

Введение

**Знакомство с языками программирования**

|  |
| --- |
| 💡 Шрифт – IBM Plex Sans  Размер текста – 12, интервал 1,15, выравнивание по ширине  Размер заголовков - 22 |

**Оглавление**

[Введение 3](#_Toc130076133)

[Термины, используемые в лекции 3](#_Toc130076134)

[Знакомство 4](#_Toc130076135)

[План курса 4](#_Toc130076136)

[План урока 5](#_Toc130076137)

[1. Почему именно С#? 5](#_Toc130076138)

[2. Подготовка окружения 7](#_Toc130076139)

[3. Вспомним «Введение в программирование» 10](#_Toc130076140)

[4. Переменные и арифметика 11](#_Toc130076141)

[4.1. Пример 1 11](#_Toc130076142)

[4.2. Пример 2 13](#_Toc130076143)

[5. Задача с двумя гирями 18](#_Toc130076144)

[6. Задача с пятью гирями 21](#_Toc130076145)

[6. Циклы 24](#_Toc130076146)

[7. Итоги 26](#_Toc130076147)

# Введение

Этот **курс**

* **познакомит** вас с синтаксисом языка программирования высокого уровня
* **даст** вам **навыки** решения базовых алгоритмических задач
* **даст** **представление** о подходах к написанию качественного кода

# Термины, используемые в лекции

**Операционная система** (ОС) – программное обеспечение, управляющее ресурсами компьютера и организующее взаимодействие пользователя с компьютером.

**Visual Studio Code** (VS Code) – текстовый редактор от компании Microsoft, включающий в себя отладчик, инструменты для работы с GIT, подсветку синтаксиса. Имеет широкий набор модулей для работы с различными языка программирования.

**Переменная** – область памяти, предназначения для хранения некоторого значения.

**Инициализация переменной** – присвоение переменой начального значения.

**Тип данных** – характеристика значения, указывающая к какому роду данных оно относится (например, числовые, символьные, строковые и т.д.)

**Типизация** – способом определения типа данных языком программирования.

**Условная конструкция** – средство языка, позволяющее создать условный алгоритм (ветвление).

**Условие** – выражение логическое типа, используемое в условных конструкциях и циклах.

**Цикл** – тип алгоритма, позволяющий повторять одинаковые или похожие действия.

**Тело цикла** – блок кода (набор инструкций), предназначенный для повторения.

**Итерация цикла** – одно выполнение тела цикла.

**Массив** – структура данных, предназначенная для хранения набора однотипных элементов, доступ к которым осуществляется по индексу.

**Метод (функция)** – блок кода, выполняющий определённые операции. Как правило, решает одну задачу, имеет имя и может быть вызван.

**Синтаксис языка** – набор правил, которые определяют допустимые кодовые конструкции в языке.

# План курса

Предлагаю ознакомиться с планом нашего курса, который состоит из 6 лекций и 8 семинарских занятий:

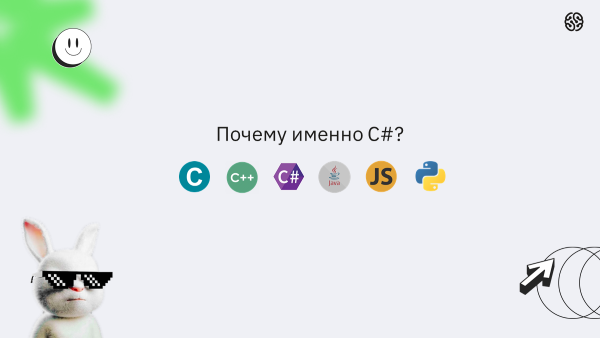
* сегодня мы познакомимся с языком C# и напишем свои первые программы.
* на второй лекции мы вспомним, что такое массивы
* третья лекция будет посвящена функциям
* На четвертой, пятой и шестой лекции мы рассмотрим подходы к написанию хорошего кода, массивы с двумя измерениями и рекурсию.

# План урока

План работ на сегодня следующий

* **поговорим** о том почему же именно C# был выбран инструментом для знакомства с программированием
* далее мы **установим** и **настроим** редактор Visual Studio Code
* **вспомним** основные понятия из курса «Введение в программирование»
* **приступим** к написанию наших первых программ.

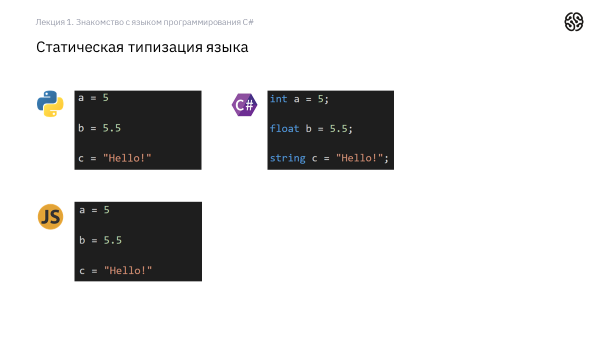
# 1. Почему именно С#?



Друзья, я думаю, что многие из вас при просмотре программы курса задались вопросом: почему именно C# был выбран в качестве основного инструмента? Командой GeekBrains было рассмотрено несколько наиболее популярных языков программирования, представленных на экране.

На текущем уровне изучения программирования все задачи курса будут решаться одинаково на любом из представленных языков. Отличия будут совершенно незначительны. Написав программу на одном из них, вы всегда сможете переписать ее на другой, изменив только синтаксис.

Давайте рассмотрим критерии, по которым язык C# был выбран для текущего курса.

****

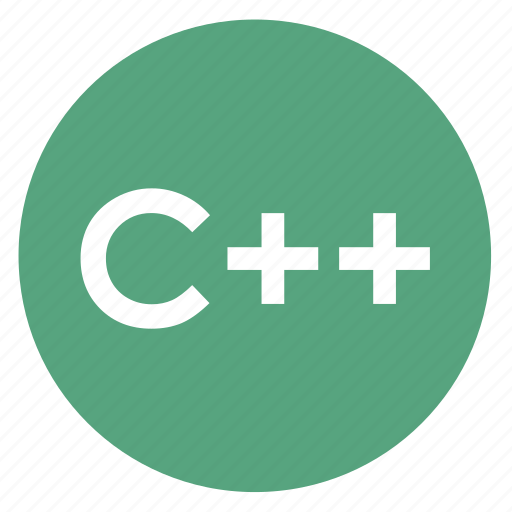
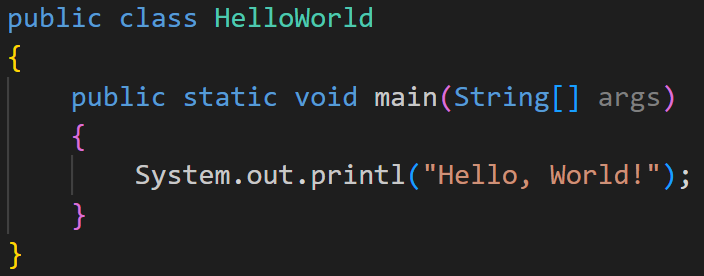
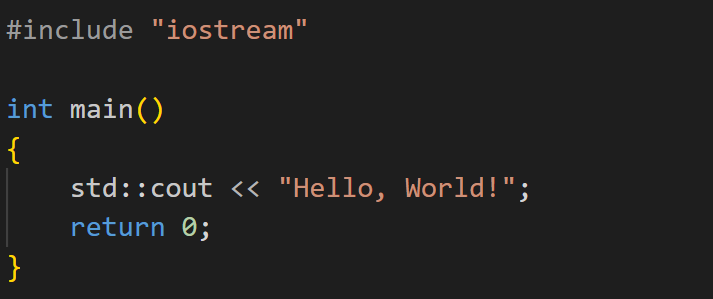
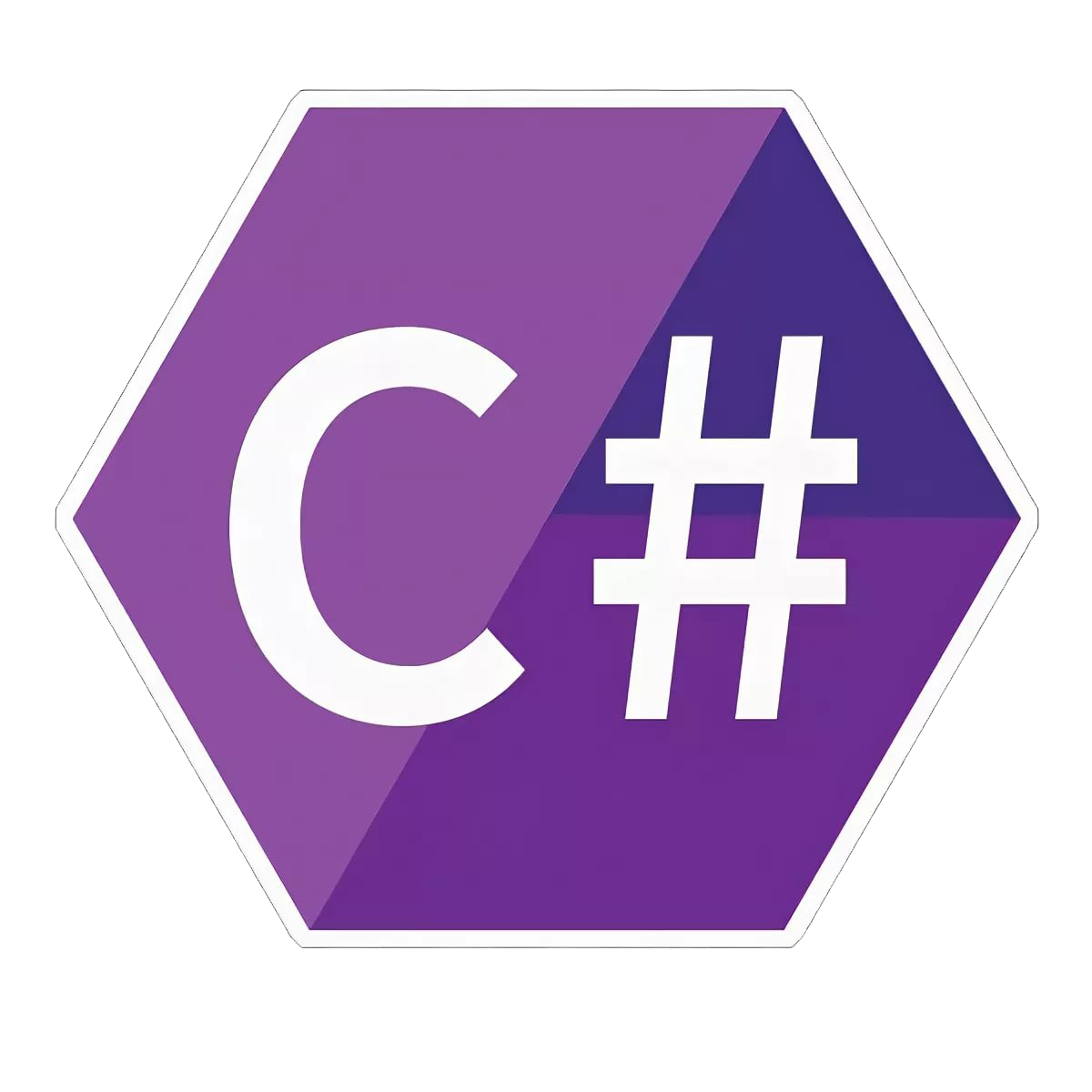
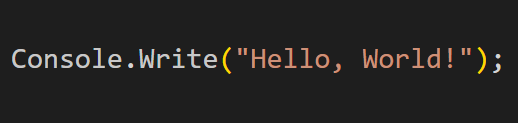
Можно видеть три фрагмента исходного кода на языках Python, JavaScript и C# с созданием трех переменных с типами данных «Целое число», «Дробное число» и «Строка». **Обратите внимание** что в Python и JavaScript тип переменной не указывается, в С# его **необходимо** указывать явно.

Дело в том, что язык C# имеет статическую типизацию, то есть **требует явного указания типа** переменной при ее создании (на самом деле есть возможность это требование обойти, однако на начальных этапах применять эту технику я не рекомендую).

Плюсы этой особенности заключаются в следующем:

* на каждом этапе написания кода вы видите и понимаете, какой тип данных имеет каждая переменная.
* Также отмечу, что переходить с разработки на языке со статической типизацией на язык с динамической значительно проще, чем наоборот. Например, с языка C# на язык Python.

Отмечу, что статическую типизацию также имеют языки С, С++ и Java**. Рассмотрим второе свойство**, которое выделяет C# среди этих языков.

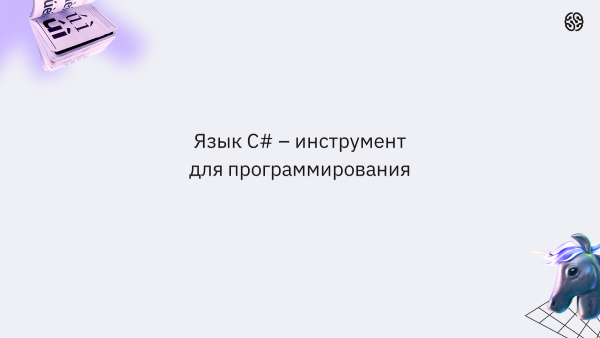
****

Представлены фрагменты кода для вывода приветственного сообщения на консоль. Обратите внимание на использование **дополнительных конструкций** в языках C++ и Java.

На начальных этапах при решении алгоритмических задач, нам не нужны классы, точки входа в приложение и прочие штуковины, которые будут нас отвлекать. В C# мы можем воспользоваться режимом, в котором все эти вещи за нас сделает компилятор, а мы сможем сосредоточиться на создании алгоритмов.

Другими словами, C# позволяет писать код также как на Python, при этом используя все плюсы статической типизации.

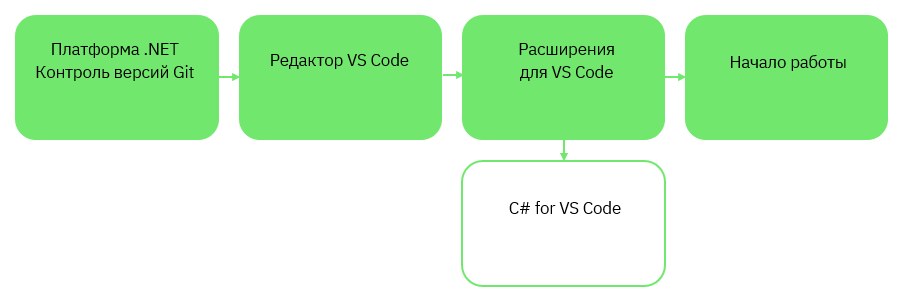
Да, если вдруг часть слов, которые я сказал, вы не поняли или слышите в первый раз - не переживайте. Можете поверить на слово, что C# – это отличный выбор для изучения основ программирования и все полученные знания вы легко сможете перенести на любой понравившийся вам язык.

****

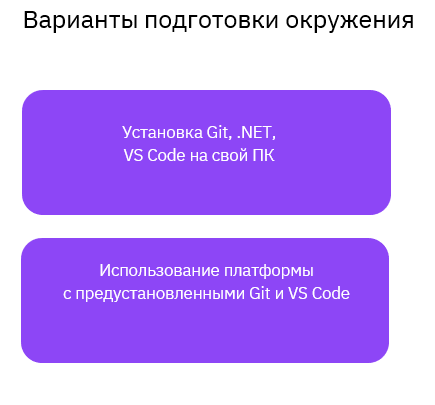
Друзья, обращаю ваше внимание что целью этого курса не является изучение синтаксиса языка C#, его продвинутых инструментов и технологий.

Цель курса – получить навыки разработки алгоритмов различных уровней сложности, а также познакомиться с подходами в разработке программ. И C# в этом деле нам понадобится как инструмент для их создания.

# 2. Подготовка окружения

****

Для использования любого языка нам потребуется установить и настроить окружение. Для языка C# – это платформа .NET. Для удобства написания когда, мы будем использовать редактор Visual Studio Code и систему контроля версий Git, которые у вас уже скорее всего установлены, если нет – инструкция по ним продублирую в описании лекции.

****

Друзья, отмечу, что наиболее приоритетный вариант настройки окружения – это самостоятельная установка и настройка инструментов разработчика.

Однако, если у вас возникнут существенные трудности при настройке окружения, то есть альтернативный вариант. Он заключается в использовании облачного сервиса, который предоставит вам удаленный рабочий стол с установленным Git, .NET и Visual Studio Code. Подключиться и работать с этим сервисом можно напрямую из браузера.

Этим сервисом можно воспользоваться в случае, если ваше аппаратное обеспечение не позволяет установить необходимые инструменты. Инструкции по работе с этим сервисом будут также прикреплены к описанию лекции.

**[Скрин-каст]**

|  |
| --- |
| * Сейчас мы рассмотрим процесс настройки окружения. Проделывать это одновременно со мной не нужно, рекомендую внимательно посмотреть, а после лекции можно будет проделать эти шаги самостоятельно. |

Для установки платформы .NET нам потребуется проделать следующие шаги:

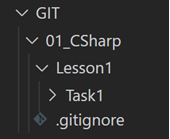
* перейдем на сайт загрузки установщика платформы .NET (<https://dotnet.microsoft.com/en-us/download/dotnet/7.0>)
* загрузим и запустим установщик для вашей платформы. В большинстве случаев это будет 64-разрядный установщик, вам нужно будет выбрать только архитектуру.

В редактор Visual Studio Code могут быть установлены расширения для удобства разработки. Эти расширения добавляют в редактор **подсветку синтаксиса**, **подсказки** разработчику и другие **полезные возможности**.

Откроем Visual Studio Code и для **установки расширения** откроем соответствующую панель, введем в поиске «C#» и выберем расширение C# for Visual Studio Code (оно будет первым в списке). Для его установки нажмем кнопку Install.

На данном этапы **мы готовы** к написанию программ на C#. В качестве первой программы предлагаю реализовать вывод сообщения «Hello, World!» на экран. О такой программе вы наверняка уже слышали, а может быть, даже её писали. В большинстве случаев, она используется для того, чтобы понять, правильно ли настроено ваше окружение.

**Создание проекта**

****

Итак, перейдём в VS Code. Работать мы будем в каталоге Lesson1. Создадим в нём каталог Task1.

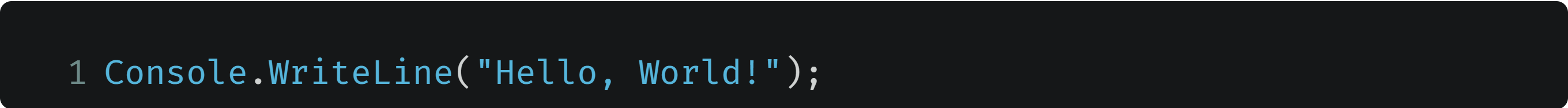
|  |
| --- |
| * Создавать каталоги можно командой mkdir из терминала, для перехода между каталогами используется команда cd. |

|  |
| --- |
| * Друзья, рекомендую создавать осмысленные иерархии каталогов, чтобы вам проще было в них ориентироваться как в локальном репозитории, так и в удаленном. |

Для написания программы нам необходимо создать проект, в котором будет располагаться файл с исходным кодом.

Перейдём в каталог Task1 и выполним там команду dotnet new console, которая создаст нам **новое консольное приложение**. После выполнения команды можно видеть, что создался файл для исходного кода Program1.cs и файл проекта .csproj. Писать код мы будем в файле Program1.cs.

**Написание кода**

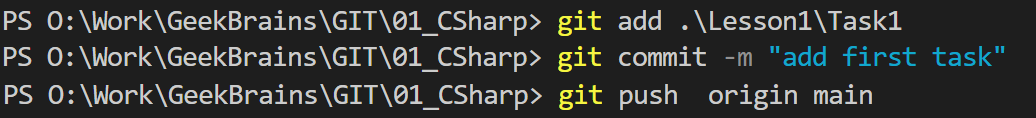


Для вывода на экран сообщений, значений переменных в C# используется метод WriteLine модуля Console.

Попробуем запустить наш код, для этого в каталоге с файлом Program1.cs выполним команду dotnet run. Эта команда **скомпилирует** исполняемый файл на основе нашего кода и **запустит** его. Как и ожидалось, на экране мы видим приветственное сообщение.

**Работа с GIT**

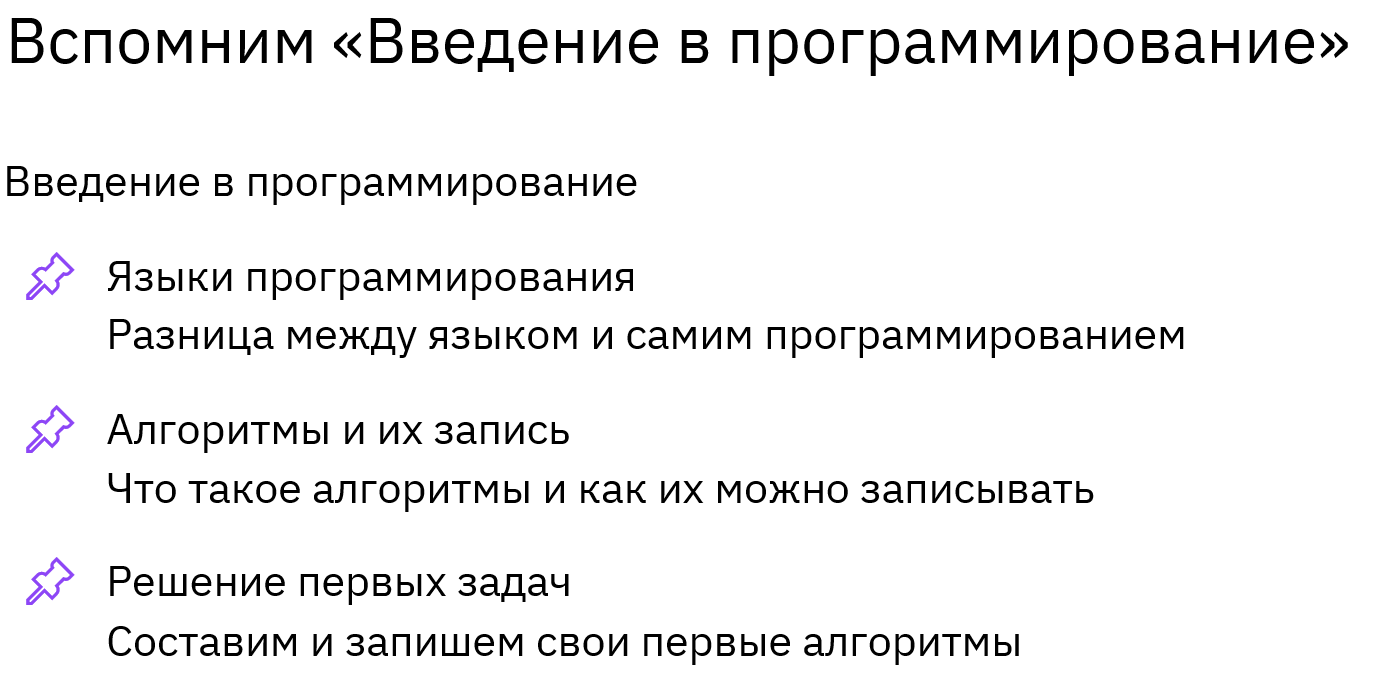
Сохраним наши наработки в репозитории Git, с которым вы практиковались на предыдущем курсе. Мой каталог 01\_CSharp уже подключен к удаленному репозиторию. Командой git add мы индексируем файлы и каталоги, которые будут включены в коммит. Далее командой git commit формируем коммит с указанием комментария и загружаем наработки на удаленный репозиторий командой git push.



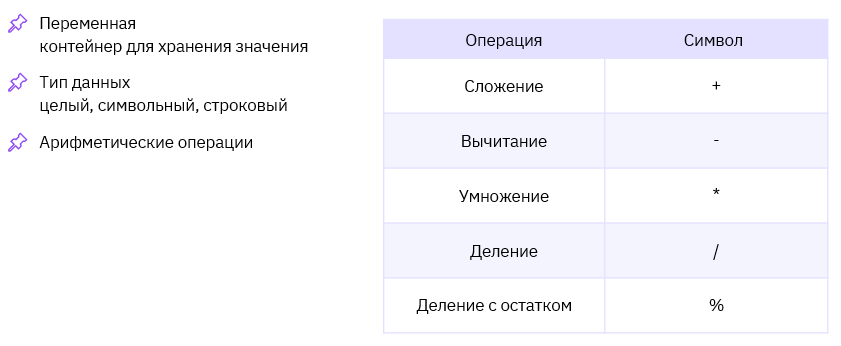
|  |
| --- |
| * Отмечу, что каталог вашего локального репозитория должен содержать файл .gitignore, который предотвратит отправку лишних файлов (временных, бинарных ) в удаленный репозиторий. |

В нашем примере это каталог 01\_CSharp, именно он подключен к удаленному репозиторию и содержит файл .gitignore, настроенный для работы с проектами .NET. Пример этого файла будет доступен в описании к лекции.

# 3. Вспомним «Введение в программирование»

****

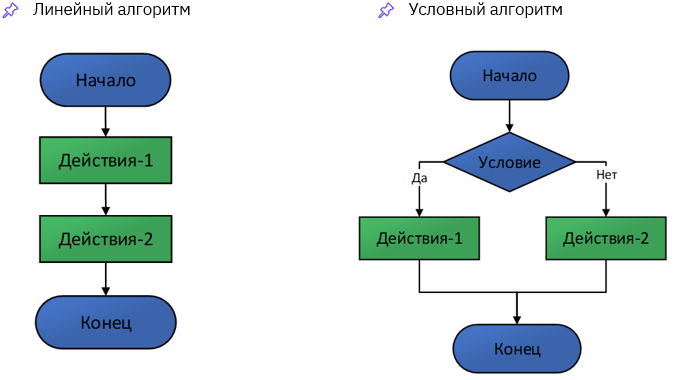
Итак, окружение настроено и проверено. Предлагаю вспомнить базовые термины из курса «Введение в программирование». Первая группа терминов связана с переменными и операциями над ним, вторая – с типами алгоритмических конструкций.

****

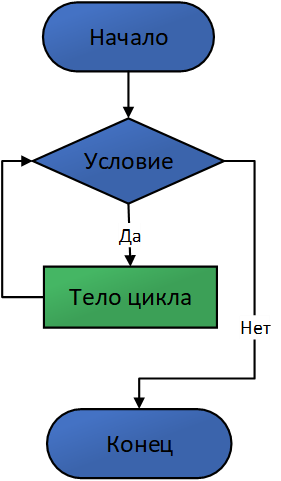
Первый термин – это переменная. Под переменной мы понимаем область памяти, которая предназначена для хранения некоторого значения. Переменной можно дать и более простое определение, например назвав ее контейнером для хранения некоторого значения.

Переменная имеет некоторую характеристику, которую мы называем типом данных. Тип данных говорит нам о том, какого рода значения могут храниться в переменной, например – числа, символы, строки.

Важное понятие для тех, кто знакомится с программированием – это арифметика или арифметические операции. Среди них можно выделить: сложение, вычитание, умножение, деление, а также более сложные операции – деление с остатком, возведение степень или извлечение квадратного корня. Практически во всех задачах текущего курса вы будете их применять.

****

Вторая группа терминов описывает типы алгоритмов. Напомню, линейный алгоритм предполагает выполнение инструкций друг за другом в порядке их записи. Условный алгоритм предполагает выбор действий в зависимости от некоторого условия, иными словами – разветвление на два и более путей выполнения.

****

И наконец, циклический алгоритм предполагает повторение каких-либо действий.

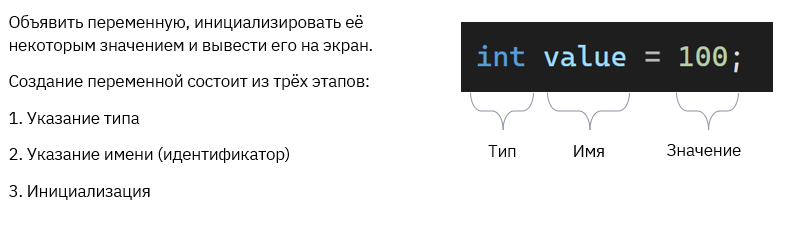
# 4. Переменные и арифметика

Итак друзья, мы настроили окружение, вспомнили базовые термины, давайте приступим к знакомству с синтаксисом языка C# и решению задач. Начать предлагаю с базовых вещей – с создания переменных и арифметики.

## 4.1. Пример 1

Итак, рассмотрим первый пример.

В нём требуется объявить переменную, задать ей начальное значение и вывести его на экран. Перед тем как начать писать код, давайте посмотрим как в C# создаются переменные.

****

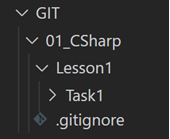
Первая часть создания переменной – это указание типа данных для значения, которое она будет хранить. О **доступных типах** данных в C# мы скажем отдельно.

Вторая часть – это имя переменной, эту часть ещё называют идентификатор. Имя должно состоять из латинских букв и цифр, допускается символ подчеркивая. Есть важные ограничения на именование переменных: имя не должно начинаться с цифр и не должно совпадать с зарезервированными словами языка C#.

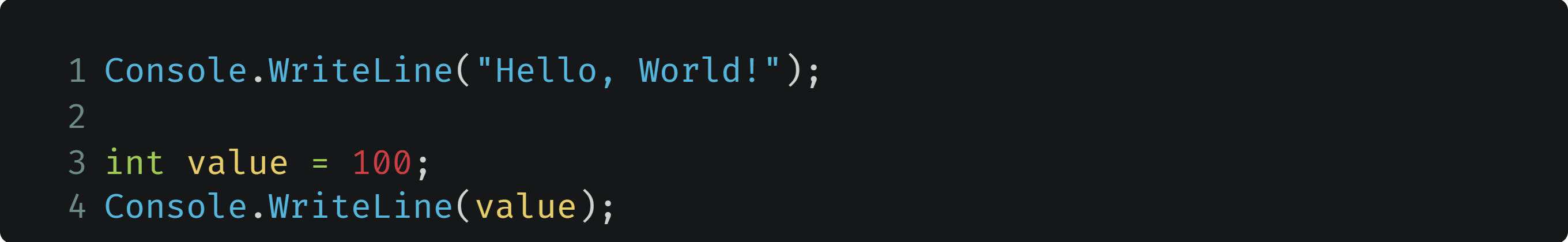
Ну и наконец третья часть – инициализация. Под этим терминов мы пониманием присвоение переменной конкретного значения.

**[Скрин-каст]**

Запрограммируем этот пример на C#. Давайте дополним наш проект Task1, который на данный момент содержит только вывод приветственного сообщения на экран.

****

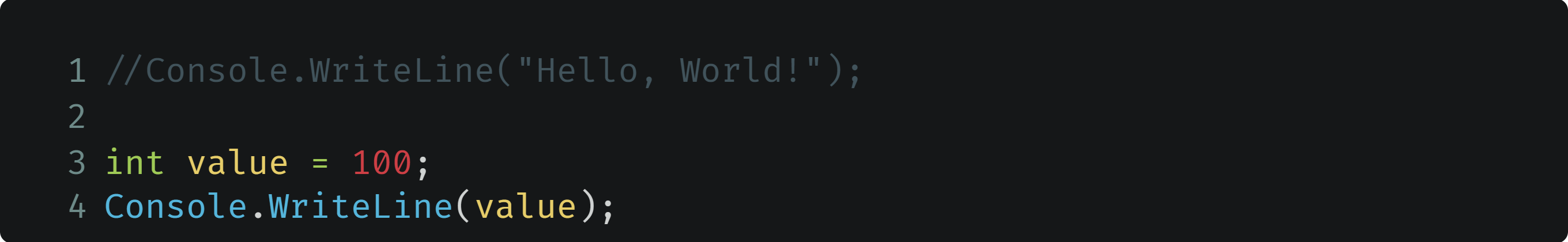
**Написание кода**



Создадим переменную, которая может хранить целые числа. Зададим тип данных int (от английского integer – целый), зададим имя, допустим value и присвоим переменной некоторое начальное значение, например 100.

Для вывода на экран воспользуемся тем же методом Console.WriteLine, который мы использовали ранее. Запустим программу с помощью команды dotnet run. На экране видим вывод сообщения «Привет мир» и значение нашей созданной переменной.

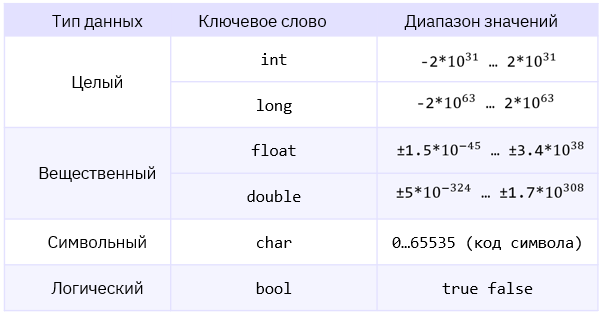
Давайте закомментируем первую строкой с выводом на экран приветственного сообщения, оставим только вывод переменной. Чтобы закомментировать строку в языке C#, перед ней необходимо поставить два символа косой черты («слэш»).



|  |
| --- |
| * Обращаю ваше внимание, что закомментированные строки выполнены не будут. Комментарии могут использоваться для отключения участков кода, а также чтобы давать пояснения к каким-либо действиям. |

Запустим программу, на экране увидим только значение переменной value.

Друзья, мы создали переменную value для хранения целых чисел. Но что, если нам **потребуется** ввестив программу **числа с дробной частью** или, например, **символы**? Для этого нам потребуется использовать другие типы данных.

****

Давайте посмотрим, какие типы данных мы можем использовать в своих программах. С первым типом мы уже познакомились, это целый тип, также его называют целочисленным. Еще один целочисленный тип носит название long, он отличается от int бОльшим размером. То есть long позволяет сохранить бОльшие по величине числа, чем int.

Второй представленный тип данных – это вещественный. Этот тип позволяет хранить дробные числа, при этом способен сохранить достаточно большую дробную часть. Существуют две его разновидности – float и double, отличаются размером. Тип double позволяет хранить дробные числа с большей точностью, чем float.

Символьный тип используется для хранения отдельных символов – будь то буквенных, цифровых или же специальных. Отмечу, что из символов можно формировать строки.

Последний в списке – это логический тип данных, который принимает всего лишь два значения: истина (true) и ложь (false). Этот тип данных мы часто можем видеть в ветвлениях и циклах при создании условий.

Давайте расширим нашу программу и добавим в неё переменные рассмотренных типов данных.

**[Скрин-каст]**

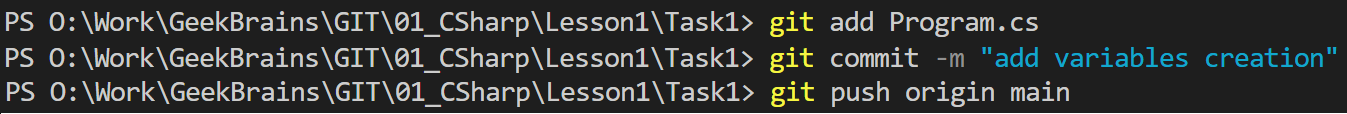
Итак, вернемся в проект Task1, откроем файл Program1.cs.



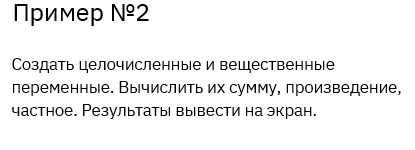
Дополним наш код переменной вещественного типа double, назовём ее например d и присвоим значение 5.55; переменной символьного типа char, назовём ее например c и присвоим значение ‘!’; переменной логического типа bool, назовём ее например b и присвоим значение true. С помощью метода Console.WriteLine выведем значения всех созданных переменных. Запустим программу командой dotnet run, как и ожидалось, на экране мы видим значения четырёх созданных переменных.

**Работа с GIT**

Сохраним наши наработки в репозитории GIT. Командой git add мы индексируем файлы и каталоги, которые будут включены в коммит. Далее командой git commit формируем коммит с указанием комментария и загружаем на удаленный репозиторий командой git push.



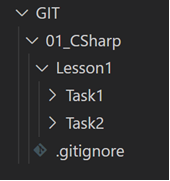
## 4.2. Пример 2

****

Друзья, рассмотрим второй пример. В этом примере мы создадим переменные целого и вещественного типа, применим к ним арифметические операции и посмотрим, какие особенности при этом могут возникать.

**[Скрин-каст]**

**Создание проекта**

****

Создадим для этого примера каталог Task2 в каталоге Lesson1, и создадим там проект командой dotnet new console.

**Написание кода**

**Операция суммирования**

Для начала создадим две целочисленные переменные i1 и i2, присвоим им некоторые значения, например 5 и 6. Затем вычислим их сумму с помощью оператора «+» и поместим результат в новую переменную sum.

**Но здесь возникает вопрос**: а какой тип данных должен быть у переменной sum? Давайте остановимся на этом вопросе чуть подробнее.

Друзья, сложение – это **бинарная операция**, то есть в ней участвуют два операнда, или два элемента (слева от плюса и справа). Так вот, тип данных результата этого сложения будет зависеть от типов данных операндов. Иными словами, тип данных результата будет формироваться на основе типа данных левого слагаемого и правого. А **каким образом он будет формироваться**? В зависимости от типа данных операндов сформулируем два правила.

**Первое правило**. Если левый и правый операнд имеют один и тот же тип, то и результат операции будет иметь этот же тип. В нашем примере, переменная sum будет иметь тип int. Добавим вывод результата на экран.



Обратите внимание, что для переменной, которая хранит сумму, мы даём более осмысленное имя, чем просто букву алфавита. Это считается хорошей практикой программирования и улучшает читаемость кода.

Запустим наш пример командой dotnet run. Видим на экране 11, как и ожидалось. Давайте рассмотрим другой случай, когда **типы данных** у операндов **разные**.



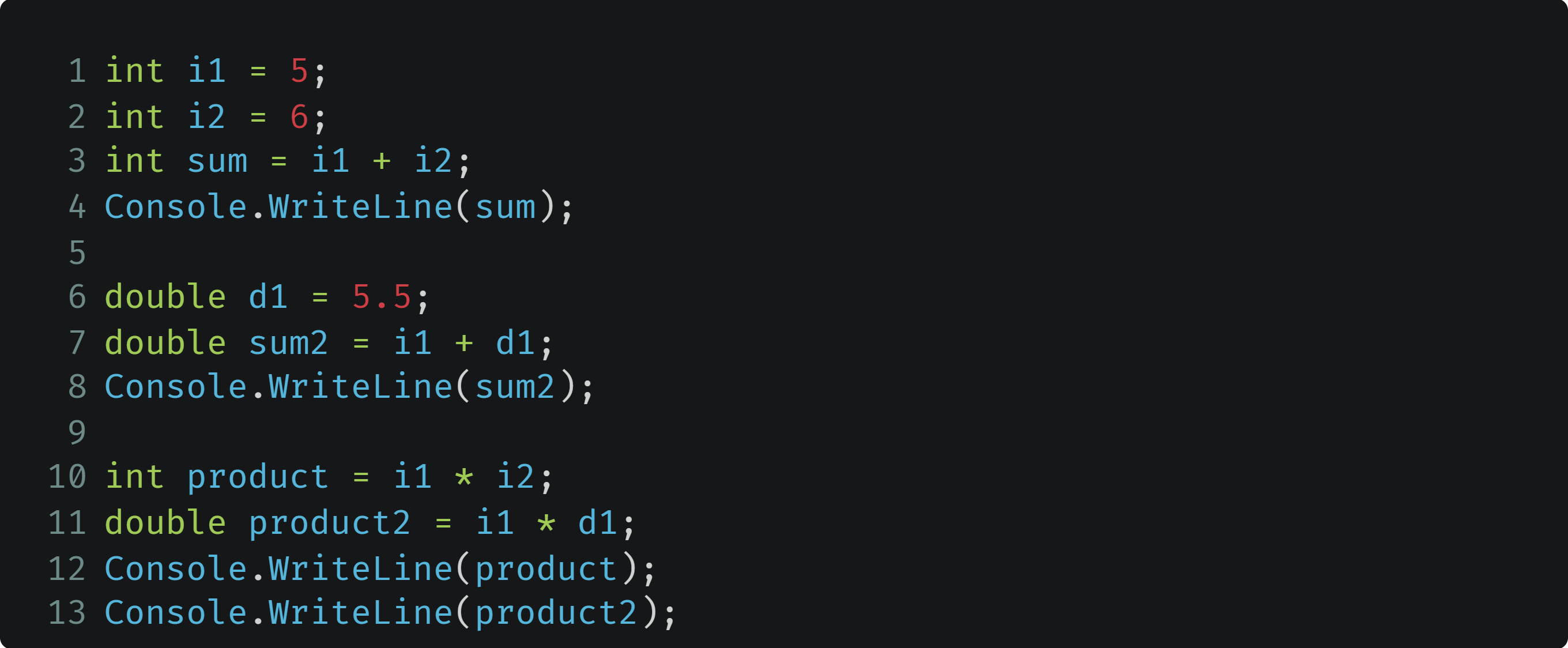
Итак, введём целочисленную переменную long l1, присвоим ей некоторое значение, например 6. Сложим переменные i1 и l1, результат поместим в новую переменную. Назовём её sum2, поскольку имя sum уже занято. Возникает вопрос, а как будет формироваться тип данных у результата сложения в этом случае?

Давайте сформулируем **второе правило**. Если типы данных у операндов разные, как в нашем примере, то результат бинарной операции будет иметь более вместительный тип. Под вместительностью понимается более большой или объемный тип. Если в операции участвуют типы int и long, то long будет более вместителен (поскольку диапазон типа данных long в разы больше, чем диапазон int). Если в операции участвуют типы int и double, то double будет более вместителен (поскольку тип данных double включает в себя целые числа и дробные).



Запустим наш пример командой dotnet run.

**Операция умножения**



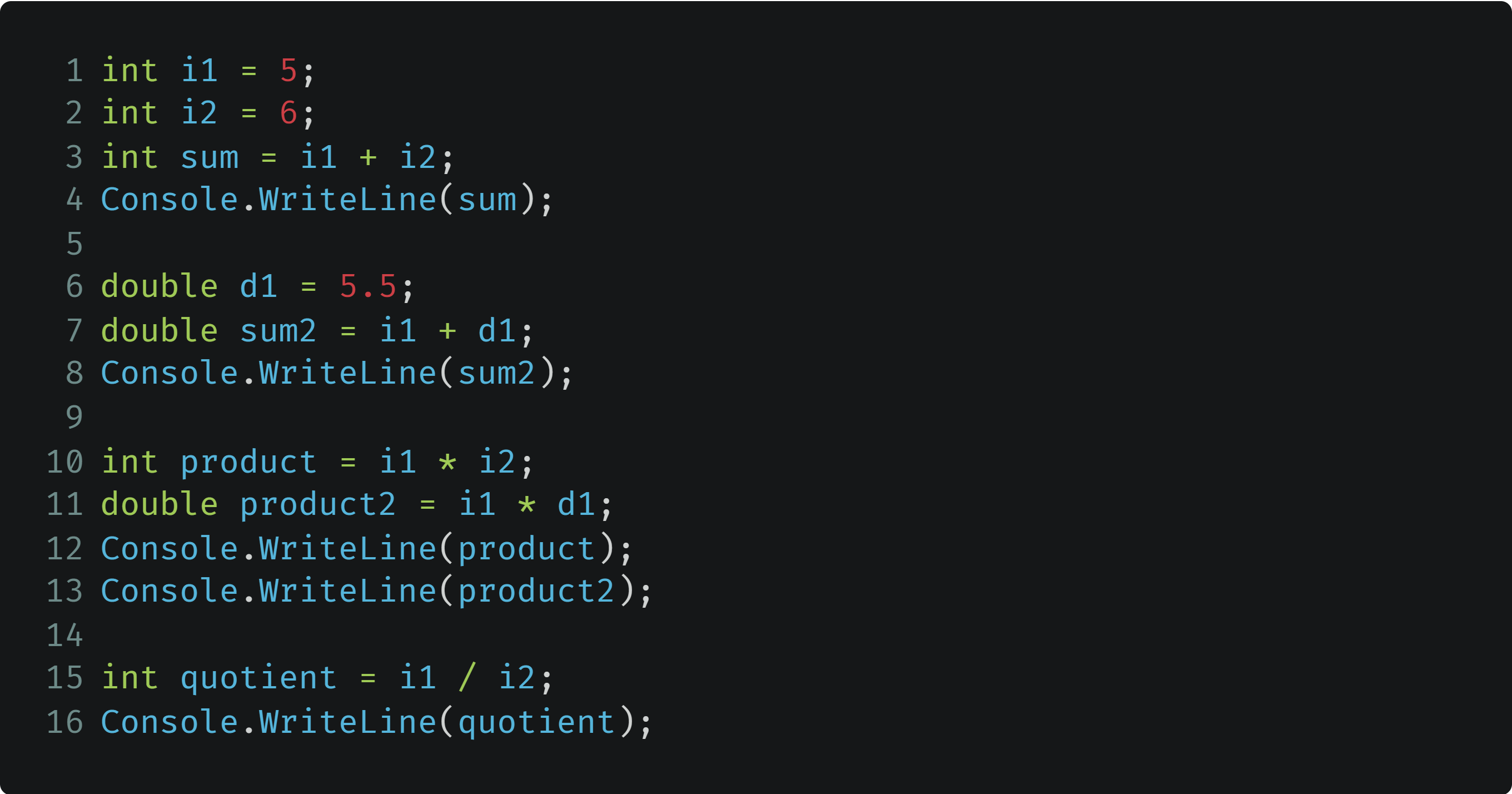
Проделаем эти же шаги, только с операцией умножения. Перемножим целочисленные переменные, результат будет иметь тип int. Дадим ему имя product, что в переводе с английского «произведение».

|  |
| --- |
| * Кстати, как у вас дела с английским? В чат можете написать свой уровень владения. |

Перемножим переменные типа int и double, результат будет иметь наиболее вместительный тип из двух. Это тип double.

**Операция деления**

Друзья, при делении переменных правило формирования типа данных у результата работает точно также, как и в сложении и умножении. Однако, при делении целых чисел результат может получаться дробным. Давайте рассмотрим этот случай.



Итак, разделим переменную i1 на i2. Обе переменные имеют тип int, следовательно результат будет иметь тоже тип int. Дадим ему имя quotient, что в переводе с английского «частное». Выведем результат на экран. Запустим наш пример командой dotnet run. Ожидаем увидеть примерно 0,8, однако на экране мы видим ноль.

|  |
| --- |
| * Как вы думаете, почему так произошло? |

На самом деле, каких-то странностей или магии здесь нет. Целый тип не предполагает хранение дробной части. При делении целых чисел мы получаем целое число и дробная часть в нём не может быть сохранена. Возникает вопрос: а как тогда сохранить дробную часть, которая может получиться при делении?

Ответ на вопрос кроется в **типе данных результата деления**. Если он будет вещественным, то он позволит сохранить дробную часть, которая получилась при делении. Для этого достаточно, чтобы **хотя бы один операнд имел вещественный тип**, тогда и результат будет иметь вещественный тип. Рассмотрим пример.



Изменим значение переменной d1 на 6. В переменной i1 у нас хранится 5. Вычислим их частное. Согласно второму правилу, он будет иметь тип double. Поместим результат деления в переменную quotient2.

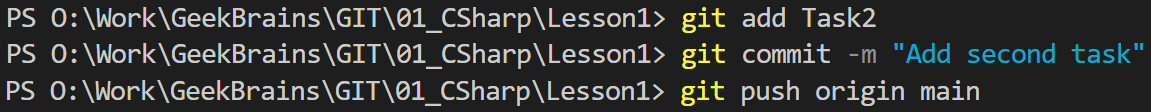
Запустим наш пример, ожидаем увидеть примерно 0,8. Наши ожидания соответствуют результату.

|  |
| --- |
| * Друзья, эти примеры могут показаться сложными, но я настоятельно рекомендую вам с ними разобраться и поэкспериментировать на своих числах. Эта важная тематика с точки зрения понимания взаимодействия различных типов данных. |

|  |
| --- |
| * Также еще один важный момент. Не стоит думать, что в языке Python или JavaScript например, эти особенности формирования типов данных отсутствуют. Они присутствуют в полной мере, только скрыты от разработчика. Язык C# позволяет нам достаточно детально с ними познакомиться и понять они устроены «под капотом». |

**Работа с GIT**

Сохраним наши наработки в репозитории GIT. Проиндексируем файлы и каталоги, которые будут включены в коммит. Сформируем коммит с указанием комментария и загрузим на удаленный репозиторий.



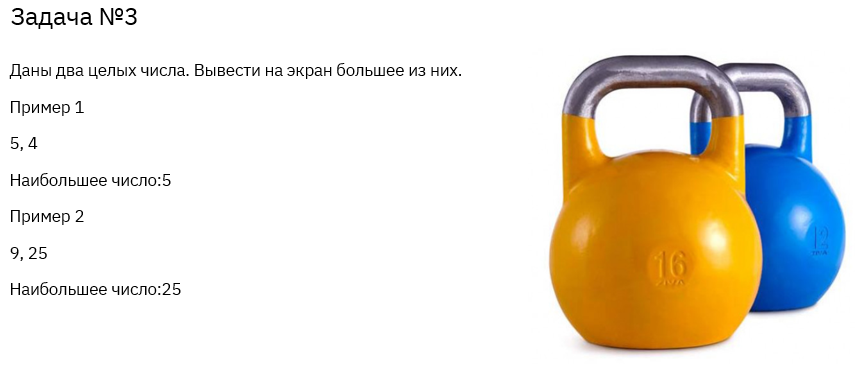
**[слайд «Домашнее задание»]**

|  |
| --- |
| * Друзья, для закрепления полученных навыков даю вам домашнее задание. Вычислить значение формулы , где a,b,c,d – некоторые целые числа. Результат вывести на экран. |

# 5. Задача с двумя гирями

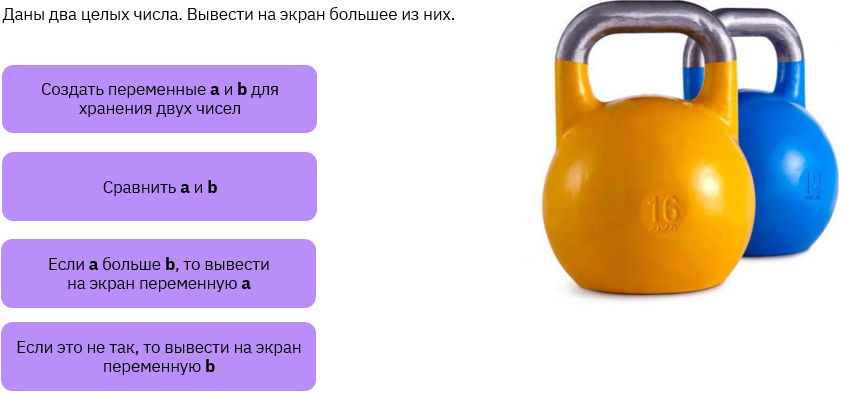
****

Друзья, давайте вспомним задачу с гирями из курса «Введение в программирование». Напомню, в этой задаче на вход поступало 5 гирь с произвольными весами и нужно было выяснить вес самой тяжелой гири. Но перед тем, как её решать с помощью языка C#, предлагаю решить несколько упрощенный вариант – когда гирь всего две и также требуется найти вес наиболее тяжелой гири.

****

Если приводить аналогию с набором чисел, то задача может быть сформулирована следующим образом: дано два целых числа, нужно определить величину большего из них.

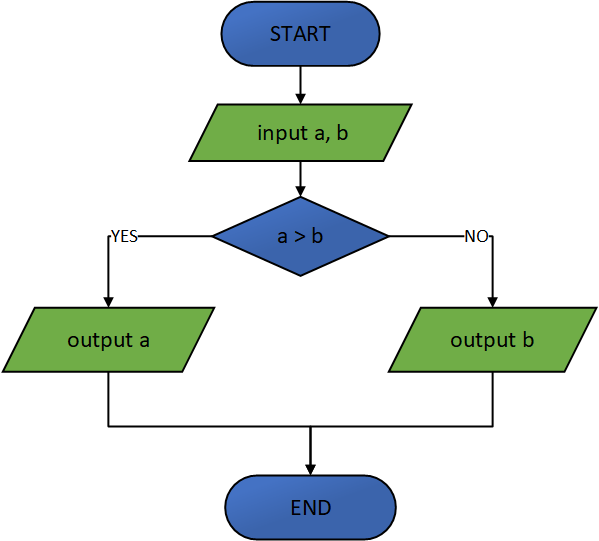
Давайте выделим основные этапы в решении этой задачи и построим блок-схему.

****

**Словесное описание алгоритма**

Сразу скажу, что **вариантов решения** этой задачи может быть **несколько**, мы с вами разберем один из возможных.

* Как и в предыдущих примерах, начнём с создания двух переменных, для хранения исходных чисел, например a и b.
* Затем путем сравнения выясним, какая из переменных a и b больше.
* Если a больше чем b, то выведем значение переменной a на экран.
* Если же это не так, то выведем значение переменной b на экран.

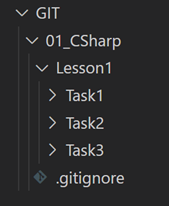
****

Изобразим блок-схему, которая соответствует выделенным этапам. Обращаю внимание, что в **случае равенства** значений переменных a и b, мы попадём в блок вывода переменной b, поскольку выражение a>b даст нам в этом случае результат НЕТ.

**[Скрин-каст]**

Итак, после словесного описания алгоритма и оформления его в виде блок-схемы давайте оживим этот алгоритм на языке C#.

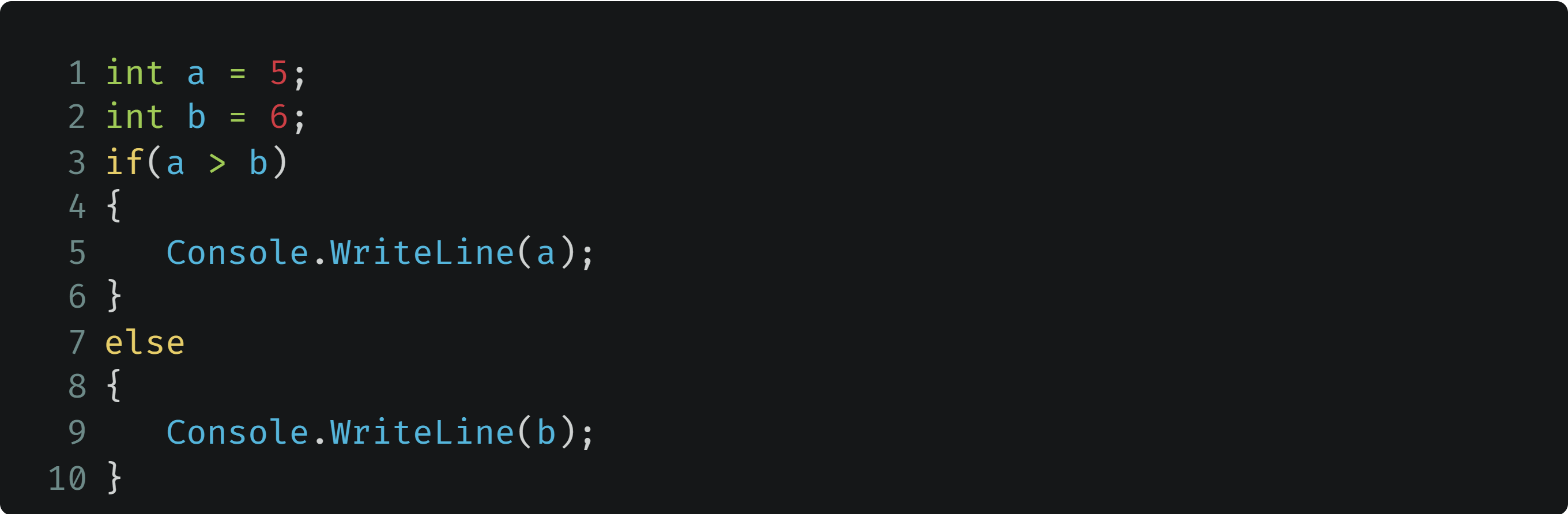
**Создание проекта**

****

Оформим решение этой задачи в новом проекте. Создадим каталог Task3, создадим в нём проект.

**Написание кода**

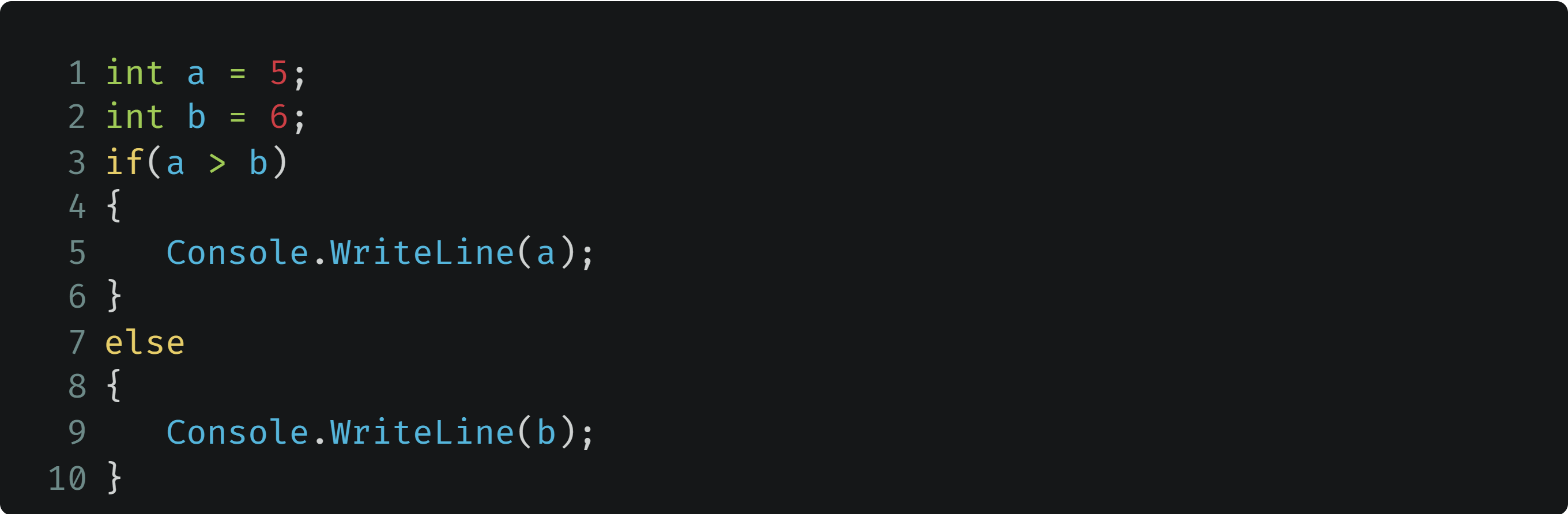
Согласно нашей блок-схеме, создадим две переменных типа int и проинициализируем их произвольными целыми числами, например 5 и 6.



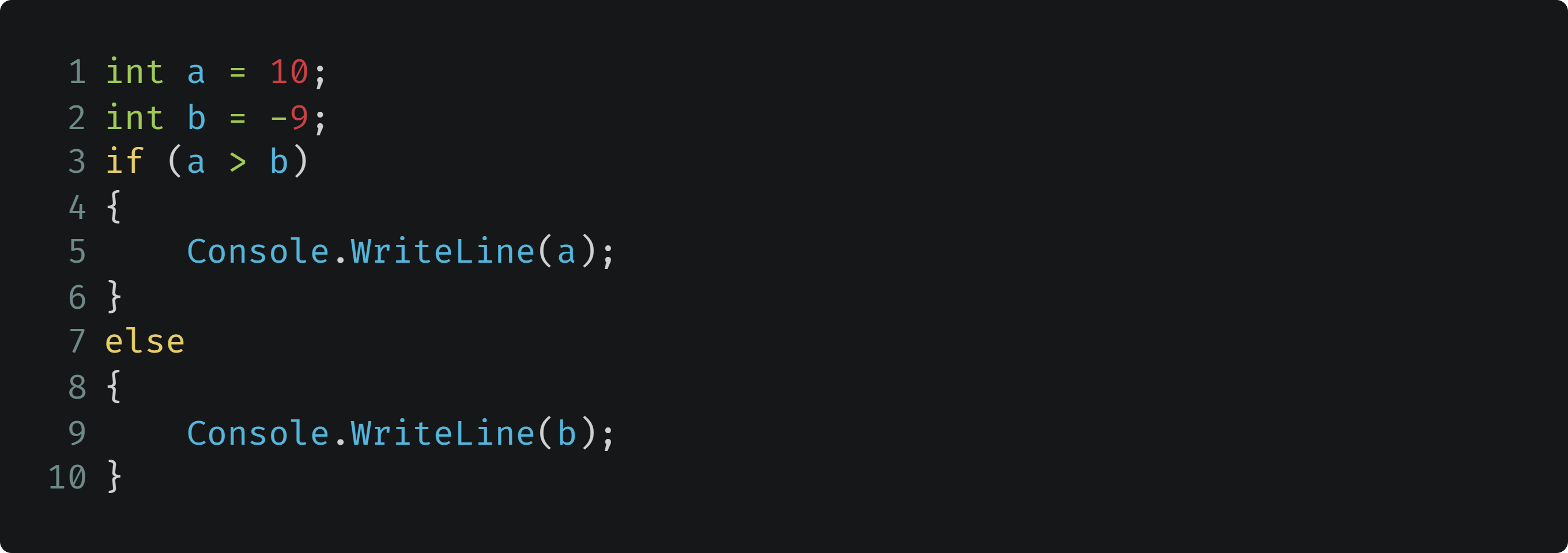
Далее в блок-схеме мы видим блок Решение, которое говорит, что используется условный алгоритм. В большинстве языков программирования для создания условий используется оператор if.

После ключевого слова if в круглых скобках мы указываем условие, то есть некоторое выражение, которое имеет только два возможных значения ИСТИНА или ЛОЖЬ, как ветви в блок-схеме, ДА или НЕТ.

Действия, относящиеся к ветви ДА, мы помещаем в фигурные скобки сразу после условия. В нашем случае, это вывод на экран переменной a с помощью метода WriteLine из модуля Console.



Для оформления действий, относящихся к ветки НЕТ, нам необходимо использовать ключевое слово else после блока фигурных скобок. Сами действия мы также помещаем в фигурные скобки. В нашем случае, это вывод на экран переменной b.



Иными словами, если условие верно, то будет выполнен код в фигурных скобках сразу после него. Если же неверно, то будет выполнен код в фигурных скобках после ключевого слова else.

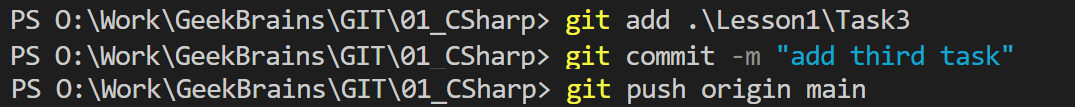
Запустим наш код. Как и ожидалось, на экране мы видим максимальное из чисел 5 и 6, это значение 6.

|  |
| --- |
| * Рекомендую запустить программу с различными значениями переменных a и b. Так вы протестируете работу алгоритма при различных начальных условиях. |

Например, зададим значения переменных a=10, b=-9. Запустим наш код. На экране мы видим число 10, что говорит о правильной работе алгоритма.

**Работа с GIT**

Сохраним наши наработки в репозитории GIT. Целиком проиндексируем созданный каталог Task2. Далее сформируем коммит и загрузим его на удаленный репозиторий.

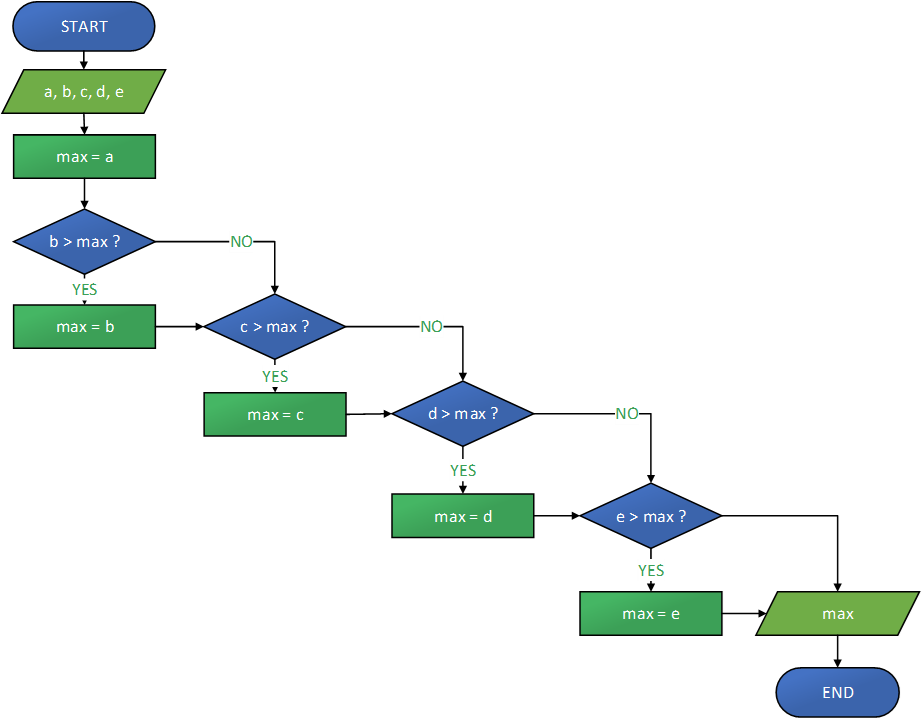


# 6. Задача с пятью гирями

****Итак, друзья, давайте рассмотрим эту же задачу с пятью гирями.

Если приводить аналогию с набором чисел, то задача может быть сформулирована следующим образом: дано пять произвольных целых чисел, нужно определить, какое из них наибольшее.

Друзья, давайте вспомним алгоритм, которой был предложен для решения этой задачи.

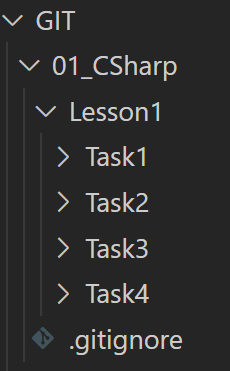


* Чтобы сохранить наибольшее значение, вводилась переменная max и ей присваивается значение первого числа (в данном случае это переменная a).
* Далее мы поочередно производили четыре сравнения чтобы выяснить, не превысило ли очередное число текущее значение максимума (это можно видеть в блоках Условие).
* Если это происходит, то мы обновляем значение переменной max, помещая в нее это очередное число (как видно в блоках Процесс ниже блоков Решение).
* Если же нет, то ничего не делаем и переходим к следующему числу. В конце алгоритма в переменной max будет храниться максимальное из пяти чисел.

Обратите внимание, что подход для решения этой задачи несколько отличается от того, который мы рассмотрели в задаче с двумя гирями. Это делается для сокращения объёма кода и повышения его читаемости.

**[Скрин-каст]**

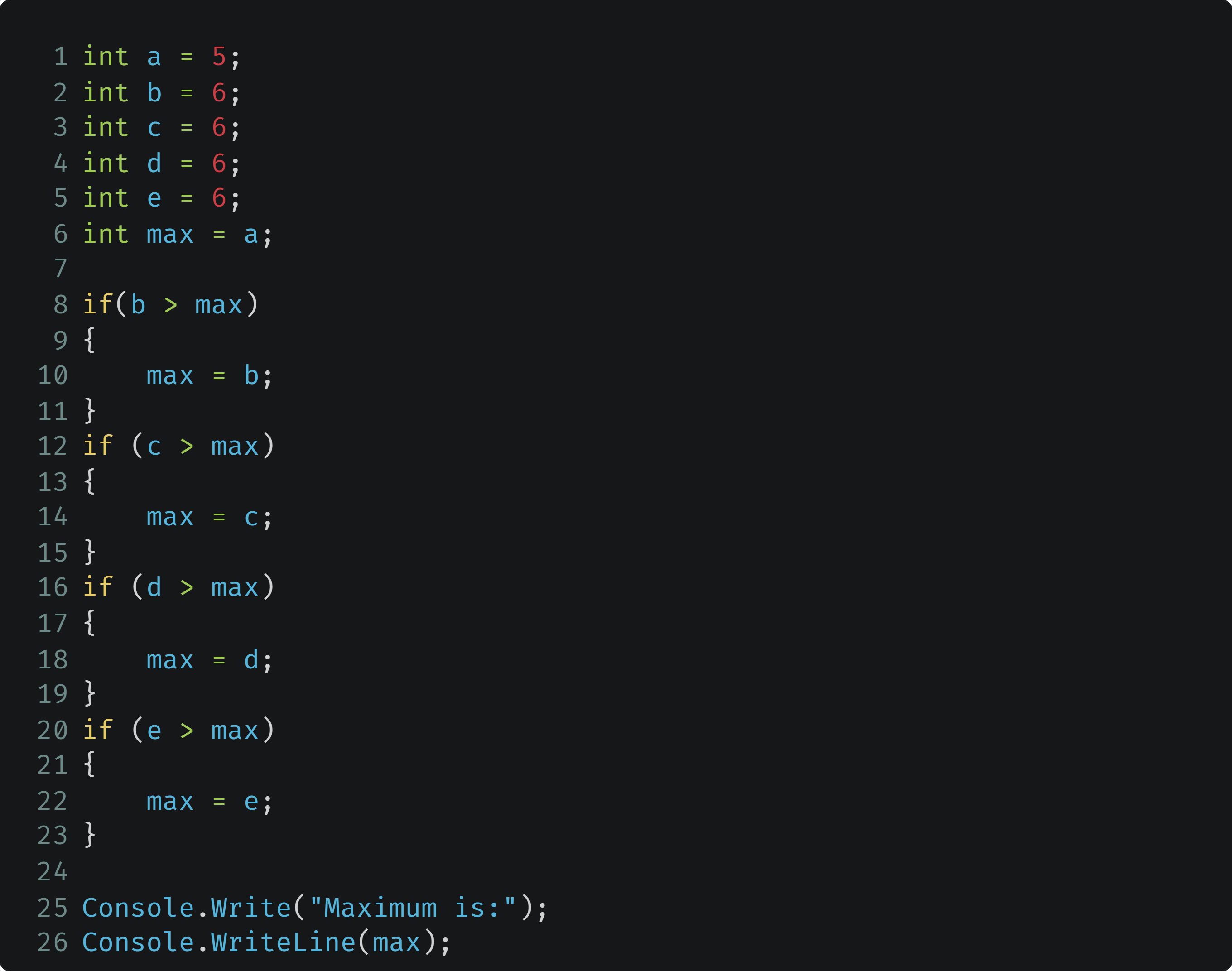
**Создание проекта**

****

Создадим каталог Task4.

Перейдём в него и создадим проект. Откроем файл Program1.cs.

**Написание кода**



Согласно блок-схеме, создадим пять переменных типа int и проинициализируем их произвольными целыми числами. Далее создадим переменную этого же для хранения максимального значения из пяти целых чисел, назовем ее max и проинициализируем первым числом.

Следующим шагом, как и в блок-схеме, произведём четыре сравнения, чтобы выяснить, не превышает ли каждое из оставшихся четырёх чисел переменную max. Если превышает, то обновим значение переменной max. Для ускорения разработки фрагменты кода можно копировать, делая необходимые замены в именах переменных.

|  |
| --- |
| * Еще раз обращаю ваше внимание, что выражение в условии оператора if должно иметь логический тип. То есть, оно может иметь только два возможных значения – TRUE или FALSE. |

|  |
| --- |
| * Также обращаю внимание, что ветвь else в данном алгоритме мы не используем, поскольку не предусмотрено какого-то отдельного действия, если очередное число оказалось меньше максимального. В данном случае мы просто переходим к следующему сравнению. |

Не забываем вывести ответ на экран, чтобы можно было проверить правильность. Воспользуемся методом Console.WriteLine.

Дополним вывод **пояснительным сообщением** «Maximum is:», после которого будет идти значение переменной max. Для этого сначала вызовем метод Console.Write с сообщением «Maximum is:», а затем уже выведем найденное максимальное значение с помощью метода Console.WriteLine.

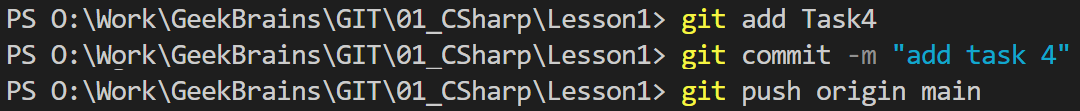
|  |
| --- |
| * Отличие методов Write и WriteLine заключается в том, что метод WriteLine добавляет в конце вывода переход на новую строку. |

Запустим наш код. Как и ожидалось, на экране мы видим максимальное из пяти введенных чисел.

|  |
| --- |
| * Рекомендую запустить программу на различных наборах входных чисел, в частности, когда наибольшее число в начале, посередине или в конце набора. Так вы протестируете работу алгоритма при различных начальных условиях. |

**Работа с GIT**

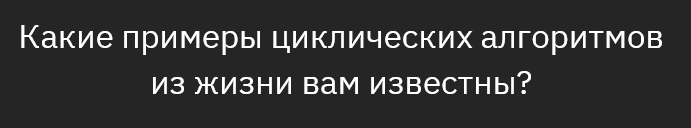
Сохраним наши наработки в репозитории GIT. Проиндексируем каталог Task3. Сформируем коммит и загрузим его на удаленный репозиторий.



# 6. Циклы

Ну что друзья, мы дошли до финальной темы сегодняшней лекции – это циклы в языке C#. В этом разделе мы вспомним оформление цикла в виде блок-схемы и запрограммируем его на языке C#.

Напомню, цикл – важнейший тип алгоритма, используемый для повторения каких-либо действий. На практике циклические конструкции встречаются достаточно часто, можно даже сказать почти всегда.

****

|  |
| --- |
| * Предлагаю вам привести 2-3 примера использования циклов в реальной жизни. Свои **идеи** можете написать в чат. |

А я пока покажу свои варианты.

****Я могу привести следующие примеры. Первый – выгрузка непрочитанных сообщений из электронной почты. В этом случае почтовый клиент **извлекает сообщения с сервера** по одному, пока они не закончатся.

Второй пример – антивирусная проверка компьютера. Антивирус сканирует каждый файл **в указанном хранилище**, пока не просмотрит их все.

Ну и третий пример – это поисковые задачи, в частности **поиск записей в каталоге**, или **поиск объектов** **в** каком-либо **хранилище**. В этом случае мы просматриваем по очереди все объекты, **пока не найдем** искомый объект или **не дойдём до конца**.

**Задача 5**

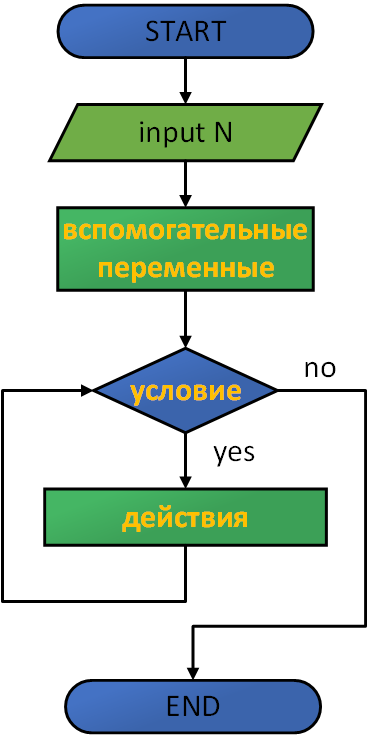
Итак друзья, давайте рассмотрим базовую задачу на применение циклических конструкций. На вход программе подаётся некоторое натуральное число N. Необходимо вывести на экран числа от 1 до этого введённого числа.

Для начала предлагаю выделить основные этапы решения этой задачи, затем мы построим блок-схему и запрограммируем наш алгоритм.

Первым шагом введём натуральное число N. Далее нам необходимо каким-то образом получить числа от 1 до N, при этом вывести каждое из них на экран.

|  |
| --- |
| * Друзья, как вы думаете, какой тип алгоритма нам здесь понадобится? |

Совершенно верно, если вашим предположением был цикл. Цикл мы используем, когда количество однотипных действий **заранее неизвестно** или же их количество **превышает 2-3 повторения**.

****

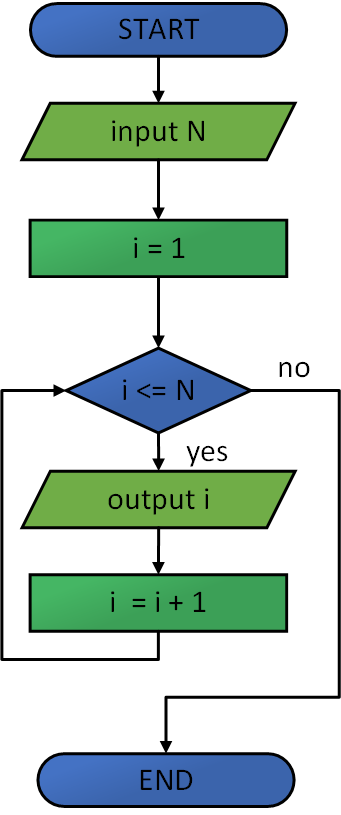
Следующим шагом C# построим блок-схему нашего алгоритма.

Для создания блок-схемы цикла нам нужно определиться с его составляющими компонентами:

1. Первый компонент – это действие, которое нужно повторять.
2. Второй – это вспомогательные переменные. Часто в их роли выступает счётчик цикла, но могут быть и другие.
3. И третий компонент – условие цикла. Напомню, под условием мы понимаем выражение логического типа, которое определяет, будет ли происходить очередная итерация или нет.

Друзья, в течение 2-3 минут предлагаю вам попробовать свои силы и самостоятельно заполнить элементы этой блок-схемы согласно задаче.

**[пауза 2:30 мин]**

****

Итак, давайте посмотрим что у вас получилось. В качестве действия мы используем вывод очередного числа на экран. Для этого мы задействуем переменную-счётчик i, которая изначально равна 1, и с каждой итерацией она увеличивается на 1.

Переменная i как раз является вспомогательной переменной. В качестве условия мы используем неравенство, которое говорит нам, до каких пор должна увеличиваться переменная-счётчик i: она будет расти от 1 и до N включительно. Как только переменная i превысит значение N, цикл остановится.

**[слайд «Оживим алгоритм»]**

Алгоритм решения задачи готов, теперь можно его оживить на языке C#.

**Создание проекта**

**[Скрин-каст]**

Создадим каталог Task5 и создадим в нём новый проект. Перейдём в файл с исходным кодом.

**Написание кода**

Создадим переменную n, проинициализируем её некоторым натуральным числом, например 10. Затем создадим переменную-счётчик i.

****

Далее, согласно нашей блок-схеме, оформим тело и условие цикла. Для задания тела цикла можно использовать оператор while (В переводе на русский звучит как «пока»). Он доступен в большинстве языках программирования.

После ключевого слова while в круглых скобках мы указываем условие i <= n. Затем в фигурных скобках мы указываем тело цикла, то есть набор инструкций (одна или более) которые будут повторяться на каждой итерации. Выведем на экран очередное число с помощью метода Write.



Не забываем модифицировать переменную-счетчик i, иначе цикл будет работать бесконечно. Попробуем запустить наш код. На экраны мы видим набор цифр 12345678910, но они написаны слитно без каких-либо разделителей.

|  |
| --- |
| * Друзья, как вы думаете, а почему так вышло? |

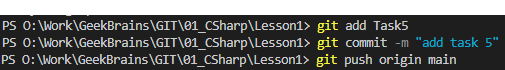
Все дело в методе Console.Write(i). Напомню, он выводит значение переменной и не добавляет каких-либо разделителей после неё. Чтобы разделить значения пробелами или запятыми, это нужно сделать в теле цикла.

Например, после вывода очередного числа, можно вывести на экран пробел с помощью того же метода Console.Write(‘ ’). Попробуем запустить наш код. Как и ожидалось, к выводу чисел добавился разделитель – пробел. И теперь вывод стал более нагляден.



Отмечу, что входной параметр N можно поменять. Например, задать его равным 100, 1000, 10000 и так далее. Прелесть использования циклов в том, что алгоритм и объём кода при этом абсолютно не изменится, но требуемый результат (вывод необходимого количества чисел) будет получен.

**Работа с GIT**

****

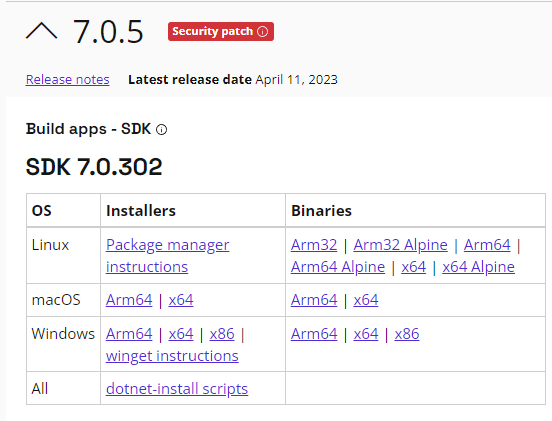
Сохраним наши наработки в репозитории GIT. Проиндексируем файлы и каталоги, сформируем коммит с указанием комментария и загрузим его на удаленный репозиторий.

# 7. Процесс установки и настройки окружения

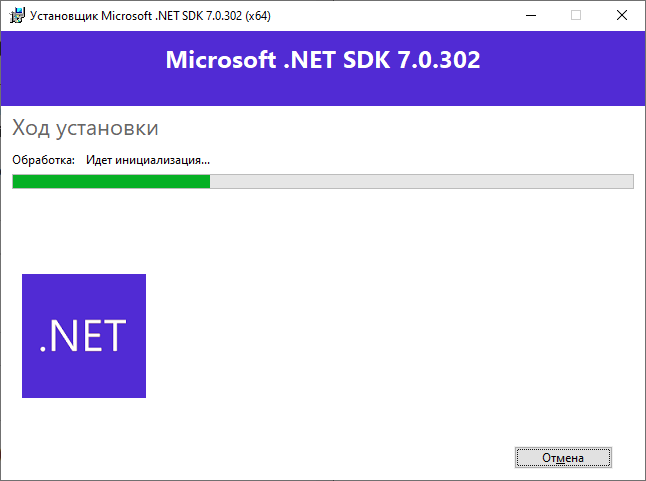
На этом основная часть сегодняшнего занятия заканчивается. В заключение для тех, у кого возникнут сложности, предлагаю посмотреть **более подробный** процесс установки платформы .NET, а также настройку текстового редактора VS Code.



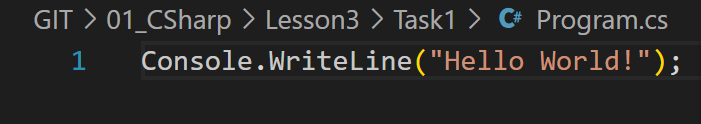
Перейдем на страницу загрузки установщика платформы .NET. На текущий момент актуальная версия платформы – седьмая. Перейдём по ссылке «All .NET 7.0 downloads», чтобы отобразить все доступные загрузки.



Как видно платформа .NET доступна для Linux, Mac и Windows. Выберем установщик для своей ОС, если у вас Windows, то в большинстве случаев используется архитектура x64. Если у вас Mac, то уточните архитектуру вашего процессора, поскольку начиная с 2020 года Apple начала переход с архитектуры Intel x64 на ARM64.

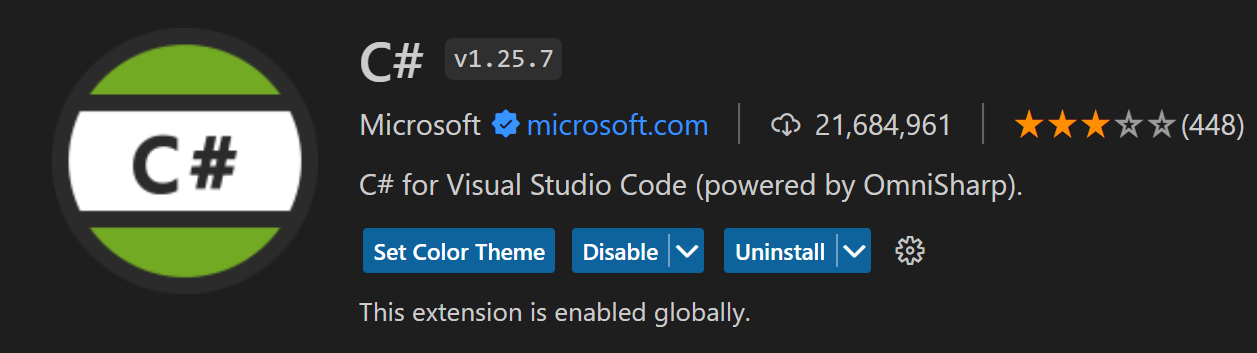


Итак, моя ОС это Windows с архитектурой x64. Запустим установщик и подтвердим установку. Установка успешно выполнена.

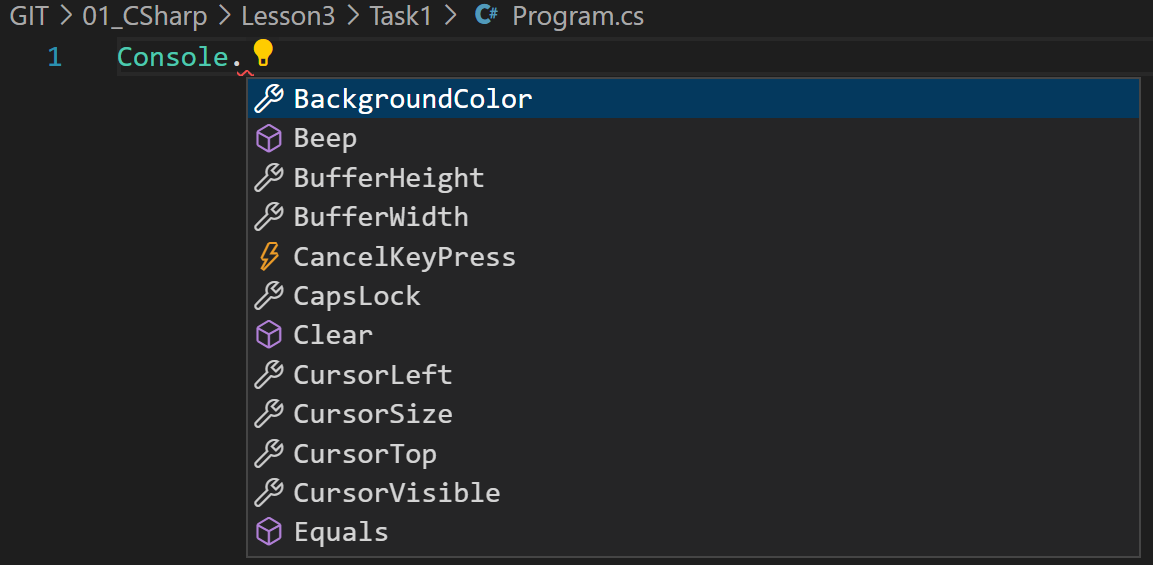


Теперь мы можем воспользоваться функционалом платформы .NET из текстового редактора Visual Studio Code. Перейдем в редактор, откроем какой-нибудь каталог в котором будем создавать проекты. Командой dotnet new console создадим консольное приложение и перейдём в файл с исходным кодом, который создан по умолчанию. Командой dotnet run мы может скомпилировать исходный код и запустить его.

Друзья, обратите внимание что некоторая подсветка синтаксиса уже присутствует, однако при попытке обратиться к модулю Console, чтобы посмотреть его доступный функционал, никаких подсказок на экране не выводится. Дело в том, что за дополнительную подсветку синтаксиса, вывод подсказок и подчеркивание ошибок отвечает соответствующее расширение.



Давайте перейдем во вкладку Расширения, введем в поиске C# и установим первое предложенное в списке. После установки может потребоваться перезапуск VS Code, чтобы расширение активировалось.



Попробуем обратиться к модулю Console, теперь у него добавилась подсветка и появились всплывающие подсказки для выбора его свойств и методов.

# 8. Итоги

Ну что друзья, на этом первая лекция подошла к концу. Давайте подведем итоги сегодняшнего занятия.

Напомню, мы рассмотрели важные характеристики языка C#, которые позволили выбрать его в качестве основного инструмента на текущем курсе. Это его статическая типизация и возможность создавать простые программы, не перегружая их сложным синтаксисом.

Затем мы настроили и проверили работоспособность окружения для разработки на C# - платформу .NET и редактор VS Code.

После чего решили блок задач на работу с переменными, арифметикой, условиями и циклами.

На этом друзья я с вами прощаюсь, благодарю за внимание, увидимся с вами на семинарах и на лекции №2.