A string in C# is immutable, and all string methods return a new string unless specified otherwise.

| **Method** | **Parameters (if any)** | **Return Type** | **Description** |
| --- | --- | --- | --- |
| **Clone()** | None | Object | Returns a reference to this string instance. (Not a deep copy.) |
| **Compare(string, string)** | - string strA: The first string to compare.  - string strB: The second string to compare. | int | Compares two strings and returns an integer indicating their relative position in the sort order. |
| **CompareOrdinal(string, string)** | - string strA - string strB | int | Compares two strings lexicographically by evaluating the numeric values of corresponding Char objects in each string. |
| **CompareTo(string)** | - string strB: The string to compare with the current instance. | int | Compares this instance with another specified string. |
| **Concat(params string[])** | - params string[]: The strings to concatenate. | string | Concatenates one or more strings. |
| **Contains(string)** | - string value: The string to check for within this instance. | bool | Returns true if the specified string occurs within this string. |
| **Copy(string)** | - string str: The string to copy. | string | Creates a new instance of a string with the same value as a specified string. *(Deprecated; prefer String.Clone() or assignment.)* |
| **EndsWith(string)** | - string value: The string to compare. - Optional: StringComparison comparisonType. | bool | Returns true if the string ends with the specified value. |
| **Equals(object)** | - object obj: The object to compare with the current instance. | bool | Checks if the current string equals the specified object. |
| **Format(string, params object[])** | - string format: A composite format string.  - params object[] args: The objects to format. | string | Replaces format items in a string with the string representation of corresponding objects. |
| **IndexOf(char)** | - char value: The character to search for.  - Optional: int startIndex, int count. | int | Returns the zero-based index of the first occurrence of the specified character. |
| **Insert(int, string)** | **- int startIndex: The position to insert.  - string value: The string to insert.** | string | Inserts a specified string at a specified index. |
| **IsNullOrEmpty(string)** | **- string value: The string to test.** | **bool** | **Indicates whether the string is null or an empty string.** |
| **IsNullOrWhiteSpace(string)** | **- string value: The string to test.** | **bool** | **Indicates whether the string is null, empty, or consists only of white-space characters.** |
| **Join(string, string[])** | **- string separator: The string separator.  - string[] value: The array of strings to join.** | **string** | **Concatenates the elements of a string array, using the specified separator.** |
| **LastIndexOf(char)** | **- char value: The character to search for.  - Optional: int startIndex, int count.** | **int** | **Returns the zero-based index of the last occurrence of the specified character.** |
| **PadLeft(int)** | **- int totalWidth: The number of characters for the resulting string.  - Optional: char paddingChar.** | **string** | **Pads the left side of the string with spaces or a specified character to reach a total width.** |
| **PadRight(int)** | **- int totalWidth: The number of characters for the resulting string.  - Optional: char paddingChar.** | **string** | **Pads the right side of the string with spaces or a specified character to reach a total width.** |
| **Remove(int)** | **- int startIndex: The starting index to remove characters.  - Optional: int count.** | **string** | **Removes all characters from the string, starting at a specified position.** |
| **Replace(string, string)** | **- string oldValue: The string to be replaced.  - string newValue: The string to replace with.** | **string** | **Returns a new string where all occurrences of oldValue are replaced by newValue.** |
| **Split(char[])** | **- char[] separator: The delimiters.  - Optional: StringSplitOptions options.** | **string[]** | **Splits a string into an array of substrings based on specified delimiters.** |
| **StartsWith(string)** | **- string value: The string to compare. - Optional: StringComparison comparisonType.** | **bool** | **Returns true if the string starts with the specified value.** |
| **Substring(int)** | **- int startIndex: The zero-based starting character position.  - Optional: int length.** | **string** | **Retrieves a substring from this instance.** |
| **ToCharArray()** | **None** | **char[]** | **Converts a string into a character array.** |
| **ToLower()** | **None** | **string** | **Returns a copy of the string in lowercase.** |
| **ToUpper()** | **None** | **string** | **Returns a copy of the string in uppercase.** |
| **Trim()** | **- Optional: char[] trimChars.** | **string** | **Removes all leading and trailing white-space characters or specified characters.** |
| **TrimEnd()** | **- Optional: char[] trimChars.** | **string** | **Removes trailing white-space characters or specified characters.** |
| **TrimStart()** | **- Optional: char[] trimChars.** | **string** | **Removes leading white-space characters or specified characters.** |

------------------------------------------- **String end---------------------------**

**Using StringBuilder in C#**

**The StringBuilder class is used to manipulate strings efficiently, especially when concatenating or modifying them repeatedly.**

**Declaration**

**StringBuilder sb = new StringBuilder();**

**Common Methods**

1. **Append**

**sb.Append("Hello ");**

**sb.Append("World!");**

1. **Insert**

**sb.Insert(5, " Beautiful");**

1. **Replace**

**sb.Replace("World", "C#");**

1. **Remove**

**sb.Remove(5, 9);**

1. **ToString**

**string result = sb.ToString();**

1. **AppendLine**

**sb.AppendLine("This is a line.");**

**Example Usage**

**StringBuilder sb = new StringBuilder();**

**sb.Append("Hello");**

**sb.Append(" ");**

**sb.AppendLine("World!");**

**sb.Replace("World", "C#");**

**Console.WriteLine(sb.ToString());**