Tuple is a lightweight data structure that allows you to group together multiple elements of different types. It provides a convenient way to represent and work with heterogeneous data. Tuples are immutable, meaning their values cannot be changed once set. You can create a tuple using the Tuple class or use tuple literals for brevity. They are commonly used when a method needs to return multiple values. Remember, tuples can enhance code readability and maintainability by encapsulating related data in a single entity.

you can declare and use tuples in various ways. Here are examples illustrating different ways to work with tuples:

1)Using Tuple Class:

```
Tuple<string, int, double> personTuple = new Tuple<string, int,
double>("John Doe", 30, 6.2);
```

2) Tuple Literal with Elements:

```
var personTuple = ("John Doe", 30, 6.2);
```

3) Tuple with Named Elements:

```
(string Name, int Age, double Height) personTuple = ("John Doe", 30,
6.2);
```

4) Tuple Deconstruction:

```
var (name, age, height) = personTuple;
Console.WriteLine($"Name: {name}, Age: {age}, Height: {height}");
```

* When unpacking the tuple is there alredy exist variable of that name then it will override that value.

5) Returning Tuple from Method:

```
public static (string, int) GetPersonInfo()
{
    // ... logic to fetch data
    return ("Jane Smith", 25);
}
```

6) Accessing Tuple Elements by Index:

```
var personTuple = ("Alice", 28, 5.8);
string name = personTuple.Item1;
int age = personTuple.Item2;
```

7) Creating Named Tuple Elements with Initialization:

```
var personTuple = (Name: "Bob", Age: 35, Height: 6.0);
```

8) Passing Tuple as Method Parameter:

```
public static void PrintPersonDetails((string, int, double) person)
{
    Console.WriteLine($"Name: {person.Item1}, Age: {person.Item2},
Height: {person.Item3}");
}
```

9) Combining Tuples:

```
var firstTuple = ("John", 25);
var secondTuple = ("Doe", 6.0);
var combinedTuple = (firstTuple.Item1, firstTuple.Item2,
secondTuple.Item2);
```

Example of usage of tuple:

Example in C# using pattern matching with tuples

```
public string GetPersonCategory((string Name, int Age) person)
{
    return person switch
    {
        var (_, age) when age < 18 => "Minor",
        var (_, age) when age >= 18 && age < 65 => "Adult",
        var (_, age) when age >= 65 => "Senior",
        _ => "Unknown"
    };
}
```

```
// Usage
var person = ("John", 25);
string category = GetPersonCategory(person);
Console.WriteLine($"Person category: {category}");
```

Discard in tuple

while Deconstructing a tuple if some value is not needed then Discard is used. Here is example of that

In this Name is not needed so it is discarded.

```
using System;

class Program
{
    static void Main()
    {
        (string Name, int Age, string Email) personTuple = ("John Doe",
25, "john@example.com");

        var (_, age, email) = personTuple;

        Console.WriteLine($"Age: {age}, Email: {email}");
    }
}
```

Real word Example:

Latitude and Longitude

```
using System;

class Program
{
    static void Main()
    {
        (double Latitude, double Longitude) coordinates1 = (37.7749, -
122.4194); // San Francisco
```

```
        (double Latitude, double Longitude) coordinates2 = (40.7128, -
74.0060);    // New York

        Console.WriteLine($"San Francisco Coordinates: Latitude
{coordinates1.Latitude}, Longitude {coordinates1.Longitude}");
        Console.WriteLine($"New York Coordinates: Latitude
{coordinates2.Latitude}, Longitude {coordinates2.Longitude}");
    }
}
```

XYZ cordinates

```
using System;

class Program
{
    static void Main()
    {

        (double X, double Y, double Z) point1 = (1.0, 2.5, 3.7);
        (double X, double Y, double Z) point2 = (-5.3, 8.1, 10.0);

        Console.WriteLine($"Point 1: X={point1.X}, Y={point1.Y},
Z={point1.Z}");
        Console.WriteLine($"Point 2: X={point2.X}, Y={point2.Y},
Z={point2.Z}");
    }
}
```