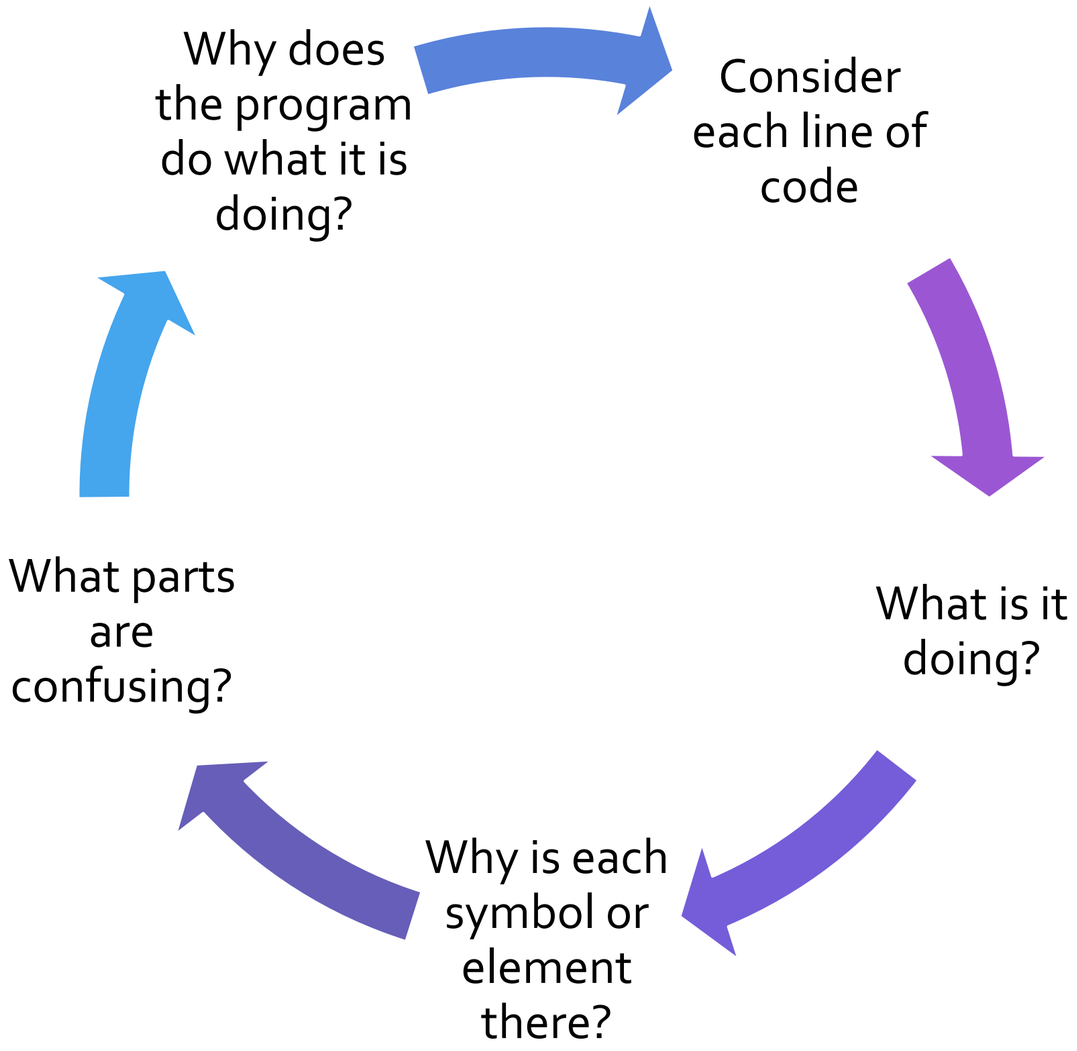
TODAY'S GOAL

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

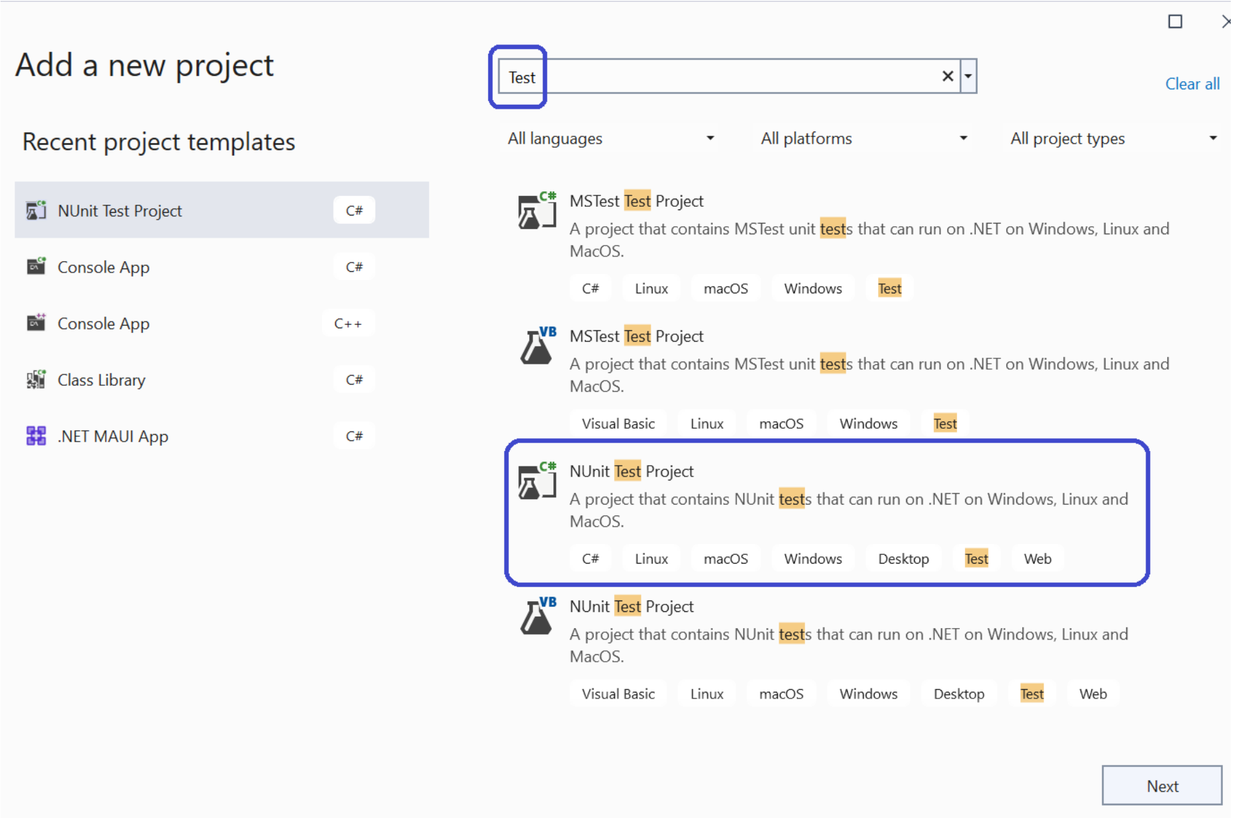
Goal Breakdown

- 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


Why does the program do what it is doing?
Consider each line of code
What is it doing?
Why is each symbol or element there?
What parts are confusing?

Test Projects: A special kind of library

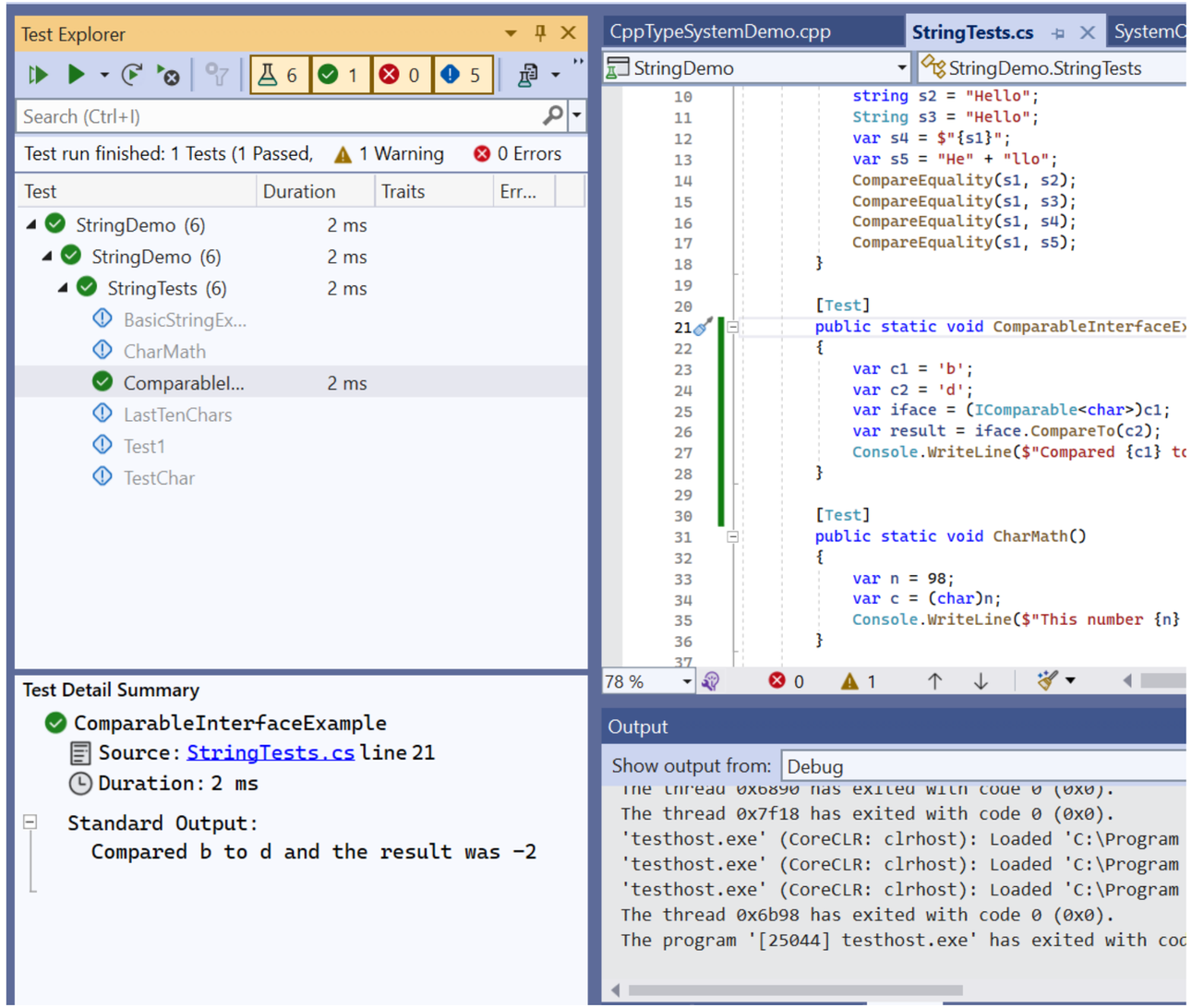


Test Projects

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



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

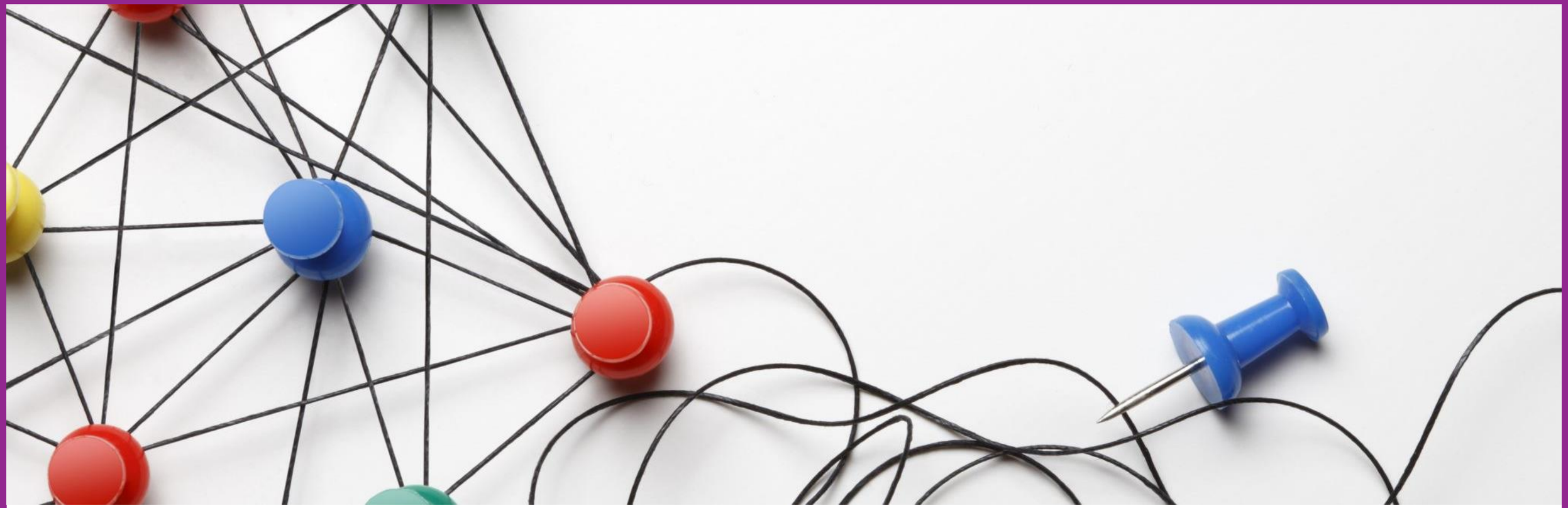
```
public sealed class String : ICloneable, IComparable, IComparable<string>, IConvertible,
IEquatable<string>, System.Collections.Generic.IEnumerable<char>
```

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

Learn / .NET / .NET API browser / System / C#

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<c
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

Char Struct (System) | Microsoft L

learn.microsoft.com/en-us/dotnet/api/system.char?view=net-6.0

Version

.NET 6

Search

- Char
  - Fields
  - Methods
  - Explicit Interface Implementations
- CharEnumerator
- CLSCompliantAttribute
- Comparison<T>
- Console
- ConsoleCancelEventArgs
- ConsoleCancelEventHandler
- ConsoleColor
- ConsoleKey
- ConsoleKeyInfo
- ConsoleModifiers

Methods

CompareTo(Char)	Compares this instance to a specified Char object and indicates whether this instance precedes, follows, or appears in the same position in the sort order as the specified Char object.
CompareTo(Object)	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 Object.
ConvertFromUtf32(Int32)	Converts the specified Unicode code point into a UTF-16 encoded string.
ConvertToUtf32(Char, Char)	Converts the value of a UTF-16 encoded surrogate pair into a Unicode code point.
ConvertToUtf32(String, Int32)	Converts the value of a UTF-16 encoded character or surrogate pair at a specified position in a string into a Unicode code point.
Equals(Char)	Returns a value that indicates whether this instance is equal to the specified Char object.
Equals(Object)	Returns a value that indicates whether this instance is equal to a specified object.
GetHashCode()	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**,[note 1][note 2] 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[3][4] 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 <u>code units</u> and <u>code points</u>
- 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($"a is {a.GetType()} and b is {b.GetTy
    Console.WriteLine($"It is {eq} that {a} and {b} are equa
}
```

```
[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

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[0]"

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.