How can I return multiple values from a function in C#?

Ask Question

I read the C++ version of this question but didn't really understand it.

Can someone please explain clearly if it can be done and how?

341

c# function return

92

edited May 23 '17 at 10:31

Community ♦

1 1

asked Apr 14 '09 at 15:11

Ash

2,927 8 34 50

3 NOTE: Use of Tuple<T> is valid from Framework 4.0 and above – SHEKHAR SHETE Sep 20 '13 at 6:26

12 In C# 7: (int, int) Method() { return (1, 2); } – Spook Sep 18 '17 at 8:50

26 Answers

Home

PUBLIC

Stack Overflow

Tags

Users

Jobs

Use .NET 4.0+'s Tuple:

535

For Example:

```
public Tuple<int, int> GetMultipleValue()
{
```

```
    return Tuple.Create(1,2);
}
```

Tuples with two values have `Item1` and `Item2` as properties.

edited Dec 16 '17 at 19:17
Felix D.
128 3 10

answered Apr 23 '12 at 10:24
Hadas
7,357 1 16 30

39 nice trick but I still see One value. – BlaShadow Jul 17 '13 at 5:14

Really helped me out. Thanks. – SearchForKnowledge Mar 30 '15 at 14:55

3 @BlaShadow: I see two values: 1 and 2 – Bigjim Jun 10 '15 at 8:44

11 I would like to note, that a tuples with two or more or parameters is always hard to read. Especially the calling method has to deal with names Item1 to ItemX, that could lead to bad usage. I would always suggest to create a custom class and give those properties proper fit names. – Jannik Sep 1 '16 at 5:44

4 It would be very very nice if instead of Item1, Item2 and so on one could use named output values. C# 7 possibly is going to provide that. – Снађошfах Sep 23 '16 at 8:53

-6

Today, programmers need time & unforgotable methods. A simple, working and quick solution:

```
private int[] SumAndSub(int A, int B)
{
    return new[] { A + B , A - B };
}
```

Using it in somewhere;

```
var results = SumAndSub(20, 5);
int sum = results[0];
int sub = results[0];
```

edited Feb 26 at 8:17

Mahdi Tahsildari
5,631 13 45 75

answered Sep 25 '18 at 19:47

aliesTech
1 2

What do you mean by "programmers need time & unforgotable methods"? – Thomas Flinkow Jan 18 at 6:49

2

```
<--Return more statements like this you can -->

public (int,string,etc) Sample( int a, int b)
{
    //your code;
    return (a,b);
}
```

You can receive code like

```
(c,d,etc) = Sample( 1,2);
```

I hope it works.

edited Jan 18 at 7:04

Prakash Pazhanisamy

876 1 9 23

answered Jan 18 at 4:30

cruzier

31 4

264

Now that C# 7 has been released, you can use the new included Tuples syntax

```
(string, string, string) LookupName(long id) // tuple return type
{
    ... // retrieve first, middle and last from data storage
    return (first, middle, last); // tuple literal
}
```

which could then be used like this:

```
var names = LookupName(id);
WriteLine($"found {names.Item1} {names.Item3}.");
```

You can also provide names to your elements (so they are not "Item1", "Item2" etc). You can do it by adding a name to the signature or the return methods:

```
(string first, string middle, string last) LookupName(long id) // tu
names
```

or

```
return (first: first, middle: middle, last: last); // named tuple el
```

They can also be deconstructed, which is a pretty nice new feature:

```
(string first, string middle, string last) = LookupName(id1); // dec
declaration
```

Check out this link to see more examples on what can be done :)

edited Aug 3 '18 at 21:36

answered Apr 5 '16 at 20:19

Francisco Noriega
6,531 9 41 67

2 Cool! Can't wait to write code this way!! – Snađoшfax Sep 23 '16 at 8:49

can we use this feature now ??? it gets error when i write with this way. – Parsa Dec 22 '16 at 15:45

4 @Parsa you can now, IF you use C# 7 – Francisco Noriega Mar 14 '17 at 0:10

5 If you're targeting anything earlier than .NET Framework 4.7 or .NET Core 2.0, you'll need to install a NuGet package. – Phil May 3 '18 at 15:19

2 You can also deconstruct like this `var (first, middle, last) = LookupName(id1)` . – IronSean Jan 15 at 20:47

19

You either return a *class instance* or use *out* parameters. Here's an example of out parameters:

```
void mymethod(out int param1, out int param2)
{
    param1 = 10;
    param2 = 20;
}
```

Call it like this:

```
int i, j;
mymethod(out i, out j);
// i will be 20 and j will be 10
```

edited Jun 20 '18 at 13:42

Bugs
4,159 9 26 37

answered Apr 14 '09 at 15:16

Keltex
22.4k 10 69 106

Ohh so dont use a function? – Ash Apr 14 '09 at 15:19

3 Remember, though that just because you can, doesn't mean you should do this. This is widely accepted as a bad practice in .Net in most cases. – Michael Meadows Apr 14 '09 at 15:21

3 Can you elaborate why is this a bad practise? – Zo Has Jan 28 '13 at 7:13

It's a bad practice in C/C++. The problem is "programming by side effect": int GetLength(char *s) { int n = 0; while (s[n] != '\0') n++; s[1] = 'X'; return (n); } int main() { char greeting[5] = { 'H', 'e', 'l', 'p', '\0' }; int len = GetLength(greeting); cout << len << ": " << greeting; // Output: 5: HXlp } In C# you would have to write: int len = GetLength(ref greeting) Which would signal a big warning flag of "Hey, greeting is not going to be the same after you call this" and greatly reduce bugs. – Dustin_00 Oct 11 '15 at 4:54

149

You can use three different ways

1. ref / out parameters

using ref:

```
static void Main(string[] args)
{
    int a = 10;
    int b = 20;
    int add = 0;
    int multiply = 0;
    Add_Multiply(a, b, ref add, ref multiply);
    Console.WriteLine(add);
    Console.WriteLine(multiply);
}

private static void Add_Multiply(int a, int b, ref int add, ref int
{
    add = a + b;
    multiply = a * b;
}
```

using out:

```
static void Main(string[] args)
{
    int a = 10;
    int b = 20;
    int add;
    int multiply;
    Add_Multiply(a, b, out add, out multiply);
    Console.WriteLine(add);
    Console.WriteLine(multiply);
}

private static void Add_Multiply(int a, int b, out int add, out int
{
    add = a + b;
    multiply = a * b;
}
```

2. struct / class

using struct:

```
struct Result
{
    public int add;
```

```
    public int multiply;
}
static void Main(string[] args)
{
    int a = 10;
    int b = 20;
    var result = Add_Multiply(a, b);
    Console.WriteLine(result.add);
    Console.WriteLine(result.multiply);
}

private static Result Add_Multiply(int a, int b)
{
    var result = new Result
    {
        add = a * b,
        multiply = a + b
    };
    return result;
}
```

using class:

```
class Result
{
    public int add;
    public int multiply;
}
static void Main(string[] args)
{
    int a = 10;
    int b = 20;
    var result = Add_Multiply(a, b);
    Console.WriteLine(result.add);
    Console.WriteLine(result.multiply);
}

private static Result Add_Multiply(int a, int b)
{
    var result = new Result
    {
        add = a * b,
        multiply = a + b
    };
    return result;
}
```

3. Tuple

Tuple class

```
static void Main(string[] args)
{
    int a = 10;
    int b = 20;
    var result = Add_Multiply(a, b);
    Console.WriteLine(result.Item1);
    Console.WriteLine(result.Item2);
}

private static Tuple<int, int> Add_Multiply(int a, int b)
{
    var tuple = new Tuple<int, int>(a + b, a * b);
    return tuple;
}
```

C# 7 Tuples

```
static void Main(string[] args)
{
    int a = 10;
    int b = 20;
    (int a_plus_b, int a_mult_b) = Add_Multiply(a, b);
    Console.WriteLine(a_plus_b);
    Console.WriteLine(a_mult_b);
}

private static (int a_plus_b, int a_mult_b) Add_Multiply(int a, int b
{
    return(a + b, a * b);
}
```

edited Dec 29 '17 at 18:57

John Cummings
900 2 8 22

answered Jun 3 '15 at 23:04

Devrim Altınkurt
1,491 1 5 4

3 Just for my own couriousity, which would you say is fastest and 'best practice'? – Netferret Jan 8 '17 at 10:12

best example for 'using struct' :) – SHEKHAR SHETE Jul 11 '17 at 6:46

1 suggesting to add the c# 7 syntax (and more people vote this up :)) – twomm Nov 9 '17 at 9:59

For your info, a small (irrelevant) typo: In the struct/class solutions you mixed up adding/multiplying. – Szak1 May 4 '18 at 15:30

I didn't know C# Tuple syntax was a thing! Learn something new after years even! – jhaagsma Oct 24 '18 at 19:52

10

No, you can't return multiple values from a function in C# (for versions lower than C# 7), at least not in the way you can do it in Python.

However, there are a couple alternatives:

You can return an array of type object with the multiple values you want in it.

```
private object[] DoSomething()
{
    return new [] { 'value1', 'value2', 3 };
}
```

You can use `out` parameters.

```
private string DoSomething(out string outparam1, out int outparam2)
{
    outparam1 = 'value2';
    outparam2 = 3;
```

```
    return 'value1';
}
```

edited Mar 14 '17 at 0:11

Francisco Noriega
6,531 9 41 67

answered Apr 14 '09 at 15:18

dustyburwell
5,230 1 22 32

8

In C#7 There is a new `Tuple` syntax:

```
static (string foo, int bar) GetTuple()
{
    return ("hello", 5);
}
```

You can return this as a record:

```
var result = GetTuple();
var foo = result.foo
// foo == "hello"
```

You can also use the new deconstructor syntax:

```
(string foo) = GetTuple();
// foo == "hello"
```

Be careful with serialisation however, all this is syntactic sugar - in the actual compiled code this will be a `Tupel<string, int>` (as per the accepted answer) with `Item1` and `Item2` instead of `foo` and `bar`. That means that serialisation (or deserialisation) will use those property names instead.

So, for serialisation declare a record class and return that instead.

Also new in C#7 is an improved syntax for `out` parameters. You can now declare the `out` inline, which is better suited in some contexts:

```
if(int.TryParse("123", out int result)) {
    // Do something with result
}
```

However, mostly you'll use this in .NET's own libraries, rather than in you own functions.

edited May 23 '17 at 12:10

Community ♦

1 1

answered Mar 9 '17 at 11:43

Keith

95k 59 236 354

Please note that, depending on what .Net version you're targeting, you may need to install Nuget package System.ValueTuple. – Licht Mar 29 '17 at 12:49

i was about to answer as above ;-) – Jeyara Apr 28 '17 at 0:57

2

You could use a dynamic object. I think it has better readability than Tuple.

```
static void Main(string[] args){
    var obj = GetMultipleValues();
    Console.WriteLine(obj.Id);
    Console.WriteLine(obj.Name);
}
```

```
private static dynamic GetMultipleValues() {
    dynamic temp = new System.Dynamic.ExpandoObject();
    temp.Id = 123;
    temp.Name = "Lorem Ipsum";
    return temp;
}
```

edited Sep 23 '16 at 9:11

Snađoшfaχ
7,475 10 45 67

answered May 27 '16 at 14:24

Ogge
552 4 15

3 You lose compile time type checking. – Micha Wiedenmann Dec 5 '16 at 13:11

0

You can also use an OperationResult

```
public OperationResult DoesSomething(int number1, int number2)
{
// Your Code
var returnValue1 = "return Value 1";
var returnValue2 = "return Value 2";

var operationResult = new OperationResult(returnValue1, returnValue2
return operationResult;
}
```

answered Sep 13 '16 at 12:21

Ruan
1,210 3 31 57

7

> Some of these answers stated that use **out parameter** but I recommend not using this due to **they don't work with async methods**. See this for more information.

Other answers stated using Tuple, which I would recommend too but using the new feature introduced in C# 7.0.

```
(string, string, string) LookupName(long id) // tuple return type
{
    ... // retrieve first, middle and last from data storage
    return (first, middle, last); // tuple literal
}

var names = LookupName(id);
WriteLine($"found {names.Item1} {names.Item3}.");
```

Further information can be found here.

edited May 23 '17 at 12:26

Community ♦

1 1

answered Aug 27 '16 at 16:24

Luis Teijon

2,395 3 20 41

1

Future version of C# is going to include named tuples. Have a look at this channel9 session for the demo https://channel9.msdn.com/Events/Build/2016/B889

Skip to 13:00 for the tuple stuff. This will allow stuff like:

```
(int sum, int count) Tally(IEnumerable<int> list)
{
// calculate stuff here
return (0,0)
}

int resultsum = Tally(numbers).sum
```

(incomplete example from video)

answered Jun 28 '16 at 8:23

Niels

690 3 9

2

From this article, you can use three options as posts above said.

KeyValuePair is quickest way.

out is at the second.

Tuple is the slowest.

Anyway, this is depend on what is the best for your scenario.

answered Nov 23 '15 at 20:15

maoyang

736 9 10

3

A method taking a delegate can provide multiple values to the caller. This borrows from my answer here and uses a little bit from Hadas's accepted answer.

```
delegate void ValuesDelegate(int upVotes, int comments);
void GetMultipleValues(ValuesDelegate callback)
```

```
{
    callback(1, 2);
}
```

Callers provide a lambda (or a named function) and intellisense helps by copying the variable names from the delegate.

```
GetMultipleValues((upVotes, comments) =>
{
     Console.WriteLine($"This post has {upVotes} Up Votes and {commer
});
```

edited May 23 '17 at 10:31

Community ♦
1 1

answered Sep 11 '15 at 5:56

Scott Turner
147 1 5

Ways to do it:

1) KeyValuePair (Best Performance - 0.32 ns):

```
KeyValuePair<int, int> Location(int p_1, int p_2, int p_3, int p_
{
     return new KeyValuePair<int,int>(p_2 - p_1, p_4-p_3);
}
```

2) Tuple - 5.40 ns:

```
Tuple<int, int> Location(int p_1, int p_2, int p_3, int p_4)
{
      return new Tuple<int, int>(p_2 - p_1, p_4-p_3);
}
```

3) out (1.64 ns) or ref 4) Create your own custom class/struct

ns -> nanoseconds

Reference: multiple-return-values.

answered Aug 27 '15 at 21:46

Adham Sabry
107 7

2

Just use in OOP manner a class like this:

```
class div
{
    public int remainder;

    public int quotient(int dividend, int divisor)
    {
        remainder = ...;
        return ...;
    }
}
```

The function member returns the quotient which most callers are primarily interested in. Additionally it stores the remainder as a data member, which is easily accessible by the caller afterwards.

This way you can have many additional "return values", very useful if you implement database or networking calls, where lots of error messages may be needed but only in case an error occurs.

I entered this solution also in the C++ question that OP is referring to.

answered Nov 21 '14 at 10:32

Roland

1,820 2 22 44

4

Here are basic `Two` methods:

1) Use of ' `out` ' as parameter You can use 'out' for both 4.0 and minor versions too.

Example of 'out':

```
using System;

namespace out_parameter
{
  class Program
   {
     //Accept two input parameter and returns two out value
     public static void rect(int len, int width, out int area, out i
      {
        area = len * width;
        perimeter = 2 * (len + width);
      }
     static void Main(string[] args)
      {
        int area, perimeter;
        // passing two parameter and getting two returning value
        Program.rect(5, 4, out area, out perimeter);
        Console.WriteLine("Area of Rectangle is {0}\t",area);
        Console.WriteLine("Perimeter of Rectangle is {0}\t", perimet
        Console.ReadLine();
      }
   }
}
```

Output:

Area of Rectangle is 20

Perimeter of Rectangle is 18

*Note:*The `out` -keyword describes parameters whose actual variable locations are copied onto the stack of the called method, where those same locations can be rewritten. This means that the calling method will access the changed parameter.

2) `Tuple<T>`

Example of Tuple:

Returning Multiple DataType values using `Tuple<T>`

```
using System;

class Program
{
    static void Main()
    {
    // Create four-item tuple; use var implicit type.
    var tuple = new Tuple<string, string[], int, int[]>("perl",
        new string[] { "java", "c#" },
        1,
        new int[] { 2, 3 });
    // Pass tuple as argument.
    M(tuple);
    }

    static void M(Tuple<string, string[], int, int[]> tuple)
    {
    // Evaluate the tuple's items.
    Console.WriteLine(tuple.Item1);
    foreach (string value in tuple.Item2)
    {
        Console.WriteLine(value);
    }
    Console.WriteLine(tuple.Item3);
    foreach (int value in tuple.Item4)
    {
        Console.WriteLine(value);
    }
    }
}
```

Output

```
perl
java
c#
1
2
3
```

NOTE: **Use of Tuple is valid from Framework 4.0 and above**. `Tuple` type is a `class` . It will be allocated in a separate location on the managed heap in memory. Once you create the `Tuple` , you cannot change the values of its `fields` . This makes the `Tuple` more like a `struct` .

edited Dec 26 '13 at 6:23

answered Sep 20 '13 at 6:41

SHEKHAR SHETE

3,798 12 63 131

0

you can try this

```
public IEnumerable<string> Get()
 {
     return new string[] { "value1", "value2" };
 }
```

answered Aug 16 '13 at 9:38

IMMORTAL

1,142 2 13 29

1 This doesn't really return *multiple values*. It returns a single, collection value. – Matthew Haugen Sep 5 '14 at 4:16

Also, why not use `yield return "value1"; yield return "value2";` as to not have to explicitly create a new `string[]` ? – Thomas Flinkow Jan 18 at 6:49

6

you can try this "KeyValuePair"

```
private KeyValuePair<int, int> GetNumbers()
{
  return new KeyValuePair<int, int>(1, 2);
}


var numbers = GetNumbers();

Console.WriteLine("Output : {0}, {1}",numbers.Key, numbers.Value);
```

Output :

Output : 1, 2

answered Mar 6 '12 at 10:53

Rikin Patel
5,884 5 54 69

4

Mainly two methods are there. 1. Use out/ref parameters 2. Return an Array of objects

answered Apr 14 '09 at 15:51

blitzkriegz
4,270 13 48 71

There's also tuples, and multiple return values as a syntactic sugar for

tuples. – ANeves Jan 4 '18 at 17:19

74

You cannot do this in C#. What you can do is have a `out` parameter or return your own class (or struct if you want it to be immutable).

Using out parameter

```
public int GetDay(DateTime date, out string name)
{
  // ...
}
```

Using custom class (or struct)

```
public DayOfWeek GetDay(DateTime date)
{
  // ...
}

public class DayOfWeek
{
  public int Day { get; set; }
  public string Name { get; set; }
}
```

edited Apr 14 '09 at 15:46

answered Apr 14 '09 at 15:16

Samuel

32.7k 8 77 83

23 An alternative in this case is to use a struct instead of a class for the return type. If the return value is stateless and transient, struct is a better choice. – Michael Meadows Apr 14 '09 at 15:44

1 This is not possible with `async` methods. `Tuple` is the way to go. (I use `out` parameters in synchronous operations though; they are indeed useful in those cases.) – Codefun64 Nov 1 '15 at 21:02

4 This is now possible in C# 7: (int, int) Method() { return (1, 2); } – Spook Sep 18 '17 at 8:50

3 Answer needs to be updated, it has become flat out wrong with recent versions of c#. will change downvote to upvote if updated. – whitneyland Dec 15 '17 at 18:58

11

There are several ways to do this. You can use `ref` parameters:

```
int Foo(ref Bar bar) { }
```

This passes a reference to the function thereby allowing the function to modify the object in the calling code's stack. While this is not technically a "returned" value it is a way to have a function do something similar. In the code above the function would return an `int` and (potentially) modify `bar`.

Another similar approach is to use an `out` parameter. An `out` parameter is identical to a `ref` parameter with an additional, compiler enforced rule. This rule is that if you pass an `out` parameter into a function, that function is required to set its value prior to returning. Besides that rule, an `out` parameter works just like a `ref` parameter.

The final approach (and the best in most cases) is to create a type that encapsulates both values and allow the function to return that:

```
class FooBar
{
    public int i { get; set; }
    public Bar b { get; set; }
}

FooBar Foo(Bar bar) { }
```

This final approach is simpler and easier to read and understand.

answered Apr 14 '09 at 15:19

Andrew Hare
281k 54 584 604

33

Previous poster is right. You cannot return multiple values from a C# method. However, you do have a couple of options:

- Return a structure that contains multiple members
- Return an instance of a class
- Use output parameters (using the **out** or **ref** keywords)
- Use a dictionary or key-value pair as output

The pros and cons here are often hard to figure out. If you return a structure, make sure it's small because structs are value type and passed on the stack. If you return an instance of a class, there are some design patterns here that you might want to use to avoid causing problems - members of classes can be modified because C# passes objects by reference (you don't have ByVal like you did in VB).

Finally you can use output parameters but I would limit the use of this to scenarios when you only have a couple (like 3 or less) of parameters - otherwise things get ugly and hard to maintain. Also, the use of output parameters can be an inhibitor to agility because your method signature will have to change every time you need to add something to the return value whereas returning a struct or class instance you can add members without modifying the method signature.

From an architectural standpoint I would recommend against using key-value pairs or dictionaries. I find this style of coding requires "secret knowledge" in code that consumes the method. It must know

ahead of time what the keys are going to be and what the values mean and if the developer working on the internal implementation changes the way the dictionary or KVP is created, it could easily create a failure cascade throughout the entire application.

answered Apr 14 '09 at 15:18

Kevin Hoffman
3,779 3 26 33

> And you can also throw an `Exception` if the second value you want to return is disjunctive from the first one: like when you want to return either a kind of successful value, or a kind of unsuccessful value. – Cœur Jun 16 '14 at 8:40

10

In C# 4, you will be able to use built-in support for tuples to handle this easily.

In the meantime, there are two options.

First, you can use ref or out parameters to assign values to your parameters, which get passed back to the calling routine.

This looks like:

```
void myFunction(ref int setMe, out int youMustSetMe);
```

Second, you can wrap up your return values into a structure or class, and pass them back as members of that structure. KeyValuePair works well for 2 - for more than 2 you would need a custom class or struct.

answered Apr 14 '09 at 15:17

Reed Copsey
472k 59 988 1282

5

Classes, Structures, Collections and Arrays can contain multiple values. Output and reference parameters can also be set in a function. Return multiple values is possible in dynamic and functional languages by means of tuples, but not in C#.

answered Apr 14 '09 at 15:16

Jose Basilio
43.9k 9 108 116

38

If you mean returning multiple values, you can either return a class/struct containing the values you want to return, or use the "out" keyword on your parameters, like so:

```
public void Foo(int input, out int output1, out string output2, out
    // set out parameters inside function
}
```

answered Apr 14 '09 at 15:15

Chris Doggett
14.8k 4 50 82

2 I don't think it good to use "out" or "ref"——because it can be totally substituted by a returned-value of your own class type. you see, if using "ref", how to assign to such parameters? (It just depends on how to code inside). If in the body of the function, the author has "newed" an instance to the parameter with "ref", this means you can just pass a "nullable" value there. Otherwises not. So that's a little ambigent. And we have better ways (1. Returning your owned class, 2. Turple). – user3230210 Apr 23 '14 at 6:54

protected by Community ♦ Feb 6 at 20:45

Thank you for your interest in this question. Because it has attracted low-quality or spam answers that had to be removed, posting an answer now requires 10 reputation on this site (the association bonus does not count).

Would you like to answer one of these unanswered questions instead?