

# C# - Ultimate Guide - Beginner to Advanced | Master class

## Section 24 – Tuples

---

### Tuples

The System.Tuple class represents a set of values of any data type.

Introduced in C# 4.0.

Useful to return multiple values from a method (or) to pass multiple values to a method.

Represents a set of values quickly without creating a separate class.

Alternative to anonymous objects (to be used as parameter types / return types).

### Step 1: Object of Tuple class

```
var referenceVariable = new Tuple<type1, type2, ...>( ) { value1, value2, ... };
```

### Step 2: Accessing Elements

```
referenceVariable.Item1 //returns value1
```

```
referenceVariable.Item2 //returns value2
```

```
...
```

### Tuple

```
Item1 = value1
```

```
Item2 = value2
```

Tuple stores only a set of values (of any data type); but doesn't store property names. So you should access them as Item1, Item2 etc.; which doesn't make sense some times.

Tuple supports up to 8 elements only by default. You can store more than 8 values by using nested tuples (tuple inside tuple).

Tuples are mainly used to pass multiple values to a method as parameter; and also return multiple values from a method.

## Value Tuples

'Value Tuples' are advancement to 'Tuple' class with simplified syntax.

Introduced in C# 7.1.

Supports unlimited values.

You will access elements with real field names; instead of Item1, Item2 etc.

Can be used as method parameters / return value; much like Tuple class.

### Step 1: Creating Value Tuple

(type fieldName1, type fieldName2, ...) referenceVariable = (value1, value2, ... );

### Step 2: Accessing Elements

referenceVariable.fieldName1 //returns value1

referenceVariable.fieldName2 //returns value2

...

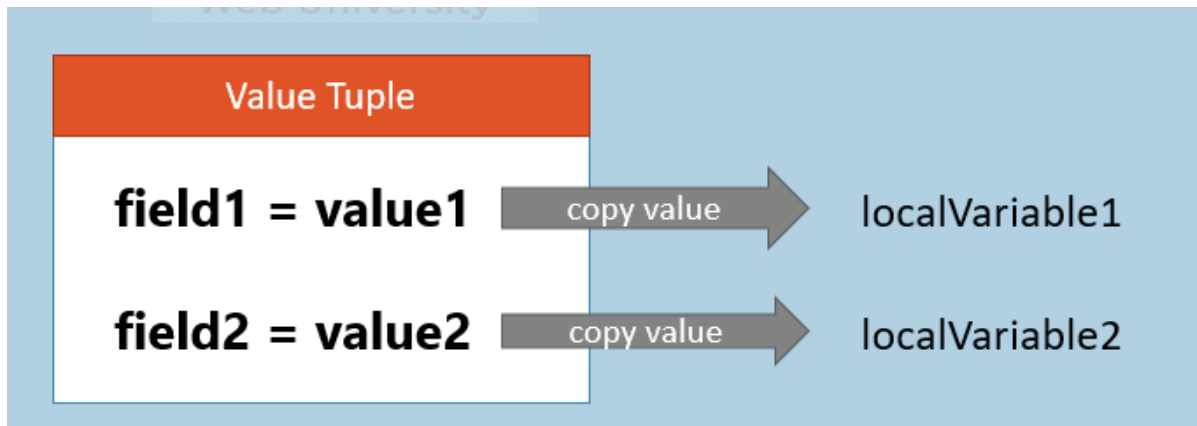
### Value Tuple

field1 = value1

field2 = value2

## Deconstruction

Deconstruction allows you to assign elements of value tuple into individual local variables.



### Step 1: Create Value Tuple

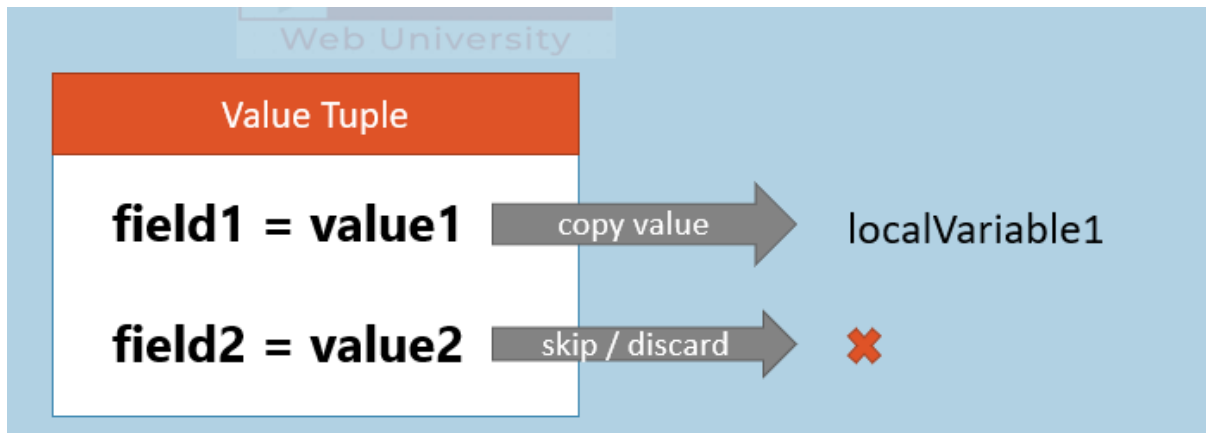
```
(type fieldName1, type fieldName2, ...) referenceVariable = (value1, value2, ... );
```

### Step 2: Deconstruction

```
(type variableName1, type variableName2, ...) = referenceVariable;
```

## Discards

Discard allows you to skip a value which you don't require, by using underscore ( `_` ).



### Step 1: Create Value Tuple

```
(type fieldName1, type fieldName2) referenceVariable = (value1, value2);
```

### Step 2: Deconstruction with Discard

```
(type variableName1, _ ) = referenceVariable;
```