# About Tuples

In C#, a tuple is a data structure which organizes data, holding two or more fields of any type.

A tuple is typically created by placing 2 or more expressions separated by commas, within a set of parentheses.

string boast = "All you need to know";

bool success = !string.IsNullOrWhiteSpace(boast);

(bool, int, string) tripple = (success, 42, boast);

As an expression like any other, a tuple can be used in a range of ways: in **[assignments](https://docs.microsoft.com/en-us/dotnet/csharp/tuples" \l "assignment-and-tuples" \t "https://exercism.org/tracks/csharp/concepts/_blank)**, to initialize a field or variable, **[return value](https://docs.microsoft.com/en-us/dotnet/csharp/tuples" \l "tuples-as-method-return-values" \t "https://exercism.org/tracks/csharp/concepts/_blank)** from a method or passed as a parameter. They can be tested for **[equality](https://docs.microsoft.com/en-us/dotnet/csharp/tuples" \l "equality-and-tuples" \t "https://exercism.org/tracks/csharp/concepts/_blank)**. Equality of tuples is illustrated fully in the pattern-matching-tuples exercise but an example is provided in the code below for the sake of completeness.

Fields are extracted using dot syntax. By default, the first field is Item1, the second Item2, etc. Non-default names are discussed below under *Naming*.

A tuple can be used as a generic parameter, e.g. IList<(bool, string)>

In addition, tuples support "deconstruction", discussed below , and can be used in pattern matching which is covered in a later exercise.

// initialization

(int, int, int) vertices = (90, 45, 45);

// assignment

vertices = (60, 60, 60);

// return value

(bool, int) GetSameOrBigger(int num1, int num2)

{

return (num1 == num2, num1 > num2 ? num1 : num2);

}

// method argumentint Add((int, int) operands)

{

return operands.Item1 + operands.Item2;

}

// equality testingvar estimateA = (42, 1729);var estimateB = (2\*3\*7, 7\*13\*19);bool result = estimateA == estimateB;// => result == true

This **[introduction](https://docs.microsoft.com/en-us/dotnet/csharp/tuples" \t "https://exercism.org/tracks/csharp/concepts/_blank)** shows how to define and use tuples.

# Naming

Field names Item1 etc. do not make for readable code. There are 3 ways to provide names to the fields a) in the type declaration or b) in the expression that creates it, c) by means of **[tuple projection initializers](https://docs.microsoft.com/en-us/dotnet/csharp/tuples" \l "tuple-projection-initializers" \t "https://exercism.org/tracks/csharp/concepts/_blank)** (not illustrated here).

// name items in declaration

(bool success, string message) results = (true, "well done!");

bool mySuccess = results.success;

string myMessaage = results.message;

// name items in creating expression

var results2 = (success: true, message: "well done!");

bool mySuccess2 = results2.success;

string myMessaage2 = results2.message;

Don't try to be too clever with the naming mechanism. It is really just syntactic sugar. For example, you cannot change the names of a tuple when you assign to it from another tuple.

# Deconstruction

Sometimes it is convenient to take a tuple and assign the fields to multiple variables and initialize them if appropriate. This mechanism is called **[deconstruction](https://docs.microsoft.com/en-us/dotnet/csharp/tuples" \l "deconstruction" \t "https://exercism.org/tracks/csharp/concepts/_blank)**. (It can also be applied to your own objects. See **[here](https://docs.microsoft.com/en-us/dotnet/csharp/tuples" \l "deconstructing-user-defined-types" \t "https://exercism.org/tracks/csharp/concepts/_blank)**).

var goodNumbers = (42, 3.142, 1729);

(int ultimateQuestion, var π, var ramanujan) = goodNumbers;

return ultimateQuestion;// => 42

# Field Assignment

The fields of tuples can be individually assigned to. However, given the trend in C# towards immutability this sort of usage is not widespread. Tuples are often used as if they were immutable which of course they are not.

(int, string) tpl;

tpl.Item1 = 1;

tpl.Item2 = "bad";

tpl.Item2 = "even worse";

// => (1, "even worse";

### **Background**

The tuples we are discussing should not be confused with **[System.Tuple](https://docs.microsoft.com/en-us/dotnet/api/system.tuple" \t "https://exercism.org/tracks/csharp/concepts/_blank)** which will probably be found only in legacy code bases.

The position is slightly more confusing when it comes to System.ValueTuple. The **[docs](https://docs.microsoft.com/en-us/dotnet/api/system.valuetuple-8" \t "https://exercism.org/tracks/csharp/concepts/_blank)** for the final overload of **[System.ValueTuple](https://docs.microsoft.com/en-us/dotnet/api/system.valuetuple-8" \t "https://exercism.org/tracks/csharp/concepts/_blank)**state that "The value tuple types provide the runtime implementation that supports tuples in C# ". By "value tuple types" here they are referring to the generic overloads of System.ValueTuple. By "tuples" they mean the things that have concerned us in this exercise. Effectively they are saying that System.ValueTuple is an implementation detail of tuples. This is unlikely to be of much interest, most of the time, to most people. Unfortunately much of the documentation on "tuples" including Microsoft's own is liberally sprinkled with references to System.ValueTuple. It is probably safe to skate over such references.

### **Other Uses and Limitations**

Pattern matching is a particularly productive use for tuples. This is covered in a later exercise.

Tuples allow for some other minor stylistic flourishes:

* Multiple assignment (particularly in constructors) .e.g (this.field1, this.field2) = (arg1, arg2);
* Swapping or recycling values e.g. (a, b) = (b, a);
* Use instead of a struct in a list
* Use as a dictionary key or the contents of a set.
* Use in LINQ. (LINQ is covered by later exercises).

The documentation describes tuples as a lightweight mechanism and that is the key to its successful use. Using a tuple instead of a class or struct can lead to tedious repetition of its field types and possibly names. This applies particularly to its use as a method argument or as the generic argument in collections.

It is best to make use of tricks like multiple assignment and swapping judiciously.

Relational operations other than equality and inequality are not supported.

Note that tuples were introduced into the language relatively recently (C# 7) so if you want to use them you should make sure your code base is using **[version](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/configure-language-version" \t "https://exercism.org/tracks/csharp/concepts/_blank)** 7 or later of the language.