

# TODAY'S GOAL

Deepen our understanding of C# by diving into strings and characters

# Goal Breakdown



Become more familiar with C# syntax and semantics



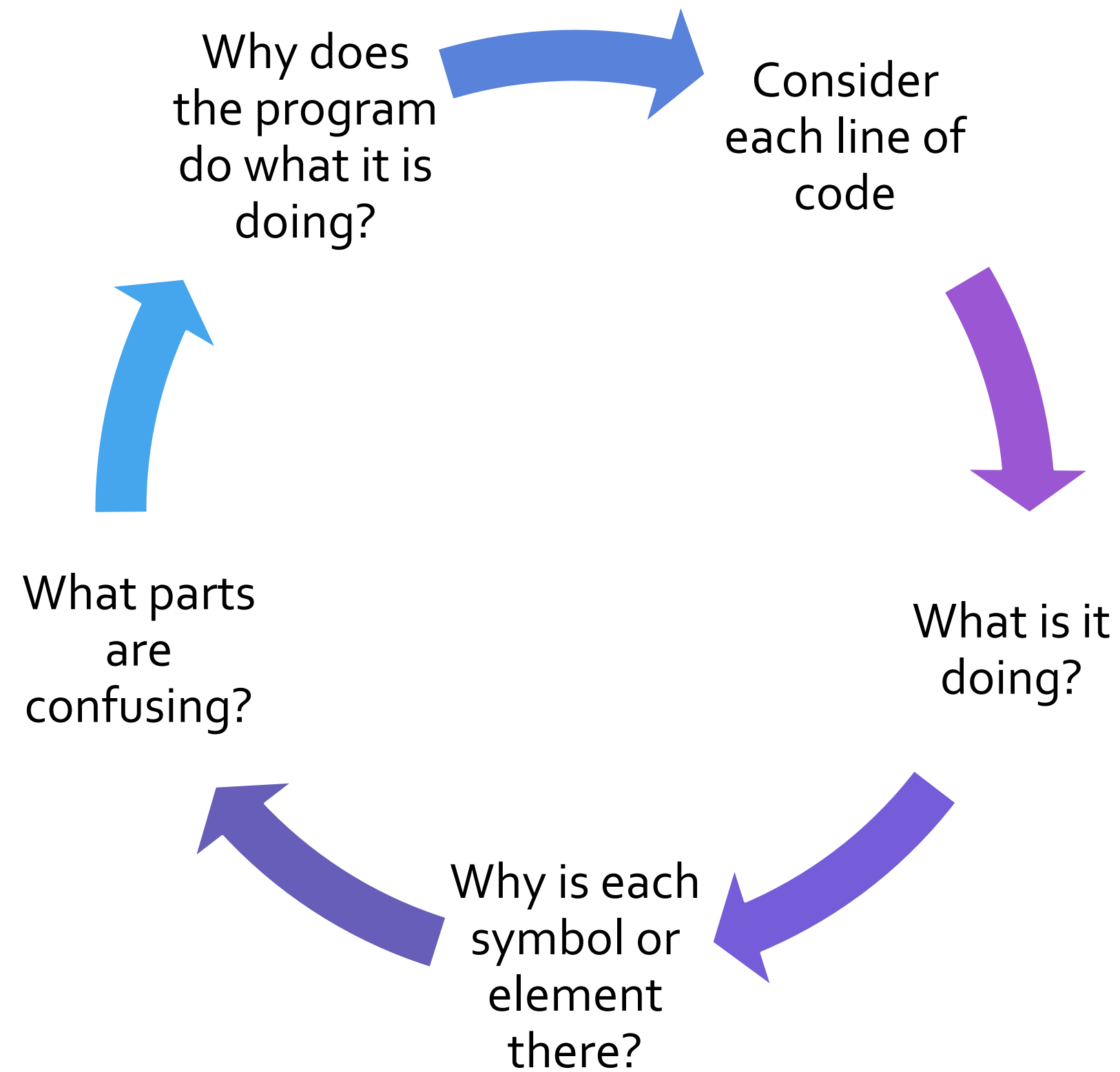
Understand the concepts introduced using real examples



How using Unit tests can help your understanding

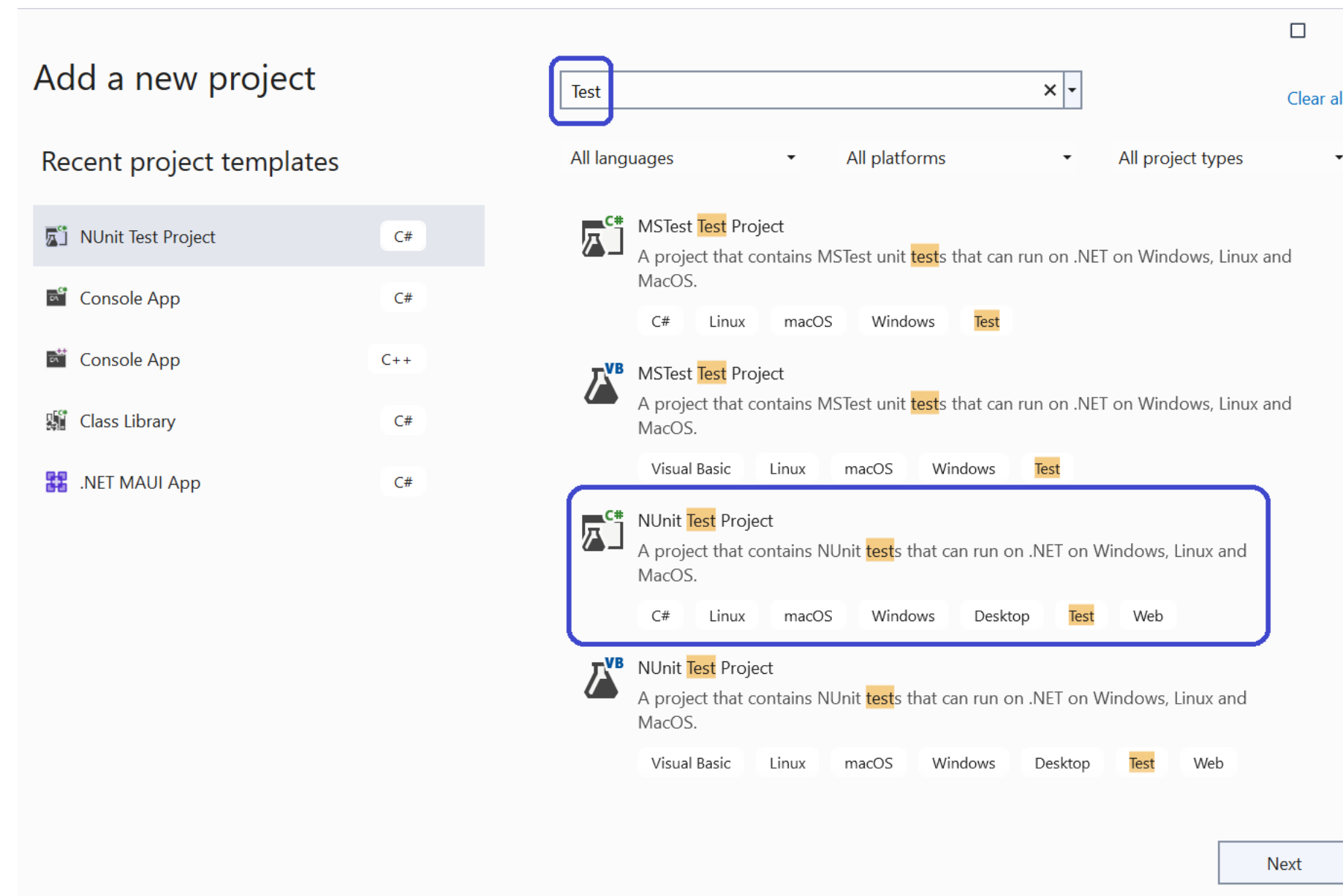


Learn about how text and characters are represented



# Advice

# Test Projects: A special kind of library



# Test Projects



They provide tool support for running tests in the IDE



We will be creating test projects in the lab



Useful for exploratory coding and ... well ... testing

Test Explorer

▶▶▶

🔍

🔧

🧪 6

✅ 1

❌ 0

⚠️ 5

📄

Search (Ctrl+I)

Test run finished: 1 Tests (1 Passed, ⚠️ 1 Warning, ❌ 0 Errors)

Test	Duration	Traits	Err...
StringDemo (6)	2 ms		
StringDemo (6)	2 ms		
StringTests (6)	2 ms		
BasicStringEx...			
CharMath			
Comparablel...	2 ms		
LastTenChars			
Test1			
TestChar			

Test Detail Summary

✅ ComparableInterfaceExample

📄 Source: [StringTests.cs](#) line 21

🕒 Duration: 2 ms

📄 Standard Output:  
Compared b to d and the result was -2

CppTypeSystemDemo.cpp

StringTests.cs

SystemC

StringDemo

StringDemo.StringTests

```
10 string s2 = "Hello";
11 String s3 = "Hello";
12 var s4 = $"{s1}";
13 var s5 = "He" + "llo";
14 CompareEquality(s1, s2);
15 CompareEquality(s1, s3);
16 CompareEquality(s1, s4);
17 CompareEquality(s1, s5);
18 }
19
20 [Test]
21 public static void ComparableInterfaceEx
22 {
23     var c1 = 'b';
24     var c2 = 'd';
25     var iface = (IComparable<char>)c1;
26     var result = iface.CompareTo(c2);
27     Console.WriteLine($"Compared {c1} to {c2} and the result was {result}");
28 }
29
30 [Test]
31 public static void CharMath()
32 {
33     var n = 98;
34     var c = (char)n;
35     Console.WriteLine($"This number {n} converted to char is {c}");
36 }
37
```

78 %

❌ 0 ⚠️ 1

Output

Show output from: Debug

The thread 0x6890 has exited with code 0 (0x0).  
The thread 0x7f18 has exited with code 0 (0x0).  
'testhost.exe' (CoreCLR: clrhost): Loaded 'C:\Program Files\dotnet\dotnet.exe'.  
'testhost.exe' (CoreCLR: clrhost): Loaded 'C:\Program Files\dotnet\dotnet.exe'.  
'testhost.exe' (CoreCLR: clrhost): Loaded 'C:\Program Files\dotnet\dotnet.exe'.  
The thread 0x6b98 has exited with code 0 (0x0).  
The program '[25044] testhost.exe' has exited with code 0 (0x0).

# THE TEST EXPLORER WINDOW

# What is a String?

- A representation of text values as sequences of characters

# Consider the Following

- Are strings value types (“structs”) and allocated on the stack?
- Or are they reference types (“classes”) and allocated on the heap?
- What about chars?
- Why?



# Strings

- <https://learn.microsoft.com/en-us/dotnet/csharp/programming-guide/strings/>
- A string is an object of type [`System.String`](#) whose value represents text.
- Internally, the text is stored as a sequential read-only collection of char objects
- The keyword “string” is an alias for the type “System.String”

# A Helper Function for Today

```
public static void CompareEquality(object a, object b)
{
    var eq = a.Equals(b);
    Console.WriteLine($"a is {a.GetType()} and b is {b.GetType()}");
    Console.WriteLine($"It is {eq} that {a} and {b} are equal");
}
```

```
[Test]
public void BasicStringExample()
{
    var s1 = "Hello";
    string s2 = "Hello";
    String s3 = "Hello";
    var s4 = $"{s1}";
    var s5 = "He" + "llo";
    CompareEquality(s1, s2);
    CompareEquality(s1, s3);
    CompareEquality(s1, s4);
    CompareEquality(s1, s5);
}
```

## STRING VARIABLE DECLARATION

# The Take-away

- No observable difference between “string” and “System.String”
- Local variable declarations can (and should) use var
- Use the “string” alias in your code
- See the [Microsoft coding guidelines](#)

# System.String Documentation

[Learn](#) / [.NET](#) / [.NET API browser](#) / [System](#) /

C#    

## String Class

Reference

 [Feedback](#)


## Definition

Namespace: [System](#)

Assembly: System.Runtime.dll

Represents text as a sequence of UTF-16 code units.

C#

 Copy

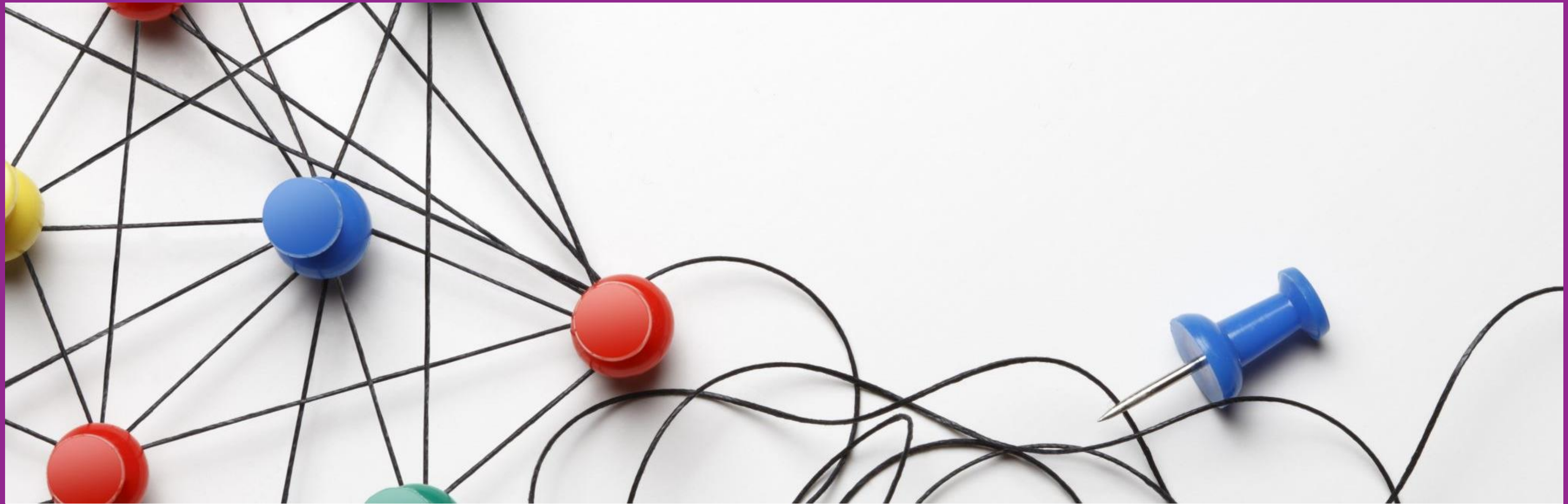
```
public sealed class String : ICloneable, IComparable, IComparable<string>, IConvertible,
IEquatable<string>, System.Collections.Generic.IEnumerable<char>
```

Inheritance [Object](#) → String

Implements [IEnumerable<Char>](#) , [IEnumerable](#) , [IComparable](#) , [IComparable<String>](#) , [IConvertible](#) ,  
[IEquatable<String>](#) , [ICloneable](#)

# Reading the documentation

- It said that it was a “class” (so it is a reference type)
- It said that it derives from “System.Object”
- This means that it shares the methods (functions) of System.Object
- We already know that everything derives from System.Object
- What is a UTF-16 Code Unit?
- What does it mean that it “implements interfaces”?



# WHY IS STRING A CLASS?

---

# So What's a Char

- A Char is primitive value type
- It has two bytes, and represents a Unicode UTF-16 character.
- It is classified as an integral type by the specification:
- The char keywords is an alias for the type System.Char
- See the [language reference](#) and the [type documentation](#)



# Char Struct

Reference

## Definition

Namespace: [System](#)

Assembly: System.Runtime.dll

Represents a character as a UTF-16 code unit.

C#

```
public readonly struct Char : IComparable, IComparable<char>, IConvertible, IEquatable<Char>, ISpanFormattable
```

Inheritance [Object](#) → [ValueType](#) → Char

Implements [IComparable](#) , [IComparable<Char>](#) , [IConvertible](#) , [IEquatable<Char>](#) , [IFormattable](#) ,  
[ISpanFormattable](#)

THE SYSTEM.CHAR  
DOCUMENTATION

# Salient Points from Documentation

- It's a struct
- It derives from ValueType
- All structs derive from ValueType automatically
- It implements a number of interfaces like IComparable

# What “Implementing Interfaces” means

- An interface is a kind of contract
- It states a set of methods and properties that a class or struct may implement
- An interface is also a type

# IComparable<T> Interface

Reference

 [Feedback](#)


## Definition

Namespace: [System](#)

Assembly: System.Runtime.dll

Defines a generalized comparison method that a value type or class implements to create a type-specific comparison method for ordering or sorting its instances.

C#

 Copy

```
public interface IComparable<in T>
```

## Methods

[CompareTo\(T\)](#)

Compares the current instance with another object of the same type and returns an integer that indicates whether the current instance precedes, follows, or occurs in the same position in the sort order as the other object.

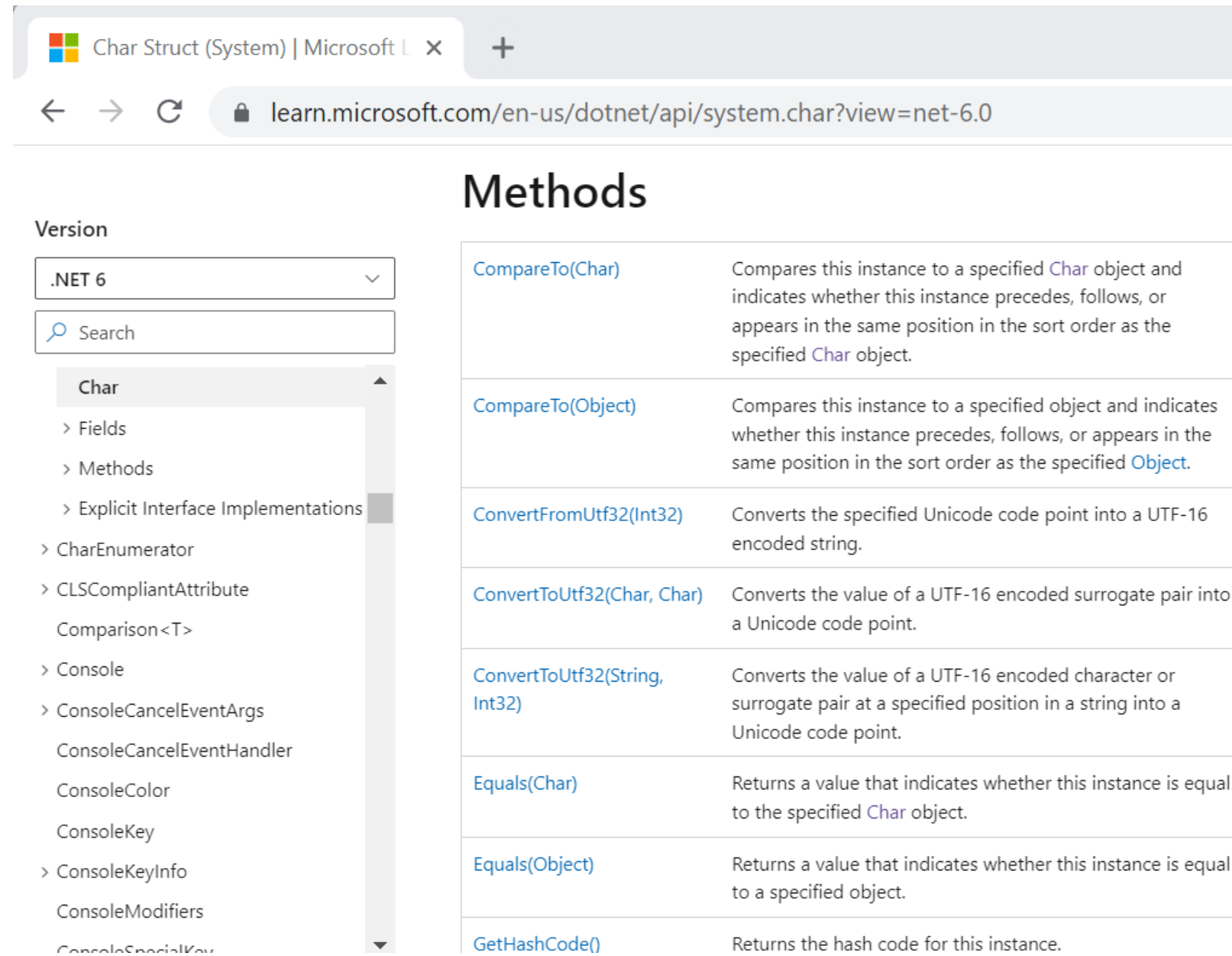
# INTERFACE EXAMPLE

# Code Example of IComparable

[Test]

```
public static void ComparableInterfaceExample()  
{  
    var c1 = 'b';  
    var c2 = 'd';  
    var iface = (IComparable<char>)c1;  
    var result = iface.CompareTo(c2);  
    Console.WriteLine($"Compared {c1} to {c2} and the result was {result}");  
}
```

# The Interface Methods are on the Type



The screenshot shows a web browser window with the address bar displaying `learn.microsoft.com/en-us/dotnet/api/system.char?view=net-6.0`. The page title is "Char Struct (System) | Microsoft Learn". The main content area is titled "Methods" and lists several methods for the `Char` type. On the left side, there is a sidebar with a "Version" dropdown set to ".NET 6", a search bar, and a tree view showing the navigation structure. The tree view includes "Char" (selected), "Fields", "Methods", "Explicit Interface Implementations", "CharEnumerator", "CLSCompliantAttribute", "Comparison<T>", "Console", "ConsoleCancelEventArgs", "ConsoleCancelEventHandler", "ConsoleColor", "ConsoleKey", "ConsoleKeyInfo", "ConsoleModifiers", and "ConsoleSpecialKey".

Methods	
<a href="#">CompareTo(Char)</a>	Compares this instance to a specified <a href="#">Char</a> object and indicates whether this instance precedes, follows, or appears in the same position in the sort order as the specified <a href="#">Char</a> object.
<a href="#">CompareTo(Object)</a>	Compares this instance to a specified object and indicates whether this instance precedes, follows, or appears in the same position in the sort order as the specified <a href="#">Object</a> .
<a href="#">ConvertFromUtf32(Int32)</a>	Converts the specified Unicode code point into a UTF-16 encoded string.
<a href="#">ConvertToUtf32(Char, Char)</a>	Converts the value of a UTF-16 encoded surrogate pair into a Unicode code point.
<a href="#">ConvertToUtf32(String, Int32)</a>	Converts the value of a UTF-16 encoded character or surrogate pair at a specified position in a string into a Unicode code point.
<a href="#">Equals(Char)</a>	Returns a value that indicates whether this instance is equal to the specified <a href="#">Char</a> object.
<a href="#">Equals(Object)</a>	Returns a value that indicates whether this instance is equal to a specified object.
<a href="#">GetHashCode()</a>	Returns the hash code for this instance.

# Glyph


- A graphic symbol that represents a character
- An element of a typeface (aka font fontamily)
- Letters (e.g., 'e') and diacritics (e.g., accents) are separate glyphys


# Char Literals

You can specify a `char` value with:

- a character literal.
- a Unicode escape sequence, which is `\u` followed by the four-symbol hexadecimal representation of a character code.
- a hexadecimal escape sequence, which is `\x` followed by the hexadecimal representation of a character code.

C#

 Copy

 Run

```
var chars = new[]
{
    'j',
    '\u006A',
    '\x006A',
    (char)106,
};
Console.WriteLine(string.Join(" ", chars)); // output: j j j j
```



# Unicode

---

From Wikipedia, the free encyclopedia

**Unicode**, formally **The Unicode Standard**,<sup>[note 1][note 2]</sup> is an information technology standard for the consistent encoding, representation, and handling of text expressed in most of the world's writing systems. The standard, which is maintained by the Unicode Consortium, defines as of the current version (15.0) 149,186 characters<sup>[3][4]</sup> covering 161 modern and historic scripts, as well as symbols, emoji (including in colors), and non-visual control and formatting codes.

# WHAT'S UNICODE?

---

# What's Utf-16

- One of multiple Unicode encodings (e.g., Utf-8, Utf-16, Utf-32, GB18030)
- It is variable length: Characters use either one or two 16-bit code units
- Not as widely used on the web as Utf-8

# What's an Encoding

- An assignment of numbers to characters

# Did you see that!

- Two “Chars” in a row are sometimes needed to properly display characters
- This is why we disambiguate code units and code points
- Code points are made up of code units
- In C# a char instance is a code unit in the Utf-16 encoding

# ASCII

- ASCII is a character encoding first published as a standard in 1963
- Only 128 Characters
- 95 are printable
- 33 are control codes
- One byte per code point

```
public static void CompareEquality(object a, object b)
{
    var eq = a.Equals(b);
    Console.WriteLine($"a is {a.GetType()} and b is {b.GetType()}");
    Console.WriteLine($"It is {eq} that {a} and {b} are equal");
}
```

[Test]

```
public static void TestChar()
{
    var c1 = 'a';
    var c2 = 'b';
    var c3 = (int)c1;
    var c4 = (char)c3;
    var c5 = c2 - 1;
    var c6 = (char)c5;
    CompareEquality(c1, c1);
    CompareEquality(c1, c2);
    CompareEquality(c1, c3);
    CompareEquality(c1, c4);
    CompareEquality(c1, c5);
    CompareEquality(c1, c6);
}
```

# COMPARING CHARACTER EQUALITY

# Let's Convert it to Bytes

- Use "BitConverter.ToBytes"
- <https://learn.microsoft.com/en-us/dotnet/api/system.bitconverter.getbytes?view=net-7.0>

# Escape Sequences

- How do you write the character used to delimit char literals?
- Or the character that represents a newline or tab?
- Or an unprintable control code like the bell
- Answer: use an escape sequence (backslash followed by code).



## Examples

The following code example demonstrates some of the methods in `Char`.

```
C# Copy Run

using System;

public class CharStructureSample
{
    public static void Main()
    {
        char chA = 'A';
        char ch1 = '1';
        string str = "test string";

        Console.WriteLine(chA.CompareTo('B')); //----- Output: "-1" (meaning 'A' is 1
        Console.WriteLine(chA.Equals('A'));    //----- Output: "True"
        Console.WriteLine(Char.GetNumericValue(ch1)); //----- Output: "1"
        Console.WriteLine(Char.IsControl('\t')); //----- Output: "True"
        Console.WriteLine(Char.IsDigit(ch1));    //----- Output: "True"
        Console.WriteLine(Char.IsLetter(','));   //----- Output: "False"
        Console.WriteLine(Char.IsLower('u'));    //----- Output: "True"
        Console.WriteLine(Char.IsNumber(ch1));   //----- Output: "True"
        Console.WriteLine(Char.IsPunctuation('.') ); //----- Output: "True"
        Console.WriteLine(Char.IsSeparator(str, 4)); //----- Output: "True"
        Console.WriteLine(Char.IsSymbol('+'));    //----- Output: "True"
        Console.WriteLine(Char.IsWhiteSpace(str, 4)); //----- Output: "True"
        Console.WriteLine(Char.Parse("S"));      //----- Output: "S"
        Console.WriteLine(Char.ToLower('M'));    //----- Output: "m"
        Console.WriteLine('x'.ToString());       //----- Output: "x"
    }
}
```

# SOME CHAR METHODS

# From String to Char

- A string (s) with one character can be converted to a char in two ways:
- One is to use "[String.Parse](#)"
- The other is to use the indexing property of the string "string[o]"

# Invoking Instance versus Static Methods



Some methods have the form  
"expression.FunctionName(<args>)"



While others have the form  
"typename.FunctionName(<args>)"

# Char is not an expression

- The word “Char” is the name of a type
- More specifically a class within the namespace
- You cannot use it where you would an expression:

```
public static void TestCharWord()  
{  
    var x = Char;  
}
```

# But you can use it to call static methods

- `Char.IsControl`
- `Char.IsLetterOrDigit`
- `Char.IsUpper`
- `Char.IsWhitespace`
- Etc.

# BACK TO STRINGS

---

# String Literals

- Regular string literals
- Verbatim string literals
- Interpolated string expressions
- Raw string literals (C# 11)

# Verbatim String Literals

- A verbatim string literal is preceded by the “@” character
- See: <https://learn.microsoft.com/en-us/dotnet/csharp/language-reference/tokens/verbatim>.



# Nul Characters

- A C# string can contain any number of embedded nul characters (' ').
- This differs from C/C++ which uses null to indicate termination
- Not to be confused with the null keyword, this is a char that encodes zero

# Useful String Functions

01

String.Join

02

String.IndexOf

03

String.LastIndexOf

04

String.Substring

05

String.Split

06

String.Format

# Useful String Operations not on String

- `Int32.ParseInt`
- `Double.TryParse()`
- `Object.ToString();`
- `Convertible?`

# CONVERTING STRINGS TO BYTES AND BACK

---

# Bit Converter does not work on Strings?!

## Overloads

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

# Remember Encodings?

- We have to use one of those:
- `System.Text.Encoding.UTF8.GetBytes(value);`
- `System.Text.Encoding.UTF16.GetBytes(value);`
- `System.Text.Encoding.ASCIIEncoding.GetBytes(value);`

# 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 them?



STRINGBUILDER CLASS



STRING.FORMAT



STRING  
INTERPOLATION  
EXPRESSION



CONCATENATION



FROM AN ARRAY OF  
CHARS



# String Operators

+ String  
concatenation

+= String  
concatenation  
and assignment

== Equality

!= Inequality

# 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

# Wait, what is a property?

---

A property is a type member that resembles a field (data member)

---

It may redirect to a field or to a function under the hood

---

It may be read-only, read-write, or write-only (rare)

---

An example is the “Length” property of arrays and strings, or “Count” property of Lists

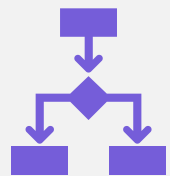
---

Like other members it may be static or instance based

# 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 hexadecimal 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 hexadecimal is 263A

The character is ☺

But I could have also just written ☺

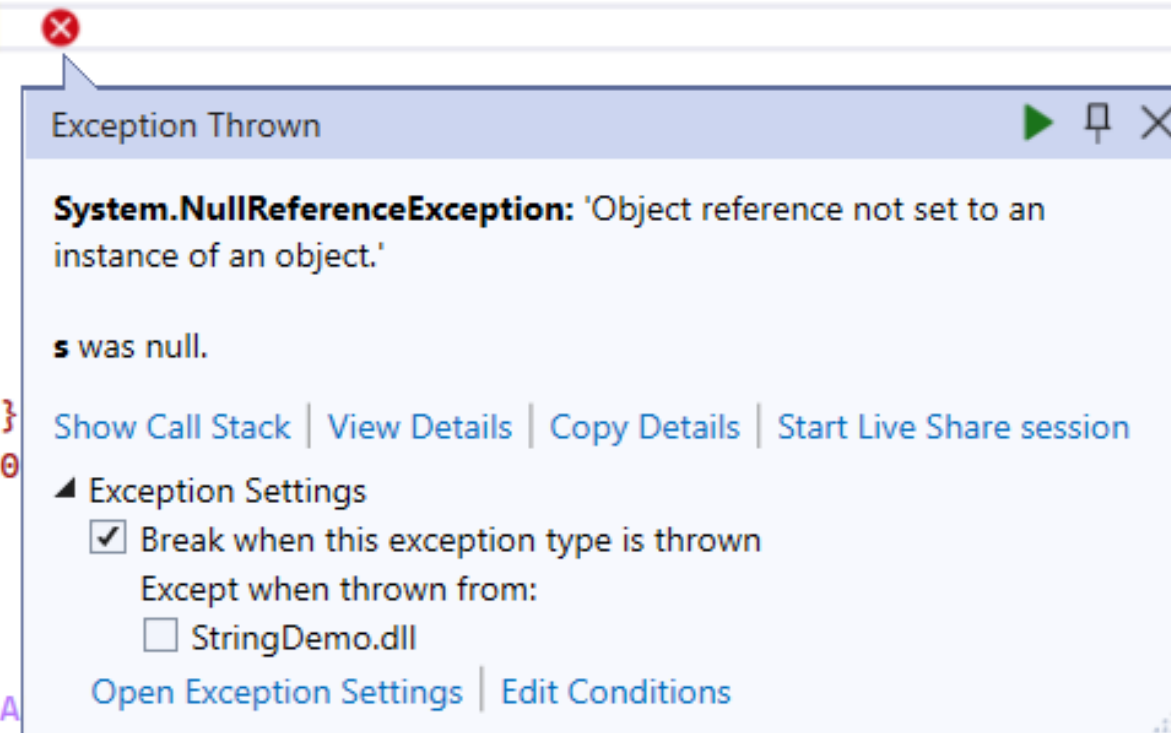
# The Null Literal

- 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”



# 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 hexadecimal is {0,10:X}");  
    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");  
}
```




# 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:


C#

 Copy

```
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:

C#

 Copy

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

```
public static void TestString(string s)
{
    if (string.IsNullOrEmpty(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);
}
```

# STRING QUERIES

# Strings are Enumerable

Strings implement  
"IEnumerable"



This means you  
can loop through  
the characters  
with a foreach

# Next Class

- Collections
- Foreach loops
- Interfaces in a more depth
- Indexers in more depth
- Iterators