STRING

String Functions:

1. Length:

- **Purpose:** Returns the number of characters in the string.
- **Syntax:** string.Length
- **Example:**

C#

```
string str = "Hello, World!";
int length = str.Length; // length will be 13
```

2. CompareTo:

- **Purpose:** Compares the current string with another string.
- **Syntax:** string.CompareTo(string other)
- **Returns:**
  - -1 if the current string is less than the other string.
  - 0 if the strings are equal.
  - 1 if the current string is greater than the other string.
- **Example:**

C#

```
string str1 = "Apple";
string str2 = "Banana";
int result = str1.CompareTo(str2); // result will be -1
```

3. Equals:

- **Purpose:** Determines whether two strings are equal.
- **Syntax:** string.Equals(string other)
- **Returns:** True if the strings are equal, otherwise false.
- **Example:**

C#

```
string str1 = "Hello";
string str2 = "Hello";
bool areEqual = str1.Equals(str2); // areEqual will be true
```

4. Contains:

- **Purpose:** Checks if a substring exists within the current string.
- **Syntax:** string.Contains(string value)
- **Returns:** True if the substring exists, otherwise false.
- **Example**

C#

```
string str = "Hello, World!";
bool contains = str.Contains("World"); // contains will be true
```

5. StartsWith:

- **Purpose:** Checks if the current string starts with a specified substring.
- **Syntax:** string.StartsWith(string value)
- **Returns:** True if the string starts with the substring, otherwise false.
- **Example:**

C#

```
string str = "Hello, World!";
bool startsWith = str.StartsWith("Hello"); // startsWith will be true
```

6. EndsWith:

- **Purpose:** Checks if the current string ends with a specified substring.
- **Syntax:** string.EndsWith(string value)
- **Returns:** True if the string ends with the substring, otherwise false.
- **Example:**

C#

```
string str = "Hello, World!";
bool endsWith = str.EndsWith("World!"); // endsWith will be true
```

7. IndexOf:

- **Purpose:** Finds the index of the first occurrence of a specified character or substring within the current string.
- **Syntax:** string.IndexOf(char value) or string.IndexOf(string value)
- **Returns:** The index of the first occurrence, or -1 if not found.
- **Example:**

C#

```
string str = "Hello, World!";
int index = str.IndexOf('o'); // index will be 4
```

8. LastIndexOf:

- **Purpose:** Finds the index of the last occurrence of a specified character or substring within the current string.
- **Syntax:** string.LastIndexOf(char value) or string.LastIndexOf(string value)
- **Returns:** The index of the last occurrence, or -1 if not found.
- **Example:**

C#

```
string str = "Hello, World!";
int index = str.LastIndexOf('o'); // index will be 7
```

9. Substring:

- **Purpose:** Extracts a substring from the current string.
- **Syntax:** string.Substring(int startIndex) or string.Substring(int startIndex, int length)
- **Returns:** The extracted substring.
- **Example:**

C#

```
string str = "Hello, World!";
string substring = str.Substring(7); // substring will be "World!"
```

10. Replace:

- **Purpose:** Replaces all occurrences of a specified character or substring with another string.
- **Syntax:** string.Replace(char oldChar, char newChar) or string.Replace(string oldString, string newString)
- **Returns:** The modified string.
- **Example:**

C#

```
string str = "Hello, World!";
string replaced = str.Replace("World", "Universe"); // replaced will be "Hello, Universe!"
```

11. ToUpper:

- **Purpose:** Converts all characters in the current string to uppercase.
- **Syntax:** string.ToUpper()
- **Returns:** The uppercase string.
- **Example:**

C#

```
string str = "Hello, World!";
string upper = str.ToUpper(); // upper will be "HELLO, WORLD!"
```

12. ToLower:

- **Purpose:** Converts all characters in the current string to lowercase.
- **Syntax:** string.ToLower()
- **Returns:** The lowercase string.
- **Example:**

```
string str = "Hello, World!";
string lower = str.ToLower(); // lower will be "hello, world!"
```

13. Trim:

- **Purpose:** Removes leading and trailing white space characters from the current string.
- **Syntax:** string.Trim()
- **Returns:** The trimmed string.
- **Example:**

C#

```
string str = "  Hello, World!  ";
string trimmed = str.Trim(); // trimmed will be "Hello, World!"
```

String Methods:

1. Split:

- **Purpose:** Splits the current string into a string array based on a specified delimiter.
- **Syntax:** string.Split(char[] separator) or string.Split(char[] separator, StringSplitOptions options)
- **Returns:** The string array.
- **Example:**

C#

```
string str = "Hello,World,How,Are,You";
string[] words = str.Split(','); // words will be ["Hello", "World",
"How", "Are", "You"]
```

2. Join:

- **Purpose:** Joins the elements of a string array into a single string using a specified delimiter.
- **Syntax:** string.Join(string separator, string[] values)
- **Returns:** The joined string.
- **Example:**

```
string[] words = {"Hello", "World", "How", "Are", "You"};
string joined = string.Join(", ", words); // joined will be "Hello,
World, How, Are, You"
```

3. Format:

- **Purpose:** Formats a string using placeholders and corresponding values.
- **Syntax:** string.Format(string format, object[] args)
- **Returns:** The formatted string.
- **Example**

C#

```
string name = "Alice";
int age = 30;
string formatted = string.Format("Hello, {0}! You are {1} years old.",
name, age);
// formatted will be "Hello, Alice! You are 30 years old."
```

4. PadLeft:

- **Purpose:** Pads the left side of the current string with a specified character to a specified length.
- **Syntax:** string.PadLeft(int totalWidth) or string.PadLeft(int totalWidth, char paddingChar)
- **Returns:** The padded string.
- **Example:**

C#

```
string str = "Hello";
string padded = str.PadLeft(10, '-'); // padded will be "-----Hello"
```

5. PadRight:

- **Purpose:** Pads the right side of the current string with a specified character to a specified length.
- **Syntax:** string.PadRight(int totalWidth) or string.PadRight(int totalWidth, char paddingChar)
- **Returns:** The padded string.
- **Example:**

C#

```
string str = "Hello";
string padded = str.PadRight(10, '-'); // padded will be "Hello-----"
```

6. Remove:

- **Purpose:** Removes a specified number of characters from the current string, starting at a specified index.
- **Syntax:** string.Remove(int startIndex) or string.Remove(int startIndex, int count)
- **Returns:** The modified string.
- **Example:**

C#

```
string str = "Hello, World!";
string removed = str.Remove(7, 5); // removed will be "Hello, !"
```

7. Insert:

- **Purpose:** Inserts a specified string into the current string at a specified index.
- **Syntax:** string.Insert(int startIndex, string value)
- **Returns:** The modified string.
- **Example**

C#

```
string str = "Hello, World!";
string inserted = str.Insert(7, "Beautiful "); // inserted will be
"Hello, Beautiful World!"
```

8. ToCharArray:

- **Purpose:** Converts the current string into a character array.
- **Syntax:** string.ToCharArray()
- **Returns:** The character array.
- **Example:**

C#

```
string str = "Hello, World!";
char[] chars
```

9. ToCharArray:

- **Purpose:** Converts the current string into a character array.
- **Syntax:** string.ToCharArray()
- **Returns:** The character array.
- **Example:**

```
string str = "Hello, World!";
char[] chars = str.ToCharArray(); // chars will be ['H', 'e', 'l',
'l', 'o', ',', ' ', 'W', 'o', 'r', 'l', 'd', '!']
```

10. Intern:

- **Purpose:** Retrieves the string from the string pool if it already exists, otherwise adds it to the pool and returns a reference to it.
- **Syntax:** string.Intern(string str)
- **Returns:** The interned string.
- **Example:**

C#

```
string str1 = "Hello";
string str2 = string.Intern(str1);
// str1 and str2 will refer to the same object in the string pool
```

11. Copy:

- **Purpose:** Copies the current string to a new character array.
- **Syntax:** string.CopyTo(int sourceIndex, char[] destination, int destinationIndex, int count)
- **Returns:** None.
- **Example:**

C#

```
string str = "Hello, World!";
char[] chars = new char[13];
str.CopyTo(0, chars, 0, 13); // chars will be ['H', 'e', 'l', 'l',
'o', ',', ' ', 'W', 'o', 'r', 'l', 'd', '!']
```

12. Normalize:

- **Purpose:** Normalizes the current string according to the specified Unicode normalization form.
- **Syntax:** string.Normalize(NormalizationForm form)
- **Returns:** The normalized string.
- **Example:**

C#

```
string str = "café";
string normalized = str.Normalize(NormalizationForm.FormC);
// normalized will be "cafe" (normalized form)
```

13. IsNormalized:

- **Purpose:** Checks if the current string is normalized according to the specified Unicode normalization form.
- **Syntax:** string.IsNormalized(NormalizationForm form)
- **Returns:** True if the string is normalized, otherwise false.
- **Example:**

C#

```
string str = "café";
bool isNormalized = str.IsNormalized(NormalizationForm.FormC);
// isNormalized will be false
```

14. IsNullOrEmpty:

- **Purpose:** Checks if the specified string is null or empty.
- **Syntax:** string.IsNullOrEmpty(string str)
- **Returns:** True if the string is null or empty, otherwise false.
- **Example:**

C#

```
string str = null;
bool isNullOrEmpty = string.IsNullOrEmpty(str);
// isNullOrEmpty will be true
```

15. IsNullOrWhiteSpace:

- **Purpose:** Checks if the specified string is null, empty, or consists only of white space characters.
- **Syntax:** string.IsNullOrWhiteSpace(string str)
- **Returns:** True if the string is null, empty, or consists only of white space characters, otherwise false.
- **Example:**

C#

```
string str = "   ";
bool isNullOrWhiteSpace = string.IsNullOrWhiteSpace(str);
// isNullOrWhiteSpace will be true
```

ARRAY

Functions and Methods:

Indexer

- **Purpose:** Access or modify elements of an array by their index.
- **Syntax:** array[index]
- **Example:**
  C#
```
int[] numbers = { 1, 2, 3, 4, 5 };
int secondElement = numbers[1]; // Accesses the second element (index 1)
numbers[3] = 10; // Modifies the fourth element (index 3)
```

Length Property

- **Purpose:** Returns the total number of elements in an array.
- **Syntax:** array.Length
- **Example:**
  C#
```
int[] numbers = { 1, 2, 3, 4, 5 };
int length = numbers.Length; // length will be 5
```

Rank Property

- **Purpose:** Returns the number of dimensions in an array.
- **Syntax:** array.Rank
- **Example:**
  C#
```
int[] numbers = { 1, 2, 3, 4, 5 }; // One-dimensional array
int rank = numbers.Rank; // rank will be 1
```

GetLength Method

- **Purpose:** Returns the length of a specific dimension in a multidimensional array.
- **Syntax:** array.GetLength(dimension)
- **Example:**
  C#
  ```
  int[,] matrix = { { 1, 2 }, { 3, 4 } }; // Two-dimensional array
  int length = matrix.GetLength(0); // Length of the first dimension
  (rows)
  ```

GetLowerBound and GetUpperBound Methods

- **Purpose:** Returns the lower and upper bounds of a specific dimension in an array.
- **Syntax:** array.GetLowerBound(dimension) and array.GetUpperBound(dimension)
- **Example:**
  C#
  ```
  int[] numbers = { 1, 2, 3, 4, 5 };
  int lowerBound = numbers.GetLowerBound(0); // Lower bound will be 0
  int upperBound = numbers.GetUpperBound(0); // Upper bound will be 4
  ```

Clone Method

- **Purpose:** Creates a shallow copy of an array.
- **Syntax:** array.Clone()
- **Example:**
  C#
  ```
  int[] numbers = { 1, 2, 3, 4, 5 };
  int[] clonedArray = numbers.Clone();
  ```

CopyTo Method

- **Purpose:** Copies a portion of an array to another array.
- **Syntax:** array.CopyTo(Array destination, int destinationIndex, int startIndex, int length)
- **Example:**
  C#
  ```
  int[] source = { 1, 2, 3, 4, 5 };
  int[] destination = new int[3];
  source.CopyTo(destination, 0, 1, 3); // Copies elements 1, 2, and 3
  to destination
  ```

Clear Method

- **Purpose:** Clears all elements of an array to their default values.
- **Syntax:** array.Clear()
- **Example:**
  C#
  ```
  int[] numbers = { 1, 2, 3, 4, 5 };
  numbers.Clear(); // All elements will be set to 0
  ```

Reverse Method

- **Purpose:** Reverses the order of elements in an array.
- **Syntax:** array.Reverse()
- **Example:**
  C#
  ```
  int[] numbers = { 1, 2, 3, 4, 5 };
  numbers.Reverse(); // numbers will be { 5, 4, 3, 2, 1 }
  ```

Sort Method

- **Purpose:** Sorts the elements of an array.
- **Syntax:** array.Sort() or Array.Sort(array, Comparer)
- **Example:**
  C#
  ```
  int[] numbers = { 3, 1, 4, 1, 5, 9, 2, 6, 5 };
  Array.Sort(numbers); // numbers will be sorted in ascending order
  ```

LIST

Common Functions and Methods:

Add Method

- **Purpose:** Adds an element to the end of the list.
- **Syntax:** list.Add(item)
- **Example:**
  C#
```
List<int> numbers = new List<int>();
numbers.Add(1);
numbers.Add(2);
numbers.Add(3);
```

Insert Method

- **Purpose:** Inserts an element at a specified index in the list.
- **Syntax:** list.Insert(index, item)
- **Example:**
  C#
```
List<string> fruits = new List<string>() { "Apple", "Banana",
"Orange" };
fruits.Insert(1, "Mango"); // Inserts "Mango" at index 1
```

Remove Method

- **Purpose:** Removes the first occurrence of a specified element from the list.
- **Syntax:** list.Remove(item)
- **Example:**
  C#
```
List<int> numbers = new List<int>() { 1, 2, 3, 2, 4 };
numbers.Remove(2); // Removes the first occurrence of 2
```

RemoveAt Method

- **Purpose:** Removes the element at a specified index from the list.
- **Syntax:** list.RemoveAt(index)
- **Example:**
  C#
  ```
  List<string> fruits = new List<string>() { "Apple", "Banana", "Orange" };
  fruits.RemoveAt(1); // Removes the element at index 1 (Banana)
  ```

Clear Method

- **Purpose:** Removes all elements from the list.
- **Syntax:** list.Clear()
- **Example:**
  C#
  ```
  List<int> numbers = new List<int>() { 1, 2, 3, 4, 5 };
  numbers.Clear(); // Empties the list
  ```

Contains Method

- **Purpose:** Checks if a specific element exists in the list.
- **Syntax:** list.Contains(item)
- **Example:**
  C#
  ```
  List<string> fruits = new List<string>() { "Apple", "Banana", "Orange" };
  bool contains = fruits.Contains("Mango"); // Contains will be false
  ```

IndexOf Method

- **Purpose:** Returns the index of the first occurrence of a specified element in the list.
- **Syntax:** list.IndexOf(item)
- **Example:**
  C#
  ```
  List<int> numbers = new List<int>() { 1, 2, 3, 2, 4 };
  int index = numbers.IndexOf(2); // Index will be 0
  ```

LastIndexOf Method

- **Purpose:** Returns the index of the last occurrence of a specified element in the list.
- **Syntax:** list.LastIndexOf(item)
- **Example:**
  C#
  ```
  List<int> numbers = new List<int>() { 1, 2, 3, 2, 4 };
  int index = numbers.LastIndexOf(2); // Index will be 3
  ```

Exists Method

- **Purpose:** Checks if any element in the list satisfies a specified condition.
- **Syntax:** list.Exists(predicate)
- **Example:**
  C#
  ```
  List<int> numbers = new List<int>() { 1, 2, 3, 4, 5 };
  bool exists = numbers.Exists(x => x > 3); // Exists will be true
  ```

Find Method

- **Purpose:** Finds the first element in the list that satisfies a specified condition.
- **Syntax:** list.Find(predicate)
- **Example:**
  C#
  ```
  List<int> numbers = new List<int>() { 1, 2, 3, 4, 5 };
  int firstEven = numbers.Find(x => x % 2 == 0); // firstEven will be
  2
  ```

FindIndex Method

- **Purpose:** Returns the index of the first element in the list that satisfies a specified condition.
- **Syntax:** list.FindIndex(predicate)
- **Example:**
  C#
  ```
  List<int> numbers = new List<int>() { 1, 2, 3, 4, 5 };
  int index = numbers.FindIndex(x => x > 3); // Index will be 2
  ```

ForEach Method

- **Purpose:** Executes a specified action on each element in the list.
- **Syntax:** list.ForEach(action)
- **Example:**
  C#
  ```
  List<string> fruits = new List<string>() { "Apple", "Banana",
  "Orange" };
  fruits.ForEach(Console.WriteLine); // Prints each fruit to the
  console
  ```

TrueForAll Method

- **Purpose:** Checks if all elements in the list satisfy a specified condition.
- **Syntax:** list.TrueForAll(predicate)
- **Example:**
  C#
  ```
  List<int> numbers = new List<int>() { 1, 2, 3, 4, 5 };
  bool allPositive = numbers.TrueForAll(x => x > 0); // allPositive
  will be true
  ```

GetRange Method

- **Purpose:** Returns a new list containing elements from the specified index to the end of the original list.
- **Syntax:** list.GetRange(startIndex, count)
- **Example:**
  C#
  ```
  List<int> numbers = new List<int>() { 1, 2, 3, 4, 5 };
  List<int> sublist = numbers.GetRange(1, 3); // sublist will contain
  { 2, 3, 4 }
  ```

RemoveAll Method

- **Purpose:** Removes all elements from the list that satisfy a specified condition.
- **Syntax:** list.RemoveAll(predicate)
- **Example:**
  C#
  ```
  List<int> numbers = new List<int>() { 1, 2, 3, 4, 5 };
  numbers.RemoveAll(x => x % 2 == 0); // Removes all even numbers
  ```

Sort Method

- **Purpose:** Sorts the elements of the list.
- **Syntax:** list.Sort() or list.Sort(comparer)
- **Example:**
  C#
  ```
  List<int> numbers = new List<int>() { 3, 1, 4, 1, 5, 9, 2, 6, 5 };
  numbers.Sort(); // Sorts in ascending order
  ```

ConvertAll Method

- **Purpose:** Converts all elements in the list to a new type using a specified converter function.
- **Syntax:** list.ConvertAll(converter)
- **Example:**
  C#
  ```
  List<string> numbers = new List<string>() { "1", "2", "3" };
  List<int> intNumbers = numbers.ConvertAll(int.Parse); // Converts
  strings to integers
  ```

ToArray Method

- **Purpose:** Converts the list to an array.
- **Syntax:** list.ToArray()
- **Example:**
  C#
  ```
  List<int> numbers = new List<int>() { 1, 2, 3, 4, 5 };
  int[] array = numbers.ToArray();
  ```

Dictionaries

Common Functions and Methods:

Add Method

- **Purpose:** Adds a new key-value pair to the dictionary.
- **Syntax:** dictionary.Add(key, value)
- **Example:**
  C#
```
Dictionary<string, int> ages = new Dictionary<string, int>();
ages.Add("Alice", 30);
ages.Add("Bob", 25);
```

ContainsKey Method

- **Purpose:** Checks if a specific key exists in the dictionary.
- **Syntax:** dictionary.ContainsKey(key)
- **Example:**
  C#
```
Dictionary<string, int> ages = new Dictionary<string, int>();
bool contains = ages.ContainsKey("Alice"); // Contains will be true
```

ContainsValue Method

- **Purpose:** Checks if a specific value exists in the dictionary.
- **Syntax:** dictionary.ContainsValue(value)
- **Example:**
  C#
```
Dictionary<string, int> ages = new Dictionary<string, int>();
bool contains = ages.ContainsValue(30); // Contains will be true
```

Remove Method

- **Purpose:** Removes a key-value pair from the dictionary.
- **Syntax:** dictionary.Remove(key)
- **Example:**
  C#
  ```
  Dictionary<string, int> ages = new Dictionary<string, int>();
  ages.Remove("Alice"); // Removes the key-value pair for "Alice"
  ```

Clear Method

- **Purpose:** Removes all key-value pairs from the dictionary.
- **Syntax:** dictionary.Clear()
- **Example:**
  C#
  ```
  Dictionary<string, int> ages = new Dictionary<string, int>();
  ages.Clear(); // Empties the dictionary
  ```

Count Property

- **Purpose:** Returns the number of key-value pairs in the dictionary.
- **Syntax:** dictionary.Count
- **Example:**
  C#
  ```
  Dictionary<string, int> ages = new Dictionary<string, int>();
  int count = ages.Count;
  ```

Keys Property

- **Purpose:** Returns a collection of keys in the dictionary.
- **Syntax:** dictionary.Keys
- **Example:**
  C#
  ```
  Dictionary<string, int> ages = new Dictionary<string, int>();
  foreach (string key in ages.Keys)
  {
      Console.WriteLine(key);
  }
  ```

Values Property

- **Purpose:** Returns a collection of values in the dictionary.
- **Syntax:** dictionary.Values
- **Example:**
  C#
  ```
  Dictionary<string, int> ages = new Dictionary<string, int>();
  foreach (int value in ages.Values)
  {
      Console.WriteLine(value);
  }
  ```

TryGetValue Method

- **Purpose:** Attempts to retrieve the value associated with a specified key.
- **Syntax:** dictionary.TryGetValue(key, out value)
- **Example:**
  C#
  ```
  Dictionary<string, int> ages = new Dictionary<string, int>();
  int age;
  if (ages.TryGetValue("Alice", out age))
  {
      Console.WriteLine("Age: " + age);
  }
  ```

ToDictionary Method

- **Purpose:** Converts the dictionary to a new dictionary of a different type.
- **Syntax:** dictionary.ToDictionary(keySelector, elementSelector)
- **Example:**
  C#
  ```
  Dictionary<string, int> numbers = new Dictionary<string, int>();
  numbers.Add("one", 1);
  numbers.Add("two", 2);

  Dictionary<int, string> reversed = numbers.ToDictionary(kvp =>
  kvp.Value, kvp => kvp.Key);
  ```

TryAdd Method

- **Purpose:** Attempts to add a new key-value pair to the dictionary.
- **Syntax:** dictionary.TryAdd(key, value)
- **Returns:** True if the key-value pair was added, false if the key already exists.
- **Example:**
  C#
  ```
  Dictionary<string, int> ages = new Dictionary<string, int>();
  bool added = ages.TryAdd("Alice", 30);
  ```

GetOrAdd Method

- **Purpose:** Attempts to retrieve the value associated with a specified key. If the key doesn't exist, it adds a new key-value pair using a specified function.
- **Syntax:** dictionary.GetOrAdd(key, valueFactory)
- **Example:**
  C#
  ```
  Dictionary<string, int> ages = new Dictionary<string, int>();
  int age = ages.GetOrAdd("Alice", () => 30);
  ```

RemoveAll Method

- **Purpose:** Removes all key-value pairs from the dictionary that satisfy a specified condition.
- **Syntax:** dictionary.RemoveAll(predicate)
- **Example:**
  C#
  ```
  Dictionary<string, int> ages = new Dictionary<string, int>();
  ages.RemoveAll(kvp => kvp.Value > 30);
  ```