

Лекция 2.
Строки.
Арифметика.
Класс Math
Чтение с консоли

Строки

- Литералы строк заключаются в двойные кавычки: “Test”, “Строка”
- Строки имеют тип `String`
`String s = “Hello”;`
- Использование кавычек внутри литерала строки (пишется `\"`):

```
String s = “\"Hello\" world”; // “Hello” world
```

- Это называется **экранированием**
- Вместо имени типа `String` можно использовать ключевое слово `string`. Рекомендую делать именно так

Специальные последовательности

- Литералы строк могут содержать специальные последовательности (есть и другие):

- `\n` перевод строки (Enter)
- `\r` возврат каретки
- `\t` табуляция
- `\\` символ `\`

- Пример:

`"This is first line\nThis is second line"`

- Результат печати:

This is first line

This is second line

Конкатенация строк

- В C# строки можно объединять друг с другом при помощи оператора +
- Это называется **конкатенацией** строк (добавление второй строки в конец первой)
- `string s1 = "Hello " + "world"; // Hello world`
- Кроме того, если к строке прибавлять значения других типов, то они будут преобразовываться в строки, и эти строки будут конкатенироваться
- `string s1 = "Hello " + 3; // Hello 3`
- `string s2 = 3 + " tigers"; // 3 tigers`

Переводы строк

- В разных операционных системах в текстовых файлах перевод строки обозначается разным образом
- В UNIX-системах это просто `\n`
- В Windows – последовательность из двух символов `\r\n`
- Так как программы на C# потенциально могут исполняться на многих ОС, то есть команда для получения нужного перевода строки, в зависимости от системы, где выполняется программа:
- `Environment.NewLine`

Пример использования переводов строк

- Не совсем верно – всегда используется UNIX-стиль
- `Console.WriteLine("Строка 1\nСтрока 2");`
- Верно (на каждой платформе свой перевод строки):
- `Console.WriteLine("Строка 1" + Environment.NewLine + "Строка 2");`

Задача

1. Вывести при помощи одного вызова `Console.WriteLine` сразу несколько строк при помощи символа перевода строки
2. Вывести следующие строки:
"30" августа 2014г.
C:\Windows\System32\Drivers\etc\hosts
3. Посчитать некоторое выражение и распечатать его результат в виде строки. Вместо 4 – ваш результат:
Результат = 4

Verbatim строки

- В C# есть особые литералы строк, начинающиеся с @:
- `string s = @"C:\Windows\System32\Drivers\etc\hosts";`
- В таких литералах не работают последовательности, начинающиеся с \
Поэтому символ \ не требует экранирования
- Такие литералы могут быть многострочными:
- `string s = @"Строка 1
Строка2";`

Verbatim строки

- Если в такой строке хочется внутри написать кавычку, то её нужно удвоить:
- `string s = @"Агентство ""Диамант""";`
`Console.WriteLine(s);`
`// Агентство "Диамант"`

Числа и арифметика

- В C# есть несколько разных типов для целых и вещественных чисел
- Вещественные типы – всегда знаковые
- Отличаются диапазоном допустимых значений, точностью и размером переменной в байтах

Название типа	Диапазон значений	Размер в байтах
float	$\pm 1.5e-45 \dots \pm 3.4e+38$	4
double	$\pm 5.0e-324 \dots \pm 1.7e+308$	8

- На практике обычно используют **double**, так как у него выше точность

Числа и арифметика

- Целочисленные типы делятся на знаковые и беззнаковые
- На практике обычно применяют только знаковые типы

Название типа	Диапазон значений	Размер в байтах
sbyte	-128..127	1
byte	0..255	1
short	-32 768..32 767	2
ushort	0..65 535	2
int	-2 147 483 648 .. 2 147 483 647	4
uint	0 .. 4 294 967 295	4
long	-9 223 372 036 854 775 808 .. 9 223 372 036 854 775 807	8
ulong	0.. 18 446 744 073 709 551 615	8

- Для целых чисел чаще всего используют `int`

Литералы чисел

- Литерал числа считается вещественным, если в нем есть точка, либо символ экспоненты
- | | |
|----------|------------------------------------|
| 1.0 | // вещественное число 1.0 (double) |
| 1 | // целое число 1 (int) |
| 1. | // вещественное число 1.0 |
| 0.4 | // вещественное число 0.4 |
| .1 | // вещественное число 0.1 |
| 1e34 | // $1 * 10$ в степени 34 |
| -1.43e-3 | // $-1.43 * 10$ в степени -3 |
| 1.0f | // вещ. число 1.0, но float |
| 1.0D | // вещ. число 1.0, double |
| 1L | // long 1.0 |

Числа и арифметика

- Арифметические операторы: $+$ $-$ $*$ $/$ $\%$
- $\%$ - остаток от целочисленного деления
`int r = 25 % 7; // 4`
- Приоритеты операторов – как в математике. Можно ставить скобки в выражениях
- Пример: $5 + 5 * (3 - 2)$

Важное замечание

- Если оба числа целые, то деление будет целочисленным. Если хотя бы одно число вещественное, то деление будет вещественным:
- $5 / 2$ // 2, т.к. оба числа целые
 $5.0 / 2$ // 2.5, т.к. первое число – вещ.

Как поделить вещественно целые числа?

- `int a = 5;`
- `int b = 2;`
- `double c = (double)a / b;`
- `(double)a` — называется **преобразованием типа** (**приведение типа**)
- Выражение `(double)a` выдает значение переменной `a` как вещественное число

Задача

- Попрактиковаться с арифметическими операторами: сделать по 1 примеру для каждого оператора для целых чисел и для вещественных чисел, вывести результат в консоль
- Операторы: $+$ $-$ $*$ $/$ $\%$
- Проверить как работает целочисленное и вещественное деление

Приоритеты операций

- Приоритеты операций как в математике:
 - Сложение и вычитание имеют равный приоритет, выполняются слева направо
 - Умножение и деление имеют равный приоритет, выполняются слева направо
 - Приоритет умножения и деления выше, чем у сложения и вычитания
 - Скобки повышают приоритет операций

Устная практика

- Чему равен x ?
- $x = 3 + 5 * 2$
- Как записать на C# следующее выражение?
- $\frac{3x+5}{2y} + 3y$

Задача на дом «Вычисление выражений»

- Посчитайте на C# следующие выражения:

- $$x = 3 - \frac{56 - 12}{44} * \frac{4}{2}$$

- $$y = \frac{2x}{33 - x}$$

- $$z = \frac{-x}{2y}$$

Переполнение типов

- Тип имеет ограниченный диапазон значений, и если за него выйти, то происходит переполнение
- `int x = 2147483647;`
`Console.WriteLine(x);`
`// 2147483647`
- `x = x + 1;`
`Console.WriteLine(x);`
`// -2147483648`
- Почему получилось именно такое число?

Хранение чисел

- Все числа хранятся в двоичной системе счисления, т.е. представляются последовательностью нулей и единиц
- Например, в переменной типа `sbyte` 8 бит. Рассмотрим, например, число 127 – максимальное значение типа `sbyte`

0	1	1	1	1	1	1	1
---	---	---	---	---	---	---	---

- Причем в целых числах первый бит предназначен для хранения знака числа. Этот бит равен 0 для положительных чисел, 1 – для отрицательных

Операция сложения

- Сложение двух чисел выполняется побитово
- Допустим, мы складываем числа 21 и 13, получаем 34

0	0	0	1	0	1	0	1
---	---	---	---	---	---	---	---

+

0	0	0	0	1	1	0	1
---	---	---	---	---	---	---	---

=

0	0	1	0	0	0	1	0
---	---	---	---	---	---	---	---

Операция сложения

- Сложим 127 и 1, получается уже некоторое отрицательное число, т.к. первый бит получился 1. И это число -128
- Но почему это именно -128?

	0	1	1	1	1	1	1
+							
	0	0	0	0	0	0	1
=							
	1	0	0	0	0	0	0

Дополнительный код

- Так выглядит число 127

0	1	1	1	1	1	1	1
---	---	---	---	---	---	---	---

- Так выглядит число -127

1	0	0	0	0	0	0	1
---	---	---	---	---	---	---	---

- Почему -127 выглядит именно так?
- А выглядит так, чтобы если сложить 127 и -127, получился 0

1	0	0	0	0	0	0	0	0
---	---	---	---	---	---	---	---	---

- Первая единица выходит за границы 8 байт, и поэтому отбрасывается
- Такой способ хранения отрицательных чисел называется **дополнительным кодом**

Переполнение типов

- `int x = 2147483647;`
`Console.WriteLine(x);`
`// 2147483647`
- `x = x + 1;`
`Console.WriteLine(x);`
`// -2147483648`
- Почему получилось именно такое число?
- Потому что число в результате сложения происходит переполнение, в двоичном виде получилось число 10000...0, а оно соответствует самому маленькому отрицательному числу в типе

Приведение вещественных типов к целым

- `int c = (int)3.5; // 3`
- При таком приведении отбрасывается дробная часть
- `int d = (int)-3.7; // -3`
- Не нужно путать с округлением

Преобразования числовых типов

- Преобразования числовых типов можно поделить на две группы:
 - **Безопасные** (без потери данных)
 - **Небезопасные** (с потерей данных)

Безопасные преобразования

- Без потери данных
- Сюда относятся:
 - Преобразование типа с меньшим диапазоном в тип с большим диапазоном
 - Например, `short` в `int` или `float` в `double`
 - Преобразования целых чисел в вещественные:
 - Например, `long` в `float` и `double`
- Безопасные преобразования можно выполнять автоматически (**неявно**)
- `int x = 3;`
`long y = x;`

Небезопасные преобразования

- С возможной потерей данных
- Сюда относятся:
 - Преобразование типа с большим диапазоном в тип с меньшим диапазоном
 - Например, `int` в `short` или `double` в `float`
 - Преобразования вещественных чисел в целые:
 - `double` в `int`
- Небезопасные преобразования можно выполнять только **явно** при помощи преобразования типов
- `long a = 3;`
`int b = (int)a;`

Пример

- `float x = 3;`
`double y = x;` *// безопасно, неявное преобр-е*
- `long a = 3;`
`int b = a;` *// небезопасно, ошибка компиляции*
`int b = (int) a;` *// надо преобразовывать явно*

Класс Math

- В C# имеется класс **Math** для выполнения более сложных арифметических операций, чем $+$ $-$ $*$ $/$ $\%$
- Например:
 - возведение в степень
 - извлечение квадратного корня
 - тригонометрические функции
 - логарифмы
 - округление

Класс Math

- Квадратный корень:
`double a = Math.Sqrt(4); // 2`
- Возведение в степень:
`double a = Math.Pow(2, 3); // 8`
- Модуль числа:
`double m = Math.Abs(-3.1); // 3.1`
- Логарифмы:
`Math.Log(x) // натуральный логарифм`
`Math.Log10(x) // десятичный логарифм`

Класс Math

- Константы Пи и е:

```
double pi = Math.PI;
```

```
double e = Math.E;
```

```
// не создавайте для них свои переменные, используйте  
// сами Math.PI и Math.E напрямую
```

- Преобразование между градусами и радианами:

```
double degree1 = 30;
```

```
double radians = (Math.PI / 180.0) * degree1;
```

```
double degree2 = (180.0 / Math.PI) * radians;
```

Класс Math

- Тригонометрические функции (в радианах):

// сначала переводим угол в радианы

```
double angle = (Math.PI / 180.0) * 30;
```

```
double sin = Math.sin(angle); // примерно 0.5
```

```
double cos = Math.cos(angle); // примерно 0.866
```

```
double tan = Math.tan(angle); // примерно 0.577
```

- Имеются обратные тригонометрические функции и гиперболические функции

Виды округления

- В классе **Math** есть 3 вида округления + есть округление при помощи приведения к целому типу

	(int)	Math.Floor	Math.Ceiling	Math.Round
Пояснение	Отбрасывает дробную часть	Округляет вниз по числовой оси	Округляет вверх по числовой оси	Округляет к ближайшему целому. Если 0.5, то вверх по числовой оси
1.4	1	1	2	1
1.5	1	1	2	2
1.6	1	1	2	2
-1.4	-1	-2	-1	-1
-1.5	-1	-2	-1	-1
-1.6	-1	-2	-1	-2

Задача

- Написать программу, которая вычисляет и печатает площадь круга с радиусом r .
- Значение r задать самим в тексте программы
- Использовать класс `Math`: тригонометрические функции, возведение в степень, квадратный корень, константа π и др.

Задача на дом «Окружность»

- Написать программу, которая вычисляет и печатает:
 - Площадь круга и длину окружности с заданным радиусом.
Значение радиуса задать самим в тексте программы
 - Посчитать радиус окружности с заданной площадью круга.
Значение площади задать самим в тексте программы
 - * Посчитать площадь сектора с заданными радиусом и углом в градусах.
Значения радиуса и угла задать в тексте программы
- Использовать класс `Math`: возведение в степень, квадратный корень, константа π и др.
- Имена переменных должны быть хорошими, не ориентируемся на математические обозначения

Четность, кратность

- `int r = x % y;`
- Когда остаток от деления x на y равен 0, то говорят, что **x кратно y**
- То есть делится нацело
- Число является **четным**, если оно кратно 2 (то есть остаток от деления на 2 равен 0)
- Иначе – число является **нечетным** (остаток равен 1)

Использование деления и остатка

- Пусть x, y – два некоторых целых числа
- `int a = x / y;`
`int r = x % y;`
- Тогда r лежит в интервале от 0 до $y - 1$ включительно
- Это свойство позволяет «зациклить» номера
- Примеры: круг; положение квартиры на площадке

Получение цифр числа

- Как получить последнюю цифру числа?
- Просто взять остаток от деления на 10
- `int x = 327;`
`int lastDigit = x % 10; // 7`
- Как получить предпоследнюю цифру числа?
- Поделить на 10, потом взять остаток от деления на 10
- `int x = 327;`
`int digit = x / 10 % 10; // 2`

Получение цифр числа

- Как получить n-ю с конца цифру числа?
- Поделить на 10^{n-1} , затем взять остаток от деления на 10
- ```
int x = 327;
int n = 3;
int digit = (x / (int)Math.Pow(10, n - 1)) % 10; // 3
```

# Задача на курс «Квартиры»

- Есть дом с  $N$  этажами и  $M$  подъездами. Все подъезды одинаковые, на каждом этаже в подъезде 4 квартиры.
- Считаем, что номера квартир на лестничной площадке распределяются так:

|   |   |
|---|---|
| 2 | 3 |
| 1 | 4 |
- То есть можно сказать, что квартира 1 – ближняя слева, квартира 2 – дальняя слева, квартира 3 – дальняя справа, квартира 4 – ближняя справа
- Прочитать с консоли числа  $N$ ,  $M$  и целое число  $K$  – номер квартиры. По введенному числу  $K$  выдать номер подъезда и этажа, где находится эта квартира, а также положение квартиры на лестничной площадке
- Выдать сообщение, если квартиры с таким номером нет в доме

# Комментарии

- В коде можно писать **комментарии**
- Это текст внутри программы, который отбрасывается компилятором
- Используются для следующих целей:
  - Поясняющий текст для улучшения понимания кода другими людьми
  - Временное отключение части кода
  - TODO – заметки, что потом надо будет что-то сделать
  - Документирование кода

# Однострочные комментарии

- Действуют от // до конца строки

- // Это комментарий

`int i = 4; // это тоже комментарий, после кода`

`// Следующая строка – не является кодом и не выполнится`

`// Console.WriteLine(i);`

`// это пример временного отключения части кода`

# Многострочные комментарии

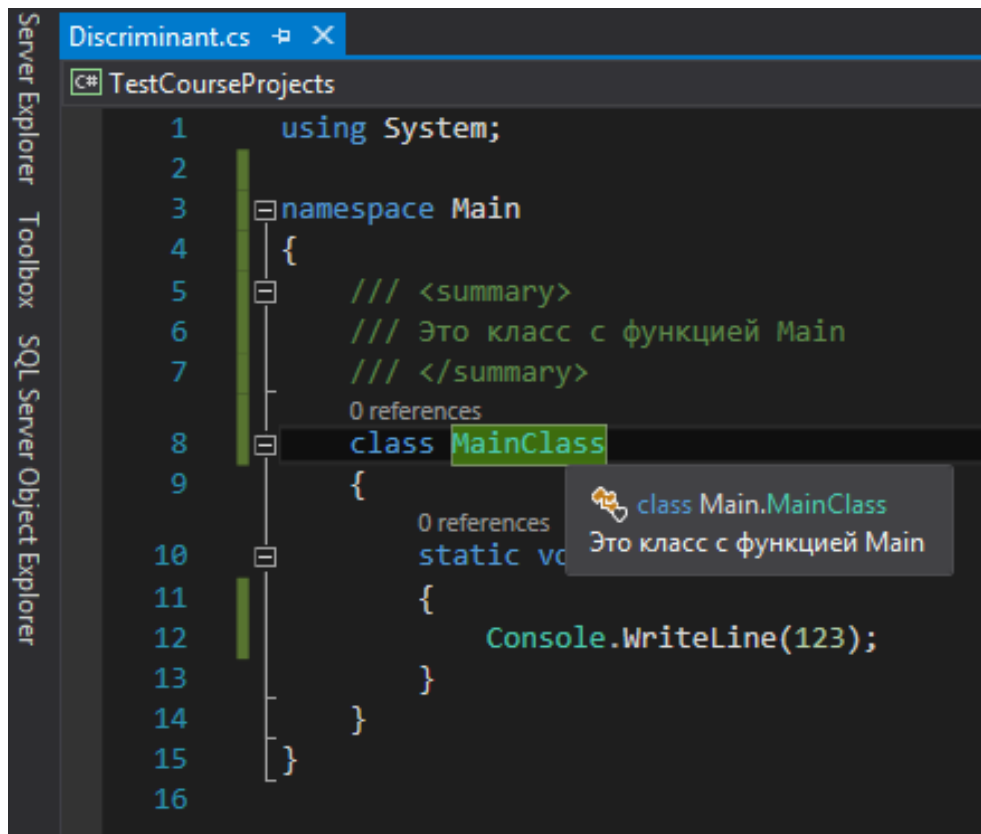
- Действуют от `/*` до `*/`  
`int a = 4; /* первая строка комментария`  
`Вторая строка комментария */`
- Запрещено использовать вложенные многострочные комментарии `/* /* */ */`

# TODO комментарии

- С точки зрения языка – это обычные комментарии
- Но среды разработки их по-особому трактуют:
  - Выделяют их цветом (к сожалению, не Visual Studio)
  - В Visual Studio есть специальный раздел с такими комментариями: View -> Task List
- `//TODO: потом реализовать чтение с консоли`  
`int width = 4;`  
`int height = 3;`  
`Console.WriteLine(width * height);`

# Документирующие комментарии

- Используются для написания документации прямо в коде
- Они однострочные, начинаются с ///
- Если навести на класс или функцию, то отобразится ее документация





# Когда нужны комментарии?

- Комментарии пишут достаточно редко
  - Когда в коде принято неочевидное решение, которое, казалось бы, можно упростить, или является неверным, но по некоторой причине «так надо»
  - Временное отключение кода
  - TODO комментарии
  - Документирующие комментарии
- Комментарии не пишутся:
  - Чтобы пояснить имя переменной/функции/класса – имена должны и так быть понятными

# Задача

- Прокомментировать текст предыдущей задачи всеми типами комментариев
- Обязательно попробуйте TODO комментарии

# Чтение с консоли

- ```
static void Main()
{
    Console.Write("Введите число: ");
    double a = Convert.ToDouble(Console.ReadLine());
    Console.Write("Введите второе число: ");
    double b = Convert.ToDouble(Console.ReadLine());
    Console.WriteLine("a - b = " + (a - b));
}
```
- Заметим, что вводить вещественные числа нужно с использованием запятой: 3,4
- Это потому что `Convert.ToDouble` учитывает языковые настройки ОС, а у нас РФ

Чтение с консоли

- В C# нет команд, которые сразу умеют прочитать с консоли число
- Поэтому приходится сначала прочитать строку, а потом преобразовать её в нужный числовой тип
- `string text = Console.ReadLine();`
`double number = Convert.ToDouble(text);`
- Но можно писать короче:
- `double number = Convert.ToDouble(Console.ReadLine());`

Чтение с консоли

- Используя другие функции класса `Convert`, можно преобразовывать строку в другие типы
- `int x = Convert.ToInt32(Console.ReadLine());`
- `long x = Convert.ToInt64(Console.ReadLine());`
- `short x = Convert.ToInt16(Console.ReadLine());`
- `float x = Convert.ToSingle(Console.ReadLine());`
- И т.д.
- Заметим, что тут имена типов, например, `int`, не совпадают с именем метода класса `Convert`, например, `Int32`
- Дело в том, что `int` – это обозначение для `Int32`, также как `string` – это обозначение типа `String`

Задача

- Написать программу, читающую с консоли длину и ширину прямоугольника, и печатающую его площадь
- * Посчитайте и периметр
- Считать, что длина и ширина – вещественные числа

Задача на дом «Приветствие»

- Написать программу, которая просит ввести ваше имя, а затем выводит в консоль приветствие. Для чтения использовать `Console.ReadLine()`

- Пример:

Введите ваше имя: `// это печатает программа`

Павел `// это ввожу я`

Привет, Павел! `// это печатает программа`

Печать в консоль

- `Console.WriteLine(аргумент)`
- Печатает переданный аргумент, а потом вставляет перевод строки (Enter)
- `Console.Write(аргумент)`
- Печатает переданный аргумент, но не вставляет перевод строки

Печать в консоль

- `Console.WriteLine(formatString, arg0...)`
- Принимает **строку форматирования** первым аргументом, а затем через запятую указываются аргументы, которые будут вставлены в эту строку
- Пример:
- `int a = 7 * 5;`
`Console.WriteLine("Result = {0}", a);` `// Result = 35`

Console.WriteLine

- `int x = 35;`
`int y = 34;`
`Console.WriteLine("X = {0}, Y = {1}", x, y);` // X = 35, Y = 34
- **Строка форматирования** – строка, некоторые части которой имеют особый смысл
- Такие части окружаются фигурными скобками, внутри указываются порядковые номера аргументов, с 0
- Аргументы, которые мы передаем в `WriteLine`, подставляются вместо этих меток
- Например, вместо `{0}` вставится значение `x = 35`, вместо `{1}` – значение `y = 34`

Зачем нужен такой вариант?

- Он позволяет формировать сложные строки из нескольких строк и переменных в читаемом виде
- Пример:
 - **Конкатенация:**
 - `Console.WriteLine("X = " + x + ", y = " + y);`
 - **Строка форматирования:**
 - `Console.WriteLine("X = {0}, y = {1}", x, y);`

Зачем нужен такой вариант?

- Он позволяет вывести данные в разном формате
- Например, у вещественного числа может быть много знаков после запятой
- `double x = 33.3333333;`
- Можно указать сколько знаков вывести, нужно ли вывести в экспоненциальной форме, нужно ли добавлять пробелы для форматирования и т.д.
- `Console.WriteLine("{0}", x);` `// 33,3333333`
`Console.WriteLine("{0:e}", x);` `// 3,3333333e+001`
`Console.WriteLine("{0:f2}", x);` `// 33,33`

Последовательности после :

- Для каждого типа данных используется свой символ

Спецификатор	Тип	Пример
D или d	Десятичное целое число	159
X или x	Шестнадцатеричное целое число	9f
F или f	Вещественное число с точкой	15.9
E или e	Вещественное число в экспоненциальной форме	1.59e+001

- После e или f можно писать число, оно означает число знаков после запятой
- `double x = 33.3333;`
`Console.WriteLine("{0:f2}", x);` // 33,33

Фигурные скобки в строке формата

- Если в строке форматирования хочется вывести фигурные скобки, то их нужно удвоить:
- ```
double x = 1;
double y = 2;
Console.WriteLine("{ { x = {0}, y = {1} } }", x, y);
// { x = 1, y = 2 }
```

# Дополнительные параметры формата

- Можно после номера последовательности указать запятую и целое число, тогда это будет ширина выводимого значения
- Если значение будет занимать меньшее число символов, то перед ним добавятся пробелы
- Например:
- `Console.WriteLine("{0,10:f2}", x);`    `// _ _ _ _ _ 33.33`  
`// имеется в виду, что перед числом 5 пробелов`

# Дополнительные параметры формата

- `Console.WriteLine("{0,10:f2}", x);`    `// _ _ _ _ _ 33.33`  
`//` имеется в виду, что перед числом 5 пробелов
- Эта опция удобна, чтобы ровно выводить данные разной длины
- `int age = 33;`  
`Console.WriteLine("{0,-10} = {1}", "Возраст", age);`
- Минус означает, что пробелы будут справа, а не слева



# Дополнительные параметры формата

- `int age = 22;`  
`int weight = 66;`  
`double temperature = 36.66;`  
`string name = "Петр";`  
`Console.WriteLine("{0,-11} = {1}", "Возраст", age);`  
`Console.WriteLine("{0,-11} = {1}", "Вес", weight);`  
`Console.WriteLine("{0,-11} = {1}", "Температура", temperature);`  
`Console.WriteLine("{0,-11} = {1}", "Имя", name);`
- Возраст = 22
- Вес = 66
- Температура = 36,660000
- Имя = Петр

# string.Format

- Если хочется просто сформировать строку, но не печатать ее, то можно воспользоваться командой `string.Format`
- Она имеет те же аргументы, что и `Console.WriteLine`
- Пример:
- `int x = 44;`  
`string result = string.Format("Result = {0}.", x);`

# Задача

- В задаче про площадь прямоугольника вывести длину, ширину и площадь прямоугольника при помощи одного `Console.WriteLine`
- \* Вывести там же и периметр