Темы урока

[**Выбор фреймворка и версии для проекта**](#_1nbaqgkx9q55) **1**

[**Полезные расширения для Visual Studio**](#_2xzzu6vkx4ba) **1**

[**Запуск приложений .NET Core в консоли**](#_tcg4x2e83y1f) **2**

[Виды развертываний (deployments) приложений .NET Core](#_jprkmejl0gf7) 2

[Запуск в консоли](#_jgv09df01plz) 3

[**Синтаксис C#**](#_xnth0g8povae) **4**

[**Словарь C#**](#_znkic6381v5) **4**

[**Переменные, буквальные значения**](#_y4xdls47ty3h) **4**

[**Строки, символы**](#_83y1n421vqtu) **5**

[**Целые числа**](#_6pzi3spl5blz) **5**

[**Числа с плавающей точкой**](#_7fnxmfoavk7p) **5**

[**Распознавание численных значений**](#_2nuprkqablko) **7**

[**Булевы величины**](#_zh8i05mo65p2) **7**

[**Домашнее задание**](#_h5sagbglk9hz) **7**

# Выбор фреймворка и версии для проекта

Чтобы изменить тип или версию фреймворка необходимо щелкнуть правой кнопкой мышки по проекту в Solution Explorer и выбрать пункт меню Properties (в самом низу).

В окошке выбрать слева таб Application, и выбрать нужный фреймворк в пункте Target Framework.

# Полезные расширения для Visual Studio

Существует огромное количество расширений для IDE Visual Studio.

Они доступны по URL <https://marketplace.visualstudio.com> или через меню Tools > Extensions and Updates:

* GitHub Extension for Visual Studio
* Markdown Editor
* File Icons
* GhostDoc
* BetterStartPage

# Запуск приложений .NET Core в консоли

## Виды развертываний (deployments) приложений .NET Core

* По-умолчанию из Visual Studio приложение собирается для “***развертывания, зависящего от платформы***” (в англоязычной документации еще говорят **Portable** или **FDD**: Framework-Dependent Deployment).
  + При таком виде сборки, в папке назначения мы получаем только файлы нашего приложения и внешних зависимостей (сторонних библиотек).
  + Для запуска нашего приложения на целевом компьютере должен быть установлен .NET Core Runtime соответствующей версии (**Runtime – для запуска! не путать с .NET Core SDK – для разработки**).
  + Проверить, какие компоненты установлены в текущей системе можно с помощью команд:
    - **dotnet --list-runtimes**
    - **dotnet --list-sdks**
* Также бывает “***автономное развертывание***” (в англоязычной документации его называют **Standalone** или **SCD**: Self-Contained Deployment). При такой сборке все компоненты для выполнения и библиотеки .NET Core, и сторонние библиотеки, то есть абсолютно все зависимости, поставляются вместе с самим приложением (чаще всего в одной папке).
  + Подготовить приложение к автономному развёртыванию можно, выполнив команду **dotnet publish -r <target-platform>**, находясь в папке проекта, например:
    - **dotnet publish -r win10-x64**
  + Можно также указать, чтобы целевой исполняемый файл был единственным (все необходимые файлы файлы, от которых зависит приложение будут упакованы в единственный файл), используя параметр **-p:PublishSingleFile=true**, например:
    - **dotnet publish -r win10-x64 -p:PublishSingleFile=true**

## 

## Запуск в консоли

Запустим наше приложение в консоли как отдельный файл сборки, а не через окружение Visual Studio. Поскольку у нас portable-сборка, наше приложение выглядит как DLL-библиотека, хотя, и является исполняемым кодом с точкой входа. Относительно солюшна файл располагается по следующему пути: “***папка\_проекта\bin\Debug\netcoreappX.X***” Запустить его можно с помощью команды dotnet :

**dotnet** имя\_сборки.dll

Можно перед этим очистить экран командой **cls**.

\* Начиная с версии .NET Core 3.0 также автоматически собирается и исполняемый файл для платформы, на которой происходит сборка (т.е. для Windows - привычный exe-файл)

# 

# Синтаксис C#

* Заявления (statements) могут состоять из одного или нескольких выражений (expressions) или переменных (variables).
* Блоки (blocks) - несколько выражений или блоков, объединенных круглыми скобками.
* Комментарии (comments)
  + // однострочные (для выделенного текста можно использовать:
    - Ctrl + K + C (закомментировать)
    - Ctrl + K + U (раскомментировать)
  + /\*  
     многострочные  
     комментарии  
    \*/

# Словарь C#

* Ключевые слова (keywords) 79+25=104
  + Предопределенные зарезервированные ключевые слова 79, например:  
     using, namespace, class, static, int, string, double, bool, var, if, switch, break, и т.д.
  + Ещё 25 ключевых слов имеют разное значение в зависимости от контекста, например:  
     add, get, set, remove, global, where, when, yield
* Тулинг
  + Notepad vs MS Word
  + Notepad vs MS Visual Studio
* Методы как глаголы, производящие действие или вычисление
* Типы данных и переменные как существительные
* Сопоставление коротких имен и типов .NET
  + int: System.Int32
  + bool: System.Boolean
  + string: System.String

# Переменные, буквальные значения

* Определение переменных
* Принципы именования переменных, пара слов о Style Guideline:
  + camelCase (рекомендуется в большинстве случаев)
  + PascalCase
  + Хорошая статья к слову о c# style guides: [C#: требования и рекомендации по написанию кода](https://habr.com/ru/post/26077/).
* Пример использования буквальных значений
* Пример с вынесением их в переменные

# Строки, символы

* Символы System.Char / char
* Строки System.String / string
* Простое объявление переменных
* Выражения при определении переменных
* Пример определения строк
* Самостоятельная работа на объявление переменных и использование буквальных значений
  + объявить переменную каждого типа
    - без инициализации,
    - с инициализацией,
    - вывести на экран результаты через Console.WriteLine()

# Целые числа

* 1-, 2-, 4- и 8-байтовые числа со знаком и без
* Простое объявление переменных
* Выражения при определении переменных
* Свойства MaxValue и MinValue для целочисленных типов
* Возможность записи буквальных значенией с разделением разрядов подчёркиванием: (10\_000\_000 = 10000000) для удобства зрительного восприятия.
* Самостоятельная работа на объявление переменных и использование буквальных значений
  + объявить переменную каждого типа
    - byte, sbyte, short, ushort, int, uint, long, ulong
    - вывести максимальное и минимальное значение для каждого типа данных на экран.

# Числа с плавающей точкой

* Вообще без деталей, просто перечислить float, double и decimal типы и их диапазоны.
* Объяснить что значит запись числа с плавающей точкой в формате XE±Y:  
  E в данном случае означает “×10 в степени, например:
  + 2.5E-3 = 2.5 × 10^-3 = 0.0025
  + 4.87E8 = 4.87 × 10^8 = 487\_000\_000
* Для float и double и свойства:
  + MinValue, MaxValue
  + Epsilon – наименьшее положительное значение больше нуля – разное для float и double
  + NaN – Представляет нечисловое значение. Например, когда нужно сравнить с результатом деления на 0:

|  |
| --- |
| float zero = 0.0f; Console.WriteLine("{0} / {1} = {2}", zero, zero, zero/zero); // 0 / 0 = NaN |

* + Функция InNan() используется когда нужно проверить на равенство значения константе NaN:

|  |
| --- |
| Single zero = 0;  if ((0 / zero) == Single.NaN) // This condition will return false.  {  Console.WriteLine("0 / 0 can be tested with Single.NaN.");  }  else  {  Console.WriteLine("0 / 0 can be tested with Single.IsNan().");  } |

* + Функция NegativeInfinity – эта константа возвращается в том случае, если результат операции имеет меньше, чем MinValue.  
    Функция IsNegativeInfinity используется для проверки на равенство минус бесконечности.
  + PositiveInfinity – эта константа возвращается в том случае, если результат операции больше, чем MaxValue.  
    Функция IsPositiveInfinity используется для проверки на равенство положительной бесконечности.
* Для decimal:
  + MinValue, MaxValue
  + One, MinusOne, Zero
* Самостоятельная работа на объявление переменных и использование буквальных значений
  + объявить переменную каждого типа
  + вывести предельные значения через Console.WriteLine()
  + вывести значения констант через Console.WriteLine()

# 

# Распознавание численных значений

* int.Parse(...) и т.п. - будет нужно для домашнего задания.
* Самостоятельная работа по парсингу числовых значений введенных с помощью Console.ReadLine().

# Булевы величины

* Коротко о логическом OR и AND для булевых величин
* Коротко о булевых операторах
* Объяснить почему не обязательно досчитывать до конца:
  + true || someBooleanValue (всегда true)
  + false && someBooleanValue (всегда false);
* Самостоятельная работа на составление условий для разных случаев.

# Домашнее задание

Вариант "**базовый**":

* Написать приложение, запрашивающее у пользователя поочерёдно 2 числа числа, а затем выводящее сумму, разницу и произведение этих чисел в консоль.

Вариант “**посложнее**”, если вы знакомы с условной конструкцией if…else:

* Написать приложение-калькулятор, запрашивающее у пользователя поочерёдно 2 числа числа, а также один из шести типов операций:
  + - сложение
  + - вычитание
  + - умножение
  + - деление
  + - остаток от деления
  + - возведение в степень
* а затем выводящее результат вычисления в консоль.

*Начиная с этого урока присылайте, пожалуйста, не просто ссылку на репозиторий, а ссылку на последний коммит с кодом домашнего задания. Это позволит в случае дополнительных вопросов и доделок удобно смотреть разницу между коммитами – что именно изменилось.*