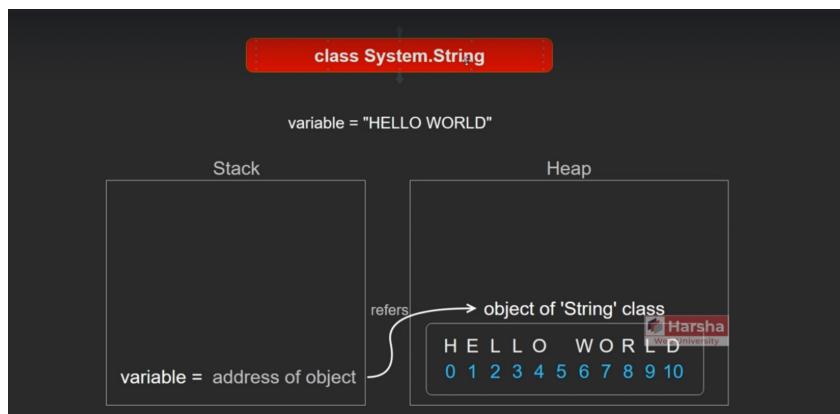


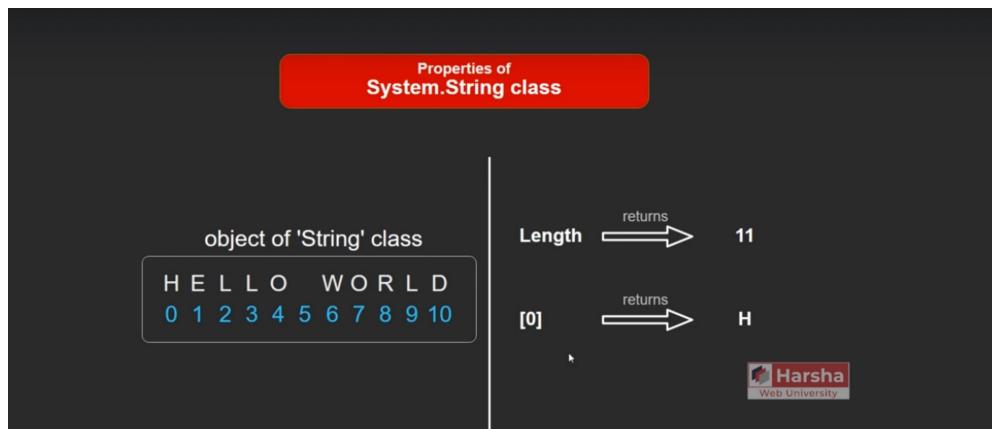
# String

In C sharp all the strings by default treated as objects of a predefined class called `System.String`



`System.String == string`

both are equal internally.



**String**

What	<ul style="list-style-type: none"> <li>The System.String is a class that represents an array of Unicode characters.</li> <li>String stores a set of characters as char[].</li> <li>Max no. of characters in the string: 2 billion.</li> <li>String is immutable. That means, it can't be modified.</li> </ul>
How	<pre>String string referenceVariable = "Hello123";</pre>

**String**

#	Property
1	int <b>Length</b> { get; }
	This property returns the no. of characters of the string.
2	char [ <b>int index</b> ] { get; }
	This indexer returns the single character at the specified character index.

**Methods of System.String class**

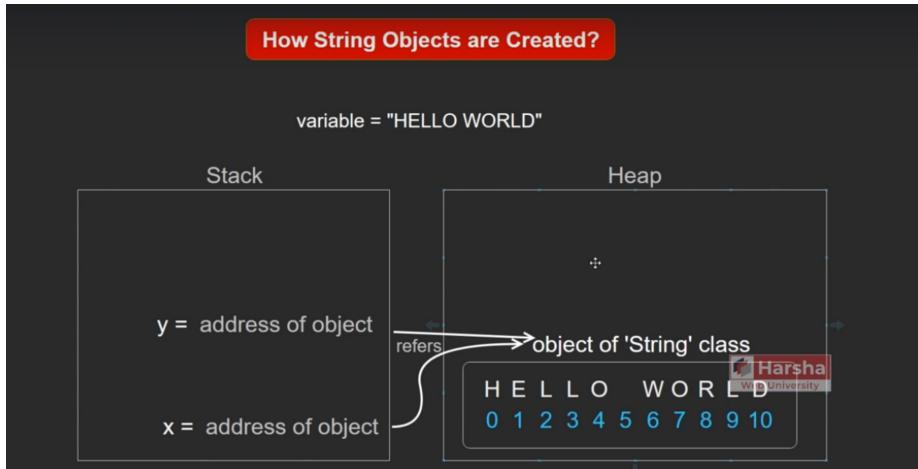
ToUpper()	Split()	StartsWith()
ToLower()	Trim()	EndsWith()
Substring()	ToCharArray()	Contains()
Replace()	Equals()	IndexOf()
Format()	Join()	LastIndexOf()
IsNullOrEmpty()	CompareTo()	IsNullOrWhiteSpace()

## How String Objects are Created

```
using System;
namespace StringObjectsExample
{
    class Program
    {
        static void Main()
        {
            string x = "HELLO WORLD";
            string y = "HELLO WORLD";
        }
    }
}
```

The CLR of dot net, that is Common Language Runtime will automatically recognize that, there is already an existing string object with the same value.

Instead of creating a new object, it will just create a reference variable and point to the same object to the memory.

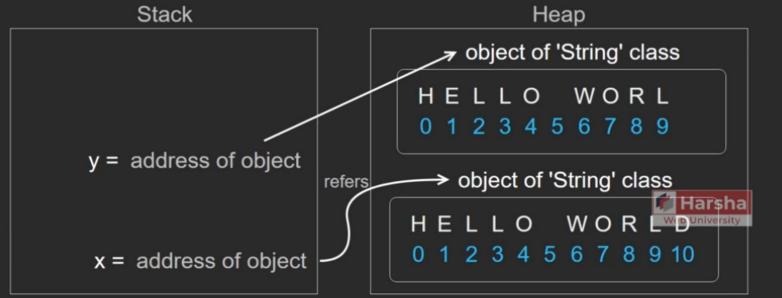


This is the internal strategy automatically done by CLR itself. This is basically meant for avoiding the memory wastage.

```
using System;
namespace StringObjectsExample
{
    class Program
    {
        static void Main()
        {
            string x = "HELLO WORLD";
            string y = "HELLO WORL";
        }
    }
}
```

### How String Objects are Created?

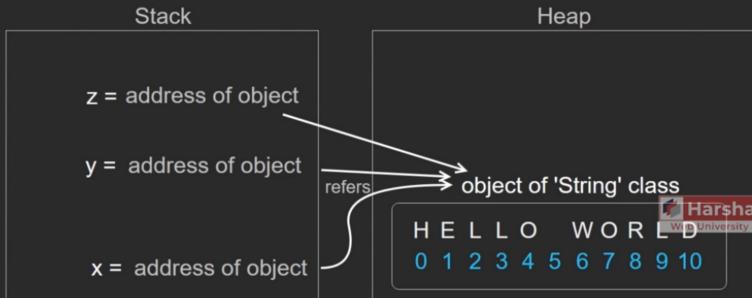
```
variable = "HELLO WORLD"
```



```
2
3  namespace StringObjectsExample
4  {
5      class Program
6      {
7          static void Main()
8          {
9              string x = "HELLO WORLD";
10             string y = "HELLO WORLD";
11             string z = x;
12         }
13     }
14 }
```

### How String Objects are Created?

```
variable = "HELLO WORLD"
```

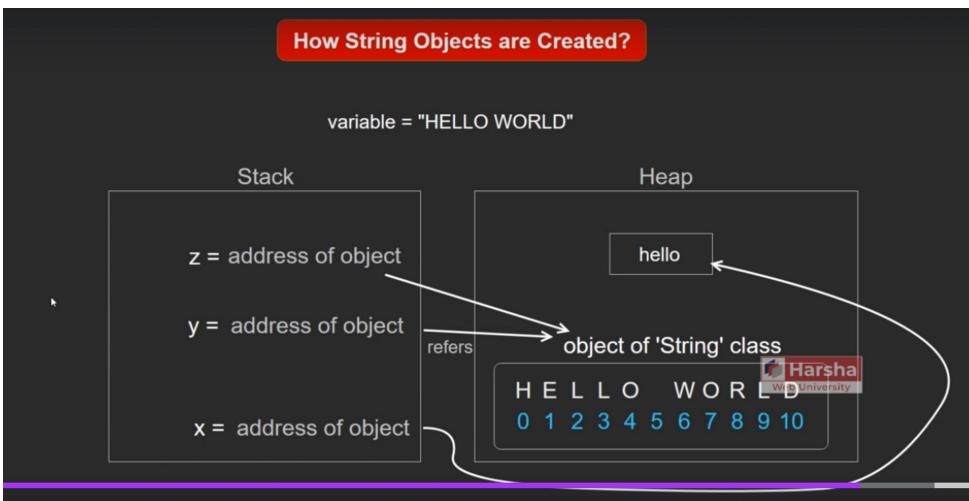


```

namespace StringObjectsExample
{
    class Program
    {
        static void Main()
        {
            string x = "HELLO WORLD";
            string y = "HELLO WORLD";
            string z = x;
            x = "hello";
        }
    }
}

```

this right hand side part of the code



```

1  using System;
2
3  namespace StringObjectsExample
4  {
5      class Program
6      {
7          static void Main()
8          {
9              string x = "HELLO WORLD";
10             string y = "HELLO WORLD";
11             string z = x;
12             x = "hello";
13             Console.WriteLine(x); //hello
14             Console.ReadKey();
15         }
16     }
17 }

```

Output window showing the result of the console application:

```

hello

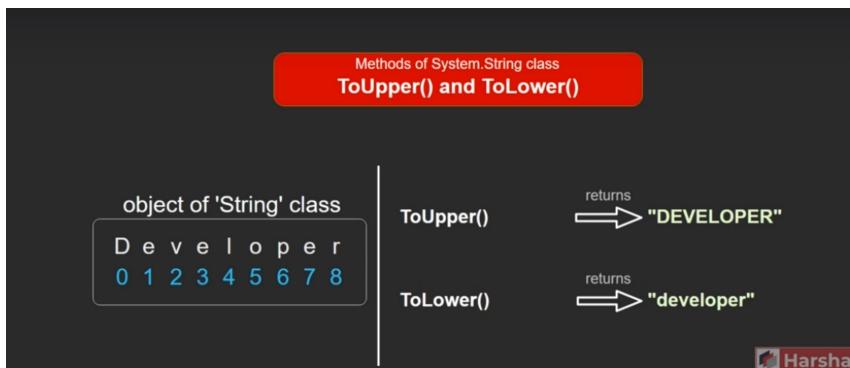
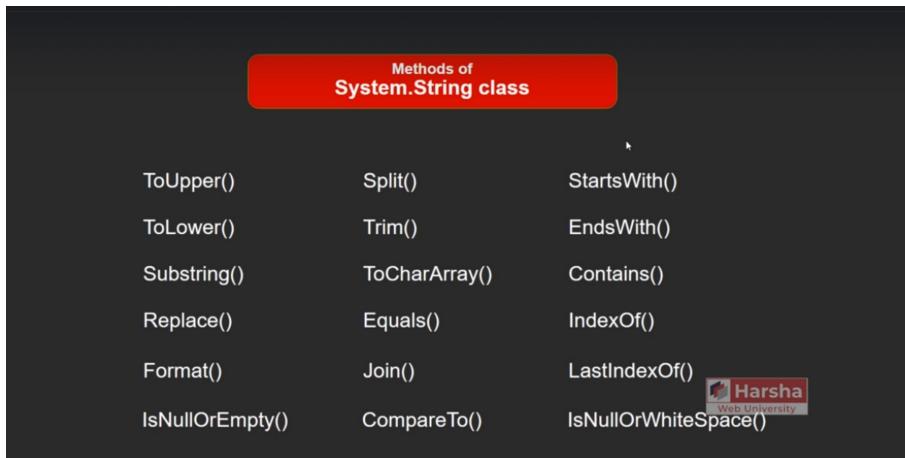
```

```
4  {
5  }
6  class Program
7  {
8  }
9  static void Main()
10 {
11     string x = "HELLO WORLD";
12     string y = "HELLO WORLD";
13     string z = x;
14     x = "hello";
15     Console.WriteLine(x); //hello
16     Console.WriteLine(y); //HELLO WORLD
17     Console.WriteLine(z); //HELLO WORLD
18     Console.ReadKey();
19 }
20 }
```

it is hello world

# Converting Strings

## Part 1

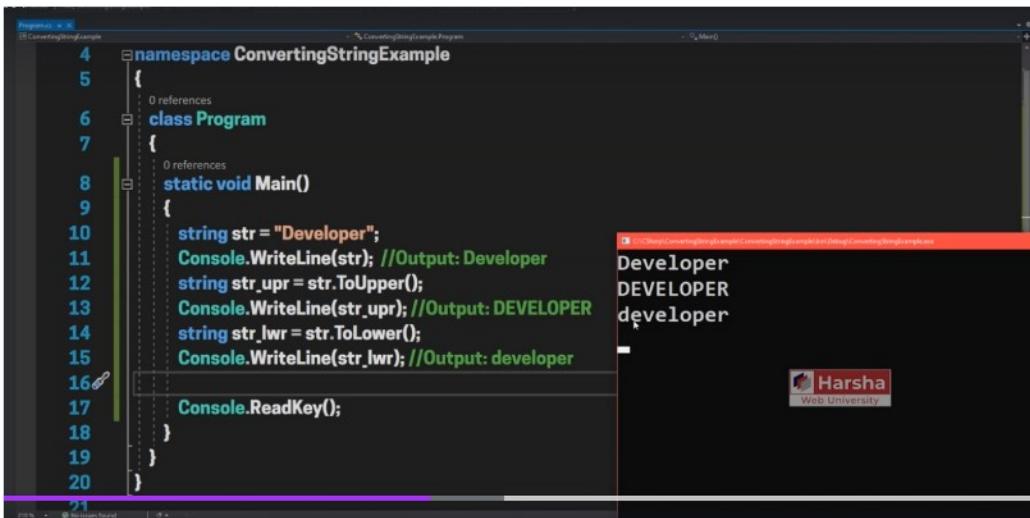


```

namespace ConvertingStringExample
{
    class Program
    {
        static void Main()
        {
            string str = "Developer";
            Console.WriteLine(str); //Output: Developer
            string str_upr = str.ToUpper();
            Console.WriteLine(str_upr);
            Console.ReadKey();
        }
    }
}

```

Internally, creates a new object and send the reference of it.

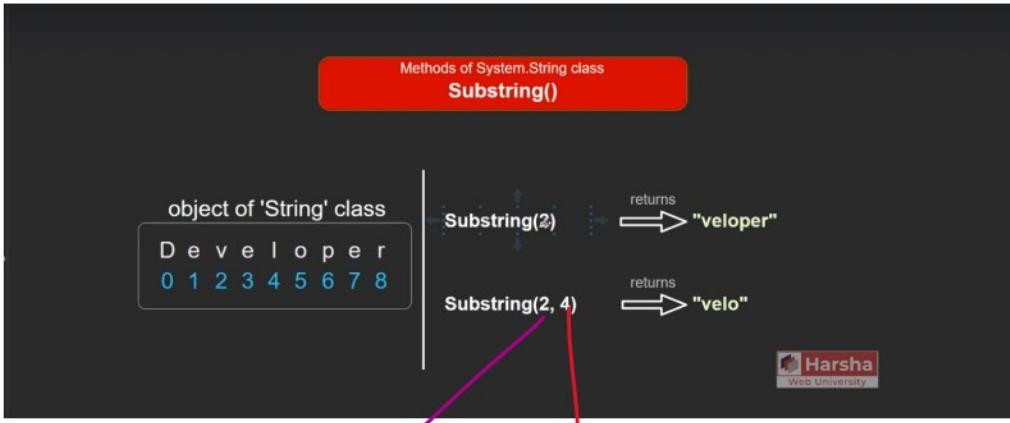


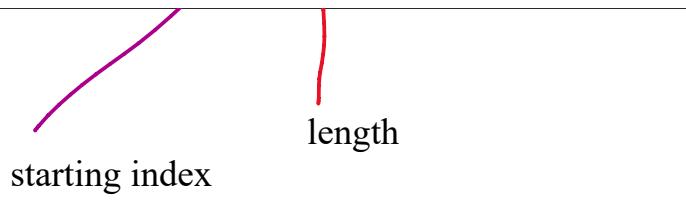
```

namespace ConvertingStringExample
{
    class Program
    {
        static void Main()
        {
            string str = "Developer";
            Console.WriteLine(str); //Output: Developer
            string str_upr = str.ToUpper();
            Console.WriteLine(str_upr); //Output: DEVELOPER
            string str_lwr = str.ToLower();
            Console.WriteLine(str_lwr); //Output: developer

            Console.ReadKey();
        }
    }
}

```





```

{
    string str = "Developer";
    Console.WriteLine(str); //Output: Developer
    string str_upr = str.ToUpper();
    Console.WriteLine("Upper: " + str_upr); //Output: DEVELOPER
    string str_lwr = str.ToLower();
    Console.WriteLine("Lower: " + str_lwr); //Output: developer

    string str_sub = str.Substring(4);
    Console.WriteLine("Substring at 4: " + str_sub);
    string str_sub2 = str.Substring(4, 3);
    Console.WriteLine("Substring at 4,3: " + str_sub2);

    Console.ReadKey();
}

```

The output window shows the following results:

```

Developer
Upper: DEVELOPER
Lower: developer
Substring at 4: loper
Substring at 4,3: ope

```

```

ConvertingStringExample.cs
  ↵ 9   {
  ↵ 10  string str = "Developer";
  ↵ 11  Console.WriteLine(str); //Output: Developer
  ↵ 12  string str_upr = str.ToUpper();
  ↵ 13  Console.WriteLine("Upper: " + str_upr); //Output: DEVELOPER
  ↵ 14  string str_lwr = str.ToLower();
  ↵ 15  Console.WriteLine("Lower: " + str_lwr); //Output: developer
  ↵ 16
  ↵ 17  string str_sub = str.Substring(4);
  ↵ 18  Console.WriteLine("Substring at 4: " + str_sub);
  ↵ 19  string str_sub2 = str.Substring(4, 3);
  ↵ 20  Console.WriteLine("Substring at 4,3: " + str_sub2);
  ↵ 21
  ↵ 22  Console.ReadKey();
  ↵ 23
  ↵ 24
  ↵ 25
  ↵ 26

```

The output window shows the following results:

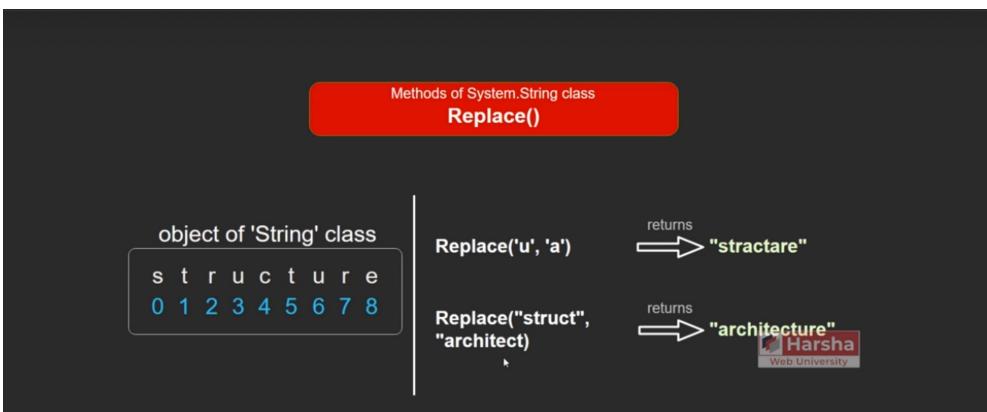
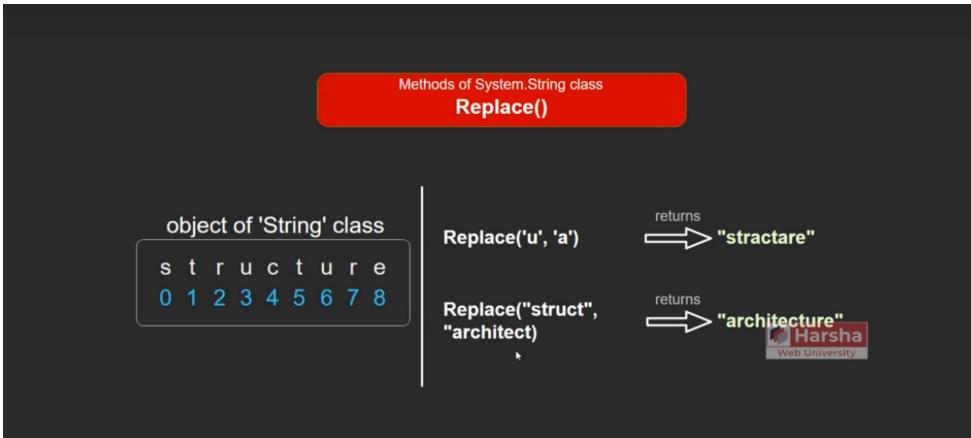
```

Harsha
Web University

```

## Converting Strings

### Part 2



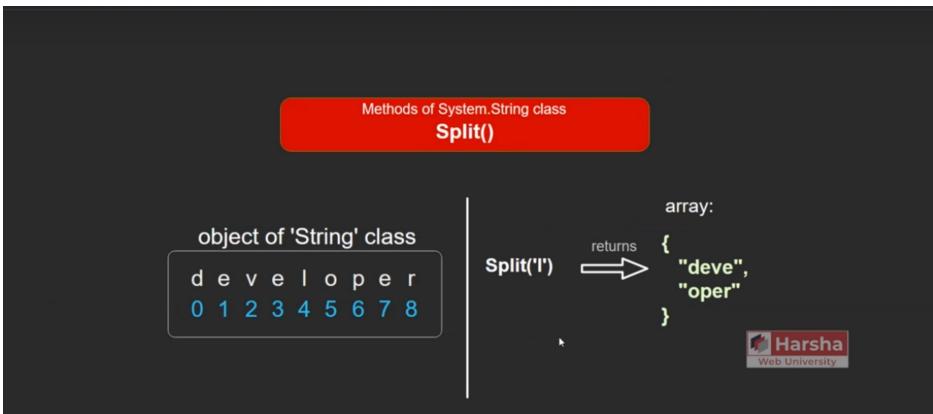
```
string str_upr = str.ToUpper();
Console.WriteLine("Upper: " + str_upr); //Output: DEVELOPER
string str_lwr = str.ToLower();
Console.WriteLine("Lower: " + str_lwr); //Output: developer

string str_sub = str.Substring(4);
Console.WriteLine("Substring at 4: " + str_sub); //Output: loper
string str_sub2 = str.Substring(4, 3);
Console.WriteLine("Substring at 4,3: " + str_sub2); //Output: lop

string str_rpl = str.Replace("e", "x");
Console.WriteLine("Replace: " + str_rpl);

Console.ReadKey();
}
```

Developer  
Upper: DEVELOPER  
Lower: developer  
Substring at 4: loper  
Substring at 4,3: lop  
Replace: Dxvxlopxr

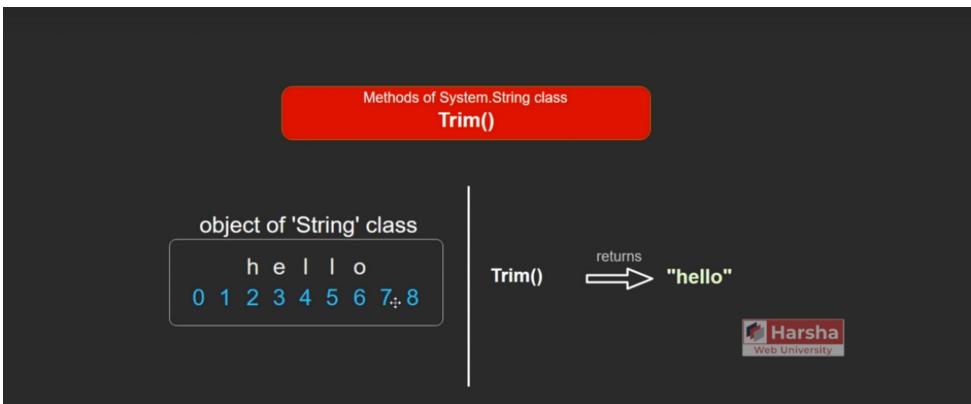


```
string message = "how are you";
string[] words = message.Split(' ');
Console.WriteLine("\nSplit:");
foreach(string word in words)
{
    Console.WriteLine(word);
}
Console.ReadKey();
```

Developer  
Upper: DEVELOPER  
Lower: developer  
Substring at 4: loper  
Substring at 4,3: lop  
Replace: Davalopar

Split:  
how  
are  
you

Harsha  
Web University

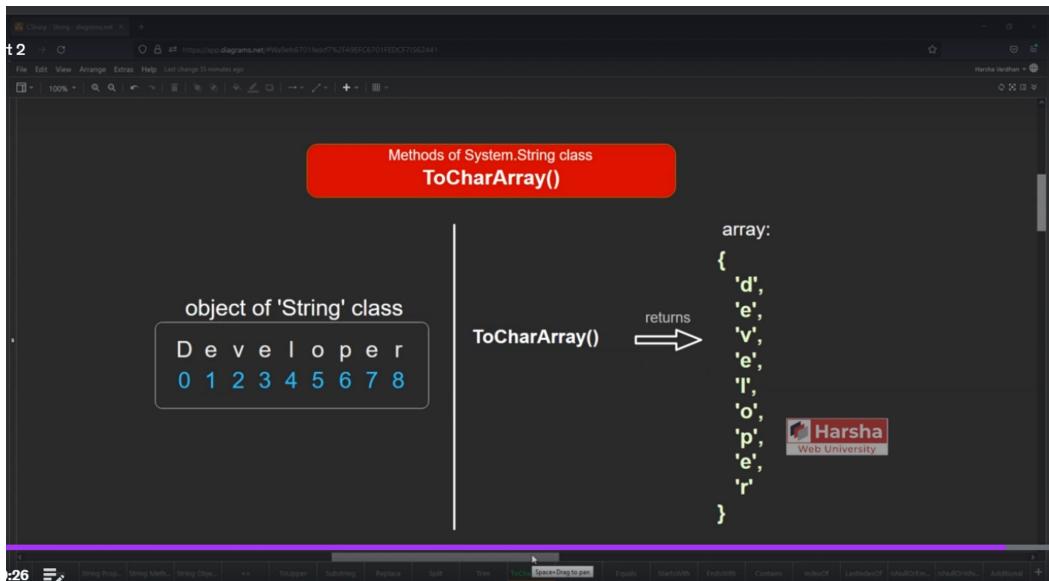


Trim() method removes unnecessary spaces from left and right side.  
**but not in middle.**

```
string message_with_spaces = " hello ";
string str_trm = message_with_spaces.Trim();
Console.WriteLine("Trim: " + str_trm);

Console.ReadKey();
```

I



```
Console.WriteLine(word); //Output: how, are, you

string message_with_spaces = " hello ";
string str_trm = message_with_spaces.Trim();
Console.WriteLine("Trim: " + str_trm); //Output

char[] characters = str.ToCharArray();
Console.WriteLine("\nCharacters:");
foreach(char ch in characters)
{
    Console.WriteLine(ch);
}
Console.ReadKey();
```

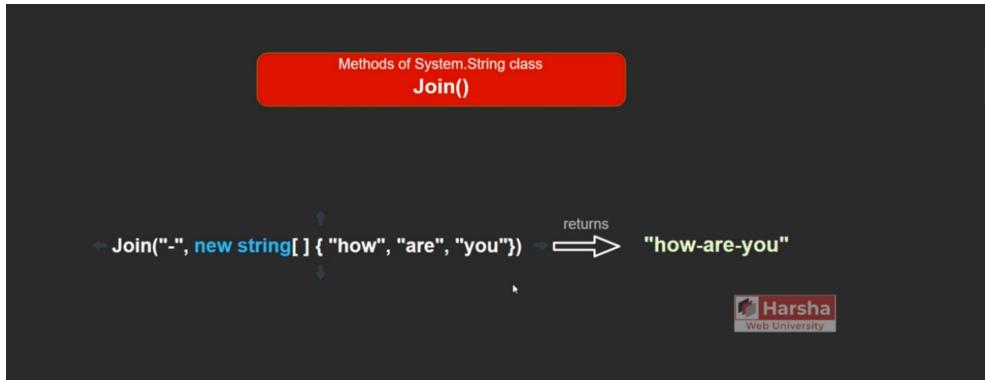
Trim: hello

Characters:

```
D
e
v
e
l
o
p
e
r
```

## Converting Strings

### Part 3



```

string[] my_words = new string[] { "how", "are", "you" }; //how-are-you
string str_jn = string.Join("~-", my_words);
Console.WriteLine("Join: " + str_jn);

char[] characters2 = new char[] { 'h', 'e', 'l', 'l', 'o' };
string str2 = new string(characters2);
earlier
Console.WriteLine("New string: " + str2);

```

## String

**Converting Strings**

#	Method
1	string <b>ToUpper( )</b> Returns the same string in upper case.
2	string <b>ToLower( )</b> Returns the same string lower case.
3	string <b>Substring(int startIndex, int length)</b> Returns a string with a set of characters starting from the "startIndex" up to the "length" no. of characters. The "length" parameter is optional.
4	string <b>Replace( string oldString, string newString )</b> Returns a string after replacing all occurrences of <b>oldString</b> with <b>newString</b> .
5	string[] <b>Split( char separator )</b> Returns a string[] after splitting the current string into an array of characters. At each occurrence of separator character, a new string is formed-up. At last, all the pieces or strings are converted as a string[].

## String

**Converting Strings**

#	Method
6	string <b>Trim()</b> Returns the same string after removing extra spaces at beginning and ending of the current string. It returns the same string if no spaces found at L.H.S. and R.H.S. of the current string.
7	string <b>ToCharArray()</b> Returns the same string as an array of characters (char[]).
8	string <b>Join(string separator, IEnumerable&lt;T&gt; values)</b> Returns the a string with joined values with separator in between each value.

**'String' class**

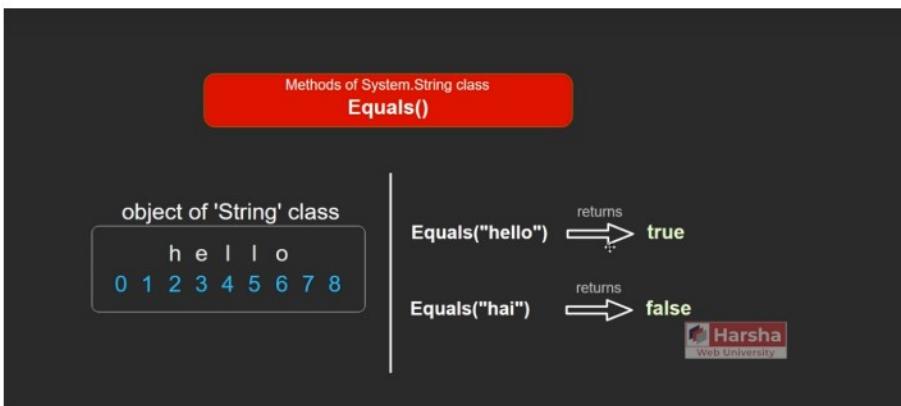
**Constructor**

Constructor	Description
<b>String( char[] )</b>	It initializes the string with specified char[].

## Checking Strings

### Part 1

Methods of System.String class		
ToUpper()	Split()	StartsWith()
ToLower()	Trim()	EndsWith()
Substring()	ToCharArray()	Contains()
Replace()	Equals()	IndexOf()
Format()	Join()	LastIndexOf()
IsNullOrEmpty()	CompareTo()	IsNullOrWhiteSpace()



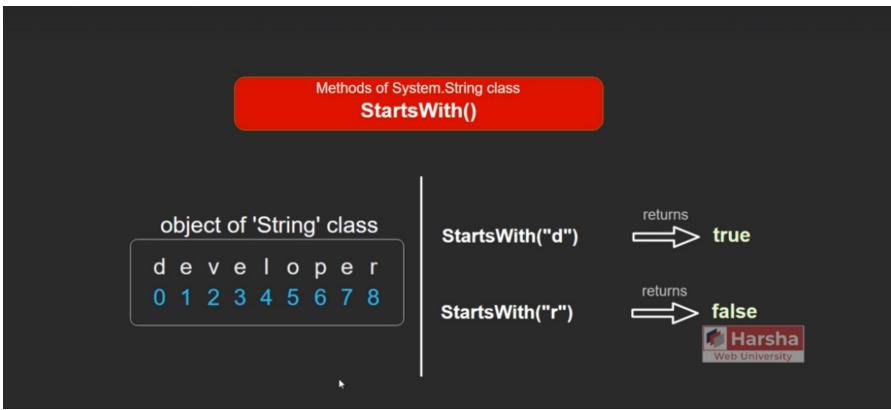
```

4  namespace CheckingStringsExample
5  {
6      class Program
7      {
8          static void Main()
9          {
10             string str = "Universe";
11             string str2 = "Universe";
12
13             bool eq = str.Equals(str2);
14             bool eq2 = str == str2;
15
16             Console.WriteLine("Equals: " + eq);
17             Console.WriteLine("==: " + eq2);
18
19             Console.ReadKey();
20         }
}

```

In C#, Equal() and ==

works in a same way for string.



```

    string str = "Universe";
    string str2 = "Universe";

    bool eq = str.Equals(str2);
    bool eq2 = str == str2;

    Console.WriteLine("Equals: " + eq);
    Console.WriteLine("==: " + eq2);

    bool sw = str.StartsWith("U");
    Console.WriteLine("StartsWith: " + sw);

    Console.ReadKey();
}

```

Output window:

```

 Equals: True
 ==: True
 StartsWith: True

```

```

9   {
10    string str = "Universe";
11    string str2 = "Universe";
12
13    bool eq = str.Equals(str2);
14    bool eq2 = str == str2;
15
16    Console.WriteLine("Equals: " + eq);
17    Console.WriteLine("==: " + eq2);
18
19    bool sw = str.StartsWith("U");
20    Console.WriteLine("StartsWith: " + sw);
21    bool sw2 = str.StartsWith("e");
22    Console.WriteLine("StartsWith: " + sw);
23
24    Console.ReadKey();
25
26 }

```

Output window:

```

 Equals: True
 ==: True
 StartsWith: True

```

Methods of System.String class  
**EndsWith()**

object of 'String' class  
d e v e l o p e r  
0 1 2 3 4 5 6 7 8

EndsWith("r")  
returns → true  
EndsWith("d")  
returns → false



```
bool ew = str.EndsWith("e");
bool ew2 = str.EndsWith("U");
Console.WriteLine("Ends with: " + ew); //Output: true
Console.WriteLine("Ends with: " + ew); //Output: false

Console.ReadKey();
```



Methods of System.String class  
**Contains()**

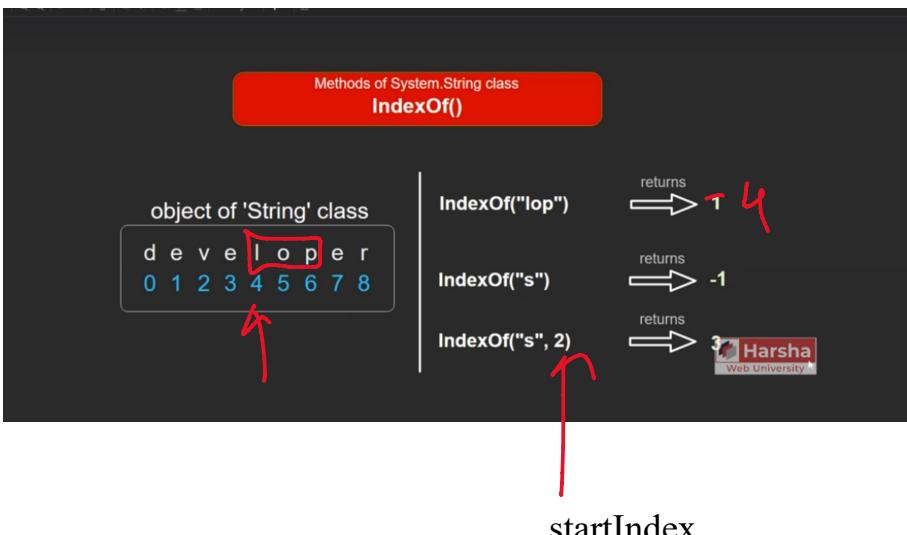
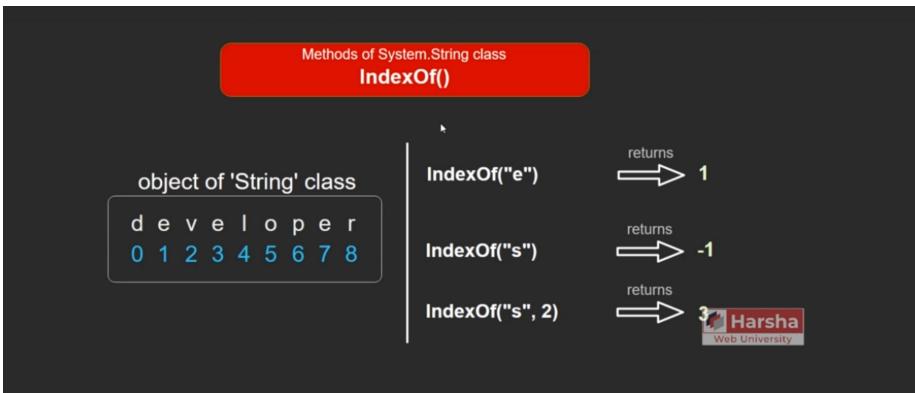
object of 'String' class  
d e v e l o p e r  
0 1 2 3 4 5 6 7 8

Contains("e")  
returns → true  
Contains("s")  
returns → false



```
bool ct = str.Contains("e");
bool ct2 = str.Contains("t");
Console.WriteLine("Contains: " + ct); //Output: true
Console.WriteLine("Contains: " + ct2); //Output: false

Console.ReadKey();
```



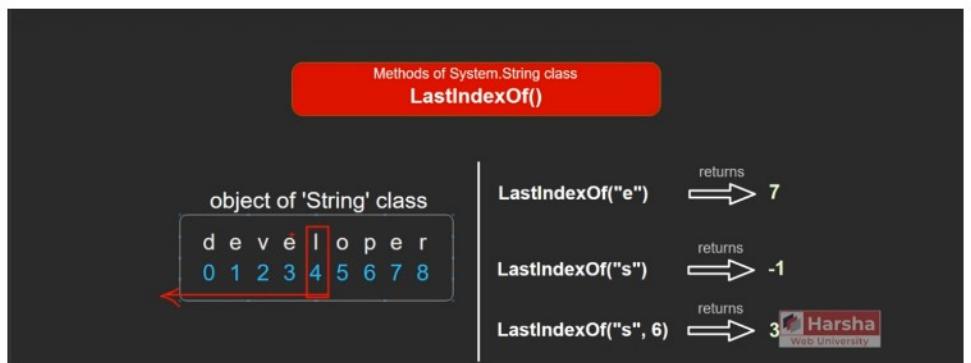
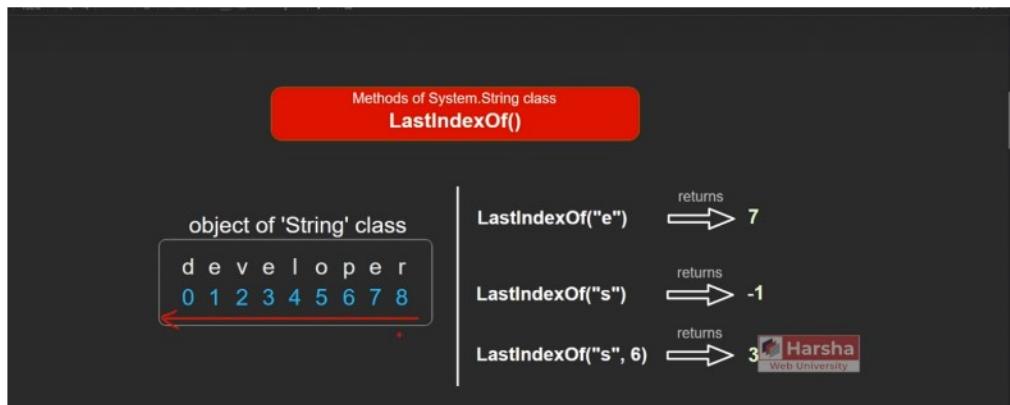
```
string name = "developer";
int ind1 = name.IndexOf("e"); //1
int ind2 = name.IndexOf("vel"); //2
int ind3 = name.IndexOf("gel"); //3
int ind4 = name.IndexOf("e", 2); //prev start index + 1 //4
int ind5 = name.IndexOf("e", 4); //prev start index + 1 //5
Console.WriteLine("Index Of e: " + ind1);
Console.WriteLine("Index Of vel: " + ind2);
Console.WriteLine("Index Of gel: " + ind3);
Console.WriteLine("Index Of e,2: " + ind4);
Console.WriteLine("Index Of ind5: " + ind5);

Console.ReadKey();
}
```

==: True  
StartsWith: True  
StartsWith: False  
Ends with: True  
Ends with: False  
Contains: True  
Contains: False  
Index Of e: 1  
Index Of vel: 2  
Index Of gel: -1  
Index Of Harsha  
Index Of Web University  
Index Of ind5: 7

# Checking Strings

## Part 2



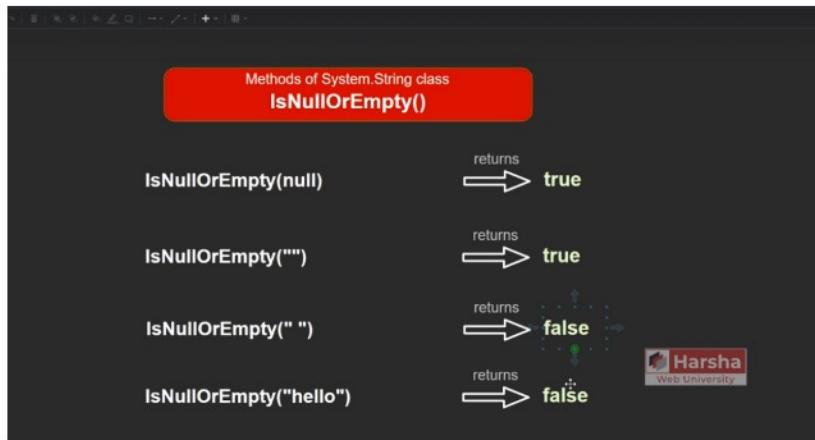
A screenshot of a C# code editor showing a program named `CheckingStringProgram.cs`. The code uses the `LastIndexOf()` method to find the last occurrence of characters 'e', 'vel', and 'gel' in the string "developer". The output window shows the results of each call.

```
int ind4 = name.IndexOf("e", 2); //prev start index +1 //3
int ind5 = name.IndexOf("e", 4); //prev start index +1 //7
Console.WriteLine("Index Of e: " + ind1);
Console.WriteLine("Index Of vel: " + ind2);
Console.WriteLine("Index Of gel: " + ind3);
Console.WriteLine("Index Of e,2: " + ind4);
Console.WriteLine("Index Of ind5: " + ind5);

int ind6 = name.LastIndexOf("e"); //
int ind7 = name.LastIndexOf("vel"); //
int ind8 = name.LastIndexOf("gel"); //
int ind9 = name.LastIndexOf("e", 2); //prev start index +1 //
int ind10 = name.LastIndexOf("e", 4); //prev start index +1 //
Console.WriteLine("LastIndexOf Of e: " + ind6);
Console.WriteLine("LastIndexOf Of vel: " + ind7);
Console.WriteLine("LastIndexOf Of gel: " + ind8);
Console.WriteLine("LastIndexOf Of e,2: " + ind9);
Console.WriteLine("LastIndexOf Of ind5: " + ind10);
```

The output window displays the following results:

- Contains: True
- Contains: False
- Index Of e: 1
- Index Of vel: 2
- Index Of gel: -1
- Index Of e,2: 3
- Index Of ind5: 7
- LastIndexOf Of e: 7
- LastIndexOf Of vel: 2
- LastIndexOf Of gel: -1
- LastIndexOf Of e,2: 1
- LastIndexOf Of ind5: 3



```

53 Console.WriteLine("LastIndexOf Of gel: " + ind8);
54 Console.WriteLine("LastIndexOf Of e,2: " + ind9);
55 Console.WriteLine("LastIndexOf Of ind5: " + ind10);
56
57 string user_input = null;
58 string user_input2 = "";
59 string user_input3 = " ";
60 bool noe1 = string.IsNullOrEmpty(user_input);
61 bool noe2 = string.IsNullOrEmpty(user_input2);
62 bool noe3 = string.IsNullOrEmpty(user_input3);
63
64 Console.WriteLine("IsNullOrEmpty - null: " + noe1);
65 Console.WriteLine("IsNullOrEmpty - \"\": " + noe2);
66 Console.WriteLine("IsNullOrEmpty - \" \"": " + noe3);
67 Console.ReadKey();
68
69 //don't store the value into property
70
71
72
73

```

Index Of vel: 2  
Index Of gel: -1  
Index Of e,2: 3  
Index Of ind5: 7  
LastIndexOf Of e: 7  
LastIndexOf Of vel: 2  
LastIndexOf Of gel: -1  
LastIndexOf Of e,2: 1  
LastIndexOf Of ind5: 3  
IsNullOrEmpty - null: True  
IsNullOrEmpty - "": True  
IsNullOrEmpty - " ":" False

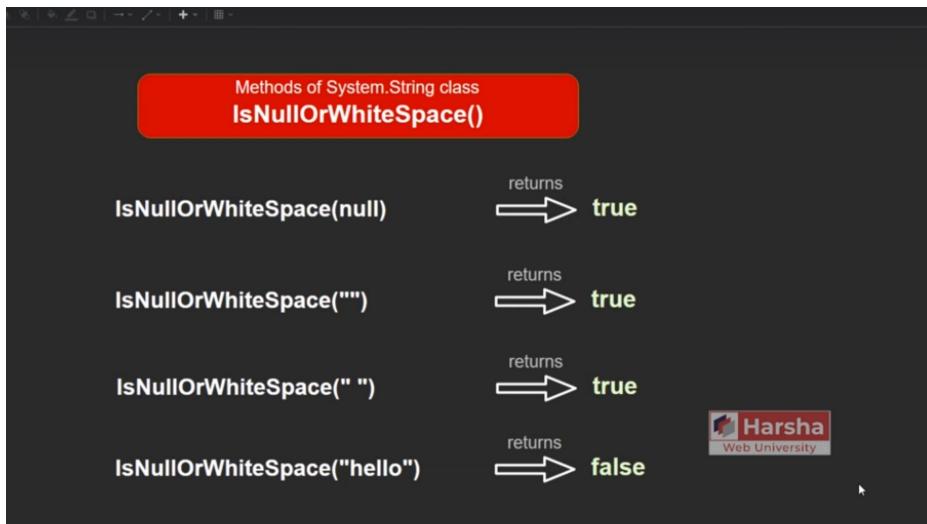
```

56
57 string user_input = null;
58 string user_input2 = "";
59 string user_input3 = " ";
60 bool noe1 = string.IsNullOrEmpty(user_input);
61 bool noe2 = string.IsNullOrEmpty(user_input2);
62 bool noe3 = string.IsNullOrEmpty(user_input3);
63
64 Console.WriteLine("IsNullOrEmpty - null: " + noe1);
65 Console.WriteLine("IsNullOrEmpty - \"\": " + noe2);
66 Console.WriteLine("IsNullOrEmpty - \" \"": " + noe3);
67 if (string.IsNullOrEmpty(user_input))
68 {
69     //don't store the value into property
70 }
71
72
73

```

Harsha  
Web University

Practice: Create a Customer class with a customerName property along with this type of condition in the set accessor.



```

    bool now1 = string.IsNullOrWhitespace(user_input);
    bool now2 = string.IsNullOrWhitespace(user_input2);
    bool now3 = string.IsNullOrWhitespace(user_input3);

    Console.WriteLine(".IsNullOrWhitespace - null: " + now1);
    Console.WriteLine(".IsNullOrWhitespace - \"\"": " + now2);
    Console.WriteLine(".IsNullOrWhitespace - \" \": " + now3);

    Console.ReadKey();
}

```

so you can use either of these methods

String	
Checking Strings	# Method
	9 <b>bool Equals( string otherString )</b> Returns a Boolean value that indicates whether the current string and the specified otherString are equal or not (each character will be compared).
	10 <b>bool StartsWith( string otherString )</b> Returns a Boolean value that indicates whether the current string begins with the specified otherString.
	11 <b>bool EndsWith( string otherString )</b> Returns a Boolean value that indicates whether the current string ends with the specified otherString.
	12 <b>bool Contains( string otherString )</b> Returns a Boolean value that indicates whether the current string contains the specified otherString anywhere.

## Checking Strings

#	Method
I3	<code>int IndexOf( string otherString, int startIndex )</code>
	Returns index of the first character of the specified otherString, where the otherString exists in the current string. Searching process starts from the specified startIndex. The startIndex is optional.
	In case if the specified otherString doesn't exist in the current string, the method returns -1.
I4	<code>int LastIndexOf( string otherString, int startIndex)</code>
	It is same as IndexOf(); but searching process takes place from Right-To-Left direction, starting from the startIndex.
I5	<code>static bool IsNullOrEmpty( string value)</code>
	It determines whether the given string value is null or empty ( "" ).
I6	<code>static bool IsNullOrWhiteSpace( string value)</code>
	It determines whether the given string value is null or white space (" ").

## Formatting Strings

Methods of System.String class

**Format()**

```
string.Format("{0} is the director of {1}", director, movie)
              returns
              → Sam Raimi is the director of Spider Man
```

returns  
→ Sam Raimi is the director of Spider Man

Sam Raimi is the director of Spider Man

Harsha Web University

A screenshot of the Visual Studio IDE showing a C# file named `FormattingStringsExample.cs`. The code defines a `Program` class with a `Main` method. It uses `String.Format` and `$`-interpolation to format strings. The output window shows the results of the console application.

```
1 using System;
2
3 namespace FormattingStringsExample
4 {
5     class Program
6     {
7         static void Main()
8         {
9             string director = "Sam Raimi", movie = "Spider man";
10            string message = string.Format("{0} is the director of {1}", director, movie);
11            string message2 = $"{director} is the director of {movie}";
12
13            Console.WriteLine(message);
14            Console.WriteLine(message2);
15            Console.ReadKey();
16        }
17    }
}
```

Output window:

```
Sam Raimi is the director of Spider man
Sam Raimi is the director of Spider man
```

A screenshot of a Java documentation page for the `String` class. The left sidebar has a section titled "Formatting Strings". A table provides details about the `Format` method.

#	Method
I7	static string Format( string format, object arg0, object arg1, ...)

Documentation for `Format`:

Returns the string after substituting the specified argument values at specified placeholders.  
A place holder is represented as `{indexOfArgument}`.

Ex: `string.Format("{0} Oriented {1}", "Object", "Programming")`  
`//Object Oriented Programming`

## Modifying Strings

## Methods of System.String class Insert()

object of 'String' class

```
D e v e l o p e r  
0 1 2 3 4 5 6 7 8
```

Insert(4, "re")

returns  
→ "Devereloper"



```
{  
    0 references  
class Program  
{  
    {  
        0 references  
        static void Main()  
        {  
            string name = "Developer";  
            //Devesadloper  
            string name_updated = name.Insert(4, "sad");  
            Console.WriteLine(name_updated);  
  
            Console.ReadKey();  
        }  
    }  
}
```

## Methods of System.String class Remove()

object of 'String' class

```
D e v e l o p e r  
0 1 2 3 4 5 6 7 8
```

Remove(2, 4)

returns  
→ "Deper"

StartIndex

no. of character  
(length)

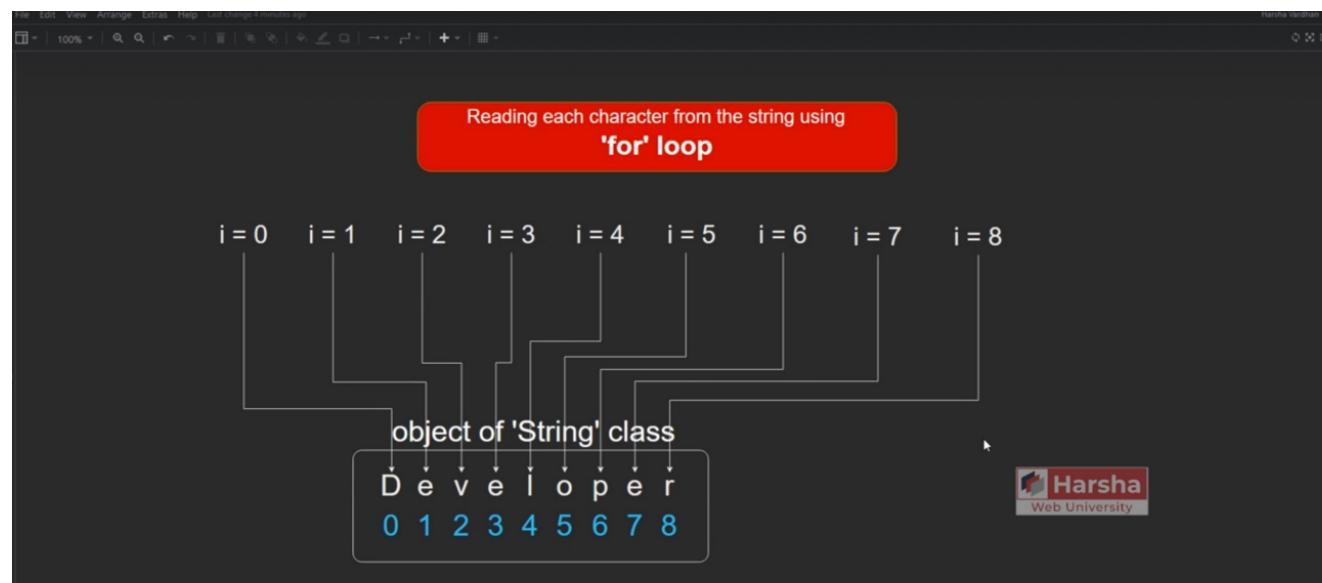


```
{\n    0 references\n    class Program\n    {\n        0 references\n        static void Main()\n        {\n            string name = "Developer";\n            //Developer\n            string name_updated = name.Insert(4, "sad");\n            Console.WriteLine(name_updated);\n\n            string name_updated2 = name.Remove(2, 4);\n            Console.WriteLine(name_updated2);\n            Console.ReadKey();\n        }\n    }\n}
```

String	
#	Method
I8	string <b>Insert(int startIndex, string value)</b> Returns a new string object which inserts the new string value at the specified index in the existing string object.
I9	string <b>Remove( int startIndex, int count)</b> It returns a new string object after removing specified count of characters at the specified start index, from the current string object.

# Strings with 'for' loop

## Part 1



System.String class provides an Indexer

```
string str = "hello";
```

```
str[2];
```

A screenshot of Visual Studio showing a C# program named "StringForLoopExample". The code uses a for loop to iterate through each character of a string and checks if it is a vowel by comparing it against an array of vowels ('A', 'E', 'I', 'O', 'U'). It counts the number of matches and prints them out.

```
static void Main()
{
    string name = "developer@example.com";
    char[] vowels = new char[] { 'A', 'E', 'I', 'O', 'U', 'a', 'e', 'i', 'o', 'u' };

    //for loop
    int vowelsCount = 0;
    for (int i = 0; i < name.Length; i++)
    {
        bool isMatch = false;
        for (int j = 0; j < vowels.Length; j++)
            if (name[i] == vowels[j])
                isMatch = true;

        if (isMatch)
            vowelsCount++;
    }

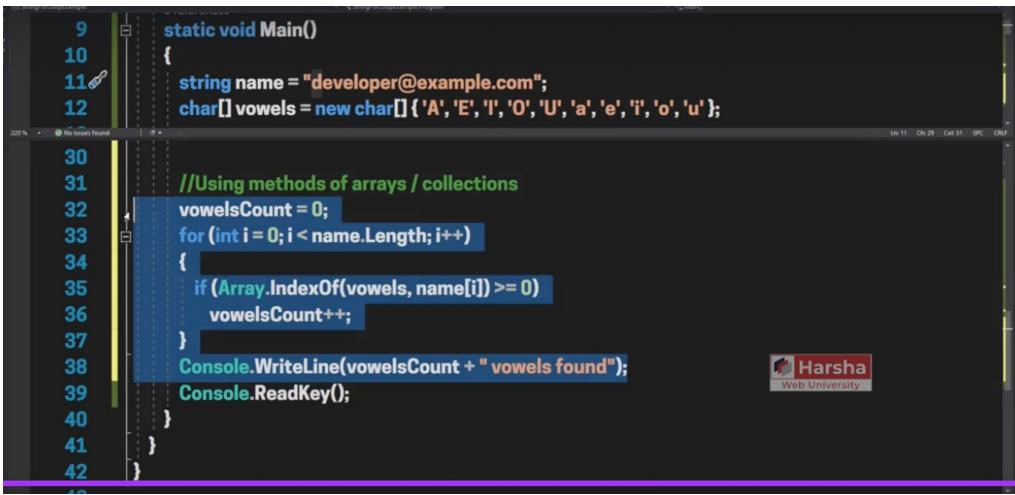
    Console.WriteLine(name[i] + " " + isMatch);
}
```

A screenshot of Visual Studio showing the execution results of the C# program. The output window displays the individual characters of the string "developer@example.com" followed by their respective boolean values indicating if they are vowels or not. The final output shows that there are 8 vowels found in the string.

```
e True
x False
a True
m False
p False
l False
e True
. False
c False
o True
m False
8 vowels found
```

## Strings with 'for' loop

Part 2



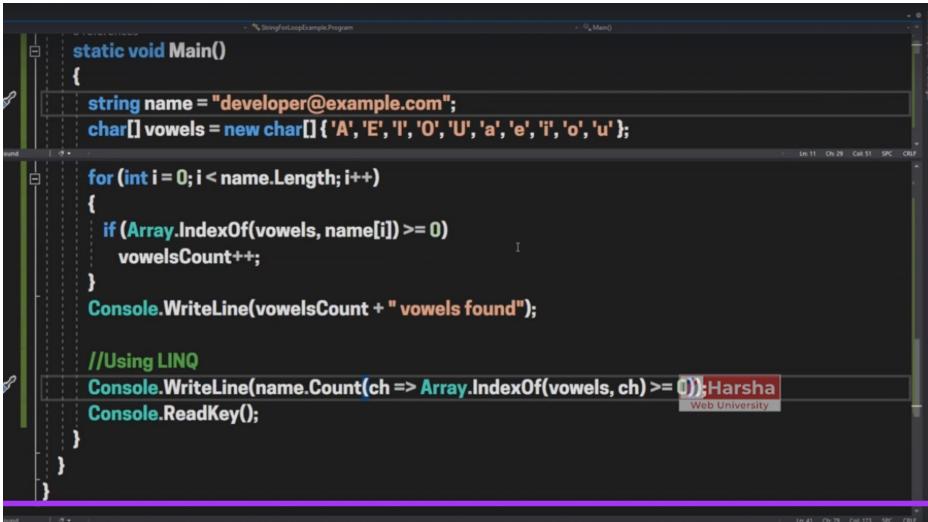
```
9 static void Main()
10 {
11     string name = "developer@example.com";
12     char[] vowels = new char[] { 'A', 'E', 'I', 'O', 'U', 'a', 'e', 'i', 'o', 'u' };
13
14     //Using methods of arrays / collections
15     vowelsCount = 0;
16     for (int i = 0; i < name.Length; i++)
17     {
18         if (Array.IndexOf(vowels, name[i]) >= 0)
19             vowelsCount++;
20     }
21     Console.WriteLine(vowelsCount + " vowels found");
22     Console.ReadKey();
23 }
24 }
```

Harsha  
Web University

In this way, we can write the same code in a shortcut way if you know the proper usage of predefined methods.

## Strings with 'for' loop

### Part 3



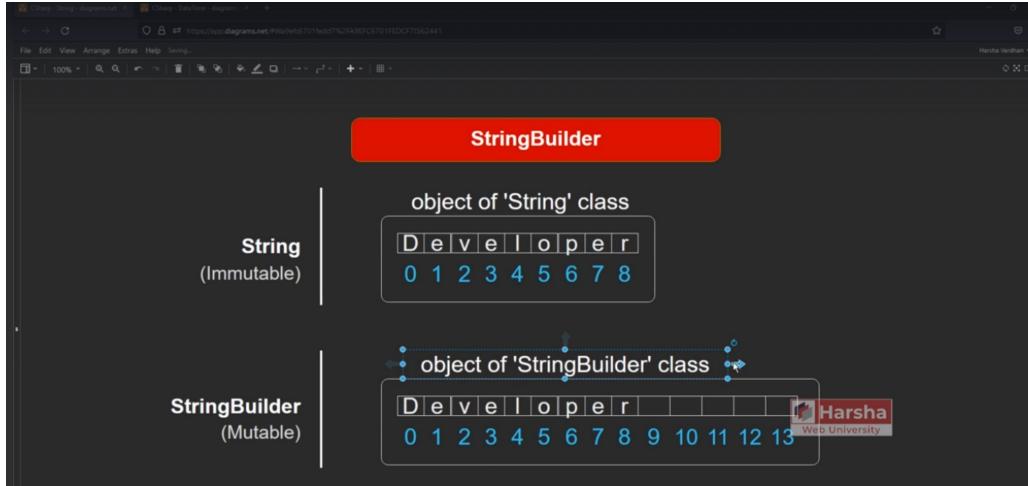
```
static void Main()
{
    string name = "developer@example.com";
    char[] vowels = new char[] { 'A', 'E', 'I', 'O', 'U', 'a', 'e', 'i', 'o', 'u' };

    for (int i = 0; i < name.Length; i++)
    {
        if (Array.IndexOf(vowels, name[i]) >= 0)
            vowelsCount++;
    }
    Console.WriteLine(vowelsCount + " vowels found");

    //Using LINQ
    Console.WriteLine(name.Count(ch => Array.IndexOf(vowels, ch) >= 0));
    Console.ReadKey();
}
```

# StringBuilder

## Part 1



string is immutable and string builder is mutable

so what does it mean exactly? the string object once created cannot be modified means it is immutable means cannot be changed once it is created.

that is the design feature of string so what is the benefit of designing the string as immutable because the string can be converted as hash. so the string can be used as key in the dictionary okay that is the advantage of string being immutable.

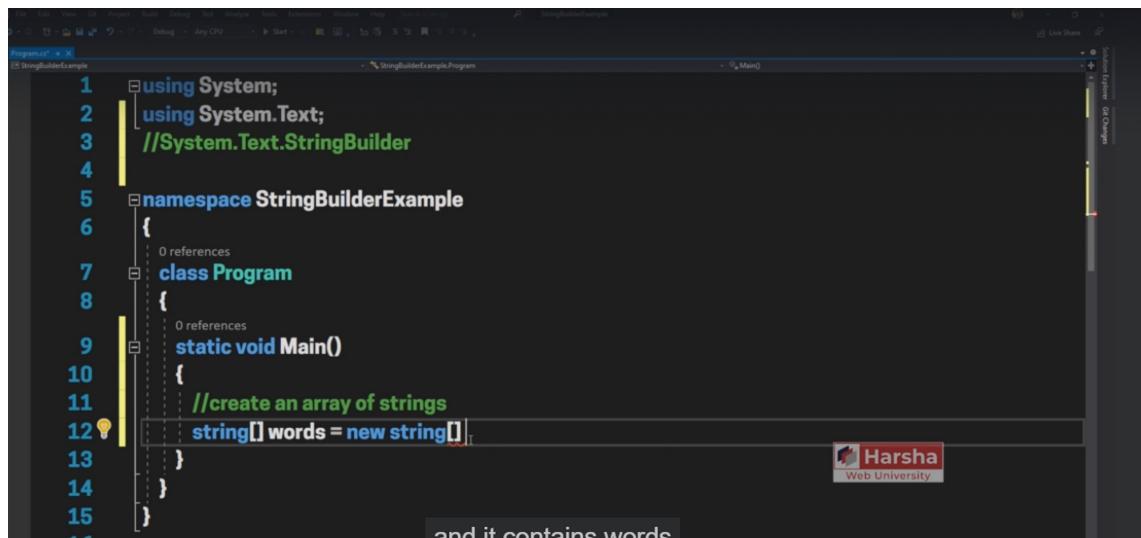
but in my case for example I would like to modify the string so what is the solution for that so if you are expecting the string should be mutable means you should be able to make the changes in the string easily then you will use the string builder that means you will create a string builder object with some initial value and later you would like to modify certain characters or you want to append some extra substring means you would like to concatenate this existing string with another string or you want to remove the characters or you wanted to do continuous modifications on this string value that is absolutely possible in the string builder so in general by default you require to use the string order to store the string values but in case in particular cases that is you would like to keep performing continuous modifications on the string value like for example you would like to modify the string value in a loop and you don't know how many times that should be executed in this kind of cases you will use the string builder over the string.

so if you don't have a plan of modification or you wanted to do just one or two modifications in the string then the string is better to use so if you want to perform continuous modifications such as

concatenating some extra string in a loop and that loop executes multiple times for this kind of cases string builder is better.

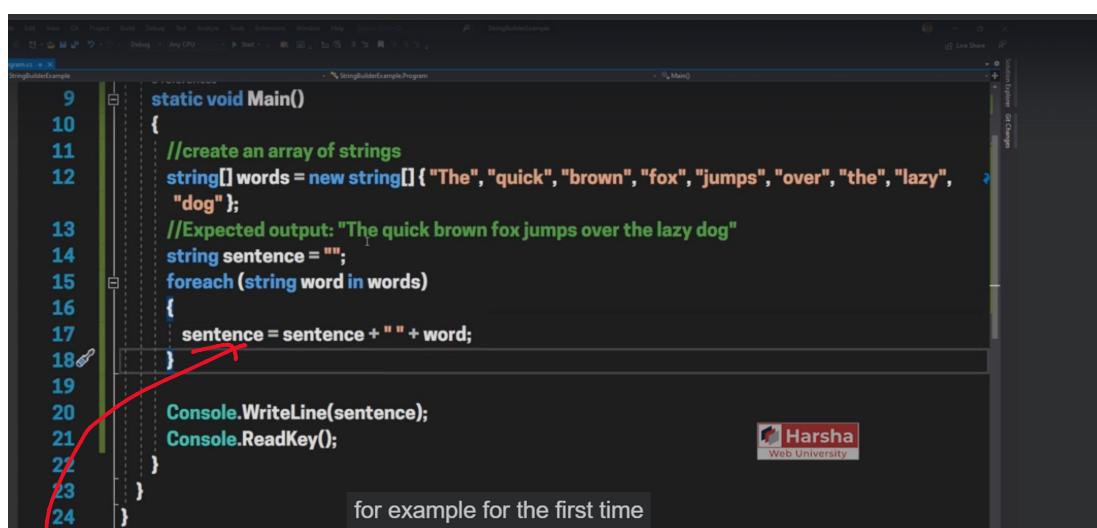
let's say for example there is a messenger application such as whatsapp.

every time when the user sends a message you would like to add that message in a string means every time when the user sends a new message you would like to add that message to an existing string but in this case the point is you really don't know how many messages will be sent by the user it can be one or two or hundreds of or thousands of messages so that is the reason in this case string builder is better than string because you would like to modify means append the strings multiple times but for normal cases such as storing the username email etc the string is okay to use.



```
1 using System;
2 using System.Text;
3 //System.Text.StringBuilder
4
5 namespace StringBuilderExample
6 {
7     class Program
8     {
9         static void Main()
10        {
11            //create an array of strings
12            string[] words = new string[1];
13        }
14    }
15 }
```

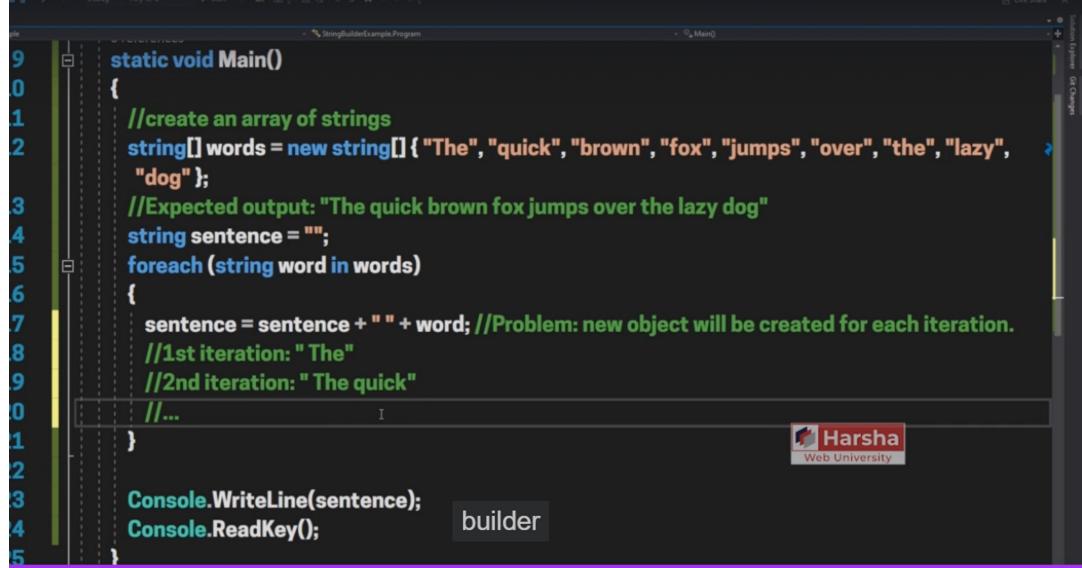
and it contains words



```
9 static void Main()
10 {
11     //create an array of strings
12     string[] words = new string[] { "The", "quick", "brown", "fox", "jumps", "over", "the", "lazy",
13     "dog" };
14     //Expected output: "The quick brown fox jumps over the lazy dog"
15     string sentence = "";
16     foreach (string word in words)
17     {
18         sentence = sentence + " " + word;
19     }
20     Console.WriteLine(sentence);
21     Console.ReadKey();
22 }
```

for example for the first time

Every concatenation creates a new string object unnecessarily.



The screenshot shows a C# code editor with the following code:

```
9 static void Main()
0 {
1     //create an array of strings
2     string[] words = new string[] { "The", "quick", "brown", "fox", "jumps", "over", "the", "lazy",
3         "dog" };
4     //Expected output: "The quick brown fox jumps over the lazy dog"
5     string sentence = "";
6     foreach (string word in words)
7     {
8         sentence = sentence + " " + word; //Problem: new object will be created for each iteration.
9         //1st iteration: " The"
10        //2nd iteration: " The quick"
11        //...
12    }
13
14    Console.WriteLine(sentence);
15    Console.ReadKey();
16 }
```

A tooltip "builder" is visible near the bottom right of the code editor area. The status bar at the bottom right shows "Harsha Web University".

## StringBuilder

### Part 2

**StringBuilder**

<b>What</b> <ul style="list-style-type: none"> <li>The System.Text.StringBuilder is a class that represents an appendable string.</li> </ul>	<p>Diagram illustrating the internal structure of a StringBuilder object. It shows a 1D array of characters ('H', 'E', 'L', 'L', 'O') with indices from 0 to 10. The character at index 4 is 'L'. Below the array, 'Length' is indicated at index 4, and 'Capacity' is indicated at index 9. A red arrow points to the 'Capacity' index.</p>
<b>How</b>	<pre>StringBuilder sb = new StringBuilder("initial string", capacity); sb.Append("additional string")</pre>

30

Space automatically increases internally without creating new object.

```

sentence = sentence + " " + word; //Problem: new object will be created for each iteration.
//1st iteration: "The"
//2nd iteration: "The quick"
//...
}

//StringBuilder
StringBuilder builder = new StringBuilder();
foreach (string word in words)
{
    builder.Append(word);
}
Console.WriteLine(builder);
Console.ReadKey();
}

```

Internally calls 'builder.ToString()' because WriteLine() method can not print the object

A screenshot of the Microsoft Visual Studio IDE. The menu bar includes File, View, Edit, Project, Build, Debug, Test, Analyze, Tools, Extensions, Windows, Help, and a language dropdown set to C#. The toolbar has icons for Undo, Redo, Save, Cut, Copy, Paste, Find, Replace, and others. The title bar says "StringBuilderExample". The solution explorer shows a project named "StringBuilderExample" with a file "Program.cs". The code editor contains the following C# code:

```
17 sentence = sentence + " " + word; //Problem: new object will be created for each iteration
18 //1st iteration: " The"
19 //2nd iteration: " The quick"
20 //...
21 }
22
23 //StringBuilder
24 StringBuilder builder = new StringBuilder();
25 foreach (string word in words)
26 {
27     builder.Append(word);
28 }
29 Console.WriteLine(builder.ToString());
30 Console.ReadKey();
31 }
32 }
33 }
34 }
```

The code demonstrates two ways to build a string from an array of words. The first approach uses a string concatenation loop, which creates a new string object for each iteration. The second approach uses a `StringBuilder` object, which is more efficient as it reuses the same underlying buffer for each append operation.

A screenshot of a terminal window titled "Windows PowerShell" with the path "C:\Windows\system32\cmd.exe". The command prompt shows the user's input and the resulting output of the C# program. The output is identical to the one shown in the Visual Studio code editor, demonstrating the two methods for building the sentence.

```
sentence = sentence + " " + word; //Problem: new object will be created for each iteration
//1st iteration: " The"
//2nd iteration: " The quick"
//...
}

//StringBuilder
StringBuilder builder = new StringBuilder();
foreach (string word in words)
{
    builder.Append(word);
    builder.Append(" ");
}
Console.WriteLine(builder.ToString());
Console.ReadKey();
```

## StringBuilder

| Harsha

### What

- > The System.Text.StringBuilder is a class that represents an appendable string.
- > The string value stored in StringBuilder can be modified, without recreating StringBuilder object.

### How



StringBuilder



```
StringBuilder sb = new StringBuilder("initial string", capacity);
sb.Append("additional string")
```

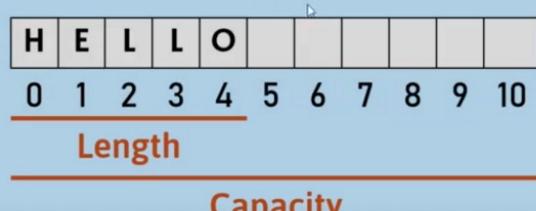
:30

## StringBuilder

| Harsha

### 'Capacity' of String Builder

- > The 'Capacity' property of String Builder represents the number of characters that can be stored in the string builder, as per current memory allocation.
- > When the programmer tries to store much more number of characters than the Capacity, String Builder automatically increases the 'Capacity' property to its double.
- > The default value of 'Capacity' is 16; the programmer can assign its value via constructor or set accessor of the 'Capacity' property.



:30

A screenshot of the Visual Studio IDE. The title bar shows "StringBuilderExample" and "StringBuilderExample.Program". The code editor displays the following C# code:

```
17 sentence = sentence + " " + word; //Problem: new object will be created for each iteration.
18 //1st iteration: " The"
19 //2nd iteration: " The quick"
20 //...
21 }
22
23 //StringBuilder
24 StringBuilder builder = new StringBuilder();
25 foreach (string word in words)
26 {
27     builder.Append(word);
28     builder.Append(" ");
29     Console.WriteLine(builder.ToString() + ", " + builder.Capacity);
30 }
31 Console.WriteLine(builder.ToString());
32 Console.ReadKey();
33 }
34 }
```

The line `Console.WriteLine(builder.ToString() + ", " + builder.Capacity);` is highlighted with a yellow rectangle. The status bar at the bottom right shows "Ln: 29 Ch: 81 Col: 159 SPC CRLF".

A screenshot of a terminal window titled "Console Output" showing the output of the program. The output is:

```
1 The , 16
1 The quick , 16
1 The quick brown , 16
1 The quick brown fox , 32
2 The quick brown fox jumps , 32
2 The quick brown fox jumps over , 32
2 The quick brown fox jumps over the , 64
2 The quick brown fox jumps over the lazy , 64
2 The quick brown fox jumps over the lazy dog , 64
2 The quick brown fox jumps over the lazy dog
2
2
2
30 }
```

## Methods of StringBuilder

## StringBuilder

Harsha

### 'StringBuilder' class Constructor

Constructor	Description
<code>StringBuilder()</code>	It initializes the <code>StringBuilder</code> type of object with the default capacity (16).
<code>StringBuilder( int capacity )</code>	It initializes the <code>StringBuilder</code> type of object with the specified capacity.
<code>StringBuilder( string value )</code>	It initializes the <code>StringBuilder</code> type of object with the specified value.
<code>StringBuilder( string value, int capacity )</code>	It initializes the <code>StringBuilder</code> type of object with the specified value and capacity.

## StringBuilder

Harsha

### StringBuilder Properties

#	Property	
1	<code>int Length { get; set; }</code>	This property gets / sets the length (number of characters) in the string builder.
2	<code>char [int index] { get; set; }</code>	This indexer gets / sets the single character at the specified character index.
3	<code>int Capacity { get; set; }</code>	This property returns the number of characters that can be stored in the current string builder (currently memory allocated). Default is 16.
4	<code>int MaxCapacity { get; }</code>	This property represents maximum number of characters up to which, the Capacity can be extended. Default is <code>int.MaxValue</code> .

```
StringBuilderExample
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
```

```
    //...
}
//Console.WriteLine(sentence);

//StringBuilder
StringBuilder builder = new StringBuilder("hello ", 20);
foreach (string word in words)
{
    builder.Append(word);
    builder.Append(" ");
    Console.WriteLine(builder.ToString() + ", " + builder.Length + ", " + builder.Capacity);
}
builder[0] = 'v';
Console.WriteLine(builder.ToString());
Console.ReadKey();
```

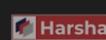


#	Method	Description
1	string Append(string value)	Adds the given string value at the end of current value of string builder.
2	string Insert(int startIndex, string value)	Returns a new string object which inserts the new string value at the specified index in the existing string object.
3	string Remove( int startIndex, int count)	It returns a new string object after removing specified count of characters at the specified start index, from the current string object.
4	string Replace( string oldString, string newString )	Returns a string after replacing all occurrences of oldString with newString.
5	string ToString()	Returns the current value of string builder as a string object.

```
StringBuilderExample
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
```

```
foreach (string word in words)
{
    builder.Append(word);
    builder.Append(" ");
    Console.WriteLine(builder.ToString() + ", " + builder.Length + ", " + builder.Capacity);
}
builder[0] = 'v';
Console.WriteLine(builder.ToString());
Console.WriteLine(builder.MaxCapacity);

Console.WriteLine(builder.Insert(5, "updated"));
Console.WriteLine(builder.Remove(builder.ToString().IndexOf("q"), 5));
Console.ReadKey();
```



```
foreach (string word in words)
{
    builder.Append(word);
    builder.Append(" ");
    Console.WriteLine(builder.ToString() + ", " + builder.Length + ", " + builder.Capacity);
}
builder[0] = 'v';
Console.WriteLine(builder.ToString());
Console.WriteLine(builder.MaxCapacity);

Console.WriteLine(builder.Insert(5, "updated"));
Console.WriteLine(builder.Remove(builder.ToString().IndexOf("q"), 5));
Console.WriteLine(builder.Replace("a", "r"));
Console.ReadKey();
}
```

StringBuilder	
#	Method
1	string <b>Append(string value)</b> Adds the given string value at the end of current value of string builder.
2	string <b>Insert(int startIndex, string value)</b> Returns a new string object which inserts the new string value at the specified index in the existing string object.
3	string <b>Remove( int startIndex, int count)</b> It returns a new string object after removing specified count of characters at the specified start index, from the current string object.
4	string <b>Replace( string oldString, string newString )</b> Returns a string after replacing all occurrences of oldString with newString.
5	string <b>ToString()</b> Returns the current value of string builder as a string object.

### When to use 'String'

- When you would like to make less number of changes to the string.
- When you perform limited number of concatenation operations.
- When you want to perform extensive search operations using methods such as IndexOf, Contains, StartsWith etc.

### When to use 'String Builder'

- When you would like to unknown number of / extensive number of changes to a string (Eg: making changes / concatenations to string through a loop).
- When you perform less number of search operations on strings.

# Date**Time**

## Part 1

The screenshot shows a diagram editor interface with a central workspace. At the top center, there is a red rounded rectangle containing the text "DateTime". Below it, a white rectangular box contains the text "Creating DateTime using 'Parse' method" in orange. Underneath that, another white box contains the code "DateTime.Parse("2025-12-31 11:59:59.999 PM")". To the right of the workspace, there is a vertical table titled "DateTime" with columns for various date and time components. The table rows are:

	DateTime
yyyy	2025
MM	12
dd	31
hh	11
HH	23
mm	59
ss	59
fff	999
tt	PM

A screenshot of the Microsoft Visual Studio IDE. The current file is `Program.cs`, which contains the following C# code:

```
4  namespace DateTimeExample
5  {
6      class Person
7      {
8          public string PersonName { get; set; }
9          public DateTime DateOfBirth { get; set; }
10     }
11    class Program
12    {
13        static void Main()
14        {
15            Person person1 = new Person();
16            person1.PersonName = "Miller";
17            person1.DateOfBirth = 2000 - 12 - 31;
18        }
19    }
20 }
```

The code uses the `DateTime` type, which is highlighted in red. A tooltip appears over the assignment statement `person1.DateOfBirth = 2000 - 12 - 31;`, stating: "Cannot implicitly convert type 'int' to 'System.DateTime'". The tooltip also includes a link to the .NET Framework source code for the `DateTime` type.

A screenshot of the Microsoft Visual Studio IDE, showing the same `Program.cs` file as the first image, but with a different date format. The code now uses the `DateTime.Parse` method to convert a string into a `DateTime` object:

```
11    class Program
12    {
13        static void Main()
14        {
15            Person person1 = new Person();
16            person1.PersonName = "Miller";
17            person1.DateOfBirth = DateTime.Parse("2000-12-31 11:59:59.999 am");
18            Console.WriteLine(person1.DateOfBirth.ToString());
19            Console.ReadKey();
20        }
21    }
22 }
```

This version of the code compiles successfully without any errors or warnings.

# DateTime

## Part 2

Properties of DateTime

Eg: 2025-12-31 11:59:59.999 PM

DateTime	
yyyy	2025
MM	12
dd	31
hh	11
HH	23
mm	59
ss	59
fff	999
tt	PM

```
int Day { get; }           int Second { get; }
int Month { get; }          int Millisecond { get; }
int Year { get; }           int DayOfYear { get; }
int Hour { get; }            DayOfWeek DayOfWeek { get; }
int Minute { get; }          static DateTime Now { get; }
```

```
Console.WriteLine("Day " + person1.DateOfBirth.Day);
Console.WriteLine("Month " + person1.DateOfBirth.Month);
Console.WriteLine("Year " + person1.DateOfBirth.Year);
Console.WriteLine("Hours " + person1.DateOfBirth.Hour);
Console.WriteLine("Minutes " + person1.DateOfBirth.Minute);
Console.WriteLine("Seconds " + person1.DateOfBirth.Second);
Console.WriteLine("Milliseconds " + person1.DateOfBirth.Millisecond);
Console.WriteLine("Day of week " + person1.DateOfBirth.DayOfWeek);
Console.WriteLine("Day of week " + (int)person1.DateOfBirth.DayOfWeek);
Console.WriteLine("Day of year " + person1.DateOfBirth.DayOfYear);
Console.ReadKey();
```

```

DateTime dt = DateTime.Now;
Console.WriteLine(dt.ToString());
Console.ReadKey();

```

```

static void Main()
{
    Person person1 = new Person();
    person1.PersonName = "Miller";
    person1.DateOfBirth = DateTime.Parse("2000-12-31 11:59:59.999 am");
    person1.DateOfBirth = Convert.ToDateTime("2000-12-31 11:59:59.999 am");
    Console.WriteLine(person1.DateOfBirth.ToString());

    Console.WriteLine("Day " + person1.DateOfBirth.Day);
    Console.WriteLine("Month " + person1.DateOfBirth.Month);
    Console.WriteLine("Year " + person1.DateOfBirth.Year);
    Console.WriteLine("Hours " + person1.DateOfBirth.Hour);
    Console.WriteLine("Minutes " + person1.DateOfBirth.Minute);
    Console.WriteLine("Seconds " + person1.DateOfBirth.Second);
    Console.WriteLine("Milliseconds " + person1.DateOfBirth.Millisecond);
    Console.WriteLine("Day of week " + person1.DateOfBirth.DayOfWeek);
    Console.WriteLine("Day of week " + (int)person1.DateOfBirth.DayOfWeek);
    Console.WriteLine("Day of year " + person1.DateOfBirth.DayOfYear);
}

```

**DateTime**

What		DateTime	
<ul style="list-style-type: none"> <li>The System.DateTime is a structure that represents date and time value.</li> <li>Default format: yyyy-MM-dd hh:mm:ss.fff tt</li> </ul>		yyyy	2025
		MM	12
		dd	31
		hh	11
		HH	23
		mm	59
		ss	59
		fff	999
		tt	PM

**How**

**Creating DateTime using 'Parse' method**

**DateTime.Parse("2025-12-31 11:59:59.999 PM")**

**Creating DateTime using 'ToDateTime' method**

**Convert.ToDateTime("2025-12-31 11:59:59.999 PM")**

## Date**Time** Formats

Part 1

## Methods of DateTime

string ToString()	string ToShortTimeString()
string ToString(string format)	string ToLongTimeString()
string ToShortDateString()	static int DaysInMonth(int year, int month)
string ToLongDateString()	static DateTime Parse(string value)
static DateTime ParseExact(string value, string format, IFormatProvider provider, DateTimeStyle style)	



```
DateTime dt = DateTime.Parse("2020-12-31 11:59:59.999");
string str1 = dt.ToString(); //MM/dd/yyyy hh:mm:ss tt
string str2 = dt.ToShortDateString(); //MM/dd/yyyy
string str3 = dt.ToLongDateString(); //dd MMMM yyyy
string str4 = dt.ToShortTimeString(); //hh:mm tt
string str5 = dt.ToLongTimeString(); //hh:mm:ss tt

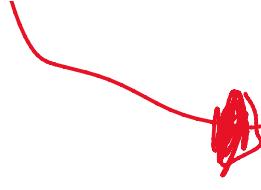
Console.WriteLine(str1);
Console.WriteLine(str2);
Console.WriteLine(str3);
Console.WriteLine(str4);
Console.WriteLine(str5);
Console.ReadKey();
}
```



```
0 references
static void Main()
{
    DateTime dt = DateTime.Parse("2020-12-31 11:59:59.999");
    string str1 = dt.ToString(); //MM/dd/yyyy hh:mm:ss tt
    string str2 = dt.ToShortDateString(); //MM/dd/yyyy
    string str3 = dt.ToLongDateString(); //dd MMMM yyyy
    string str4 = dt.ToShortTimeString(); //hh:mm tt
    string str5 = dt.ToLongTimeString(); //hh:mm:ss tt
    string str6 = dt.ToString("dd-MM-yyyy HH:mm:ss");

    Console.WriteLine(str1);
    Console.WriteLine(str2);
    Console.WriteLine(str3);
    Console.WriteLine(str4);
    Console.WriteLine(str5);
    Console.ReadKey();
}
```





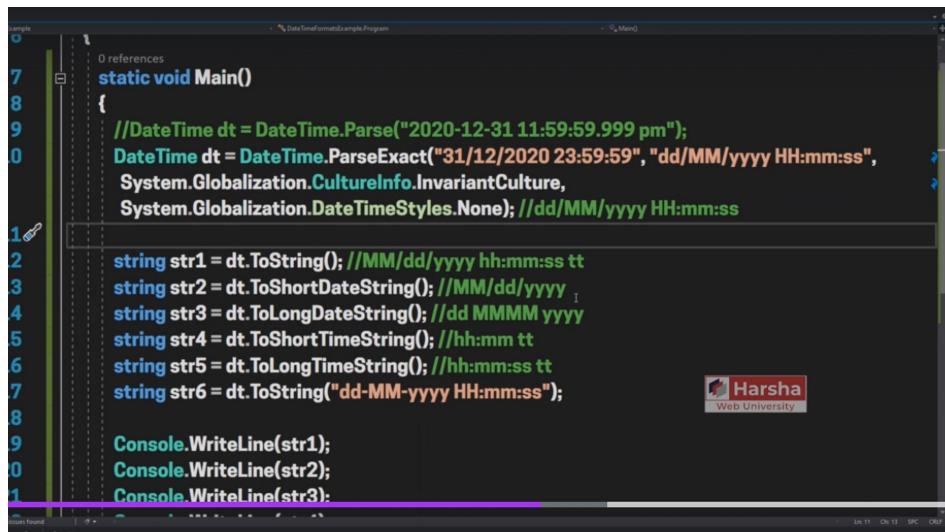
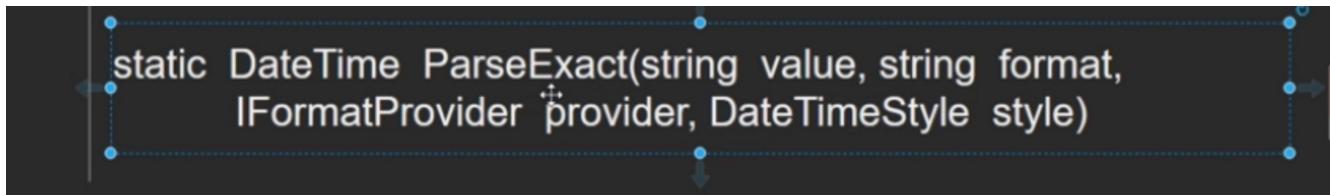
you can supply any other format.

## Date Time Formats

### Part 2

DateTime.Parse() => this method works if the user enters (year-month-date) or (month-date-year) format.

but it doesn't work with other custom specific format. in that case, use



The screenshot shows a C# code editor with the following code:

```
static void Main()  
{  
    //DateTime dt = DateTime.Parse("2020-12-31 11:59:59.999 pm");  
    DateTime dt = DateTime.ParseExact("31/12/2020 23:59:59", "dd/MM/yyyy HH:mm:ss",  
        System.Globalization.CultureInfo.InvariantCulture,  
        System.Globalization.DateTimeStyles.None); //dd/MM/yyyy HH:mm:ss  
  
    string str1 = dt.ToString(); //MM/dd/yyyy hh:mm:ss tt  
    string str2 = dt.ToShortDateString(); //MM/dd/yyyy  
    string str3 = dt.ToLongDateString(); //dd MMMM yyyy  
    string str4 = dt.ToShortTimeString(); //hh:mm tt  
    string str5 = dt.ToLongTimeString(); //hh:mm:ss tt  
    string str6 = dt.ToString("dd-MM-yyyy HH:mm:ss");  
  
    Console.WriteLine(str1);  
    Console.WriteLine(str2);  
    Console.WriteLine(str3);
```

A screenshot of the Visual Studio IDE showing a C# code editor. The code demonstrates the use of `DateTime.ParseExact` to parse a date string. A red box highlights the line where the date string is defined: `DateTime dt = DateTime.ParseExact("31/12/2020 23:59:59", "dd/MM/yyyy HH:mm:ss", System.Globalization.CultureInfo.InvariantCulture, System.Globalization.DateTimeStyles.None);`. The code also shows various methods to format the resulting `dt` object.

```
7 static void Main()
8 {
9     //DateTime dt = DateTime.Parse("2020-12-31 11:59:59.999 pm");
10    DateTime dt = DateTime.ParseExact("31/12/2020 23:59:59", "dd/MM/yyyy HH:mm:ss",
11        System.Globalization.CultureInfo.InvariantCulture,
12        System.Globalization.DateTimeStyles.None); //dd/MM/yyyy HH:mm:ss
13
14    string str1 = dt.ToString(); //MM/dd/yyyy hh:mm:ss tt
15    string str2 = dt.ToShortDateString(); //MM/dd/yyyy
16    string str3 = dt.ToLongDateString(); //dd MMMM yyyy
17    string str4 = dt.ToShortTimeString(); //hh:mm tt
18    string str5 = dt.ToLongTimeString(); //hh:mm:ss tt
19    string str6 = dt.ToString("dd-MM-yyyy HH:mm:ss");
```

A screenshot of the Visual Studio IDE showing the continuation of the C# program. It includes several `Console.WriteLine` statements to output the formatted strings and a call to `Console.ReadKey` at the end of the method. The code block starts from line 11 of the previous screenshot.

```
11
12    string str1 = dt.ToString(); //MM/dd/yyyy hh:mm:ss tt
13    string str2 = dt.ToShortDateString(); //MM/dd/yyyy
14    string str3 = dt.ToLongDateString(); //dd MMMM yyyy
15    string str4 = dt.ToShortTimeString(); //hh:mm tt
16    string str5 = dt.ToLongTimeString(); //hh:mm:ss tt
17    string str6 = dt.ToString("dd-MM-yyyy HH:mm:ss");
18
19    Console.WriteLine(str1);
20    Console.WriteLine(str2);
21    Console.WriteLine(str3);
22    Console.WriteLine(str4);
23    Console.WriteLine(str5);
24    Console.WriteLine(str6);
25    Console.WriteLine(DateTime.DaysInMonth(2022, 7));
26    Console.ReadKey();
27 }
```

A screenshot of a presentation slide titled "DateTime Methods". The slide lists four methods of the `DateTime` class:

#	Method	Description
1	<code>string ToString()</code>	It returns the date & time value in the default date format, based on current Windows settings.
2	<code>string ToString(string format)</code>	It returns the date & time value in the specified format.
3	<code>string ToShortDateString()</code>	It returns the date value in the default short date format, based on current Windows settings. Eg: MM/dd/yyyy   12/31/2030
4	<code>string ToLongDateString()</code>	It returns the date value in the default long date format, based on current Windows settings. Eg: dd MMMM yyyy   31 December 2030

## DateTime

### DateTime Methods

#	Method
5	<code>string ToShortTimeString()</code> It returns the date value in the default short time format, based on current Windows settings. Eg: hh:mm tt   11:59 pm
6	<code>string ToLongTimeString()</code> It returns the date value in the default long time format, based on current Windows settings. Eg: hh:mm:ss tt   11:59:59 pm
7	<code>static int DaysInMonth(int year, int month)</code> It returns number of days in the specified month in the specified year.

Web University

## DateTime

### DateTime Properties

#	Property
1	<code>int Day { get; }</code> It returns the day (1 to 31) of current date & time value.
2	<code>int Month { get; }</code> It returns the month (1 to 12) of current date & time value.
2	<code>int Year { get; }</code> It returns the year (1 to 9999) of current date & time value.
3	<code>int Hour { get; }</code> It returns the hour (0 to 23) of current date & time value.
4	<code>int Minute { get; }</code> It returns the minute (0 to 59) of current date & time value.
5	<code>int Second { get; }</code> It returns the second (0 to 59) of current date & time value.

Harsha

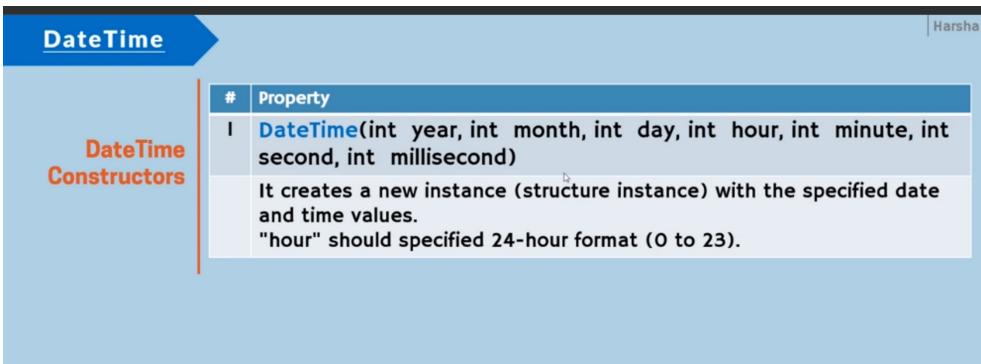
## DateTime

### DateTime Properties

#	Property
6	<code>int Millisecond { get; }</code> It returns the millisecond (0 to 999) of current date & time value.
7	<code>int DayOfYear { get; }</code> It returns the day of year (1 to 366) based on current date & time value.
8	<code>DayOfWeek DayOfWeek { get; }</code> It returns the day of week - Sunday to Saturday (0 to 6) based on current date & time value.
9	<code>static DateTime Now { get; }</code> It returns an instance of DateTime structure that represents the current system date & time.

Harsha

Web University



The screenshot shows a Visual Studio code editor window with the title "% DateTimFormatExample Program". The code demonstrates various ways to format a `DateTime` object:

```
static void Main()
{
    //DateTime dt = DateTime.Parse("2020-12-31 11:59:59.999 pm");
    //DateTime dt = DateTime.ParseExact("31/12/2020 23:59:59", "dd/MM/yyyy HH:mm:ss",
    System.Globalization.CultureInfo.InvariantCulture,
    System.Globalization.DateTimeStyles.None); //dd/MM/yyyy HH:mm:ss

    DateTime dt = new DateTime(2020, 12, 31, 23, 59, 59, 999);
    string str1 = dt.ToString(); //MM/dd/yy 12/31/2020 11:59:59 PM
    string str2 = dt.ToShortDateString(); //12/31/2020
    string str3 = dt.ToLongDateString(); //31 December 2020
    string str4 = dt.ToShortTimeString(); //11:59 PM
    string str5 = dt.ToLongTimeString(); //11:59:59 PM
    string str6 = dt.ToString("dd-MM-yy"); //31-12-2020 23:59:59
    before
    Console.WriteLine(str1);
    Console.WriteLine(str2);
}
```

## Date Subtraction

## Subtracting DateTime

```
int CompareTo(DateTime value)
TimeSpan Subtract(DateTime value)
```

System.TimeSpan is a structure that represents difference between two dates.

```
class Employee
{
    public string EmployeeName { get; set; }
    public DateTime DateOfJoining { get; set; }
    public double Experience { get; set; }
}

class Program
{
    static void Main()
    {
        Employee emp = new Employee();
    }
}
```

### Date Time

#	Method
I	int <b>CompareTo(DateTime value)</b>  It returns: -1: This instance value is earlier than value. 0: This instance value is equal to value. 1: This instance value is later than value.
II	TimeSpan <b>Subtract(DateTime value)</b>  It returns an instance of TimeSpan structure, that represents date difference between the current instance and given date value.

## Subtracting DateTime

```
int CompareTo(DateTime value)  
TimeSpan Subtract(DateTime value)
```

System.TimeSpan is a  
structure that represents  
difference between two dates.

```
5     1 reference  
6     public string EmployeeName { get; set; }  
7     3 references  
8     public DateTime DateOfJoining { get; set; }  
9     0 references  
10    public double ExperienceYears { get; set; }  
11    0 references  
10   public double ExperienceMonths { get; set; }  
11 }
```

The screenshot shows a code editor with the Employee class defined. The ExperienceMonths property is highlighted with a red border. The code editor interface includes a sidebar with file navigation and a status bar at the bottom.

```
static void Main()  
{  
    Employee emp = new Employee() { EmployeeName = "John",  
        ("2015-01-01");  
        //Today: 2021-10-11  
    DateTime today = DateTime.Now;  
    if (today.CompareTo(emp.DateOfJoining) == 1) //1, 0, -1  
    {  
        TimeSpan ts = today.Subtract(emp.DateOfJoining);  
        emp.ExperienceYears = Math.Floor(ts.TotalDays / 365);  
        Console.WriteLine(emp.ExperienceYears);  
        emp.ExperienceMonths = (ts.TotalDays - (emp.ExperienceYears * 365)) / 30;  
        Console.WriteLine(emp.ExperienceMonths);  
    }  
    else  
    {  
        represents the experience of 9 months  
        Console.WriteLine("Date of joining is greater than today's date. Subtraction is not possible");  
    }  
}
```

The screenshot shows the Main() method being executed. A tooltip for the DateTime.Parse method is visible, showing its parameters and return type. The console output shows the calculated experience values. The code editor interface includes a sidebar with file navigation and a status bar at the bottom.

A screenshot of a C# code editor window titled "SubtractWithExceptionProgram". The code is as follows:

```
static void Main()
{
    Employee emp = new Employee() { EmployeeName = "John", DateOfJoining = DateTime.Parse
        ("2015-01-01") };
    //Today: 2021-10-11
    DateTime today = DateTime.Now;
    if (today.CompareTo(emp.DateOfJoining) == 1) //1, 0, -1
    {
        TimeSpan ts = today.Subtract(emp.DateOfJoining);
        emp.ExperienceYears = Math.Floor(ts.TotalDays / 365);

        emp.ExperienceMonths = Math.Floor((ts.TotalDays - (emp.ExperienceYears * 365)) / 30);
        Console.WriteLine(emp.ExperienceYears + " years and " + emp.ExperienceMonths + "
            months");
    }
    else
    {
        Console.WriteLine("Date of joining is greater than today's date. Subtraction is not
            possible.");
    }
}
```

The cursor is positioned at the end of the first line of the subtraction logic. A tooltip from "Harsha Web University" provides information about the current context.

## Date Addition

A slide titled "Methods of DateTime" featuring a red button-like background. The slide lists several methods of the DateTime class:

- DateTime AddDays(double value)
- DateTime AddMonths(double value)
- DateTime AddYears(double value)
- DateTime AddHours(double value)
- DateTime AddMinutes(double value)
- DateTime AddSeconds(double value)
- DateTime AddMilliseconds(double value)

A logo for "Harsha Web University" is visible in the bottom right corner.

```

4 {
5     0 references
6     class Program
7     {
8         0 references
9         static void Main()
10        {
11            DateTime dt = DateTime.Parse("2022-01-01 12:00 am");
12            DateTime dt_after_10_days = dt.AddDays(10);
13            Console.WriteLine("After 10 days: " + dt_after_10_days);
14            DateTime dt_before_10_days = dt.AddDays(-10);
15            Console.WriteLine("Before 10 days: " + dt_before_10_days);
16
17            DateTime dt_after_20_months_and_5_days = dt.AddMonths(20).AddDays(5);
18            Console.WriteLine("After 20 months and 5 days: " + dt_after_20_months_and_5_days);
19            Console.ReadKey();
20        }
}

```

DateTime		
DateTime Methods	#	Method
	I2	DateTime <b>AddDays(double value)</b> It creates and returns a new instance of DateTime structure, after adding specified days (+ve or -ve) to the current date & time value.
	I3	DateTime <b>AddMonths(double value)</b> It creates and returns a new instance of DateTime structure, after adding specified months (+ve or -ve) to the current date & time value.
	I4	DateTime <b>AddYears(double value)</b> It creates and returns a new instance of DateTime structure, after adding specified years (+ve or -ve) to the current date & time value.
	I5	DateTime <b>AddHours(double value)</b> It creates and returns a new instance of DateTime structure, after adding specified hours (+ve or -ve) to the current date & time value.

DateTime		
DateTime Methods	#	Method
	I6	DateTime <b>AddMinutes(double value)</b> It creates and returns a new instance of DateTime structure, after adding specified minutes (+ve or -ve) to the current date & time value.
	I7	DateTime <b>AddSeconds(double value)</b> It creates and returns a new instance of DateTime structure, after adding specified seconds (+ve or -ve) to the current date & time value.
	I8	DateTime <b>AddMilliseconds(double value)</b> It creates and returns a new instance of DateTime structure, after adding specified milliseconds to the current date & time value.

# 'Math' class

## Part 1

### Methods of System.Math class

```
static double Pow(double x, double y)  
static double Min(double val1, double val2)  
static double Max(double val1, double val2)  
static double Floor(double value)  
static double Ceiling(double value)  
static double Round(double value)
```



### Math

#### Math Methods

#	Method
1	static double Pow(double x, double y) It returns the value of x power y.
2	static double Min(double val1, double val2) It returns the minimum (smaller) among given two numbers.
3	static double Max(double val1, double val2) It returns the maximum (bigger) among given two numbers.
4	static double Floor(double value) It rounds-down the given number. Eg: 10.29 becomes 10.
5	static double Ceiling(double value) It rounds-up the given number. Eg: 10.29 becomes 11.
6	static double Round(double value) It rounds up / down the given number, to the nearest integer value. Eg: 10.29 becomes 10 and 10.59 becomes 11.

```
double a = Math.Pow(10, 4); //10 power 4 (10 * 10 * 10 * 10)
double b = Math.Min(5.623, 10.82); //smaller among given numbers
double c = Math.Max(5.623, 10.82); //bigger among given numbers
double d = Math.Floor(20.83984); //returns the integer part
double e = Math.Ceiling(20.23984); //returns the next integer
double f = Math.Round(20.23984); //round down
double g = Math.Round(20.253984); //round up
Console.WriteLine(a); //Output: 10000
Console.WriteLine(b); //Output: 5.623
Console.WriteLine(c); //Output: 10.82
Console.WriteLine(d); //Output: 20
Console.WriteLine(e); //Output: 21
Console.WriteLine(f); //Output: 20
Console.WriteLine(g); //Output: 21
```

20.553984

It considers the first digit in the fraction part.

```
double g = Math.Round(20.53984); //round up
double h = Math.Round(20.53984, 2); //round up
```

Console.WriteLine(a); //Output: 10000  
Console.WriteLine(b); //Output: 5.623  
Console.WriteLine(c); //Output: 10.82  
Console.WriteLine(d); //Output: 20  
Console.WriteLine(e); //Output: 21  
Console.WriteLine(f); //Output: 20  
Console.WriteLine(g); //Output: 21  
Console.WriteLine(h); //Output: 20.54

# 'Math' class

## Part 2

**Math**

**Math Methods**

#	Method
7	static double <b>Round(double value, int decimals)</b> It rounds up / down the given number, to the specified fractional digits. Eg: Math.Round(value, 2) 10.272 becomes 10.27 and 10.275 becomes 10.38.
8	static int <b>Sign(double value)</b> It returns: -1: if the value is a negative value 0: if the value is equal to 0 1: if the value is a positive value
9	static double <b>Abs(double value)</b> It returns the positive value of given number.
10	static int <b>DivRem(int val1, int val2, out int result)</b> It returns the remainder value of val1/val2; and provides the quotient value as 'result' out parameter.

**Harsha**  
Web University

```
double h = Math.Round(20.53984, 2); //round up
double i = Math.Sign(-10); // -1 if negative value
double j = Math.Abs(-10); //returns positive number based on given negative number

Console.WriteLine(a); //Output: 10000
Console.WriteLine(b); //Output: 5.623
Console.WriteLine(c); //Output: 10.82
Console.WriteLine(d); //Output: 20
Console.WriteLine(e); //Output: 21
Console.WriteLine(f); //Output: 20
Console.WriteLine(g); //Output: 21
Console.WriteLine(h); //Output: 20.54
Console.WriteLine(i); //Output: -1
Console.WriteLine(j); //Output: 10

Console.ReadKey();
```

Output:

```
10000
5.623
10.82
20
21
20
21
20.54
-1
10
```

**Harsha**  
Web University

10 static int DivRem(int val1, int val2, out int result)

It returns the quotient value of val1/val2; and provides the remainder value as 'result' out parameter.

```
//10 / 3 = Quotient: 3; remainder: 1
int rem;
double quo = Math.DivRem(10, 3, out rem);
Console.WriteLine(a); //Output: 10000
Console.WriteLine(b); //Output: 5.623
Console.WriteLine(c); //Output: 10.82
Console.WriteLine(d); //Output: 20
Console.WriteLine(e); //Output: 21
Console.WriteLine(f); //Output: 20
Console.WriteLine(g); //Output: 21
Console.WriteLine(h); //Output: 20.54
Console.WriteLine(i); //Output: -1
Console.WriteLine(j); //Output: 10
Console.WriteLine("Quotient: " + quo + ", Remainder: " + rem);

Console.ReadKey();
```



```
32 Console.WriteLine("Quotient: " + quo + ", Remainder: " + rem);
33 Console.WriteLine(Math.Floor(10 / 3));
34 Console.WriteLine(10 % 3);
35 Console.ReadKey();
36 }
```

```
double j = Math.Abs(-10); //returns positive number based on given negative number
//10 / 3 = Quotient: 3; remainder: 1
int rem;
double quo = Math.DivRem(10, 3, out rem);
Console.WriteLine(a); //Output: 10000
Console.WriteLine(b); //Output: 5.623
Console.WriteLine(c); //Output: 10.82
Console.WriteLine(d); //Output: 20
Console.WriteLine(e); //Output: 21
Console.WriteLine(f); //Output: 20
Console.WriteLine(g); //Output: 21
Console.WriteLine(h); //Output: 20.54
Console.WriteLine(i); //Output: -1
Console.WriteLine(j); //Output: 10
Console.WriteLine("Quotient: " + quo + ", Remainder: " + rem);
Console.WriteLine(Math.Floor(Convert.ToDouble(10) / 3));
Console.WriteLine(10 % 3);
Console.ReadKey();
```



**Math**

**Math Methods**

#	Method
11	static double Sqrt(double value)
	It returns square root value of given number.

```

double k = Math.Sqrt(25); //Output: 5.0

Console.WriteLine(a); //Output: 10000
Console.WriteLine(b); //Output: 5.623
Console.WriteLine(c); //Output: 10.82
Console.WriteLine(d); //Output: 20
Console.WriteLine(e); //Output: 21
Console.WriteLine(f); //Output: 20
Console.WriteLine(g); //Output: 21
Console.WriteLine(h); //Output: 20.54
Console.WriteLine(i); //Output: -1
Console.WriteLine(j); //Output: 10
Console.WriteLine("Quotient: " + quo + ", Remainder: " + rem); //3 and 1
Console.WriteLine(Math.Floor(Convert.ToDouble(10) / 3)); //3
Console.WriteLine(10 % 3); //1
Console.WriteLine(k); //25 this variable
Console.ReadKey();

```

expects double but we supplied int. because of 'implicit casting'.

## Regular Expressions

Regular expressions or predefined formats based on which you can validate your string values for example you're expecting that the user must enter only digits but the user might enter any kind of values at runtime by using that `Console.ReadLine()` even in the windows forms applications or in web applications also in the text boxes the user might enter any kind of values so there is no guarantee that the user enters only digits but you are expecting only digits suppose in that case there should be a mechanism that can validate the user inputs whether it meets the specified set of characters that you want or not.

## Regular Expressions



```
c:\csharp\RegExExample>
Enter a digit:
oIJEJ3483948u
False

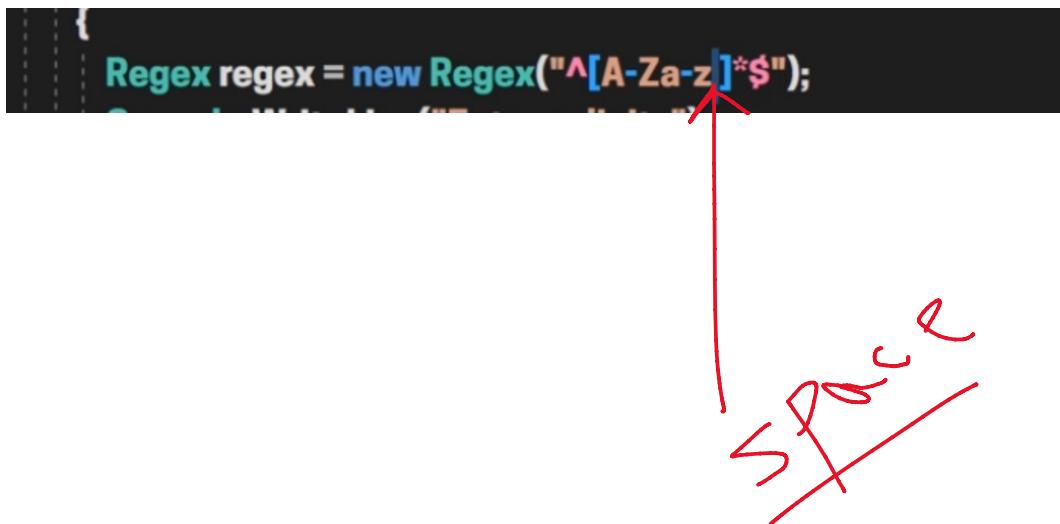
static void Main()
{
    Regex regex = new Regex("^[0-9]*$");
    Console.WriteLine("Enter a digit:");
    string inputValue = Console.ReadLine();
    bool result = regex.IsMatch(inputValue);
    Console.WriteLine(result);
}
```

```
c:\csharp\RegExExample>
0 references
static void Main()
{
    Regex regex = new Regex("^[0-9]*$");
    Console.WriteLine("Enter a digit:");
    string inputValue = Console.ReadLine();
    bool result = regex.IsMatch(inputValue);
    Console.WriteLine(result);
    if (result == true)
    {
        Console.WriteLine("Valid number");
    }
    else
    {
        Console.WriteLine("Invalid number");
    }
    Console.ReadKey();
}
```

A screenshot of the Visual Studio IDE. On the left is the code editor with the following C# code:

```
7
8     {
9         static void Main()
10        {
11            Regex regex = new Regex("^[A-Za-z]*$");
12            Console.WriteLine("Enter a digit:");
13            string inputValue = Console.ReadLine();
14            bool result = regex.IsMatch(inputValue);
15            Console.WriteLine(result);
16            if (result == true)
17            {
18                Console.WriteLine("Valid number");
19            }
20            else
21            {
22                Console.WriteLine("Invalid number");
23            }
24            Console.ReadKey();
25        }
26    }
```

The Solution Explorer on the right shows a single project named "RegExExample". A watermark for "Harsha Web University" is visible in the bottom right corner.



A screenshot of a web-based regular expression reference table. At the top is a red button labeled "Regular Expressions". Below it is a table with two columns:

Symbol	Meaning
[0-9]	1, 2, 3, 4, 5, 6, 7, 8, 9, 0
[a-z]	a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z
[A-Z]	A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, Q, R, S, T, U, , W, X, Y, Z
[1,2,3]	1, 2, 3
\d	1, 2, 3, 4, 5, 6, 7, 8, 9, 0
\w	word
\s	White space

A watermark for "Harsha Web University" is visible in the bottom right corner.

## Regex

| Harsha

### What

- › Regular Expression is a pattern that contains set of conditions of a string value.
- › Example: `^[a-zA-Z]*$` - for alphabets and spaces only
- › The 'Regex' class represents regular expression to check whether the string value matches with specified pattern or not.
- › Useful for validations.

### How

## Regex



Harsha

```
Regex referenceVariable = new Regex( "Your Pattern Here");  
referenceVariable.IsMatch( value ); //returns true or false
```