**Кортежи**

Функция кортежей предоставляет краткий синтаксис для группировки нескольких элементов данных в упрощенной структуре данных. Если коротко и простым языком, то кортежи позволяют возвращать более одного значения, в отличие от обычного метода, которые может возвращать только одно, или ни одного.

Ключевое слово Tuple <int,string> MethodName()

{

return Tuple.Create(1991, Nikolay)

}

Кортежи имеет смысл применять, в ситуациях где в одном методе могут меняться более

одной переменной, при том что с этими переменными было бы неплохо иметь дело. Например,

у нас есть база учеников с их оценками, и мы хотим вывести фамилии учеников и из какого

они класса, если их средняя оценка больше 4,7, для того чтобы отправить их на олимпиаду.

Или нам интересно посмотреть на график, какие числа выпали на рандоме от 0 до 1.

И закрасить броски, который больше 0,5 в белый, которые меньше в чёрный, и наглядно посмотреть

какие именно были значение у рандома и как себя ведёт генератор псевдослучайных чисел.

MethodName(random.NextDouble();)

Можно использовать, например, для обращения к базе данных, для того чтобы получить год рождения

и фамилию.

Принимаем на вход в методе MethodName(14356), где число 14356 это ID клиента в базе данных.

Затем в теле метода делаем то, что нам, необходимо, например обращаемся к базе данных

с данным ID, в ответ получаем класс. Попутно проверяем является ли совершеннолетним, и если да,

то возвращаем фамилию.

Общее между кортежами и классами то, что можно обращаться к разным данным. В случае с классами

используя свойства, в случае с кортежами используя MethodName().Item2; где 2 - это номер

возвращаемого класса, в данном случае string. Можно использовать и Item1, чтобы вернуть

значение 1991, например.

**Индексаторы**

Применение индексаторов упрощает работу с синтаксисом, позволяя создавать классы, структуры и интерфейсы,

к которым клиентские приложения могут обращаться так же, как к массиву.

Создадим класс, назовем его Student. Сделаем индексатор по оценкам по трём предметам.

public class Student

{

int[] grades = new int[3]{ 4, 3, 5}

public int this[int index]

{

set{grades[index] = value;}

get{return grades[index];}

}

}

Создадим в программе экземпляр нашего класса Student.

class Program

{

static void Main(string[] args)

{

Student \_student = new Student();

// На этом этапе сформировался экземпляр класса Student, в нём уже есть массив grades с оценками.

// Попробуем до них добраться и изменить.

Console.WriteLine(\_student[2]);

// Выведет на экран третью оценку, в данном случае это 5, т.к. нумерация в массивах начинается с 0.

// Изменим же эту оценку на тройку!

\_student[2] = 3;

Console.WriteLine(\_student[2]);

// Теперь мы увидим оценку, 3, вместо 5, т.к. мы её сами поменяли.

// Всё что можно сделать с индексаторами, можно сделать и с простым свойством с массивом,

// индексаторы не несут в себе упрощения, кроме того, что не надо запоминать как называется массив.

// Не слышал от коллег чтобы ими пользовались в банковской сфере или в машиностроительной.

// За свой опыт в гейм индустрии я так же не сталкивался.

}

}