

Лекция 6.

Функции

Функции

- **Функция** – это часть программы, к которой можно обращаться как к одной команде
- Например, мы можем написать программу для вычисления среднего арифметического диапазона чисел от начального до конечного числа
- Затем оформить ее в виде функции, и после этого можем обращаться к ней как всего к одной команде
- [https://ru.wikipedia.org/wiki/Функция_\(программирование\)](https://ru.wikipedia.org/wiki/Функция_(программирование))
- **!! Чтобы функции влезали на слайд, в презентации фигурные скобки будут не всегда на следующей строке. Правильно располагать их как в `if-else`, циклах и т.д.**

До преобразования в функцию

- ```
public static void Main()
{
 int start = 3;
 int end = 5;
 int sum = 0;
 int count = 0;
 for (int i = start; i <= end; i++)
 {
 sum += i;
 ++count;
 }
 double average = (double)sum / count;
 Console.WriteLine(average);
}
```

Если в программе надо считать средние арифметические для разных диапазонов чисел, то код придется продублировать

# После преобразования в функцию

- ```
public static double GetAverage(int start, int end) {  
    int sum = 0;  
    int count = 0;  
    for (int i = start; i <= end; i++) {  
        sum += i;  
        ++count;  
    }  
    return (double)sum / count;  
}
```

Объявление функции

```
public static void Main() {  
    double average = GetAverage(3, 5);  
    Console.WriteLine(average);  
    Console.WriteLine(GetAverage(10, 100));  
}
```

Вызов функции

После преобразования в функцию

- ```
public static double GetAverage(int start, int end) {
 int sum = 0;
 int count = 0;
 for (int i = start; i <= end; i++) {
 sum += i;
 ++count;
 }
 return (double)sum / count;
}
```

```
public static void Main() {
 double average = GetAverage(3, 5);
 Console.WriteLine(average);
 Console.WriteLine(GetAverage(10, 100));
}
```

Функцию можно  
вызывать много раз с  
разными параметрами  
(аргументами)

# Объявление функции

Модификатор  
видимости

Метод относится  
к классу в целом

Тип результата  
функции

Название  
функции

Тело  
функции

Список  
аргументов,  
через запятую

```
public static int GetSquare(int x) {
 int result = x * x;
 return result;
}
```

- Каждая функция в C# имеет:
  - название
  - возвращаемый тип (тип результата)
  - список аргументов
  - тело функции (код функции)

# Объявление функции

Модификатор  
видимости

Метод относится  
к классу в целом

Тип результата  
функции

Название  
функции

Тело  
функции

Список  
аргументов,  
через запятую

```
public static int GetSquare(int x) {
 int result = x * x;
 return result;
}
```

- Оператор **return** завершает функцию и выдает результат выражения в качестве результата вызова функции
- Пока что везде пишем **static**, смысл изучим позже
- Смысл слова **public** тоже рассмотрим позже, пока тоже пишите его

# Объявление функций

- Функции объявляются непосредственно внутри класса, на одном уровне
- Нельзя объявить функцию внутри другой функции или снаружи класса

- `public class Main`

- `{`

- `public static int F(int x)`

- `{`

- `return x * 5;`

- `}`

- `public static void Main(string[] args)`

- `{`

- `}`

- `}`

Порядок объявления функций не важен



# Как вызвать функцию

- Вызов функции из этого же класса:  
`int y = GetSquare(3); // 9`
- Если вызываем функцию другого класса:  
`int y = Main.GetSquare(3); // 9`
- Мы уже работали со вторым вариантом – это функции из класса `Math`:
- `int x = Math.Sqrt(2 * 3);`
- Чтобы вызвать такую функцию мы сначала пишем имя класса – `Math`, а потом через точку – имя функции `Sqrt`

# Вызов функции

- При вызове функции сначала вычисляются аргументы, затем исполнение переходит внутрь вызываемой функции
- `int x = Math.Sqrt(2 * 3);` // в функцию передается 6, а не  $2 * 3$
- После вычисления результата функции, исполнение возвращается к месту, откуда была вызвана функция, и туда передается результат вызова функции

# Оператор return

- ```
public static int F(int x)
{
    return x * 5;
}
```
- **return** – англ. возвращать
- После оператора **return** функция завершается, и программа вместе с результатом функции **возвращается** на место, откуда функция была вызвана
- Поэтому говорят «**функция возвращает значение**», то есть выдаёт значение
- «Функция f принимает целочисленный параметр x и возвращает целое число»

Аргументы функции

- Список аргументов идет в скобках после названия функции
- `public static int F(int x) { /* тело функции */ }`
- Аргументы задаются как переменные – указывается тип и имя аргумента
- Внутри функции аргумент ведет себя как локальная переменная
- Внутри функции нельзя объявлять локальные переменные с именами, которые совпадают с именем какого-либо аргумента этой функции

Аргументы функции

- Если аргументов несколько, то они идут через запятую
- `public static int F(int x, int y) { /* тело функции */ }`
- Функция может не иметь аргументов, тогда скобки оставляют пустыми
- `public static int F() { /* тело функции */ }`

Имена функций

- Имена функций должны:
 - Быть в верблюжьей нотации с заглавной буквы
 - Начинаться с глагола, т.к. функция – это действие (команда)
 - Отражать смысл функции
- Примеры имен:
 - GetAverage, GetAllCountries, SetCapacity, LoadConfiguration, SaveDocument, IsVisible и т.д.
 - Глаголы могут быть любыми, но чаще всего применяется около 10 разных глаголов

Задачи

- Написать функцию, которая принимает вещественные числа x и y , и вычисляет $3x + 4y$
- Вызвать ее из Main несколько раз с разными аргументами
- Написать функцию, вычисляющую среднее арифметическое целых чисел от `begin` до `end` включительно
- Вызвать ее из Main
- Написать функции для вычисления минимума и максимума из двух целых чисел
- Вызвать функции из Main

Warning в функции

- Тут warning:
- ```
public static int GetSquare(int x) {
 int result = x * x;
 return result;
}
```
- Смысл следующий – переменная result не нужна, можно просто сделать `return` нужного выражения:
- ```
public static int GetSquare(int x) {  
    return x * x;  
}
```
- Пожалуйста, всегда исправляйте этот warning, это очень часто встречается

Модификаторы видимости

- **Модификатор видимости** – это ключевое слово, которое задает уровень доступа к функции
- Есть модификаторы доступа:
 - `public` – функция видна всюду (в том числе из других классов)
 - `private` – функция видна только в этом же классе
 - Есть и другие, но их пока не будем рассматривать

Модификаторы видимости

- Если к `private` функции обратиться из другого класса, будет ошибка компиляции

- ```
public class A {
 private static int F() { /* тело функции */ }
}
```

```
public class Main {
 public static void Main() {
 int x = A.F(); // ошибка компиляции – f является private
 }
}
```

- Нам пока не важно какой модификатор использовать, можете использовать любой

# Static и не-static функции

- Все функции объявляются внутри классов
- Функции могут быть объявлены со словом `static` или без него
- Если функция объявлена без слова `static`, то чтобы вызвать ее, мы должны создать объект, и вызывать эту функцию через объект
- Пример не `static` функции:
- `string s = "123";`  
`s = s.PadLeft(5);` // функция `PadLeft` – не `static`  
// пришлось создать объект (строку) и вызвать функцию  
// от него

# Static функции

- Если функция объявлена как `static`, то чтобы вызвать ее, нам не нужно создавать объект, мы можем обратиться к ней по имени класса, в котором она объявлена
- ```
public class Math
{
    public static double Sqrt(double x)
    {
        // ...
    }
}
```
- ```
double x = Math.Sqrt(2);
```

# Локальные переменные

- Внутри функций можно объявлять переменные
- Переменные доступны только внутри этой функции. Поэтому они называются **локальными переменными**
- Пример:
- ```
public static double F(double x, double y)
{
    double z = x * x; // z, v – локальные переменные
    double v = y * y;
    return z + v;
}
```

Несколько веток исполнения

- В функции может быть несколько `return`'ов
- Это позволяет досрочно завершить функцию

- ```
public static int GetSign(int a)
{
 if (a == 0)
 {
 return 0;
 }
 return (a > 0) ? 1 : -1;
}
```

$$\text{sign } x = \begin{cases} 0, & x = 0 \\ 1, & x > 0 \\ -1, & x < 0 \end{cases}$$

- Каждая ветка исполнения должна иметь свой `return`!

# Несколько веток исполнения

- ```
public static int GetSign(int a) {  
    if (a == 0) {  
        return 0;  
    }  
    if (a > 0) {  
        return 1;  
    }  
    if (a < 0) {  
        return -1;  
    }  
    // ошибка компиляции – нет return'а  
}
```
- Компилятор не понимает связь между условиями
- Для него есть 3 `if`'а, которые все могут не выполняться, а для этого случая сейчас нет `return`'а

Несколько веток исполнения

- ```
public static int GetSign(int a) {
 if (a == 0) {
 return 0;
 }
 if (a > 0) {
 return 1;
 } else {
 return -1;
 }
}
```
- Так будет работать, т.к. в **if-else** всегда выполняется ровно одна из веток. И в них обеих есть **return**
- Заодно стало на одну проверку меньше

# Возвращаемый тип void

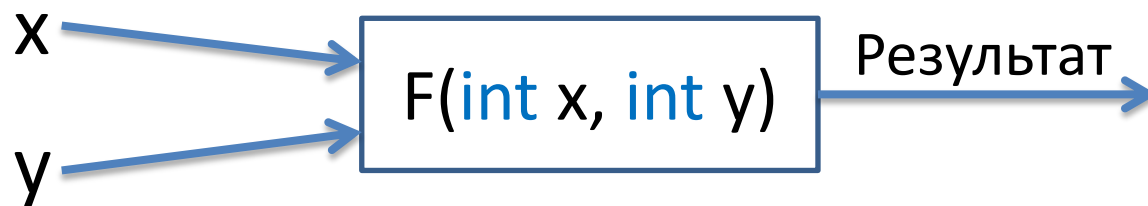
- Ключевое слово `void` указывается для возвращаемого типа функции, если функция не должна выдавать результат
- Пример:
- ```
public static void PrintText(string s) {  
    Console.WriteLine(s);  
}
```
- В таких функциях можно не писать `return`
- Такие функции в некоторых языках называют **процедурами**
- Main также является `void` функцией, и мы там `return` не писали

Досрочное завершение функции

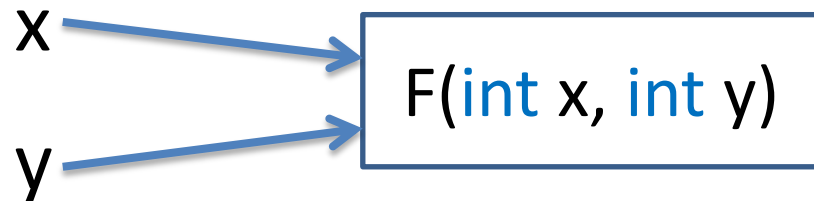
- `return` позволяет досрочно завершить функцию с возвращаемым типом `void`
- ```
public static void PrintIfNotZero(int a)
{
 if (a == 0)
 {
 return;
 }
 Console.WriteLine(a);
}
```
- Это очень удобно использовать в Main

# Количество результатов функции

- Если функция не `void`, то она всегда выдает ровно 1 результат



- Если функция `void`, то она не выдает результат



- Функция не может выдавать несколько результатов одновременно

# Вызов функции

- Вызов функции (если она возвращает не `void`), является выражением
- Т.е. результат вызова функции можно передавать в другие функции, присваивать переменным и т.д.
- `Console.WriteLine(F(2.0, 4.0));`  
`double x = F(4, 6);`  
`// F – некоторая наша функция`

# Передача аргументов

- ```
public static void F(int x)
{
    x = 5;
}
```

Аргумент x никак не связан с переменной x в другой функции

- Код в функции Main:

В функции передаются копии аргументов

- ```
int x = 4;
F(x);
Console.WriteLine(x);
```

 // 4, а не 5!

Вообще, изменять значение аргументов функции (как в функции f) является плохим стилем

# Хороший стиль для функций

- Функция должна иметь имя, которое отражает то, что функция делает
- Желательно, чтобы функция выполняла одно действие
- Тело функции не должно быть слишком длинным. В этом случае функцию следует разбить на несколько маленьких функций

# Пример – вычисление суммы чисел

- Пример – функция для вычисления суммы чисел от begin до end целых чисел
- ```
public static int Sum(int begin, int end)
{
    int sum = 0;
    for (int i = begin; i <= end; ++i)
    {
        sum += i;
    }
    return sum;
}
```


Задача на дом «Print and read»

- Написать функцию, которая объединяет в себе две операции: вывод пользователю приглашения для ввода в консоль и чтение `int`'а с консоли
- Функция должна принимать строку и возвращать прочитанное число
- Из функции Main несколько раз вызвать данную функцию с разными значениями аргументов
- Пример: `int a = PrintAndRead("Введите число:")`

Задача на дом «Перевод температур»

- Написать программу, которая переводит температуру из градусов Цельсия в градусы Кельвина и Фаренгейта (Фаренгейта – на дом)
- Например, прочитав число – температуру в шкале Цельсия и напечатать две строки – в градусах Кельвина и Фаренгейта
- Перевод градусов Цельсия в градусы Кельвина и перевод в градусы Фаренгейта оформить отдельными функциями
- Формулы найти в интернете

Перегрузка функций (overload)

- В одном классе можно иметь несколько функций с одинаковым названием, если эти функции отличаются набором аргументов
- Это называется **перегрузкой функций (overload)**
- `public static void F(int a)`
 {
 // код
 }
- `public static void F(double a)`
 {
 // код
 }

Тут функции отличаются
типом аргумента

Перегрузка функций (overload)

- `public static void F(int a)`
`{`
 `// код`

Тут функции отличаются
количеством аргументов

- `public static void F(int a, int b)`
`{`
 `// код`
`}`

Перегрузка функций (overload)

- `public static void F(int a, double b)`

```
{  
    // код  
}
```

Тут функции отличаются
порядком типов аргументов

- `public static void F(double a, int b)`

```
{  
    // код  
}
```

Надо заметить, что при
перегрузке не важно имя
аргументов – важны их
типы и порядок

Перегрузка функций (overload)

- Но нельзя перегружать функции, которые отличаются только типом возвращаемого значения
- ```
public static int F(int a)
{
 // код
}
```
- ```
public static void F(int a)
{
    // код
}
```
- **// будет ошибка компиляции**

Задача «Перегрузка»

- Объявить в классе несколько функций с именем `GetTypeSize`
- Каждая функция должна принимать один аргумент одного из следующих типов: `byte`, `short`, `int`, `long`, `float`, `double`
- Функция должна возвращать `int` – количество байт, требуемых под переменную данного типа
- В функции `Main` сделать по одному примеру использования для каждой перегруженной функции

Задача «Стоимость заказа»

- Написать функцию для расчета стоимости заказа с учетом скидок
- Всего есть два вида товаров, в заказ может входить некоторое количество товаров одного типа и другого типа
- Скидка 5% начисляется, если суммарное количество товаров в заказе не меньше 10
- Скидка 5% начисляется, если суммарная стоимость заказа не меньше 1000 рублей
- Если выполняются оба условия, то скидка 10% от начальной стоимости