**CSharp6 Tutorial**

**Lesson01 C# 6.0 Features**

**Notes: -**

**1-This sixth iteration of the C# language is provided by the Roslyn compiler. This compiler came out with version 4.6 of the .NET Framework, however it can generate code in a backward compatible manner to allow targeting earlier framework versions. C# version 6 code can be compiled in a fully backwards compatible manner to .NET 4.0. It can also be used for earlier frameworks, however some features that require additional framework support may not function correctly. (released on VS 2015 on .Net 4.6)**

**2-As well as adding some new language features it includes a complete rewrite of the compiler. Previously csc.exe was a native Win32 application written in C++, with C# 6 it is now a .NET managed application written in C#. This rewrite was known as project "Roslyn" and the code is now open source and available on GitHub.**

**Features: -**

**1-Auto-property initializers**

**2-Await in catch and finally**

**3-Disable Warnings Enhancements**

**4-Exception filters**

**5-Expression-bodied function members**

**6-Improved overload resolution**

**7-Index initializers**

**8-Minor changes and bugfixes**

**9-Null propagation**

**10-Operator nameof**

**11-String interpolation**

**12-Using an extension method for collection initialization**

**13-Using static type**

**Lesson02 Auto Property Initializers**

**Notes: -**

**1-old way on working with property before C# 6.0**

**public class Coordinate{**

**private int \_x = 34;**

**public int X { get { return \_x; } set { \_x = value; } }**

**private readonly int \_y = 89;**

**public int Y { get { return \_y; } }**

**private readonly int \_z;**

**public int Z { get { return \_z; } }**

**public Coordinate(){\_z = 42;}**

**2-we can make property initializer value as list below:-**

**using System;**

**using System.Collections.Generic;**

**namespace AutoPropInitPro{**

**public class Coordinate{**

**// get or set auto-property with initializer**

**public int X { get; set; } = 34;**

**// read-only auto-property with initializer**

**public int Y { get; } = 89;**

**// Read-only auto-property with expression body , instead of using {get;} as read only property**

**public ICollection<string> Users3 => new HashSet<string>() { "OP1", "OP2" };**

**//set auto-property with initializer with detect access modifier, internal , internal protected , private**

**public string Name { get; protected set; } = "Cheeze";**

**//we can set readonly list**

**public List<string> Ingredients { get; } = new List<string> { "dough", "sauce", "cheese" };**

**//this way we using expression body on the get accessor to access to the private field as below**

**private ICollection<string> \_User4 = new HashSet<string>() { "OP1", "OP2" };**

**public ICollection<string> Users4 { get => \_User4;set {\_User4 = value; }}}**

**public class Rectangle{**

**//to initialize auto properties as below**

**public double Length { get; set; } = 1;**

**//can set Width property at different level of layer , so cannot change outside**

**public double Width { get; private set; } = 1;**

**//we can use static method on set properities as below**

**public double Area { get; set; } = CalculateArea(1, 1);**

**public static double CalculateArea(double length, double width){return length \* width;}}**

**class Program{**

**static void Main(string[] args){**

**Coordinate cord = new Coordinate();**

**//it will not add new items but readonly better way that the {get;} , its readonly property which is not added after initialized**

**cord.Users3.Add("asd");**

**//becouse its read only you cannot assign**

**//cord.Y = 11;**

**//becouse its readonly list you cannot add new items**

**//cord.Ingredients.Add("adsad");**

**Rectangle rect = new Rectangle();**

**//cannot set private set field as below**

**//rect.Width = 2223;**

**Console.WriteLine(rect.Area.ToString());**

**Console.ReadLine();}}}**

**Lesson03 Await in catch and finally**

**It is possible to use await expression to apply await operator to Tasks or**

**Task (Of TResult) in the catch and finally blocks in C#6.**

**(in the previous versions it’s not possible to use await in try, catch , finally)**

**static void Main(string[] args){**

**try{**

**//since C#6**

**//await service.InitializeAsync();}**

**catch (Exception e){**

**//since C#6**

**//await logger.LogAsync(e);}**

**finally{**

**//since C#6**

**//await service.CloseAsync();}**

**//in the previous version of C# 6.0 we have to use**

**bool error = false;**

**Exception ex = null;**

**try{**

**// Since C#5**

**//await service.InitializeAsync();}**

**catch (Exception e){**

**// Declare bool or place exception inside variable**

**error = true;**

**ex = e;}**

**// If you don't use the exception**

**if (error){// Handle async task}**

**// If want to use information from the exception**

**if (ex != null){//await logger.LogAsync(e);}**

**// Close the service, since this isn't possible in the finally**

**//await service.CloseAsync();}**

**Lesson04 Disable Warnings Enhancements**

**In C# 5.0 and earlier the developer could only suppress warnings by number. With the introduction of Roslyn Analyzers, C# needs a way to disable warnings issued from specific libraries. With C# 6.0 the pragma directive can suppress warnings by name.**

**Before:**

**#pragma warning disable 0501**

**C# 6.0:**

**#pragma warning disable CS0501**

**Lesson05 Exception Filters**

**Notes: -**

**1-Exception filters give developers the ability to add a condition (in the form of a boolean expression) to a catch block, allowing the catch to execute only if the condition evaluates to true.**

**2-Exception filters allow the propagation of debug information in the original exception, whereas using an if statement inside a catch block and re-throwing the exception stops the propagation of debug information in the original exception. With exception filters, the exception continues to propagate upwards in the call stack unless the condition is met. As a result, exception filters make the debugging experience much easier. Instead of stopping on the throw statement, the debugger will stop on the statement throwing the exception, with the current state and all local variables preserved. Crash dumps are affected in a similar way.**

**3-Exception filters have been supported by the CLR since the beginning and they've been accessible from VB.NET and F# for over a decade by exposing a part of the CLR's exception handling model. Only after the release of C# 6.0 has the functionality also been available for C# developers.**

**using System;**

**using System.Data.SqlClient;**

**namespace ExceptionFilterPro{**

**class Program{**

**static void Main(string[] args){**

**// Check01();**

**// Check03();**

**// Check04();**

**Check05();}**

**static void Check01(){**

**var SqlErrorToIgnore = 123;**

**try{// DoSQLOperations();}**

**//it will make filter to the exception check so not enter until the condition return true**

**catch (SqlException ex) when (ex.Number != SqlErrorToIgnore){**

**throw new Exception("An error occurred accessing the database", ex);}}**

**static void Check02(){**

**//try**

**//{ ... }**

**//catch (Exception ex) when (someCondition) //If someCondition evaluates to true , the rest of the catches are ignored.**

**//{ ... }**

**//catch (NotImplementedException ex) when (someMethod()) //someMethod() will only run if someCondition evaluates to false**

**//{ ... }**

**//catch (Exception ex) // If both when clauses evaluate to false//{ ... }}**

**static void Check03(){**

**int a = 7;**

**int b = 0;**

**try{DoSomethingThatMightFail();}**

**catch (Exception ex) when (a / b == 0){**

**// This block is never reached because a / b throws an ignored**

**// DivideByZeroException which is treated as false.}**

**catch (Exception ex){**

**// This block is reached since the DivideByZeroException in the**

**// previous when clause is ignored.}}**

**static void DoSomethingThatMightFail(){**

**// This will always throw an ArgumentNullException.**

**Type.GetType(null);}**

**static void Check04(){**

**int a = 0, b = 0;**

**//it will make exception on the same line not entered to the catch section**

**try{int c = a / b;}**

**catch (DivideByZeroException) when (a != 0){throw;}}**

**static void Check05(){**

**try{// DoSomethingThatMightFail(s);}**

**catch (Exception ex){**

**Log(ex, "An error occurred");**

**throw;}**

**static void Log(Exception ex, string message, params object[] args){**

**// Debug.Print(message, args);}}**

**static void Check06(){**

**Console.WriteLine("Start");**

**try{SomeOperation();}**

**catch (Exception) when (EvaluatesTo()){Console.WriteLine("Catch");}**

**finally{Console.WriteLine("Outer Finally");}}**

**private static bool EvaluatesTo(){**

**// Console.WriteLine($"EvaluatesTo: {Flag}");**

**return true;}**

**private static void SomeOperation(){**

**try{**

**//Flag = true;**

**throw new Exception("Boom");}**

**finally{**

**//Flag = false;**

**Console.WriteLine("Inner Finally");}}}}**

**Lesson06 Expression bodied function**

**Notes:-**

**using System.Collections.Generic;**

**using System.Linq;**

**//we can use expression boided function over many types such as property , methods , indexers**

**namespace ExpressionBoidedFuncPro{**

**public class Employee{**

**//1-Properities and fields**

**public decimal BasePrice, Taxes;**

**public decimal TotalPrice => BasePrice + Taxes;**

**//its equivalent to**

**//public decimal TotalPrice{get{return BasePrice + Taxes;}}**

**//2-Indexers**

**public Dictionary<string, string> lstKeys = new Dictionary<string, string>();**

**public object this[string key] => lstKeys[key];**

**//its equivalent to**

**//public object this[string key]{get{return dictionary[key];}}**

**//3-Methods**

**static int Multiply(int a, int b) => a \* b;**

**//its eqivalent to**

**//static int Multiply(int a, int b){return a \* b; }**

**//limitations**

**//expression function can't contain block statements and any other statements that contain blocks: if, switch, for, foreach, while, do, try, etc.**

**IEnumerable<string> Digits => Enumerable.Range(0, 10).Select(i => i.ToString());**

**//its eqivalent to**

**//IEnumerable<string> Digits{get{for (int i = 0; i < 10; i++)yield return i.ToString();}}}}**

**Lesson07 Improved overloaded resolutions**

**Notes: -**

**1-Following snippet shows an example of passing a method group**

**(as opposed to a lambda) when a delegate is expected. Overload resolution will now resolve this instead of raising an ambiguous overload error due to the ability of C# 6 to check the return type of the method that was passed.**

**using System;**

**namespace ImpOverResPro{**

**class Program{**

**public static void Main(){**

**Overloaded(DoSomething);**

**Foo(() => () => 7);**

**Foo(() => () => 7l);}**

**static void Overloaded(Action action){Console.WriteLine("overload with action called");}**

**static void Overloaded(Func<int> function){Console.WriteLine("overload with Func<int> called");}**

**static int DoSomething(){**

**Console.WriteLine(0);**

**return 0;}**

**static void Foo(Func<Func<long>> func) { }**

**static void Foo(Func<Func<int>> func) { }}}**

**Lesson08 Indexer Initializer**

**Notes: -**

**1-Index initializers make it possible to create and initialize objects with indexes at the same time.**

**This makes initializing Dictionaries very easy:**

**using System;**

**using System.Collections.Generic;**

**namespace IndexInitPro{**

**public class MyClassWithIndexer{**

**public int this[string index]{**

**set{Console.WriteLine($"Index: {index}, value: {value}");}}**

**public string this[int index]{**

**set{Console.WriteLine($"Index: {index}, value: {value}");}}}**

**class Program{**

**static void Main(string[] args){**

**//indexer initalizer with set key string**

**var dict = new Dictionary<string, int>(){**

**["foo"] = 34,**

**["bar"] = 42};**

**//it will initialize indexer with difffernt key types**

**var x = new MyClassWithIndexer(){**

**["foo"] = 34,**

**["bar"] = 42,**

**[10] = "Ten",**

**[42] = "Meaning of life"};**

**Console.WriteLine("Hello World!");}}}**

**Lesson09 Null Propagation**

**Notes: -**

**1-If the developer instead uses the? (null-conditional) operator, the expression will evaluate to null instead of throwing an exception.**

**2-the null-coalesing operator ?? to return default value if the expression resolves to null**

**Example:-**

**namespace NullPropagationPro{**

**//If the developer instead uses the?. (null-conditional) operator, the expression will evaluate to null instead of throwing an exception.**

**class Program{**

**static void Main(string[] args){**

**//1-teacherName is null if GetTeacher() returns null**

**//var teacherName = classroom.GetTeacher()?.Name;**

**//2-teacherName is null if GetTeacher() returns null OR classroom is null**

**//var teacherName = classroom?.GetTeacher()?.Name;**

**//3-compile time error: implicit conversion from bool? to bool not allowed**

**//bool hasCertification = classroom?.GetTeacher()?.HasCertification;**

**//4-works just fine, hasCertification will be null if any part of the chain is null**

**//bool? hasCertification = classroom?.GetTeacher()?.HasCertification;**

**//5-you can combine the null-conditional operator with the Null-coalescing Operator (??) to return a default value if the expression resolves to null**

**//A-teacherName will be "No Name" when GetTeacher()**

**//B-returns null OR classroom is null OR Name is null**

**//var teacherName = classroom?.GetTeacher()?.Name ?? "No Name";**

**//6-you can use null propagation with indexer**

**//var firstStudentName = classroom?.Students?[0]?.Name;**

**//7-you can use propagation with void functions**

**//List<string> list = null;**

**//list?.Add("hi");**

**//Limitations**

**//1-cannot use null propagation with property assigment , event subscription ,etc....**

**// Error: The left-hand side of an assignment must be a variable, property or indexer**

**//Process.GetProcessById(1337)?.EnableRaisingEvents = true;**

**// Error: The event can only appear on the left hand side of += or -=**

**//Process.GetProcessById(1337)?.Exited += OnProcessExited;**

**//2-you have to make sure that the name is not null**

**//int? nameLength = person?.Name.Length;}}}**

**Lesson10 Operator nameof**

**Notes: -**

**1-** **The nameof operator returns the name of a code element as a string.**

**using System;**

**namespace OperatorNameOfPro{**

**class Program{**

**static void Main(string[] args){**

**Console.WriteLine(SayHello(null));**

**Console.WriteLine(SayHello("Ahmad"));**

**//we can access to the property name as below**

**Console.WriteLine(Strings.Foo);**

**Console.WriteLine(Strings.Bar);**

**Console.ReadLine();}**

**public static string SayHello(string greeted){**

**if (greeted == null)**

**//it will return the name of the property or class ,etc...**

**return nameof(greeted);**

**return $"Hello ${greeted}";}**

**public static class Strings{**

**// Rather than Foo = "Foo"**

**public const string Foo = nameof(Foo);**

**// Rather than Bar = "Bar"**

**public const string Bar = nameof(Bar); }}}**

**Lesson11 String Interpolation**

**1-string interpolation introduced on C# 6.0 which provide more useful string interpolation formatting as below**

**using System;**

**namespace StringInterPro{**

**class Program{**

**static void Main(string[] args){**

**Console.WriteLine(CallFunc1());**

**Console.WriteLine(CallFunc2());**

**Console.WriteLine(CallFunc3());**

**Console.WriteLine(CallFunc4());**

**Console.WriteLine(CallFunc5());**

**Console.WriteLine(CallFunc6());**

**Console.ReadLine();}**

**static string CallFunc1(){**

**int foo = 34;int bar = 42;**

**return $"The foo is {foo}, and the bar is {bar}.";}**

**static string CallFunc2(){**

**var foo = 34;var bar = 42;**

**// String interpolation notation (new style)**

**return $"The foo is {{foo}}, and the bar is {{bar}}.";}**

**//we can use string interpolation among multiple lines**

**static string CallFunc3(){**

**var foo = 34;var bar = 42;**

**return $@"In case it wasn't clear:**

**\u00B9**

**The foo**

**is {foo},and the baris {bar}.";}**

**//you can apply mathmatic operations with the string interpolation as below , and apply formatting to property as below**

**static string CallFunc4(){**

**var foo = 34;**

**var bar = 42;**

**return @$"And the greater one is: { Math.Max(foo, bar) }**

**Price: {foo:c4}**

**Today: {DateTime.Now:dddd,MMMM dd - yyyyy}";}**

**//you can apply conditional statement inside string interpolation**

**static string CallFunc5(){**

**var foo = 34;var bar = 42;**

**return $"{(foo > bar ? "Foo is larger than bar" : "Bar is larger than Foo")}";}**

**//the escape character such as \" working as expected as the normal state**

**static string CallFunc6(){**

**var foo = 34;**

**return $"Foo is: {foo}. In a non-verbatim string, we need to escape \" and \\ with backslashes.";}}}**

**Lesson12 using Static Type**

**Notes: -**

**1-The using static [Namespace.Type] directive allows the importing of static members of types and enumeration values. Extension methods are imported as extension methods (from just one type), not into top-level scope.**

**using static System.Console;**

**using static System.ConsoleColor;**

**using static System.Math;**

**class Program{**

**static void Main(){**

**BackgroundColor = DarkBlue;**

**WriteLine(Sqrt(2));}}**