SCALA TUPLES

Scala tuple combines a fixed number of items together so that they can be passed around as a whole. Unlike an array or list, a tuple can hold objects with different types but they are also immutable. Here is an example of a tuple holding an integer, a string, and the console:

```
val t = (1, "hello", Console)
```

Which is syntactic sugar *shortcut* for the following:

```
val t = new Tuple3(1, "hello", Console)
```

The actual type of a tuple depends upon the number and of elements it contains and the types of those elements. Thus, the type of $99, \text{"}Luftballons\text{"}$ is Tuple2[Int, String]. The type of $'u','r', \text{"}the\text{"}, 1, 4, \text{"}me\text{"}$ is Tuple6[Char, Char, String, Int, Int, String]

Tuples are of type Tuple1, Tuple2, Tuple3 and so on. There currently is an upper limit of 22 in the Scala if you need more, then you can use a collection, not a tuple. For each TupleN type, where 1 <= N <= 22, Scala defines a number of element-access methods. Given the following definition:

```
val t = (4,3,2,1)
```

To access elements of a tuple t, you can use method t._1 to access the first element, t._2 to access the second, and so on. For example, the following expression computes the sum of all elements of t:

```
val sum = t._1 + t._2 + t._3 + t._4
```

You can use Tupel to write a method that takes a List[Double] and returns the count, the sum, and the sum of squares returned in a three-element Tuple, a Tuple3[Int, Double, Double]. They are also useful to pass a list of data values as messages between actors in concurrent programming. Following is the example showing usage of a tuple:

```
object Test {
   def main(args: Array[String]) {
      val t = (4,3,2,1)

      val sum = t._1 + t._2 + t._3 + t._4

      println( "Sum of elements: "  + sum )
   }
}
```

When the above code is compiled and executed, it produces the following result:

```
C:/>scalac Test.scala
C:/>scala Test
Sum of elements: 10

C:/>
```

Iterate over the Tuple:

You can use **Tuple.productIterator** method to iterate over all the elements of a Tuple. Following is the example to concatenate two Maps:

```
object Test {
   def main(args: Array[String]) {
      val t = (4,3,2,1)
```

```
      t.productIterator.foreach{ i =>println("Value = " + i )}
   }
}
```

When the above code is compiled and executed, it produces the following result:

```
C:/>scalac Test.scala
C:/>scala Test
Value = 4
Value = 3
Value = 2
Value = 1

C:/>
```

Convert to String:

You can use **Tuple.toString** method to concatenate all the elements of the tuple into a string. Following is the example to show the usage:

```
object Test {
   def main(args: Array[String]) {
      val t = new Tuple3(1, "hello", Console)

      println("Concatenated String: " + t.toString() )
   }
}
```

When the above code is compiled and executed, it produces the following result:

```
C:/>scalac Test.scala
C:/>scala Test
Concatenated String: (1,hello,scala.Console$@281acd47)

C:/>
```

Swap the Elements:

You can use **Tuple.swap** method to swap the elements of a Tuple2. Following is the example to show the usage:

```
object Test {
   def main(args: Array[String]) {
      val t = new Tuple2("Scala", "hello")

      println("Swapped Tuple: " + t.swap )
   }
}
```

When the above code is compiled and executed, it produces the following result:

```
C:/>scalac Test.scala
C:/>scala Test
Swapped tuple: (hello,Scala)

C:/>
```