# C# 9.0:

1. **Init property:**

public class Member {

public int Id {

get;

init;

} // set is replaced with init

public string Name {

get;

set;

}

public string Address {

get;

set;

}

}

1. **Record Type:**

* Immutable means it cannot change. By default, Record types are immutable.
* Record Type - it is a compact and easy way to write reference types (immutable) that automatically behave like value type.
* The properties of an instance of a reference type cannot change after its initialisation and hence make them great fit in situations when you're going to need to evaluate objects to see if they are equal to each other. Records utilize value-based equality

public record PersonRecord(string FirstName, string LastName);

public record Member

{

public int ID { get; init; }

public string FirstName { get; init; }

public string LastName { get; init; }

public string Address { get; init; }

}

var member = new Member

{

Id=1,

FirstName="Moshe",

LastName="Cohen",

Address = "Haifa"

};

var newMember = member with { Address = "Tel Aviv" };

Records can also be defined :

public record Member (string FirstName, int ID);

and Initialized:

var m = new Member ("Joe", 12455);

תרגיל

* צור\י record:

public record Person(string Name, DateOnly BirthDate)

* צור\י מערך של אנשים כאשר חלקם בעלי אותם תכונות (שם ותאריך לידה)
* הדפס\י לconsole כמה אנשים יש, וכמה אנשים שונים יש. (ניתן להשתמש בפקודת הlinq .Distinct()
* השתמש\י בgroupBy כדי להדפיס כמה פעמים חוזר כל אדם במערך.
* חזור\י על אותם שלבים כאשר Person הוא class

פתרון:

public record Person(string Name, DateOnly BirthDate);

IEnumerable<Person> people = new Person[]

{

new("Joe", new(1997, 4, 28)),

new("Jane", new(1982, 5, 14)),

new("Joe", new(1986, 7, 19)),

new("Joe", new(1997, 4, 28)),

new("Jane", new(1982, 5, 14)),

};

Console.WriteLine($" Total {people.Count()}");

Console.WriteLine($"Distinct {people.Distinct().Count()}");

// Total 5

// Distinct 3

var repeats = people

.GroupBy(x => x)

.Select(g => (g.Key, g.Count()));

foreach ((Person person, int count) in repeats)

{

Console.WriteLine($"{count} x {person}");

}

// 2 x Person { Name = Joe, BirthDate = 4/28/1997 }

// 2 x Person { Name = Jane, BirthDate = 5/14/1982 }

// 1 x Person { Name = Joe, BirthDate = 7/19/1986 }

1. **Pattern matching enhancements:**

Use of is to:

* Check type:

Person p = new Person();

If(p is Person){

…

}

* match an expression against a pattern:

static bool IsFirstFridayOfOctober(DateTime date) =>

date is { Month: 10, Day: <=7, DayOfWeek: DayOfWeek.Friday };

* Negation pattern

if (result is not null)

{

…

}

switch (shape)

{

case Rectangle r when r.Length == r.Height:

Console.WriteLine("Area of Sqaure is : " + r.Length \* r.Height);

break;

case Rectangle r:

Console.WriteLine("Area of Rectangle is : " + r.Length \* r.Height);

break;

case Circle c:

Console.WriteLine("Area of Circle is : " + c.Radius \* c.Radius \* Shape.PI);

break;

case Triangle t:

Console.WriteLine("Area of Triangle is : " + 0.5 \* t.Base \* t.Height);

break;

default:

throw new ArgumentException(message: "Invalid Shape",paramName: nameof(shape));

int? maybe = 12;

if (maybe is int number)

{

Console.WriteLine($"The nullable int 'maybe' has the value {number}");

}

else

{

Console.WriteLine("The nullable int 'maybe' doesn't hold a value");

}

int number = maybe ?? default(int);

int number = maybe==null? default(int): maybe.Value;

int number = maybe.GetValueOrDefault();

1. **Top-level statements**

Top-level statements remove unnecessary ceremony from many applications.

1. **Fit and finish features:**

* private List<WeatherObservation> \_observations = new();
* WeatherStation station = new() { Location = "Seattle, WA" };

# C# 10:

1. **Global using directives**

You can add the global modifier to any [using directive](https://learn.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/using-directive) to instruct the compiler that the directive applies to all source files in the compilation. This is typically all source files in a project.

Create a file called GlobalUsing.cs

Add all global usings there:

global using System.Runtime.Serialization.Formatters.Binary;

1. **Using File-Scoped Namespaces:**

Old syntax:

Namespace shapes

{

Public class Circle

{

}

}

New Syntax:

Namespace shapes;

Public class Circle

{

}

The main limitation we have when using file-scoped namespace is that a file can only contain a single namespace declaration.