C#

Canal Fessor Bruno (youtube.com/canalfessorbruno)

+

Documentação da Microsoft

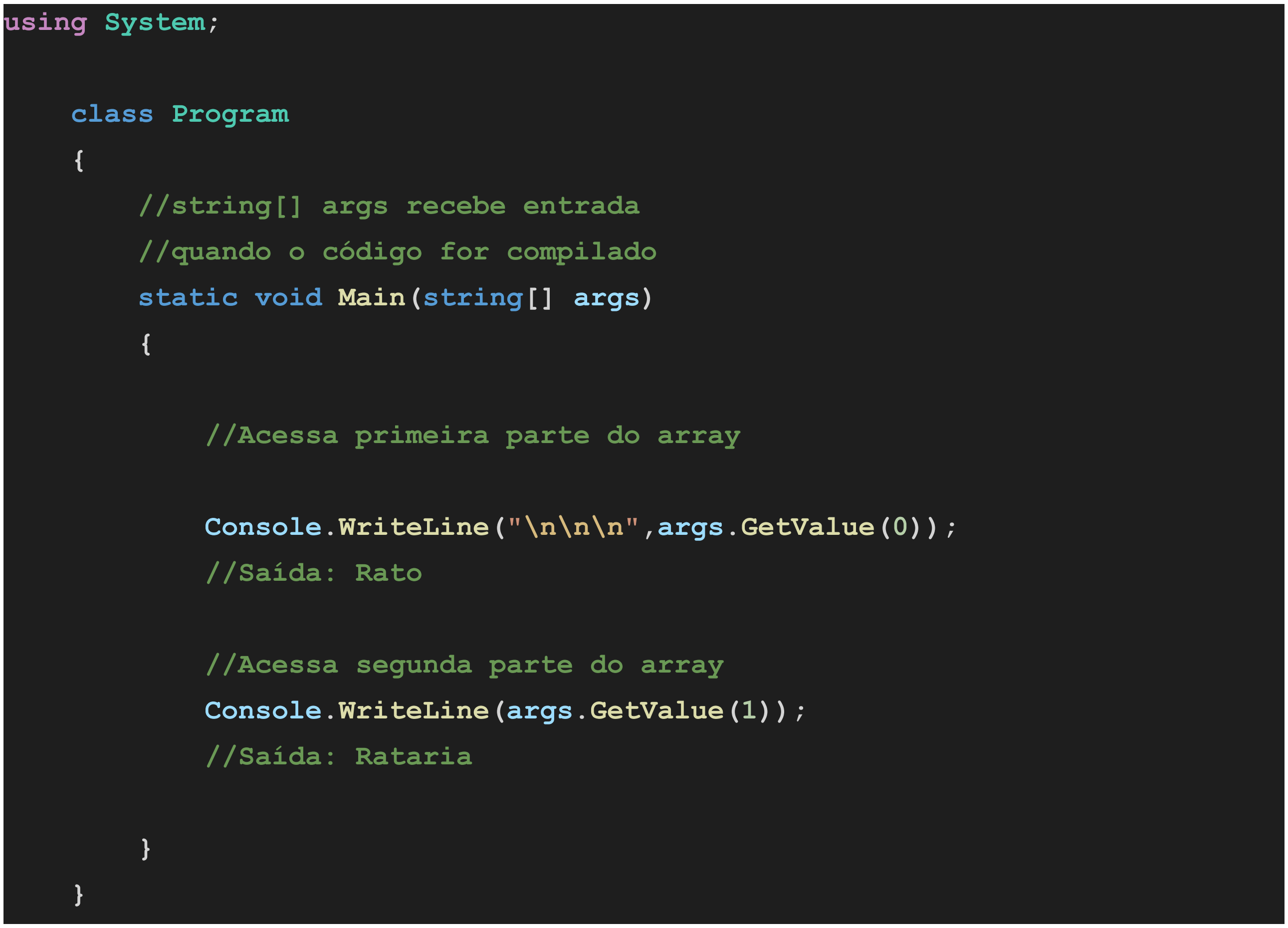
Links:

(MS)
https://docs.microsoft.com/pt-br/dotnet/csharp/

(CFB)
https://www.youtube.com/playlist?list=PLx4x_zx8csUglgKTmgfVFEhWWBQCasNGi

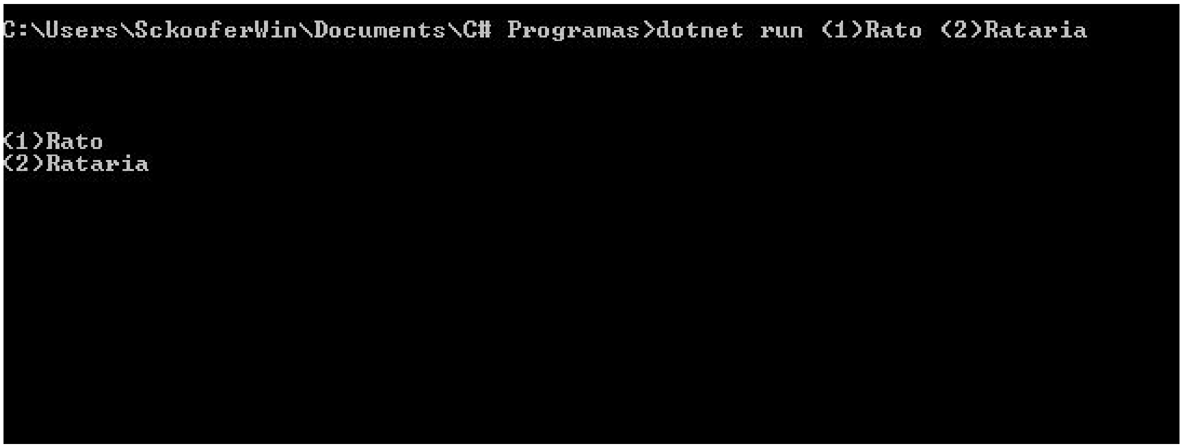
2019-2020

Primeiro programa no padrão c# .net - Curso Programação Completo C# - Aula 02



```
using System;

    class Program
    {
        //string[] args recebe entrada
        //quando o código for compilado
        static void Main(string[] args)
        {

            //Acessa primeira parte do array

            Console.WriteLine("\n\n\n",args.GetValue(0));
            //Saída: Rato

            //Acessa segunda parte do array
            Console.WriteLine(args.GetValue(1));
            //Saída: Rataria


        }
    }
```

O valor foi inserido (Entrada) logo após o comando dotnet run. Sendo usado logo em seguida dentro da palavra chave Console.WriteLine();



```
C:\Users\SckooferWin\Documents\C# Programas>dotnet run (1)Rato (2)Rataria



(1)Rato
(2)Rataria
```

Documentação da Microsoft

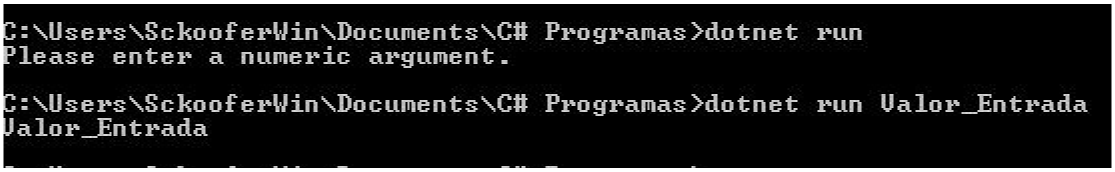
https://docs.microsoft.com/pt-br/dotnet/csharp/programming-guide/main-and-command-args/command-line-arguments

You can send arguments to the `Main` method by defining the method in one of the following ways:

```
            static int Main(string[] args){}
            static void Main(string[] args){}
```

```
using System;

     //The args array cannot be null. So, it's safe to access
     //the Length property without null checking.
    class Program
    {
        static int Main(string[] args)
        {
            if (args.Length == 0)
            {
               System.Console.WriteLine("Please enter a numeric argument.");
                return 1;
            }
            Console.WriteLine("{0}",args);
            return 0;
        }
    }
```

The `args` array cannot be null. So, it's safe to access the `Length` property without null checking.



```
C:\Users\SckooferWin\Documents\C# Programas>dotnet run
Please enter a numeric argument.

C:\Users\SckooferWin\Documents\C# Programas>dotnet run Valor_Entrada
Valor_Entrada
```

You can also convert the string arguments to numeric types by using the Convert class or the `Parse` method. For example, the following statement converts the `string` to a `long` number by using the Parse method:

```
long num = Int64.Parse(args[0]);
//It is also possible to use the C# type long, which aliases Int64:
long num = long.Parse(args[0]);
//You can also use the Convert class method ToInt64 to do the same thing:
long num = Convert.ToInt64(s);
```

------> The following example shows how to use command-line arguments in a console application. The application takes one argument at run time, converts the argument to an integer, and calculates the factorial of the number. If no arguments are supplied, the application issues a message that explains the correct usage of the program.

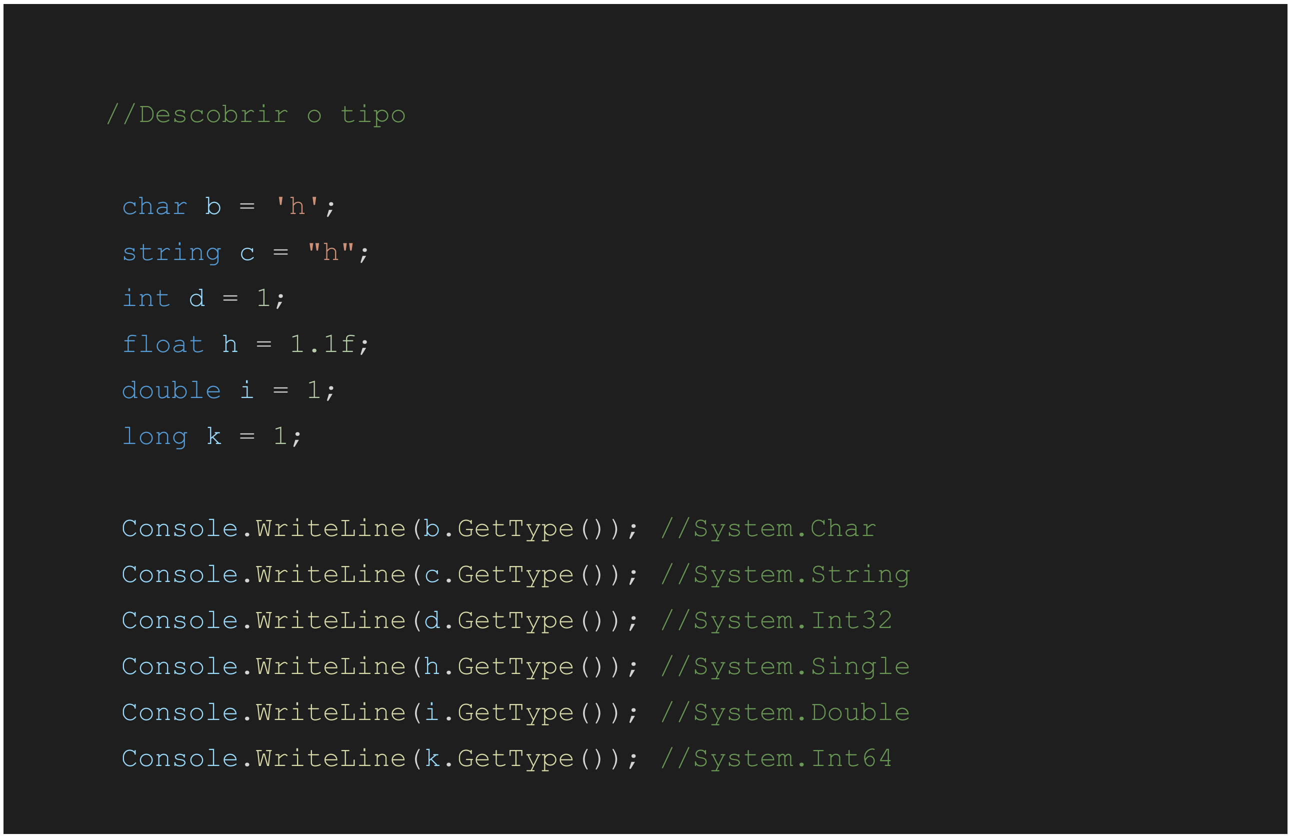
```
// Add a using directive for System if the directive isn't already present.

public class Functions
{
    public static long Factorial(int n)
    {
        // Test for invalid input.
        if ((n < 0) || (n > 20))
        {
            return -1;
        }
        // Calculate the factorial iteratively rather than recursively.
        long tempResult = 1;
        for (int i = 1; i <= n; i++)
        {
            tempResult *= i;
        }
        return tempResult;
    }
}
```

```
class MainClass
{
    static int Main(string[] args)
    {
        // Test if input arguments were supplied.
        if (args.Length == 0)
        {
            Console.WriteLine("Please enter a numeric argument.");
            Console.WriteLine("Usage: Factorial <num>");
            return 1;
        }
        // Try to convert the input arguments to numbers. This will throw
        // an exception if the argument is not a number.
        // num = int.Parse(args[0]);
        int num;
        bool test = int.TryParse(args[0], out num);
        if (!test)
        {
            Console.WriteLine("Please enter a numeric argument.");
            Console.WriteLine("Usage: Factorial <num>");
            return 1;
        }
        // Calculate factorial.
        long result = Functions.Factorial(num);

        // Print result.
        if (result == -1)
            Console.WriteLine("Input must be >= 0 and <= 20.");
        else
            Console.WriteLine($"The Factorial of {num} is {result}.");
        return 0;
    }
}

// If 3 is entered on command line, the
// output reads: The factorial of 3 is 6.
```

Variáveis - Curso Programação Completo C# - Aula 03

Fortemente tipado significa que o tipo da variável é importante

```
        byte    //0 ..255
        sbyte   //-128 ..127

        short   //-32,768 ..32,767
        ushort  //0 ..65,535

        int     //-2,147,483,648 ..2,147,483,647
        uint    //0 ..4,294,967,295

        long    -9,223,372,036,854,775,808..9,223,372,036,854,775,807
        ulong   //0 ..18,446,744,073,709,551,615

        float   //-3.402823e38 ..3.402823e38
        double  //-1.79769313486232e308 ..1.79769313486232e308

        decimal  //-7922816251426435..7922816251426433
        char     //U+0000 .. U+ffff

       //Tipo implícito, "the compiler determines the type"
        var variavel = 10;
        //Tipo explícito, "is strongly typed only if declared the type"
        int variavel2 = 10;
        //Inicializa a variável com valor zero.
        var x = new int();
        //Texto.
        string variavel3 = “palavra”;
    //Como que descubro o valor Máximo e Mínimo de um tipo?
    // Sitaxe:  tipo       nome       =     tipo.MaxValue/MinValue

         double Nome_Variavel = double.MaxValue;  //255
         byte Nome_Variavel = byte.MinValue;     //0
```



```
//Descobrir o tipo

 char b = 'h';
 string c = "h";
 int d = 1;
 float h = 1.1f;
 double i = 1;
 long k = 1;

 Console.WriteLine(b.GetType()); //System.Char
 Console.WriteLine(c.GetType()); //System.String
 Console.WriteLine(d.GetType()); //System.Int32
 Console.WriteLine(h.GetType()); //System.Single
 Console.WriteLine(i.GetType()); //System.Double
 Console.WriteLine(k.GetType()); //System.Int64
```

Formatando a saída no console - Curso Programação Completo C# - Aula 06

```
    //Uso de índices. Formatação composta de cadeia de caracteres
    Console.WriteLine("Var1--> {0}\nVar2--> {1}", variavel1, variavel2);

    /*Interporlação de Strings / String interpolation
     A interpolação de cadeia de caracteres fornece uma sintaxe mais
     legível e conveniente para criar cadeias  de caracteres formatadas do
     que o recurso de formatação composta de cadeia de caracteres.*/
    Console.WriteLine("",$"{variavel1}"+$"{variavel2}");

    //Intercala entre elementos e variáveis
Console.WriteLine("Var1" + variavel1 + "Segunda Var2"  + variavel2+"\n");

    //Tabulação
    Console.WriteLine("\t");

    //{0,5}Coloca espaço entre a string e a saída
    Console.WriteLine("Saída com espaçamento:",string1);
    //{0:c}  adiciona cifrão. Precisa do ZERO e da VÍRGULA, ponto . NÃO!
    Console.WriteLine("Saída com cifrão: {0:c}",VariavelDouble);
    //{0:p}  adiciona o símbolo da porcentagem
    Console.WriteLine("Saída com porcentagem:{0:p}",porcentagem);

    //////////////////////Saída no console////////////////

    Var1--> 1
    Var2--> 2
    Interpolação
    Intercala elementos
    Primeira Var.....: 1
    Segunda Var......: 2
    Saída com espaçamento...........: SAIDA
    Saída com cifrão................: $5.50
    Saída com porcentagem...........: 10.000%
```

Constantes em C# - Curso Programação Completo C# - Aula 07

Bruno: “ao inserir uma constante, o valor dela não poderá ser alterado ao longo do programa”.

```
        const string NomeVariavel="CFB Cursos";
        const double Variavel =  3.1475;

        Console.WriteLine($"{canal}");
```

===

DOCUMENTAÇÃO DA MICROSOFT

The use of the class name qualifier helps ensure that you and others who use the constant understand that it is constant and cannot be modified.

```
 static class Constants
 {
     public const double Pi = 3.14159;
     public const int SpeedOfLight = 300000; // km per sec.
 }
 class Program
 {
     static void Main()
     {
         double radius = 5.3;
         double area = Constants.Pi * (radius * radius);
         int secsFromSun = 149476000 / Constants.SpeedOfLight; // in km
}
 }
```

Lendo valores do teclado - Curso Programação Completo C# - Aula 08

```
    /*A conversão entre tipos que pode ser realizada automaticamente é
conhecida como Implícita :
    Ex:*/

     byte valor1 = 10;
     byte valor2 = 23;
     long total = valor1 + valor2;

      //Conversão explícita:
     long valor = 3000;
     int a = (int) valor ;
     int v1,v2,soma;


     Console.Read();       // Leitura
     Console.ReadLine();   //  Leitura e quebra de linha


     /*Nesse caso, os métodos tem retorno,
     esse retorno é um valor atribuído à
     variável*/

     // Converção para o tipo INT
     valor_inteiro = int.Parse(Console.ReadLine());

     // Funciona semelhantemente como o Parse
     valor_inteiro = Convert.ToInt32(Console.ReadLine());
```

DOCUMENTAÇÃO DA MICROSOFT

https://docs.microsoft.com/pt-br/dotnet/csharp/programming-guide/types/casting-and-type-conversions

No C#, você pode realizar os seguintes tipos de conversões:
-> **Conversões Implícitas**
-> **Conversões Explícitas** (conversão)
-> **Conversões definidas pelo usuário** (User-defined Conversions).
-> **Conversões com classes auxiliares** (Conversions with helper classes)

Implicit Conversion: No special syntax is required because the conversion always succeeds and no data will be lost. Examples include conversions from smaller to larger integral types, and conversions from derived classes to base classes.

For “built-in numeric types” (integral numeric type, floating-point numeric type (números inteiros e reais que também são chamados de tipos numéricos internos)), an implicit conversion can be made when the value to be stored can fit into the variable without being truncated or rounded off.

For integral types, this means the range of the source type is a proper subset (subconjunto apropriado) of the range for the target type. For example, a variable of type long (64-bit integer) can store any value that an int (32-bit integer) can store.

```
//In the following example, the compiler implicitly converts the value
//of num on the right to a type long before assigning it to bigNum.
int      num          =     2147483647;
long      bigNum      =     num;
```

For reference types, an implicit conversion always exists from a class to any one of its direct or indirect base classes or interfaces. No special syntax is necessary because a derived class always contains all the members of a base class.

```
Derived d = new Derived();
Base b = d; // Always OK.
```

Explicit Conversion (casts): Explicit conversions require a cast expression.

Casting is required when information might be lost in the conversion, or when the conversion might not succeed for other reasons. Typical examples include numeric conversion to a type that has less precision or a smaller range, and conversion of a base-class instance to a derived class.

To perform a cast, specify the type that you are casting to in parentheses in front of the value or variable to be converted. The following program casts a double to an int. The program will not compile without the cast.

```
double x = 1234.7;
// Cast double to int
int a = (int)x;
Console.WriteLine(a);    // output: 1234

/------------------------------------------------------/
// Create a new derived type.
Giraffe g = new Giraffe();

// Implicit conversion to base type is safe.
Animal a = g;

// Explicit conversion is required to cast back
// to derived type. Note: This will compile but will
// throw an exception at run time if the right-side
// object is not in fact a Giraffe.
Giraffe g2 = (Giraffe)a;
```

A cast operation between reference types does not change the run-time type of the underlying object; it only changes the type of the value that is being used as a reference to that object.

Type conversion exceptions at run time (Exceções de conversão de tipo em tempo de execução):

In some reference type conversions, the compiler cannot determine whether a cast will be valid. Is possible, even for a correctly compiled cast operation, to fail during at run time (tempo de execução).

A type cast that fails at run time will cause an InvalidCastException to be thrown.

Uma InvalidCastException exceção é causada por um erro do desenvolvedor e não deve ser manipulada em um try/catch bloco. Em vez disso, a causa da exceção deve ser eliminada.

O C# fornece o operador is para habilitar o teste de compatibilidade antes de realmente executar uma conversão.

User-defined conversions (conversões definidas pelo usuário):
Are performed by ***special methods*** that you can define to enable explicit and implicit conversions between custom types that do not have a base class–derived class relationship.

https://docs.microsoft.com/pt-br/dotnet/csharp/language-reference/operators/user-defined-conversion-operators

If a custom conversion can throw an exception or lose information, define it as an explicit conversion.

User-defined conversions are not considered by the is and as operators. Use a cast expression to invoke a user-defined explicit conversion.

*(How to define a **custom** explicit or implicit type conversion?)*

```csharp
public readonly struct Digit
{
   private readonly byte digit;

   public Digit(byte digit)
   {
       if (digit > 9)
       {
           throw new ArgumentOutOfRangeException(nameof(digit), "Digit cannot be greater than nine.");
       }
       this.digit = digit;
   }

   public static implicit operator byte(Digit d) => d.digit;
   public static explicit operator Digit(byte b) => new Digit(b);

   public override string ToString() => $"{digit}";
}

public static class UserDefinedConversions
{
   public static void Main()
   {
       var d = new Digit(7);

       byte number = d;
       Console.WriteLine(number);  // output: 7

       Digit digit = (Digit)number;
       Console.WriteLine(digit);  // output: 7
   }
}
```

Conversions with helper classes (Conversões com classes auxiliares):

To convert between non-compatible types, such as integers and System.DateTime objects, or hexadecimal strings and byte arrays, you can use the System.BitConverter class, the System.Convert class, and the Parse methods of the built-in numeric types, such as Int32.Parse.

https://docs.microsoft.com/pt-br/dotnet/csharp/programming-guide/types/how-to-convert-a-string-to-a-number#calling-the-parse-and-tryparse-methods

```
//Entrada de inteiros no console
        valor = int.Parse(Console.ReadLine());
//The Convert.ToInt32 method uses Parse internally.
              valor = Convert.ToInt32(Console.ReadLine());
```

You can convert a string to a number by calling the Parse or TryParse method found on the various numeric types (int, long, double, etc.), or by using methods in the System.Convert class.
If you have a string, it is slightly more efficient and straightforward to call a TryParse method, for example: int.TryParse("11", out number) Or Parse method, for example
 var number = int.Parse("11")).

```
string string1 = "50";
      int result = -5;                          //Saída:    51
      result = int.Parse(string1);
      Console.WriteLine($"{result + 1}");

                        /*Perceba que, result valia -5, passou a valer
                   a mesma coisa que string1, mas no tipo numérico*/

int number = 0;
      bool conver;                                    //Saída: 11
      conver= int.TryParse("11", out number);
```

Using a Convert method is more useful for general objects that implement IConvertible. You can use Parse or TryParse methods on the numeric type you expect the string contains, such as the System.Int32 type.

The Parse method returns the converted number.
The TryParse method returns a Boolean value that indicates whether the conversion succeeded, and returns the converted number in an out parameter.
If the string is not in a valid format, Parse throws an **exception**, whereas TryParse returns **false**.

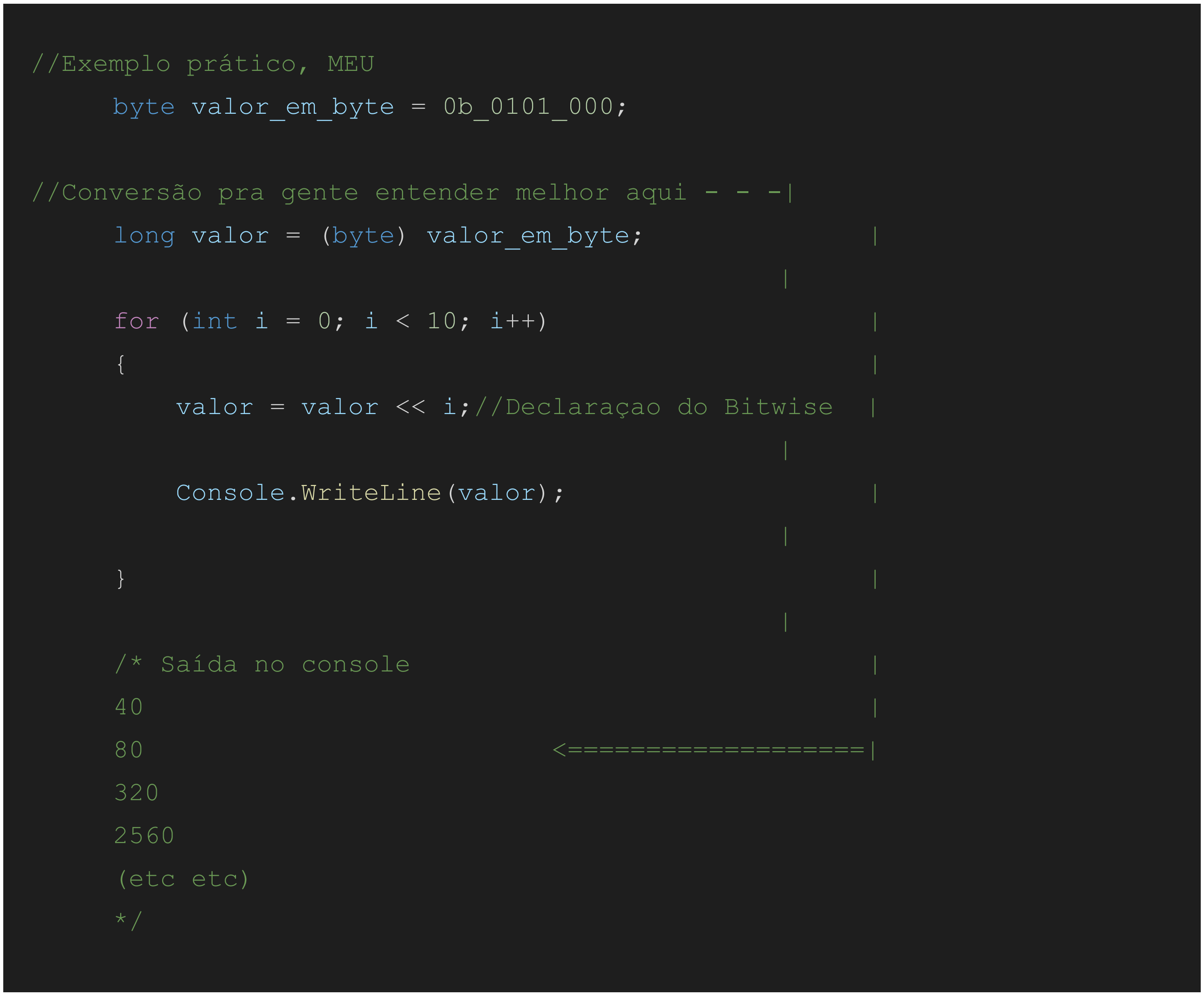
```
        //converte pra string

        int valor = 0;

        string palavra = valor.ToString();

//ou


        var x = 2;
        var result = Convert.ToString(x);
        Console.WriteLine(x);
```

Operações de Bitwise - Curso Programação Completo C# - Aula 09

Operações de bitwise (ou operadores de sift). Basicamente servem para deslocar os bits para esquerda ou para direita dentro de variáveis inteiras (inteiras e suas variações).

Então nós temos o operador bitwise que vai deslocar para esquerda e o operador que vai deslocar para a direita.

```
//---------------------------------------------------//

<<      //Bitwise para a esquerda dobra o valor
>>      //Bitwise para a direita vai diminuir o valor pela metade

/*Exemplo:

00001010           //Equivale à 10

00001010    <<     //Dobra o valor

00010100           //Agora, equivale à 20


//---------------------------------------------------//

Exemplo 2:

00010100           //Equivale à 20

00010100    >>     //Divide pela metade

00001010           //Agora, equivale à 10


//---------------------------------------------------//
```



```
//Exemplo prático, MEU
     byte valor_em_byte = 0b_0101_000;

//Conversão pra gente entender melhor aqui - - -|
     long valor = (byte) valor_em_byte;              |
                                                |
     for (int i = 0; i < 10; i++)                    |
     {                                               |
         valor = valor << i;//Declaraçao do Bitwise  |
                                                |
         Console.WriteLine(valor);                   |
                                                |
     }                                               |
                                                |
     /* Saída no console                             |
     40                                              |
     80                          <===================|
     320
     2560
     (etc etc)
     */
```

DOCUMENTAÇÃO DA MICROSOFT

https://docs.microsoft.com/pt-br/dotnet/csharp/language-reference/operators/bitwise-and-shift-operators#bitwise-complement-operator-

Operador unário “Unary”
~ (bitwise complement) operator

Operadores de deslocamento binário “Binary”
<< (left shift) and >> (right shift) sift operators

Operadores Binary ”Binary”
& (logical AND), | (logical OR), and ^ (logical exclusive OR) operators

Those operators are defined for the **int**, **uint**, **long**, and **ulong** types. When both operands are of other integral types (**sbyte**, **byte**, **short**, **ushort**, or **char**), their values are converted to the int type, which is also the result type of an operation.
When operands are of different integral types, their values are converted to the closest containing integral type.

The “ **&”** **“|”** and **“^”** operators are also defined for operands of the bool type.

Bitwise and shift operations never cause overflow and produce the same results in checked and unchecked contexts.
{
https://docs.microsoft.com/pt-br/dotnet/csharp/language-reference/keywords/checked-and-unchecked
Checked: In a checked context, arithmetic overflow raises an exception.
Unchecked: In an unchecked context, arithmetic overflow is ignored and the result is truncated by discarding any high-order bits that don't fit in the destination type.

The following operations are affected by the overflow checking:

=>Expressions using the following predefined operators on integral types:
“ **++** “ “ **--** “ “ **unary -** “ “ **+** “ “ **-** “
“ ***** “ “ **/** “
=>Explicit numeric conversions between integral types, or from **float** or **double** to an integral type.

If neither checked nor unchecked is specified, the default context for non-constant expressions (expressions that are evaluated at run time) is defined by the value of the -checked compiler option. By default the value of that option is unset and arithmetic operations are executed in an unchecked context.

For constant expressions (expressions that can be fully evaluated at compile time), the default context is always checked. Unless a constant expression is explicitly placed in an unchecked context, overflows that occur during the compile-time evaluation of the expression cause compile-time errors.

```
static void Main()
 {
   int int1;


   //(CS0220) The operation overflows at compile time in checked mode
   //Não deveria funcionar. Depois do uso de unchecked inicia negativo
   checked
   {
       int1 = 2147483647 + 10;
   }
   //Ignora o erro CS0220, o programa continua mesmo com overflow
   unchecked
   {
       int1 = 2147483647 + 10;
       Console.WriteLine(int1);
       //saída: -2147483639
   }

 }
```

Se o ambiente unchecked for removido, ocorrerá um erro de compilação.

}

Bitwise complement operator “ ~ “

The ~ operator produces a bitwise complement of its operand by reversing each bit:

```
uint a = 0b_0000_1111_0000_1111_0000_1111_0000_1100;

uint b = ~a; //Atribui o valor de ' a ' e inverte

Console.WriteLine(Convert.ToString(b, toBase: 2));

// Output: 11110000111100001111000011110011
```

You can also use the ~ symbol to declare finalizers (which are also called **destructors** (são os destrutores/destruidores de classes)). Are used to perform any necessary final clean-up when a class instance is being collected by the garbage collector.

{

```
class Car
{
   ~Car()  // finalizer
   {
       // cleanup statements
   }
}
```

(mais a frente vamos ter aula sobre isso)

}

Left-shift operator ‘ << ‘

The << operator shifts (alterna) its left-hand operand left by the number of bits defined by its right-hand operand.

The left-shift operation discards the high-order bits that are outside the range of the result type and sets the low-order empty bit positions to zero, as the following example shows:

```
uint x = 0b_1100_1001_0000_0000_0000_0000_0001_0001;
Console.WriteLine($"Before: {Convert.ToString(x, toBase: 2)}");

uint y = x << 4; //Avança quatro casas
Console.WriteLine($"After:  {Convert.ToString(y, toBase: 2)}");
// Output:
// Before: 11001001000000000000000000010001
// After:  10010000000000000000000100010000
```

Because the shift operators are defined only for the int, uint, long, and ulong types, the result of an operation always contains at least 32 bits. If the left-hand operand is of another integral type (sbyte, byte, short, ushort, or char), its value is converted to the int type, as the following example shows:

```
byte a = 0b_1111_0001;
var b = a << 8;

Console.WriteLine(b.GetType()); //Mostra o tipo
Console.WriteLine($"Shifted byte: {Convert.ToString(b, toBase:2)}");
// Output:
// System.Int32
// Shifted byte: 1111000100000000
```

->
https://docs.microsoft.com/pt-br/dotnet/csharp/language-reference/operators/bitwise-and-shift-operators#shift-count-of-the-shift-operators

Para os operadores de deslocamento << e >> , o tipo do operando à direita deve ser int ou um tipo que tenha uma conversão numérica implícita predefinida para int .

Para as expressões x << count e x >> count, a contagem real de deslocamento depende do tipo de x da seguinte maneira:

- Se o tipo de x for int ou uint , a contagem de deslocamento será definida pelos *cinco* bits de ordem inferior do operando à direita. Ou seja, a contagem de deslocamentos é calculada a partir de count & 0x1F (ou count & 0b_1_1111).

- Se o tipo de x for long ou ulong , a contagem de deslocamento será definida pelos *seis* bits de ordem inferior do operando à direita. Ou seja, a contagem de deslocamentos é calculada a partir de count & 0x3F (ou count & 0b_11_1111).

```
        int count1 = 0b_0000_0001;
       int count2 = 0b_1110_0001;

        int a = 0b_0001;

        Console.WriteLine($"{a} << {count1} is {a << count1}; {a} << {count2}
is {a << count2}");

        // Output:
       // 1 << 1 is 2; 1 << 225 is 2

        int b = 0b_0100;

        Console.WriteLine($"{b} >> {count1} is {b >> count1}; {b} >> {count2}
is {b >> count2}");

        // Output:
         // 4 >> 1 is 2; 4 >> 225 is 2
```

<!--Não Finalizou Completamente-->

A partir daqui, não tem mais aulas do CFB Cursos.
Todos os títulos abaixo são títulos dos vídeos, porém o conteúdo das páginas são links para a documentação oficial da Microsoft

-
-
-
-
-
-
-
-
-
-
-
-
-
-
-
-
-
-
-
-
-
-
-
-
-
-
-
-
-
-
-
-
-
-
-
-

Enumeradores (enum) - Curso Programação Completo C# - Aula 10

https://docs.microsoft.com/pt-br/dotnet/csharp/language-reference/builtin-types/enum

Classe enum

https://docs.microsoft.com/pt-br/dotnet/api/system.enum?view=netcore-3.1

Conversões de tipos (typecast) - Curso Programação Completo C# - Aula 11

https://docs.microsoft.com/pt-br/dotnet/csharp/programming-guide/types/casting-and-type-conversions

Array / Vetor - Curso Programação Completo C# - Aula 17

https://docs.microsoft.com/pt-br/dotnet/csharp/programming-guide/arrays/single-dimensional-arrays#see-also

Classe Array

https://docs.microsoft.com/pt-br/dotnet/api/system.array?view=netcore-3.1

Matrizes / Vetores Bidimensionais - Curso Programação Completo C# - Aula 18

https://docs.microsoft.com/pt-br/dotnet/csharp/programming-guide/arrays/

Matrizes multidimensionais

https://docs.microsoft.com/pt-br/dotnet/csharp/programming-guide/arrays/multidimensional-arrays

Matrizes denteadas

https://docs.microsoft.com/pt-br/dotnet/csharp/programming-guide/arrays/jagged-arrays

Loop FOREACH / Estruturas de iteração - Curso Programação Completo C# - Aula 22

https://docs.microsoft.com/pt-br/dotnet/csharp/language-reference/keywords/foreach-in

Métodos - Curso Programação Completo C# - Aula 24
Methods
https://docs.microsoft.com/pt-br/dotnet/csharp/programming-guide/classes-and-structs/methods
Methods in
https://docs.microsoft.com/pt-br/dotnet/csharp/methods
Method Parameters
https://docs.microsoft.com/pt-br/dotnet/csharp/language-reference/keywords/method-parameters

Passagem por valor e por referência - Curso Programação Completo C# - Aula 25
ref
https://docs.microsoft.com/pt-br/dotnet/csharp/language-reference/keywords/ref#passing-an-argument-by-reference
Passing Parameters
https://docs.microsoft.com/pt-br/dotnet/csharp/programming-guide/classes-and-structs/passing-parameters
Passing Reference-Type Parameters
https://docs.microsoft.com/pt-br/dotnet/csharp/programming-guide/classes-and-structs/passing-reference-type-parameters

Argumento out - Curso Programação Completo C# - Aula 26
out
https://docs.microsoft.com/pt-br/dotnet/csharp/language-reference/keywords/out
out parameter modifier
https://docs.microsoft.com/pt-br/dotnet/csharp/language-reference/keywords/out-parameter-modifier

Argumento params - Curso Programação Completo C# - Aula 27
params
https://docs.microsoft.com/pt-br/dotnet/csharp/language-reference/keywords/params

Classes e Objetos - Curso Programação Completo C# - Aula 28
https://docs.microsoft.com/pt-br/dotnet/csharp/programming-guide/classes-and-structs/classes

Classes and structs
https://docs.microsoft.com/pt-br/dotnet/csharp/programming-guide/classes-and-structs/
Built-in reference types
https://docs.microsoft.com/pt-br/dotnet/csharp/language-reference/builtin-types/reference-types
Objects
https://docs.microsoft.com/pt-br/dotnet/csharp/programming-guide/classes-and-structs/objects
Object-Oriented programming
https://docs.microsoft.com/pt-br/dotnet/csharp/tutorials/intro-to-csharp/object-oriented-programming

Construtores e Destrutores - Curso Programação Completo C# - Aula 29
Constructors
https://docs.microsoft.com/pt-br/dotnet/csharp/programming-guide/classes-and-structs/constructors
Finalizers
https://docs.microsoft.com/pt-br/dotnet/csharp/programming-guide/classes-and-structs/destructors
Using Constructors
https://docs.microsoft.com/pt-br/dotnet/csharp/programming-guide/classes-and-structs/using-constructors#c-language-specification
Instance Constructors
https://docs.microsoft.com/pt-br/dotnet/csharp/programming-guide/classes-and-structs/instance-constructors
Private Constructors
https://docs.microsoft.com/pt-br/dotnet/csharp/programming-guide/classes-and-structs/private-constructors
Static Constructors
https://docs.microsoft.com/pt-br/dotnet/csharp/programming-guide/classes-and-structs/static-constructors
How to write a copy constructor
https://docs.microsoft.com/pt-br/dotnet/csharp/programming-guide/classes-and-structs/how-to-write-a-copy-constructor

Sobrecarga de Construtores - Curso Programação Completo C# - Aula 30
Eu não achei isso na documentação da Microsoft.

Classes static - Curso Programação Completo C# - Aula 31
https://docs.microsoft.com/pt-br/dotnet/csharp/programming-guide/classes-and-structs/static-classes-and-static-class-members
static
https://docs.microsoft.com/pt-br/dotnet/csharp/language-reference/keywords/static
Diretiva using static
https://docs.microsoft.com/pt-br/dotnet/csharp/language-reference/keywords/using-static

This - Curso Programação Completo C# - Aula 32
https://docs.microsoft.com/pt-br/dotnet/csharp/language-reference/keywords/this

Public vs Private - Curso Programação Completo C# - Aula 33
public
https://docs.microsoft.com/pt-br/dotnet/csharp/language-reference/keywords/public
private
https://docs.microsoft.com/pt-br/dotnet/csharp/language-reference/keywords/private
Conteúdo Extra
https://docs.microsoft.com/pt-br/dotnet/csharp/programming-guide/classes-and-structs/access-modifiers

Herança - Curso Programação Completo C# - Aula 34
Herança
https://docs.microsoft.com/pt-br/dotnet/csharp/programming-guide/classes-and-structs/inheritance
Herança em C# e .NET
https://docs.microsoft.com/pt-br/dotnet/csharp/tutorials/inheritance

Cadeia de herança e Construtor da classe base - Curso Programação Completo C# - Aula 35
Não achei na documentação

Membros Protected - Curso Programação Completo C# - Aula 36

Protected

https://docs.microsoft.com/pt-br/dotnet/csharp/language-reference/keywords/protected

private protected

https://docs.microsoft.com/pt-br/dotnet/csharp/language-reference/keywords/private-protected

Herança/Ordem de execução dos construtores - Curso Programação Completo C# - Aula 37

Não achei na documentação

Métodos virtuais - Curso Programação Completo C# - Aula 38

https://docs.microsoft.com/pt-br/dotnet/csharp/language-reference/keywords/virtual

Override

https://docs.microsoft.com/pt-br/dotnet/csharp/language-reference/keywords/override

Classes e métodos abstratos - Curso Programação Completo C# - Aula 39

Abstract and Sealed Classes and Class Members

https://docs.microsoft.com/pt-br/dotnet/csharp/programming-guide/classes-and-structs/abstract-and-sealed-classes-and-class-members

How to define abstract properties

https://docs.microsoft.com/pt-br/dotnet/csharp/programming-guide/classes-and-structs/how-to-define-abstract-properties

abstract

https://docs.microsoft.com/pt-br/dotnet/csharp/language-reference/keywords/abstract

Classe Sealed - Curso Programação Completo C# - Aula 40

https://docs.microsoft.com/pt-br/dotnet/csharp/language-reference/keywords/sealed

Acessors GET e SET - Curso Programação Completo C# - Aula 41

Properties

https://docs.microsoft.com/pt-br/dotnet/csharp/programming-guide/classes-and-structs/properties

get

https://docs.microsoft.com/pt-br/dotnet/csharp/language-reference/keywords/get

set

https://docs.microsoft.com/pt-br/dotnet/csharp/language-reference/keywords/set

Indexadores de Classes - Curso Programação Completo C# - Aula 42
Indexers
https://docs.microsoft.com/pt-br/dotnet/csharp/programming-guide/indexers/
Using indexers
https://docs.microsoft.com/pt-br/dotnet/csharp/programming-guide/indexers/using-indexers
Indexers in Interfaces
https://docs.microsoft.com/pt-br/dotnet/csharp/programming-guide/indexers/indexers-in-interfaces
Comparison Between Properties and Indexers
https://docs.microsoft.com/pt-br/dotnet/csharp/programming-guide/indexers/comparison-between-properties-and-indexers
Restricting Accessor Accessibility
https://docs.microsoft.com/pt-br/dotnet/csharp/programming-guide/classes-and-structs/restricting-accessor-accessibility

Interfaces - Curso Programação Completo C# - Aula 43
https://docs.microsoft.com/pt-br/dotnet/csharp/language-reference/language-specification/interfaces
Interfaces
https://docs.microsoft.com/pt-br/dotnet/csharp/programming-guide/interfaces/
interface
https://docs.microsoft.com/pt-br/dotnet/csharp/language-reference/keywords/interface

Struct - Curso Programação Completo C# - Aula 44
Structure types
https://docs.microsoft.com/pt-br/dotnet/csharp/language-reference/builtin-types/struct
Structs
https://docs.microsoft.com/pt-br/dotnet/csharp/language-reference/language-specification/structs
Write safe and efficient C# code
https://docs.microsoft.com/pt-br/dotnet/csharp/write-safe-efficient-code

Métodos que retornam objetos - Curso Programação Completo C# - Aula 46
https://social.msdn.microsoft.com/Forums/pt-BR/e3779982-89db-4383-96a3-03e84f1a3c03/metodo-que-retorna-um-objeto?forum=vscsharppt

Sobrecarga de métodos - Curso Programação Completo C# - Aula 47
Não achei na documentação
Recursividade - Curso Programação Completo C# - Aula 48
Não achei na documentação
Métodos e Variáveis estáticos (static) - Curso Programação Completo C# - Aula 49
Não achei na documentação

Delegates - Curso Programação Completo C# - Aula 50
Delegates
https://docs.microsoft.com/pt-br/dotnet/csharp/programming-guide/delegates/
Delegate Classe
https://docs.microsoft.com/pt-br/dotnet/api/system.delegate?view=netcore-3.1
Using Delegates
https://docs.microsoft.com/pt-br/dotnet/csharp/programming-guide/delegates/using-delegates
Introduction to Delegates
https://docs.microsoft.com/en-us/dotnet/csharp/delegates-overview
Action Delegate
https://docs.microsoft.com/en-us/dotnet/api/system.action?view=netcore-3.1
delegate operator
https://docs.microsoft.com/pt-br/dotnet/csharp/language-reference/operators/delegate-operator
How to declare, instantiate, and use a Delegate
https://docs.microsoft.com/pt-br/dotnet/csharp/programming-guide/delegates/how-to-declare-instantiate-and-use-a-delegate
Delegates
https://docs.microsoft.com/pt-br/dotnet/csharp/language-reference/language-specification/delegates
Delegates with Named vs. Anonymous Methods
https://docs.microsoft.com/pt-br/dotnet/csharp/programming-guide/delegates/delegates-with-named-vs-anonymous-methods
Delegates and lambdas
https://docs.microsoft.com/pt-br/dotnet/standard/delegates-lambdas

Argumentos de entrada do programa - Curso Programação Completo C# - Aula 51
Isso já foi explicado

Exceções - Try Catch Finally - P1 - Curso Programação Completo C# - Aula 52

try-finally
https://docs.microsoft.com/pt-br/dotnet/csharp/language-reference/keywords/try-finally

Try-catch-finallyhttps://docs.microsoft.com/pt-br/dotnet/csharp/language-reference/keywords/try-catch-finally

How to use finally blocks
https://docs.microsoft.com/pt-br/dotnet/standard/exceptions/how-to-use-finally-blocks

Best practices for exceptions
https://docs.microsoft.com/pt-br/dotnet/standard/exceptions/best-practices-for-exceptions

Try-catch
https://docs.microsoft.com/pt-br/dotnet/csharp/language-reference/keywords/try-catch

Exception Handling
https://docs.microsoft.com/pt-br/dotnet/csharp/programming-guide/exceptions/exception-handling

Exceptions and Exception Handling
https://docs.microsoft.com/pt-br/dotnet/csharp/programming-guide/exceptions/

Try-finally
https://docs.microsoft.com/pt-br/cpp/cpp/try-finally-statement?view=msvc-160

Try-catch-finallyhttps://docs.microsoft.com/sk-sk/dotnet/csharp/language-reference/keywords/try-catch-finally

Exception Classe
https://docs.microsoft.com/pt-br/dotnet/api/system.exception?view=net-5.0

Handling and throwing exceptions in .NET
https://docs.microsoft.com/pt-br/dotnet/standard/exceptions/

Creating and Throwing Exceptions
https://docs.microsoft.com/pt-br/dotnet/csharp/programming-guide/exceptions/creating-and-throwing-except ions

How to explicitly throw exceptions
https://docs.microsoft.com/pt-br/dotnet/standard/exceptions/how-to-explicitly-throw-exceptions

Use exceptions
https://docs.microsoft.com/pt-br/dotnet/csharp/programming-guide/exceptions/using-exceptions

How to create user-defined exceptions with localized exception messages
https://docs.microsoft.com/pt-br/dotnet/standard/exceptions/how-to-create-localized-exception-messages

How to create user-defined exceptions
https://docs.microsoft.com/pt-br/dotnet/standard/exceptions/how-to-create-user-defined-exceptions

Exceções - Try Catch Finally - P2 - Curso Programação Completo C# - Aula 53
Isso já foi explicado

Namespaces - Curso Programação Completo C# - Aula 54
Namespaces
https://docs.microsoft.com/pt-br/dotnet/csharp/programming-guide/namespaces/
System Namespace
https://docs.microsoft.com/pt-br/dotnet/api/system?view=dotnet-plat-ext-5.0
namespace
https://docs.microsoft.com/pt-br/dotnet/csharp/language-reference/keywords/namespace
Using namespaces
https://docs.microsoft.com/pt-br/dotnet/csharp/programming-guide/namespaces/using-namespaces
How to use the My namespace
https://docs.microsoft.com/pt-br/dotnet/csharp/programming-guide/namespaces/how-to-use-the-my-namespace

Coleção Dictionary - Curso Programação Completo C# - Aula 55
How to initialize a dictionary with a collection initializer
https://docs.microsoft.com/pt-br/dotnet/csharp/programming-guide/classes-and-structs/how-to-initialize-a-dictionary-with-a-collection-initializer
Dictionary<TKey,TValue> Classe
https://docs.microsoft.com/pt-br/dotnet/api/system.collections.generic.dictionary-2?view=net-5.0
LinkedList<T> Classe
https://docs.microsoft.com/pt-br/dotnet/api/system.collections.generic.linkedlist-1?view=net-5.0
LinkedListNode<T> Classe
https://docs.microsoft.com/pt-br/dotnet/api/system.collections.generic.linkedlistnode-1?view=net-5.0
When to use generic collections
https://docs.microsoft.com/pt-br/dotnet/standard/collections/when-to-use-generic-collections
Coleção LinkedList / Lista duplamente encadeada - Curso Programação Completo C# - Aula 56
Coleção Queue (Fila) - Curso Programação Completo C# - Aula 59

[redacted]