**C# Tuple**

A tuple is a data structure which consists of multiple parts. It is the easiest way to represent a data set which has multiple values of different types. It was introduced in**.NET Framework 4.0.** In a tuple, we can add elements from 1 to 8. If try to add elements greater than eight, then the compiler will throw an error. Tuples are generally used when we want to create a data structure which contains objects with their properties and don’t want to create a separate type for that.

**Features of Tuples**

* It allows us to represent multiple data into a single data set.
* It allows us to create, manipulate, and access data sets.
* It returns multiple values from a method without using [**out**parameter](https://www.geeksforgeeks.org/out-parameter-with-examples-in-c-sharp/).
* It can also store duplicate elements.
* It allows us to pass multiple values to a method with the help of single parameters.

**Example:**

1

// Example of Tuple

2

using System;

3

​

4

public class Geeks

5

{

6

// Main Method

7

static public void Main()

8

{

// Creating a Tuple

var tuple = (123, "Hello", true);

// Accessing the Elements

Console.WriteLine(tuple.Item1);

Console.WriteLine(tuple.Item2);

Console.WriteLine(tuple.Item3);

}

17

}

**Output**

123

Hello

True

Before tuples, we have three ways to return more than one value from the method in C# which are Using Class or Struct types, Out parameters and Anonymous types which are returned through a Dynamic Return Type. But after Tuples, it becomes easy to represent a single set of data. There is no need to create any separate data structure. Instead, for this.

**Creating Tuple**

**In C#, there are mainly 2 ways to create the tuple**

* Using Constructor
* Using Create Method

**1. Using Constructor**

We can create a tuple by using the constructor of the tuple using[the tuple<T> class.](https://www.geeksforgeeks.org/c-sharp-tuple-class/) Where we can store elements starting from one to eight. If we try to add more than 8 compiler will throw an error.

**Syntax:**

/ Constructor for single elements  
Tuple <T1>(T1)  
  
// Constructor for two elements  
Tuple <T1, T2>(T1, T2)  
.  
.  
 // Constructor for eight elements  
Tuple <T1, T2, T3, T4, T5, T6, T7, TRest>(T1, T2, T3, T4, T5, T6, T7, TRest)

**Example:**

// Using tuple constructor

using System;

public class Geeks

{

static public void Main()

{

// Tuple with one element

Tuple<string> t1 = new Tuple<string>("GeeksforGeeks");

// Tuple with three elements

Tuple<string, string, int> t2 = new Tuple<string, string, int>("C#", "Python", 29);

// Tuple with eight elements

Tuple<int, int, int, int, int, int, int, Tuple<int>> t3 =

new Tuple<int, int, int, int, int, int, int, Tuple<int>>(1, 2, 3, 4, 5, 6, 7, new Tuple<int>(8));

Console.WriteLine(t1.Item1);

Console.WriteLine(t2.Item1);

Console.WriteLine(t3.Item1);

}

}

**Output**

**2. Using the Create Method**

When we use the tuple constructor to create a tuple we need to provide the type of each element stored in the tuple which makes your code cumbersome. So, C# provides another class that is Tuple class which contains the static methods for creating tuple objects without providing the type of each element.

**Syntax:**

// Method for 1-tuple  
Create(T1)  
  
// Method for 2-tuple  
Create(T1, T2)  
.  
.  
// Method for 8-tuple  
Create(T1, T2, T3, T4, T5, T6, T7, T8)

**Example:**

1

// Create tuple using tuple constructor

using System;

​public class Geeks

{

// Main method

static public void Main()

{

// Creating 1-tuple

// Using Create Method

var t1 = Tuple.Create("India");

// Creating 4-tuple

// Using Create Method

var t2 = Tuple.Create("C#", "Python", 29);

​

// Creating 8-tuple

// Using Create Method

var t3 = Tuple.Create(1,2,3,4,5,6,7,8);

Console.WriteLine(t1.Item1);

Console.WriteLine(t2.Item1);

Console.WriteLine(t3.Item1);

}

}

**Output**

GeeksforGeeks

C#

1

**Accessing a Tuple**

We can access the elements of a tuple by using the Item<elementNumber> property, here elementNumber is from 1 to 7 like Item1, Item 2, Item3, Item4, Item5, Item6, Item 7, etc. and the last element of 8-tuple is accessible by using Rest property as shown in the below example:

**Example:**

1

// Accessing the tuple

2

// using Item and Rest property

3

using System;

4

​

5

public class Geeks

6

{

7

static public void Main()

8

{

9

​

10

var t1 = Tuple.Create("GeeksforGeeks");

11

​

12

Console.WriteLine("1st element of t1 : " + t1.Item1);

13

Console.WriteLine();

14

​

15

// Creating 4-tuple

16

// Using Create Method

17

var t2 = Tuple.Create(12, 30, 40, 50);

18

​

19

// Accessing the element of Tuple

20

// Using Item property

21

Console.WriteLine("1st element of t2: " + t2.Item1);

22

Console.WriteLine();

23

​

24

// Creating 8-tuple

25

// Using Create Method

26

var t3 = Tuple.Create(13, "Geeks",

27

67, 89.90, 'g', 39939, "geek", 10);

28

​

29

// Accessing the element of Tuple

30

// Using Item property

31

// And print the 8th element of tuple

32

// using Rest property

33

Console.WriteLine("1st element of t3: " + t3.Item1);

34

Console.WriteLine("2nd lement of t3: " + t3.Item2);

35

Console.WriteLine("8th element of t3: " + t3.Rest);

36

}

37

}

**Output**

1st element of t1 : GeeksforGeeks

1st element of t2: 12

1st element of t3: 13

2nd lement of t3: Geeks

8th element of t3: (10)

**Nested Tuples**

In C#, we can create a tuple into another tuple we use nested tuples when you want to add more than eight elements in the same tuple. The nested tuple is accessible by using the Rest property as shown below example. We can add a nested tuple anywhere in the sequence, but it is recommended that place a nested tuple at the end of the sequence so that they can easily access from the Rest property.

**Example 1:**

1

// Illustration of nested tuple

2

using System;

3

​

4

public class Geeks

5

{

6

static public void Main()

7

{

8

// Nested Tuple

9

var My\_Tuple = Tuple.Create(13, "Geeks", 67, 89.90,

10

'g', 39939, "geek", Tuple.Create(12, 30, 40, 50));

11

12

Console.WriteLine("Element of My\_Tuple: " + My\_Tuple.Item1);

13

Console.WriteLine("Element of My\_Tuple: " + My\_Tuple.Item2);

14

15

// Accessing nested touple using rest property

16

Console.WriteLine("Element of Nested tuple: " + My\_Tuple.Rest);

17

Console.WriteLine("Element of Nested tuple: " + My\_Tuple.Rest.Item1.Item1);

18

}

19

}

**Output**

Element of My\_Tuple: 13

Element of My\_Tuple: Geeks

Element of Nested tuple: ((12, 30, 40, 50))

Element of Nested tuple: 12

**Example 2:**Accessing nested tuple other than the last place

1

// C# program to illustrate nested tuple

2

using System;

3

​

4

public class Geeks

5

{

6

static public void Main()

7

{

8

// Nested Tuple

9

// Here nested tuple is present

10

// at the place of 2nd element

11

var My\_Tuple = Tuple.Create(13, Tuple.Create(12, 30, 40,

12

50), 67, 89.90, 'g', 39939, 123, "geeks");

13

​

14

// Accessing nested Tuple

15

// at 2nd position

16

Console.WriteLine("1st element of tuple: " + My\_Tuple.Item1);

17

Console.WriteLine("1st Element of Nested Tuple: " + My\_Tuple.Item2.Item1);

18

Console.WriteLine("2nd Element of Nested Tuple: " + My\_Tuple.Item2.Item2);

19

Console.WriteLine("3rd Element of Nested Tuple: " + My\_Tuple.Item2.Item3);

20

Console.WriteLine("4th Element of Nested Tuple: " + My\_Tuple.Item2.Item4);

21

}

22

}

**Output**

1st element of tuple: 13

1st Element of Nested Tuple: 12

2nd Element of Nested Tuple: 30

3rd Element of Nested Tuple: 40

4th Element of Nested Tuple: 50

**Tuple as a Method Parameter**

In C#, we are allowed to pass a tuple as a method parameter as shown in the below example. Here we pass a tuple named mytuple in the PrintTheTuple() method and the elements of the tuple are accessed by using the Item<elementNumber> property.

**Example:**

1

// Tuple as a method parameter.

2

using System;

3

​

4

public class Geeks

5

{

6

static public void Main()

7

{

8

// Creating a tuple

9

var mytuple = Tuple.Create("GeeksforGeeks", 123, 90.8);

10

​

11

// Pass the tuple in the

12

// PrintTheTuple method

13

PrintTheTuple(mytuple);

14

}

15

​

16

static void PrintTheTuple(Tuple<string, int, double> mytuple)

17

{

18

Console.WriteLine("Element: " + mytuple.Item1);

19

Console.WriteLine("Element: " + mytuple.Item2);

20

Console.WriteLine("Element: " + mytuple.Item3);

21

}

22

}

**Output**

Element: GeeksforGeeks

Element: 123

Element: 90.8

**Tuple as a Return Type**

In C#, methods are allowed to use a tuple as a return type. In other words, a method can return a tuple as shown in the below example:

**Example:**

1

// C# program to illustrate

2

// how a method return tuple

3

using System;

4

​

5

public class Geeks

6

{

7

static public void Main()

8

{

9

// Return tuple is stored in mytuple

10

var mytuple = PrintTuple();

11

Console.WriteLine(mytuple.Item1);

12

Console.WriteLine(mytuple.Item2);

13

Console.WriteLine(mytuple.Item3);

14

}

15

​

16

// PrintTuple method return a tuple

17

static Tuple<string, string, string> PrintTuple()

18

{

19

return Tuple.Create("Geeks", "For", "Geeks");

20

}

21

}

**Output**

Geeks

For

Geeks

**Limitations of Tuple**

* It is of reference type not of value type.
* It is limited to eight elements. This means you cannot store more than eight elements without a nested tuple.
* These are only accessed by using the Item<elementNumber> property.