Ответы на вопросы, **LAB\_01**

1) **.NET Framework** — программная платформа, выпущенная компанией [Microsoft](https://ru.wikipedia.org/wiki/Microsoft) в [2002 году](https://ru.wikipedia.org/wiki/2002_%D0%B3%D0%BE%D0%B4). Основой платформы является общеязыковая среда исполнения [Common Language Runtime (CLR)](https://ru.wikipedia.org/wiki/Common_Language_Runtime), которая подходит для различных языков программирования. Функциональные возможности CLR доступны в любых языках программирования, использующих эту среду. В настоящее время .NET Framework развивается в виде [.NET](https://ru.wikipedia.org/wiki/.NET). Он включает в себя большую [библиотеку классов](https://translated.turbopages.org/proxy_u/en-ru.ru.14a6f6c4-63151a4e-e38dccc4-74722d776562/https/en.wikipedia.org/wiki/Class_library), называемую [Framework Class Library](https://translated.turbopages.org/proxy_u/en-ru.ru.14a6f6c4-63151a4e-e38dccc4-74722d776562/https/en.wikipedia.org/wiki/Framework_Class_Library) (FCL), и обеспечивает [языковую совместимость](https://translated.turbopages.org/proxy_u/en-ru.ru.14a6f6c4-63151a4e-e38dccc4-74722d776562/https/en.wikipedia.org/wiki/Language_interoperability) (каждый язык может использовать код, написанный на других языках) на нескольких [языках программирования](https://translated.turbopages.org/proxy_u/en-ru.ru.14a6f6c4-63151a4e-e38dccc4-74722d776562/https/en.wikipedia.org/wiki/Programming_language). Программы, написанные для .NET Framework, выполняются в [программной](https://translated.turbopages.org/proxy_u/en-ru.ru.14a6f6c4-63151a4e-e38dccc4-74722d776562/https/en.wikipedia.org/wiki/Software) среде (в отличие от [аппаратной](https://translated.turbopages.org/proxy_u/en-ru.ru.14a6f6c4-63151a4e-e38dccc4-74722d776562/https/en.wikipedia.org/wiki/Computer_hardware) среды), называемой [Common Language Runtime](https://translated.turbopages.org/proxy_u/en-ru.ru.14a6f6c4-63151a4e-e38dccc4-74722d776562/https/en.wikipedia.org/wiki/Common_Language_Runtime) (CLR). Среда CLR - это [виртуальная машина приложения](https://translated.turbopages.org/proxy_u/en-ru.ru.14a6f6c4-63151a4e-e38dccc4-74722d776562/https/en.wikipedia.org/wiki/Process_virtual_machine), которая предоставляет такие службы, как безопасность, [управление памятью](https://translated.turbopages.org/proxy_u/en-ru.ru.14a6f6c4-63151a4e-e38dccc4-74722d776562/https/en.wikipedia.org/wiki/Memory_management) и [обработка исключений](https://translated.turbopages.org/proxy_u/en-ru.ru.14a6f6c4-63151a4e-e38dccc4-74722d776562/https/en.wikipedia.org/wiki/Exception_handling). Таким образом, компьютерный код, написанный с использованием .NET Framework, называется "[управляемым кодом](https://translated.turbopages.org/proxy_u/en-ru.ru.14a6f6c4-63151a4e-e38dccc4-74722d776562/https/en.wikipedia.org/wiki/Managed_code)". FCL и CLR вместе составляют .NET Framework.

2) .NET предоставляет среду выполнения (среду CLR), которая выполняет код и предлагает службы, облегчающие процесс разработки.

Base Class Library, или так называемая .NET FCL (англ. Framework Class Library), сокращённо BCL — стандартная библиотека классов платформы «.NET Framework».

Интерфейс командной строки (CLI)-это средство взаимодействия с компьютерной программой, при котором пользователь (или клиент) выдает программе команды в виде последовательных строк текста (командных строк). Программа, которая обрабатывает интерфейс, называется интерпретатором командной строки или процессором командной строки.

IL (Intermediate Language) — код на специальном языке, напоминающим ассемблер, но написанном для .NET. В него преобразуется код из других языков верхнего уровня (c#, VisualBasic).

3) Just-in-Time, компиляция «точно в нужное время»), динамическая компиляция (англ. dynamic translation) — технология увеличения производительности программных систем, использующих байт-код, путём компиляции байт-кода в машинный код или в другой формат непосредственно во время работы программы.

4) CTS (Common Type System) предназначена для выполнения следующих задач:

Создание платформы для выполнения на разных языках.

Предоставление объектно-ориентированной модели для поддержки реализации различных языков в реализации .NET.

Определение набора правил, которым должны следовать все языки при работе с типами.

Предоставление библиотеки, которая содержит базовые типы-примитивы, используемые при разработке приложений (например, Boolean, Byte, Char и т. д.).

Т.е. для совместимости с разными ЯП

5)System.Object — [класс](https://dic.academic.ru/dic.nsf/ruwiki/41093) в [C#](https://dic.academic.ru/dic.nsf/ruwiki/718619), который является [базовым](https://dic.academic.ru/dic.nsf/ruwiki/797671) для всех типов данных (как структурных, так и ссылочных). Это класс определяет общее полиморфическое поведение для всех типов в мире [.NET](https://dic.academic.ru/dic.nsf/ruwiki/91404). System.Object Не обязательно указывать в программе, как базовый — это подразумевается само собой. Однако ничто не мешает это сделать

Пример:

public class Helloword : System.Object

{

public Helloword()

{

Console.Writeline("Hello World!");

}

}

6) **В** **mscorlib**.**dll** файл будет присутствовать в пути C:/ WINNT/ Microsoft.NET/ Фреймворк/ (номер версии)/ **mscorlib**.**dll**. Это простое приложение определяет тип под названием App, который имеет один статический открытый метод Main. Внутри Main **находится** консоль справочной системы и код Microsoft intermediate language (MSIL).

7) **Сбо́рка** (англ. build) (предметное имя существительное) — подготовленный для использования информационный продукт. Чаще всего **сборка** — исполняемый файл — двоичный файл, содержащий исполняемый код (машинные инструкции) программы или библиотеки. **Сборка** в терминологии .**NET** (англ. assembly) — двоичный файл (exe или dll), содержащий номер версии информационного продукта (ИП), метаданные и инструкции для виртуальной машины .**NET**.

8)**Существует** **два** **вида** **сборок:** **однофайловые** **сборки; многофайловые** **сборки.** Сборка, которая состоит из одного единого модуля (\*.dll или \*.exe) называется однофайловой. В однофайловых сборках все необходимые CIL -инструкции, метаданные и манифесты размещаются в одном, четко определенном пакете. Сборка, которая состоит из многих файлов двоичного кода .NET, называется многофайловой. Каждый из этих файлов называется модулем.

9) **Манифест** сборки (**assembly** **manifest**) - это внутренняя часть сборки, которая позволяет ей быть самоописанной. **Assembly** **manifest** позволяет идентифицировать сборку, указывает файлы, которые включаются в реализацию сборки, описывает типы и ресурсы, используемые в сборке, указывает зависимости от других сборок, а также набор прав доступа, которые необходимы сборке для корректной работы.

10) Global Assembly Cache (англ. **GAC** — Глобальный Кэш Сборок) — в инфраструктуре .NET — подсистема, хранящая сборки CLI (CLI assembly) в централизованном репозитории. Система разработана для борьбы с проблемами разделяемых библиотек.

11) **Управляемый код** компилируется не в машинный код, а на промежуточный язык, который интерпретируется и выполняется некоторой службой на компьютере.

**Неуправляемый код** компилируется в машинный код и, следовательно, выполняется непосредственно операционной системой. Поэтому он способен делать разрушительные / мощные вещи, которых не делает управляемый код.

12) **Метод** **Main** — это точка входа приложения C#. (Библиотекам и службам точка входа в виде **метода** **Main** не требуется.) Когда приложение запускается, первым вызывается именно **метод** **Main**. В программе на C# может существовать только одна точка входа.

13) Директива using позволяет использовать типы, определенные в пространстве имен, без указания полного пространства имен этого типа. В базовой форме директива using импортирует все типы из одного пространства имен, как показано в следующем примере:

using System.Text;

14) **Пространство** **имен** — это совокупность классов, логически **связанных** **между** **собой**. **Между** **сборками** **и** **пространствами** **имен** нет прямого соответствия: в **сборке** может хранится несколько **пространств** **имен**, а разные классы одного **пространства** **имен** могут быть определены в разных **сборках**.

15) **Примитивный** **тип** - это несоставной **тип** **данных**, тоесть такие **типы** как int, double, boolean, single, byte и т.д., занимающие в памяти строго определенное количество байт. String, Object. enum уже не **примитивные**, а составные **типы** **данных**, тоесть состоящие из ряда **примитивных**.

16-17) В **C#** существуют две разновидности **типов**: **ссылочные** **типы** и **типы** значений. В переменных **ссылочных** **типов** хранятся **ссылки** на их данные (объекты), а переменные **типа** значений содержат свои данные непосредственно. Две переменные **ссылочного** **типа** могут ссылаться на один и тот же объект, поэтому операции над одной переменной могут затрагивать объект, на который ссылается другая переменная.

18) Главное **отличие** - это где храниться **значение** переменной. Для **ссылочных** **типов** - это куча, для **значимых** - это стек.

19) **Упаковка** (boxing) предполагает преобразование объекта **значимого** **типа** (например, **типа** int) к **типу** object. При **упаковке** общеязыковая среда. CLR обертывает **значение** в объект **типа**. System.Object и сохраняет его в управляемой куче (хипе). **Распаковка** (unboxing), наоборот, предполагает преобразование объекта **типа** object к **значимому** **типу**.

20) Единственное **различие**, вероятно, **заключается** в том, чтобы указать диапазон значений, поддерживаемых **int** **и** **System**::**Int32**. **System**::**Int32** делает 32-разрядность более понятной для тех, кто читает код. Следует использовать **int** там, где просто необходимо "целое число", и использовать **System**::**Int32**, где важен размер (криптографический код, структуры), чтобы читатели не обращали на это внимания, сохраняя то же самое.

21) Этот тип является статическим, но объект типа dynamic обходит проверку статического типа. В большинстве случаев он работает как тип object. Во время компиляции предполагается, что элемент, типизированный как dynamic, поддерживает любые операции.

22) Резюме **Основное** **различие** **между** **var** **и** динамическими ключевыми словами **состоит** в том, что время привязки отличается: **var** - раннее связывание, динамическое связывание будет выполняться во время выполнения. **var** на самом деле является синтаксическим сахаром, выдаваемым нам компилятором.

Еще одно отличие заключается в том, что когда мы имеем дело с ключевым словом dynamic, мы не получим никаких ошибок во время компиляции. Например, в приведенном ниже коде, если мы скомпилируем, мы не получим никаких ошибок, но во время выполнения мы столкнемся с исключением.

1. динамический y = "строковое значение";
2. **int** length2 = y.длина;

23) **Неявно** **типизированные** **переменные-это** **те** **переменные, которые** **объявляются** **без** **явного** **указания** **типа**

24)Nullable value type (**тип** **значения, допускающий** **null**) – это тип, который позволяет представить не только все значения своего нижележащего типа, но и значение null.

25) Строковые литералы представляют строки. Строки заключаются в двойные кавычки:

Console.WriteLine(“Привет”);

Строка: string str\_1 = "Hello";

**Операции со строками:**

Объединение строк, сравнение строк, поиск в строке, разделение строк, обрезка строки, вставка, удаление строк, замена, смена регистра, сцепление, поиск подстроки, разбиение строки на слова…

26)string s1 = "Hello, World!"; ++ ввести с клавиатуры (Console.ReadLine())…

27)

* **Compare**: сравнивает две строки с учетом текущей культуры (локали) пользователя
* **CompareOrdinal**: сравнивает две строки без учета локали
* **Contains**: определяет, содержится ли подстрока в строке
* **Concat**: соединяет строки
* **CopyTo**: копирует часть строки, начиная с определенного индекса в массив
* **EndsWith**: определяет, совпадает ли конец строки с подстрокой
* **Format**: форматирует строку
* **IndexOf**: находит индекс первого вхождения символа или подстроки в строке
* **Insert**: вставляет в строку подстроку
* **Join**: соединяет элементы массива строк
* **LastIndexOf**: находит индекс последнего вхождения символа или подстроки в строке
* **Replace**: замещает в строке символ или подстроку другим символом или подстрокой
* **Split**: разделяет одну строку на массив строк
* **Substring**: извлекает из строки подстроку, начиная с указанной позиции
* **ToLower**: переводит все символы строки в нижний регистр
* **ToUpper**: переводит все символы строки в верхний регистр
* **Trim**: удаляет начальные и конечные пробелы из строки

28) **NULL** - что значение поля не определено, а "" - это **пусто**

**29)** Статический метод String.Compare позволяет тщательно сравнивать две строки. Этот метод учитывает язык и региональные параметры. Эту функцию можно использовать для сравнения двух строк или подстрок двух строк. **CompareOrdinal**: сравнивает две строки без учета локали.

30) **String** создает новый объект **String**. Между тем. **StringBuilder** содержит в себе массив символов, этот массив будет автоматически заменен на больший массив, если это необходимо, и скопирует символы из старого массива. Если вам необходимо объединить несколько строк много раз, вы должны использовать. **StringBuilder**, это помогает повысить эффективность программы.

31)Явное преобразование для конкретного типа данных. Пример:

Double dMin = Convert.ToDouble(Console.ReadLine());

32) Консольный вывод:

string hello = "Привет мир";

Console.WriteLine(hello);

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

string name = "Tom";

int age = 34;

double height = 1.7;

Console.WriteLine($"Имя: {name}  Возраст: {age}  Рост: {height}м");

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

string name = "Tom";

int age = 34;

double height = 1.7;

Console.WriteLine("Имя: {0}  Возраст: {2}  Рост: {1}м", name, height, age);

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Консольный ввод:

string? name = Console.ReadLine();

int age = Convert.ToInt32(Console.ReadLine()); //преобразование типа

33) //Двумерный массив (матрица)

int[,] numbers = { { 1, 2, 3 }, { 4, 5, 6 } };

int[,] nums1;

int[,] nums2 = new int[2, 3];

int[,] nums3 = new int[2, 3] { { 0, 1, 2 }, { 3, 4, 5 } };

int[,] nums4 = new int[,] { { 0, 1, 2 }, { 3, 4, 5 } };

int[,] nums5 = new [,]{ { 0, 1, 2 }, { 3, 4, 5 } };

int[,] nums6 = { { 0, 1, 2 }, { 3, 4, 5 } };

//Одномерный массив

string[] people = { "Tom", "Sam", "Bob" };

34) в C# можно также создавать специальный тип двумерного массива, называемый ***ступенчатым массивом***. Ступенчатый массив представляет собой массив массивов, в котором длина каждого массива может быть разной.  Для создания указать вручную длину каждой строки.

double[][] myArr = new double[3][];

myArr[0] = new double[2];

myArr[1] = new double[3];

myArr[2] = new double[4];

35) Оператор цикла foreach предназначен для перебора элементов коллекции или массива. Общая форма оператора foreach следующая

foreach(type *identifier* in *container*)

{

// операторы

// ...

}

где

* *type* – тип переменной с именем *identifier*;
* *identifier* – имя переменной, которая используется в качестве итератора. Переменная *identifier* приобретает значение следующего элемента цикла на каждом шаге выполнения цикла foreach. Тип переменной *identifier* должен совпадать с типом массива или коллекции *container*. Связь между *identifier* и *container* реализуется с помощью союза in;
* *container*– имя коллекции или массива, который просматривается.

Оператор цикла foreach работает следующим образом. При вхождении в цикл, переменной *identifier* присваивается первый элемент массива (коллекции) *container*. На каждом следующем шаге итерации выбирается следующий элемент из *container*, который сохраняется в переменной *identifier*. Цикл завершается, когда будут пересмотрены все элементы массива (коллекции) *container*.

36)**Слово** **Кортеж** **означает** **“структура** **данных, состоящая** **из** **нескольких** **частей”**. Таким образом, кортеж-это структура данных, которая дает вам самый простой способ представить набор данных, который имеет несколько значений, которые могут быть/не быть связаны друг с другом.

//Кортеж из 5 элементов с типами int, string, char, string, ulong

(int, string, char, string, ulong) kortezh = (19, "Hi", 'F', "world", 9999999);

37) **Локальными** **функциями** в **C#** называют **функции**, которые определены внутри других методов. Как правило, **локальная** **функция** содержит действия, применяемые лишь в рамках ее метода.

38) В C# допускается указывать, будет ли в коде сгенерировано исключение при переполнении, с помощью ключевых слов checked и unchecked. Так, если требуется указать, что выражение будет проверяться на переполнение, следует использовать ключевое слово checked, а если требуется проигнорировать переполнение — ключевое слово unchecked. В последнем случае результат усекается, чтобы не выйти за пределы диапазона представления чисел для целевого типа выражения.

39) По умолчанию – unchecked

40) **Ключевое** **слово** **fixed** может **применяться** к массивам типов значений, к строкам и к полям управляемого класса, имеющим тип значения. Дескрипторы сборщика мусора - это инструмент, позволяющий ссылаться на управляемый объект, находящийся в динамической памяти сборщика мусора, с помощью неизменяемого значения дескриптора (даже если адрес объекта изменится), который можно передавать даже неуправляемому коду. Оператор fixed задает указатель на управляемую переменную и "закрепляет" эту переменную во время выполнения оператора.