Combining multiple returned values using tuples

Each method can only return a single value that has a single type. That type could be a simple type, such as `string` in the previous example, a complex type, such as `Person`, or a collection type, such as `List<Person>`.

Imagine that we want to define a method named `GetTheData` that returns both a `string` value and an `int` value. We could define a new class named `TextAndNumber` with a `string` field and an `int` field, and return an instance of that complex type, as shown in the following code:

```
public class TextAndNumber
{
  public string Text;
  public int Number;
}

public class Processor
{
  public TextAndNumber GetTheData()
  {
    return new TextAndNumber
    {
      Text = "What's the meaning of life?",
      Number = 42
    };
  }
}
```

But defining a class just to combine two values together is unnecessary, because in modern versions of C# we can use tuples. I pronounce them as tuh-ples but I have heard other developers pronounce them as too-ples. To-may-toe, to-mah-toe, po-tay-toe, po-tah-toe, I guess.

Tuples have been a part of some languages such as F# since their first version, but .NET only added support for them in .NET 4.0 with the `System.Tuple` type.

It was only in C# 7.0 that C# added language syntax support for tuples and at the same time, .NET added a new `System.ValueTuple` type that is more efficient in some common scenarios than the old .NET 4.0 `System.Tuple` type, and the C# tuple uses the more efficient one.

`System.ValueTuple` is not part of .NET Standard 1.6, and therefore not available by default in .NET Core 1.0 or 1.1 projects. `System.ValueTuple` is built in with .NET Standard 2.0, and therefore, .NET Core 2.0 and later.

Let's explore tuples.

1. In the `Person` class, add statements to define a method that returns a `string` and `int` tuple, as shown in the following code:

```
public (string, int) GetFruit()
{
  return ("Apples", 5);
}
```

2. In the `Main` method, add statements to call the `GetFruit` method and then output the tuple's fields, as shown in the following code:

```
(string, int) fruit = bob.GetFruit();
WriteLine($"{fruit.Item1}, {fruit.Item2} there are.");
```

3. Run the application and view the result, as shown in the following output:

```
Apples, 5 there are.
```

Naming the fields of a tuple

To access the fields of a tuple, the default names are `Item1`, `Item2`, and so on.

You can explicitly specify the field names.

1. In the `Person` class, add statements to define a method that returns a tuple with named fields, as shown in the following code:

```
public (string Name, int Number) GetNamedFruit()
{
  return (Name: "Apples", Number: 5);
}
```

2. In the `Main` method, add statements to call the method and output the tuple's named fields, as shown in the following code:

```
var fruitNamed = bob.GetNamedFruit();
WriteLine($"There are {fruitNamed.Number} {fruitNamed.Name}.");
```

3. Run the application and view the result, as shown in the following output:

```
There are 5 Apples.
```

Inferring tuple names

If you are constructing a tuple from another object, you can use a feature introduced in C# 7.1 called **tuple name inference**.

1. In the `Main` method, create two tuples, made of a `string` and `int` value each, as shown in the following code:

```
var thing1 = ("Neville", 4);
WriteLine($"{thing1.Item1} has {thing1.Item2} children.");

var thing2 = (bob.Name, bob.Children.Count);
WriteLine($"{thing2.Name} has {thing2.Count}
children.");
```

In C# 7.0, both things would use the `Item1` and `Item2` naming schemes. In C# 7.1 and later, the second thing can infer the names `Name` and `Count`.

Deconstructing tuples

You can also deconstruct tuples into separate variables. The deconstructing declaration has the same syntax as named field tuples, but without a variable name for the tuple, as shown in the following code:

```
// store return value in a tuple variable with two fields
(string name, int age) tupleWithNamedFields = GetPerson();
// tupleWithNamedFields.name
// tupleWithNamedFields.age

// deconstruct return value into two separate variables
(string name, int age) = GetPerson();
// name
// age
```

This has the effect of splitting the tuple into its parts and assigning those parts to new variables.

1 In the `Main` method, add the following code:

```
(string fruitName, int fruitNumber) = bob.GetFruit();
WriteLine($"Deconstructed: {fruitName}, {fruitNumber}");
```

2 Run the application and view the result, as shown in the following output:

```
Deconstructed: Apples, 5
```