**C Sharp 7 Tutorial**

**Notes: -**

**1-C Sharp 7 features is list as below**

**Binary literals**

**Digit separators**

**Extended expression bodied members list**

**Language support for Tuples**

**Local functions**

**out var declaration**

**Pattern Matching**

**ref return and ref local**

**throw expressions**

**ValueTask**

**Lesson01 Binary Literals**

**1-The 0b prefix can be used to represent Binary literals.**

**2-Binary literals allow constructing numbers from zeroes and ones, which makes seeing which bits are set in the binary representation of a number much easier. This can be useful for working with binary flags.**

**Example:-**

**using System;**

**namespace BinaryLiteralsPro{**

**//With binary literals it is more obvious which bits are set, and using them does not require understanding hexadecimal numbers and bitwise arithmetic:**

**[Flags]**

**public enum DaysOfWeek{**

**Monday = 0b00000001,**

**Tuesday = 0b00000010,**

**Wednesday = 0b00000100,**

**Thursday = 0b00001000,**

**Friday = 0b00010000,**

**Saturday = 0b00100000,**

**Sunday = 0b01000000,**

**//we can assign multiple item values on the item enum**

**Weekdays = Monday | Tuesday | Wednesday | Thursday | Friday,**

**Weekends = Saturday | Sunday}**

**class Program{**

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

**// Using a binary literal:**

**// bits: 76543210**

**int a1 = 0b00100010; // binary: explicitly specify bits**

**// Existing methods:**

**int a2 = 0x22; // hexadecimal: every digit corresponds to 4 bits**

**int a3 = 34; // decimal: hard to visualise which bits are set**

**int a4 = (1 << 5) | (1 << 1); // bitwise arithmetic: combining non-zero bits**

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

**Lesson02 Digit Separators**

**Notes: -**

**1-The underscore \_ may be used as a digit separator. Being able to group digits in large numeric literals has a significant impact on readability.**

**Example:-**

**using System;**

**namespace DigitSepPro{**

**//Where the \_ digit separator may not be used:**

**//1-at the beginning of the value(\_121)**

**//2-at the end of the value(121\_ or 121.05\_)**

**//3-next to the decimal (10\_.0)**

**//4-next to the exponent character(1.1e\_1)**

**//5-next to the type specifier(10\_f)**

**//6-immediately following the 0x or 0b in binary and hexadecimal literals(might be changed to allow e.g.**

**class Program{**

**//in order to make more readable value we using digit seperator as below**

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

**int bin = 0b1001\_1010\_0001\_0100;**

**int hex = 0x1b\_a0\_44\_fe;**

**int dec = 33\_554\_432;**

**int weird = 1\_2\_\_3\_\_\_4\_\_\_\_5\_\_\_\_\_6\_\_\_\_\_\_7\_\_\_\_\_\_\_8\_\_\_\_\_\_\_\_9;**

**double real = 1\_000.111\_1e-1\_000;**

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

**Lesson03 Extended Expression Bodied members list**

**1-C# 7.0 adds accessors, constructors and finalizers to the list of things that can have expression bodies:**

**using System;**

**using System.Collections.Concurrent;**

**namespace ExtExprBoidPro{**

**//we now can assign extended expression on constructor and destructor and fields as below**

**class Person{**

**private static ConcurrentDictionary<int, string> names = new ConcurrentDictionary<int, string>();**

**private int id = GetId();**

**public Person(string name) => names.TryAdd(id, name); // constructors**

**~Person() => names.TryRemove(id, out \_); // finalizers**

**public string Name{get => names[id]; set => names[id] = value;}**

**public static int GetId(){return 5;}}**

**class Program{**

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

**var emp = new Person("Mohammed Ali");**

**emp = null;**

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

**Lesson04 Language Support for Tuples**

**Notes: -**

**1-A tuple is an ordered, finite list of elements. Tuples are commonly used in programming as a means to work with one single entity collectively instead of individually working with each of the tuple's elements, and to represent individual rows (ie. "records") in a relational database.**

**Example:-**

**using System;**

**using System.Collections.Generic;**

**using System.Linq;**

**namespace AdvTuplesPro{**

**class Program{**

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

**//1-named Tuple properities**

**var result = CallFunc1();**

**Console.WriteLine($"{result.sum} : {result.count}");**

**//2-Tuple Desconstruction**

**var (s, c) = result;**

**Console.WriteLine($"After Destruct : S = {s} , C = {c}");**

**//3-Tuple Swapping**

**(s, c) = (c, s);**

**Console.WriteLine($"After Swapping : S = {s} , C = {c}");**

**//4-Class Deconstruct : we using the destructor**

**var person = new Person { FirstName = "John", LastName = "Smith" };**

**//it will call the destructor as below**

**var (localFirstName, localLastName) = person;**

**Console.WriteLine($"After Deconstruct class : first name : {localFirstName} , last name :{localLastName}");**

**//5-Tuple Initialization**

**var name = (first: "John", middle: "Q", last: "Smith");**

**Console.WriteLine($"Full Name {name.first} {name.middle} {name.last}");**

**//6-Tuple Initialization within method**

**var lst = new List<int>() { 100, 110, 120 };**

**var resMath = Measure(lst);**

**Console.WriteLine($"Sum : {resMath.sum} , Avg : {resMath.average}");**

**//7-working with Collection Tuples**

**var matchedRes = FindMatchingValue("test1","test2");**

**Console.ReadLine();}**

**//we can make function with return tuple with names each property as below**

**public static (int sum, int count) CallFunc1(){return (1, 2);}**

**//Tuple Initialization within method**

**public static (int sum, double average) Measure(List<int> items){**

**var stats = (sum: 0, average: 0d);**

**stats.sum = items.Sum();**

**stats.average = items.Average();**

**return stats;}**

**//Tuple Initialization with async method with using ValueTyple Type**

**//public async Task<(string value, int count)> GetValueAsync()**

**//{**

**// string fooBar = await \_stackoverflow.GetStringAsync();**

**// int num = await \_stackoverflow.GetIntAsync();**

**// return (fooBar, num);**

**//}**

**//we can create Collection of tuple in the new format in C# 7.0**

**private static readonly List<(string firstThingy, string secondThingyLabel, string foundValue)> labels = new List<(string firstThingy, string secondThingyLabel, string foundValue)>(){**

**("test1", "test2", "Value"),**

**("test1", "test1", "Value2"),**

**("test2", "test2", "Value3")};**

**public static string FindMatchingValue(string firstElement, string secondElement){**

**var result = labels?**

**.Where(w => w.firstThingy == firstElement && w.secondThingyLabel == secondElement)**

**.FirstOrDefault();**

**if (result == null)**

**throw new ArgumentException("combo not found");**

**return result.Value.foundValue;}}**

**class Person{**

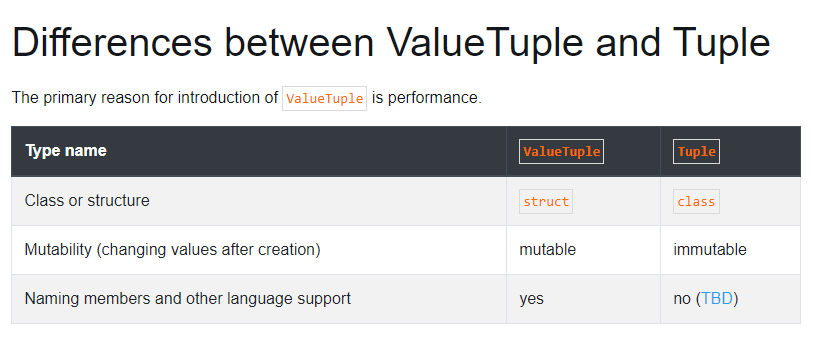
**public string FirstName { get; set; }**

**public string LastName { get; set; }**

**public void Deconstruct(out string firstName, out string lastName){**

**firstName = FirstName;**

**lastName = LastName;}}}**



**Lesson05 Local Functions**

**Notes: -**

**1-Local functions are defined within a method and aren't available outside of it. They have access to all local variables and support iterators, async/await and lambda syntax. This way, repetitions specific to a function can be functionalized without crowding the class. As a side effect, this improves intellisense suggestion performance.**

**using System;**

**using System.Collections.Generic;**

**using System.IO;**

**using System.Linq;**

**using System.Text.RegularExpressions;**

**using System.Threading.Tasks;**

**namespace LocalFuncPro{**

**class Program{**

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

**//1-basic usage of local functions**

**GetCylinderVolume(22, 13);**

**//2-using local function with LINQ operations customize**

**var lst = new List<string>() { "Lst1", "Lst2", "Lst3" };**

**var res = lst.Where(x => x.Equals("Lst1"));**

**var input1 =Path.Combine(Environment.CurrentDirectory,"Input1.txt");**

**var input2 = Path.Combine(Environment.CurrentDirectory, "Input2.txt");**

**var output = Path.Combine(Environment.CurrentDirectory, "Output.txt");**

**WriteEmailsAsync(input1, input2, output).GetAwaiter().GetResult();**

**Console.ReadLine();}**

**//we see that we define multiple inner local functions as below**

**static double GetCylinderVolume(double radius, double height){**

**return getVolume();**

**double getVolume(){**

**// You can declare inner-local functions in a local function**

**double GetCircleArea(double r) => Math.PI \* r \* r;**

**// ALL parents' variables are accessible even though parent doesn't have any input.**

**return GetCircleArea(radius) \* height;}}**

**async static Task WriteEmailsAsync(string file1,string file2,string outputFile){**

**var emailRegex = new Regex(@"(?i)[a-z0-9\_.+-]+@[a-z0-9-]+\.[a-z0-9-.]+");**

**IEnumerable<string> emails1 = await getEmailsFromFileAsync(file1);**

**IEnumerable<string> emails2 = await getEmailsFromFileAsync(file2);**

**await writeLinesToFileAsync(emails1.Concat(emails2), outputFile);**

**//local function to read from file**

**async Task<IEnumerable<string>> getEmailsFromFileAsync(string fileName){**

**string text;**

**using (StreamReader reader = File.OpenText(fileName)){**

**text = await reader.ReadToEndAsync();}**

**return from Match emailMatch in emailRegex.Matches(text) select emailMatch.Value;}**

**//local function to write to target file**

**async Task writeLinesToFileAsync(IEnumerable<string> lines, string fileName){**

**using (StreamWriter writer = File.CreateText(fileName)){**

**foreach (string line in lines){**

**await writer.WriteLineAsync(line);}}}}}}**

**using System;**

**using System.Collections.Generic;**

**using System.Text;**

**namespace LocalFuncPro{**

**public static class IEnumerableExt{**

**//Local functions considerably simplify code for LINQ operators, where you usually have to separate argument checks from**

**//actual logic to make argument checks instant, not delayed until after iteration started.**

**public static IEnumerable<TSource> Where<TSource>(this IEnumerable<TSource> source, Func<TSource, bool> predicate){**

**if (source == null)**

**throw new ArgumentNullException(nameof(source));**

**if (predicate == null)**

**throw new ArgumentNullException(nameof(predicate));**

**return iterator();**

**IEnumerable<TSource> iterator(){**

**foreach (TSource element in source)**

**if (predicate(element))**

**yield return element;}}}}**

**Lesson06 Out Var Declaration**

**Notes:-**

**1-A common pattern in C# is using bool TryParse(object input, out object value) to safely parse objects.**

**2-The out var declaration is a simple feature to improve readability. It allows a variable to be declared at the same time that is it passed as an out parameter.**

**using System;**

**using System.Linq;**

**namespace OutVarPro{**

**class Program{**

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

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

**CallFunc1("1");**

**// I only care about x**

**//p.GetCoordinates(out var x, out \_);**

**CallFunc2();**

**//Note that out var declarations are of limited use in LINQ queries as expressions are interpreted as expression lambda bodies,**

**//so the scope of the introduced variables is limited to these lambdas. For example, the following code will not work:**

**//var nums =**

**// from item in seq**

**// let success = int.TryParse(item, out var tmp)**

**// select success ? tmp : 0; // Error: The name 'tmp' does not exist in the current context**

**Console.ReadLine();}**

**static void CallFunc1(string no){**

**//internally it will created at the top level of the function so it will accessiable at all rest code on the level of method**

**if (int.TryParse(no, out int val)){Foo(ref val); // ok}**

**else{Foo(ref val); // value is zero}**

**//we can call it outside the scope and inside the scope**

**Foo(ref val);**

**static void Foo(ref int val){val = val + 1;}}**

**//out params can be used with anonmous types**

**static void CallFunc2(){**

**var a = new[] { 1, 2, 3, 4, 5, 6, 7, 8, 9 };**

**var groupedByMod2 = a.Select(x => new{**

**Source = x,Mod2 = x % 2}).GroupBy(x => x.Mod2).ToDictionary(g => g.Key, g => g.ToArray());**

**if (groupedByMod2.TryGetValue(1, out var oddElements)){**

**Console.WriteLine(oddElements.Length);}}}}**

**Lesson07 Pattern Matching**

**Notes: -**

**1-Pattern matching extensions for C# enable many of the benefits of pattern matching from functional languages, but in a way that smoothly integrates with the feel of the underlying language**

**using System;**

**namespace PatternMatchingPro{**

**class Geometry{}**

**class Rectangle : Geometry{**

**public int Width { get; set; }**

**public int Height { get; set; }}**

**class Square : Geometry{**

**public int Width { get; set; }}**

**class Triangle : Geometry{**

**public int Width { get; set; }**

**public int Height { get; set; }**

**public int Base { get; set; }}**

**class Program{**

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

**PatternMatching();**

**Console.ReadLine();}**

**public static void PatternMatching(){**

**Geometry g = new Square { Width = 5 };**

**switch (g){**

**case Triangle t:**

**Console.WriteLine($"{t.Width} {t.Height} {t.Base}");**

**break;**

**case Rectangle sq when sq.Width == sq.Height:**

**Console.WriteLine($"Square rectangle: {sq.Width} {sq.Height}");**

**break;**

**case Rectangle r:**

**Console.WriteLine($"{r.Width} {r.Height}");**

**break;**

**case Square s:**

**Console.WriteLine($"{s.Width}");**

**break;**

**default:**

**Console.WriteLine("<other>");**

**break;}}}}**

**Lesson08 Pattern Matching**

**1-Pattern matching extensions for C# enable many of the benefits of pattern matching from functional languages, but in a way that smoothly integrates with the feel of the underlying language**

**using System;**

**namespace PatternMatchingPro{**

**class Geometry{}**

**class Rectangle : Geometry{**

**public int Width { get; set; }**

**public int Height { get; set; }}**

**class Square : Geometry{**

**public int Width { get; set; }}**

**class Triangle : Geometry{**

**public int Width { get; set; }**

**public int Height { get; set; }**

**public int Base { get; set; }}**

**class Program{**

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

**PatternMatching();**

**var o = "hello";**

**string s = o as string;**

**if (s != null){// do something with s}**

**//it will check for variable o is null or not and create new instance at the same time**

**if (o is string s1){//Do something with s};**

**Console.ReadLine();}**

**public static void PatternMatching(){**

**Geometry g = new Square { Width = 5 };**

**switch (g){**

**case Triangle t:**

**Console.WriteLine($"{t.Width} {t.Height} {t.Base}");**

**break;**

**case Rectangle sq when sq.Width == sq.Height:**

**Console.WriteLine($"Square rectangle: {sq.Width} {sq.Height}");**

**break;**

**case Rectangle r:**

**Console.WriteLine($"{r.Width} {r.Height}");**

**break;**

**case Square s:**

**Console.WriteLine($"{s.Width}");**

**break;**

**default:**

**Console.WriteLine("<other>");**

**break;}}}}**

**Lesson09 ref return and ref local**

**Notes: -**

**1-C# 7.0 allows throwing as an expression in certain places:**

**using System;**

**namespace ThrowExprPro{**

**//we can apply throw expression in easy way throw ? or ??**

**class Person{**

**public string Name { get; }**

**public Person(string name) => Name = name ?? throw new ArgumentNullException(nameof(name));**

**public string GetFirstName(){**

**var parts = Name.Split(' ');**

**return (parts.Length > 0) ? parts[0] : throw new InvalidOperationException("No name!");}**

**public string GetLastName() => throw new NotImplementedException();}**

**class Program{**

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

**//before C# 7.0 if you want ot check if null or not**

**var spoons = "dinner,desert,soup".Split(',');**

**var spoonsArray = spoons.Length > 0 ? spoons : null;**

**if (spoonsArray == null){**

**throw new Exception("There are no spoons");}**

**//with C# 7.0 the check makes easier as below**

**var spoonsArray1 = spoons.Length > 0 ? spoons : throw new Exception("There are no spoons");**

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

**Lesson10 Value Task**

**1-Task<T> is a class and causes the unnecessary overhead of its allocation when the result is immediately available.**

**2-ValueTask<T> is a structure and has been introduced to prevent the allocation of a Task object in case the result of the async operation is already available at the time of awaiting.**

**3-The above code snippet does not create the entire async state machine magic but it allocates a Task object in the managed heap. To avoid this allocation, you might want to take advantage of a ValueTask instead as shown in the code snippet given below.**

**4-ValueTask is a value type with two fields, whereas Task is a reference type with a single field**

**5-Use Task when you have a piece of code that will always be asynchronous, i.e., when the operation will not immediately complete. Take advantage of ValueTask when the result of an asynchronous operation is already available or when you already have a cached result.**

**Example:-**

**using System;**

**using System.Diagnostics;**

**using System.Threading.Tasks;**

**namespace ValueTaskPro{**

**class Program{**

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

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

**Stopwatch sc1 = new Stopwatch();**

**sc1.Start();**

**IFoo<int> thing = new SynchronousFoo<int>();**

**var x = thing.BarAsync();**

**sc1.Stop();**

**Console.WriteLine("Time For sync operation is " + sc1.ElapsedMilliseconds.ToString());**

**sc1.Reset();**

**sc1.Start();**

**IFoo<int> thing2 = new AsynchronousFoo<int>();**

**var x2 = thing.BarAsync();**

**sc1.Stop();**

**Console.WriteLine("Time For async operation is " + sc1.ElapsedMilliseconds.ToString());**

**sc1.Reset();**

**sc1.Start();**

**TestTask(3).GetAwaiter().GetResult();**

**sc1.Stop();**

**Console.WriteLine("Time For TestTask operation is " + sc1.ElapsedMilliseconds.ToString());**

**sc1.Reset();**

**sc1.Start();**

**TestValueTask(3).GetAwaiter().GetResult();**

**sc1.Stop();**

**Console.WriteLine("Time For TestValueTask operation is " + sc1.ElapsedMilliseconds.ToString());**

**Console.ReadLine();}**

**//used for async operation like read from database , etc.. , 77 ns**

**static async Task<int> TestTask(int d){**

**await Task.Delay(d);**

**return 10;}**

**//in case that the operation in sync it will make result 14 ns**

**static async ValueTask<int> TestValueTask(int d){**

**await Task.Delay(d);**

**return 10;}}}**