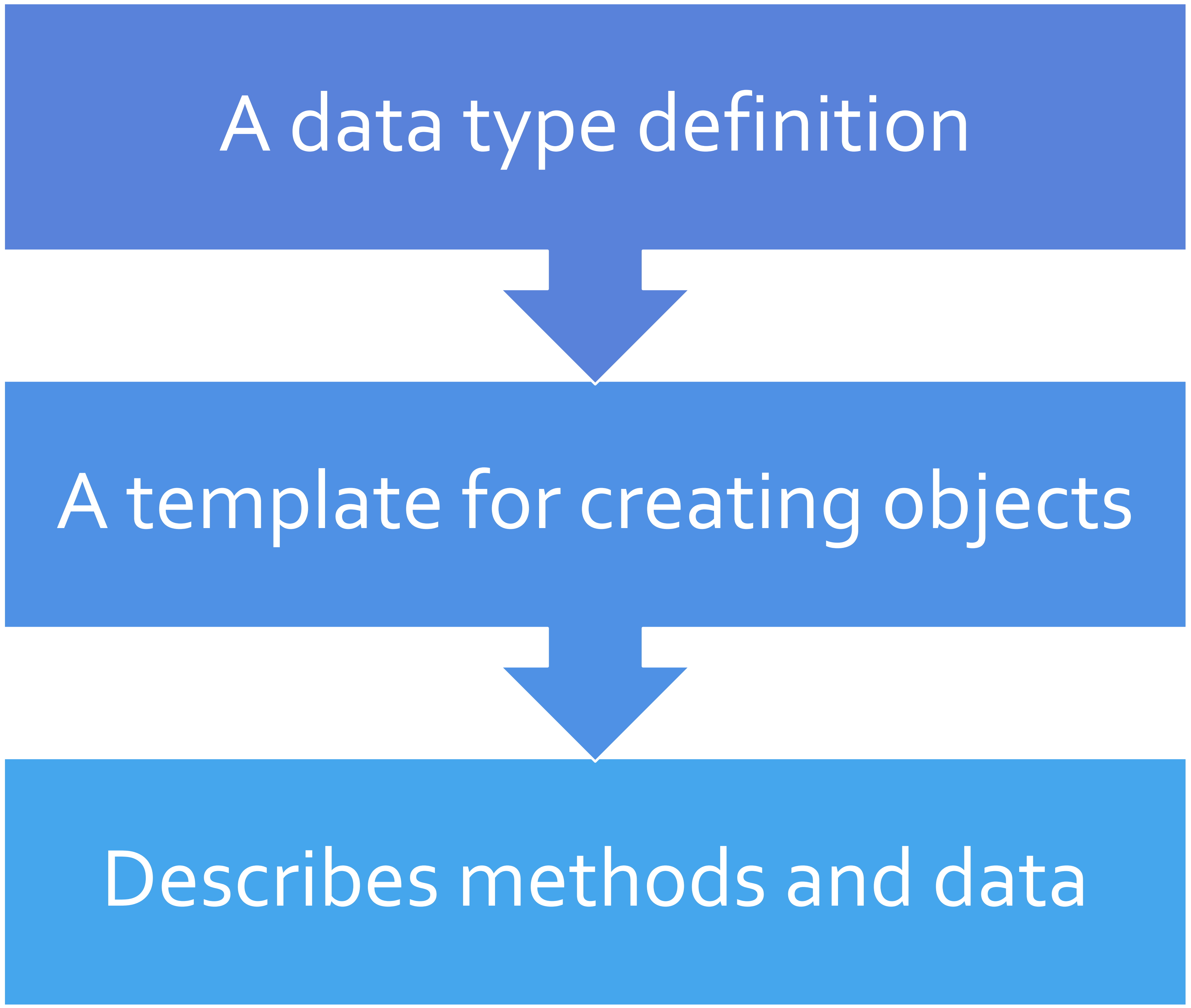
TODAY'S GOAL

Deepen our understanding of Object-Oriented Programming and Strings

WHAT IS AN OBJECT IN OOP?

An object is an instance of a class

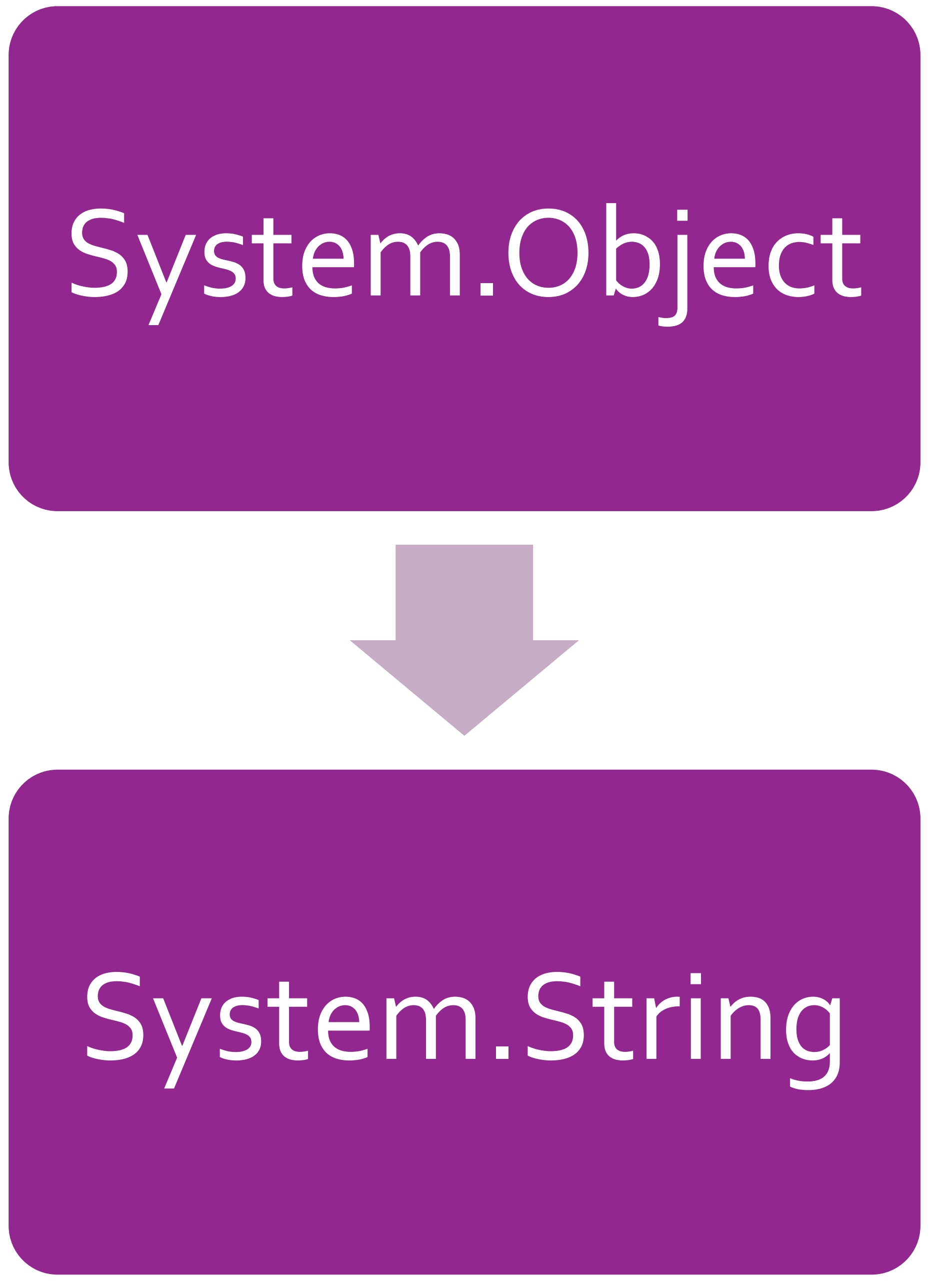
What is a class in OOP?



Classes in C#

- System.String is a class
- System.Object is a class
- System.Type is also a class
- A class is a kind of data type
- Classes describe the data and methods contained by “instances” of the class
- Instances of a class are values, known as objects
- Classes can derive from (inherit) another class (at most one)
- Classes without an explicit base class, derive from System.Object

Example of a Class Hierarchy



Variables

- All variables have a fixed type determined at compile-time
- A variable refers to an instance of a type (or null)
- Variables may be initialized when declared (best practice)
- Variables may be reassigned (use sparingly)
- Variables cannot be assigned a value of an incorrect type

Parameters and Arguments

- Function parameters are a special kind of variable (also called “formal arguments”)
- When a function is invoked the function parameters are bound to values
- Values provided during invocation to parameters are called arguments

This is indeed confusing!

The keyword “object” is a synonym for the type “System.Object”

Indicates that the accepted type of parameters, local variables, and return types is treated as a System.Object

In an object-oriented programming, all, or most, values are also called objects.

Objects and Values

- Consider: var s = “hello”;
- This is an implicitly typed variable declaration statement.
- Implicitly typed because it uses the “var” keyword
- The variable declared is named “s”
- The following is equivalent:
- string s = “hello”.

Implicitly Typed Variables

A local variable declared with the "var" keyword is implicitly typed

Requires initialization upon declaration

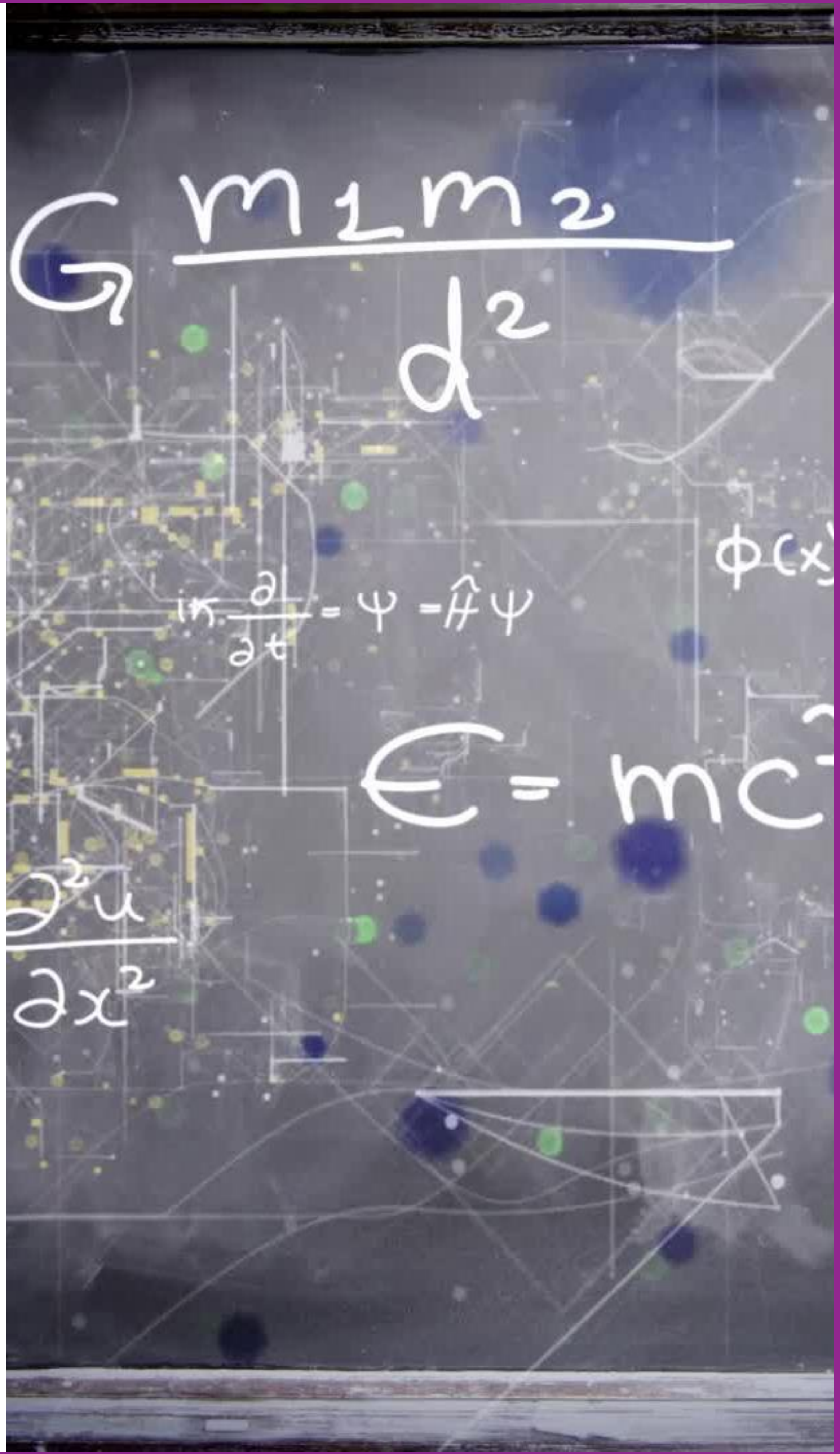
The variable takes the precise type of the expression

Casting to Object

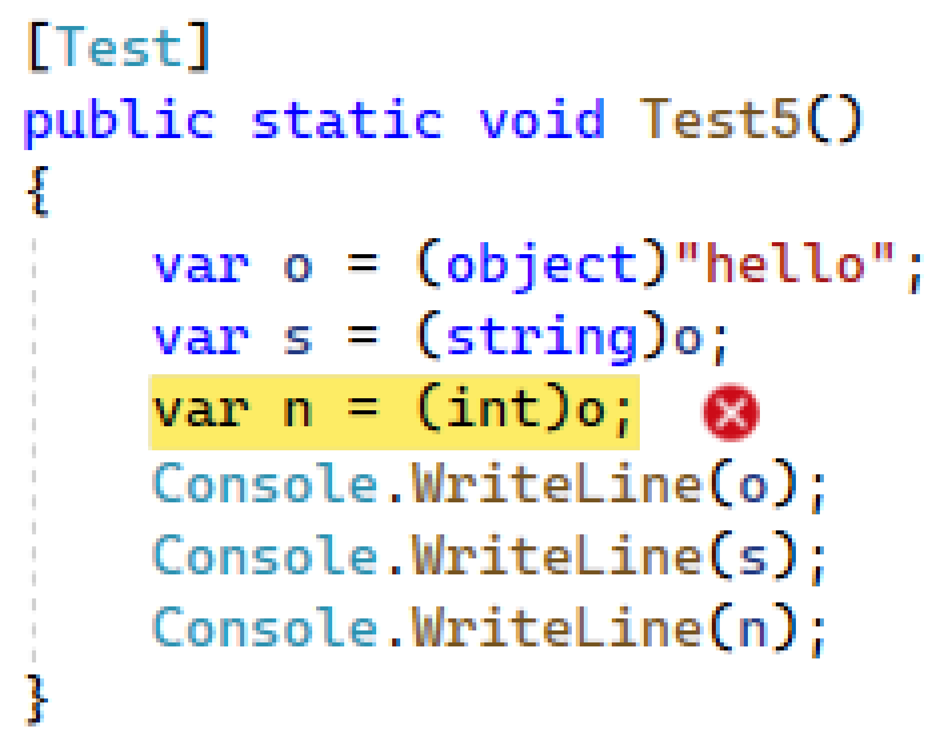
- Everything can be cast to an object:
- var x = (object)"Hello";
- This is equivalent to:
- object x = "Hello";

Upcast

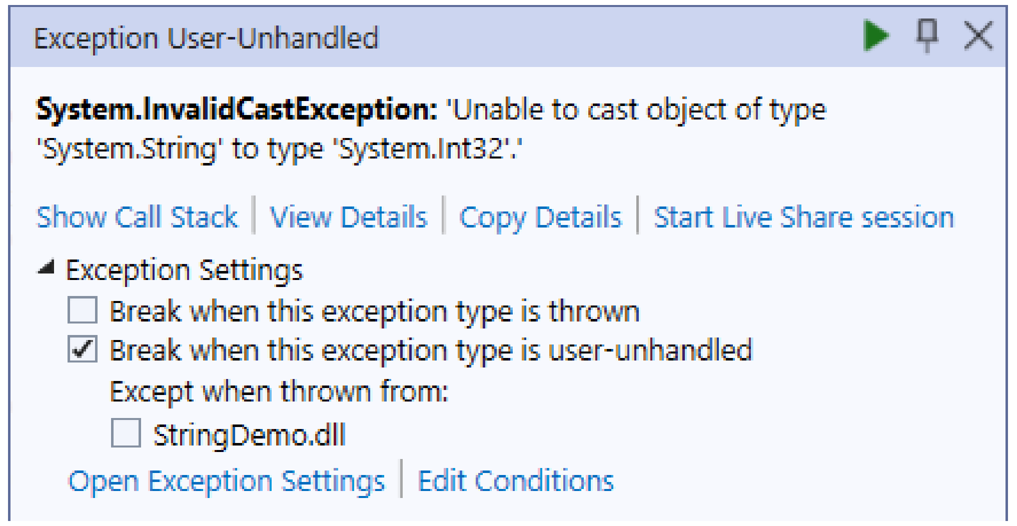
- System.String derives from System.Object
- System.Object is called the base class
- System.String is called the derived class
- Any cast to a base class is called an upcast
- Upcasts are implicit: no conversion operator is required
- They are always successful (think about why)


$G\frac{m_1m_2}{d^2}$
$E=mc^2$
$i\hbar\frac{\partial}{\partial t}\psi=\hat{H}\psi$
$\phi(x)$
$\frac{\partial^2 u}{\partial x^2}$

Downcast



```
[Test]
public static void Test5()
{
    var o = (object)"hello";
    var s = (string)o;
    var n = (int)o;
    Console.WriteLine(o);
    Console.WriteLine(s);
    Console.WriteLine(n);
}
```



Exception User-Unhandled

System.InvalidCastException: 'Unable to cast object of type 'System.String' to type 'System.Int32'.'

Show Call Stack | View Details | Copy Details | Start Live Share session

Exception Settings

- [ ] Break when this exception type is thrown
- [x] Break when this exception type is user-unhandled

Except when thrown from:

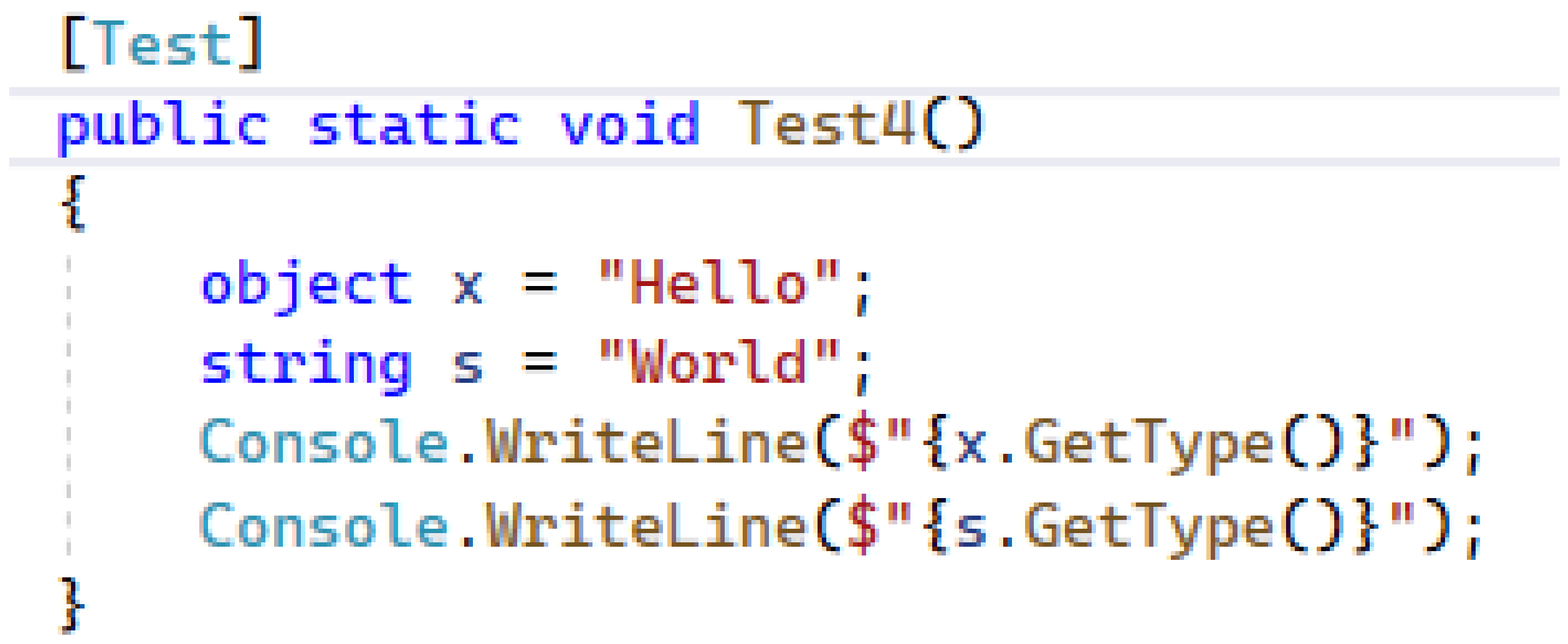
- [ ] StringDemo.dll

Open Exception Settings | Edit Conditions

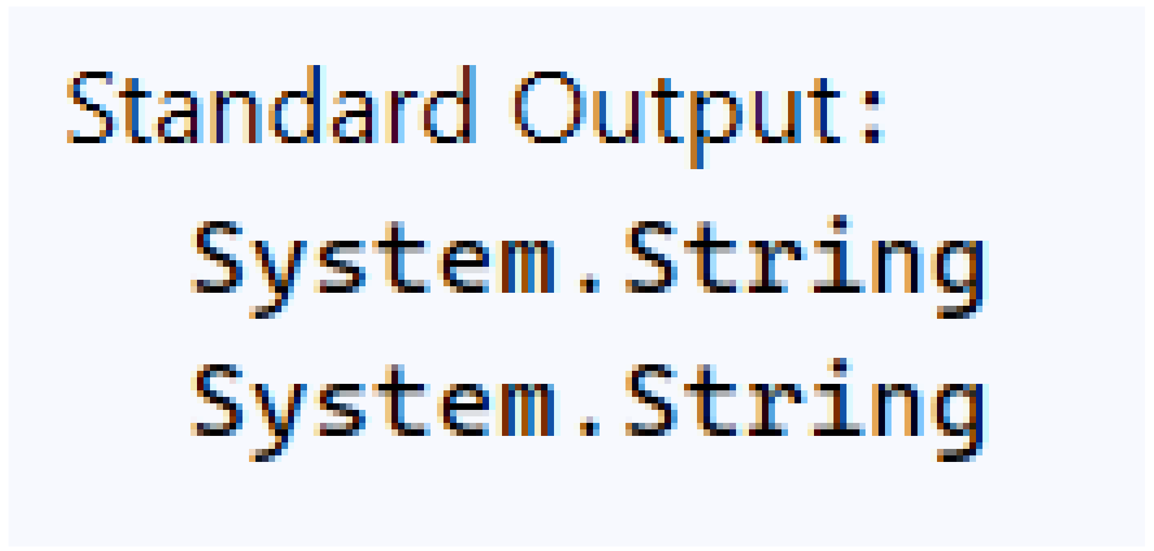
- Casting from a base class to a derived class
- System.Object to System.String is a downcast
- Downcasts are always explicit
- They may fail at run-time (think about why)

The runtime type of an object

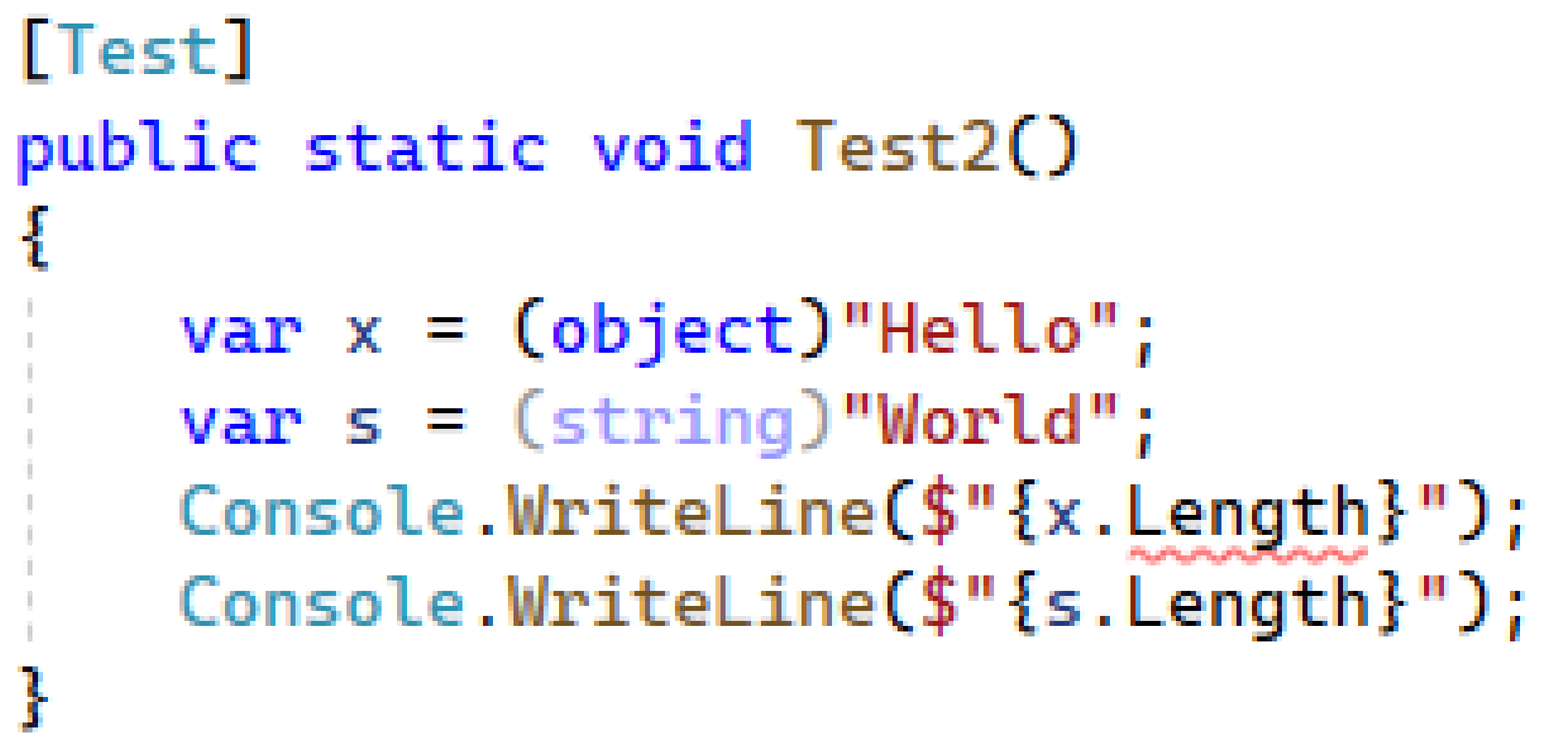
- This is the value returned by calling "GetType()"
- The run-time type of a value is unaffected by any casts
- it never changes.



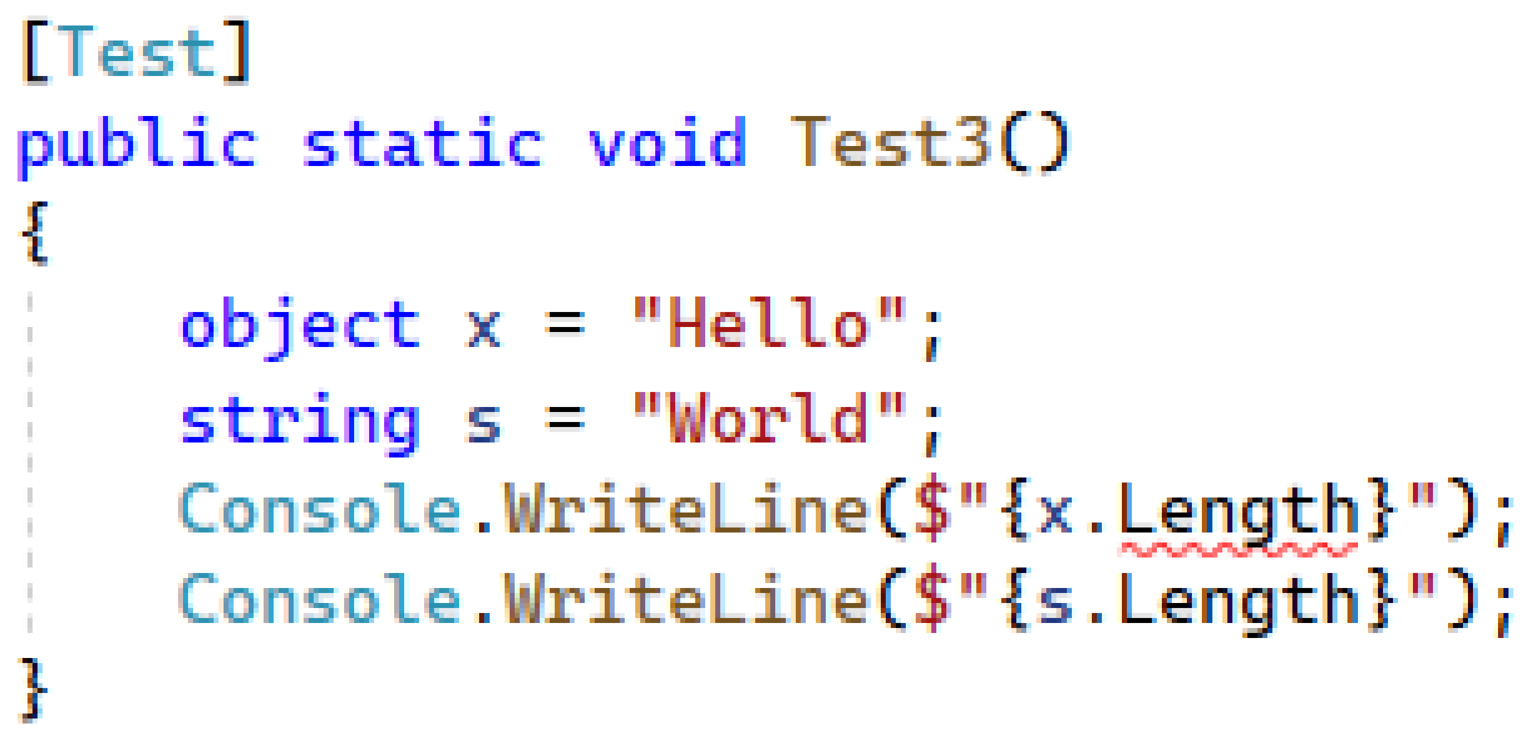
```
[Test]
public static void Test4()
{
    object x = "Hello";
    string s = "World";
    Console.WriteLine($"{x.GetType()}");
    Console.WriteLine($"{s.GetType()}");
}
```



```
Standard Output:
    System.String
    System.String
```



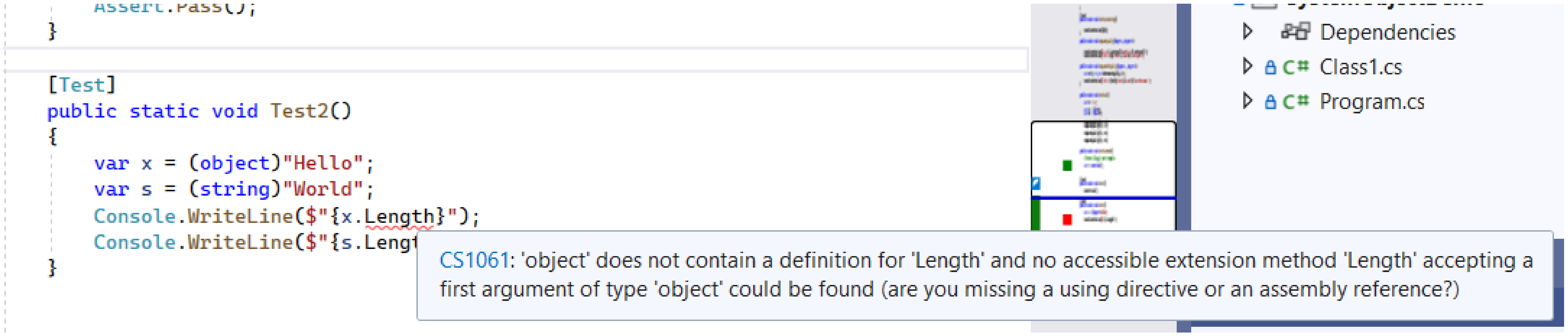
```
[Test]
public static void Test2()
{
    var x = (object)"Hello";
    var s = (string)"World";
    Console.WriteLine($"{x.Length}");
    Console.WriteLine($"{s.Length}");
}
```



```
[Test]
public static void Test3()
{
    object x = "Hello";
    string s = "World";
    Console.WriteLine($"{x.Length}");
    Console.WriteLine($"{s.Length}");
}
```

WHAT DOES THE FOLLOWING DO?

Note: they are both equivalent



```
        Assert.Pass();
    }

    [Test]
    public static void Test2()
    {
        var x = (object)"Hello";
        var s = (string)"World";
        Console.WriteLine($"{x.Length}");
        Console.WriteLine($"{s.Lengt
    }
```

CS1061: 'object' does not contain a definition for 'Length' and no accessible extension method 'Length' accepting a first argument of type 'object' could be found (are you missing a using directive or an assembly reference?)

EITHER WAY WE GET AN ERROR!

Why? Because objects don’t have “Length” properties

Types Restrict Methods & Properties

A variable or expression of type "T" only provides access to methods and properties of T.

Regardless of the run-time type of the value

What does "." mean?

- When calling a function it usually looks like "Console.WriteLine"
- What is the "."
- What is to the left?
- What is to the right?

Member Access Expression (“.”)

- Getting a member associated with a type, namespace, or object
- That member might be a field, method, property, event, type, namespace
- Left-hand side might be an expression, type, namespace
- The member might be static or not

What are Literals

Expressions that cannot be reduced

Like values embedded in code

Escape Characters

- Escape characters are specially characters in string and character literals.
- \t – Tab
- \n – Newline
- \f – Form feed
- \” – Double Quotes
- \\ - Backslash
- \o – Null character

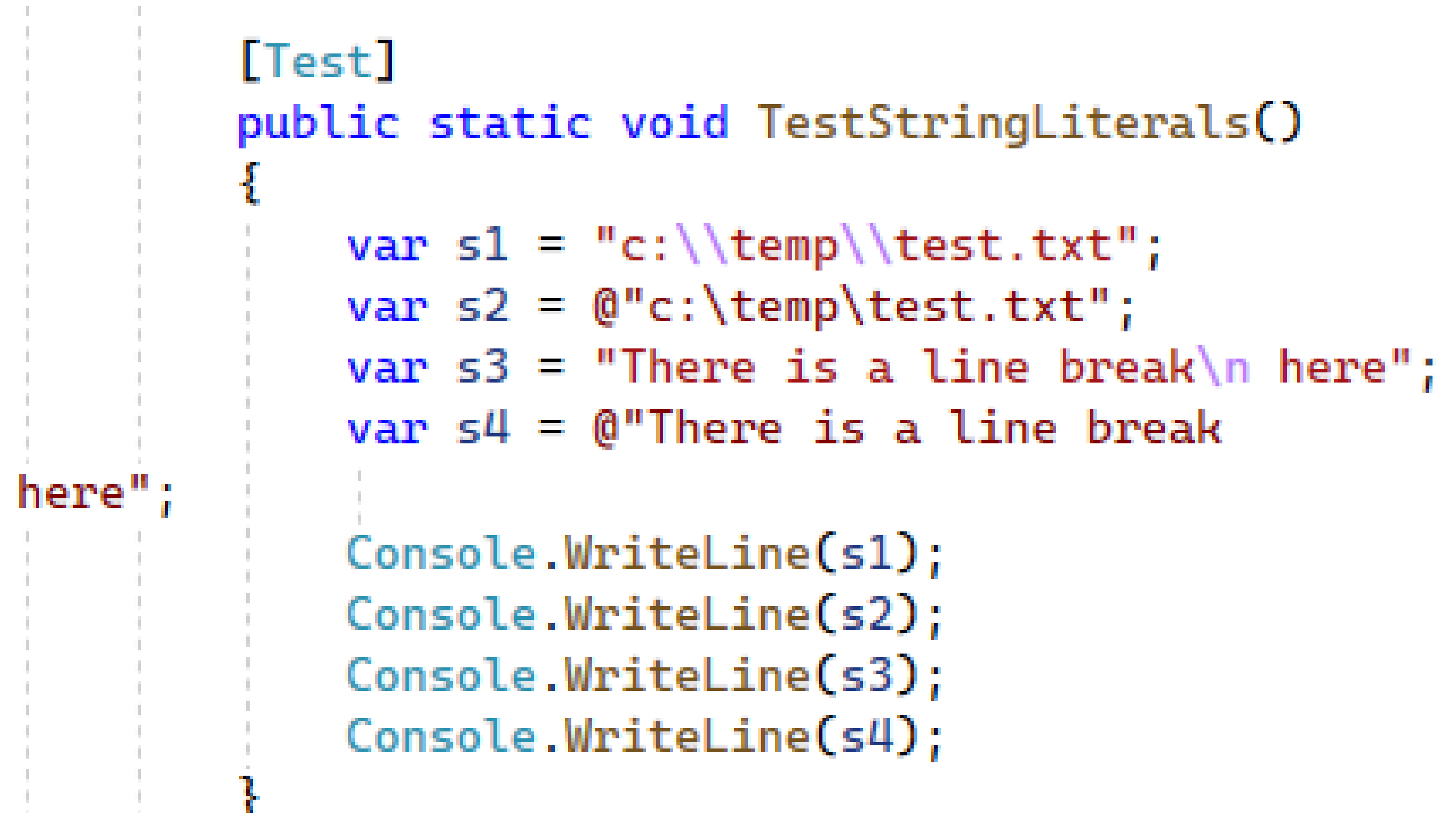
String Literals

Regular string literals:

- use escape characters
- no embedded newlines

Verbatim string literals:

- no escape characters
- embedded newlines
- Prefix with @ symbol



```
[Test]
public static void TestStringLiterals()
{
    var s1 = "c:\\temp\\test.txt";
    var s2 = @"c:\temp\test.txt";
    var s3 = "There is a line break\n here";
    var s4 = @"There is a line break
here";
    Console.WriteLine(s1);
    Console.WriteLine(s2);
    Console.WriteLine(s3);
    Console.WriteLine(s4);
}
```

Null Characters

- A C# string can contain any number of embedded null characters ('\o’).
- The null character has the ASCII code (and Unicode) of zero.
- This differs from C/C++ which uses null to indicate termination
- Not to be confused with the null keyword

Useful String Functions

01

String.IndexOf

02

String.Substring

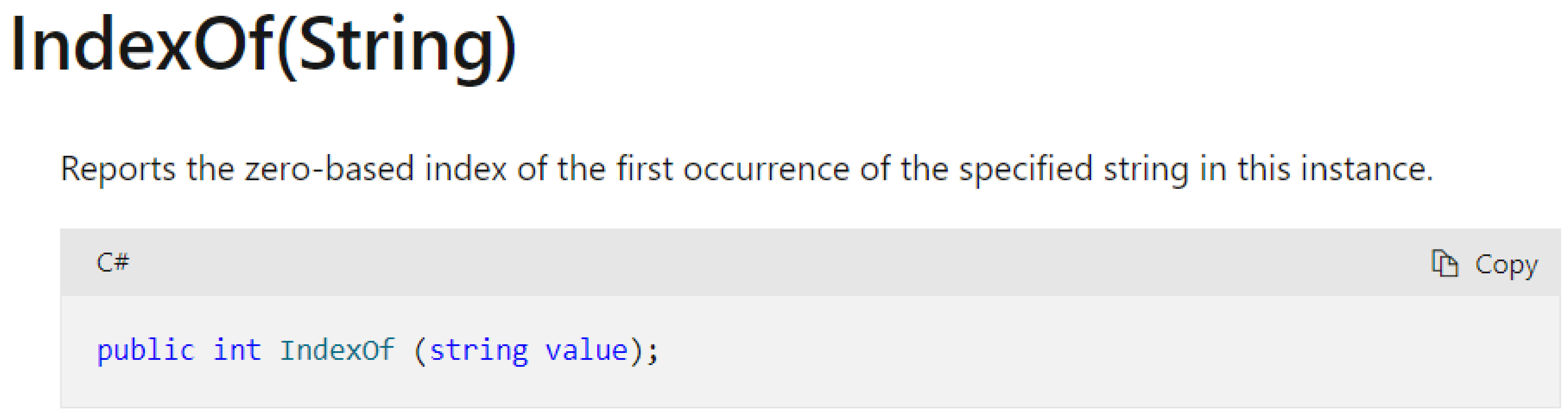
03

String.Split

04

String.Join

String.IndexOf()



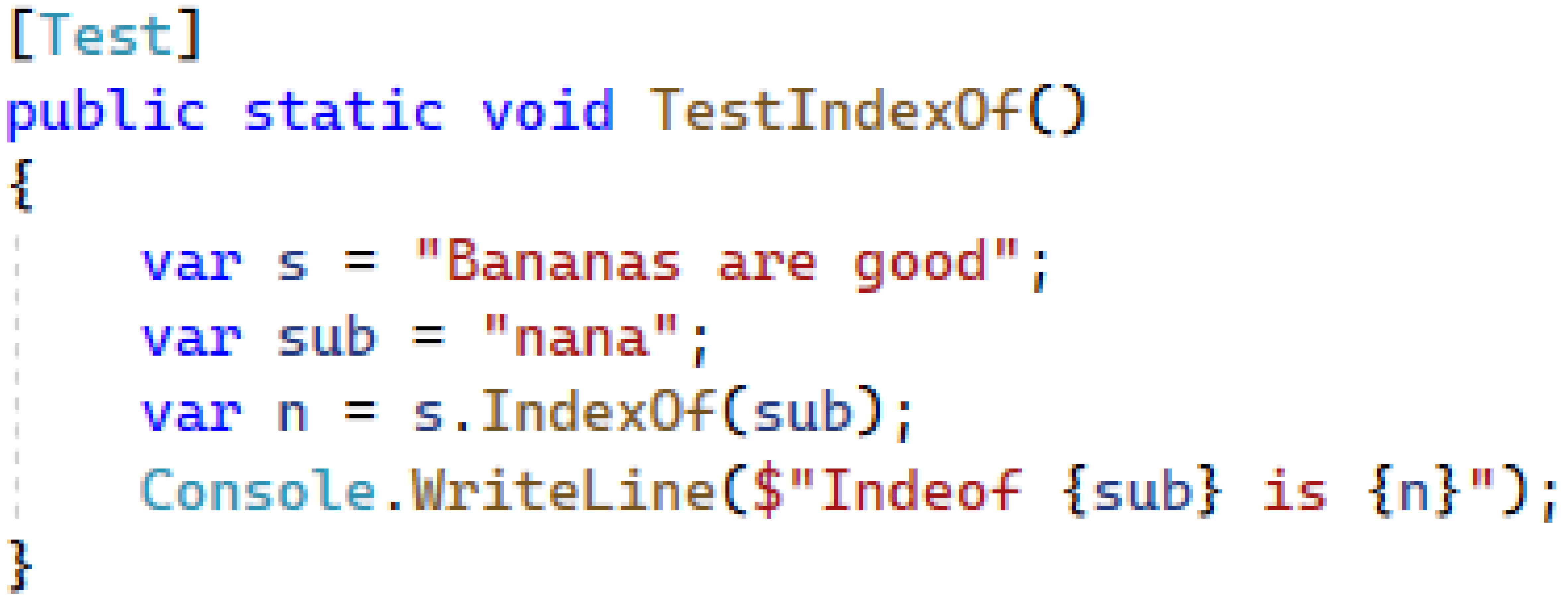
IndexOf(String)

Reports the zero-based index of the first occurrence of the specified string in this instance.

C#

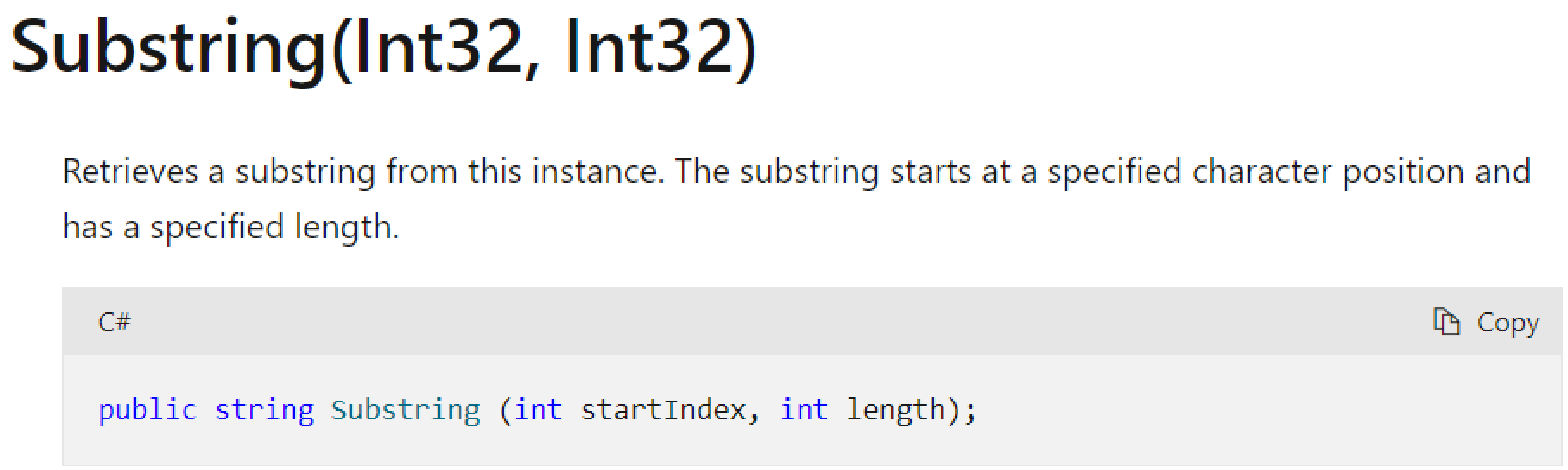
```csharp
public int IndexOf (string value);
```

String.IndexOf Demo



```
[Test]
public static void TestIndexOf()
{
    var s = "Bananas are good";
    var sub = "nana";
    var n = s.IndexOf(sub);
    Console.WriteLine($"Indeof {sub} is {n}");
}
```

String.Substring

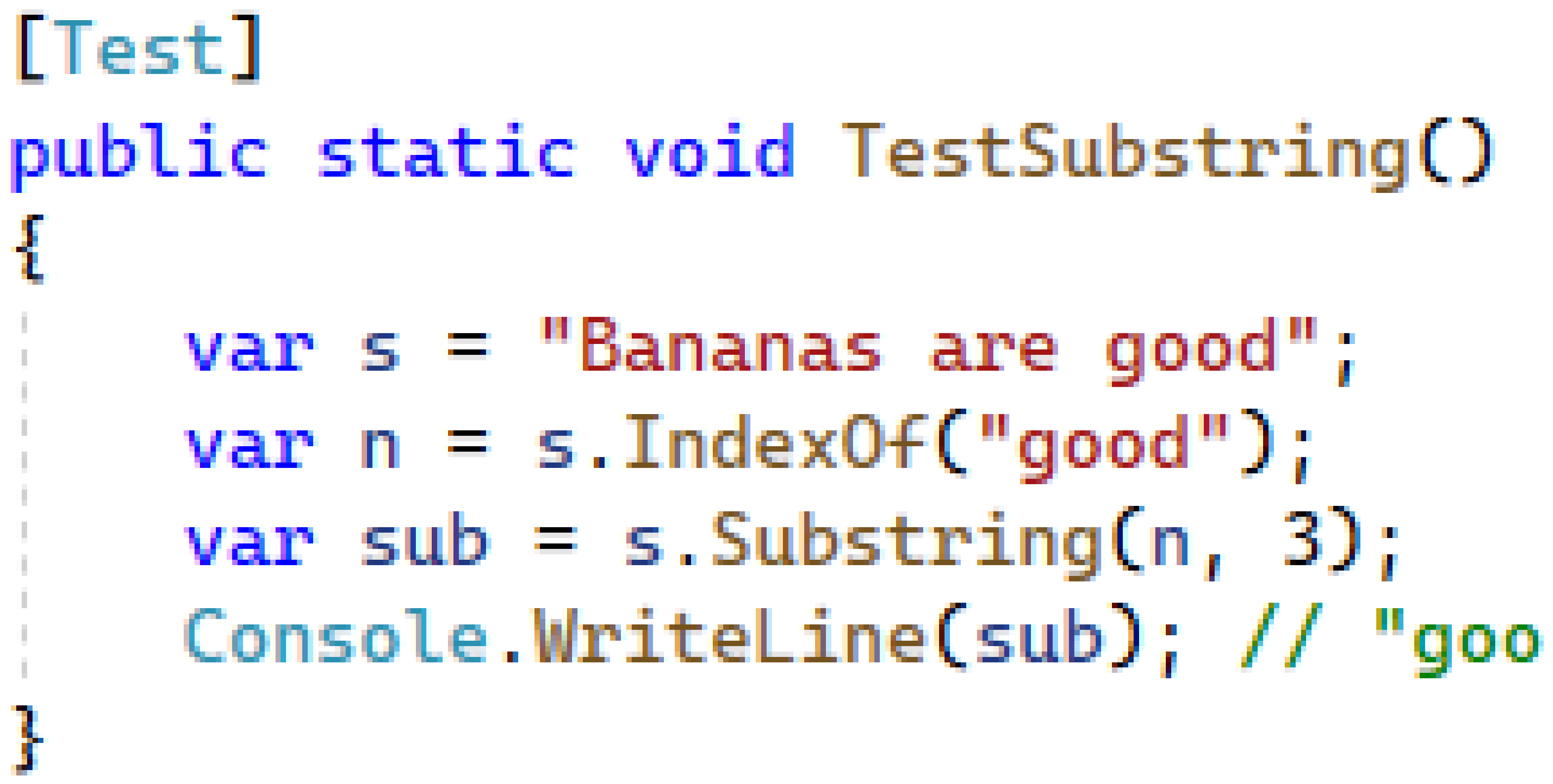


Substring(Int32, Int32)

Retrieves a substring from this instance. The substring starts at a specified character position and has a specified length.

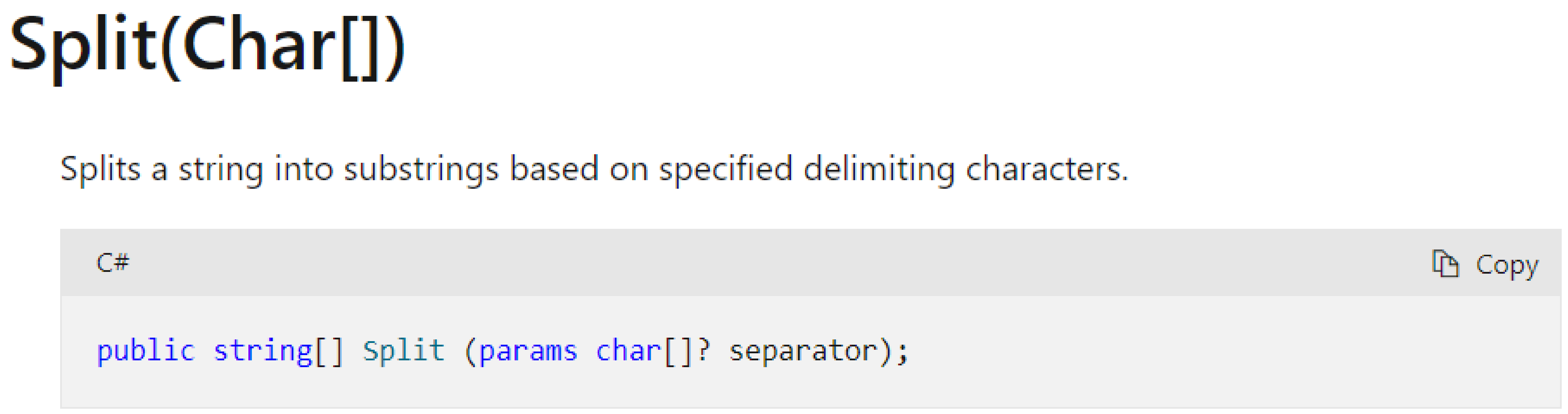
```csharp
public string Substring (int startIndex, int length);
```

String.Substring Demo



```
[Test]
public static void TestSubstring()
{
    var s = "Bananas are good";
    var n = s.IndexOf("good");
    var sub = s.Substring(n, 3);
    Console.WriteLine(sub); // "goo
}
```

String.Split



Split(Char[])

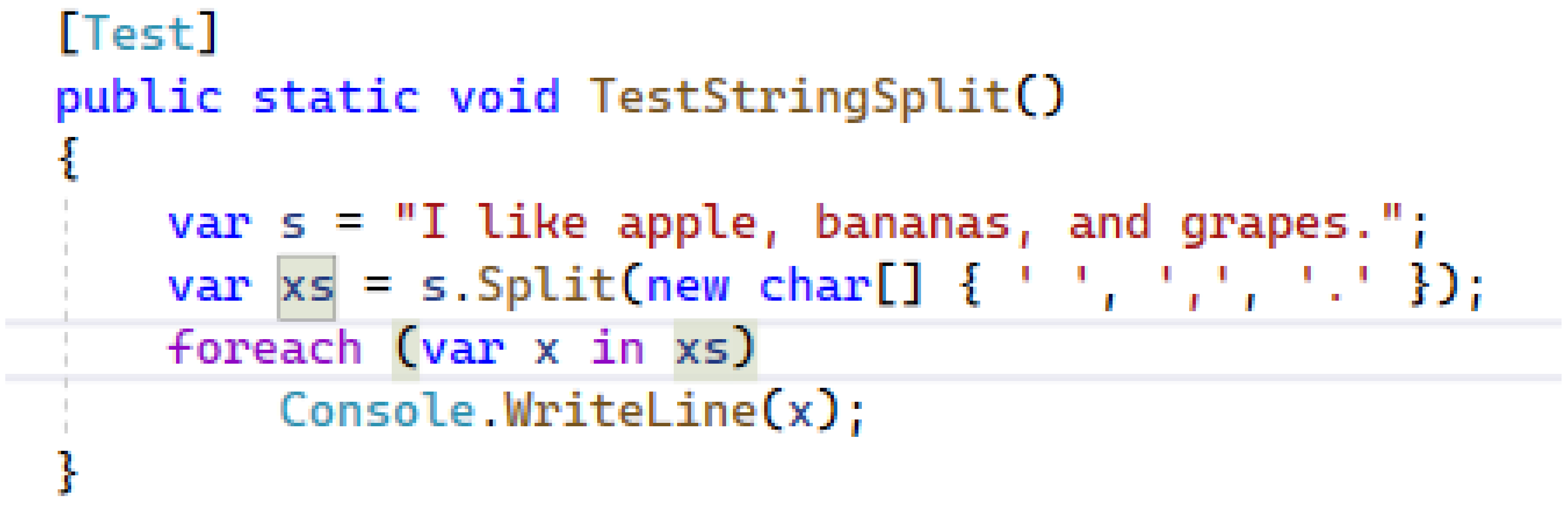
Splits a string into substrings based on specified delimiting characters.

C#

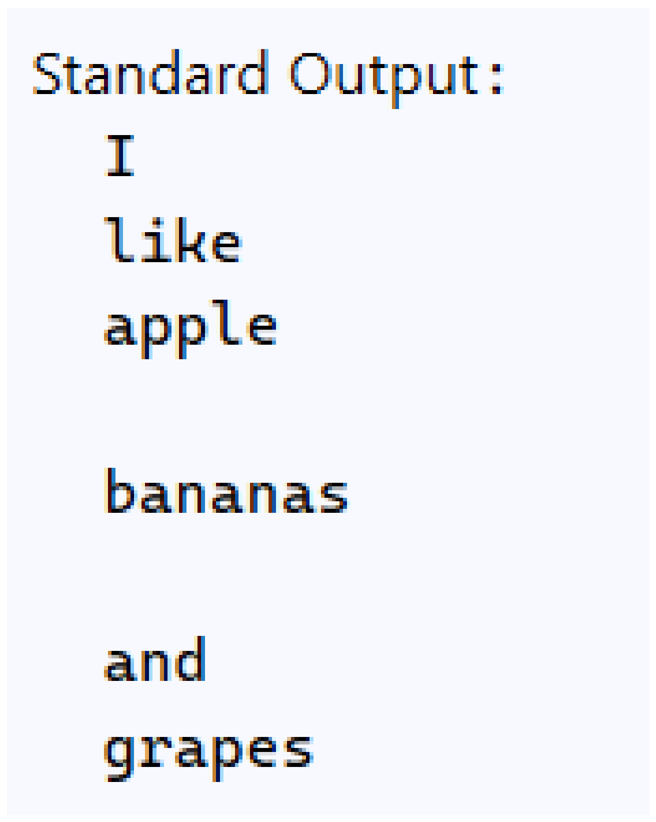
Copy

```csharp
public string[] Split (params char[]? separator);
```

String.Split() Demo



```
[Test]
public static void TestStringSplit()
{
    var s = "I like apple, bananas, and grapes.";
    var xs = s.Split(new char[] { ' ', ',', '.' });
    foreach (var x in xs)
        Console.WriteLine(x);
}
```



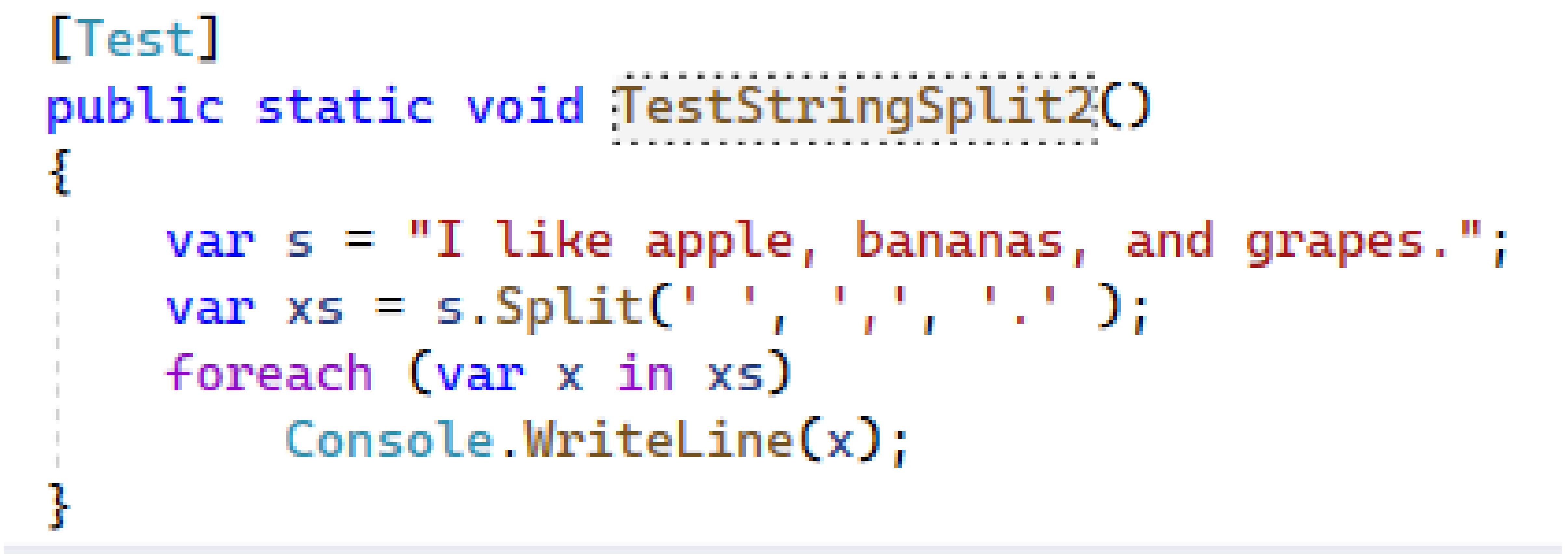
```
Standard Output:
  I
  like
  apple

  bananas

  and
  grapes
```

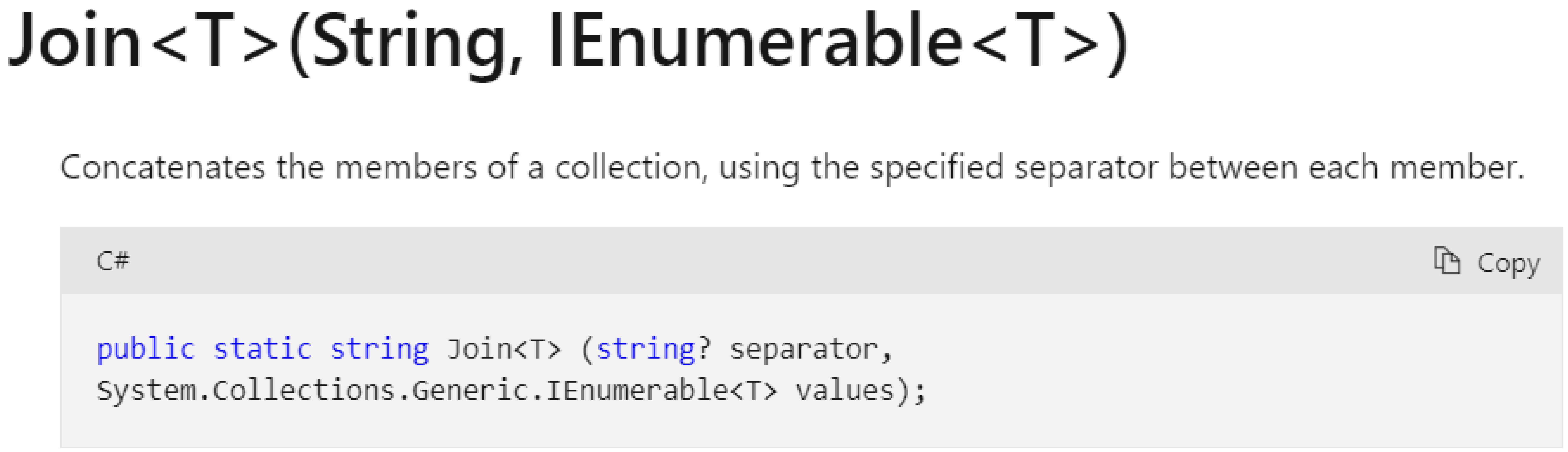
Params: Variable Length Arguments

The params keywords means that I can do this instead as well: notice no array.



```
[Test]
public static void TestStringSplit2()
{
    var s = "I like apple, bananas, and grapes.";
    var xs = s.Split(' ', ',', '.' );
    foreach (var x in xs)
        Console.WriteLine(x);
}
```

String.Join



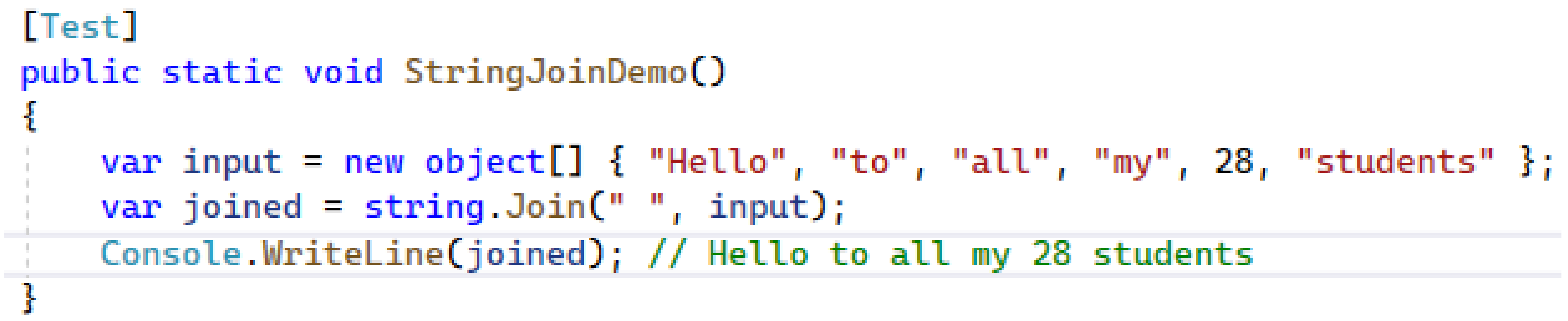
Join<T>(String, IEnumerable<T>)

Concatenates the members of a collection, using the specified separator between each member.

C#

```csharp
public static string Join<T> (string? separator,
System.Collections.Generic.IEnumerable<T> values);
```

String.Join Demo



```csharp
[Test]
public static void StringJoinDemo()
{
    var input = new object[] { "Hello", "to", "all", "my", 28, "students" };
    var joined = string.Join(" ", input);
    Console.WriteLine(joined); // Hello to all my 28 students
}
```

Invoking Instance versus Static Methods

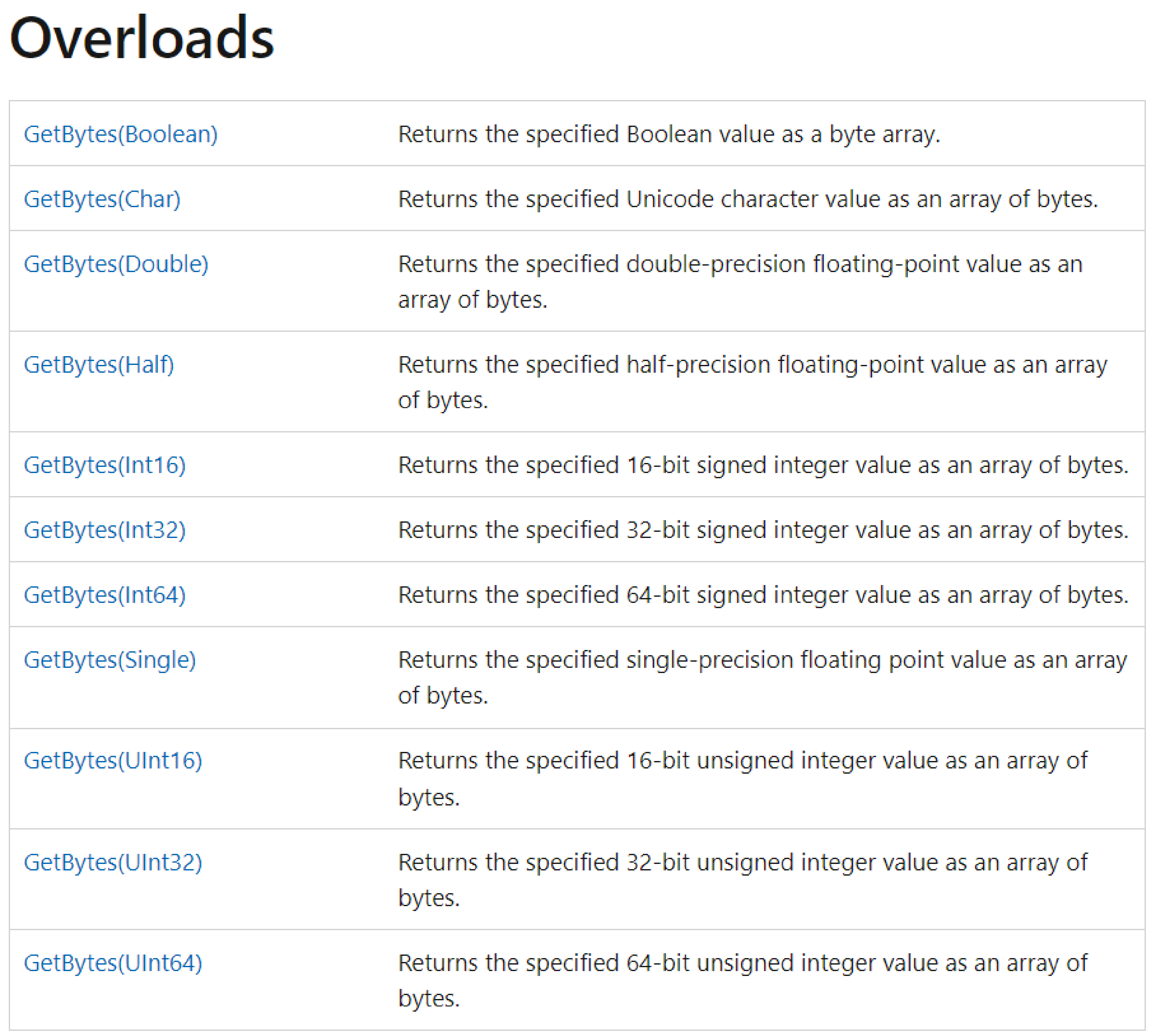
Instance methods have the form “expression.FunctionName(args)”

Static method have the form “typename.FunctionName(args)”

WHY IS STRING.JOIN STATIC?

CONVERTING BYTES TO/FROM STRINGS

BIT CONVERTER DOES NOT WORK ON STRINGS?!



Overloads

GetBytes(Boolean)	Returns the specified Boolean value as a byte array.
GetBytes(Char)	Returns the specified Unicode character value as an array of bytes.
GetBytes(Double)	Returns the specified double-precision floating-point value as an array of bytes.
GetBytes(Half)	Returns the specified half-precision floating-point value as an array of bytes.
GetBytes(Int16)	Returns the specified 16-bit signed integer value as an array of bytes.
GetBytes(Int32)	Returns the specified 32-bit signed integer value as an array of bytes.
GetBytes(Int64)	Returns the specified 64-bit signed integer value as an array of bytes.
GetBytes(Single)	Returns the specified single-precision floating point value as an array of bytes.
GetBytes(UInt16)	Returns the specified 16-bit unsigned integer value as an array of bytes.
GetBytes(UInt32)	Returns the specified 32-bit unsigned integer value as an array of bytes.
GetBytes(UInt64)	Returns the specified 64-bit unsigned integer value as an array of bytes.

string value = "\u00C4 \uD802\u0033 \u00AE"; byte[] bytes= System.Text.Encoding.UTF8.GetBytes(value);
System.Text.Encoding.UTF8.GetBytes(value);

Remember Encodings?

We need to choose one, such as

System.Text.Encoding.UTF8.GetBytes()

System.Text.Encoding.UTF16.GetBytes()

System.Text.Encoding.ASCIIEncoding.GetBytes()

String Constructors

String(Char, Int32) - Initializes a new instance of the String class to the value indicated by a specified Unicode character repeated a specified number of times.

String(Char[]) - Initializes a new instance of the String class to the Unicode characters indicated in the specified character array.

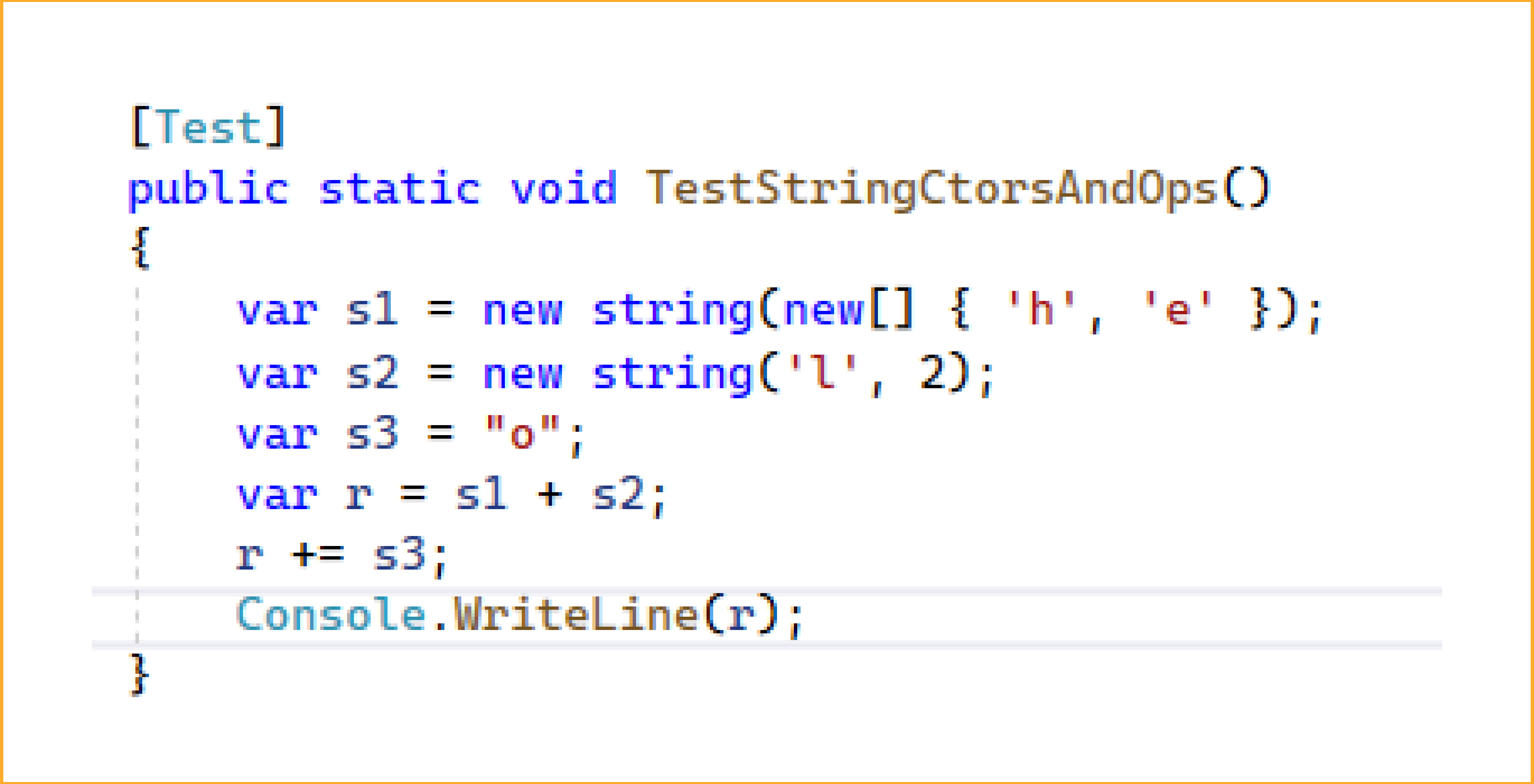
String Operators

+ String concatenation

+= String concatenation and assignment

== Equality

!= Inequality

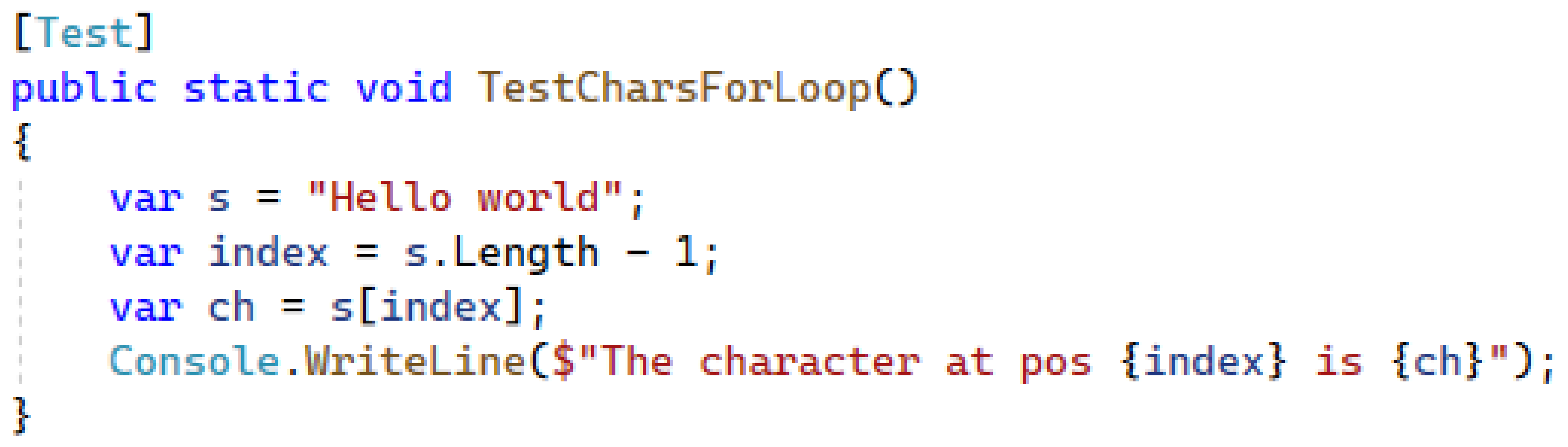


```
[Test]
public static void TestStringCtorsAndOps()
{
    var s1 = new string(new[] { 'h', 'e' });
    var s2 = new string('l', 2);
    var s3 = "o";
    var r = s1 + s2;
    r += s3;
    Console.WriteLine(r);
}
```

Strings are Like Arrays

- They have a Length property
- They support indexing using an integer index
- In other words you can get the nth character using a subscript

Demo String Length and Indexing



```
[Test]
public static void TestCharsForLoop()
{
    var s = "Hello world";
    var index = s.Length - 1;
    var ch = s[index];
    Console.WriteLine($"The character at pos {index} is {ch}");
}
```

Wait, what is a property?

A member that resembles a field

May redirect to a field or to a function

May be read-only or read-write

May be static or non-static

String Immutability

String objects are immutable: they can't be changed after they've been created

Methods and C# operators either query a string or create a new string object

So how do you build strings?

- StringBuilder class
- String.Format
- String interpolation expression
- Concatenation
- From an array of chars

What about memory?

- Do we care?
- If two strings return “equal” and have the same hash-code?
- They are effectively equal
- You could call “Object.ReferenceEquals()”, but don’t.

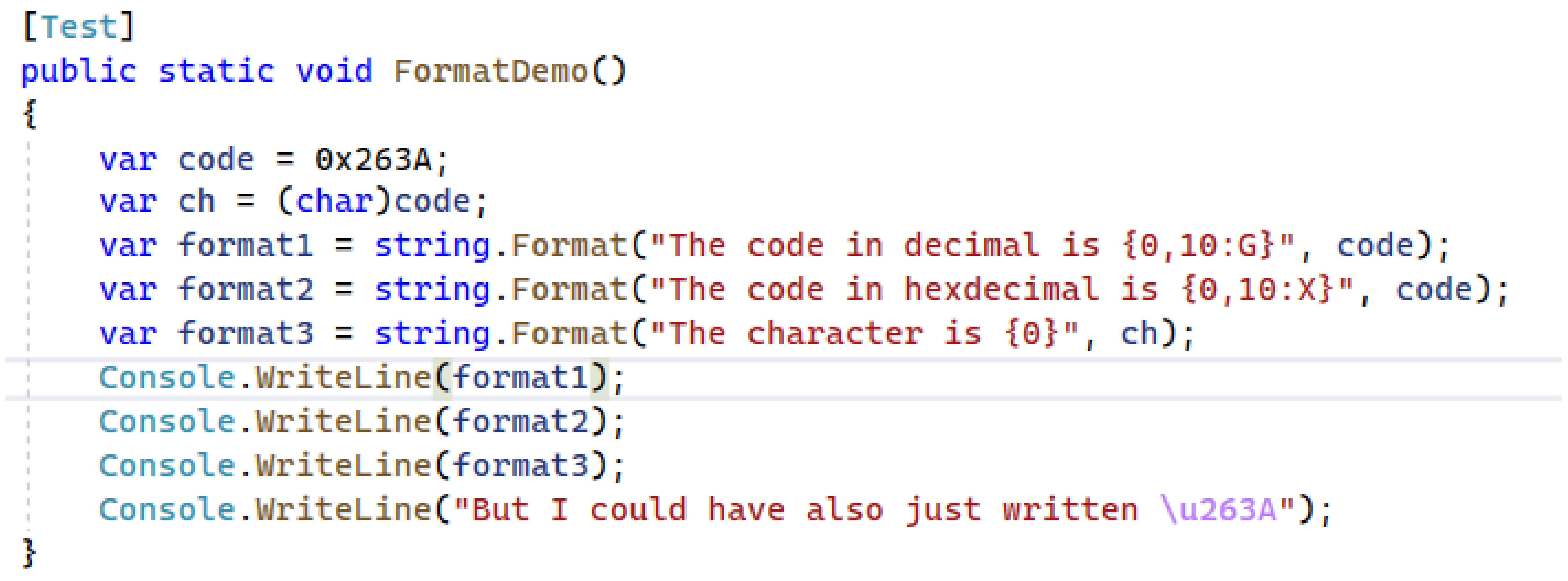
Indexers

- An indexer allows a type instance to be indexed like an array or dictionary
- An indexer can accept any type of parameters (like an int, string, object.)

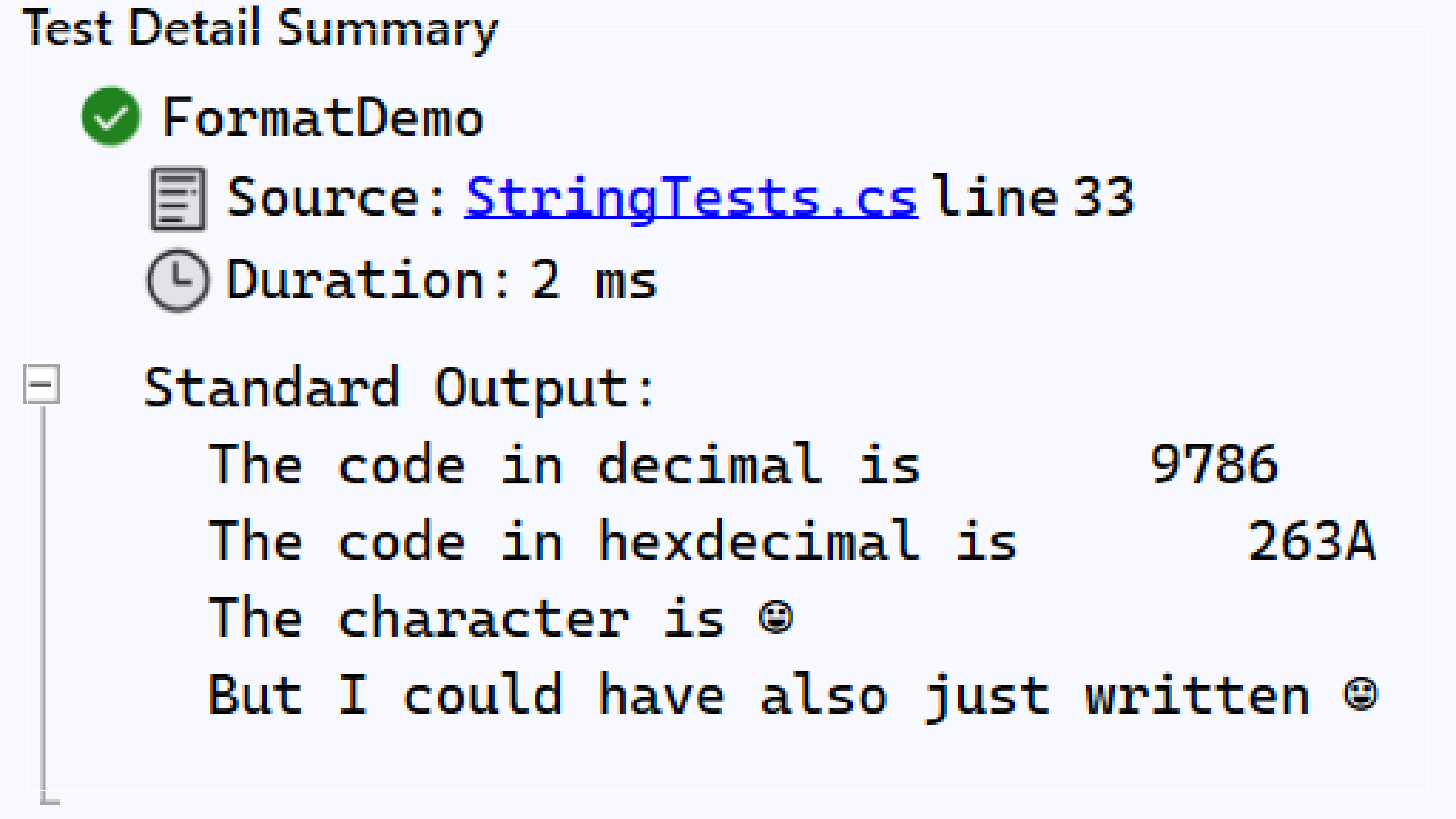
String Formatting

- Before string interpolation we had string formatting routines
- Like a safe and powerful version of the C function sprintf().
- String.Format()

String Formatting



```
[Test]
public static void FormatDemo()
{
    var code = 0x263A;
    var ch = (char)code;
    var format1 = string.Format("The code in decimal is {0,10:G}", code);
    var format2 = string.Format("The code in hexdecimal is {0,10:X}", code);
    var format3 = string.Format("The character is {0}", ch);
    Console.WriteLine(format1);
    Console.WriteLine(format2);
    Console.WriteLine(format3);
    Console.WriteLine("But I could have also just written \u263A");
}
```


Test Detail Summary

✓ FormatDemo

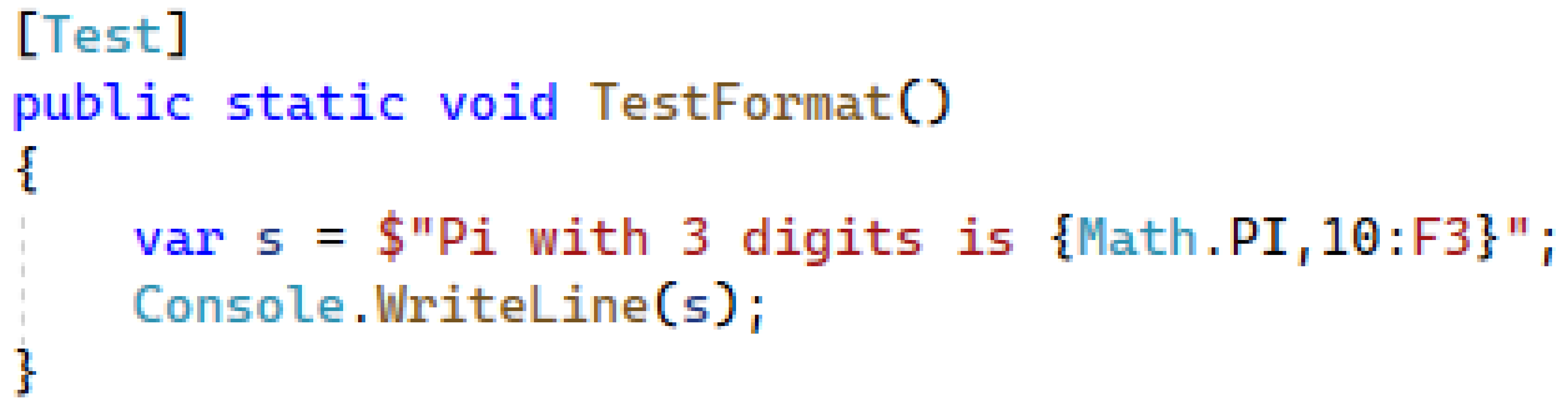
Source: StringTests.cs line 33

Duration: 2 ms

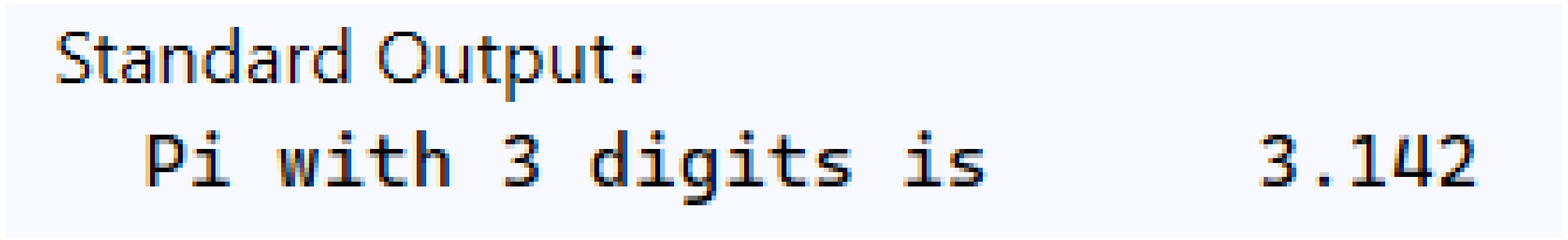
Standard Output:

```
The code in decimal is        9786
The code in hexdecimal is        263A
The character is ☺
But I could have also just written ☺
```

Formatting with String interpolation



```csharp
[Test]
public static void TestFormat()
{
    var s = $"Pi with 3 digits is {Math.PI,10:F3}";
    Console.WriteLine(s);
}
```



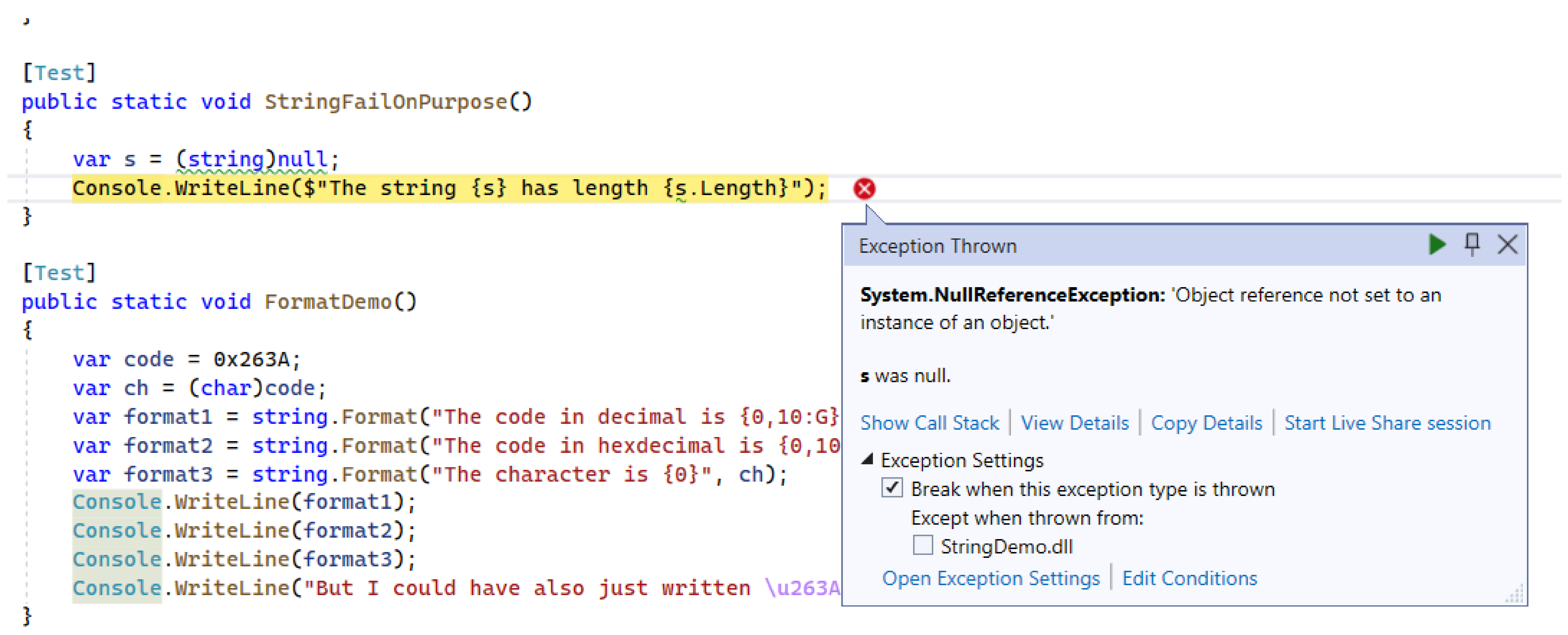
Standard Output:

```
Pi with 3 digits is      3.142
```

The Null Literal

- The null keyword represents a reference that does not refer to an object.
- It has a special type (called the null type) but can be cast to any reference type
- Reference variables are assigned null by default
- In other words it means “no value”
- Different from the empty string (“”)

NullReferenceException



```
,

[Test]
public static void StringFailOnPurpose()
{
    var s = (string)null;
    Console.WriteLine($"The string {s} has length {s.Length}");
}

[Test]
public static void FormatDemo()
{
    var code = 0x263A;
    var ch = (char)code;
    var format1 = string.Format("The code in decimal is {0,10:G}
    var format2 = string.Format("The code in hexdecimal is {0,10
    var format3 = string.Format("The character is {0}", ch);
    Console.WriteLine(format1);
    Console.WriteLine(format2);
    Console.WriteLine(format3);
    Console.WriteLine("But I could have also just written \u263A
}
```

Exception Thrown

System.NullReferenceException: 'Object reference not set to an instance of an object.'

s was null.

Show Call Stack | View Details | Copy Details | Start Live Share session

Exception Settings

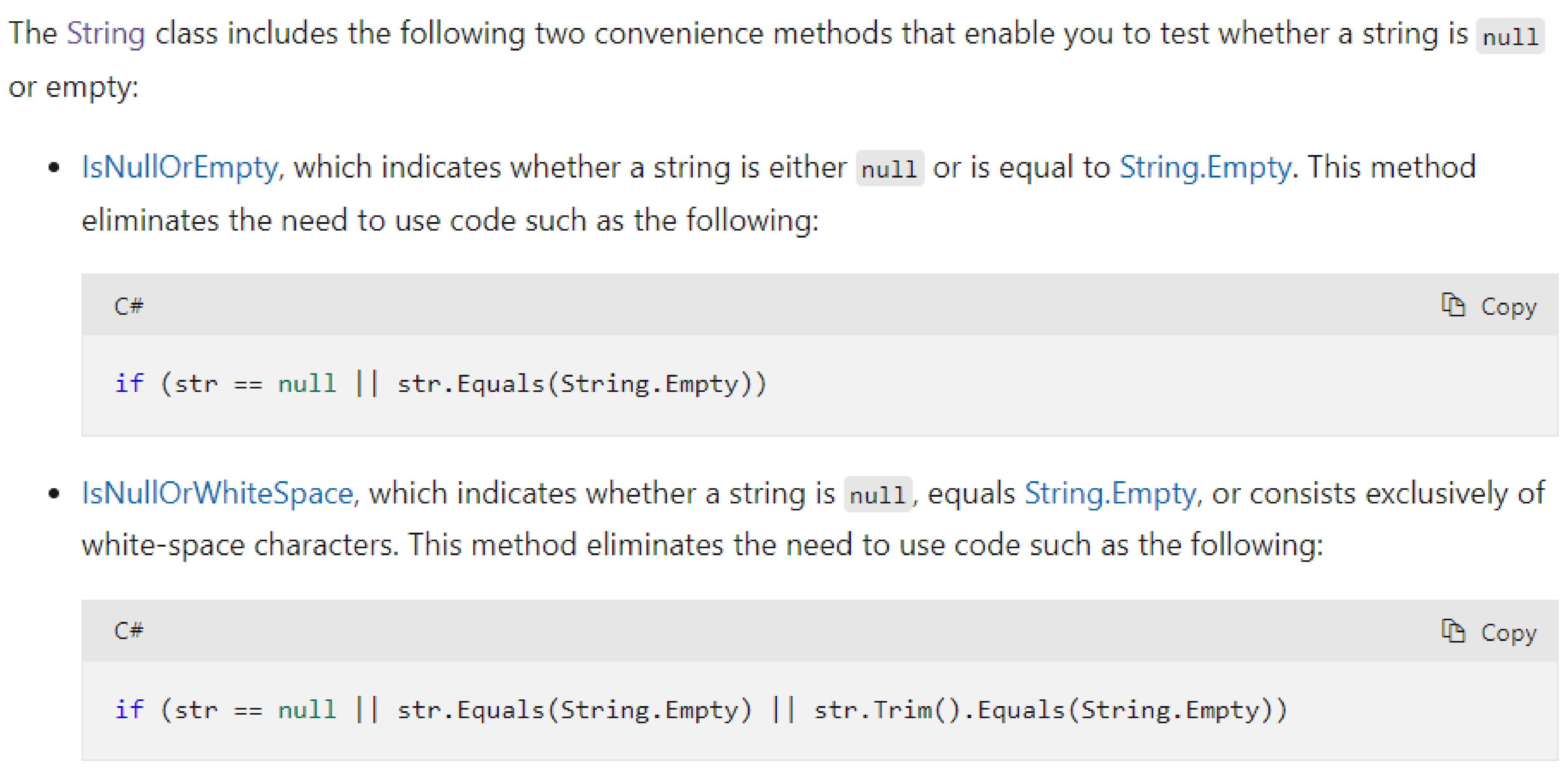
[✓] Break when this exception type is thrown

Except when thrown from:

[ ] StringDemo.dll

Open Exception Settings | Edit Conditions

Checking if Strings are Null or Empty



The String class includes the following two convenience methods that enable you to test whether a string is `null` or empty:

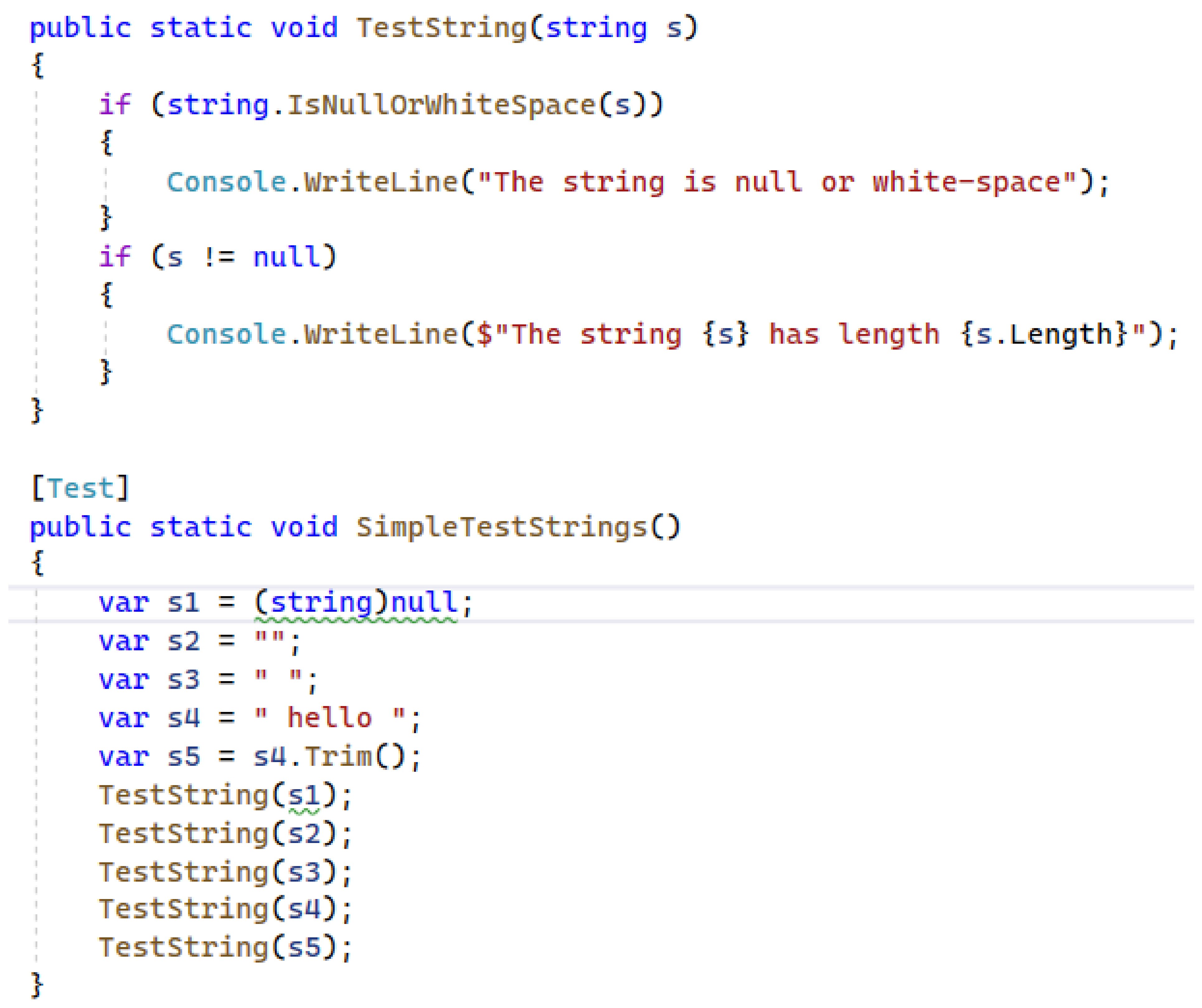
- IsNullOrEmpty, which indicates whether a string is either `null` or is equal to String.Empty. This method eliminates the need to use code such as the following:

  ```csharp
  if (str == null || str.Equals(String.Empty))
  ```

- IsNullOrWhiteSpace, which indicates whether a string is `null`, equals String.Empty, or consists exclusively of white-space characters. This method eliminates the need to use code such as the following:

  ```csharp
  if (str == null || str.Equals(String.Empty) || str.Trim().Equals(String.Empty))
  ```

STRING QUERIES



```
public static void TestString(string s)
{
    if (string.IsNullOrWhiteSpace(s))
    {
        Console.WriteLine("The string is null or white-space");
    }
    if (s != null)
    {
        Console.WriteLine($"The string {s} has length {s.Length}");
    }
}

[Test]
public static void SimpleTestStrings()
{
    var s1 = (string)null;
    var s2 = "";
    var s3 = " ";
    var s4 = " hello ";
    var s5 = s4.Trim();
    TestString(s1);
    TestString(s2);
    TestString(s3);
    TestString(s4);
    TestString(s5);
}
```

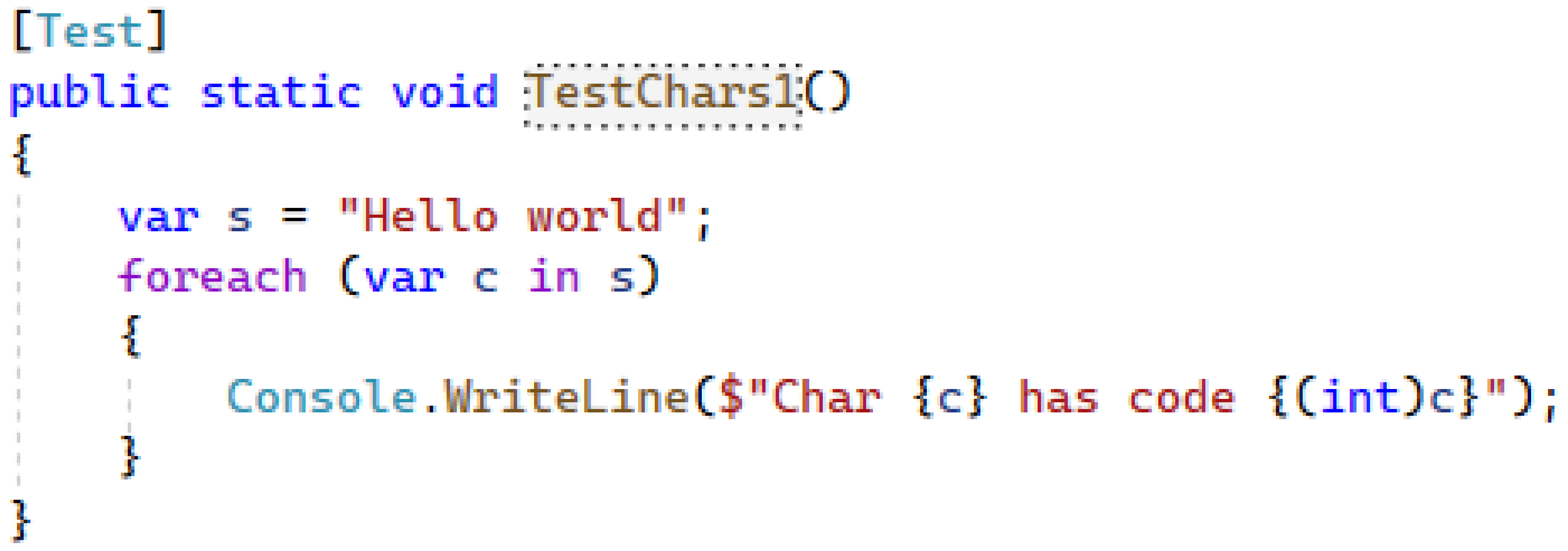
Strings implement IEnumerable

Strings implement “IEnumerable”

→

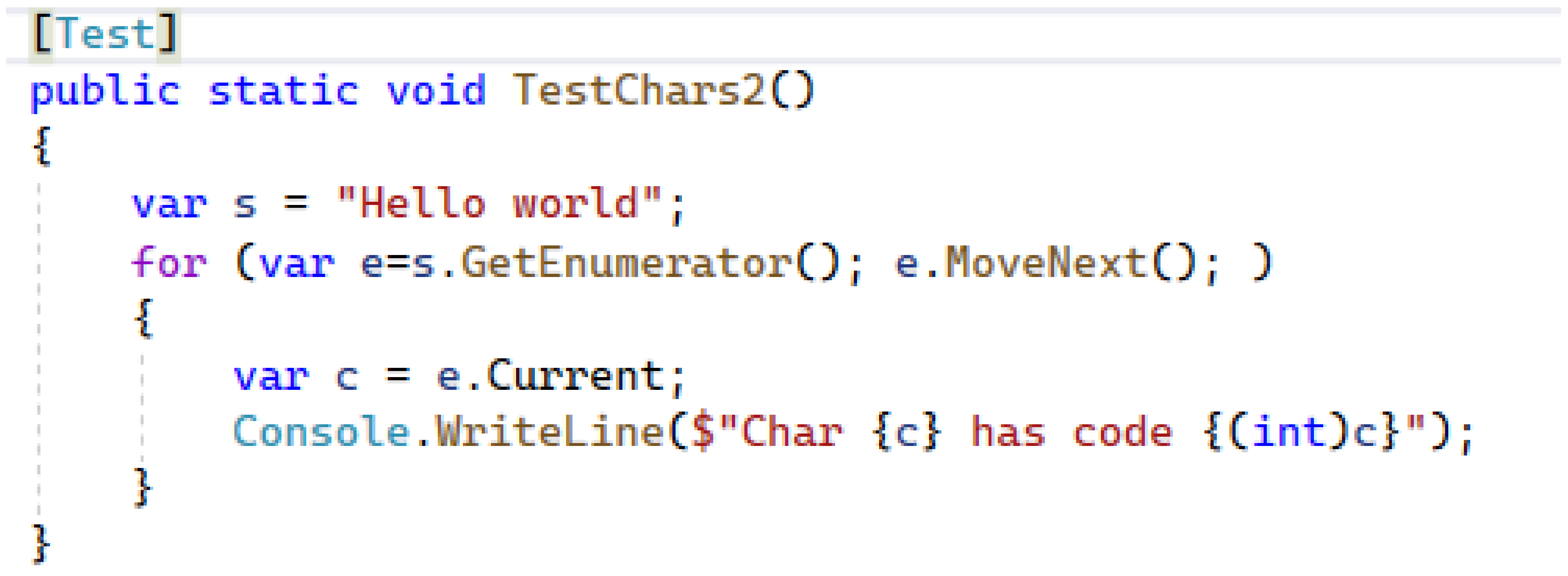
This means you can loop through the characters with a foreach

Foreach



```
[Test]
public static void TestChars1()
{
    var s = "Hello world";
    foreach (var c in s)
    {
        Console.WriteLine($"Char {c} has code {(int)c}");
    }
}
```

Foreach is a For Loop



```
[Test]
public static void TestChars2()
{
    var s = "Hello world";
    for (var e=s.GetEnumerator(); e.MoveNext(); )
    {
        var c = e.Current;
        Console.WriteLine($"Char {c} has code {(int)c}");
    }
}
```

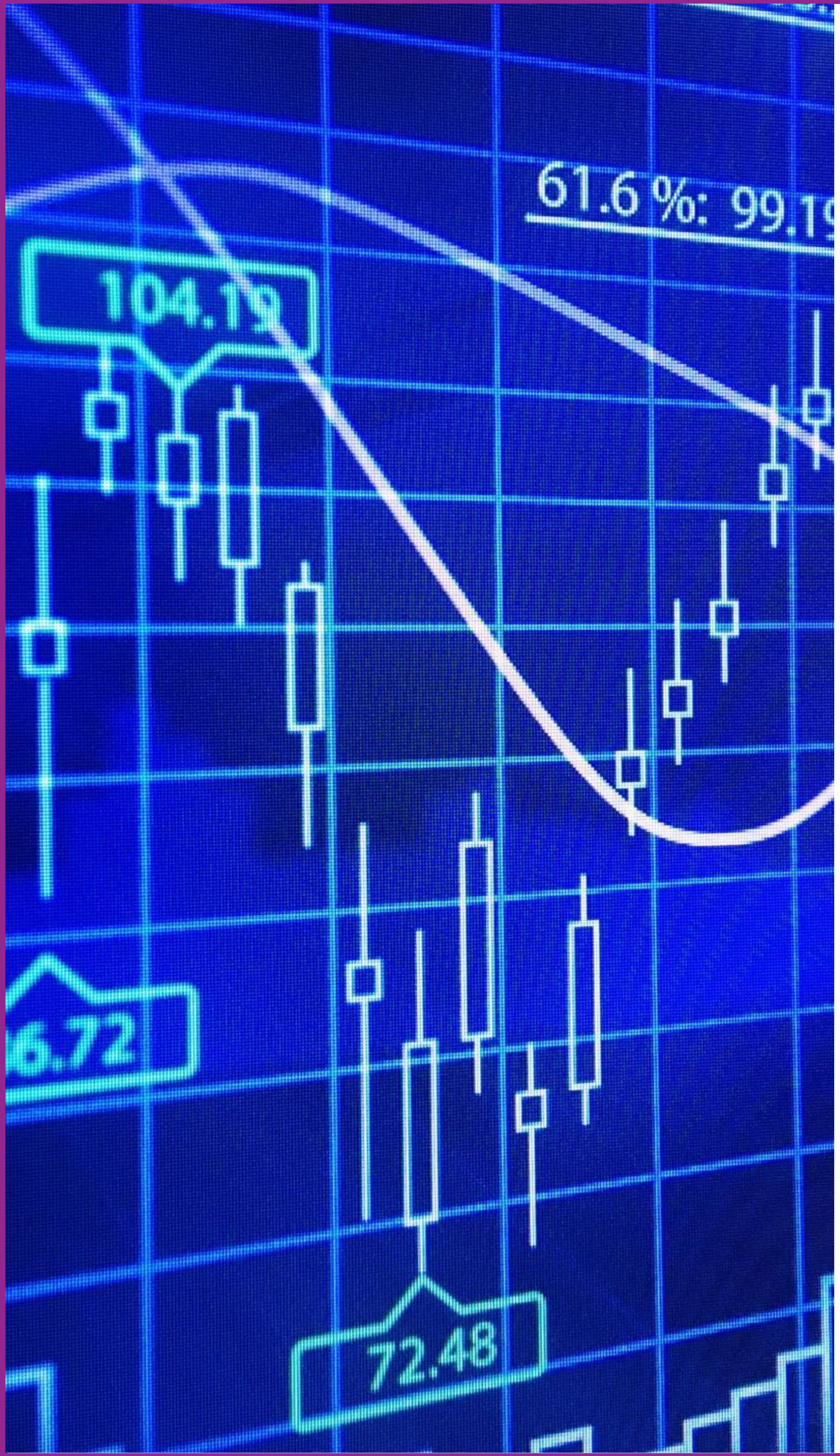
https://learn.microsoft.com/en-us/dotnet/api/system.collections.ienumerable.getenumerator?view=net-7.0

Interview Questions with Strings

- Get all duplicated characters in a string.
- Get all unique characters in a string.
- Reverse a string.
- Reverse each word in a string
- Get the word count in a string
- Check if a string is a palindrome or not
- Check max occurrence of a character in the string.
- Get all possible substring in a string.
- Get the first char of each word in capital letter
- Check if two strings are anagrams
- Remove duplicated characters
- Check if a function has all unique characters

Review

- Do all variables have types?
- How is the type of an implicitly typed variable declaration determined?
- What does the “var” keyword indicate?
- Can I access methods specific to a string (like Length) on a variable of type “object”?
- Can a variable of type “object” refer to a “string” object?
- How can I determine the run-time type of an object?
- What is an instance of a class called?
- What does System.String inherit from?
- Casting from System.String to System.Object is an upcast or downcast?
- Are upcasts explicit or implicit?
- Can I change the type of a value?
- Are types valid expressions?



Next Class

- **Collections**
- **Building our First Class**