separator: The character or regular expression used to specify where to split the string. If omitted or undefined, the entire string is split into single-character substrings.
- limit (optional): An integer specifying a limit on the number of splits to be found. The split method still splits the string completely if this argument is not specified.

e.g.-1

```
<html>
<body>
  <script>
    var str = "JavaScript is a GREAT Language";
    var a = str.split(" ");
    console.log(a);
  </script>
</body>
</html>
```

e.g.-2

```
<html>
<body>
 <script>
   var str = "samosa, kachodi,  bedai,jalebi";
   var a = str.split(",",2);
   console.log(a);
 </script>
</body>
</html>
```

repeat()

The repeat() method in JavaScript is used to construct and return a new string by concatenating the specified string a given number of times. It takes an integer parameter specifying the number of times to repeat the string. The original string remains unchanged.

Syntax:

```
string.repeat(count)
```

- count: An integer specifying the number of times the string should be repeated. It must be a non-negative integer. If count is negative or if it is equal to Infinity, a RangeError will be thrown. If count is equal to 0, an empty string is returned.

e.g.

```
<html>
<body>
  <script>
    var str = "JavaScript is a GREAT Language";
    var a = str.repeat(2);
    document.write(a);
  </script>
</body>
</html>
```

substr()

The substr() method in JavaScript is used to extract a portion of a string and return it as a new string. It takes two parameters: the starting index from which to begin extraction and the number of characters to extract. If the second parameter is omitted, the method extracts characters to the end of the string.

Syntax:

```
string.substr(startIndex)
string.substr(startIndex, length)
```

- startIndex: The index at which to begin extraction. If negative, it indicates an offset from the end of the string. If startIndex is greater than or equal to the length of the string, an empty string is returned.
- length (optional): The number of characters to extract. If omitted or undefined, characters are extracted to the end of the string.

Note:

The method returns a new string containing the extracted characters.

e.g.-1

```
<html>
<body>
   <script>
      var str = "JavaScript is a GREAT Language";
      var a = str.substr(2);
      document.write(a);
   </script>
</body>
</html>
```

e.g.-2

```
<html>
<body>
   <script>
      var str = "JavaScript is a GREAT Language";
      var a = str.substring(3, 7);
      document.write(a);
     </script>
</body>
</html>
```

Handling Negative Indices:

e.g.

```
let str = "Hello World";
let substr = str.substr(-5);
console.log(substr); // Output: "World"
```

toString()

In JavaScript, the toString() method is used to convert a value to its string representation. It is available on most JavaScript objects, including numbers, booleans, arrays, and even functions.

e.g.-1

```
<html>
<body>
  <script>
    let arr = [1, 2, 3];
    let str = arr.toString();
    console.log(str); // Output: "1,2,3"
  </script>
</body>
</html>
```

e.g.-2

```
<html>
<body>
   <script>
      var i = 50;
      var a = i.toString();
      document.write(a + 20);
    </script>
</body>
</html>
```

includes()

The includes() method in JavaScript is used to determine whether a string contains another string within it. It returns true if the specified substring is found within the string, and false otherwise. This method is case-sensitive.

Syntax:

```
string.includes(substring,  startIndex)
```

- substring: The substring to search for within the string.
- startIndex (optional): The index at which to begin searching for the substring. If omitted, the search starts at index 0.

The method returns a boolean value: true if the substring is found within the string, and false otherwise.

e.g.

```
<html>
<body>
   <script>
      var str = "JavaScript is a GREAT Language";
      var a = str.includes("GREAT"); //case sensitive
      document.write(a);
    </script>
</body>
</html>
```

startsWith()

The startsWith() method in JavaScript is used to determine whether a string starts with the characters of a specified string. It returns true if the string begins with the characters of the specified substring, and false otherwise. This method is case-sensitive.

Syntax:

```
string.startsWith(searchString,  startIndex)
```

- searchString: The substring to search for at the beginning of the string.
- startIndex (optional): The index at which to begin searching for the substring. If omitted, the search starts at index 0.

The method returns a boolean value: true if the string starts with the specified substring, and false otherwise.

e.g.

```
<html>
<body>
  <script>
     var str = "JavaScript is a GREAT Language";
     var a = str.startsWith("JavaScript");
     console.log(a)
   </script>
</body>
</html>
```

endsWith()

The endsWith() method in JavaScript is used to determine whether a string ends with the characters of a specified string. It returns true if the string ends with the characters of the specified substring, and false otherwise. This method is case-sensitive.

Syntax

```
string.endsWith(searchString,  length)
```

- searchString: The substring to search for at the end of the string.
- length (optional): The number of characters from the end of the string to consider when performing the search. If omitted, the length of the string is used.

The method returns a boolean value: true if the string ends with the specified substring, and false otherwise.

e.g.

```
<html>
<body>
  <script>
     var str = "JavaScript is a GREAT Language";
     var a = str.endsWith("Language");
     console.log(a)
   </script>
</body>
</html>
```

search()

The search() method in JavaScript is used to search for a specified substring within a string. It returns the index of the first occurrence of the specified substring, or -1 if the substring is not found. This method allows you to search for a substring using a regular expression pattern.

Syntax:

```
string.search(regexp)
```

- regexp: A regular expression pattern to search for within the string.
- The method returns an integer value: the index of the first occurrence of the substring that matches the regular expression pattern, or -1 if no match is found.

e.g.

```
<html>
<body>
   <script>
      var str = "JavaScript is a GREAT Language";
      var a = str.search("is");
      console.log(a);
   </script>
</body>
</html>
```

charAt()

The charAt() method in JavaScript is used to retrieve the character at a specified index within a string. It takes one parameter: the index of the character you want to retrieve. The index is zero-based, meaning the first character has an index of 0, the second character has an index of 1, and so on. If the specified index is out of range (i.e., less than 0 or greater than or equal to the length of the string), an empty string is returned.

Syntax

```
string.charAt(index)
```

- index: The index of the character to retrieve from the string.

The method returns a string containing the character at the specified index.

e.g.

```
<html>
<body>
   <script>
      var str="javascript";
      document.write(str.charAt(2));
      </script>
<body>
</html>
```

indexOf()

The indexOf() method in JavaScript is used to search for a specified substring within a string. It returns the index of the first occurrence of the specified substring, or -1 if the substring is not found. This method allows you to search for a substring starting from a specified index within the string.

Syntax

string.indexOf(searchValue, **startIndex**)

- searchValue: The substring to search for within the string.
- startIndex (optional): The index at which to begin searching for the substring. If omitted, the search starts at index 0.

The method returns an integer value: the index of the first occurrence of the substring within the string, or -1 if the substring is not found.

e.g.

```
<html>
<body>
```

```
  <script>
    var s1="He is krishna and living at gwalior "
    var n=s1.indexOf("i",4);
    document.write(n);
    </script>
<body>
</html>
```

lastIndexOf()

The lastIndexOf() method in JavaScript is similar to the indexOf() method, but it searches for the last occurrence of a specified substring within a string, rather than the first occurrence. It returns the index of the last occurrence of the specified substring, or -1 if the substring is not found.
Syntax

```
string.lastIndexOf(searchValue,  endIndex)
```

- searchValue: The substring to search for within the string.
- endIndex (optional): The index at which to stop searching for the substring. If omitted, the search goes through the entire string from right to left.

The method returns an integer value: the index of the last occurrence of the substring within the string, or -1 if the substring is not found.

e.g.

```
<html>
<body>
  <script>
    var s1="He is krishna and living at gwalior"
    var n=s1.lastIndexOf("i");
    document.write(n);
    </script>
<body>
</html>
```

match()

The match() method in JavaScript is used to search a string for a specified pattern (regular expression), and returns an array containing the matched substrings or null if no match is found.
Syntax

```
string.match(regexp)
```

- regexp: A regular expression pattern to search for within the string.

The method returns an array containing the matched substrings if any, along with additional information, or null if no match is found. If the regular expression pattern includes capturing groups, the returned array will include the matched substrings along with the captured groups.
e.g.

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

e.g.
```
<html>
<body>
  <script>
    let str = "The rain in Spain falls mainly in the plain";
    let matches = str.match(/[a-z]ain/g);
    console.log(matches);
  </script>
  <body>
</html>
```

replace()

The replace() method in JavaScript is used to search a string for a specified substring or regular expression pattern, and then replace occurrences of that substring or pattern with a replacement string. It returns a new string with the replacements made, leaving the original string unchanged.
Syntax:
```
string.replace(searchValue,  replaceValue)
```

- searchValue: The substring or regular expression pattern to search for within the string.
- replaceValue: The string that will replace the matched substring or pattern.

The method returns a new string with the specified substring or pattern replaced by the replacement string. If the searchValue is a regular expression with the global flag (/g), all occurrences of the pattern will be replaced. Otherwise, only the first occurrence will be replaced.

e.g.
```
<html>
<body>
  <script>
    let str = "Hello World";
    let newStr = str.replace("World", "Universe");
    console.log(newStr); // Output: "Hello Universe"
    console.log(str); // Output: "Hello World"
  </script>
  <body>
</html>
```

valueOf()

The valueOf() method in JavaScript is used to retrieve the primitive value of an object. It is called automatically by JavaScript when a primitive value representation of an object is needed, such as when using an object in a context where a primitive value is expected, like concatenating with a string or performing arithmetic operations.
e.g.
```
<html>
<body>
  <script>
    let strObject = new String("45");
    console.log(strObject.valueOf() * 2);
  </script>
  <body>
</html>
```

toLocaleUpperCase()

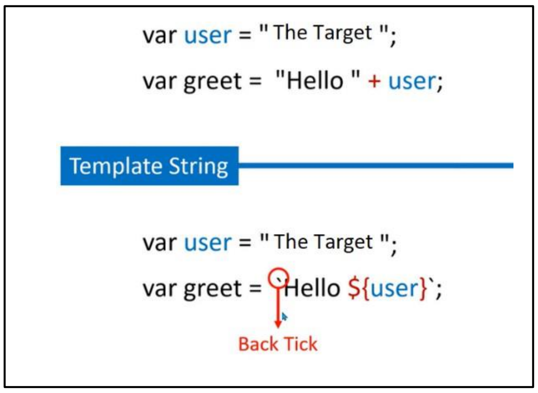
The toLocaleUpperCase() method in JavaScript is similar to the toUpperCase() method, but it converts all the characters in a string to uppercase according to the locale-specific case mappings. This means that it takes into account the language and cultural conventions of the locale (e.g., language, country, region) specified by the user or system.
e.g.

```
<html>
<body>
  <script>
    let str = "hallo welt";
    let upperCaseStr = str.toLocaleUpperCase("de-DE");
    console.log(upperCaseStr); // Output: "HALLO WELT"
  </script>
  <body>
</html>
```

String templates

String templates, also known as template literals, are a feature in JavaScript that allow for easy and more readable string interpolation and multiline strings. They use backticks (`) instead of single or double quotes.

String templates provide a more concise and readable way to construct strings, especially when including variables or expressions. They're widely used in modern JavaScript development for generating dynamic content and constructing complex strings.



```
var user = " The Target ";
var greet =  "Hello " + user;
```

```
var user = " The Target ";
var greet = `Hello ${user}`;
```

e.g.-1

```
<html>
<body>
  <script>
    let name = "Alice";
    let age = 30;
    // Using string templates
    let greeting = `Hello, my name is ${name} and I am ${age} years old.`;
    console.log(greeting);
  </script>
  <body>
</html>
```

In this example, ${name} and ${age} are placeholders inside the backticks. They will be replaced with the values of the name and age variables, respectively, when the string is evaluated.

String templates also support multiline strings:

e.g.-2

```
<html>
<body>
  <script>
    let multilineString = `
  This is a multiline string.
  It can span across multiple lines
  without the need for escape characters.
`;
console.log(multilineString);
</script>
<body>
</html>
```

e.g.-3

```
<html>
<body>
  <script>
    let a = 10;
    let b = 20;
    let result = `The sum of ${a} and ${b} is ${a + b}.`;
    console.log(result); // Output: "The sum of 10 and 20 is 30."
  </script>
  <body>
</html>
```

Akhilesh Kumar Gupta
(Technical Training specialist)
TechCarrel LLP