Pentalog

.NET Framework and Visual Studio

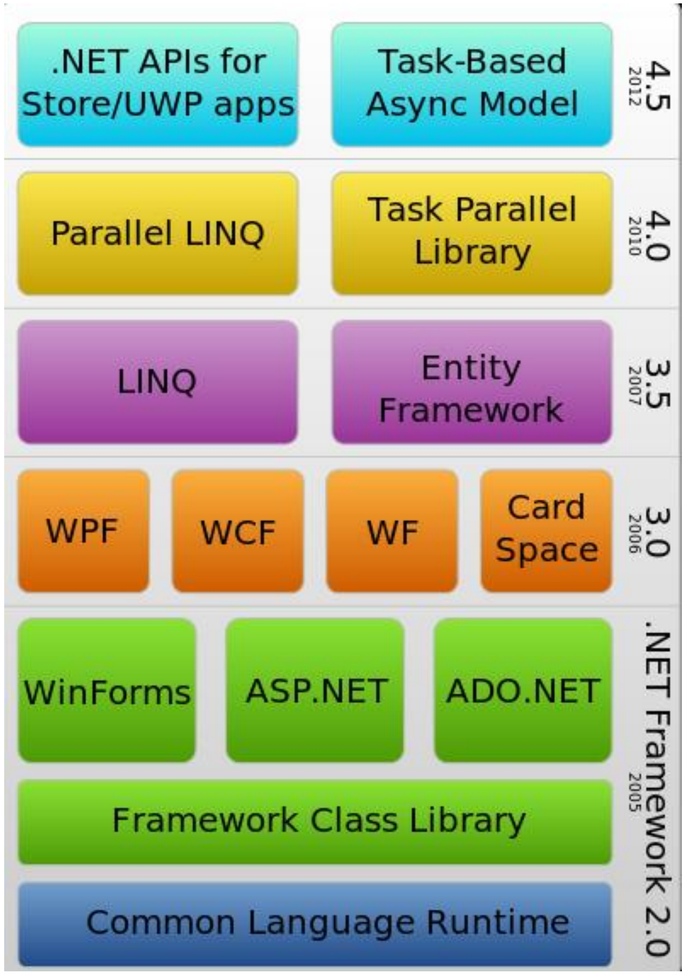
DATA ANALYSIS
UX/UI
FRONT-END
TECHNOLOGY
NEAR/OFFSHORE
AGILITY
BACK-END
SPRINT
KANBAN
CAMPAIGNS
GROWTH HACKING
SCRUM
BACKLOG
DEVOPS
DESIGN
SEO
CONTINUOUS INTEGRATION
MOBILE
QA
AUTOMATION
RESPONSIVE
UNIT TESTING

What is the .NET Framework?

- The .NET Framework is a software framework developed by Microsoft in order to help developers with writing applications
- It supports multiple types of applications: desktop, web and mobile
- It supports several programming languages, the main one being C#
- It includes automatic memory management (garbage collection)
- It can interoperate with unmanaged code
- The main integrated development environment used for developing .NET apps is Visual Studio

.NET Framework components

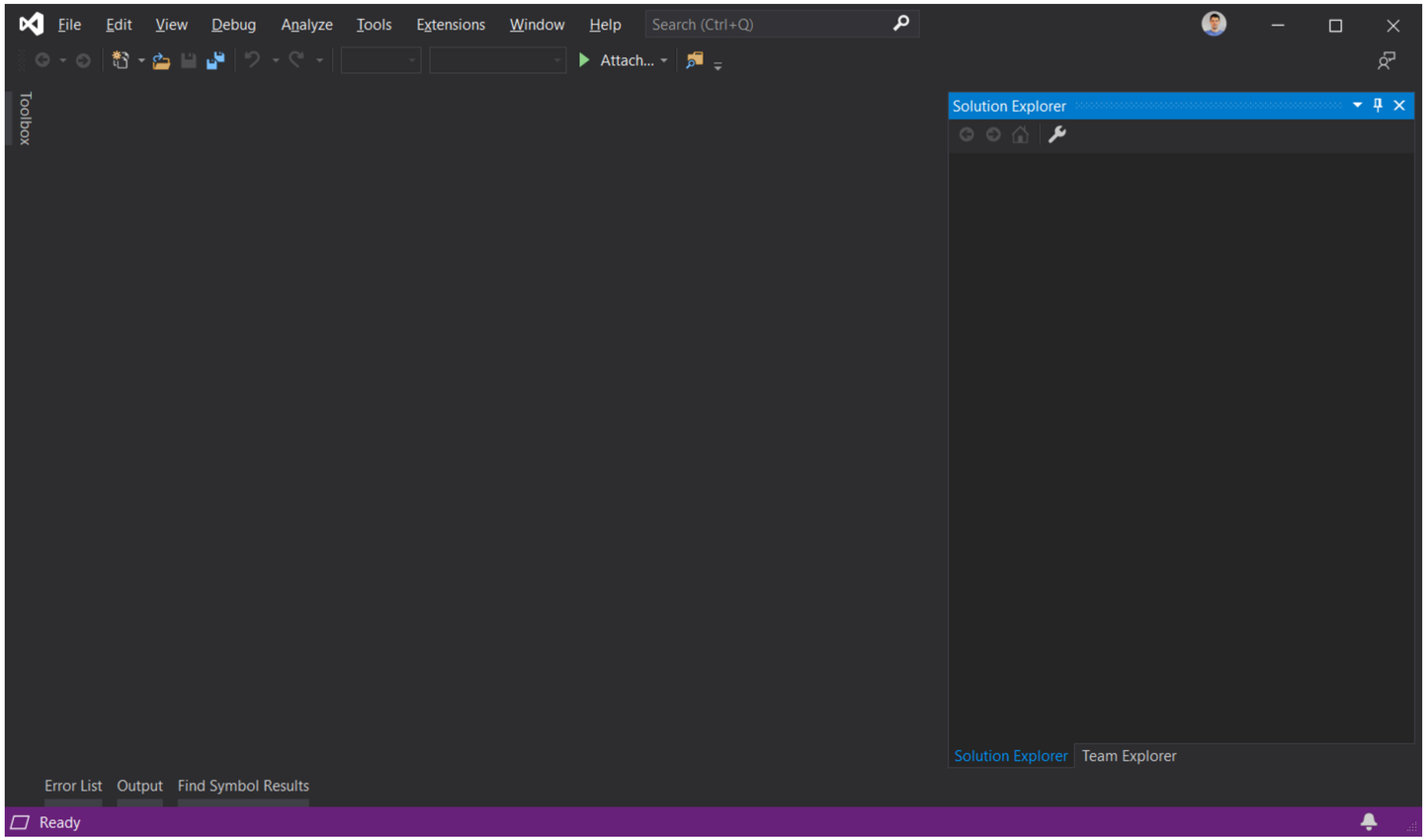
- Applications written for the .NET Framework run in a software environment called the Common Language Runtime
- .NET also includes a class library called Framework Class Library, which is a collection of reusable classes, interfaces and types
- It also contains several other components, used for different tasks, like data access, creating GUIs, communicating with web services and manipulating data



What is Visual Studio?

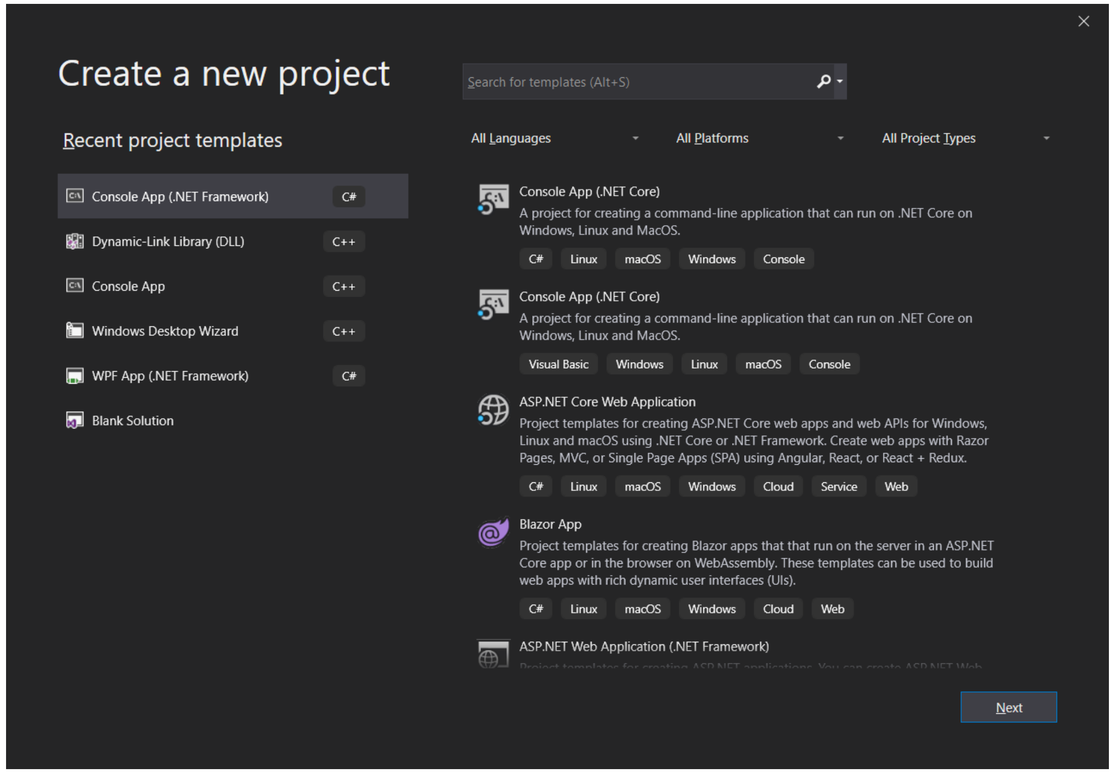
- Visual Studio is an IDE developed by Microsoft, and is the main tool for developers wanting to write Windows applications, web sites, web services and mobile apps
- It can be used to write and debug both .NET applications as well as native code (C/C++)
- It supports multiple types of projects, programming languages and software frameworks

User Interface



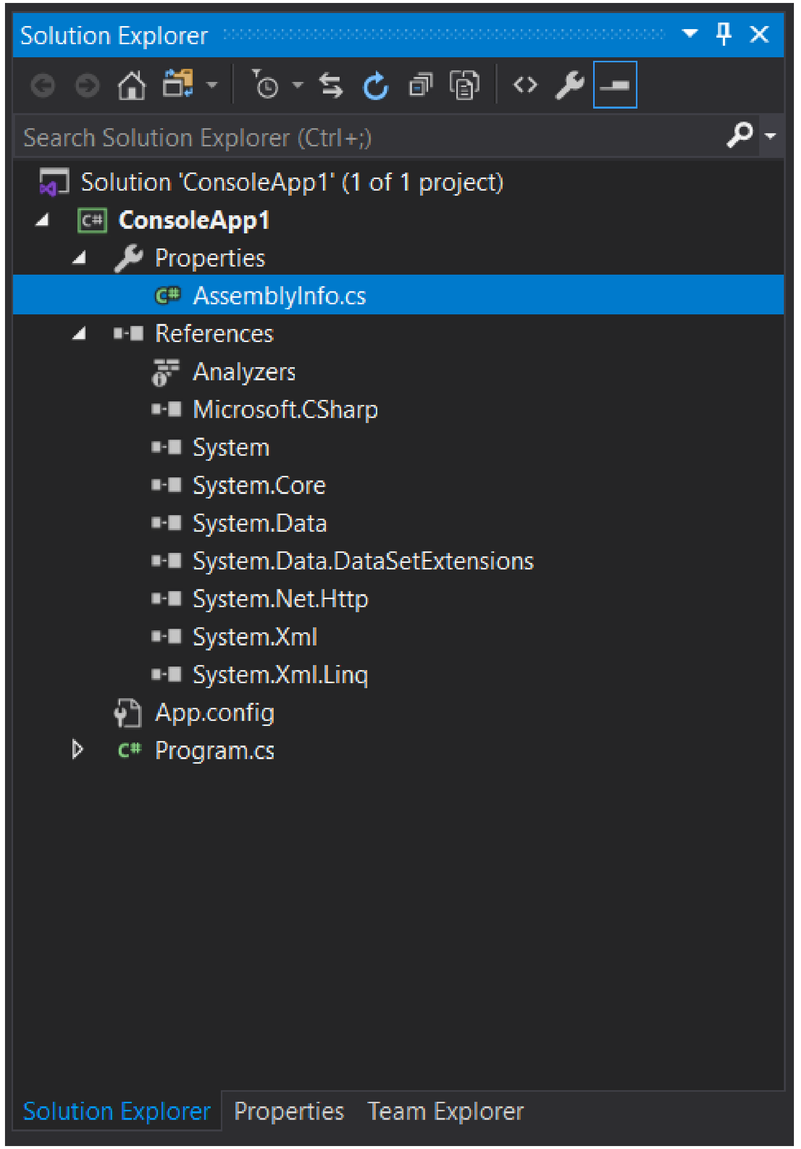
Creating a new project

- To create a new project, we can select the File -> New -> Project command
- From the list on the right, we can select a template for our project. We can search for templates or use the filters to narrow down our search
- In the next page we can type a name for our project, the location of the project on disk and the version of the .NET Framework we want to target



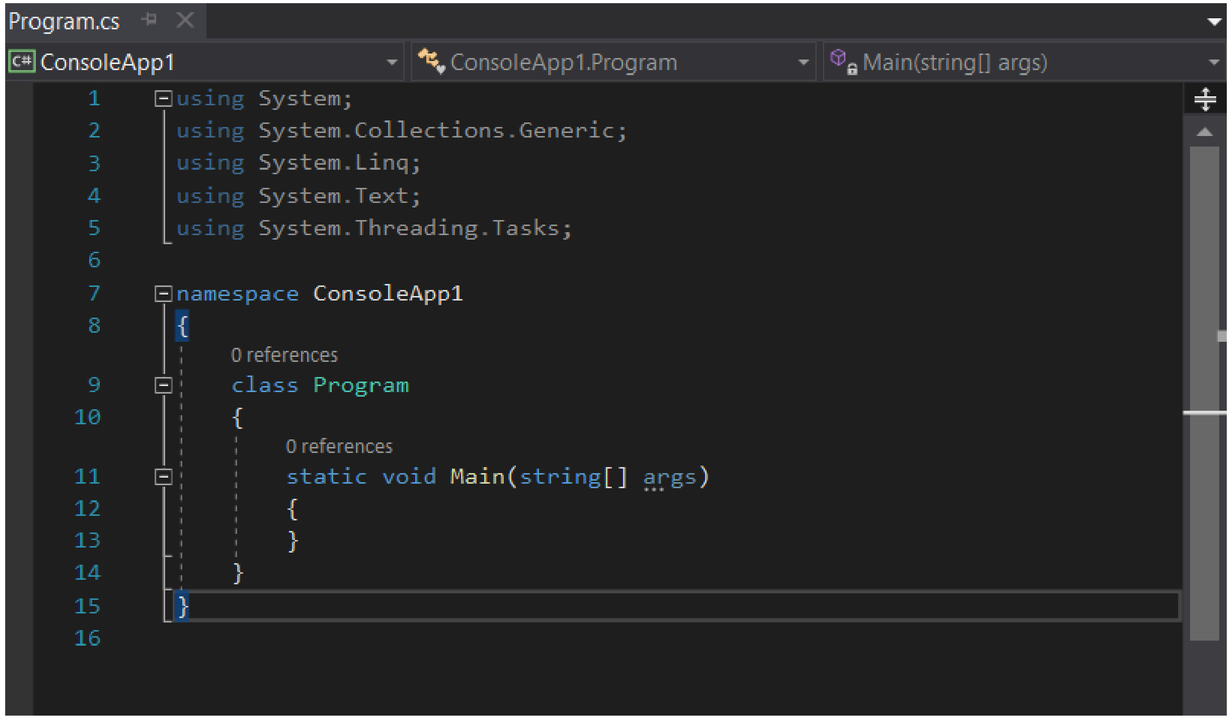
Structure of a simple console app

- Solution
- Project
- Properties: AssemblyInfo.cs file
- Referenced assemblies
- Application configuration file
- Application code: Program.cs



Structure of a simple console app

- Import used namespaces
- Namespace declaration
- Class declarations
- Method declarations



The C# Language

- It is a general-purpose, multi-paradigm programming language
- It is similar in syntax to C and C++
- Statements are executed in order, one after the other
- Each statement must end with a semicolon

Hello World!

- To run the app, use the Debug -> Start debugging command
- If you want the window to remain on-screen, use Debug -> Start without debugging
- The main entry point into the program is the static Main method

```
using System;

namespace ConsoleApp1
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Hello World!");
        }
    }
}
```

Variables

- To declare a variable, we must write its type, then its name, and end the declaration with a semicolon
- Variables can optionally be initialized by typing = and then the value
- Multiple variables can be declared on the same line by separating them with colons

```
using System;

namespace ConsoleApp1
{
    class Program
    {
        static void Main(string[] args)
        {
            int x = 5;
            Console.WriteLine("The value of x is {0}", x);
        }
    }
}
```

Constants

- To declare a constant, we use the const keyword before the type of the constant
- Constants must be initialized when they are declared

```
using System;

namespace ConsoleApp1
{
    class Program
    {
        static void Main(string[] args)
        {
            const int y = 6;
            //y = 5; // Compile error
            Console.WriteLine("The value of y is {0}", y);
        }
    }
}
```

Data types

- Value types:
  - Simple types:
    - Integral: byte, short, int, long
    - Characters: char
    - Floating point: float, double
    - High-precision: decimal
    - Boolean: bool
  - Enum types
  - Struct types
  - Nullable value types
- Reference types
  - Class types
    - Base class: object
    - Strings: string
    - User-defined types
  - Interface types
  - Array types
  - Delegate types

Operators

- They are program elements that are applied to one or more operands in an expression or statement

```
int x = 5;
int y = 6;

int sum = x + y;
int dif = x - y;
int mul = x * y;
int div = x / y;

Console.WriteLine("The value of sum is {0}", sum);
Console.WriteLine("The value of dif is {0}", dif);
Console.WriteLine("The value of mul is {0}", mul);
Console.WriteLine("The value of div is {0}", div);
```

Operators

- Unary operators: `+` `-` `!` `~` `++x` `--x` `(T)x`
- Arithmetic operators: `*` `/` `%` `+` `-`
- Shift operators: `<<` `>>`
- Relational and type testing operators: `<` `>` `<=` `>=` `is` `as` `==` `!=`
- Logical operators: `&` `^` `|`
- Conditional logical operators: `&&` `||`
- Null coalescing operator: `??`
- Conditional operator: `?:`
- Assignment operators: `=` `*=` `/=` `%=` `-=` `<<=` `>>=` `&=` `^=` `|=`

Calling methods

- To call a method, we write the name of the method, then in brackets we write the parameters that we want to pass to that method, separated by colons

```
using System;

namespace ConsoleApp1
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Hello World!");
        }
    }
}
```

Defining classes

- A class can be declared inside a namespace
- To declare a namespace, we use the namespace keyword, followed by its name and its body in curly braces
- To define a class, we use the class keyword, followed by the name of the class, and then the body of the class inside curly braces

```
namespace CSharpLanguage.Topics
{
    class DefiningClasses
    {
    }
}
```

Defining methods

- Methods can be defined inside the body of a class
- To define a method, we write the following items:
  - Access modifiers (public, private, etc)
  - Return type
  - Method name
  - In parentheses, separated by commas: method parameters (data type and name)
  - In curly braces: method body

```
static void AddNumbers(int a, int b)
{
    int result = a + b;
    Console.WriteLine("The result is {0}", result);
}
```

Input / output

- The Console class is used in console applications to interact with the user
- To write text to the console, we use Console.WriteLine
- To read text from the console, we use Console.ReadLine

```
using System;

namespace ConsoleApp1
{
    class Program
    {
        static void Main(string[] args)
        {
            string input;
            Console.WriteLine("Enter some text:");
            input = Console.ReadLine();
            Console.WriteLine("The text you entered is {0}", input);
        }
    }
}
```

String parsing

- To extract a data type from a string, we use the Parse method of the data type we want to extract the number to (int, float, double, bool, etc)

```
using System;

namespace ConsoleApp1
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Type a number:");
            string str = Console.ReadLine();
            int nr = int.Parse(str);
            Console.WriteLine("The number after that is {0}", nr + 1);
        }
    }
}
```

Generating a random number

- First we need to create a random number generator, using the Random class
- Then, we can call the Next() method in order to get a random number
- We can specify a minimum and maximum value for the random number in the call to the Next method

```
using System;

namespace ConsoleApp1
{
    class Program
    {
        static void Main(string[] args)
        {
            Random rng = new Random();
            int r = rng.Next(1,100);
            Console.WriteLine(r);
        }
    }
}
```

Selection statements

- They select a number of possible statements for execution based on the value of an expression
- There are two selection statements in C#:
  - The if statement
  - The switch statement

The if statement

- It is used to execute a block of code depending on the logical value of an expression
- The expression must evaluate to a Boolean, and it must be enclosed in brackets

```
Console.WriteLine("Type a number:");
string str = Console.ReadLine();
int nr = int.Parse(str);
if (nr > 5)
{
    Console.WriteLine("The number is greater than 5");
}
else
{
    Console.WriteLine("The number is not greater than 5");
}
```

The switch statement

- It allows us to execute a block of code based on the value of an expression that can be found in a list of candidate values
- The candidate values must be constants, and they must be of the following types: byte, short, int, long, char, string and enumeration

```
Console.WriteLine("Type a number:");
string str = Console.ReadLine();
int nr = int.Parse(str);
switch(nr)
{
    case 1:
        Console.WriteLine("The number is 1");
        break;
    case 2:
        Console.WriteLine("The number is 2");
        break;
    default:
        Console.WriteLine("The number is not in the list");
        break;
}
```

The switch statement

- Case labels must contain constant expressions
- A case label can be empty
- If a case label is not empty, it must be terminated with break or goto
- The default section can be omitted
- The default section must also be terminated with break or goto

Iteration statements

- There are 4 iteration statements in C#:
  - while
  - do … while
  - for
  - foreach

The while statement

- It executes a block of code as long as the value of a logical expression is true
- The expression is tested before the first time the block is executed

```
int a = 30;
int b = 7;
int x = 0;

while (a > 0)
{
    a = a - b;
    x++;
}

Console.WriteLine("b can be subtracted from a {0}
times", x);
```

The do … while statement

- It executes a block of code one or more times, as long as the value of a logical expression is true
- The expression is tested after the first time the block is executed

```
int x = 0;

do
{
    Console.WriteLine("Enter a negative number");
    string str = Console.ReadLine();
    x = int.Parse(str);
} while (x >= 0);

Console.WriteLine("You entered the number {0}", x);
```

The for statement

- It executes an initialization statement, after that it will execute a block of code as long as an expression is true
- It can also execute a statement before continuing to the next step
- All three parts of the expression are optional

```
for (int i = 0; i < 10; i++)
{
    Console.WriteLine(i);
}
```

The foreach statement

- It enumerates the elements of a collection, and executes a block of code for each element in the collection
- The collection must implement the IEnumerable interface
- The iteration variable cannot be modified inside the loop

```
int[] array = { 1, 2, 3, 4 };
foreach(int number in array)
{
    Console.WriteLine(number + 1);
}
```

Jump statements

- They unconditionally transfer control to another block of code
- The **break** statement: exits the nearest switch, while, do … while, for or foreach statement
- The **continue** statement: starts a new iteration of the nearest enclosing while, do, for or foreach statement
- The **goto** statement: transfers control to a statement that is marked by a label
- The **return** statement: transfers control to the caller of the method where the return is present
- The **throw** statement: throws an exception, it transfers control to the first catch clause that can handle the exception

Homework

- Create a “guess the number” game
- You must generate a random number between 0 and 100 (the target number)
- You ask the user to guess the number, by entering a number in the console
- If the user guesses correctly, the game ends
- If the user’s guess was smaller than the target number, tell the user that the number was too small
- If the user’s guess was larger than the target number, tell the user that the number was too big
- Allow the user to guess again until he guesses correctly

Reference

- https://en.wikipedia.org/wiki/.NET_Framework
- https://www.microsoft.com/net/
- https://www.visualstudio.com
- https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/reference-tables-for-types
- https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/statements-expressions-operators/operators
- https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/index