

Тернарный оператор «?:»

Этот оператор используется для сокращения объема кода. Им можно заменять простые по сложности операторы if-else. Тернарный оператор имеет такую структуру:

логическое выражение ? выражение1 : выражение2

Сначала вычисляется логическое выражение.

Если оно истинно, то вычисляется выражение1, в противном случае - вычисляется выражение2.

Пример:

```
int b, c;  
c = -4;  
if (c >= 0)  
    {b = c;}  
else  
    {b = c*c; }
```

```
int b, c;  
c = -4;
```

```
b = c >= 0 ? c : c*c;
```

Логическое выражение

Выражение 1

логическое выражение ? выражение1 : выражение2

Результат b = 16

Пример:

```
int b, c;
c = -4;
if (c >= 0)
    {b = c;}
else
    {b = c * c; }
```

```
int b, c;
c = -4;
```

```
b = c >= 0 ? c : c * c;
```

Логическое выражение

Выражение 2

логическое выражение ? выражение1 : выражение2

Пример:

```
if (a % 2 == 0) //проверяем число на чётность путем нахождения остатка
от деления числа на 2
{
    Console.WriteLine("Число " + a + " - чётное");
}
else
{
    Console.WriteLine("Число " + a + " - нечётное");
}
```

Или

```
Console.WriteLine( a % 2 == 0 ? "Число чётное" : "Число нечётное" );
```

Оператор `return`

Оператор `return` завершает выполнение функции и возвращает управление вызывающей функции.

Выполнение возобновляется в вызывающей функции в точке сразу после вызова.

Оператор `return` может возвращать значение, передавая его вызывающей функции.

Программирование на языке Си#

Функции

Если переменные хранят некоторые значения, то (функции) методы содержат собой набор операторов, которые выполняют определенные действия. По сути метод - это именованный блок кода, который выполняет некоторые действия.

В чем разница между методами и функциями?

В ООП языках, таких как C #, Java и т.д. используется термин метод, а для не-ООП программирования, таких как консольные "C" и других – функция.

В чем разница между процедурами и функциями?
Процедура – выполняет, а функция выполняет и возвращает в основную программу.

Например, по умолчанию консольная программа на языке C# должна содержать как минимум один метод - метод Main, который является точкой входа в приложение:

```
static void Main(string[] args)
{
}

```

Ключевое слово `static` является модификатором. Далее идет тип возвращаемого значения. В данном случае ключевое слово `void` указывает на то, что метод ничего не возвращает.

Далее идет название метода - `Main` и в скобках параметры - `string[] args`. И в фигурные скобки заключено тело метода - все действия, которые он выполняет. В данном случае метод `Main` пуст, он не содержит никаких операторов и по сути ничего не выполняет.


```
1 using System;
2
3 namespace HelloApp
4 {
5     class Program
6     {
7         static void Main(string[] args)
8         {
9
10        }
11
12        static void SayHello()
13        {
14            Console.WriteLine("Hello");
15        }
16        static void SayGoodbye()
17        {
18            Console.WriteLine("GoodBye");
19        }
20    }
21 }
```

В данном случае определены еще два метода: SayHello и SayGoodbye.

Оба метода определены в рамках класса Program, они имеют модификатор static, а в качестве возвращаемого типа для них определен тип void. То есть данные методы ничего не возвращают, просто производят некоторые действия. И также оба метода не имеют никаких параметров, поэтому после названия метода указаны пустые скобки.

```
1 using System;
2
3 namespace HelloApp
4 {
5     class Program
6     {
7         static void Main(string[] args)
8         {
9
10        }
11
12        static void SayHello()
13        {
14            Console.WriteLine("Hello");
15        }
16        static void SayGoodbye()
17        {
18            Console.WriteLine("GoodBye");
19        }
20    }
21 }
```

Оба метода выводят на консоль некоторую строку. Причем для вывода на консоль методы используют другой метод, который определен в .NET по умолчанию - `Console.WriteLine()`.

Но если мы запустим данную программу, то мы не увидим никаких сообщений, которые должны выводить методы `SayHello` и `SayGoodbye`.

Потому что стартовой точкой является метод `Main`. При запуске программы выполняется только метод `Main` и все операторы, которые составляют тело этого метода. Все остальные методы не выполняются.

ВЫЗОВ МЕТОДА

Чтобы использовать методы SayHello и SayGoodbye в программе, нам надо вызвать их в методе Main.

Для вызова метода указывается его имя, после которого в скобках идут значения для его параметров (если метод принимает параметры).

```
1 | название_метода (значения_для_параметров_метода);
```

Например, вызовем методы SayHello и SayGoodbye:

```
{
  class Program
  {
    static void Main(string[] args)
    {
      SayHello();
      SayGoodbye();

      Console.ReadKey();
    }

    static void SayHello()
    {
      Console.WriteLine("Hello");
    }
    static void SayGoodbye()
    {
      Console.WriteLine("GoodBye");
    }
  }
}
```

Преимуществом методов является то, что их можно повторно и многократно вызывать в различных частях программы.

Возвращение значения

Если метод имеет любой другой тип, отличный от `void`, то такой метод обязан вернуть значение этого типа. Для этого применяется оператор **return**, после которого идет возвращаемое значение:

```
1 return возвращаемое значение;
```

Пример:

```
1 static string GetHello()  
2 {  
3     return "Hello";  
4 }  
5 static int GetSum()  
6 {  
7     int x = 2;  
8     int y = 3;  
9     int z = x + y;  
10    return z;  
11 }
```

Метод `GetHello` имеет тип `string`, следовательно, он должен вернуть строку. Поэтому в теле метода используется оператор `return`, после которого указана возвращаемая строка.

Метод `GetSum` имеет тип `int`, следовательно, он должен вернуть значение типа `int` - целое число. Поэтому в теле метода используется оператор `return`, после которого указано возвращаемое число (в данном случае результат суммы переменных `x` и `y`).

Где ошибка?

```
1 static string GetHello()  
2 {  
3     Console.WriteLine("Hello");  
4 }
```

Нет return

```
1 static int GetSum()  
2 {  
3     int x = 2;  
4     int y = 3;  
5     return "5";  
6 }
```

**Тип метода int , а
возвращает символ**

```
1 static string GetHello()  
2 {  
3     return "Hello";  
4     Console.WriteLine("After return");  
5 }
```

Return раньше чем вывод

Синтаксис объявления функции (метода):

```
[атрибуты] [спецификаторы] тип_результата имя_метода  
([список_формальных_параметров])  
{  
    тело_метода;  
    return значение;  
}
```

```
[модификаторы] тип_возвращаемого_значения название_метода ([параметры])  
{  
    // тело метода  
}
```

Модификаторы и параметры необязательны.

Задача:

У нас есть разные люди с данными в виде отдельных фамилии, имени, отчества, которые надо вывести на экран - вида Пушкин Александр Сергеевич и Пушкин А.С.

Чтобы задача была правдоподобней можно имитировать ввод данных пользователем или загрузку из внешнего источника, но все это будет пустой тратой времени - реальные приложения все равно работают с графическим и/или веб-интерфейсом.

Просто держим в уме что в реальности людей не два, а две тысячи и заранее их имена не известны.

```
string name = "Александр";  
string otchestvo = "Сергеевич";  
string surname = "Пушкин";  
  
string name2 = "Наталья";  
string otchestvo2 = "Николаевна";  
string surname2 = "Гончарова";  
  
System.Console.WriteLine(surname + " " + name + " " + otchestvo);  
System.Console.WriteLine(surname2 + " " + name2 + " " + otchestvo2);  
  
System.Console.ReadLine();
```

Логичнее всего вынести повторяющиеся куски кода в отдельное место, дать им имя и во всех остальных местах программы вызывать их по этому имени. Иными словами *сделать функции* - куски *кода с собственными именем*, которые принимают на вход какие-то данные, что-то с ними делают и возвращают обратно какие-то данные (хотя возможен вариант когда они ничего не принимают и не возвращают, просто что-то делают!!!!).

Пример функции (метода)

```
public static string CreateFio(string surname, string name, string
otchestvo)
{
    string fio = surname + " " + name + " " + otchestvo;
    return fio;
}
```

string означает, что функция вернет назад строку,
CreateFio(string surname, string name, string otchestvo) - название функции
и описание того, что она принимает на вход три строки.

```
[атрибуты] [спецификаторы] тип_результата имя_метода  
([список_формальных_параметров])  
{  
    тело_метода;  
    return значение;  
}
```

```
public static string CreateFio(string surname, string name, string  
otchestvo)  
{  
    string fio = surname + " " + name + " " + otchestvo;  
    return fio;  
}
```

Если бы функция ничего не принимала и ничего не возвращала, ее описание выглядело бы так:

Пример:

```
class Program
{
    static void Func() // дополнительная функция (метод)
    {
        Console.Write( "x=");
        double x,y;
        x=Convert.ToInt32(Console.ReadLine());
        y=x*x;
        Console.Write( "y="+y);
    }
    static void Main() // точка входа в программу
    {
        Func() ;// первый вызов функции
        Func() ;// второй вызов функции
    }
}
```

Если функция принимает значения и возвращает результат, ее описание выглядело бы так:

Пример:

```
class Program
{
    static void Func() // дополнительная функция (метод)
    {
        return x*x;
    }

    static void Main() // точка входа в программу
    {
        double x,y;
        Console.Write( "x=");
        x=ConvertToInt32(Console.ReadLine());
        y=Func(x) ;// первый вызов функции
        Console.Write( "y="+y);
        Console.Write( "x=");
        x=ConvertToInt32(Console.ReadLine());
        y=Func(x) ;// второй вызов функции
        Console.Write( "y="+y);
    }
}
```

В чем разница?

```
class Program
{
    static void Func() // дополнительная функция (метод)
    {
        Console.Write( "x=");
        double x,y;
        x=Convert.ToInt32(Console.ReadLine());
        y=x*x;
        Console.Write( "y="+y);
    }
    static void Main() // точка входа в программу
    {
        Func() ;// первый вызов функции
        Func() ;// второй вызов функции
    }
}
```

```
class Program
{
    static void Func() // дополнительная функция (метод)
    {
        return x*x;
    }
    static void Main() // точка входа в программу
    {
        double x,y;
        Console.Write( "x=");
        x=Convert.ToInt32(Console.ReadLine());
        y=Func(x) ;// первый вызов функции
        Console.Write( "y="+y);
        Console.Write( "x=");
        x=Convert.ToInt32(Console.ReadLine());
        y=Func(x) ;// второй вызов функции
        Console.Write( "y="+y);
    }
}
```

Напишите функцию `static int min (int a, int b, int c, int d)`, находящую наибольшее из трех данных чисел.

Входные данные

Вводится три целых числа.

Выходные данные

Необходимо вывести наибольшее из 3-х данных чисел.

Пример:

```
class Program
{
    static int Func(int x, int y)    // 1 – описание метода Func
    {
        return (x > y) ? x : y;
    }
    static void Main()
    {
        int a,b,c, max;
        Console.Write( "a=");
        a=ConvertToInt32(Console.ReadLine());
        Console.Write( "b=");
        b=ConvertToInt32(Console.ReadLine());
        Console.Write( "c=");
        c=ConvertToInt32(Console.ReadLine());
        max=Func(Func(a,b),c)
        Console.Write( "max="+max);
    }
}
```

1. Вводится три числа.

```
Console.Write( "a=");  
    a=Convert.ToInt32(Console.ReadLine());  
    Console.Write( "b=");  
    b=Convert.ToInt32(Console.ReadLine());  
    Console.Write( "c=");  
    c=Convert.ToInt32(Console.ReadLine());
```

2. Функция нахождения большего из двух.

```
static int Func(int x, int y)
{
    return (x > y) ? x : y;
}
```

3. Вызов функции нахождения большего из двух.

```
int max = (Func(a, b))
```

3. Вызов функции нахождения большего из двух два раза.

```
int max = Func(Func(a, b), c);
```

4. Вывод максимального.

```
Console.Write( "max="+max);
```