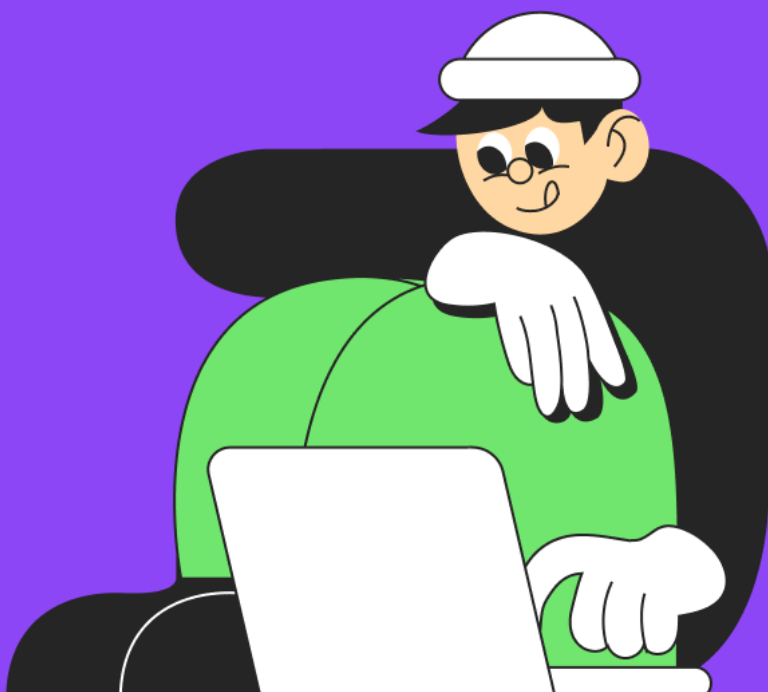
GeekBrains

Функции: продолжение



Оглавление

[00:01:36]

Вступление

Приветствую вас на лекции по введению в программирование. Мы продолжаем разбираться с методами. На предыдущем занятии мы узнали, для чего они нужны. Узнали, что они, как правило, решают подзадачи. Сегодня мы углубимся в тему и выделим 4 основных группы методов, а также будем использовать их на практике. И познакомимся с новыми языковыми конструкциями, в частности, циклом. Приступим.

[00:02:06]

Функции в программирование.

Мы знаем о том, что любой метод, как правило, имеет возвращаемый тип. Также у любого метода есть идентификатор или имя, набор аргументов и тело метода. Сегодня настало время разобраться, что значит, метод что-то возвращает или возвращает void. Что же это за void такой?

- Первая группа методов — не принимает никаких аргументов и ничего не возвращают.
- Вторая группа методов — принимает какие-то аргументы, но ничего не возвращают.

Эти две группы — так называемые void методы, которые ничего не возвращают.

- Третья группа методов — может что-то возвращать, но не принимает никаких аргументов. Например, может служить для генерации случайных данных.
- Четвёртая группа методов — что-то принимает (аргументы, данные). И что-то возвращает для дальнейшей работы.

Есть ли ещё какие-то отдельные виды методов? Да есть, но они частично сводятся к этим четырём.

[00:03:17]

Разберём группы методов

Первая группа методов

Методы первой группы очень простые и выглядят следующим образом. Допустим:

```
void Metod1()

{

Console.WriteLine("Автор ...");

}
```

Обратите внимание, у нас ключевое слово **void**. В скобках нет никаких аргументов. И есть тело метода, которое что-то может, например, показывать на экран. То есть в конце каждой созданной программы вы, быть может, хотите указывать своё авторство.

Следующий пункт, который может быть интересен или нужен, это вариант, как вызываются подобного рода методы. А вызываются они очень просто, вы должны будете указать:

Metod1() — где **Metod1**, является идентификатором метода.

Пожалуйста, будьте внимательны, потому что в некоторых случаях начинающие вызывают методы без использования скобок. В этом случае при выполнении программы будет ошибка. Проверим это в терминале с помощью команды dotnet run и убедимся в том, что действительно будет ошибка. При этом если мы напишем всё правильно, то на экране мы увидим:

Автор ...

Это первая группа методов. Для дальнейшей демонстрации я буду выключать его вызов.

//Metod1() — добавление **//** превратит команду в комментарий, мало влияющий на код.

[00:04:42]

Вторая группа методов

Следующая группа методов это пусть так и называется **Metod2**.

void Metod2(string msg) — где **void** ключевое слово, дальше идентификатор, в скобках указаны какие-то аргументы.

{

Console.WriteLine(msg); — оператор, в скобках указан принятый аргумент.

}

Metod2("Текст сообщения"); — где **Metod2** является идентификатором, а в скобках указан текст, выводимый в консоли.

Это методы, которые ничего не возвращают, но в то же время могут принимать какие-то аргументы. Я ещё сразу же запущу и продемонстрирую тот факт, что вызов **Metod2** будет отображать в консоли именно тот текст, который мы указали.

[00:05:43]

Именованные аргументы

Отмечу, что ещё есть так называемые именованные аргументы, когда у нас явно может быть указано какому аргументу, какое значение мы хотим указать. Это часто бывает нужно, если методы принимают какое-то количество аргументов, отличное от 1. Продемонстрирую это.

```
void Metod21(string msg, int count)
{
        int i = 0;
        while (i < count)
        {
```

Console.WriteLine(msg); - где переменная **count** отображает на экране определённое количество сообщений **msg.**

count++; - а надо **i++;**

}

}

Metod21(“Текст”, 4); - метод вызывает Текст, после запятой указано количество вызовов, в нашем случае 4.

Здесь будет использоваться значение переменной **count**, чтобы показывать на экране определённое количество сообщений, которые будут передаваться непосредственно в наш метод.

Увеличение счётчика на 1 называют инкрементом, а уменьшение на 1 очень часто называют декрементом. Так что можете несколько слов программистских записать в словарик.

За комментируем вызов **//Method2**. И отправим на выполнение **Method21**. Ожидаем увидеть слово Текст 4 раза, но видим больше. Почему? Потому что я увеличиваю **count**, а нужно увеличивать **i**. Напоминаю, что если вдруг случилось зацикливание программы, то это можно исправить с помощью Ctrl+c. Причём неважно будет у вас Mac или Windows. Перезапустим и убедимся в том, что всё будет хорошо. И теперь наблюдаем 4 раза слово Текст.

Теперь идея в том, что мы можем в том числе явно указывать к какому аргументу, какое значение мы хотим присвоить, через такую конструкцию.

Было: **Metod21(“Текст”, 4);**

Стало: **Metod21(msg: “Текст”, count: 4);**

Явно указывая наименование аргумента, не обязательно писать их по порядку.

Metod21(count: 4, msg: “Текст”);

Это тоже особенность, то есть можно, например, написать **count: 4**, дальше написать msg: Новый текст. Теперь если запускать в таком формате, то мы увидим Новый текст на экране точно же 4 раза. Ура, всё работает.

[00:08:48]

Третья группа методов

Эти методы, которые что-то возвращают, но ничего не принимают. Если метод что-то возвращает, мы в обязательном порядке должны указать тип данных, значение которого ожидаем. Для нас это будет **Metod3**.

int Metod3() - не принимает никакие аргументы

{

return DataTime.Now.Year; - обязательное использование оператора **return**,

}

int year = Metod3(); - вызываем метод, в левой части используем идентификатор переменной (**year**) и через оператор присваивания (**=**) кладём нужное значение

Console.WriteLine(year);

В дальнейшем используем переменную **year**, и то значение, которое нам вернул метод. То есть **return DataTime.Now.Year** провёл какую-то работу и в переменную **year** будет положен результат работы метода, дальше мы можем его использовать. Перезапустим консоль. В результате ожидаем в консоли увидеть 2021. Действительно, всё хорошо.

[00:10:40]

Четвёртая группа методов

Самая важная группа методов, это методы, которые что-то принимают и что-то возвращают.

```
string Metod4(int count, string text)
{
        int i = 0;
        string result = String.Empty;

        while (i<count)
        {
                result = result + text;
                i++;
        }
        return result;
}
string res = Metod4(10, “asdf”);
Console.WriteLine(res);
```

Возвращать будем строку **string,** по традиции называем метод **Metod4.** Передавать будем **int count** и условный тип **char**, новый тип данных для вас. Соответственно, что мы здесь делаем? Мы будем **string** компоновать друг за другом **count** раз.

Сделаем это. Для начала возьмем цикл, дальше нам потребуется переменная куда мы будем класть результат, конечный **string result**. Изначально можно и нужно в неё положит какое-то значение. Этим значением является пустая строка, чтобы не только вы понимали написанный код, более правильно, будет написать **string.Empty**. То есть таким образом мы можем просто прочитать, **result** у нас изначально будет пустой строкой. После этого используем конструкцию **while**. В которой пока **i < count**. Обязательно не забываем увеличивать счётчик и класть в **result = result + text** (это строчка, которую мы указали). Здесь, кстати в общем случае можно, например, правой кнопкой сделать **rename** и написать условный текст, тогда будет не только символ, но и текст. Таким образом, у нас везде текст меняется, и соответственно, наименование аргумента будет изменяно.

После того, как данный метод отработает, используем классический и известный оператор **return**, в котором указываем результат или переменную значение, которой ожидаем получить из метода. Чтобы вызвать этот метод мы должны будем, создать нужную нам переменную, дальше по порядочку указать, например, значение **10** и текст, который мы будем склеивать 10 раз, пусть это будет условный **asdf** текст. После этого можем показать на экране результат, который этот метод будет возвращать. Очистим консоль и запустим метод. Для проверки заменим условный текст на букву **z.** Ожидаем увидеть эту последовательность 10 раз.

```
string res = Metod4(10, “z”);

Console.WriteLine(res);
```

И действительно, она есть. Мы получаем результирующую строку, состоящую из 10 букв Z.

[00:14:03]

Условные виды методов

Это условные 4 вида методов, вы ни в каких книгах, скорее всего, не найдете какую-то явную градацию видов методов. Она нужна, просто для того, чтобы было примерно понятно каким

образом их различать. Мы условно выделяем 2 группы или 2 вида, которые ничего не возвращают. Они всегда начинают с ключевого слова **void**, то есть описание метода или тип, который он возвращает или возвращает пустоту, тут можно по-разному говорить. И 3, и 4, который я условно пронумеровал для того, чтобы в дальнейшем можно было обращаться такой градации.

В любом случае мы будем сейчас писать другие методы, тренироваться и увидим то, что как раз таки именно 4 группа наиболее часто используется. Даже в том случае, если вы будете использовать какой-то встроенный функционал. В частности, например, генерация псевдослучайных чисел, это и есть тот метод 4 позиции, только у нас будет приниматься 2 аргумента только числа и возвращаются тоже число.

[00:14:56]

Цикл for

Итак, следующий пункт, который важен для нас — это новый цикл, точнее, ещё один цикл, наиболее часто использующийся в разработке. Он не так часто используется при описании блок-схем, хотя он был у вас в модуле введения в программирование. Сейчас мы будем его тренировать. Итак, это цикл со счётчиком или ещё называется цикл **for**. Синтаксически он просто в себе собирает все в кучу. Не нужно будет отдельно инициализировать счетчик, где-то в теле цикла что-то нужно будет увеличивать, где-то проверять условия. Цикл **for** как раз собирает всё в одном месте.

Поправим наш 4 метод, который был завязан на цикле **while**. Я его просто за комментирую и продублирую для того, чтобы у вас остались всевозможные вариации. Итак, синтаксис цикла **for** примерно следующий.

```
string Metod4(int count, string text)
{
        string result = String.Empty;

        for (int i = 0; i<count; i++) - вначале ключевое слово, затем инициализация счётчика, после проверка условия и инкремент (увеличение счётчика).
        {
                result = result + text;
        }
        return result;
}
string res = Metod4(10, “asdf”);
Console.WriteLine(res);
```

Теперь смотрите, насколько наша программа стала красивее. Ключевое слово **while** мы убрали, как и счётчик. В итоге собрали всё в одном месте, и теперь не нужно бегать от одной части программы к другой, чтобы каким-то образом что-то не забыть или где-то подправить и т. д. Теперь запустим и убедимся в том, что результат работы никаким образом не поменялся.

Отмечу факт, что в любом случае все циклы взаимозаменяемы. В языке C# есть цикл **do while** (цикл с постусловием), если захотите, думаю, вы сможете его самостоятельно изучить, а нам он, скорее всего, не понадобится. Наверное, 99% задач, которую будете решать, будут легко выполняться при помощи цикла for, но иногда можно использовать файл.

[00:17:11]

Цикл в цикле

Мы узнали новую синтаксическую конструкцию. Теперь пора узнать, что можно использовать цикл внутри цикла. Где это бывает нужно? Самый простой пример, вам захотелось закрасить какую-то прямоугольную область. В этой области есть строки и столбцы. Соответственно, мы будем пробегать по всем строкам и столбцам, и что-то делать. Может быть, одним цветом закрашивать, может быть разным это зависит от задачи. Но пока посмотрим, каким образом можно использовать эту возможность.

Классической демонстрацией использования циклов в цикле я предлагаю рассмотреть задачу вывода таблицы умножения на экран. Итак, идея следующая.

```
for (int i = 2; i <= 10; i++)
{
        for (int j = 2; j <= 10; j++)
        {
                Console.WriteLine($"{i} * {j} = {i * j}");
        }
        Console.WriteLine();
}
```

У нас есть цикл **for**, он очень легко строится. Дальше, мы указываем начальное значение. Таблица умножения начинается с 2. Затем говорим, что пока счётчик **i** меньше или равен 10, надо его увеличивать. После возьмём второй цикл, обратите внимание, что в первом (внешнем) цикле использовался счётчик **i**, значит, внутренний будет **j**, который мы также будем менять. Например, от 2 до меньше или равен 10. А телом второго цикла мы укажем непосредственное произведение. Сделать это можно различными способами. Я использую интерполяцию строк.

```
Console.WriteLine($"{i} * {j} = {i * j}")
```

Посмотрим, что у нас из этого получится. Ожидаем увидеть, что-то похожее на таблицу умножения. Действительно. 2*2=4, 2*3=6 и так далее. Вроде бы всё ок.

Но есть проблема, связанная с тем, что вся таблица идёт без разделений на части. Чтобы это исправить после того, как отработает второй цикл, в нашем случае:

```
for (int j = 2; j <= 10; j++)
{
         Console.WriteLine($"{i} * {j} = {i * j}");
}
```

Делаем переход на новую строку.

```
Console.WriteLine();
```

То есть, выполняется вывод умножения на отдельное число, потом искусственный разрыв строк и вывод умножения на новое число. Так получается более красиво. Отдельно умножение на 2, 3, 4 и так далее.

Привыкайте к использованию цикла **for**, теперь я максимально часто буду использовать именно его.

[00:20:34]

Тренировочная задача

Следующим пунктом предлагаю потренировать использование методов и цикла for. Здесь самым классическим примером будут являться программы обработки текста.

Итак, задача. Дан текст. В нём нужно все пробелы заменить чёрточками, маленькие буквы «к» заменить большими «К», а большие «С» заменить на маленькие «с».

Первым делом, которым вы должны себе отметить. Это ясна ли вам задача? В частности:

- Что значит «Дан текст»? Как вы уже помните, из предыдущих лекций, дан – это непонятно. Ввёл пользователь? Считали из файла? Взяли с базы данных? Из какого-то сервиса может быть, скачали и так далее. Поэтому чётко определяем для себя, что значит, дан. В нашем случае мы будем считать, что он просто будет храниться как отдельная строка.
- Что значит «чёрточками»? Например, в русском языке мы можем сразу же выделить тире, дефис и ещё можно добавить минус.
- Какой алфавит у нас? Это может быть кириллица, в этом случае буква «к» одна. А может быть латиница, тогда буква «к» уже какой-то другой символом. При этом буквы «С» и «с» мало того что на одной кнопке находится, так ещё и выглядят одинаково. И хотя для нас они выглядят вроде бы одинаково, для компьютера это абсолютно разные символы.

Перейдём к написанию кода. Для начала рекомендую вам всегда в начале задачи, которую вы решаете, добавлять её условия в качестве комментария, чтобы можно было понять, что делать.

```
//=====Работа с текстом
// Дан текст. В тексте нужно все пробелы заменить чёрточками,
// маленькие буквы “к” заменить большими “К”,
// а большие “С” маленькими “с”.
// Ясна ли задача?
```

Дальше. У нас есть сам текст. Можете в чат написать или в комментариях написать, о том узнали ли вы произведения, откуда взят этот текст? И дальше мы уже начинаем писать код.

```
string text = “— Я думаю, — сказал князь, улыбаясь, — что,”
    ”ежели бы вас послали вместо нашего милого Винценгероде,”
    ”вы бы взяли приступом согласие прусского короля.”
    ”Вы так красноречивы. Вы дадите мне чаю?”;
```

В нашем случае требуется небольшое пояснение, если будет какая-то строка, то для этой строки есть некоторое количество вспомогательного функционала.

```
// string s = “qwerty”
//            012345
```

В частности, если требуется обратиться к конкретному символу строки, мы можем это делать, начиная отсчитывать позицию символов с 0. То есть «**q**» — это 0, «**w**» — 1, «**e**» — 2 и так далее. Чтобы получить конкретный символ, мы можем через квадратные скобки обратиться, указать идентификатор строки. Например:

```
// s[3] // r
```

В этом случае будьте, пожалуйста, внимательны нулевой символ «**w**» это 1 символ, дальше 2 символ это будет у нас 3. То есть, буква **r** будет именно s[3]. Теперь приступим к написанию кода.

Итак, метод у нас будет принимать сроку и символы, которые нужно будет менять. Соответственно, старый символ и на который нужно будет заменить. Возвращаться точно так же будет строка, поэтому сразу можно сделать вывод о том, что это условно четвёртый вид методов. Назовём его **Replace**. Далее **string** и какой-то входной текст. Затем указываем конкретный символ **oldValue** и конкретный символ, на который мы будем менять, **newValue**.

```
string Replase(string text, char oldValue, char newValue)
{
        string result = String.Empty;
        int length = text Length;
        for (int i = 0; i < length; i++)
        {
                if(text[i] ==oldValue) result + $"{NewValue}";
                else result + $"{str[i]}";
        }
        return result
}
string newText = replace(text, ' ', ' | ');
 Console.WriteLine(newText);
```

Заводим новую строку **result**, чтобы не запутаться. Напоминаю, что инициализация пустой строки выглядит как **string result = String.Empty**. Чтобы ничего не забыть, мы можем сразу же вернуть этот результат. Здесь уже можем писать код, чтобы получить длину строки. Сделать это можно при помощи обращения к соответствующему свойству, показывающему количество символов в строке. В этом случае **textLength**, на примере **string s = "qwerty"**, выдаст 6. Далее, воспользовавшись циклом **for**, мы пройдём от нулевого символа до конца нашей строки. Выполняем следующее действие, если текущий символ, для нас это текст **i** совпал с символом, который мы хотим заменить, то в результат мы должны положить новое значение в виде строки **newValue**. Если совпадений не обнаружено, то в **result**, нужно добавить текущий символ, который и был. Можно оставить код в таком виде.

Запустим и, посмотрим, что у нас получится. **String newText = replace(text, ' ', ' | ')** и здесь мы указываем, что на что меняем. В нашем случае просили пробелы заменить на чёрточки, для этого я использую минус. Хотя, с другой стороны, чтобы было видно это самоизменения, возьмём вертикальную строчку и следующим этапом **Console.WriteLine** мы хотим увидеть **newText**. Очистим и запустим терминал, посмотрим на результат. Любой код запускается со второго раза. Я допустил ошибку, написал **result** плюс что-то, но непонятно куда я это должен сохранять. Поэтому правильный код:

```
string Replase(string text, char oldValue, char newValue)
{
        string result = String.Empty;
        int length = text Length;
        for (int i = 0; i < length; i++)
        {
                if(text[i] == oldValue) result = result + $"{NewValue}";
                else result + result $"{text[i]}";
        }
        return result
}
```

string newText = Replace(text, ‘ ’, ‘ | ‘);
Console.WriteLine(newText);

И сразу же обращайте внимание, если вас имя аргумента, текст, то и, конечно же, обращаться нужно к тексту. Поэтому перезапустим и будем надеяться, что со 2 раза оно заработает. Ура, со 2 раза заработало, и мы наблюдаем действительно изменения текста. Все пробелы, они заменились соответственно таким вертикальными чёрточками.

Любопытно, что, по факту, нас просили заменить одни символы другими, но это нужно сделать для разных символов. Учитывая то, что мы написали в методе 1, в котором сказали есть старое значение, нужно будет заменить его на новое значение. В этом случае мы можем переиспользовать этот метод, просто указав в качестве аргументов те символы, которые нужно заменить на то, что нужно заменить. Так и сделаем.

В нашем случаем полученный текст, мы можем в дальнейшем начать обрабатывать здесь. Для красоты я буду разделять каждый вывод пустой строкой. Мы получаем **newText** и дальше требуется в качестве аргумента **Replace** передать наш старый текст, и опять получить новый, только теперь заменить маленькие буковки «**к**» большими. Показать результат. И так далее.

Думаю, вы сможете самостоятельно, в качестве тренировки, взять и проверить, как работает этот метод, если мы вместо больших «**С**» попробуем поставить маленькие «**с**».

string newText = Replace(text, ‘ ’, ‘ | ‘);
Console.WriteLine(newText);
Console.WriteLine();
newText = Replace(text, ‘ к ’, ‘ К ‘);
Console.WriteLine(newText);

Запустим и посмотрим, что у нас получится. Итак, сначала у нас был какой-то текст, дальше мы его показываем уже с заменёнными пробелами и затем хотим поменять маленькие буквы «**к**» на большие. И, собственно, мы это видим.

[00:28:33]

Упорядочить массивы

Итак, вы попробовали создавать методы для работы с текстом. Теперь снова поработаем с массивами. В качестве задачи для работы с массивами, я выбрал упорядочивание данных внутри массива. Существуют разные алгоритмы, которые вы, скорее всего, будете писать в дальнейшем. Остановимся на одном из самых простых. Это так называемый алгоритм сортировки методом выбора, ещё его называют алгоритм сортировки методом минимакса или иногда просто называют методом максимального, или выбора максимального или выбора минимального и так далее. Узнаем, в чём особенность (суть) этого алгоритма. Есть какая-то последовательность чисел. Наша задача — выбрать самый первый элемент и в оставшейся части, с учётом нашей текущей позиции определить минимальный. После того как он найден, нужно поменять выделенный (рабочий элемент), на который сейчас указывает стрелочка, это 6, с единицей, являющейся для нас минимальной, в общем, выделенном куске.

Было: 6 8 3 2 1 4 5 7
Стало: 1 8 3 2 6 4 5 7

Поменяли. Дальше у нас следующий шаг, который будет выбирать очередной рабочий элемент для нас. Это теперь 8. Мы явно указываем то, что 1 уже отсортированы к ней больше касаться

не нужно. Дальше наша задача во всём неотсортированном кусочке выбрать снова минимальный. Поменять его местами с рабочим.

Было: 1 8 3 2 6 4 5 7
Стало: 1 2 3 8 6 4 5 7

В таком случае получается подмассив или кусок массива из первых двух элементов отсортирован. Дальше переходим к следующему элементу, снова находим минимальный, и здесь получается так, что нам ничего ни с чем менять не нужно, то есть, он остаётся на своей позиции. На следующем этапе переходим к следующему рабочему элементу. Снова находим минимальный, снова меняем их местами, таким образом, мы продолжаем до того момента, пока весь массив не будет отсортирован.

Результат: 1 2 3 4 5 6 7 8

На самом деле мы сами указали достаточно много действий, но в то же время они могут быть выражены тремя пунктами.

1. Найти позицию минимального элемента в неотсортированной части массива.
2. Произвести обмен этого значения со значением первой неотсортированной позиции.
3. Повторять пока есть неотсортированные элементы.

[00:30:42]

Напишем код

```
int[] arr = {1, 5, 4, 3, 2, 6, 7, 1, 1};

void PtintArray(int[] array)

{

        int count = array.Length

        for (int i = 0; i < count; i++)

        {

                Console.Write($"{array[i]}");

        }

        Console.WriteLine();

}

PrintArray(arr)
```

Для начала заведём массив, который будем сортировать, **int[] arr = {1, 5, 4, 3, 2, 6, 7, 1, 1}**. Для этого алгоритма абсолютно не принципиально, если внутри массива будут повторяющиеся элементы. На следующем этапе решим небольшую подзадачу, отвечающую за вывод данных массива на экран. Напишем отдельный метод. Причём сделаем это несколькими способами. Первый это метод **void**, назовём его **PrintArray**. В качестве аргумента будет приходить массив. Дальше. Получение, количество элементов (буду давать разные наименования, чтобы вы тоже потихонечку к этому привыкали) **Length**. Здесь цикл **for,** в котором мы пробегаем по всем элементам нашего массива. Показываем на экране. Можно это сделать так, чтобы вывод

осуществлялся в одну строку. Напишем действие **array[i]** и через пробел будем указывать. После того как вывод окончен, выведем на экран пустую строку **Console.WriteLine**. Сразу будем тестировать. Итак, **PrintArray(arr)**, где **arr** это наименование нашего метода. Очистим консоль, запустим и посмотрим на результат. Вроде всё хорошо. Удивительно, но код запустился с 1 раза. Это прекрасно.

[00:32:53]

Следующий этап. Нам нужно написать метод, который будет упорядочивать наш массив. Назовём его **selectionSort**. В качестве аргумента точно таким же образом, как и в методе выше, будет приходить некий массив array. Теперь писать, значит, нам нужно в первую очередь пробежаться по всем элементам нашего массива. Чтобы не нарушать общность, сделаем определение обращение к длине нашего массива через array.Length. Но здесь есть отдельная особенность, о которой мы чуть-чуть позже поговорим.

```
int[] arr = {1, 5, 4, 3, 2, 6, 7, 1, 1};
void PtintArray(int[] array)
{
        int count = array.Length
        for (int i = 0; i < count; i++)
        {
                Console.Write($"{array[i]}");
        }
        Console.WriteLine();
}
 void selectionSort(int[] array)
{
        for (int i = 0; i < array.Length - 1; i++)
        {
                int minPosition = i;
                for (int j = i+1; j < array.Length; j++)
                {
                        if(array[j] < array[minPosition]) minPosition = j;
                        {
                                minPosition = j
                        }
                }
                int temporary = array[i]
                array[i] = array[minPosition];
                array[minPosition] = temporary;
        }
}
PrintArray(arr);
SelectionSort(arr);
PrintArray(arr);
```

На следующем этапе мы определяем позицию, на которую смотрим. Назовём её **minPosition.** И запоминаем позицию рабочего элемента, для которого мы в дальнейшем будем производить какие-то действия. Здесь самое любопытное то, что после того, как мы выполним, какой-то блок кода, пока что его оставлю пустым, нам потребуется поменять значение нашей минимальной позиции, с найденной нами позицией. Поэтому так и сделаем. Значит, **int temporary = array**. Наша рабочая оппозиция, напоминаю, что она будет вычисляться индексом **i**. Соответственно, в **i** позицию, мы должны будем положить элемент, который будет найден в процессе работы, этого пока не написанного кусочка кода. То есть, это простой обмен двух

переменных местами. Вы должны были это делать на семинарах. И соответственно, в эту минимальную позицию, мы кладём элемент, который был временным.

[00:35:09]

Дальше наша задача — заполнить блок кода, всё, что мы здесь делаем, это ищем самый минимальный элемент. Как это будем делать? Мы воспользуемся циклом внутри цикла, как это было в самом начале лекции **for j**. Начальная позиция, от которой мы будем начинать эти действия, как вы помните, это кусочек массива, который был отсортирован, мы уже его не касаемся, а то, что нас отсортировано, начинается как раз с позиции **i+1**. И идём мы до последнего элемента **array.Length**.

[00:35:47]

В этом блоке кода всё, что мы делаем — это ищем минимальный элемент. Чтобы это сделать, мы смотрим текущий, если он меньше того элемента, который мы предполагали на минимальной позиции, то нужно сохранить текущую позицию. Этот блок кода ищет максимальный элемент и здесь производится swap.

[00:36:30]

Дальше самый важный пункт заключается в том, что если мы начинаем позицию поиска максимального от индекса **i+1**, где **i** меняется до максимального значения позиции нашего массива, то в этом случае мы должны будем искусственно одну единицу отнять. Тогда получится, что ровно это **i+1** даст общее количество элементов.

Итак, попробуем продублировать этот блок кода. И вызвать упорядочивание массива **SelectionSort(arr)**. Хочется верить, что кот заработает с первого раза.

[00:37:15]

Заключение

Мы научились, работать с новым циклом. Узнали новую особенность, использование цикла внутри цикла, посмотрели, где это используется. В некоторых случаях цикл в цикле и в цикле может быть, то есть не обязательно только два раза, может быть и тройная и т. д.

Подведём небольшие итоги и в качестве домашней или самостоятельной работы я предлагаю вам подумать о том, каким образом адаптировать код упорядочивания массива от минимального элемента к максимальному, таким образом, чтобы упорядочение производилось в обратном порядке. Сначала будет самый большой элемент. Потом меньше, меньше, меньше, в итоге будет самый маленький.

Подводя итог, мы можем сказать, что циклов много не бывает. Обязательно помним, что циклы можно использовать внутри циклов. Мы выделили 4 условных вида методов, когда ничего не принимают 2 группы, и они обязательно характеризуются ключевым словом void в самом своём описании. И 2 группы методов, которые какой-то результат могут возвращать. Помним, что у методов, скорее всего, будут входные аргументы. Также первые 2 ничего не возвращают. Они характеризуются ключевым словом void. Оставшиеся 2 группы обязательно должны возвращать результаты и характеризуются типом данных, значение которого будет возвращаться.

Мы узнали, что есть имена аргументов. Есть именованные аргументы и каким образом к ним можно обращаться. В обязательном порядке методов. Забегая вперёд, скажу, что признаком

хорошего тона является описание методов, в котором не больше 5 аргументов. Самое большое, что есть в библиотеках языка C# это 16. Но это очень специфичная задача, и я настоятельно рекомендую не прибегать к такого рода описанию методов. Хотя иногда бывают задачи, в которых очень нужно и важно указать такое количество аргументов. От себя отмечу, что, скорее всего, будет лучше перепроектировать свой метод, сделать что-то иное.

На этом я с вами прощаюсь. Надеюсь, что вы для себя что-то новое узнали, обязательно потренируйтесь, скачайте исходники, посмотрите, пишите в общий чат, задавайте вопросики. Я всегда готов помочь. Пока.