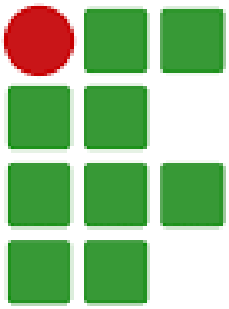


INSTITUTO FEDERAL
São Paulo
Câmpus Cubatão

ALLANIS ALVES DA SILVA
BEATRIZ ALENCAR GREGÓRIO DA SILVA
GABRIELLE APARECIDA RODRIGUES DA SILVA
GUILHERME DOS SANTOS BARBOSA
JULIA FERNANDA SANTANA AQUINO
KHAUÃ NASCIMENTO DINIZ DE OLIVEIRA
LEONARDO DOMINGOS DO NASCIMENTO
LEONARDO XAVIER DO NASCIMENTO
MATHEUS CARNEIRO PALMEIRA
THAYS REGINA DA SILVA RODRIGUES
VALENTINE STROBEL VASQUEZ MUNITA

LINGUAGEM F#

CUBATÃO – SP
2021

ALLANIS ALVES DA SILVA
BEATRIZ ALENCAR GREGÓRIO DA SILVA
GABRIELLE APARECIDA RODRIGUES DA SILVA
GUILHERME DOS SANTOS BARBOSA
JULIA FERNANDA SANTANA AQUINO
KHAUÃ NASCIMENTO DINIZ DE OLIVEIRA
LEONARDO DOMINGOS DO NASCIMENTO
LEONARDO XAVIER DO NASCIMENTO
MATHEUS CARNEIRO PALMEIRA
THAYS REGINA DA SILVA RODRIGUES
VALENTINE STROBEL VASQUEZ MUNITA

LINGUAGEM F#

Trabalho de Conclusão de Curso apresentado como requisito parcial para obtenção do diploma tecnólogo, pelo Curso de informática do Instituto Federal de São Paulo.

Orientador: Prof.º Maurício Neves Asenjo

CUBATÃO – SP
2021

RESUMO

No decorrer da história humana, se é nítido a evolução dada às tecnologias, frente à facilitação na resolução dos problemas: desde as pedras sendo afiadas para a caça, até a descoberta do fogo, dentre tantas outras. As linguagens de programação não são diferentes: foram criadas para que, a partir de máquinas, pudesse se utilizar meios e sequências lógicas a fim de solucionar problemas complexos. Nos tempos em que as máquinas começam a reger a sociedade, a interação entre humano-máquina torna-se cada vez mais necessária. Os dialetos entre este novo meio de comunicação, são os que conhecemos hoje como linguagens de programação. Para seu aperfeiçoamento, foram desenvolvidas várias linguagens, cada qual com sua finalidade específica e, neste estudo, aprofunda-se na linguagem F#, que lida com problemas complexos, com sua linha de código sendo de simples execução e manutenção, dado o fato de seus comandos serem considerados intuitivos.

Palavras-chave: linguagens de programação; programação funcional; linguagem F#.

ABSTRACT

Throughout human history, the evolution given to technologies is clear in the face of facilitating the resolution of problems: from stones being sharpened for hunting, to the discovery of fire, among many others that are part of evolution. Programming languages are no different: they were created so that, from machines, one could use means and logical sequences in order to solve complex problems. In times when machines are beginning to rule society, the interaction between human-machines becomes more and more necessary. The dialects between this new means of communication are what we know today as programming languages. For its improvement, several languages were developed, each with its specific purpose and, in this study, it goes deeper into the F# language, which deals with complex problems, with its line of code being simple to execute and maintain, given the fact that its commands to be considered intuitive.

Key words: programming languages; functional programming; F# language.

SUMÁRIO

1 INTRODUÇÃO 7
1.1 HISTÓRICO E PRINCIPAIS CARACTERÍSTICAS 7
1.2 POSSIBILIDADES DE IDE'S 8
1.3 DEFINIÇÃO DA IDE ESCOLHIDA A SER UTILIZADA PARA USO DA LINGUAGEM F# 10
1.4 PROCESSO DE DOWNLOAD E INSTALAÇÃO 10
1.5 EXECUÇÃO DE "HELLO WORLD!" 14
2 COMANDOS DE E/S VIA CONSOLE 15
2.1 OPERADORES 15
2.1.1 OPERADORES ARITMÉTICOS 17
2.1.2 OPERADORES BOOLEANOS 20
2.1.3 OPERADORES DE COMPARAÇÃO 21
2.1.4 OPERADORES BIT A BIT 23
2.2 FUNÇÕES MATEMÁTICAS 26
2.3 DECISÃO LÓGICA 28
2.4 LAÇOS DE REPETIÇÃO E ARRAYS 30
2.5 DESENVOLVIMENTO DE PROGRAMAS 33
2.6 LAÇOS DE REPETIÇÃO: EXERCÍCIOS RESOLVIDOS 39
2.7 ARRAYS, VETORES E MATRIZES: EXERCÍCIOS RESOLVIDOS 45
3 CONEXÃO DE FORMULÁRIOS (FORMS) COM A LINGUAGEM F# 54
3.1 CARACTERÍSTICAS DOS OBJETOS UTILIZADOS NOS FORMS 63
3.2 CRIANDO FORMS EM F# 64
4 CRIANDO CLASSES EM F# 74
5 O QUE É BANCO DE DADOS? 91
5.1 O QUE É CRUD? 92
5.2 TUTORIAL – INSTALAÇAO DO BANCO DE DADOS AZURE 93
5.3 CONECTANDO O AZURE AO PROGRAMA EM F# 97

5.4 CRIANDO BANCO DE DADOS EM F# 99
6 APLICAÇÃO DE API 106
6.1 PROVEDORES DE TIPO 106
6.2 DIFERENÇA ENTRE CONSOLE E WINDOWS FORMS 107
6.3 EXEMPLO DE CÓDIGO 108
REFERÊNCIAS 111

1 INTRODUÇÃO

No estudo apresentado será retratado de maneira assídua a linguagem F#. A linguagem referida é uma linguagem funcional, o que significa que as expressões de sua programação são tratadas como objetos. Objetos esses que são imutáveis, diferentemente dos códigos geridos a orientação a objetos, onde os estados das variáveis e/ou objetos criados podem sofrer mutações em seu conteúdo ao longo do código, o que expõe ao risco de sobrescrita ou 'roubo' de valores no resultado de uma variável e/ou objeto.

As informações sobre as vantagens/desvantagens, definição, história, seu uso, dentre outras informações mais detalhadas serão apresentados de modo a que o entendimento seja de fácil desenvoltura, ilustrando conceitos de programação, características e especificações desta linguagem e suas possíveis IDE's.

1.1 HISTÓRICO E PRINCIPAIS CARACTERÍSTICAS

F# foi desenvolvido em 2005 pela pesquisa da Microsoft. A linguagem, essencialmente, era uma implementação .Net de OCaml, pois combinava a sintaxe e o poder da linguagem funcional com as milhares de funções de biblioteca disponíveis em todas as linguagens .NET. Após o lançamento inicial da linguagem em 2005 sob a licença Apache, ela passou por muitas mudanças e os desenvolvedores fizeram várias versões que eram melhores em muitos aspectos do que a anterior. O lançamento sob a licença Apache tornou a linguagem de código aberto, o que significa que ela pode ser usada, modificada e distribuída sem pagar royalties ao desenvolvedor. A primeira versão do F# foi 1.x, lançada em maio de 2005. Esta versão era compatível apenas com o Windows e implantava um runtime .NET 1.0-3.5. A principal fraqueza desta versão era a base estreita da plataforma. Esse problema foi resolvido nas versões sucessivas e Linux e OS X foram adicionados às plataformas suportadas na versão 2.0, lançada em abril de 2010. A próxima versão, F# 3.0, apresentava a adição de JavaScript e GPU nas plataformas suportadas. Esta versão foi lançada em 2012. A versão estável mais recente é a versão 4.0, lançada em 20 de julho de 2015.

Suas principais características são:

- Linguagem funcional;
- Sintaxe fluida, direta e simples (devido as referências as outras linguagens de programação como C# e OCaml);

- Possui comandos intuitivos;
- Atua com dois tipos de sintaxe (verbosa e a leve):

I. Leve: é caracterizada por ser curta e utilizar a endentação por espaços para determinar o início e fim de um bloco de código;

II. Verbosa: não é muito utilizada, usa das palavras “begin”, “in” e “end” para delimitar o início e o fim de um bloco.

- Não utiliza ponto e vírgula (;) na conclusão da linha de código;
- É multiplataforma;
- Tem funções de primeira classe;
- Programação assíncrona;
- Tipos de dados avançados.

1.2 POSSIBILIDADES DE IDE’S

As IDE’s nos permitem fazer aplicações/criações gráficas de software que concedem aos desenvolvedores auxílio em questão de técnicas já integradas na mesma, sem precisar integrar manualmente diversos utilitários. Abaixo citaremos exemplos de IDE’s que utilizam a linguagem F#, que provê flexibilidade, desempenho e legibilidade para criação:

- **Visual Studio**

O Visual Studio é uma IDE (Integrated Development Environment) da Microsoft. É um software que permite os usuários escreverem códigos em uma vasta possibilidade de linguagens, entre elas está C#, F#, Visual Basic e C++. Nele é possível desenvolver aplicativos da Web, para desktops e mobile. Essa ferramenta possibilita para os usuários funcionalidades que facilitam na programação, como, Visual Studio Mobile Center (nele é possível criar monitorar e testar aplicativos – Android, iOS e Windows -), Visual Studio Team Services (que possibilita criar um ambiente de programação para equipes), Visual Studio Dev Essentials (através desse pacote podemos criar aplicativos para diversas plataformas, como o Windows, Linux e macOS), entre outros.

- **Visual Studio Code**

O Visual Studio Code foi lançado no ano de 2015 pela Microsoft. É um editor de código de ferramenta leve e multiplataforma, além disso possui a possibilidade de trabalhar com mais de trinta linguagens de programação.

O VS Code é um aplicativo gratuito e está disponível para Windows, Mac e Linux. Suas principais ferramentas são: edição focalizada em código, navegação de código, compreensão de código, depuração, controle de versão Git, ASP.NET 5 e Node.js.

- **Ionide:**

O Ionide é um projeto de código aberto da comunidade F# e dentro do VS Code ele funciona como um plug in com potencial em ferramentas de plataforma cruzada F#, que transfigura o VS Code em uma completa plataforma IDE F#. Foi implementado recentemente, por meados de 2014/2015. Essa ferramenta leva proficuidades existentes da linguagem F# para dentro do VS Code.

Pode ser instalado no Linux, OS X e Windows, usando o instalador do pacote Atom. Uma coisa muito elogiada sobre o desenvolvimento de plug-ins para Atom é sua flexibilidade, logo, o Ionide pôde fornecer suporte no editor para outras ferramentas populares e interessantes usadas e criadas pela comunidade F#.

- **MonoDevelop**

MonoDevelop é um ambiente de desenvolvimento integrado de código aberto compatível para Linux Mac e Windows, porém, seu foco principal é o desenvolvimento de projetos que usam estrutura mono e .NET framework. Ele contém funções parecidas com as do VS e NetBeans, como sua interface gráfica tanto para web quanto para usuário e possui uma API (Programa de Aplicação Interface) que inclui métodos para acessar bancos de dados projetados em MySql.

Ele começou como sendo SharpDevelop, mas hoje é uma parte independente e possui muitos recursos que vão bem além daquilo que o SD possui, tendo sua aquisição gratuita.

- **JetBrains Rider**

O JetBrains Rider é uma IDE .NET. Como oferece suporte ao .NET Core multiplataforma, .NET Framework e a projetos baseados em Mono o usuário consegue

desenvolver diversos tipos de aplicativos, como aplicativos de desktop .NET. Ele pode ser executado em diversas plataformas como o Windows e Linux.

Entre os recursos que esta plataforma oferece estão, análise de código, refatoração, executor de testes de unidade, banco de dados e SQL, navegação e pesquisa, tecnologias de front-end, entre outros. Além disso, o Rider possui suporte para o Docker, para C#, desenvolvimento Web, depurador, desenvolvimento de jogos, entre outros.

1.3 DEFINIÇÃO DA IDE ESCOLHIDA A SER UTILIZADA PARA USO DA LINGUAGEM F#

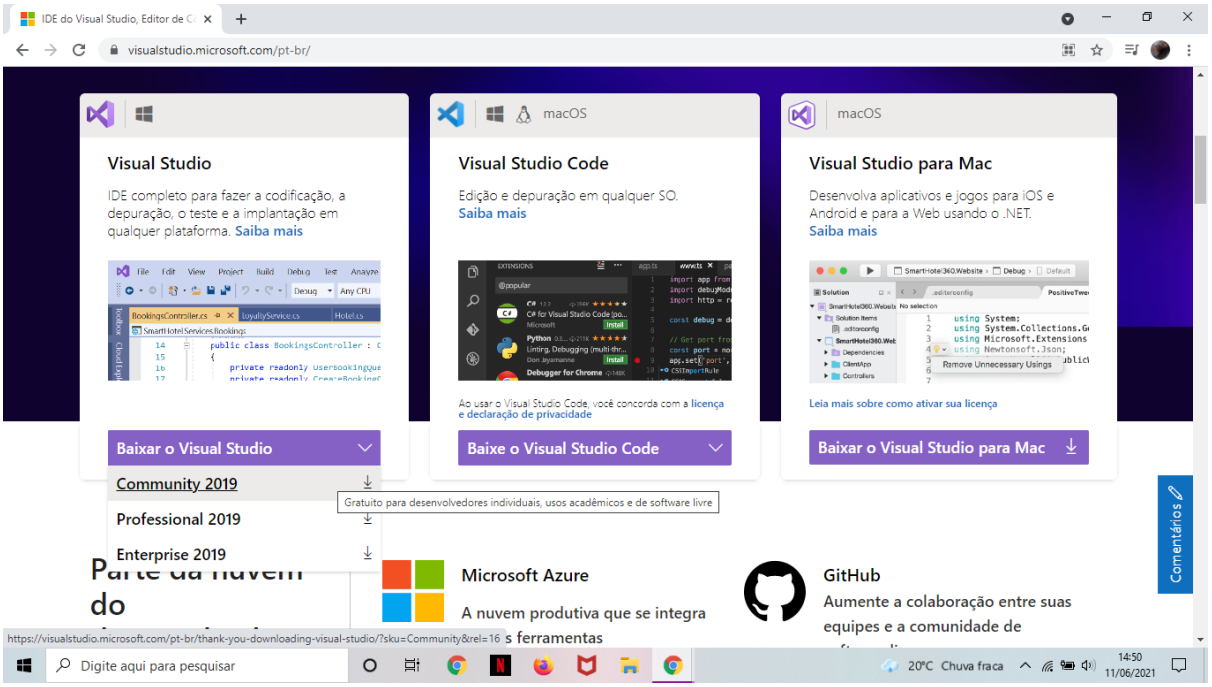
O editor Visual Studio Code foi lançado pela Microsoft em 2015, com o propósito de dar melhores condições ao desenvolvimento de aplicações web. Contemplando os softwares da Windows, Linux e macOS, o Visual Studio Code, ou, como é comumente chamado, o VS Code é um editor de código-fonte multiplataforma, isto é, um Ambiente de Desenvolvimento Integrado (ou IDE), que possui diversas funcionalidades que facilitam a vida dos desenvolvedores de aplicações web.

O VS Code é disponibilizado de forma totalmente gratuita, além de possuir open source localizado na plataforma GitHub.

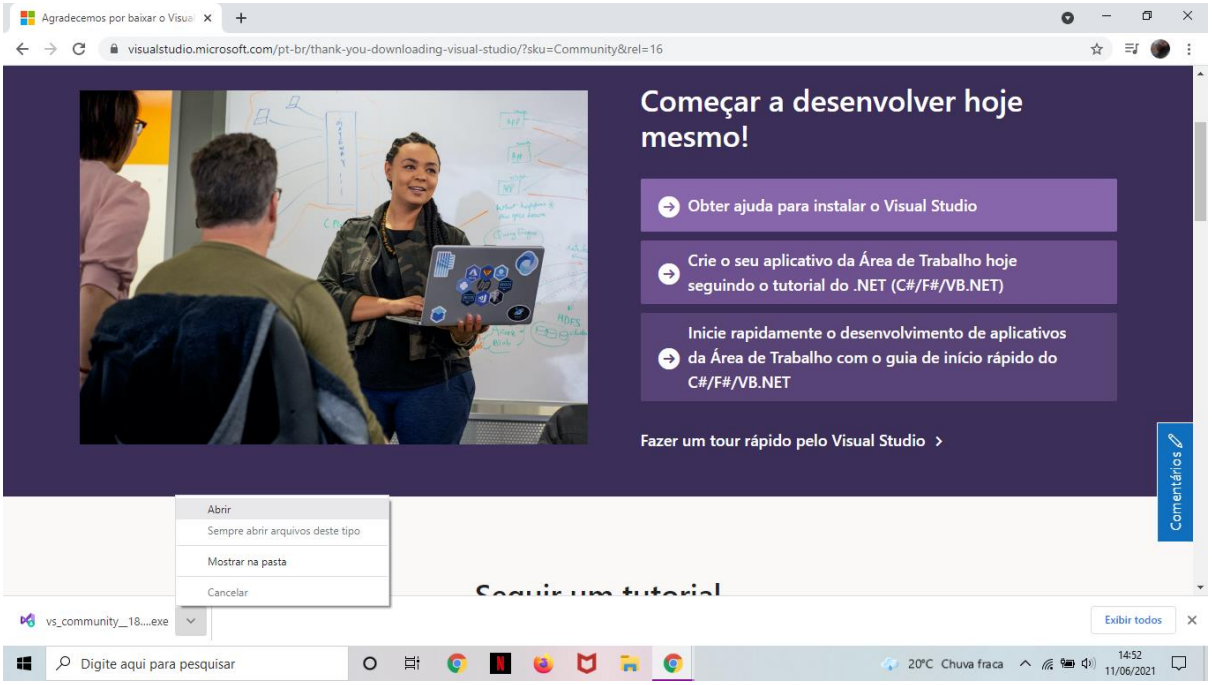
Ao longo desta, alinhado à programação F#, veremos recursos importantes dessa ferramenta de desenvolvimento.

1.4 PROCESSO DE DOWNLOAD E INSTALAÇÃO

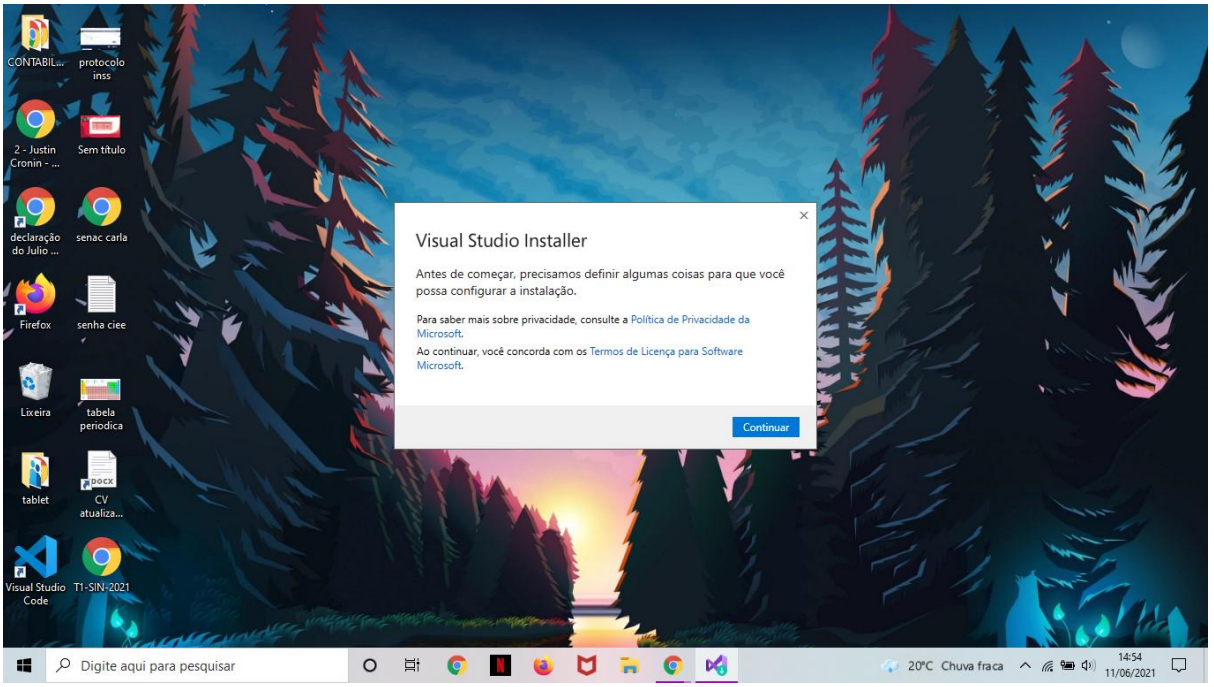
Para iniciar a instalação da IDE, é necessário entrar no link para download (https://visualstudio.microsoft.com/pt-br/), para poder baixar o Visual Studio.



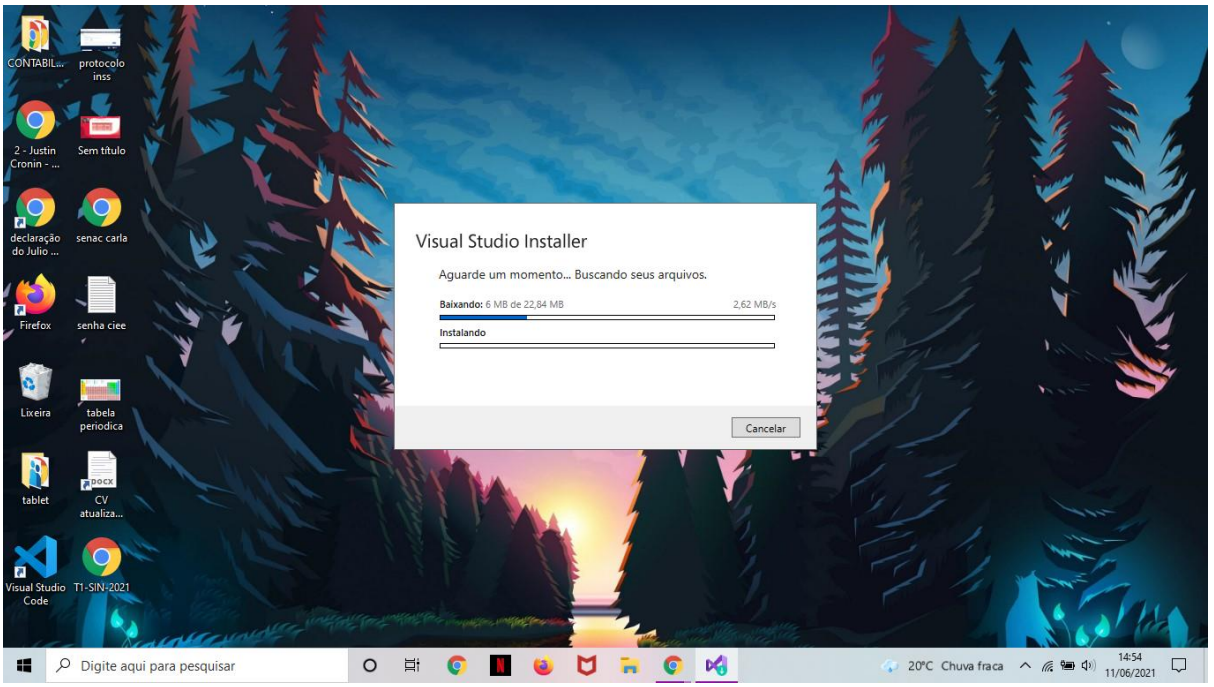
Feito o download do arquivo, clique em “abrir” para começar a instalação.



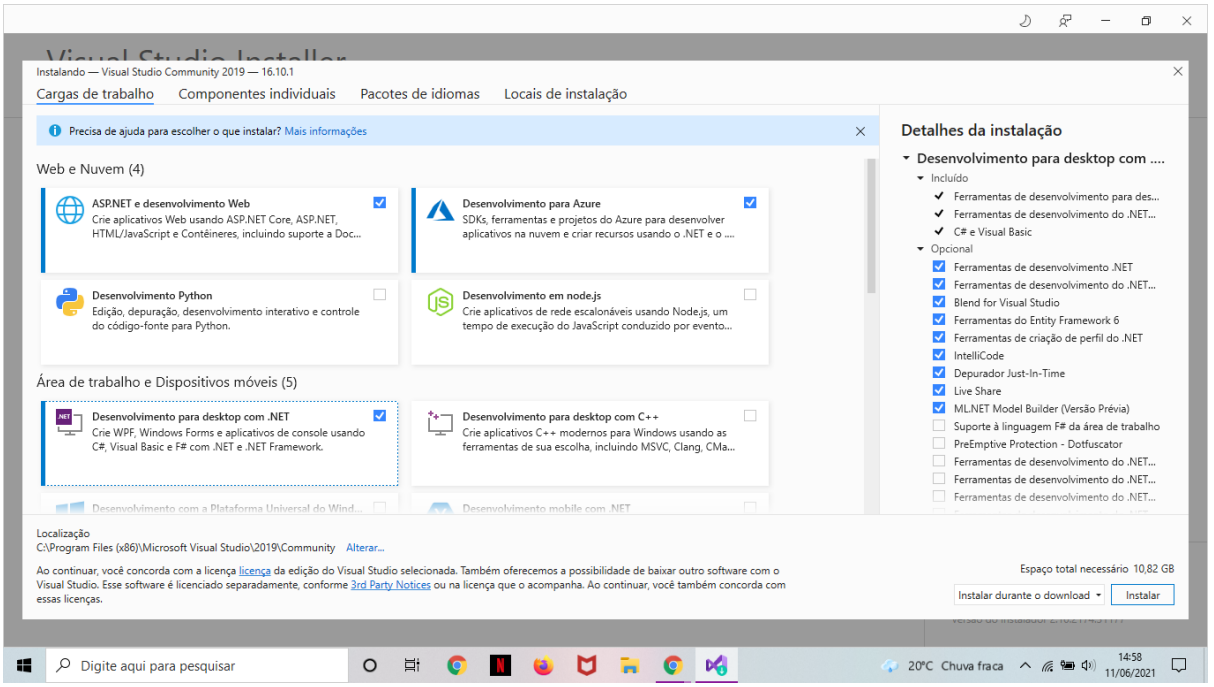
Clique em "continuar” para dar início a configuração da instalação.



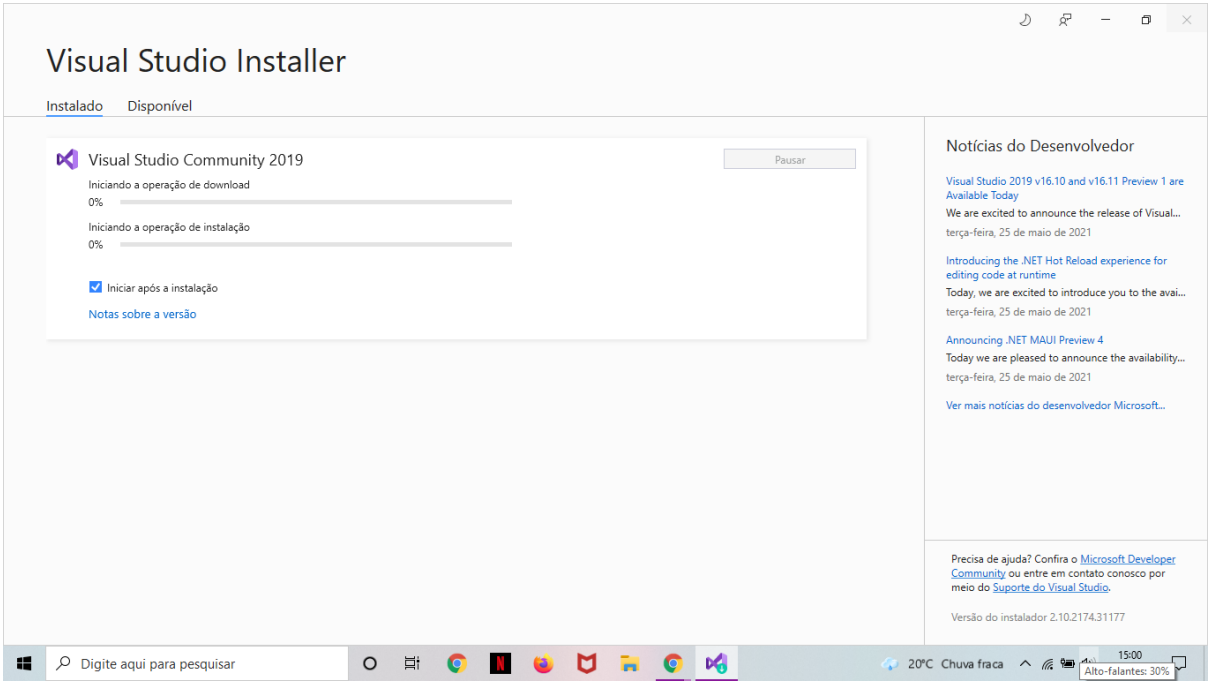
Após clicar em “continuar”, espere por alguns momentos.



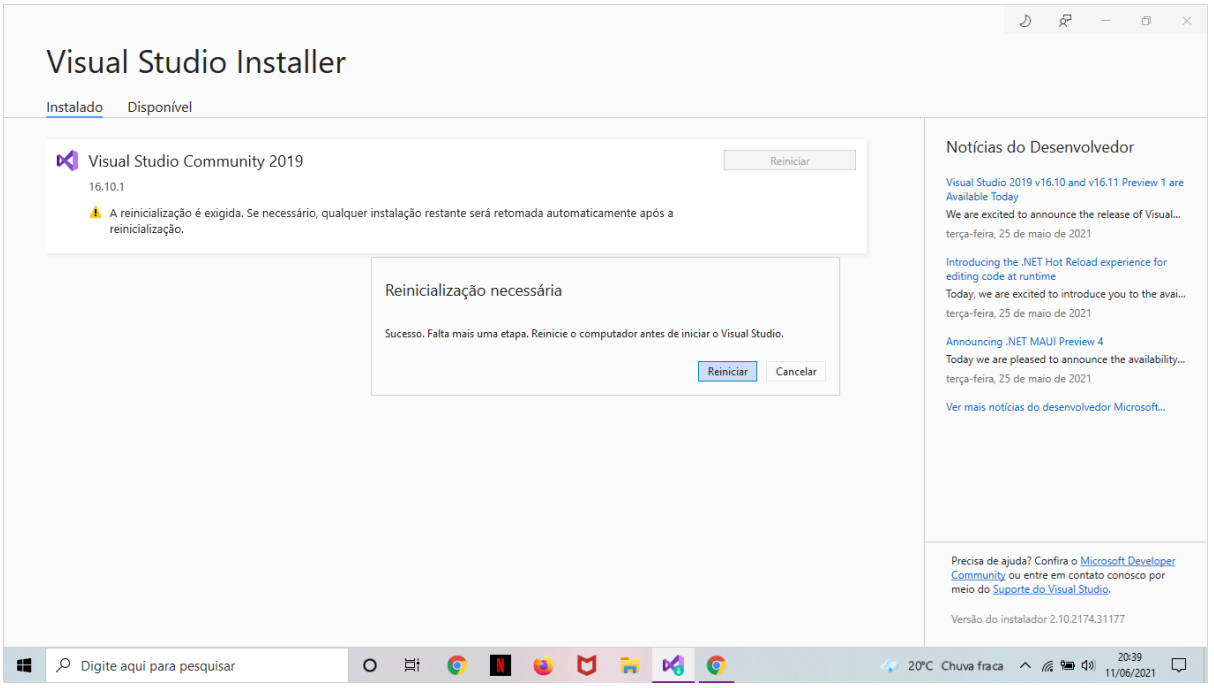
Aqui você deverá selecionar as aplicações que darão suporte a linguagem que será utilizada, como o “Desenvolvimento para desktop com .NET”, e em seguida clique em “instalar”.



Espere a finalização do processo de instalação.



Para terminar, reinicie sua máquina.



Para fazer uso da IDE e da linguagem F# entre com a sua conta da Microsoft.

1.5 EXECUÇÃO DE “HELLO WORLD!”

Importando o “namespace”, que serve para fornecer uma maneira de manter um conjunto de nomes separado de outro. Os nomes de classe declarados em um “**namespace”** não entram em conflito com os mesmos nomes de classe declarados em outro, e para isso utilizaremos o System padrão, logo em seguida utilizaremos o comando “printfn” para mostrar na tela.

Trecho de código:

```
1.  open System  // namespace
2.
3.  printfn “Hello Word! em F#”
4.  Console.ReadKey()
```

Dessa forma conseguimos dar início aos conhecimentos sobre os comandos básicos como o de imprimir na tela, trabalhando com o console.

2 COMANDOS DE E/S VIA CONSOLE

Em C# para fazer uma pergunta ou exibir algum tipo de mensagem ao usuário, utilizamos “Console.Write” ou “Console.Writeline”, o que muda em F#, apesar de possuir a mesma funcionalidade do que é usado em C#, os comandos para exibir na tela as mensagens são:

- printf → exibe na tela a mensagem, e a entrada de valores aparece na mesma linha;
- printfn → exibe na tela a mensagem, e pula uma linha para entrada de valores.

Para exibir na tela alguma variável, você deve usar algum desses argumentos de acordo com o tipo da variável:

%c	Char
%d	Inteiro
%f	Real
%s	String

Em C# para ler e armazenar os valores que entram utilizasse “Console.Readline()”, o que também é utilizado em F#. Assim como em C#, os comandos para limpar tela e terminar depuração são iguais em F#, que respectivamente é representado por:

- Console.Clear();
- Console.ReadKey().

2.1 OPERADORES

F# fornece um conjunto rico e diversificado de operadores que você pode usar com tipos de string, numéricos, de coleções e booleanos. São diversos os operadores definidos na linguagem e suas bibliotecas, então, para mostrar de uma maneira mais prática, em vez de mostrar todos individualmente, saindo de uma maneira “caricata”, mostraremos, em linha de código, como usar e definir alguns operadores mais conhecidos, também utilizados em F#, para uma melhor visualização.

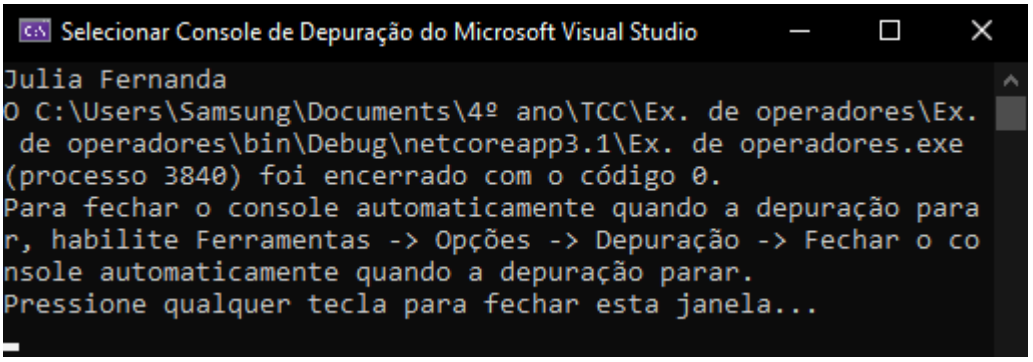
De mesmo modo que C#, os operadores F# estão sobrecarregados, ou seja, quem programa pode usar mais de um tipo com um operador, porém, se opondo ao C#, os dois operandos devem ser do mesmo tipo, caso contrário gerará erro ao compilar. F# também permite que os usuários definam/redefinam os operadores.

O conjunto de regras dos operadores seguem um caminho parecido ao C# em relação à resolução de sobrecarga do operador. Portanto, qualquer classe presente na biblioteca de base .NET Framework (BCL), ou qualquer biblioteca .NET, que foi escrita para oferecer suporte à sobrecarga de operador em C#, terá suporte em F#.

Dando um “pré-início” nos exemplos, vejamos o operador '+'. Ele pode ser usado para conciliar strings, bem como para adicionar um Sistema, como demonstrado a seguir:

```
open System


let nome = "Julia" + " " + "Fernanda"
Console.Write(nome);
```

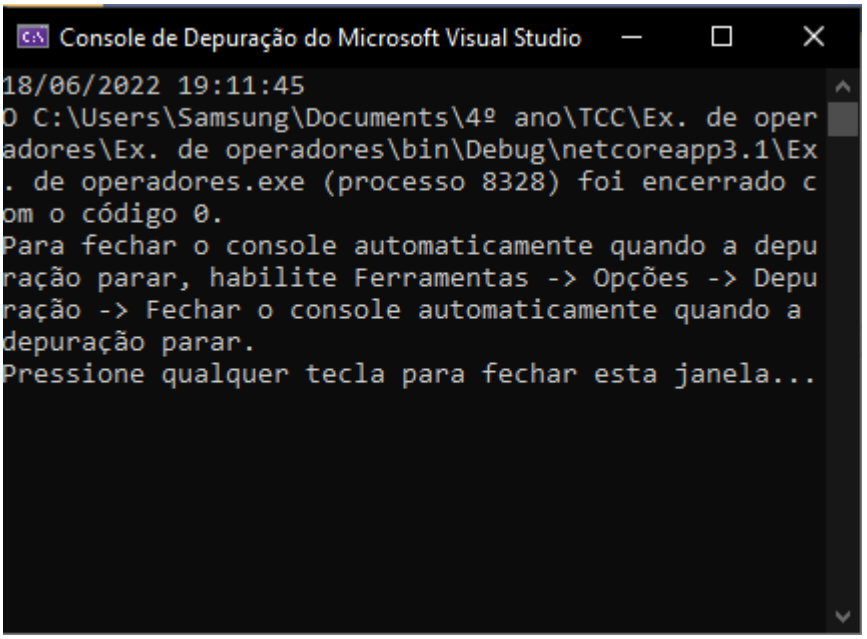
Que terá uma saída assim:



Como citado, com esse operador também é possível adicionar um Sistema, como o System.TimeSpan (que representa um período de tempo) para um System.DateTime (definido como a data e hora atuais a hora da máquina, expressas como a hora local), porque esses tipos oferecem suporte a uma sobrecarga do operador (permite alterar o comportamento de um operador da linguagem):

```
open System


let n = DateTime.Now + new TimeSpan (365, 0, 0, 0, 0)
Console.Write(n)
```

- Depuração:



PARA DEPURAÇÃO, SE LEVA EM CONTA A DATA ATUAL EM QUE FOR TESTAR ESSA LINHA DE CÓDIGO; NA FOTO A DATA CONSTAVA NO DIA EM QUE O CÓDIGO FORA DESENVOLVIDO.

Uma informação a ser levada em consideração, é que F# oferece suporte a dois tipos de operadores: o prefixo e o tipo de “Infix”.

O prefixo é aquele em que o operador precede seu operando (ex.: **-**a), ou seja, vem antes. Já o “infix” aparece entre o primeiro e o segundo operandos; levando dois ou mais argumentos (ex.: a **+** b).

Os ‘tipos’ de operadores do F# podem ser classificadas em:

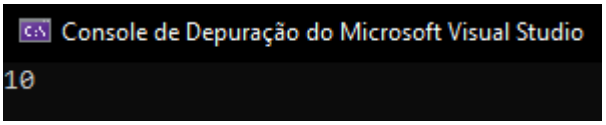
I. Operadores aritméticos;
II. booleanos;
III. de comparação;
IV. bit a bit.

2.1.1 OPERADORES ARITMÉTICOS

Os operadores aritméticos são os que fora denominado anteriormente como sendo os “mais conhecidos”, composto por:

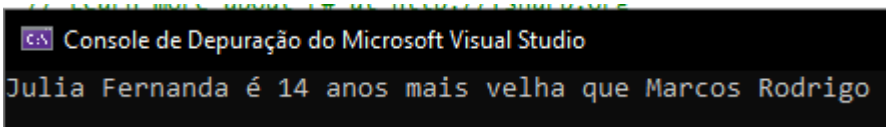
- **Adição (+)**

```
open System

let num1 = 4
let num2 = 6

Console.WriteLine((num1+num2))
```



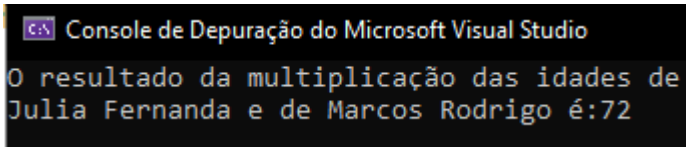
- **Subtração (-)**

```
open System


let nome1 = "Julia" + " " + "Fernanda"

let nome2 = "Marcos" + " " + "Rodrigo"

let idade1 = 18;

let idade2 = 4;

let r = idade1 - idade2

Console.WriteLine(nome1 + " " + "é" + " " + r.ToString() + " "
                  + "anos mais velha que" + " " + nome2 )
```



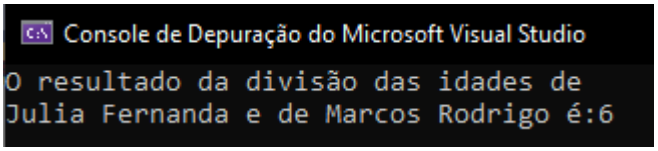
- **Multiplicação (*)**

```
open System


let nome1 = "Julia" + " " + "Fernanda"

let nome2 = "Marcos" + " " + "Rodrigo"

let idade1 = 18;

let idade2 = 4;

//let r = idade1 - idade2

let r = idade1 * idade2

Console.WriteLine("O resultado da multiplicação das idades de\n" + nome1 +
                  " " + "e de" + " " + nome2 + " " + "é:" + r.ToString())
```



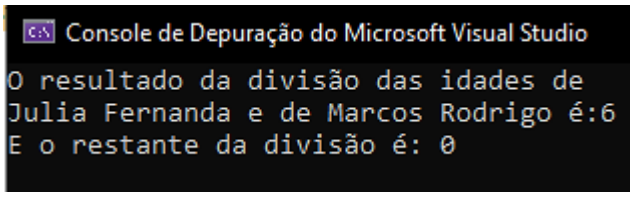
- **Divisão (/)**

```
open System

let nome1 = "Julia" + " " + "Fernanda"

let nome2 = "Marcos" + " " + "Rodrigo"

let idade1 = 18;

let idade2 = 3;

let r = idade1 / idade2

Console.WriteLine("O resultado da divisão das idades de\n" + nome1 +
                  " " + "e de" + " " + nome2 + " " + "é:" + r.ToString())
```



- **Resto (%)**

```
open System

let nome1 = "Julia" + " " + "Fernanda"

let nome2 = "Marcos" + " " + "Rodrigo"

let idade1 = 18;

let idade2 = 3;

let r = idade1 / idade2

let rest = idade1%idade2

Console.WriteLine("O resultado da divisão das idades de\n" + nome1 +
                  " " + "e de" + " " + nome2 + " " + "é:" + r.ToString()
                  + "\nE o restante da divisão é: " + rest.ToString())
```



- **Exponenciação (**)** -usado apenas para ponto flutuante-

```
open System

let nota1 = 6.0

let nota2 = 2.0

let esp = nota1**nota2

Console.WriteLine ("A exponenciação:" +
nota1.ToString() + "^" + nota2.ToString()
+ " \né:" + esp.ToString())
```

```
Console de Depuração do Microsoft Visual Studio
A exponenciação:6^2
é:36
```

2.1.2 OPERADORES BOOLEANOS

Os operadores booleanos localizam registros que contêm os termos correspondentes em um dos campos especificados, ampliando ou reduzindo resultados de busca. Seus componentes são:

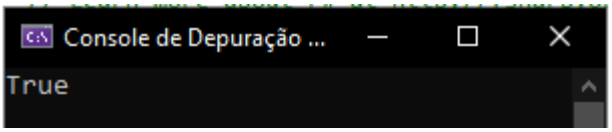
- **AND (&&)**

```
open System

//as duas condições
//diferentes de 0
let a = true
let b = true

//verificação da afirmação,
//caso seja verdade
if a&&b then

//então irá exibir "true"
Console.WriteLine(a&&b)
```

```
Console de Depuração ...
True
```

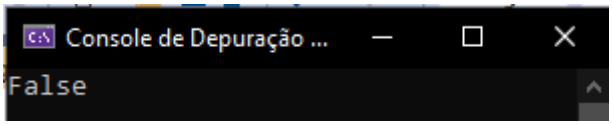
- **OU (||)**

```
open System

//uma das condições
//são diferentes de 0
let a = false
let b = true

//verificação da afirmação,
//caso seja verdade
if a||b then

//então irá exibir "true"
Console.WriteLine(a||b)
```



- **Não (not)**

```
open System

//uma das condições é
//igual a zero
let a = true
let b = false

//verificação da afirmação,
//caso seja verdade
not (a&&b)

//então irá exibir "false"
Console.WriteLine(a&&b)
```



2.1.3 OPERADORES DE COMPARAÇÃO

Como o nome bem direto, os operadores de comparação comparam os valores e realiza determinadas ações de acordo com a resposta, como por exemplo, se alguma ação deverá ser tomada ou não. São eles:

- **Igualdade (=)**

```
open System

let a = 4
let b = 4

if a=b then

Console.WriteLine((a=b))
```

```
Console de Depuração do Microsoft Visual Studio
True
```

- **Diferença (<>)**

```
open System

let a = 4
let b = 2

if a<>b then

Console.WriteLine(a<>b)
```

```
Console de Depuração do Microsoft Visual Studio
True
```

- **Maior que (>)**

```
open System

let a = 4
let b = 2

if a>b then

Console.WriteLine(a>b)
```

```
Console de Depuração do Microsoft Visual Studio
True
```

- **Menor que (<)**

```
open System

let a = 2
let b = 4

if a<b then

Console.WriteLine(a<b)
```

```
Console de Depuração do Microsoft Visual Studio
True
```

- **Maior ou igual (>=)**

```
open System

let a = 6
let b = 4

if a>=b then

Console.WriteLine(a>=b)
```

```
Console de Depuração do Microsoft Visual Studio
True
```

- **Menor ou igual (<=)**

```
open System

let a = 3
let b = 4

if a<=b then

Console.WriteLine(a<=b)
```

```
Console de Depuração do Microsoft Visual Studio
True
```

2.1.4 OPERADORES BIT A BIT

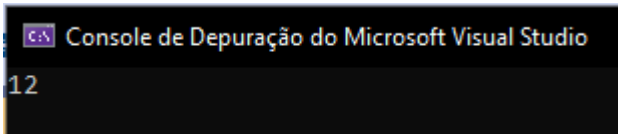
Em relação aos “bit a bit”, são trabalhadas tabelas verdadeiras, trabalhando em bits e realizam operação bit a bit seguindo essa representação:

p	q	p &&& q	p ||| q	p ^^^ q
0	0	0	0	0
0	1	0	1	1
1	1	1	1	0
1	0	0	1	1

Esse método não é muito utilizado, mas, para via de aprendizado, seus operadores são:

- AND (&&&)

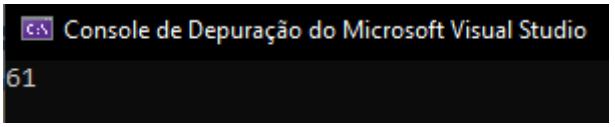
```
open System

let a = 60
let b = 13

Console.WriteLine((a &&& b))
```



12 seguindo a tabela = 0000 1100

- OR (|||)

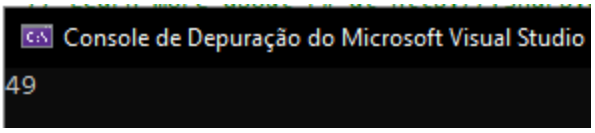
```
open System

let a = 60
let b = 13

Console.WriteLine((a ||| b))
```



61 seguindo a tabela = 0011 1101

- XOR (^^^) (‘X’ → exclusivo)

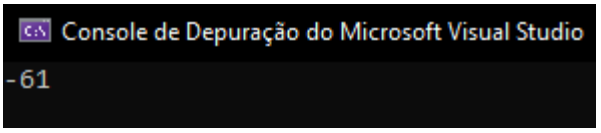
```
open System

let a = 60
let b = 13

Console.WriteLine(a ^^^ b)
```



49 na tabela = 0011 0001

- Inverter (~~~)

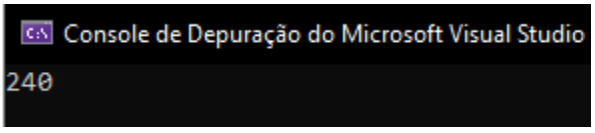
```
open System

let a = 60
let b = 13

Console.WriteLine(~~~ a)
```



(-61 na tabela = 1100 0011 - em complemento 2)

- Deslocamento à esquerda (<<<) - definido pelo número de bits especificado pelo operando à direita.

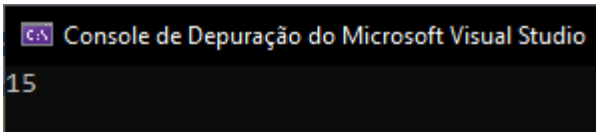
```
open System

let a = 60
let b = 13

Console.WriteLine(a <<< 2)
```



240 na tabela = 1111 0000

- Deslocamento à direita (>>>)

```
open System

let a = 60
let b = 13

Console.WriteLine(a >>> 2)
```



15 na tabela = 0000 1111

2.2 FUNÇÕES MATEMÁTICAS

- **Função Math**

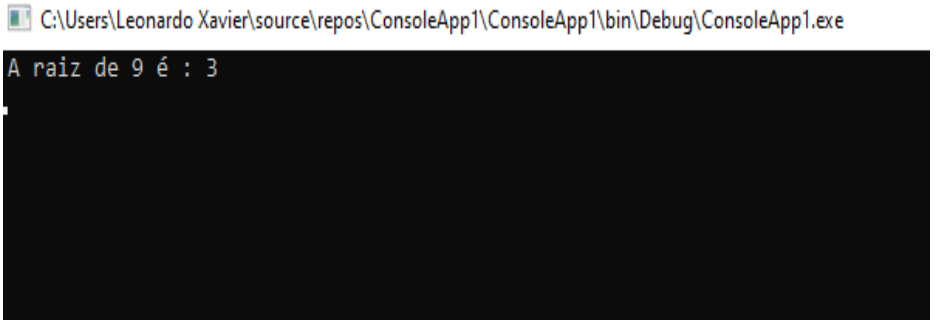
Na matemática, as funções recebem argumentos e resultam em um valor. Já em programação há a inserção de dados (argumentos), que se segmentam em uma resposta numérica. Assim, a função Math é um objeto que dispõe de propriedades e métodos (apresentados na tabela a seguir), que facilitam a programação, e interceptam tanto a linguagem C# quanto F#.

Função	Definição
Math.Abs	Retorna o valor absoluto de um valor passado por parametro.
Math.Acos	Retorna o ângulo a partir do número do cosseno especificado.
Math.Asin	Retorna o ângulo a partir do número do seno especificado.
Math.Atan	Retorna o ângulo a partir do número da tangente especificado.
Math.Cos	Retorna o cosseno do ângulo.
Math.Exp	Retorna "e" elevado a potência especificada.
Math.Log	Retorna o logaritmo de um número e base especificados.
Math.Log10	Retorna o logaritmo (base 10), de um número especificado.
Math.Pi	Retorna a relação entre a circunferência e o diâmetro de um círculo (π)
Math.Pow	Retorna o número inserido elevado a potência especificada.
Math.Sin	Retorna o seno do ângulo.
Math.Sqrt	Retorna a raiz quadrada do número especificado.

- Exemplo de programa usando função “Math.Sqrt” (raiz quadrada) em C#:

```
1   using System;
2   using System.Collections.Generic;
3   using System.Linq;
4   using System.Text;
5   using System.Threading.Tasks;
6
7   namespace ConsoleApp1
8   {
        0 referências
9       class Program
10      {
            0 referências
11          static void Main(string[] args)
12          {
13
14              double numRaiz = 9;
15              double resultadoRaiz;
16              resultadoRaiz = Math.Sqrt(numRaiz);
17
18              //com o valor NOVE declarado para a variável numRaiz, o resultado será TRÊS.
19              Console.WriteLine(" A raiz de " + numRaiz.ToString() + " é : " + resultadoRaiz.ToString());
20
21
22
23              Console.ReadKey();
24          }
25      }
26  }
```

- Resultado da compilação:



2.3 DECISÃO LÓGICA

A sintaxe de decisão lógica da linguagem F# não se diverge muito da sintaxe em C#, podemos perceber isso com os exemplos utilizados paras estruturas de decisão lógica IF e SWITCH-CASE:

- **Condição IF**

A correspondência “if” é uma estrutura de decisão lógica onde é usada para tomada de decisões, onde “se” a condição for verdadeira é executado alguma tarefa, podendo ter o uso do “else” que significa senão, sendo uma opção caso a condição do “if” não se valide (se a condição for verdadeira faça isso, senão, aquilo). Lembrando que é possível usar um “else if” para acrescentar mais de uma condição para o laço.

Em **C#** a sintaxe do “if” não é tão complexa, para iniciar um laço basta digitar o termo if(condicao) e, em seguida, a tarefa entre chaves “{}”. **Exemplo de código:**

```
class Program {
    static void Main(string[] args) {

      int a = 10;
      int b = 20;

      Console.WriteLine("Valor de A: {0}", a);
      Console.WriteLine("Valor de B: {0}", b);

      if(a > 10){
        Console.WriteLine("A é maior");
      }
else if (a < b){
        Console.WriteLine("B é maior");
      }
      else{
        Console.WriteLine("São iguais");
      }
    }
```

- **Compilando:**

Valor de A: 10; Valor de B: 20; B é maior.

Já em **F#** o uso do “if” é um pouco mais simples, não precisando de parênteses nem chaves, podendo ser chamada utilizando o termo “if condicao then”, podendo conter também o “else if” ou “elif” (que são a mesma coisa), ou somente o “else”.

Exemplo de código:

```
open System
let a = 20 |> int
let b = 15 |> int

printfn "Valor de A: %d" a
printfn "Valor de B: %d" b

if a > b then
printfn "A é maior"
elif a < b then
printfn "B é maior"
else
printfn "São iguais"
```

- **Compilando:**

Valor de A: 20; Valor de B: 15; A é maior

- **Condição SWITCH-CASE**

A correspondência Switch-case oferece uma maneira bem mais flexível de testar os dados de acordo com certas condições, realizando alguma tarefa quando a condição é encontrada.

Em C# um switch-case é chamado com o termo “switch (condição)”, sendo dividido em “cases” diferentes para oferecer diversas possibilidades, e para cada delas uma tarefa diferente, exemplo: “case 1, 2” e “default”. Levamos em consideração que ao final de todo case o termo break é utilizado para identificação do final da tarefa.

Em F# o switch-case é chamado com o termo “match condição with” não precisando da digitação do termo case, e sim a digitação direta da digitação, exemplo: 1, 2, _. O termo default (padrão), utilizado em C# tem a mesma funcionalidade de _ em F#, que também não precisa de break.

2.4 LAÇOS DE REPETIÇÃO E ARRAYS

Como sabemos, usamos laços para realizar tarefas que necessitam de um ou mais repetições, são úteis para verificação, validação, atualização e outros diversos recursos. Em linguagens comuns podemos utilizar qualquer tipo de laço, optando pelo que melhor se ajustar a necessidade do código, porém como F# é uma linguagem funcional, temos uma diferença em relação ao que estamos acostumados. Por se tratar de uma linguagem funcional, seus laços necessitam de recursividade, que nada mais é do que funções que invocam a si mesmas, permitindo que uma operação seja realizada várias vezes. Recursividade em programação funcional pode assumir várias formas e é em geral uma técnica mais poderosa que o uso de laços mesmo que seus conceitos se assemelhem, já que em prática podemos sim, notar a diferença.

- **FOR**

O laço “for” é geralmente usado para quando há a necessidade de repetir várias linhas de código por um número fixo de vezes. Assim, os valores iniciais e finais são especificados e o corpo do laço é repetido várias vezes, até que o contador seja menor ou maior que o valor final. E dentro desse laço temos duas ramificações:

- **FOR IN**

Esse loop é usado para iterar sobre as correspondências de um padrão em uma coleção enumerável, como uma expressão de intervalo, sequência, lista, matriz ou outro constructo que dá suporte à enumeração.

Exemplo:

```
let function1() =
for i in 1 .. 10 do
printf "%d " i
printfn ""
function1()
```

A saída do código anterior é uma contagem de números inteiros do 1 ao 10.

- **FOR TO**

Essa expressão é usada para iterar em um loop em um intervalo de valores de uma variável de loop.

Exemplo:

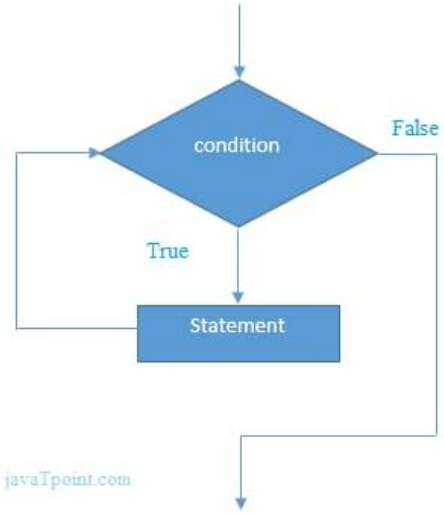
```
// Início do loop simples.
 let function1() =
 for i = 1 to 10 do
 printf "%d " i
printfn ""
 // Esse loop contará ao contrário.
 let function2() =
 for i = 10 downto 1 do
 printf "%d " i
 printfn ""

function1()
function2()
 // Loop para buscar os valores iniciais e finais.
 let beginning x y = x - 2*y
let ending x y = x + 2*y
 let function3 x y =
 for i = (beginning x y) to (ending x y) do
 printf "%d " i
 printfn ""
function3 10 4
```

A saída do código anterior será:

```
1 2 3 4 5 6 7 8 9 10
10 9 8 7 6 5 4 3 2 1
2 3 4 5 6 7 8 9 10 11 12 13 14 15
16 17 18
```

- **WHILE**

O laço “WHILE” é geralmente utilizado para executar um looping temporário que se repete enquanto uma condição teste é verdadeira. Quando a expressão é falsa, o código pula o looping e segue seu fluxo normal.



Exemplo:

```
let lookForValue value maxValue =
let mutable continueLooping = true
let randomNumberGenerator = new Random()
while continueLooping do
// Generate a random number between 1 and maxValue.
let rand = randomNumberGenerator.Next(maxValue)
printf "%d " rand
if rand = value then
printfn "\nFound a %d!" value
continueLooping <- false

lookForValue 10 20
```

A saída do código anterior é um fluxo de números aleatórios entre 1 e 20, o último deles é 10.

- **ARRAY**

Arrays são matrizes de valores do mesmo tipo, ou seja, um conjunto de elementos dentro da mesma variável. São declaradas pelo programador que escolhe também seu tipo e o número máximo de elementos que serão concatenados.

Exemplos (neste caso, criaremos uma matriz com 1 linha e 3 colunas):

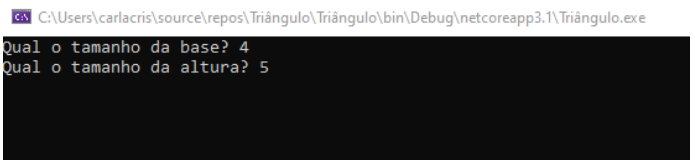
```
        let array1 = [| 1; 2; 3 |]


        //Neste, uma com 3 linhas e 1 coluna.
        let array1 =
         [|
            1
            2
            3
         |]
//E neste, usaremos uma expressão para criar uma
matriz com quadrados inteiros de 1 a 10.
        let array3 = [| for i in 1 .. 10 -> i * i |]
```

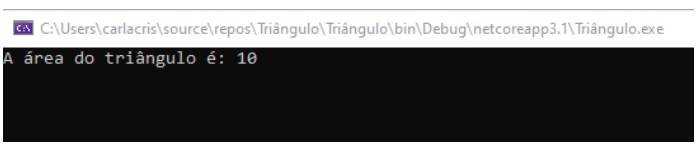
2.5 DESENVOLVIMENTO DE PROGRAMAS

I. A partir da digitação da base e altura de um triângulo o programa deverá calcular sua área e exibi-la no monitor.

```
open System

//Declaração de variáveis
let mutable b = 0
let mutable h = 0
let mutable area = 0

//Entrada e leitura dos valores da base
let entrada() =
printf "Qual o tamanho da base? "
let a = Console.ReadLine()
b <- a |> int

//Entrada e leitura dos valores da altura
printf "Qual o tamanho da altura? "
let c = Console.ReadLine()
h <- c |> int

//Faz o cálculo da área
let calculo() =
Console.Clear()
area <- (b * h) / 2

//Exibe o cálculo da área na tela
printf "A área do triângulo é: %d " area
//Chama as funções
entrada()
calculo()
```

Depuração:

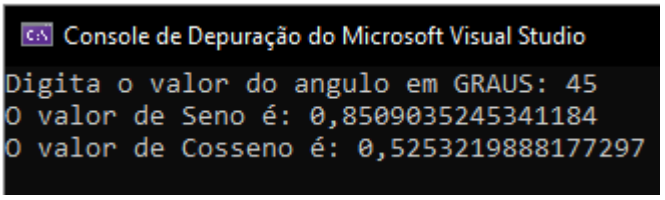




II. O programa deverá pedir para você digitar o valor de um ângulo em graus e na sequência mostrar o valor do seno e cosseno deste ângulo.

```
        open System

        //pedindo valor de entrada ao usuário
        Console.Write("Digita o valor do angulo em GRAUS: ")

        //guardando valor digitado numa variável
        let angulo = Console.ReadLine()

        //variável usada para conversão do valor
        //digitado pelo usuário para ser usado na função
matemática

        let num = Convert.ToDouble(angulo)

        //variável contendo a função matemática do Seno

        let valor1 = Math.Sin(num)
        Console.WriteLine("O valor de Seno é: {0}", valor1)

        //variável contendo a função matemática do Cosseno
        let valor2 = Math.Cos(num)
        Console.WriteLine("O valor de Cosseno é: {0}", valor2)
```

Depuração:



III. O programa deverá solicitar a digitação de suas 4 notas bimestrais, feito isso deverá calcular e exibir a sua média final (média aritmética entre as 4 notas). Feito isso deverá também mostrar as mensagens: “Você está aprovado!”, “Você está reprovado!” ou “Você está de exame” de acordo com o seguinte critério: Média final maior ou igual a seis o aluno está aprovado, menor que três reprovado, entre 3 e 6 de exame.

```
open System

//pedido de entrada dos valores
Console.WriteLine("Digite a primeira nota:")
//armazenamento dos valores digitados em uma variável
let n1 = Console.ReadLine()

//convertendo e armazenando o valor digitado em
//outra variável para ser possível utilizar seu
//valor na conta da média final
let bim1 = Convert.ToDouble(n1)

Console.WriteLine("Digite a segunda nota:");
let n2 = Console.ReadLine()

let bim2 = Convert.ToDouble(n2)

Console.WriteLine("Digite a terceira nota:");
let n3 = Console.ReadLine()

let bim3 = Convert.ToDouble(n3)

Console.WriteLine("Digite a quarta nota:");
let n4 = Console.ReadLine()

let bim4 = Convert.ToDouble(n4)

//variável com a conta da média final
let  media = (bim1 + bim2 + bim3 + bim4)/4.

//exibição da média final
Console.WriteLine("Sua média final é: " + " " + media.ToString())

//1ª condição: se a nota for maior ou igual a 6
if media >= 6.0 then
//então exibirá a mensagem
Console.WriteLine("\nVocê está aprovado! :)")

//2ª condição: se a nota for menor que 3
else
if media < 3.0 then
//então exibirá a mensagem
```

```
Console.WriteLine("\nVocê está reprovado :(")

//3ª condição: se a nota for menor ou igual a 3 ou
//menor ou igual a 6 (ou seja, entre 3 e 6)
else
if media >= 3.0 || media <= 6.0 then
//então exibirá a mensagem
Console.WriteLine("\nVocê está de exame final :(")
```

IV. O programa deverá nos solicitar a digitação de dois números e um caractere, sendo que este poderá ser “+”, “-”, “*” ou “/”. Mediante o caractere digitado fazer o respectivo cálculo e exibir o resultado, se o caractere não corresponder a nenhum dos 4 caracteres em questão exibir mensagem de erro. Este programa obrigatoriamente deverá ser feito usando um ninho de if’s.

```
open System

//pedindo valores
Console.WriteLine("Digite o 1º número: ")
//armezenando valores em variável inteira para conversão
let num1 = Int32.Parse(Console.ReadLine())

Console.WriteLine("\nDigite o 2º número: ")
let num2 = Int32.Parse(Console.ReadLine())

//pedindo sinal para operação
Console.WriteLine("\nDigite um sinal:  \n|   +   |   -   |   /   |   *   |")
//armazenando escolha numa variável
let sin = Console.ReadLine()

//condição que força digitação de um operando se não for digitado nenhum
//e se o que for digitado não for correspondente aos requeridos
if sin = "" || sin <> "+" || sin <> "-" || sin <> "/" || sin <> "*" then

Console.WriteLine("Operador não correspondente. Por favor, execute sua
escolha:\n")
Console.WriteLine("\n|   +   |   -   |   /   |   *   |")
let sin = Console.ReadLine()


//se a escolha for "+"
```

```
if sin = "+" then

//faça a conta
let soma = num1+num2

//exiba
Console.WriteLine("Você escolheu a operação: soma\n")
Console.WriteLine(num1.ToString() + " + " + num2.ToString() + " = " +
soma.ToString())

//e assim por diante
else
if sin = "-" then

let subtracao = num1 - num2

Console.WriteLine("Você escolheu a operação: subtração\n")
Console.WriteLine(num1.ToString() + " - " + num2.ToString() + " = " +
subtracao.ToString())

else
if sin = "/" then

let divisao = num1/num2

Console.WriteLine("Você escolheu a operação: divisão\n")
Console.WriteLine(num1.ToString() + " / " + num2.ToString() + " = " +
divisao.ToString())

else
if sin = "*" then

let multiplicacao = num1*num2

Console.WriteLine("Você escolheu a operação: multiplicação\n")
Console.WriteLine(num1.ToString() + " * " + num2.ToString() + " = " +
multiplicacao.ToString())
```

V. Idem ao exercício anterior, porém agora a solução deverá ser desenvolvida usando um “switch-case”, no caso do F#, match ‘condição’.

```
open System
```

```
let mutable valor1 = 0
let mutable valor2 = 0
let mutable operador = ""
let mutable resultado = 0

printfn "Digite o primeiro valor: "
let a = Console.ReadLine()

printfn "Digite o segundo valor: "
let b = Console.ReadLine()

valor1 <- a |> int
valor2 <- b |> int
printfn "\nDigite um sinal:  \n|   +   |   -   |   /   |   *   |"
let op = Console.ReadLine()

operador <- op |> string

let result =
   match operador with
   | operador when operador = "+"  ->
     resultado <- valor1 + valor2
     printf "Soma: %d" resultado
   | operador when operador = "-"  ->
       resultado <- valor1 - valor2
       printf "Subtração: %d" resultado
   | operador when operador = "/"  ->
       resultado <- valor1 / valor2
       printf "Divisão: %d" resultado
   | operador when operador = "*"  ->
       resultado <- valor1 * valor2
       printf "Multiplicação: %d" resultado
   |_ -> printfn "Operador inválido"
```

2.6 LAÇOS DE REPETIÇÃO: EXERCÍCIOS RESOLVIDOS

I. O programa deve começar nos solicitando a digitação de um número, inteiro e positivo, a partir de então o programa deverá exibir na tela a tabuada deste número. A entrada de dados não deverá ser validada, vamos acreditar que o usuário sempre digitará um valor devido.

```
open System
//criando um contador para multiplicar o valor digitado de 0 até 10
let mutable count = 0

//pedindo um número
Console.WriteLine("Digite um número para tabuada: ")
//armazenando numa variável
let nt = Int32.Parse(Console.ReadLine())

//se o número digitado for menor do que 0 -negativo- então
if nt <= 0 then

//avisa que o número não é válido
Console.WriteLine("Número inválido. Fora do intervalo")

//e pede pra digitar novamente
Console.WriteLine("Digite um número para tabuada: ")
let nt = Int32.Parse(Console.ReadLine())

//dando limite ao contador
while count <= 10 do

//estrutura da tabuada
let cont = nt * count

//exibição da tabuada
Console.WriteLine(nt.ToString() + "X" + count.ToString() + " = " +
cont.ToString())

count <- count + 1
```

II. O programa deverá exibir na tela os “n” primeiros termos da série: 2, 5, 10, 17, 26... onde o valor de “n” deverá ser inicialmente digitado. Em tempo esclareço que tal série tem como termo geral $(x^2 + 1)$ onde x = {1, 2, 3, 4, 5 ...}.

```
open System
//pedindo limite de termos
Console.WriteLine("Digite o número de termos:")
//armazenando numa variável
let n = Int32.Parse(Console.ReadLine())
Console.WriteLine()

//definindo começo e fim da sequência
for i = 1 to n do

//armazenando o termo geral da definição da sequência
//numa variável
let r = (i*i) + 1

//exibindo resultado
Console.WriteLine(r)
```

III. Temos aqui um clássico, o programa deverá listar no vídeo os termos da série de Fibonacci (1, 1, 2, 3, 5, 8, 13, 21 ...) menores que 1000.

```
open System

//declarando variáveis com os valores iniciais
let mutable proximo = 0
let mutable atual = 0
let mutable anterior = 1

//enquanto o próximo número da sequência for menor
//do que o último número menor que 1000 da sequência
//fibonacci
while proximo < 987 do

//faça: a variável próxima continua recebendo o valor
//de acordo com a definição da sequência fibonacci
proximo <- atual + anterior

//comando para exibir na tela
Console.WriteLine(proximo)
```

IV. Entrar com dois valores via teclado, onde o segundo deverá ser maior que o primeiro. Caso contrário solicitar novamente a digitação do segundo valor, o que deve ser repetido até que o usuário atenda a condição definida.

```
open System

//declaração das variáveis
let mutable x = 0
let mutable y = 0

//criando função entrada
let entrada() =

//pedindo 1º número
printf "Digite o 1º número: "
//armazenando numa variável
let num = Console.ReadLine()

//guardando o valor convertido da variável do número digitado em outra variável
x <- num |>int

//criando função de validação
let valida()=

//enquanto o 2º número for maior do que o primeiro
while (y <= x) do

//faça novamente o pedido do 2º número
printf "O 2º número será aceito apenas se for maior: "
let num2 = Console.ReadLine()
y <- num2 |> int

//chama as funções
entrada()
valida()
```

V. Entrar via teclado com o sexo de determinado usuário, aceitar somente “F” ou “M” como respostas válidas, caso o valor digitado seja indevido repetir o processo até que se digite um dos dois caracteres válidos.

```
open System

//Declaração de variáveis
let mutable s = ""
//Entrada de dados
let entrada() =
printf "Informe o sexo (M - masculino F - feminino):
"
let a = Console.ReadLine()
s <- a |> string
//Função que avalia os dados
let avalia() =
//Verifica se o sexo digitado é diferente "F" ou "M"
while s <> "F" && s <> "M" do
//Pede para informar o sexo novamente caso seja
diferente
printf "Informe o sexo novamente: "
s <- Console.ReadLine()

//Chama as funções
entrada()
avalia()
```

VI. O programa deverá nos permitir a digitação de um número inteiro e positivo, essa entrada deverá ser repetida até que o usuário satisfaça a condição. Feito isso o programa deverá calcular e exibir o fatorial deste número. Ao fim questionar se desejamos continuar ou não, e essa entrada só deverá aceitar como resposta “S” ou “N” e a pergunta deverá ser repetida até que o usuário responda corretamente. Se a resposta for “S” então deveremos voltar ao início do programa e repetir tudo novamente, caso contrário o programa deverá ser encerrado

```
open System

//declarando variável p/ continuação do programa
let mutable s = ""

//função p/ calcular fatorial
let rec fatorial x : int =
if x < 1 then 1
```

```
else x * fatorial (x - 1)

//programa principal
let rec programa() =

//criação de variáveis
let mutable n = 0
let mutable i = 0
let mutable nF = 1
let mutable num = 0

//entrada de valor pelo usuário
printfn "Digite um valor inteiro positivo: "
num <- Convert.ToInt32(Console.ReadLine())

//laço para travar valores em POSITIVOS INTEIROS
while num <= 0 do
printfn "Digite um valor inteiro positivo: "
num <- Convert.ToInt32(Console.ReadLine())

//chamando função e passando valor para variável nF
nF <- fatorial num

//exibição do resultado
printfn "Fatorial de %d é igual a: %d" num nF

//pergunta e laço para se usuário quer continuar ou não
printfn "Deseja realizar uma nova pesquisa (S/N): "
let a = Console.ReadLine()
s <- a |> string
if s = "S" || s = "s"  then
programa()
else
Console.ReadKey()

//Chamando a função principal
programa()
```

2.7 ARRAYS, VETORES E MATRIZES: EXERCÍCIOS RESOLVIDOS

I. O programa deverá nos permitir digitar e armazenar dez números na memória do computador. Feito isso exibir os valores digitados em tela na ordem inversa à da digitação.

```
open System

//Declaração de variaveis
let mutable X = [|0..9|]
let mutable i = 9


let Entrada()=

for i = 0 to 9 do
//Entrada e leitura de valores
printfn "Valor: "
let y = Console.ReadLine()
//Passando os valores para o vetor X
X.[i] <- y |>int

let Saida()=
while i >= 0 do
printf " Valor inverso: %d. \n" X.[i]
i <- i - 1

Entrada()
Saida()
```

II. O programa deverá nos permitir digitar e armazenar dez números na memória do computador. Feito isso criar um laço capaz de calcular a somatória desses 10 valores e então exibir a média desses valores.

```
open System

//Declaração de variáveis
let X = [|0..9|]
let entrada() =
       let mutable count = 0
       for i= 0 to 9 do
       count <- count +  1
```

```
        //Entrada dos valores
        printf "%d° valor: " count
        let a = Console.ReadLine()
//atribuindo valores para o array x
        X.[i] <- a |> int

//chamando a função
entrada()
//Soma os vetores
        let mutable soma = X.[0] + X.[1]+ X.[2]+ X.[3]+ X.[4]+ X.[5]+ X.[6]+
X.[7]+ X.[8]+ X.[9]
//Calculando a média
let somad =Convert.ToDouble(soma)
let mutable media=somad/10.0

//Mostra a soma e a média dos vetores
printfn "O a soma do vetor é: %d" soma
        printfn "A média do vetor é: %f" media
```

III. O programa deverá nos permitir digitar e armazenar dez números na memória do computador. Feito isso criar um laço capaz de identificar o maior e o menor dos números digitados e exibi-los ao final.

```
open System

//Declaração de variáveis
let X = [|0..9|]
let mutable maiorvalor = 0
let mutable menorvalor = 0

let entrada() =
    let mutable count = 0
    for i= 0 to 9 do
    count <- count +  1

   //Entrada e leitura dos valores
   printf "%d° valor: " count
   let a = Console.ReadLine()
   //Passagem dos valores para o vetor x
   X.[i] <- a |> int
//Verifica qual é o maior valor
```

```
let Maior() =
    maiorvalor <- X.[0]
    for i= 0 to 9 do
    if X.[i]> maiorvalor then
      maiorvalor <- X.[i]

 //Verifica qual é o menor valor
let Menor() =
     menorvalor <- X.[0]
     for i= 0 to 9 do
     if X.[i]< menorvalor then
      menorvalor <- X.[i]

//Exibe na tela o maior e menor valor
let saida() =
    Console.Clear()
    printfn "O maior valor é: %d" maiorvalor
    printfn "O menor valor é: %d" menorvalor

//Chama as funções
entrada()
Maior()
Menor()
saida()
```

IV. O programa deverá nos permitir digitar e armazenar o nome e idade de dez pessoas. Feito isso deverá nos solicitar a digitação de um nome e então proceder a pesquisa e informar a idade do sujeito pesquisado caso ele se encontre armazenado, caso contrário informar o fato através de mensagem “Pessoa não localizada”, ao final verificar se nova consulta é desejada, validar a resposta do usuário no sentido de só aceitar “S” ou “N”. Obviamente que no caso de nova consulta a digitação dos dez nomes/idades não deve ser repetido.

```
open System

       //Declaração de variáveis e dos array
let nome = Array.create 10 "" //declaração da array para armazenar o nome
let idade =  [|0..9|] //declaração da array para armazenar a idade
let mutable s = ""
let mutable count = 0
```

```
//Entrada de dados
let entrada() =
       for i=0 to 9 do
       count <- count + 1

       printf "%d° nome: " count
       let a = Console.ReadLine()

       printf "%d° idade: " count
       let b = Console.ReadLine()

       nome.[i] <- a
       idade.[i] <- b |> int

//Pesquisa dos nomes
let pesquisa() =
       printfn "Qual nome deseja pesquisar: "
       let c = Convert.ToString(Console.ReadLine())

       for i =0 to 9 do

       if nome.[i] = c then

       printfn "A idade é: %d" idade.[i]

       if nome.[i] <> c then
       printfn "Pessoa não encontrada"

//Nova pesquisa
let novapesq() =
       printf "Deseja realizar uma nova pesquisa (S/N): "
       let a = Console.ReadLine()

       s <- a |> string
       if s = "S" || s = "s"  then
       pesquisa() //Chamando a função pesquisa

//Chamando as funções
entrada()
pesquisa()
novapesq()
```

V. Determinado cinema tem 20 fileiras (de 1 a 20) com 15 cadeiras (de 1 a 15) cada uma, portanto estamos falando de uma sala com 300 assentos, que deve ser reproduzida através de um array de 20 linhas por 15 colunas. O programa deve começar solicitando do usuário a digitação de seu nome, o número da fileira e cadeira em que deseja se sentar, se o assento estiver vazio reservá-lo registrando no array seu nome, caso contrário informar que o assento está ocupado. Feito isso o programa deverá nos questionar se desejamos nova reserva, validando nossa resposta e repetindo todo o processo.

```
open System

//Declaração de váriaveis
let mutable reserva = Array2D.create 15 20 ""
let mutable nome = ""
let mutable n_fileira = 0
let mutable assento = 0
let mutable k = 0
let mutable resposta = ""
let mutable var = false

//Entrada dos valores
let entrada()=

while resposta <> "N" do

     //Aramazena o nome do usuário
     printfn "\nQual o seu nome?"
     let a = Console.ReadLine()
     nome <- a

     //Armazena a fileira que o usuário deseja
     printfn "\nQual fileira deseja?"
     let f = Console.ReadLine()
     n_fileira <-  f |> int

     //Verfica se a fileira existe
     if n_fileira > 0 && n_fileira <= 20   then

       //Armazena o assento que o usuário deseja
       printfn "\nQual o assento desejado?"
       let ass = Console.ReadLine()
```

```
                assento <- ass |> int

                //Verifica se o assento existe
                if assento > 0 && assento <= 15 then

                  //Verifica se o assento não está ocupado
                  if reserva. [assento - 1, n_fileira - 1] = "" then

                     //Reserva o assento para o usuário
                     reserva. [assento - 1, n_fileira - 1] <- nome

                     //Verifica se o usuário deseja fazer uma nova reserva e armazena a
resposta
                     printfn "\nDeseja fazer uma nova reserva? S/N"
                     let a = Console.ReadLine()
                     resposta <- a

                     //Verifica se a resposta é válida
                     if resposta <> "S" && resposta <> "N" then
                        //Se a resposta não for nem "S" e nem "N" var recebe false
                        var <- false

                        //Enquanto a  resposta não for a desejada o programa continuára
perguntando até obter uma resposta válida
                        while var <> true do

                           printf "\nDigite uma resposta válida!"

                           //Pergunta se a pessoa deseja fazer uma reserva
                           printfn "\nDeseja fazer uma reserva? S/N"
                           let c = Console.ReadLine()
                           resposta <- c

                           //Se a resposta for válida var recebe true e saí do loop
                           if resposta = "S" || resposta = "N" then
                              var <- true

                     //Se a resposta for válida var recebe true e saí do loop
                     else
                        var <- true
                  else
```

```
                    //Mostra que o assento já está ocupado
                    printfn "\nEste asssento já está ocupado!"

                    //Verifica se o usuário deseja fazer uma nova reserva e armazena a
resposta
                    printfn "\nDeseja fazer uma reserva? S/N"
                    let a = Console.ReadLine()
                    resposta <- a

                    //Verifica se a resposta é válida
                    if resposta <> "S" && resposta <> "N" then
                    //Se a resposta não for nem "S" e nem "N" var recebe false
                      var <- false

                      //Enquanto a resposta não for a desejada o programa continuará
perguntando até obter uma resposta válida
                      while var <> true do

                         printf "\nDigite uma resposta válida!"

                         //Pergunta se a pessoa deseja fazer uma reserva
                         printfn "\nDeseja fazer uma reserva? S/N"
                         let c = Console.ReadLine()
                         resposta <- c

                         //Se a resposta for válida var recebe true e saí do loop
                         if resposta = "S" || resposta = "N" then
                            var <- true

                    //Se a resposta for válida var recebe true e saí do loop
                    else
                      var <- true

                else

                  //Mostra que o assento digitado não existe
                  printf "\nEste assento não é válido"

                  //Verifica se o usuário deseja fazer uma nova reserva e armazena a
resposta
                  printfn "\nDeseja fazer uma reserva? S/N"
                  let a = Console.ReadLine()
```

```
                resposta <- a

                //Verifica se a resposta é válida
                if resposta <> "S" && resposta <> "N" then
                //Se a resposta não for nem "S" e nem "N" var recebe false
                   var <- false

                   //Enquanto a resposta não for a desejada o programa continuará
perguntando até obter uma resposta válida
                   while var <> true do

                      printf "\nDigite uma resposta válida!"

                      //Pergunta se a pessoa deseja fazer uma reserva
                      printfn "\nDeseja fazer uma reserva? S/N"
                      let c = Console.ReadLine()
                      resposta <- c

                      //Se a resposta for válida var recebe true e saí do loop
                      if resposta = "S" || resposta = "N" then
                         var <- true

                //Se a resposta for válida var recebe true e saí do loop
                else
                   var <- true

             else
                  //Mostra que a fileira digitada não existe
                  printfn "\nEsta fileira não é válida"

                  //Verifica se o usuário deseja fazer uma nova reserva e armazena a
resposta
                  printfn "\nDeseja fazer uma reserva? S/N"
                  let a = Console.ReadLine()

                  resposta <- a

                  //Verifica se a resposta é válida
                  if resposta <> "S" && resposta <> "N" then
                  //Se a resposta não for nem "S" e nem "N" var recebe false
                     var <- false
```

```
                    //Enquanto a resposta não for a desejada o programa continuará
perguntando até obter uma resposta válida
                    while var <> true do

                        printf "\nDigite uma resposta válida!"

                        //Pergunta se a pessoa deseja fazer uma reserva
                        printfn "\nDeseja fazer uma reserva? S/N"
                        let c = Console.ReadLine()
                        resposta <- c

                        //Se a resposta for válida var recebe true e saí do loop
                        if resposta = "S" || resposta = "N" then
                            var <- true

                //Se a resposta for válida var recebe true e saí do loop
                else
                    var <- true

        //Exibi os valores armazenados no array bidimensional
        let saida() =

            for i=0 to 14 do

                printfn "%A " reserva. [i , *]

        entrada()
        saida()
```

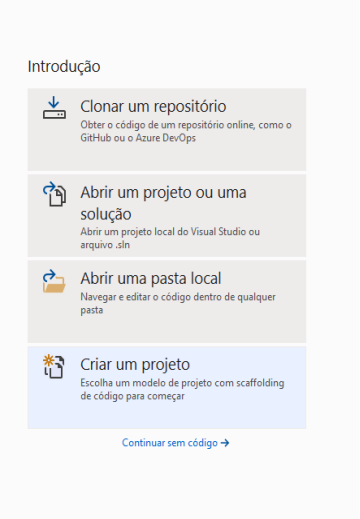
3 CONEXÃO DE FORMULÁRIOS (FORMS) COM A LINGUAGEM F#

Não existem ferramentas que possibilitem a manipulação de Interface Gráfica do Usuário (GUI) diretamente com a linguagem F#. Assim, quando se encontra a necessidade de fazer uma aplicação GUI com a linguagem, tem sido recomendada a seguinte abordagem:

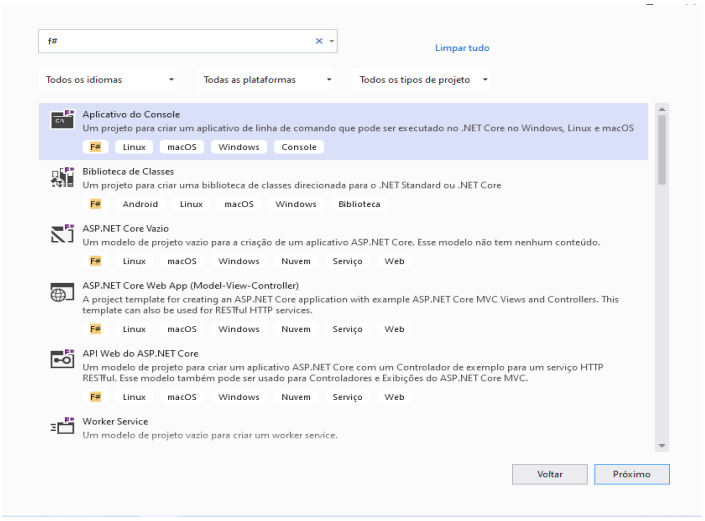
- Criar a lógica da Interface Gráfica em C#, usando o Forms UI Designer;
- Usar o F# para criar bibliotecas e chamá-las a partir da Interface Gráfica.

Para a utilização desta abordagem acima citada, serão referenciados alguns passos a passo.

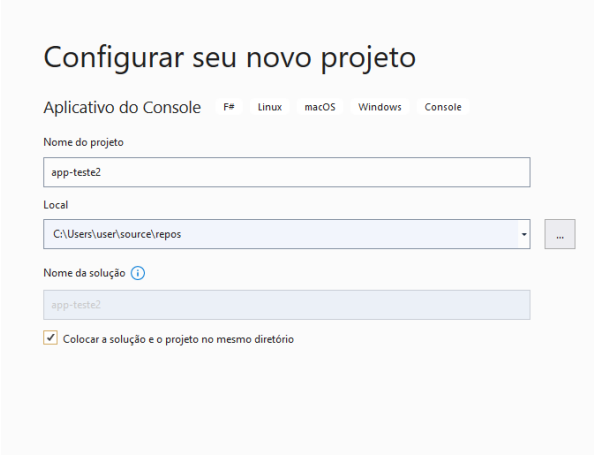
1) Crie um novo projeto.



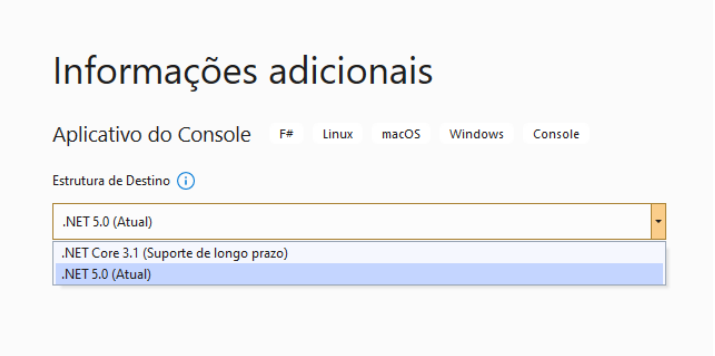
2) Crie um aplicativo para console em F#.



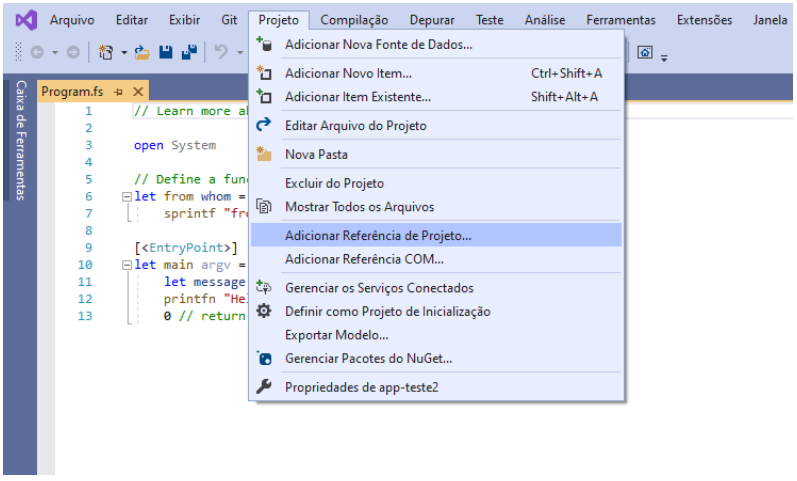
3) Configure e nomeie o projeto.



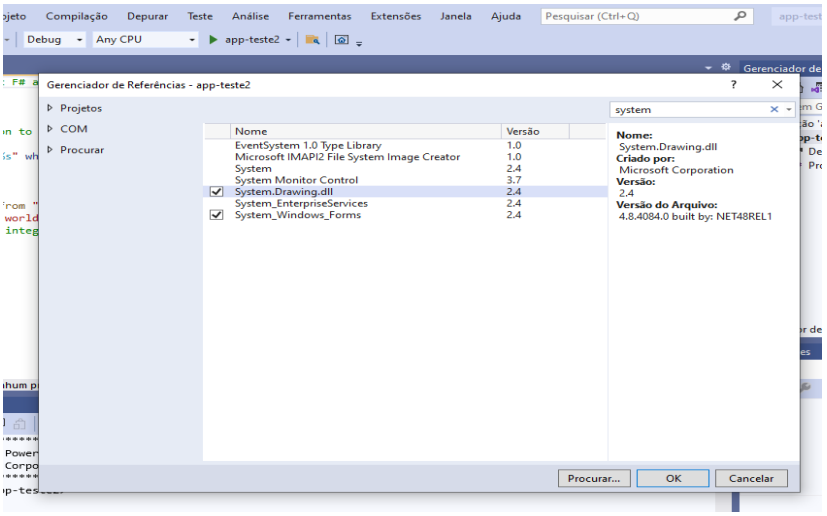
4) Altere a estrutura de destino para .NET 5.0 (atual).



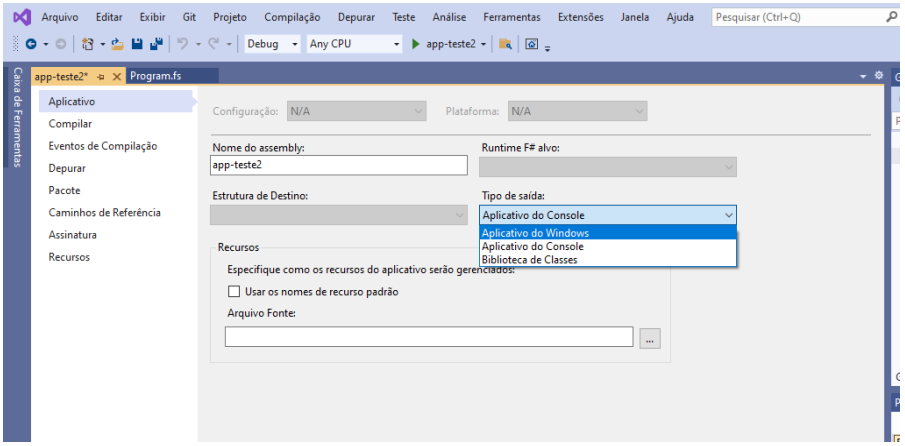
5) Vá em “Projeto” e “Adicionar Referência de Projeto”.



6) Selecione “System.Drawing.dll” e “System_Windows_Forms”, e dê OK.

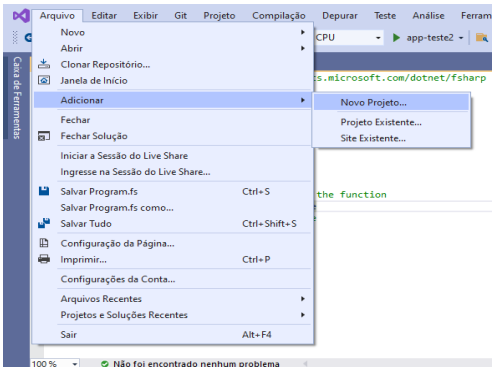


7) Abra “Projeto”; “Propriedades”; e altere “Tipo de saída” para “Aplicativo do Windows”. Dê CTRL + S.

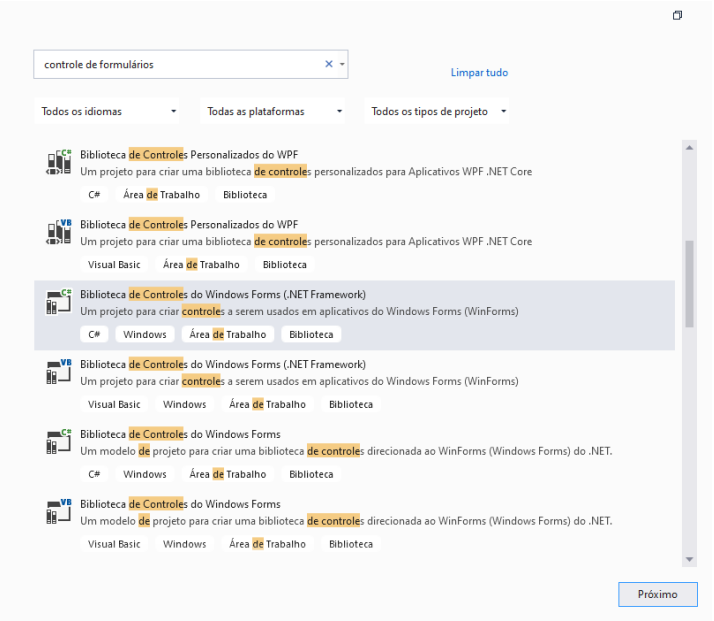


Está criada a parte para a execução de bibliotecas.

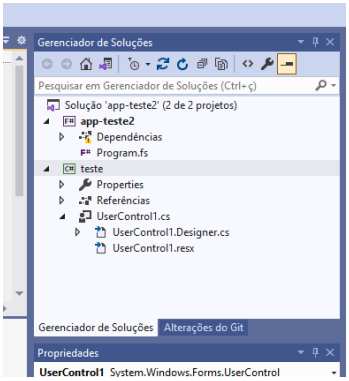
8) Vá em “Arquivo”; “Adicionar”; clique em “Novo Projeto”.

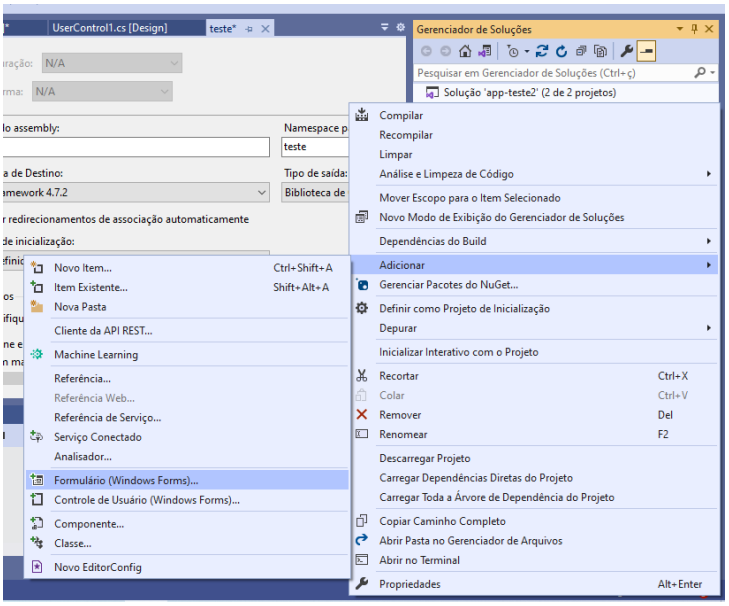


9) Clique em “Biblioteca de Controles do Windows Forms (.NET Framework)”. O .NET Framework a ser usado deve ser o .NET Framework 4.7.2.

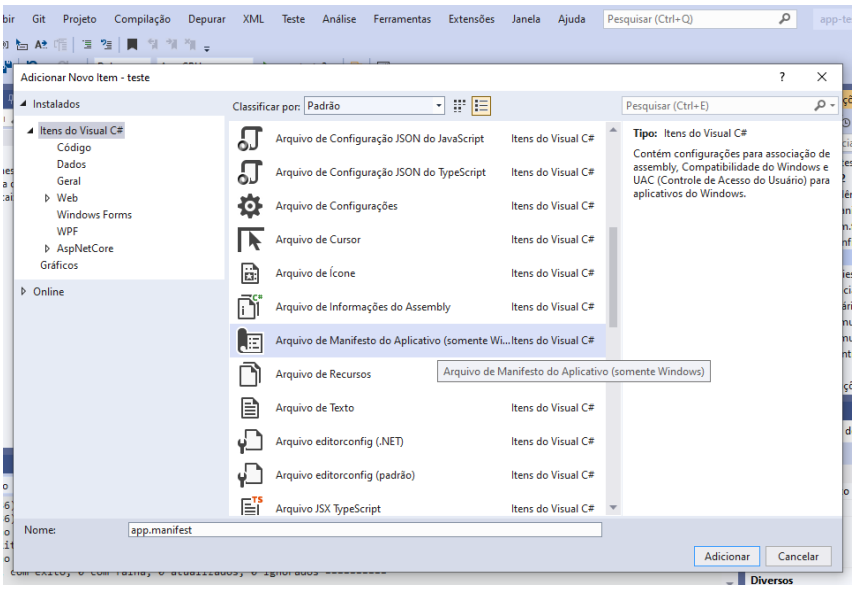


10) Sobre o arquivo do designer adicionado ao projeto (em C#), dê clique com o botão direito do mouse; clique em “Adicionar”; depois, “Formulário (Windows Forms)”; e nomeie-o.

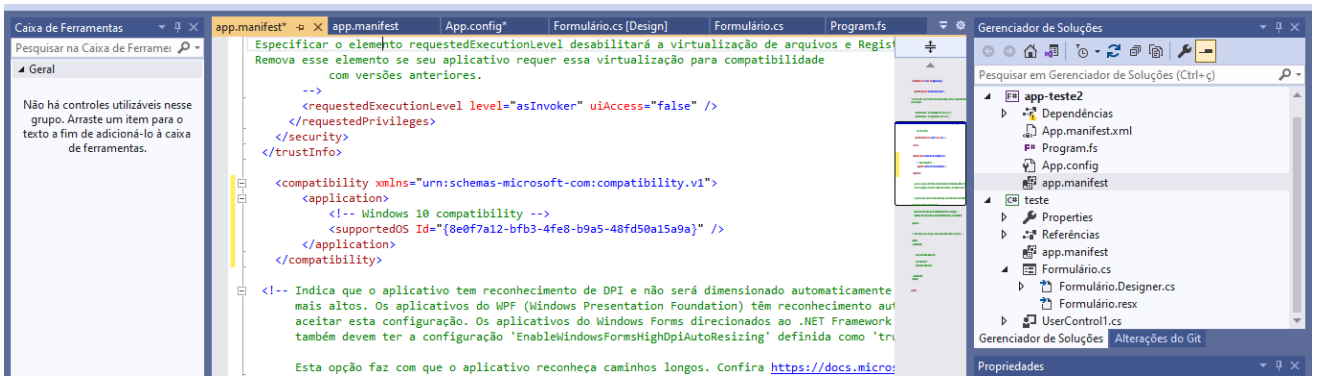




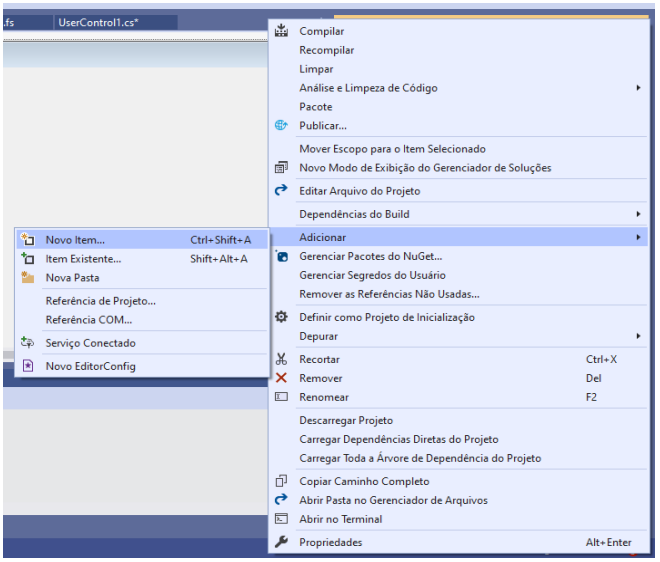
11) Sobre o arquivo do designer do projeto (em C#), dê clique com o botão direito do mouse; clique em “Adicionar”; “Novo Item”; “Arquivo de Manifesto do Aplicativo”.

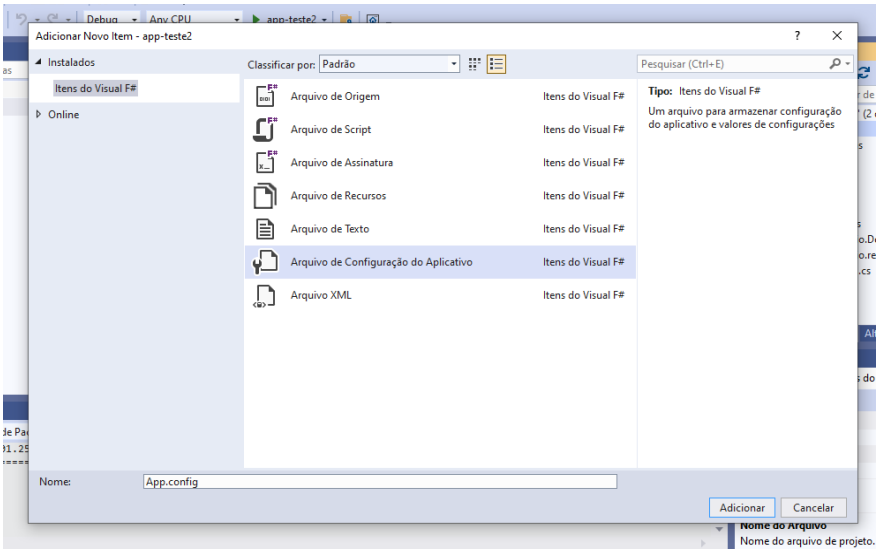


12) Passe a pasta que se abrir (app.manifest) para o lado do projeto em F#. Após, modifique-a, declarando compatibilidade com o Windows que se usa [no caso, o Windows utilizado é o Windows 10]. insira o seguinte comando:

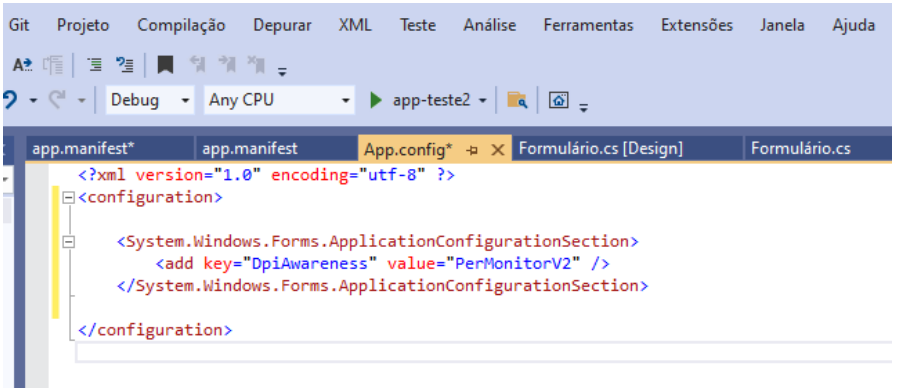


13) Sobre o arquivo do código em F# do projeto, dê clique com o botão direito do mouse; clique em “Adicionar”; depois, “Novo Item”; “Arquivo de Configuração do Aplicativo”.

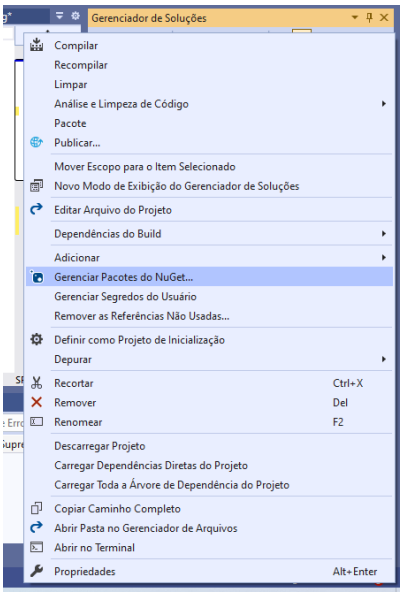


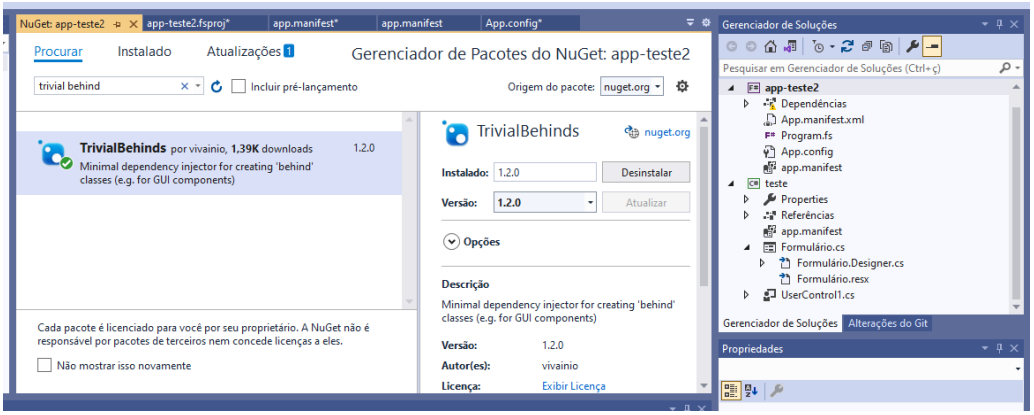


14) Na pasta que se abrir (app.config), modifique-a. É necessário que se habilite o reconhecimento de DPI por monitor. insira o seguinte comando:

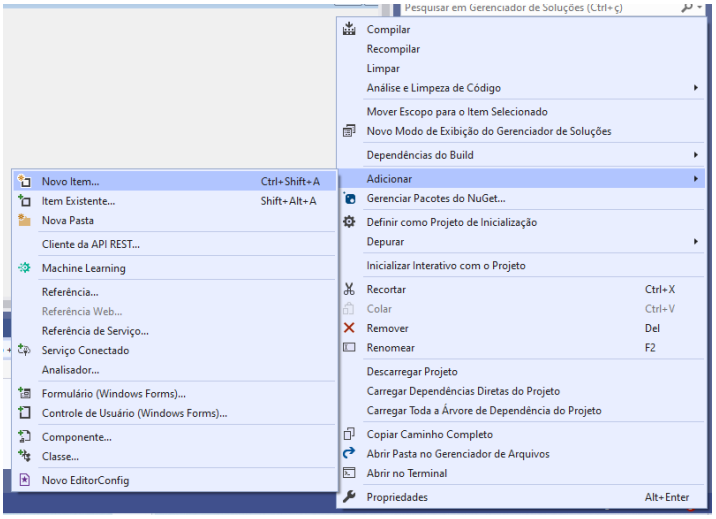


15) Clicando com o botão direito do mouse sobre ambos os projetos (em C# e em F#), vá em “Gerenciar Pacotes do NuGet”. Depois, procure por “Trivial Behinds” e instale o pacote em ambos.

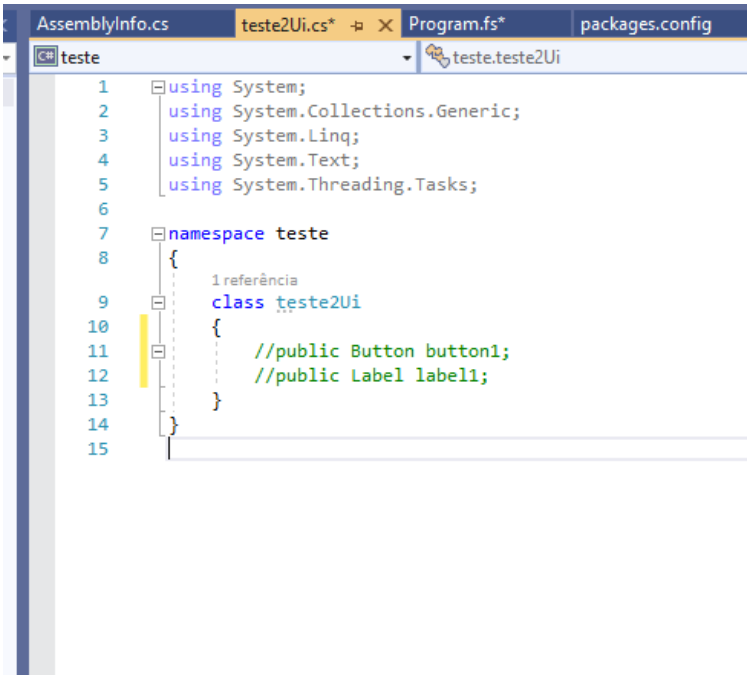




16) Adicione nova classe (*Class*) ao projeto do designer (em C#).

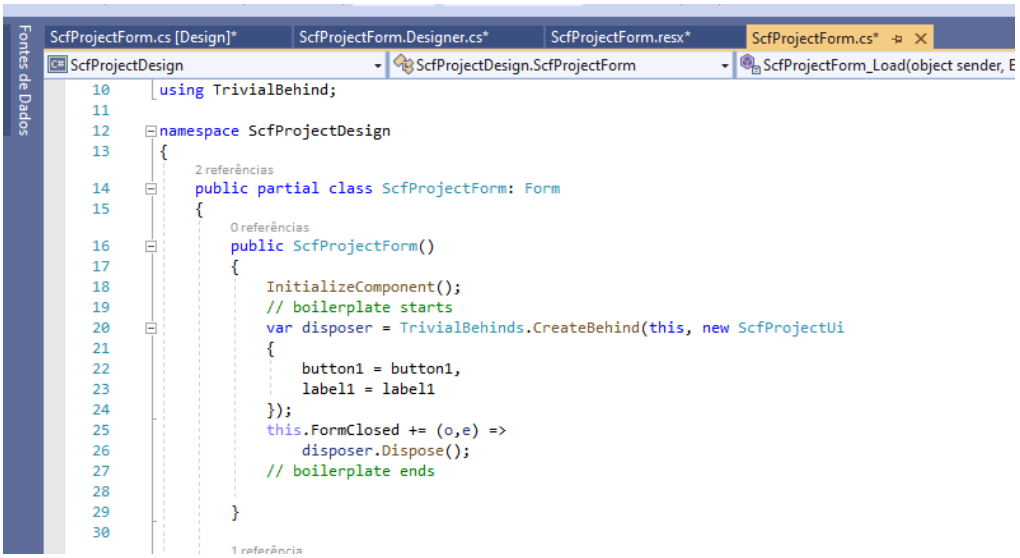


17) Dentro da classe (.class), instanciará todos os componentes (atributos) que forem criados no formulário.



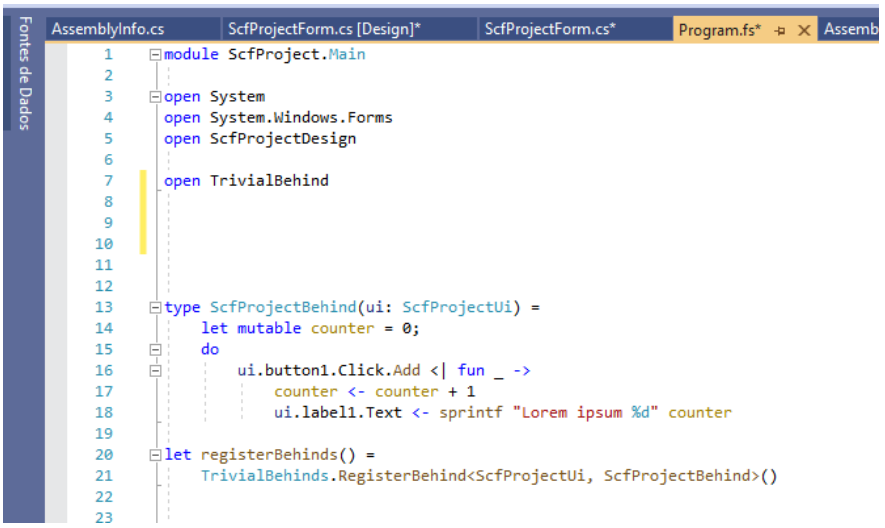
18) Faça conexão com a biblioteca Trivial Behinds em ambos os projetos.

- Em C# (Designer).



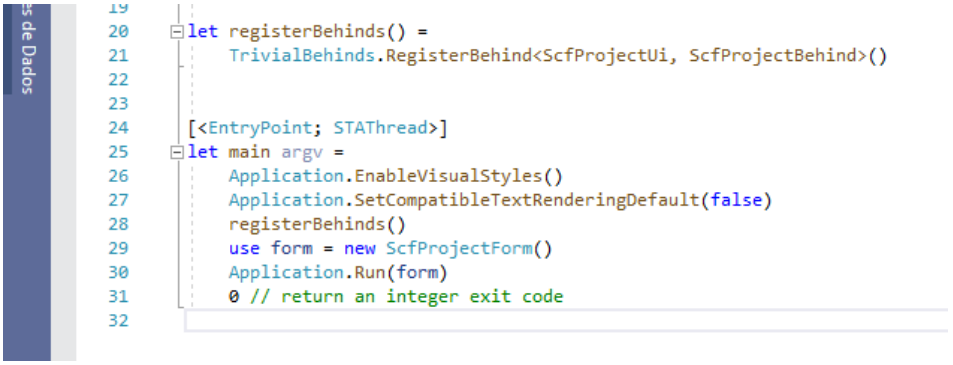
Neste primeiro caso, vale atentar-se para a declaração de uso (*using*) da biblioteca no início do código. É importante salientar também a conexão estabelecida com a classe (*class*) criada.

- Em F# (código).



Aqui, será feita a criação do módulo Main (declarando que o Main está neste projeto); chamará as bibliotecas (Windows.Forms e Trivial Behinds), bem como o projeto do designer; passará a programar os atributos que estão na classe em C#, mas na linguagem e projeto em F#.

Por fim, no Main do projeto em F#, será utilizado o método “Application.EnableVisualStyles()”, que consistirá na correção da renderização da fonte.



```
19
20  let registerBehinds() =
21      TrivialBehinds.RegisterBehind<ScfProjectUi, ScfProjectBehind>()
22
23
24  [<EntryPoint; STAThread>]
25  let main argv =
26      Application.EnableVisualStyles()
27      Application.SetCompatibleTextRenderingDefault(false)
28      registerBehinds()
29      use form = new ScfProjectForm()
30      Application.Run(form)
31      0 // return an integer exit code
32
```

3.1 CARACTERÍSTICAS DOS OBJETOS UTILIZADOS NOS FORMS

Para a criação dos formulários (ou forms) é necessário veículos de interação entre o programa e o usuário, sendo necessário o uso de objetos, eventos e demais propriedades provenientes das funcionalidades gráficas. Assim, visando uma melhor compreensão desses objetos, podemos listar:

- Os bottons (ou botões): Considerado o objeto mais popular, os botões de comando tratam da manipulação de eventos, sendo acionados pelo clique do mouse ou pela tecla “ENTER”;
- Os “CheckBoxes”: Utilizados com a finalidade de marca e desmarcar as opções que podem ser apresentadas em formato de texto, imagem ou ambos;
- Os “RadioButtons”: Possuem função idêntica aos “CheckBoxes”, porém são organizados de forma grupada, isto é, de forma dependente, assim, quando uma opção é assinalada, as outras são indisponíveis;
- Os “Panels”: Estes são utilizados para agrupar partes de um formulário, a fim de organizar as informações contidas no mesmo;
- Os “GroupBoxes”: Desempenham a mesma função dos “Panels”, porém apresentam um título ao agrupamento, e são mais utilizados para manipular “RadioButtons”;
- Os “TextBoxes”: Utilizados para a inserção de dados dos usuários.
- As “Labels”: Apenas para a inserção de textos a fim de facilitar a comunicação dos usuários.

Assim, através desses objetos principais pode-se criar uma infinidade de formulários para os mais diversos intuitos, manipulando e tratando a entrada e saída de dados.

3.2 CRIANDO FORMS EM F#

Antes de começar a fazer o código em F#, é preciso instanciar todos os componentes que foram criados no formulário dentro da classe, depois disso é feito a conexão com a biblioteca nos projetos, assim como no exemplo que foi dado para a conexão de formulários para utilizar a linguagem F#.

Declaração com radioButton:

```
using System.Windows.Forms;

namespace ScfProjectDesign
{
    1 referência
    public class ScfProjectUi
    {
        public RadioButton radioButton1;
        public RadioButton radioButton2;
        public TextBox textBox1;
        public TextBox textBox2;
        public Button button1;
        public Button button2;
    }
}
```

```
namespace ScfProjectDesign
{
    2 referências
    public partial class ScfProjectForm: Form
    {
        0 referências
        public ScfProjectForm()
        {
            InitializeComponent();
            // boilerplate starts
            var disposer = TrivialBehinds.CreateBehind(this, new ScfProjectUi
            {
                radioButton1 = radioButton1,
                radioButton2 = radioButton2,
                textBox1 = textBox1,
                textBox2 = textBox2,
                button1 = button1,
                button2 = button2
            });
            this.FormClosed += (o,e) =>
                disposer.Dispose();
            // boilerplate ends

        }
```

I) Salário Horista com radioButton:

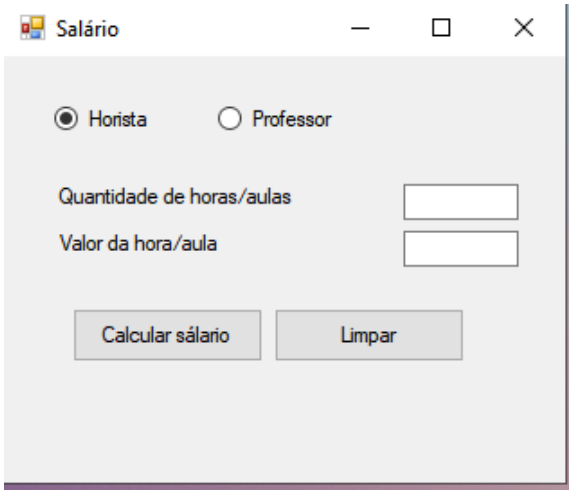
- Linha de Código:

```
module ScfProject.Main
```

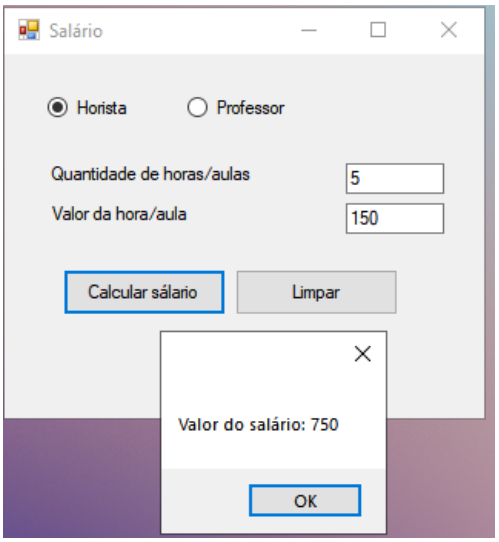
```
open System
open System.Windows.Forms
open ScfProjectDesign

open TrivialBehind

//implementando a biblioteca
type ScfProjectBehind(ui:ScfProjectUi) =

//declarando variáveis mutáveis
     let mutable qtd = 0.0
     let mutable valor = 0.0
     let mutable sb = 0.00

     //aplicando a funcionalidade 'cálculo' para o botão calcular
     do
      ui.button1.Click.Add <| fun _ ->

       //se o radioButton horista estiver ‘checado’ como a escolha para cálculo, então
        if ui.radioButton1.Checked then

//armazena à variável 'qtd' o valor de quantidade de horas trabalhadas, digitado no
primeiro campo de texto
           qtd <- ui.textBox1.Text |> double

//armazena à variável 'valor' o valor monetário das horas trabalhadas, digitado no segundo campo de texto
           valor <- ui.textBox2.Text |> double

          //armazena o cálculo para obter o salário na variável 'sb'
           sb <- qtd * valor

          //armazenando o resultado de sb numa variável string, para exibição
           let b = sb |> string

          //exibindo uma MessageBox informando o valor do salário calculado
           MessageBox.Show(sprintf "Valor do salário: %s" b) |>ignore

//Caso a opção não for a horista, ou seja, se o radioButton professor estiver ‘checado’ como a escolha para cálculo, então
```

```
            else

      //armazena na variável 'qtd' o valor de quantidade de horas trabalhadas, digitado
no primeiro campo de texto
            qtd <- ui.textBox1.Text |> double

      //armazena na variável 'valor' o valor monetário das horas trabalhadas, digitado
no segundo campo de texto
            valor <- ui.textBox2.Text |> double

            //armazena o cálculo para obter o salário na variável 'sb'
            sb <- (qtd * valor) * 1.25

            //armazenando o resultado de sb numa variável do tipo string, para
exibição
            let b = sb |> string

            //exibindo uma MessageBox informando o valor do salário calculado
            MessageBox.Show(sprintf "Valor do salário: %s" b) |>ignore

            ui.button2.Click.Add <| fun _ ->
            ui.textBox1.Clear()
            ui.textBox2.Clear()

      let registerBehinds() =
       TrivialBehinds.RegisterBehind<ScfProjectUi, ScfProjectBehind>()

      [<EntryPoint; STAThread>]
      let main argv =
          Application.EnableVisualStyles()
          Application.SetCompatibleTextRenderingDefault(false)
          registerBehinds()
          use form = new ScfProjectForm()
          Application.Run(form)
          0 // return an integer exit code
```

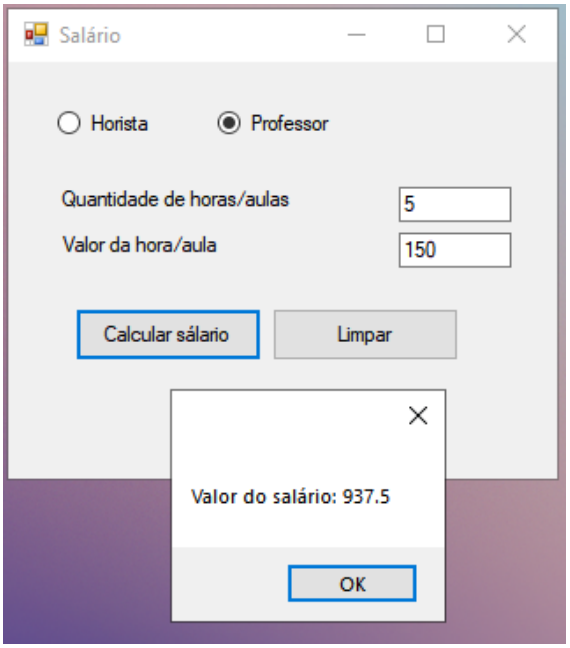
- Depuração:



- Depuração Horista:



- Depuração Professor:



II) Salário Horista com comboBox:

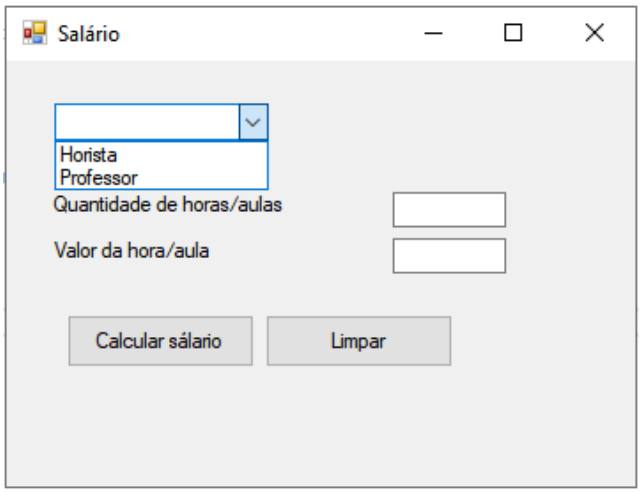
- Linha de Código:

```
module ScfProject.Main

open System
open System.Windows.Forms
open ScfProjectDesign

open TrivialBehind

//implementando a biblioteca
type ScfProjectBehind(ui:ScfProjectUi) =

//declarando variáveis mutáveis
     let mutable qtd = 0.0
     let mutable valor = 0.0
     let mutable sb = 0.00

     //aplicando a funcionalidade 'cálculo' para o botão calcular
     do
      ui.button1.Click.Add <| fun _ ->

      //se a escolha para cálculo, for o horista, ou seja, o primeiro índice
armazenado na comboBox, então
        if ui.comboBox2.SelectedIndex = 0 then
```

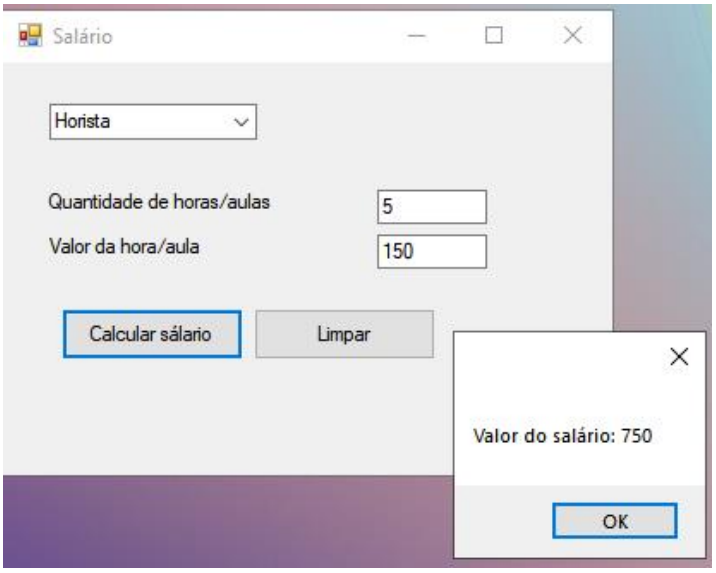
```
        //armazena na variável 'qtd' o valor de quantidade de horas trabalhadas,
digitado no primeiro campo de texto
        qtd <- ui.textBox1.Text |> double

        //armazena na variável 'valor' o valor monetário das horas trabalhadas,
digitado no segundo campo de texto
        valor <- ui.textBox2.Text |> double

        //armazena o cálculo para obter o salário na variável 'sb'
        sb <- qtd * valor

        //armazenando o resultado de sb numa variável do tipo string, para
exibição
        let b = sb |> string

        //exibindo uma MessageBox informando o valor do salário calculado
        MessageBox.Show(sprintf "Valor do salário: %s" b) |>ignore

        //caso a escolha para cálculo, for professor, ou seja, o segundo índice
armazenado na comboBox, então
        else

        //armazena na variável 'qtd' o valor de quantidade de horas trabalhadas,
digitado no primeiro campo de texto
        qtd <- ui.textBox1.Text |> double

        //armazena na variável 'valor' o valor monetário das horas trabalhadas,
digitado no segundo campo de text
        valor <- ui.textBox2.Text |> double

        //armazena o cálculo para obter o salário na variável 'sb'
        sb <- (qtd * valor) * 1.25

        //armazenando o resultado de sb numa variável string, para exibição
        let b = sb |> string

        //exibindo uma MessageBox informando o valor do salário calculado
        MessageBox.Show(sprintf "Valor do salário: %s" b) |>ignore


      //aplicando funcionalidade ao botão 'limpar'
```

```
        ui.button2.Click.Add <| fun _ ->

        //limpando os valores digitados nos campos de texto
        ui.textBox1.Clear()
        ui.textBox2.Clear()


let registerBehinds() =
 TrivialBehinds.RegisterBehind<ScfProjectUi, ScfProjectBehind>()

[<EntryPoint; STAThread>]
let main argv =
    Application.EnableVisualStyles()
    Application.SetCompatibleTextRenderingDefault(false)
    registerBehinds()
    use form = new ScfProjectForm()
    Application.Run(form)
    0 // return an integer exit code
```

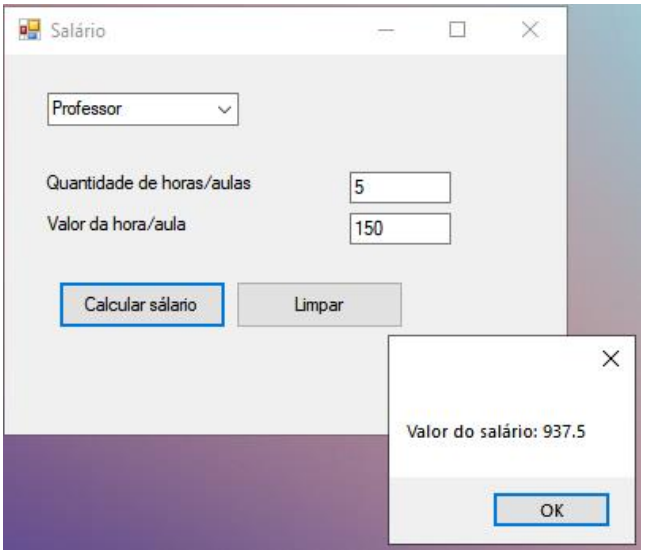
- Depuração:



- Depuração Horista:



- Depuração Professor:



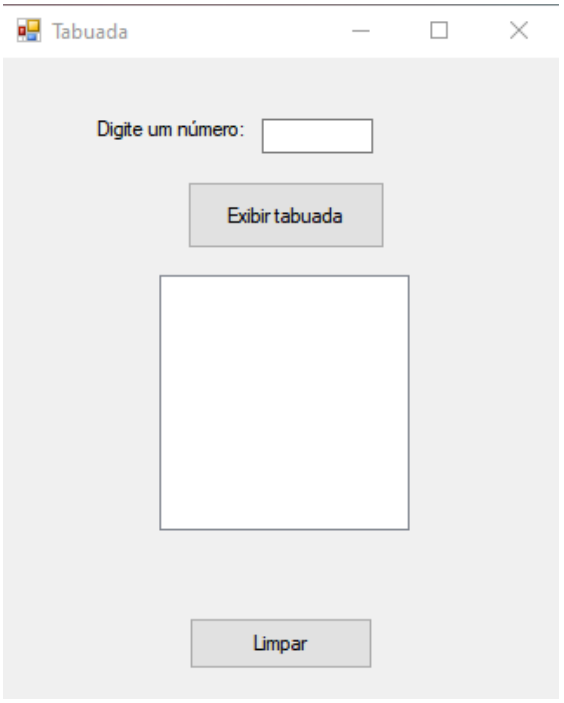
III) Tabuada

- Linha de Código:

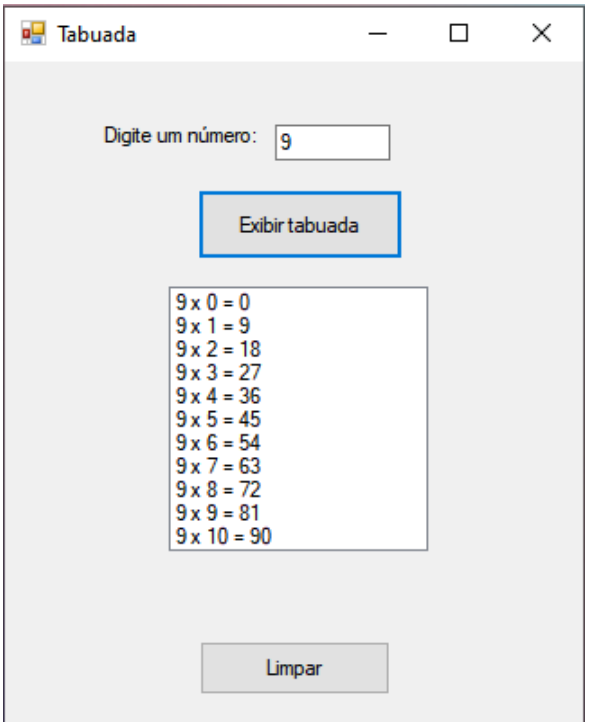
```
module ScfProject.Main
open System
open System.Windows.Forms
open ScfProjectDesign
open TrivialBehind

type ScfProjectBehind(ui:ScfProjectUi) =

    //Declaração de variáveis
    let mutable x = 0
```

```
    let mutable tab = 0

    do
     ui.button1.Click.Add <| fun _ ->

       //Atribuição de valor à variável
       let num = Console.ReadLine()
       x <- ui.textBox1.Text |> int

       //Estrutura de repetição com contador para formular a tabuada de 0 a 10
       for i = 0 to 10 do

       //Cálculo da tabuada atribuído a variável 'tab'
       tab <- x * i

       //Mostrando resultado da conta do número requerido na listBox1
       ui.listBox1.Items.Add (x.ToString() + " x " + i.ToString() + " = " +
tab.ToString()) |> ignore

       //Limpando campos para uma nova consulta
       ui.button2.Click.Add <| fun _ ->
       ui.listBox1.Items.Clear()
       ui.textBox1.Clear()


let registerBehinds() =
 TrivialBehinds.RegisterBehind<ScfProjectUi, ScfProjectBehind>()


[<EntryPoint; STAThread>]
let main argv =
    Application.EnableVisualStyles()
    Application.SetCompatibleTextRenderingDefault(false)
    registerBehinds()
    use form = new ScfProjectForm()
    Application.Run(form)
    0 // return an integer exit code
```

- Depuração:



- Depuração em teste:



4 CRIANDO CLASSES EM F#

Esse capítulo foi construído baseado no anterior, mas com um “*upgrade*”. Dessa vez, a constituição dos programas realizados anteriormente teve como princípio a linguagem orientada a objetos mostrando a declaração de classes.

As classes são o que dão origem para os objetos, e os objetos instanciam as classes. Elas são tipos; tipos esses que representam os objetos que podem ter propriedades, métodos e eventos. Eles são usados para modelar as ações, processos, e quaisquer entidades conceituais em aplicações.

F# consegue ser OO (orientado a objetos), sendo assim, pode possuir classes; interfaces; etc. Para identificar o começo da classe, diferentemente do C# o qual a *keyword* é `class`, no F# iniciamos com `type`. O `type` é o identificador válido do FSharp, com acesso padrão público. Outra comparação com C# é o ganho de tempo na hora da declaração pela redução da linha de código.

Os construtores em F# funcionam de um jeito diferente de outras linguagens .Net, isso, pois os argumentos do construtor primário são descritos como lista de parâmetros, logo, eis o porquê de o corpo do construtor consistir nas ligações de `let` e `do`. Sem alongar mais em texto, a seguir, exemplos práticos das ‘novas’ linhas de código:

Programa 1:

- **Área do Triângulo**

Classe Triângulo

```
module Triângulo

open System
open System.Windows.Forms
open ScfProjectDesign
open TrivialBehind

//Variavel de auxiliar
let mutable area = ""

type public Triangulo(bas:string, altura:string) = class

     // Setando a base
     let setBase(b: string)
         bas = b

    // Retorno do valor da base
     let getBase q =
         q = bas

    // Setando a altura
```

```
     let setAlt(_alt: string)
         altura = _alt

    // Retorno do valor da altura
     let getAlt v =
          v = altura

     //Retorno do cálculo da área
     let getArea =
         area <- ((float(bas) * float(altura)) / 2.00).ToString()

    // Atribuições
     member this.bas = bas
     member this.altura = altura
     member this.area = area
end
```

Classe TriânguloBLL

```
module TriânguloBLL

open Erro
open Triângulo

let mutable mens = ""

type ValidaDados(_bas:string, _alt:string) = class
    //Variáveis auxiliares
    let mutable v1 = _bas
    let mutable v2 = _alt

    // Verificações sobre o campo e eus respectivos valores
    let Verifica =
      if v1.Length = 0 then
         mens <- "É necessário preencher o campo Base"
      else
      if v2.Length = 0 then
         mens <- "É necessário preencher o campo Altura"
         else
         if v1.Length < 0 then
             mens <- "É necessário preencher o campo com um valor positivo"
          else
          if v2.Length < 0 then
             mens <- "É necessário preencher o campo com um valor positivo"

    //Atribuição da mensagem
    member this.mens = mens
 end
```

Classe TriânguloIHM

```
module TriânguloIHM.Main

open System
open System.Windows.Forms
open ScfProjectDesign
open TrivialBehind
open Triângulo
open TriânguloBLL

//Variável auxiliar
```

```
let mutable area = 0.0;

type public TrianguloIHM(ui: ScfProjectUi) =

    //Validação dos Campos
    do
         ui.btncalculo.Click.Add <| fun _ ->

            let b = ValidaDados(ui.txtBase.Text, ui.txtAltura.Text)

            if ui.txtBase.Text.Length = 0 then
                MessageBox.Show(mens) |>ignore
             else
             if ui.txtAltura.Text.Length = 0 then
                 MessageBox.Show(mens) |>ignore
             else
              let ar = Triangulo(ui.txtBase.Text, ui.txtAltura.Text)

              ui.txtArea.Text <- ar.area.ToString()
              ui.txtAltura.Enabled <- false
              ui.txtBase.Enabled <- false

        //Funcao para limpar os campos quando o botao "limpar" quando for clicado
        ui.btnclear.Click.Add <| fun _ ->
                //Limpando os campos:
               ui.txtAltura.Enabled <- true
               ui.txtBase.Enabled <- true
               ui.txtBase.Text<-null
               ui.txtAltura.Text<-null
               ui.txtArea.Text<-null

let registerBehinds() =
  TrivialBehinds.RegisterBehind<ScfProjectUi, TrianguloIHM>()

[<EntryPoint; STAThread>]
let main argv =
    Application.EnableVisualStyles()
    Application.SetCompatibleTextRenderingDefault(false)
    registerBehinds()
    use form = new ScfProjectForm()
    Application.Run(form)
    0 // return an integer exit code
```

Classe Erro

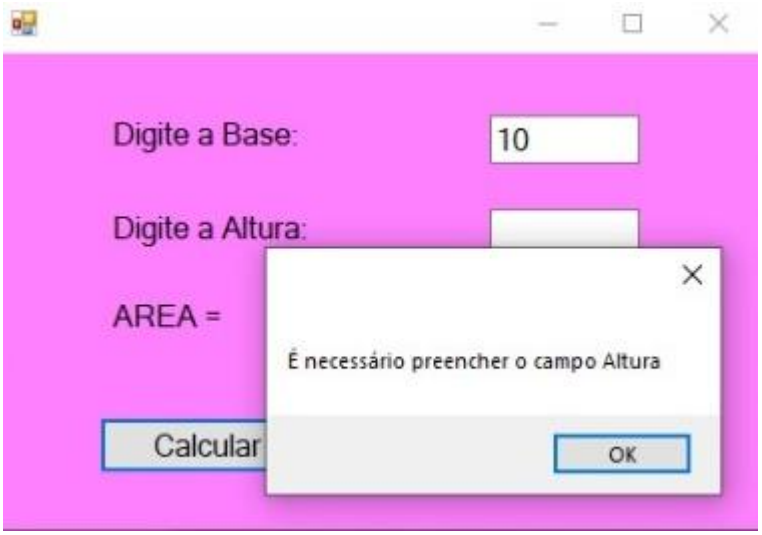
```
module Erro

//Variáveis auxiliares
let mutable mens_ = ""
let mutable erro2 = false

type cErro(erro:bool) = class

//Setando o Erro
let setErro _erro =
_erro = erro

//Setando a mensagem respectiva ao erro
let setMens mens =
mens = mens_

//Retorno do erro
```

```
let getErro er =
er  = erro

//Retorno da mensagem de acordo com o erro
let getMens me =
me = mens_

//Atribuições
member this.erro = erro
member this.erro2 = erro
member this.mens_ = mens_
end
```

- Depuração inicial sem inserção de dados nos campos:

- Depuração com inserção de dados e feito o cálculo da área:

- Depuração com teste à classe erro (clicando no botão calcular com um dos campos em branco):



Programa 2:

- **Equação do 2° Grau**

Classe Equação2G

```
module Equação2G
//Variáveis
let mutable delta = ""
let mutable X1 = ""
let mutable X2 = ""
let mutable a = 0.00

type Equação2g(aux1:string, aux2:string, aux3:string) = class

    //Retornando o valor de delta
    let getDelta =
        delta <- ((float(aux2) * float(aux2)) - (4.00 * float(aux1) *
float(aux3))).ToString()

    //Atribuindo o valor de delta e a função raiz a variável
    let raizvl =
        a <- sqrt (float(delta))

    //Retornando o valor de X1
    let getX1 =
        X1 <- ((-float(aux2) + a) / (2.00 * float(aux1))).ToString()

    //Retornando o valor de X2
    let getX2 =
        X2 <- ((-float(aux2) - a) / (2.00 * float(aux1))).ToString()

    member this.aux1 = aux1
    member this.aux2 = aux2
    member this.aux3 = aux3

end
```

Classe Equação2GBLL

```
module Equação2GBLL
```

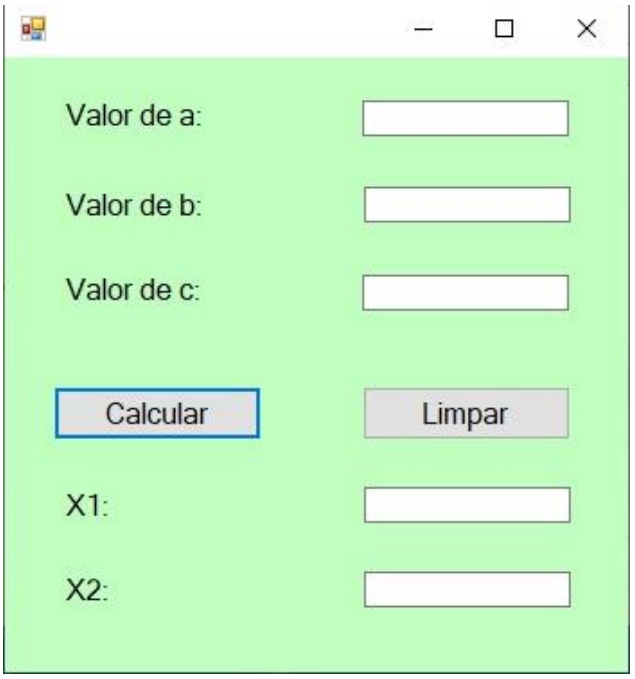
```
open Erro
open Equação2G

let mutable mens = ""

type ValidaDados(_a:string, _b:string, _c:string) = class
    //Variáveis auxiliares
    let mutable v1 = _a
    let mutable v2 = _b
    let mutable v3 = _c

    // Verificações sobre o campo e seus respectivos valores
    let Verifica =
      if v1.Length = 0 then
         mens <- "É necessário preencher o campo do valor A"
      else
      if v2.Length = 0 then
         mens <- "É necessário preencher o campo do valor B"
         else
      if v3.Length = 0 then
         mens <- "É necessário preencher o campo do valor C"

    member this.mens = mens
end
```

Classe Equação2GIHM

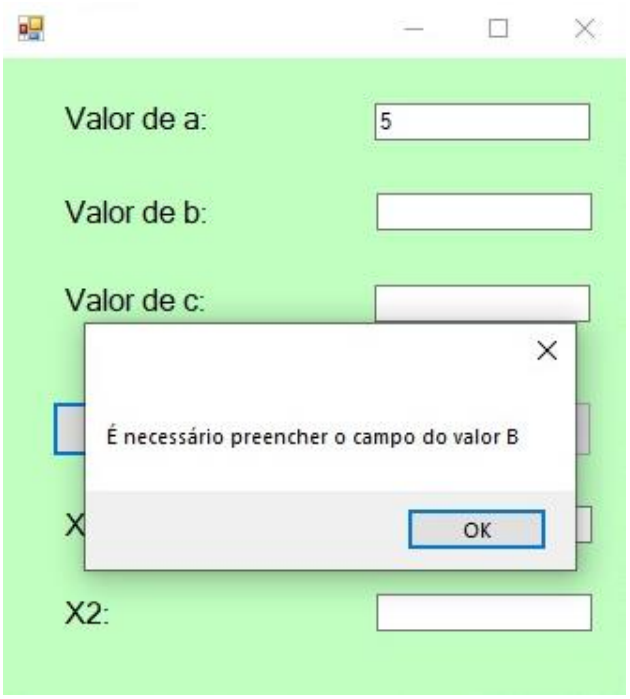
```
module Equação2GIHM.Main

open System
open System.Windows.Forms
open ScfProjectDesign
open TrivialBehind
open Equação2GBLL
open Equação2G

type public EquaçãoIHM(ui: ScfProjectUi) =

    do
        ui.btncalculo.Click.Add <| fun _ ->
           //Chamando a função ValidaDados da classe Equação2GBBL
           ValidaDados(ui.textBox1.Text, ui.textBox2.Text, ui.textBox3.Text) |>
ignore

           //Encadeamento para validação de campos
           if ui.textBox1.Text.Length = 0 then
               MessageBox.Show(mens) |>ignore
           else
           if ui.textBox2.Text.Length = 0 then
               MessageBox.Show(mens) |>ignore
           else
           if ui.textBox3.Text.Length = 0 then
              MessageBox.Show(mens) |>ignore
           else
              let    a    =    Equação2g(ui.textBox1.Text,    ui.textBox2.Text,
ui.textBox3.Text)

              ui.textBox4.Text <- Equação2G.X1
              ui.textBox5.Text <- Equação2G.X2
              ui.textBox1.Enable <- false
```

```
            ui.textBox2.Enable <- false
        //Limpar todos os campos quando o botão "Limpar" for acionado
        ui.btnclear.Click.Add <| fun _ ->
           ui.textBox1.Clear()
           ui.textBox2.Clear()
           ui.textBox3.Clear()
           ui.textBox4.Clear()
           ui.textBox5.Clear()

let registerBehinds() =
  TrivialBehinds.RegisterBehind<ScfProjectUi, EquaçãoIHM>()

[<EntryPoint; STAThread>]
let main argv =
    Application.EnableVisualStyles()
    Application.SetCompatibleTextRenderingDefault(false)
    registerBehinds()
    use form = new ScfProjectForm()
    Application.Run(form)
    0 // return an integer exit code
```

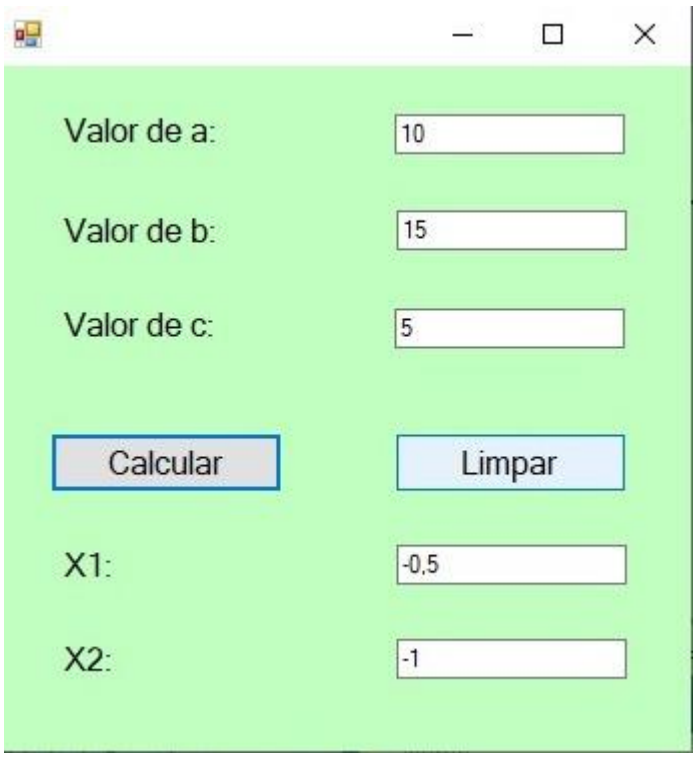
- Depuração inicial, sem inserção de dados nos campos:



- Depuração testando a validação de campos, clicando no botão calcular com campos sem dados:



- Depuração finalizada, após inserção de dados e efetuado o cálculo e exibição do resultado:



Programa 3:

- **Salário Horista**

Classe Horista

```
module Horista

//declarando variável auxiliar
let mutable sbruto = ""

type public Horista1(qtd:string, valor:string) = class

     //setando a 'quantidade de horas'
     let setQtd(_qtd: string)
      qtd = _qtd

     //retornando o valor da 'quantidade de horas'
     let getQtd q =
         q = valor

     //setando o 'valor da hora'
     let setValor(_valor: string)
         valor = _valor

     //retornando o valor do 'valor da hora'
     let getValor v =
          v = valor

     //retornando o valor total/salário bruto já com o cálculo
     let getSBruto =
         sbruto <- ((float(qtd) * float(valor)) / 2.00).ToString()

     //Atribuindo
     member this.qtd = qtd
     member this.valor = valor
     member this.sbruto = sbruto
end
```

Classe HoristaBLL

```
module HoristaBLL

open Erro
open Horista

let mutable mens = ""

type ValidaDados(_vl1:string, _vl2:string) = class

//declarando variáveis auxiliares

    let mutable v1 = _vl1
    let mutable v2 = _vl2

    //declaração das verificações dos campos (se recebeu valor ou não)
    let _Verifica =
      if v1.Length = 0 then
         mens <- "O campo QUANTIDADE DE HORAS é de preenchimento obrigatório..."
      else
      if v2.Length = 0 then
         mens <- "O campo VALOR DA HORA é de preenchimento obrigatório..."


    //se alguns dos campos estiver vazio, exibe uma mensagem de aviso de qual campo
precisa receber o valor
```

```
    member this.mens = mens

end
```

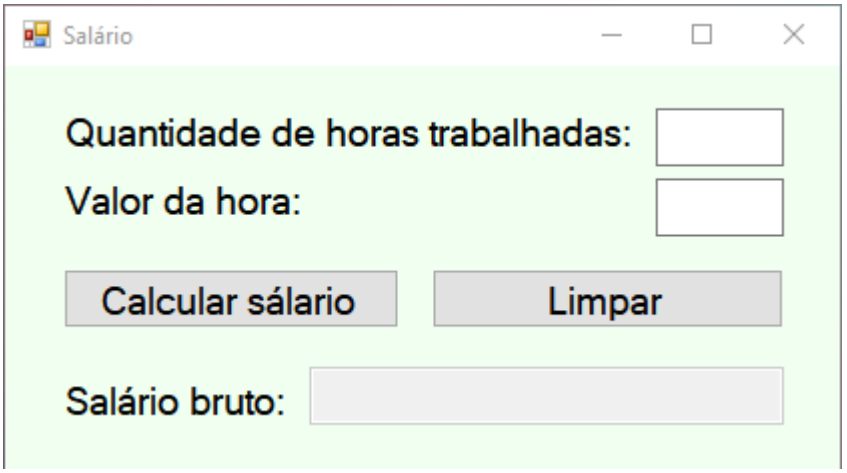
Classe HoristaIHM

```
module HoristaIHM

open System
open System.Windows.Forms
open ScfProjectDesign
open Horista
open HoristaBLL
open TrivialBehind


type HORISTA_IHM(ui:ScfProjectUi) =


      //validação dos campos
      do
      //quando o botão 'calcular salário' for clicado
         ui.button1.Click.Add <| fun _ ->


            let hBLL = ValidaDados(ui.textBox1.Text,ui.textBox2.Text)


            if ui.textBox1.Text.Length = 0 then
               MessageBox.Show(mens) |>ignore
            else
            if ui.textBox2.Text.Length = 0 then
                MessageBox.Show(mens) |>ignore
            else
               let aj = Horista1(ui.textBox1.Text,ui.textBox2.Text)
               ui.textBox3.Text <- aj.sbruto.ToString()


//quando o botão 'limpar' for clicado
         ui.button2.Click.Add <| fun _ ->
         //limpa os campos

             ui.textBox1.Clear()
             ui.textBox2.Clear()
             ui.textBox3.Clear()


let registerBehinds() =
  TrivialBehinds.RegisterBehind<ScfProjectUi, HORISTA_IHM>()

[<EntryPoint; STAThread>]
let main argv =
    Application.EnableVisualStyles()
    Application.SetCompatibleTextRenderingDefault(false)
    registerBehinds()
    use form = new ScfProjectForm()
    Application.Run(form)
    0 // return an integer exit code
```

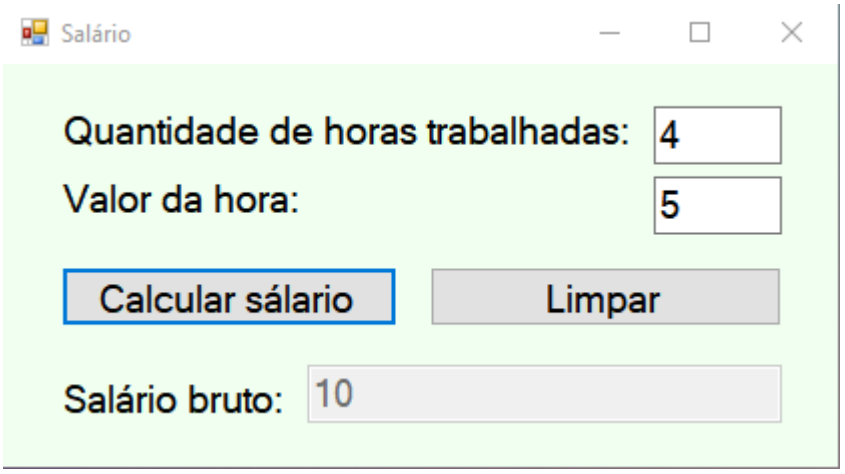
Classe erro

```
module Erro

//declarando variáveis auxiliares
let mutable mens_ = ""
let mutable erro2 = false

type cErro(erro:bool) = class

    //setando o erro
    let setErro _erro =
        _erro = erro

    //setando a mensagem de erro
    let setMens mens =
        mens = mens_

    //Retornando o erro
    let getErro er =
        er  = erro

    //Retornando a mensagem
    let getMens me =
        me = mens_

    //Atribuindo
    member this.erro = erro
    member this.erro2 = erro
    member this.mens_ = mens_
end
```

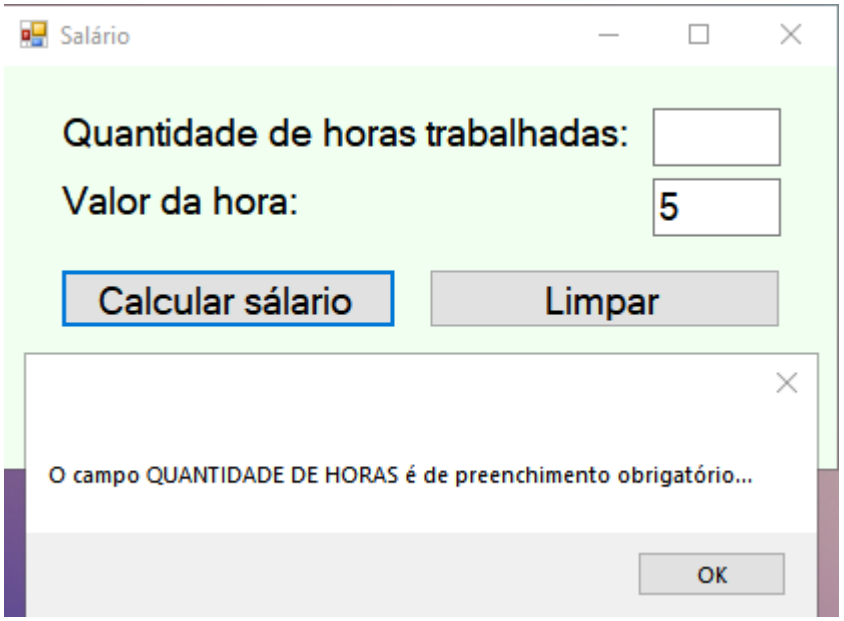
- Depuração da tela inicial



- Depuração com exemplo de cálculo



- Depuração com exemplo de mensagem de erro:



Programa 4:

- **Cadastro Livro**

Classe Livro.fs

```
module Livro

let mutable codigo = ""
let mutable titulo = ""
let mutable autor = ""
let mutable editora = ""
let mutable ano = ""

//seteando código
type setCodigo(cod:string) = class
     let getCod =
         codigo <- cod
```

```
            member this.codigo = cod
        end
        //setando titulo
        type setTitulo(tit:string) = class
            let getTitulo =
                titulo <- tit
            member this.titulo = tit
        end
        //setando autor
        type setAutor(aut:string) = class
            let getAut =
                autor <- aut
            member this.autor = aut
        end
        //setando editora
        type setEditora(edit:string) = class
            let getEditora =
                editora <- edit
            member this.editora = edit
        end
        //setando ano
        type setAno(_ano:string) = class
            let getAno =
                ano <- _ano
            member this.ano = _ano
        end
```

Classe LivroBLL

```
        module LivroBLL

        //chamando classes Livro.fs e Erro.fs
        open Livro
        open Erro

        //criando variáveis auxiliares
        let mutable mens1 = ""
        let mutable mens = ""

        type ValidaCod(cod:string) = class
            //variável auxiliar
            let mutable v1 = cod
            //verificação do código
            let Verifica =
                if v1.Length = 0 then
                   mens1 <- "O código é de preenchimento obrigatório!"

            member this.mens1 = mens1
        end


        type ValidaDados(_vl1:string, _vl2:string, _vl3:string, _vl4:string, _vl5:string)
= class
            //variáveis auxiliares
            let mutable v1 = _vl1
            let mutable v2 = _vl2
            let mutable v3 = _vl3
            let mutable v4 = _vl4
            let mutable v5 = _vl5

            //verificação dos campos
            let Verifica =
```

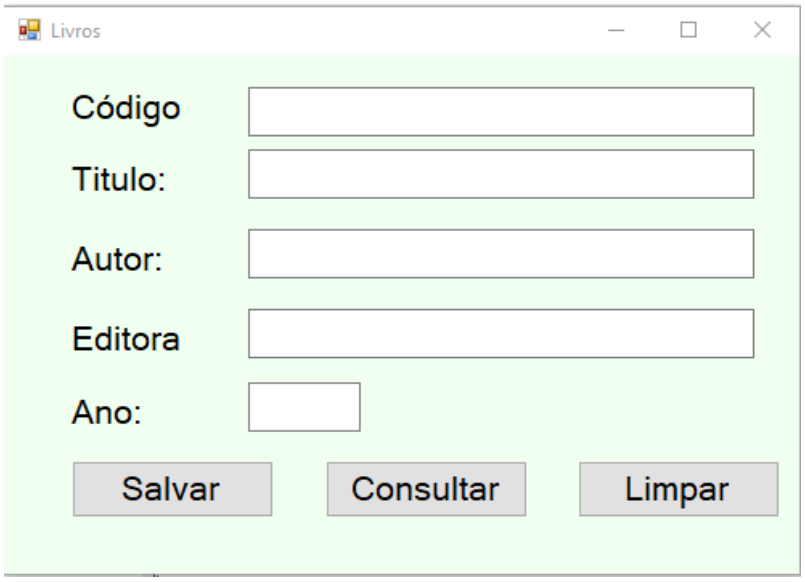
```
            if v1.Length = 0 then
               mens <- "O código é de preenchimento obrigatório!"
            else
            if v2.Length = 0 then
               mens <- "O título é de preenchimento obrigatório!"
            else
            if v3.Length = 0 then
               mens <- "O autor é de preenchimento obrigatório..."
            else
            if v4.Length = 0 then
               mens <- "O editora é de preenchimento obrigatório!"
            else
            if v5.Length = 0 then
               mens <- "O ano é de preenchimento obrigatório!"

        //atribuição da respectiva mensagem a cada campo
        member this.mens = mens


    end
```

Classe LivroIHM

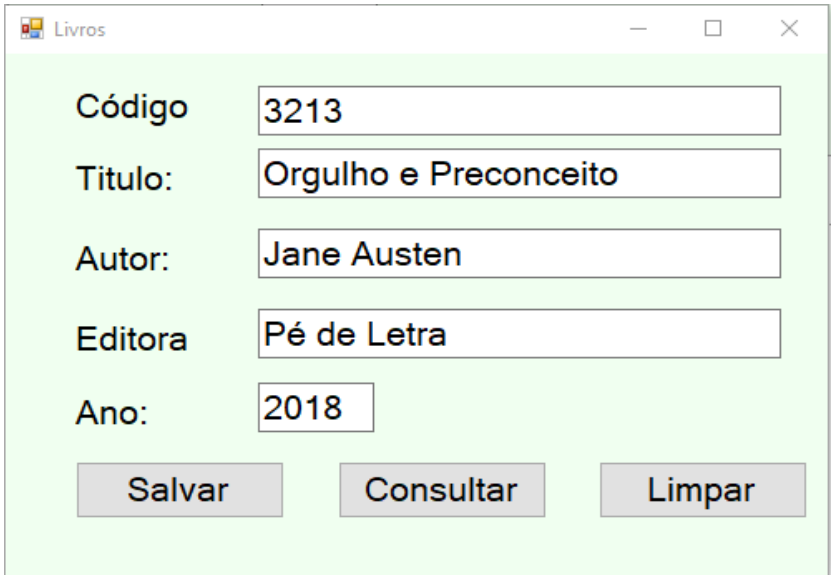
```
    module LivroIHM

    //ligação de todo o sistema formulário com as classes
    open System
    open System.Windows.Forms
    open ScfProjectDesign
    open Livro
    open LivroBLL
    open TrivialBehind

    //chamada do formulário
    type LIVRO_IHM(ui:ScfProjectUi) =
           //validação dos campos
           do
              //função salvar
              ui.button1.Click.Add <| fun _ ->

                  setCodigo(ui.textBox6.Text) |> ignore
                  setTitulo(ui.textBox2.Text) |> ignore
                  setAutor(ui.textBox3.Text)  |> ignore
                  setEditora(ui.textBox4.Text)|> ignore
                  setAno(ui.textBox5.Text)|> ignore

                  LivroBLL.ValidaDados(ui.textBox6.Text,           ui.textBox2.Text,
ui.textBox3.Text, ui.textBox4.Text, ui.textBox5.Text) |> ignore

                  if ui.textBox6.Text.Length = 0 then
                      MessageBox.Show(mens) |>ignore
                  else
                  if ui.textBox2.Text.Length = 0 then
                      MessageBox.Show(mens) |>ignore
                  else
                  if ui.textBox3.Text.Length = 0 then
                     MessageBox.Show(mens) |>ignore
                  else
                  if ui.textBox4.Text.Length = 0 then
                     MessageBox.Show(mens) |>ignore
                  else
                  if ui.textBox5.Text.Length = 0 then
```

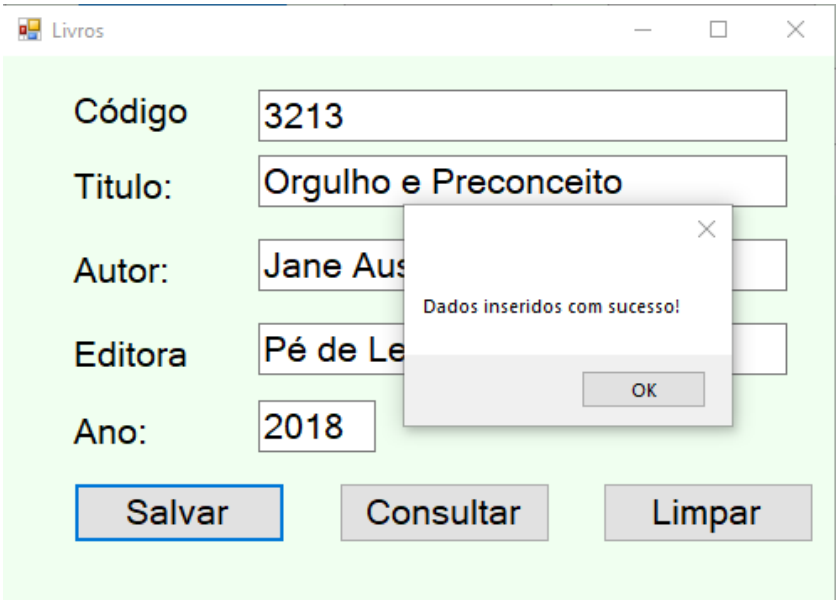
```
                MessageBox.Show(mens) |>ignore
            else
                MessageBox.Show("Dados inseridos com sucesso!") |>ignore

            ui.textBox1.Text <- ""
            ui.textBox2.Text <- ""
            ui.textBox3.Text <- ""
            ui.textBox4.Text <- ""
            ui.textBox5.Text <- ""
            ui.textBox6.Text <- ""
        //função consultar
        ui.button2.Click.Add <| fun _ ->
            setCodigo(ui.textBox6.Text) |>ignore
            LivroBLL.ValidaCod(ui.textBox6.Text) |>ignore

            if ui.textBox6.Text.Length = 0 then
                MessageBox.Show(mens1) |>ignore
            else
              ui.textBox2.Text <- titulo
              ui.textBox3.Text <- autor
              ui.textBox4.Text <- editora
              ui.textBox5.Text <- ano


        //função limpar
        ui.button3.Click.Add <| fun _ ->

                 ui.textBox1.Clear()
                 ui.textBox2.Clear()
                 ui.textBox3.Clear()
                 ui.textBox4.Clear()
                 ui.textBox5.Clear()
                 ui.textBox6.Clear()


let registerBehinds() =
    TrivialBehinds.RegisterBehind<ScfProjectUi, LIVRO_IHM>()

[<EntryPoint; STAThread>]
let main argv =
  Application.EnableVisualStyles()
  Application.SetCompatibleTextRenderingDefault(false)
  registerBehinds()
  use form = new ScfProjectForm()
  Application.Run(form)
  0 // return an integer exit code
```

- Depuração da tela inicial:

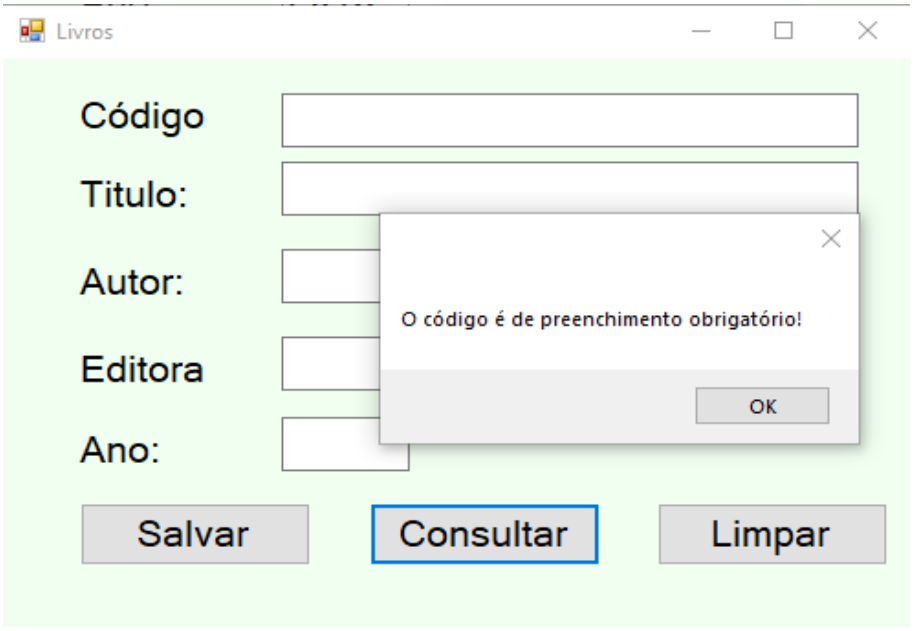


- Depuração com exemplo de cadastro:





- Depuração com exemplo de mensagem de erro



5 O QUE É BANCO DE DADOS?

Antes de iniciar a explicação sobre Banco de Dados, precisamos pontuar e dar algumas definições de palavras importantes para o auxílio no entendimento do assunto. Sendo assim, segundo TOREY et al (2007) ***dado*** é o componente básico de um arquivo, além de ser um elemento que tem significado no mundo real, e que compõe um sistema de arquivos. Exemplificando, podemos citar nome, sobrenome, cidade, bairro e outros. A ***informação*** depois de realizada a interpretação dos dados, consegue-se associar um significado aos dados ou processá-los. Normalmente a informação vem de convenções utilizadas por pessoas por meio de associações aos dados. Já o **conhecimento** é todo o discernimento, é obtido por meio de critérios, e apreciação aos dados e informações.

O dado é um componente básico para compor arquivos. Já o registro nos dará uma informação completa, afinal ele é formado por uma sequência de dados juntos. Um exemplo de registro é o cadastro de funcionários de uma determinada empresa que, geralmente, contém os dados pessoais (nome, RG, CPF, endereço, telefone etc), dados médicos, dados judiciais entre outros tópicos comuns.

Quando há vários registros de bastante gente que são compradores de uma marca, criamos um arquivo chamado “arquivo comprador”. Para ficar clara a diferença entre arquivo, registro e dado, vamos ao exemplo de um cliente da loja de roupas, representando um de seus registros.

O registro é composto por seis itens de dados (campos): código, CPF, nome, rua, bairro, cidade. Dentro do Banco de Dados, as fichas de todos os clientes que estão inseridos formarão o arquivo cliente.

A definição de Banco de Dados é bem ampla, tendo essas como exemplo:

- “Um Banco de Dados é um conjunto de arquivos relacionados entre si” (Chu, 1983);
- “Um Banco de Dados é uma coleção de dados operacionais armazenados, sendo usados pelos sistemas de aplicação de uma determinada organização” (C. J. Date, 1985);
- “Um Banco de Dados é uma coleção de dados relacionais” (Elmasri e Navathe, 2005);
- “Um Banco de Dados é um objeto mais complexo, é uma coleção de dados armazenados e inter-relacionados, que atende às necessidades de vários

usuários dentro de uma ou mais organizações, ou seja, coleções inter-relacionadas de muitos tipos diferentes de tabelas.” (TOREY et al,p.2,2007).

Através das definições literárias, podemos dizer que Banco de Dados é uma forma de coleção de dados relacionados que tem informação sobre algo do mundo real, como por exemplo: lojas, escritórios, bibliotecas ou bancos; ele possui coerência lógica entre os dados e significado. Um Banco de Dados sempre estará associado a aplicações onde existem usuários com interesse aos dados relacionados.

Para falarmos sobre a história do Banco de Dados, temos que voltar aos registros de bibliotecas, negócios em geral, registros de polícia, fichas de pacientes e qualquer informação armazenada de maneira impressa para serem consultadas posteriormente (foi onde tudo começou). Existia um histórico muito longo de informações armazenadas desta maneira e também uma metodologia de indexação e recuperação da informação quando se precisava dela. Esta história não pode ser ignorada, pois há muito a se aprender com os sucessos e fracassos dessas pessoas que manipulavam tais informações. Práticas bem sucedidas e bons projetos de bancos de dados marcam bastante aquela época, onde muito se aprendeu para a criação de bons projetos que alcançam boa performance, segurança e confiabilidade.

5.1 O QUE É CRUD?

Como já sabemos, o CRUD é um acrônimo para as palavras em inglês, Create, Read, Update e Delete. Estas são as operações que podemos utilizar quando estamos resistindo dados de alguma forma, seja uma lista, um arquivo em seu computador ou até mesmo um dado na nuvem. as operações CRUD são as quatro operações pilares do banco de dados.

A abreviação CRUD mapeada para o padrão SQL

- Create (INSERT) – Cria uma nova instancia de dados;
- Read (SELECT) – Ler um dado ou uma lista completa de um data-base;
- Update (UPDATE) – Atualizar um dado ou editar um dado existente;
- Delete (DELETE) – Deletar ou remover algum dado do data-base.

Exemplos das operações:

- Create

```
Create table cliente
(
```

```
    id_cliente integer auto_increment primary key,
    nome varchar(40),
      telefone varchar(12),
    endereço varchar(50)
)
INSERT INTO cliente
(nome, telefone, endereço)
VALUES
(‘Joana das Flores’,
‘12345678’, ‘Rua das Flores’);
```

- Read

```
SELECT * FROM cliente;
```

- Update

```
UPDATE cliente
SET nome= 'jao'
WHERE nome= 'Joana das Flores';
```

- Delete

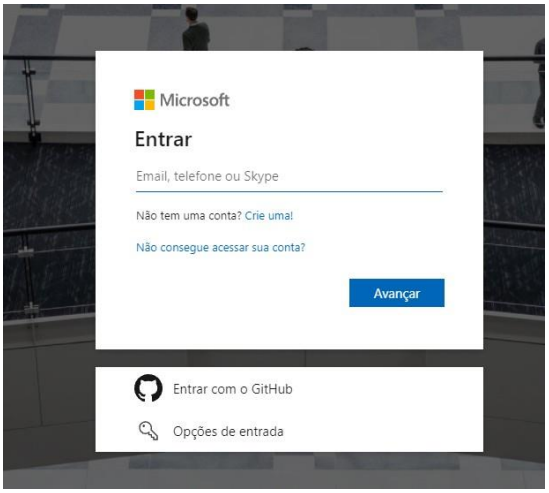
```
DELETE FROM cliente
```

5.2 TUTORIAL – INSTALAÇAO DO BANCO DE DADOS AZURE

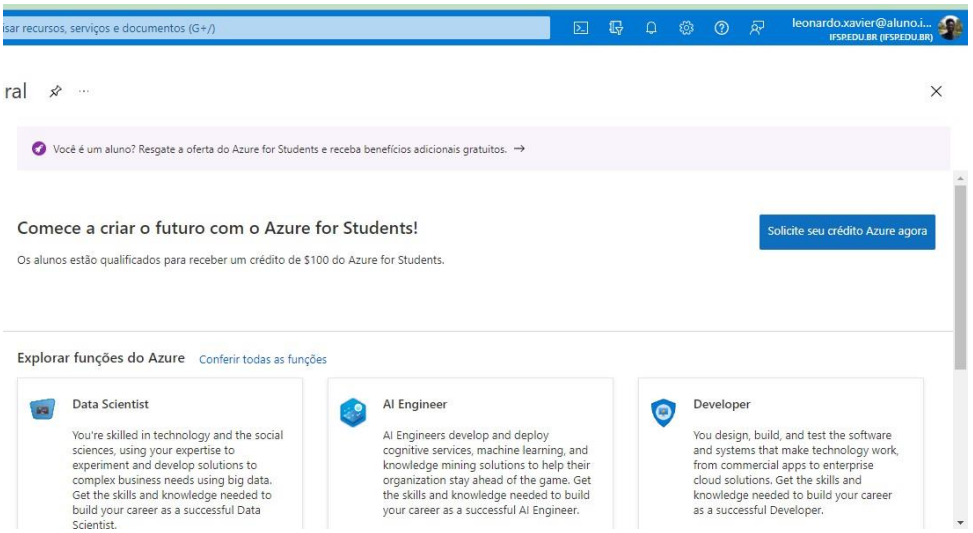
Crie uma conta gratuita no Azure. Na condição de estudante (para não precisar fazer verificação de identidade com cartão de crédito), acesse o seguinte site para baixá-lo:

https://portal.azure.com/#blade/Microsoft_Azure_Education/EducationMenuBlade/overview

Insira os dados de e-mail da Microsoft. De preferência, um e-mail institucional da instituição de ensino (no nosso caso, usamos o @aluno.ifsp.edu.br). Caso não tenha conta na Microsoft, crie uma.

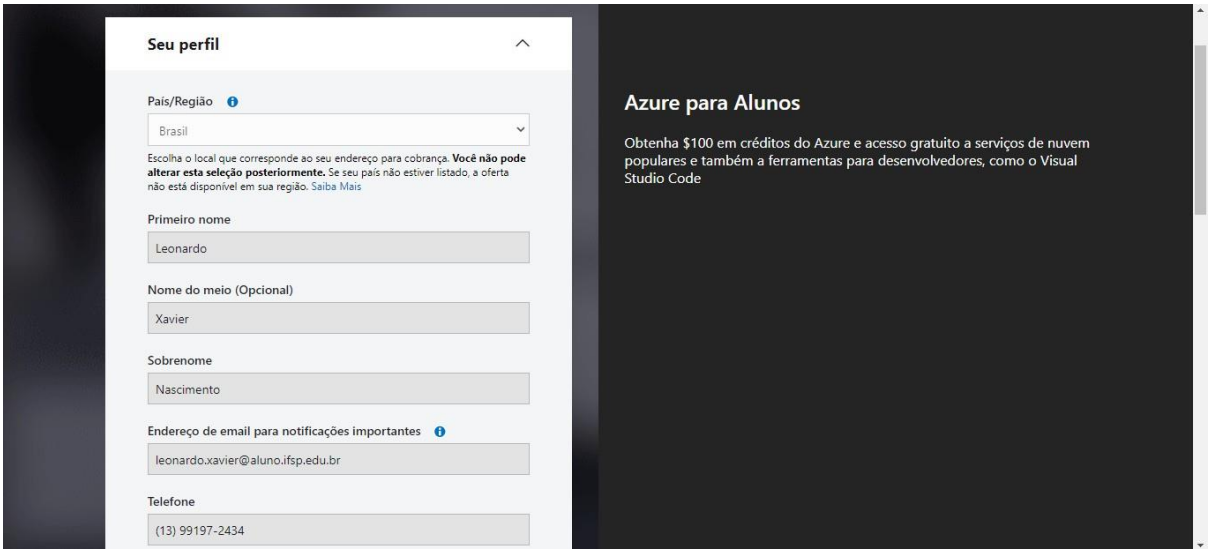


Siga os próximos procedimentos para a criação da conta Azure.



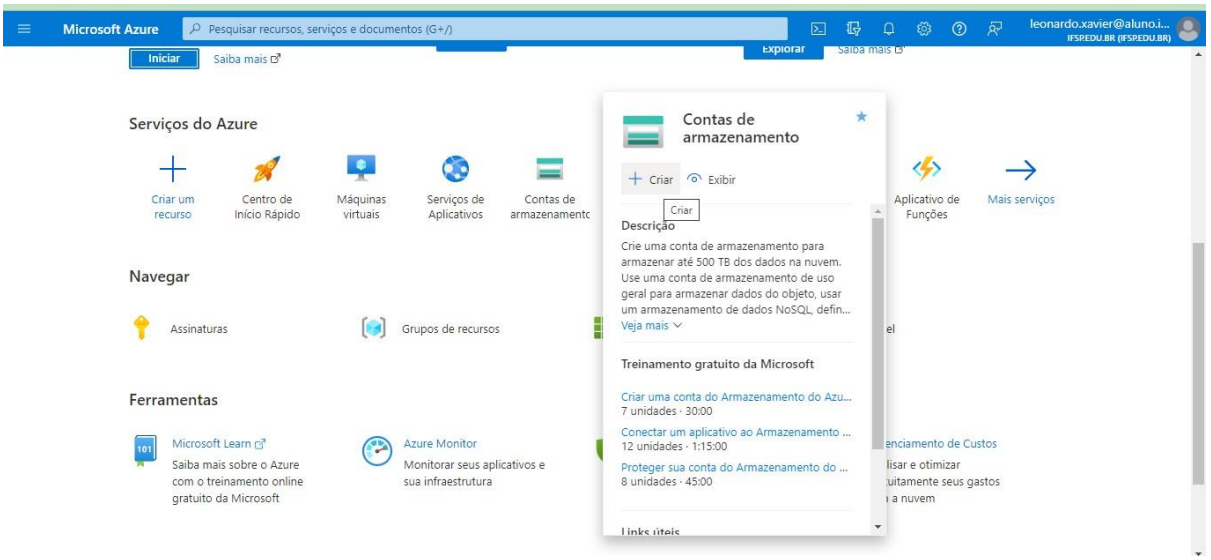
Como demonstrado na imagem acima, clique no botão “Solicite seu crédito Azure agora”. Refere-se a um crédito de 100 dólares que a Microsoft concede aos estudantes, para o uso de funcionalidades no programa.

Preencha com os seus dados e continue a criação da conta.



Siga os próximos passos na página que se abre após a criação da conta Azure.

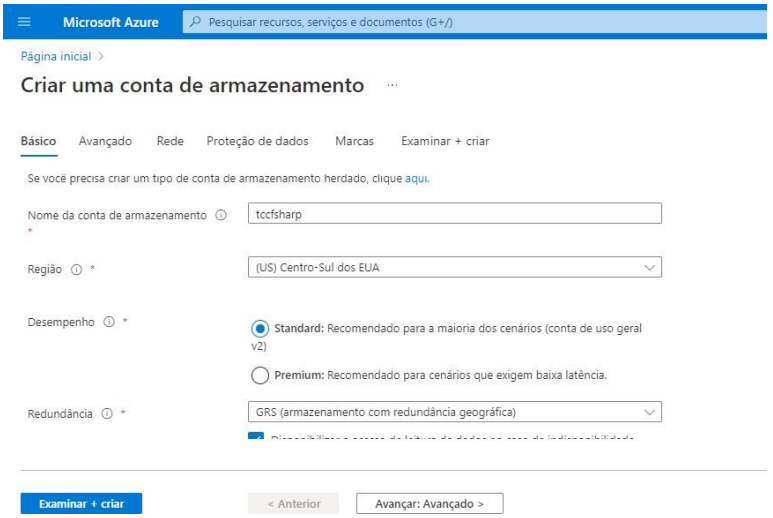
1 Clique em “Contas de armazenamento” (parte central da página, ou cantosuperior esquerdo); em seguida, “Criar”.



2 Coloque as informações que se pede.

- Em “Assinatura” → “Grupo de recursos”: crie um novo grupo e nomeie-o;
- “Em detalhes da instância”: dê nome à conta de armazenamento que estáse criando (pode ser composto por letras **minúsculas** e números, contendo de 3 a 24 caracteres);

- Selecione a região (Sul do Brasil);
- Para as demais informações poderão deixar as seleções feitas automaticamente, pois seguiram a configuração padrão.



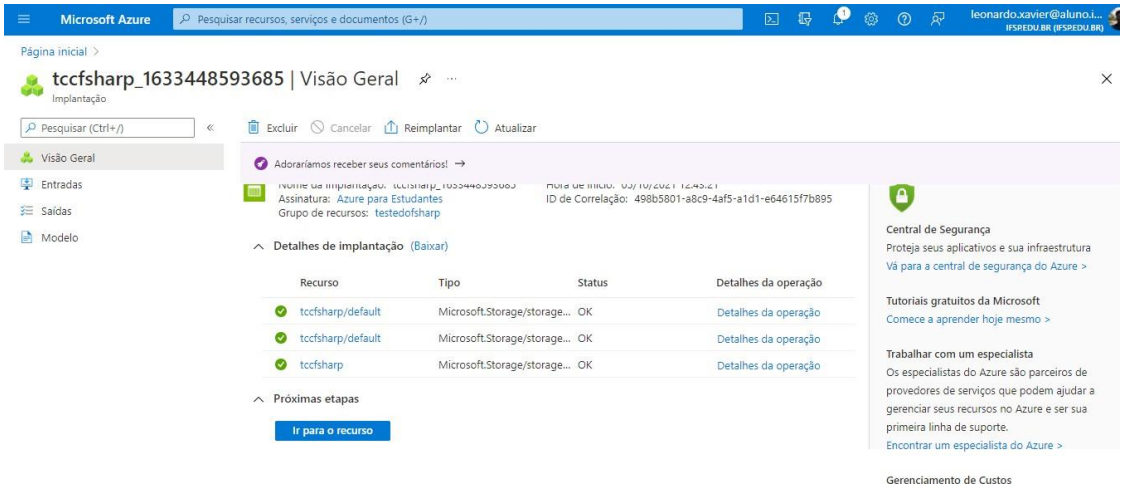
Nas próximas abas (Avançado, Rede, Proteção de Dados, Marcas), deixe o que já está selecionado, pois são configurações padrão.

Em seguida, clique no botão no canto inferior esquerdo “Examinar + criar”.

Se a validação for aprovada, você pode continuar a criar a conta de armazenamento; se a validação falhar, o portal indicará quais configurações precisam ser modificadas.

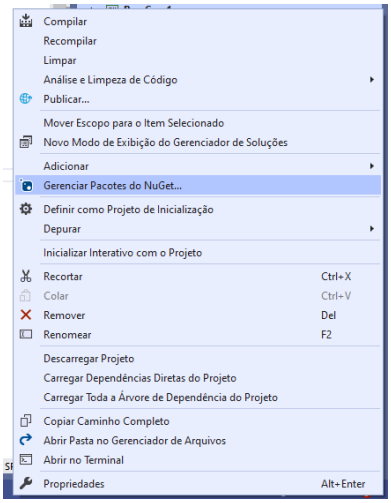
Com a validação aprovada, aperte em “Criar”, no canto inferior esquerdo novamente.

Pronto! Está criada a conta de armazenamento.

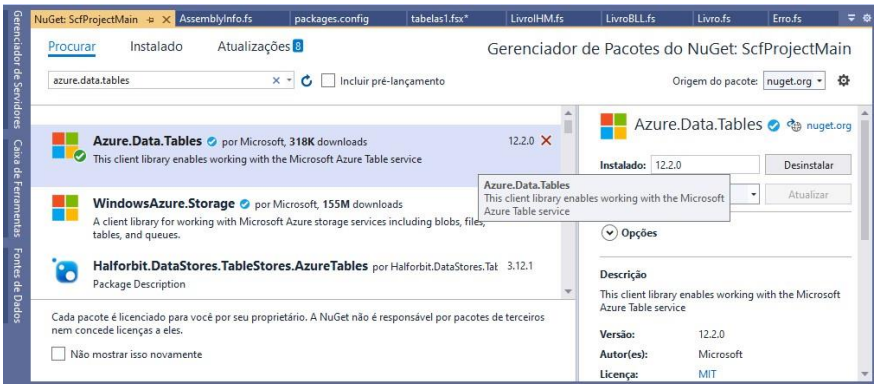


5.3 CONECTANDO O AZURE AO PROGRAMA EM F#

No Visual Studio, clique com o botão esquerdo do mouse sobre o projeto na barra lateral direita; clique em “Gerenciar Pacotes do NuGet”.



Em “Procurar”, pesquise por “Azure.Data.Tables” no “NuGet” e baixe a extensão; pesquise por “Azure.Data.AppConfiguration” e também baixe-a.



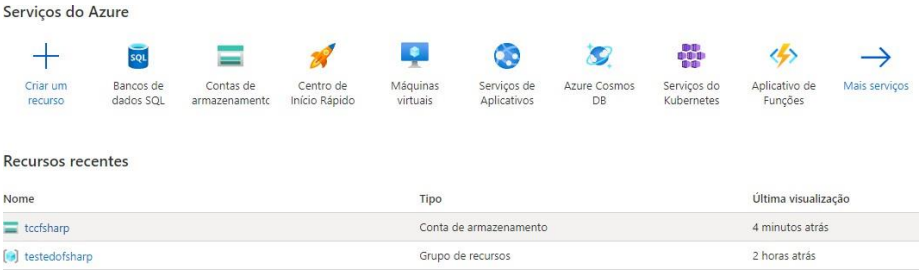
Agora, no código do programa, ou em arquivo script no mesmo projeto, com

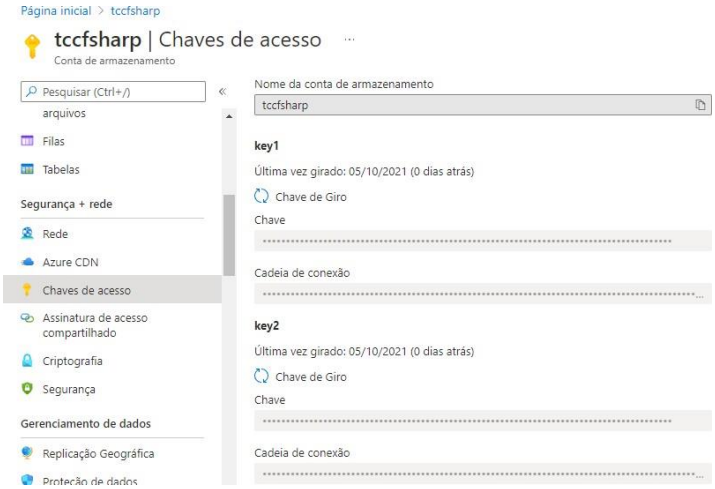
extensão *.fsx*, execute os passos para a conexão com a cadeia de armazenamento Azure.

Abaixo, a declaração do *namespace* para a utilização do Azure (chamamento da biblioteca Azure).

```
open System
open Azure
open Azure.Data.Tables open
System.Linq
open System.Collections.Generic
```

Em seguida, vá no Azure e procure por “Chaves de acesso” da sua conta de armazenamento.





Subindo o cursor lateral, em relação à imagem anterior, clique no ícone “Mostrar as chaves” no canto superior esquerdo da página. Em seguida, copie a “Cadeia de conexão” que aparecerá mais abaixo.

Volte ao programa em F# e copie a chave da cadeia, de acordo com o script de conexão.

```
let storageConnString = ".. 
```

5.4 CRIANDO BANCO DE DADOS EM F#

Após muitas tentativas de realizar o funcionamento da conexão da nossa linguagem F#, com o seu uso no banco de dados, notamos que não tem a possibilidade de usar o banco de dados junto com interface gráfica, pois quando pesquisamos sobre o assunto reparamos que o fsharp é limitado no uso do banco de dados, portanto não é possível usá-lo juntamente com a sua interface gráfica.

Nós tentamos então de alguma maneira fazer o código em C# para acessá-los no F#, o que não deu certo novamente, pois, não conseguimos estabelecer um acesso entre os arquivos, e acreditamos que seja por causa de falta de "reconhecimento" de ambos, logo, se os arquivos não se reconhecem não tem possibilidade de usarmos as variáveis, estabelecendo uma falta de conexão e nos impossibilitando de ter qualquer tipo de acesso.

Devido a essas dificuldades resolvemos simular o acesso ao banco de dados utilizando os elementos de programação do F#, para isso foi utilizado um vetor bidimensional para simular o armazenamento de uma tabela, e com o uso de algumas estruturas de programação foi feita a simulação dos recursos CRUD.

- **Cadastro Livro**

Classe Livro.fs

```
module Livro

let mutable codigo = ""
let mutable titulo = ""
let mutable autor = ""
let mutable editora = ""
let mutable ano = ""

// SETANDO O CÓDIGO
type setCodigo(cod:string) = class
     let getCod =
         codigo <- cod
     member this.codigo = cod
end

// SETANDO O TÍTULO
type setTitulo(tit:string) = class
    let getTitulo =
        titulo <- tit
    member this.titulo = tit
end
```

```
        // SETANDO O AUTOR
        type setAutor(aut:string) = class
            let getAut =
                autor <- aut
            member this.autor = aut
        end

        // SETANDO A EDITORA
        type setEditora(edit:string) = class
            let getEditora =
                editora <- edit
            member this.editora = edit
        end

        // SETANDO O ANO
        type setAno(_ano:string) = class
            let getAno =
                ano <- _ano
            member this.ano = _ano
        end
```

Classe LivroBLL

```
        module LivroBLL

        open Livro
        open Erro

        // VARIÁVEIS AUXILIARES
        let mutable mens1 = ""
        let mutable mens = ""

        //VERIFICAÇÃO DO CÓDIGO DO LIVRO
        type ValidaCod(cod:string) = class

             let mutable v1 = cod

             let Verifica =
                 if v1.Length = 0 then
                    mens1 <- "O código é de preenchimento obrigatório!"

             member this.mens1 = mens1
        end

        // VALIDAÇÃO, VERIFICAÇÃO DE DADOS E ATRIBUIÇÕES
        type ValidaDados(_vl1:string, _vl2:string, _vl3:string, _vl4:string,
_vl5:string) = class
            let mutable v1 = _vl1
            let mutable v2 = _vl2
            let mutable v3 = _vl3
            let mutable v4 = _vl4
            let mutable v5 = _vl5


              // VERFICAÇÃO DE DADO RESPECTIVO A MENSAGEM
            let Verifica =
              if v1.Length = 0 then
                 mens <- "O código é de preenchimento obrigatório!"
              else
              if v2.Length = 0 then
                 mens <- "O título é de preenchimento obrigatório!"
              else
```

```
            if v3.Length = 0 then
               mens <- "O autor é de preenchimento obrigatório..."
            else
            if v4.Length = 0 then
               mens <- "O editora é de preenchimento obrigatório!"
            else
            if v5.Length = 0 then
               mens <- "O ano é de preenchimento obrigatório!"

         // ATRIBUIÇÃO DA MENSAGEM
         member this.mens = mens

      end
```

Classe LivroIHM

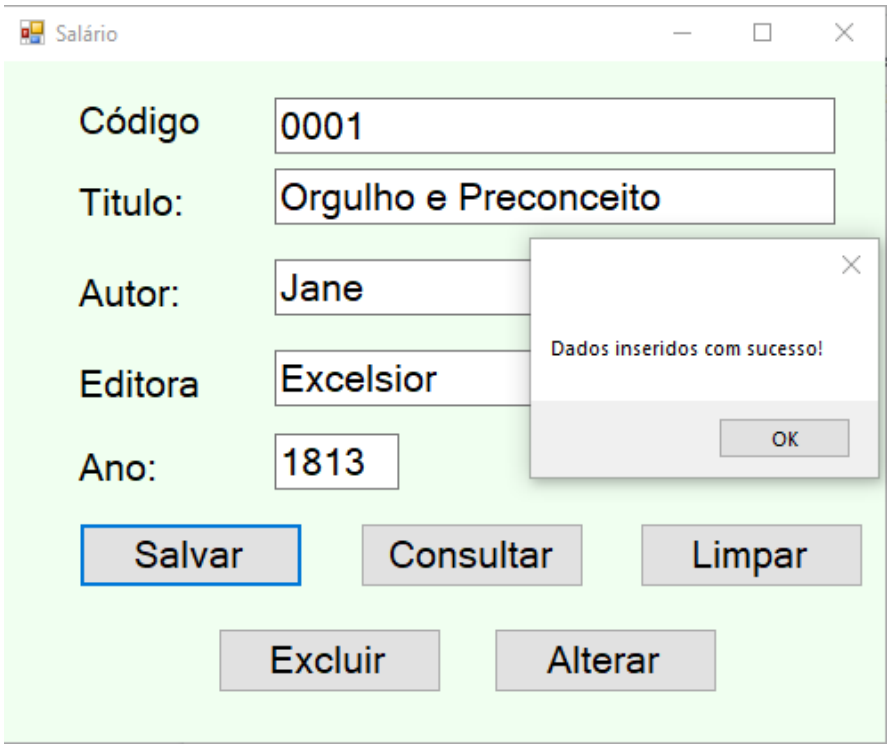
```
      module LivroIHM

      open System
      open System.Windows.Forms
      open ScfProjectDesign
      open Livro
      open LivroBLL
      open TrivialBehind

      type LIVRO_IHM(ui:ScfProjectUi) =

           // CRIAÇÃO DE UM VETOR BIDIMENSIONAL
            let mutable tabela = Array2D.create 100 5 ""
            let mutable count = 0

           // PROGRAMAÇÃO DO BOTÃO DE ADIÇÃO DE DADOS
            do
               ui.button1.Click.Add <| fun _ ->

                  setCodigo(ui.textBox6.Text) |> ignore
                  setTitulo(ui.textBox2.Text) |> ignore
                  setAutor(ui.textBox3.Text)  |> ignore
                  setEditora(ui.textBox4.Text)|> ignore
                  setAno(ui.textBox5.Text)|> ignore

                // CHAMAR A FUNÇÃO VALIDADADOS E ATRIBUIR COMO PARAMETRO
INFORMAÇÕES INSERIDAS PELO USUÁRIO
                  LivroBLL.ValidaDados(ui.textBox6.Text, ui.textBox2.Text,
ui.textBox3.Text, ui.textBox4.Text, ui.textBox5.Text) |> ignore

                // VALIDAÇÃO DOS TEXTBOXS DE INSERÇÃO DE DADOS DO USUÁRIO
                  if ui.textBox6.Text.Length = 0 then
                      MessageBox.Show(mens) |>ignore
                  else
                  if ui.textBox2.Text.Length = 0 then
                      MessageBox.Show(mens) |>ignore
                  else
                  if ui.textBox3.Text.Length = 0 then
                     MessageBox.Show(mens) |>ignore
                  else
                  if ui.textBox4.Text.Length = 0 then
                     MessageBox.Show(mens) |>ignore
                  else
                  if ui.textBox5.Text.Length = 0 then
                     MessageBox.Show(mens) |>ignore
                  else
                 // ADIÇÃO NA TABELA
```

```
        tabela.[count, 0] <- ui.textBox6.Text
        tabela.[count, 1] <- ui.textBox2.Text
        tabela.[count, 2] <- ui.textBox3.Text
        tabela.[count, 3] <- ui.textBox4.Text
        tabela.[count, 4] <- ui.textBox5.Text

        count <- count + 1

        MessageBox.Show("Dados inseridos com sucesso!") |>ignore

    // LIMPEZA DOS CAMPOS
      ui.textBox1.Text <- ""
      ui.textBox2.Text <- ""
      ui.textBox3.Text <- ""
      ui.textBox4.Text <- ""
      ui.textBox5.Text <- ""
      ui.textBox6.Text <- ""

    // PROGRAMAÇÃO DO BOTÃO DE EXCLUSÃO
      ui.button2.Click.Add <| fun _ ->
      setCodigo(ui.textBox6.Text) |>ignore
     LivroBLL.ValidaCod(ui.textBox6.Text) |>ignore

           if ui.textBox6.Text.Length = 0 then
              MessageBox.Show(mens1) |>ignore
           else
             for i=0 to count do
           if ui.textBox6.Text = tabela.[i,0] then
                  ui.textBox2.Text <- tabela.[i,1]
                  ui.textBox3.Text <- tabela.[i,2]
                  ui.textBox4.Text <- tabela.[i,3]
                  ui.textBox5.Text <- tabela.[i,4]


      ui.button3.Click.Add <| fun _ ->

          ui.textBox1.Clear()
          ui.textBox2.Clear()
          ui.textBox3.Clear()
          ui.textBox4.Clear()
          ui.textBox5.Clear()
          ui.textBox6.Clear()


      ui.button4.Click.Add <| fun _ ->
          for i=0 to count do
                 if ui.textBox6.Text = tabela.[i,0] then
          tabela.[i,0] <- ""
          tabela.[i,1] <- ""
          tabela.[i,2] <- ""
          tabela.[i,3] <- ""
          tabela.[i,4] <- ""

       MessageBox.Show("Dados excluído com sucesso!") |>ignore


    // PROGRAMÇÃO DO BOTÃO DE ALTERAR OS DADOS
  ui.button5.Click.Add <| fun _ ->

  for i=0 to count do
  if ui.textBox6.Text = tabela.[i,0] then
```

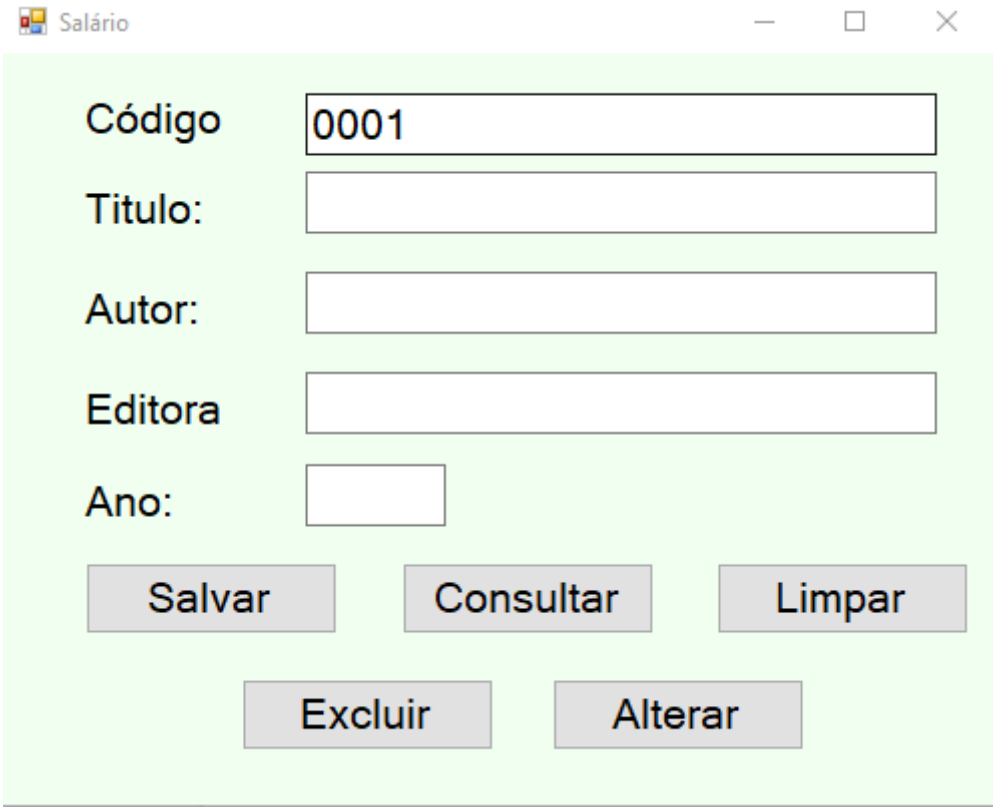
```
            if ui.textBox2.Text <> "" then
             tabela.[i,1] <- ui.textBox2.Text
             MessageBox.Show("Dados alterado com sucesso!") |>ignore
            else
            if ui.textBox3.Text <> "" then
             tabela.[i,2] <- ui.textBox3.Text
             MessageBox.Show("Dados alterado com sucesso!") |>ignore

            else
            if ui.textBox4.Text <> "" then
             tabela.[i,3] <- ui.textBox4.Text
             MessageBox.Show("Dados alterado com sucesso!") |>ignore

            else
            if ui.textBox5.Text <> "" then
             tabela.[i,4] <- ui.textBox5.Text
             MessageBox.Show("Dados alterado com sucesso!") |>ignore
```

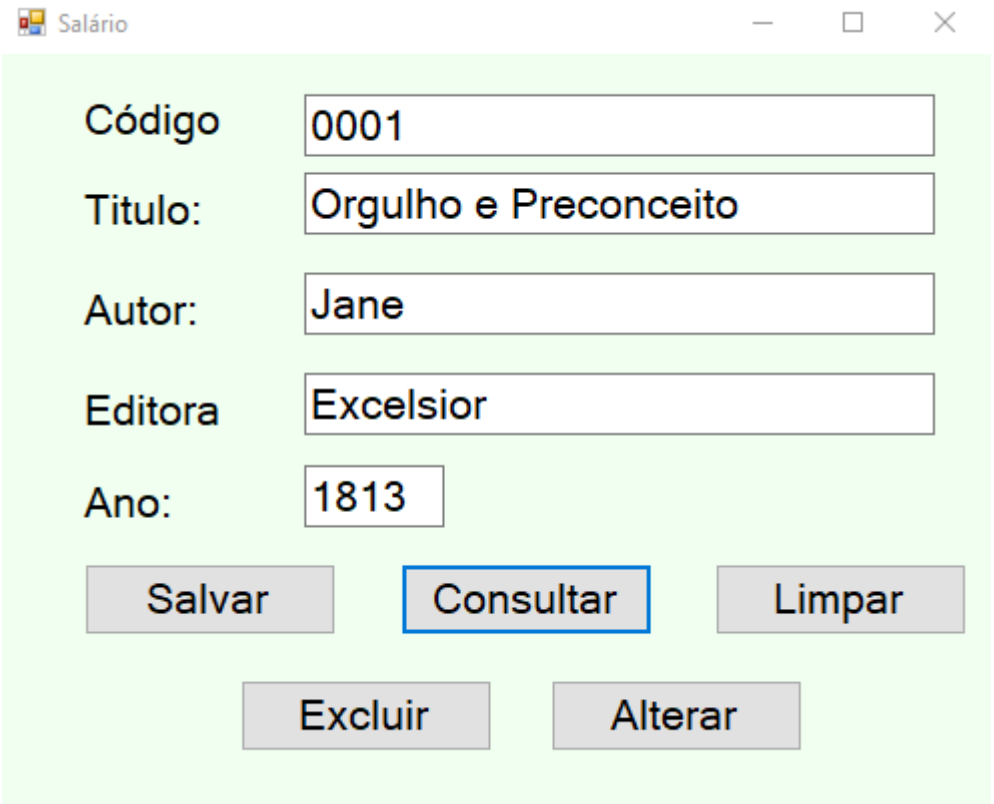
```
let registerBehinds() =
    TrivialBehinds.RegisterBehind<ScfProjectUi, LIVRO_IHM>()

[<EntryPoint; STAThread>]
let main argv =
  Application.EnableVisualStyles()
  Application.SetCompatibleTextRenderingDefault(false)
  registerBehinds()
  use form = new ScfProjectForm()
  Application.Run(form)
  0 // return an integer exit code
```

- Depuração do código – salvando os dados do livro

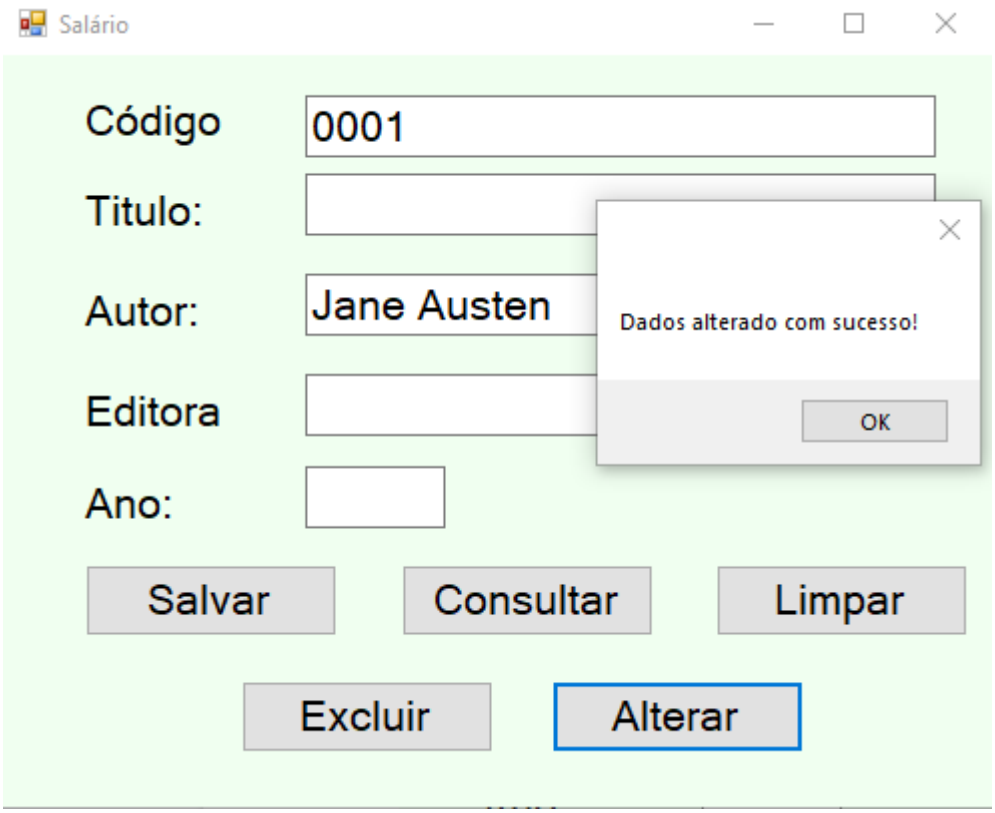


- Consultando os dados

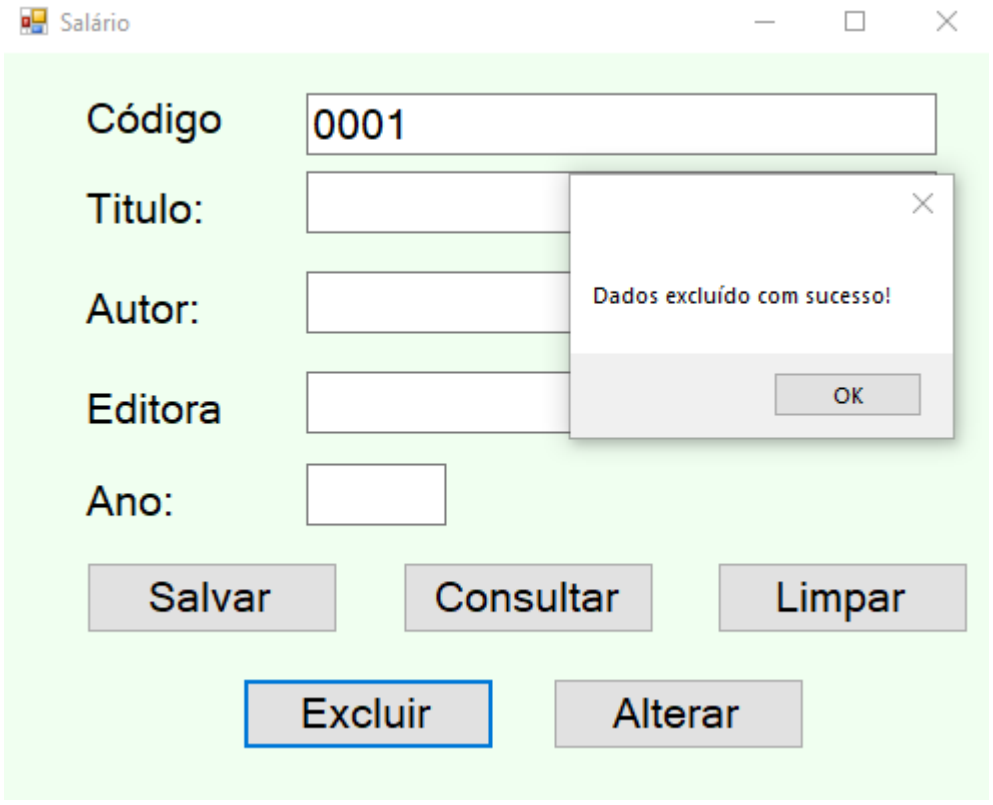




- Alterando os dados:



- Excluindo os dados



6 APLICAÇÃO DE API

API (Application Programming Interface) é uma interface de programação de aplicações, ou seja, permite a criação de plataformas de maneira mais simples e prática, é um conjunto de definições e protocolos usados no desenvolvimento e na integração de software de aplicações. As APIs simplificam a forma como os desenvolvedores integram novos componentes de aplicações a uma arquitetura preexistente. Por isso, elas ajudam na colaboração entre as empresas e as equipes de TI.

6.1 PROVEDORES DE TIPO

Um provedor de tipos é um componente que fornece tipos, propriedades e métodos para se utilizar em um programa e dá suporte significativo para a programação avançada em informações. Os provedores de tipo geram os Tipos Fornecidos, que são gerados pelo compilador do F# e são baseados em uma fonte de dados externa.

A depender dos parâmetros de entrada, determinados tipos podem ser fornecidos. A entrada pode ser uma fonte de dados de exemplo, uma URL apontando diretamente para um serviço externo ou de uma cadeia de conexão para uma fonte de dados.

Provedores de tipo podem ser criados pelo programador, ou podem ser referenciados aqueles já existentes. No F#, comumente, são utilizados alguns provedores de serviços de dados empresariais e da internet. Alguns exemplos:

- O FSharp.Data, que inclui provedores de tipos para formatos de documento JSON, XML, CSV e HTML.
- O swaggerprovider, que inclui dois provedores de tipo geração que geram o modelo de objeto e clientes http para APIs descritas por esquemas OpenApi 3,0 e Swagger 2,0.
- O FSharp. Data. sqlclient tem um conjunto de provedores de tipos para inserção verificada em tempo de compilação de T-SQL em F#.
- E diversos outros da biblioteca .Net em compatibilidade com F#.

Alerta-se que o mecanismo do provedor de tipos é projetado principalmente para se injetar dados estáveis e espaços de informações de serviço. Ou seja, não é projetado para injetar espaços de informações cujo esquema muda durante a execução do programa de maneira relevante.

6.2 DIFERENÇA ENTRE CONSOLE E WINDOWS FORMS

Aplicações console(Console aplpication) não possuem formulários, interface gráfica, elas rodam dentro de uma janela do MSDOS.Normalmente esse tipo de aplicação não possui nenhuma interação com o usuário, possuindo certa limitação para execução de determinadas funcionalidades. No entanto, aplicações console, como aplicações gráficas, permitem que você interaja com banco de dados usando os componentes de acesso a dados como dbExpress, IBExpress, etc. Você pode ainda criar Data Modules para armazenar seus componentes de acesso e pesquisa a base de dados.

Já o Windows Forms é uma estrutura de interface do usuário para criar Windows aplicativos de área de trabalho. Ele fornece uma das maneiras mais produtivas de criar aplicativos de área de trabalho com base no designer visual fornecido no Visual Studio. Funcionalidades como o posicionamento do tipo "arrastar e soltar" de controles visuais facilitam a criação de aplicativos da área de trabalho.

Com o Windows Forms, você desenvolve aplicativos graficamente ricos que são fáceis de implantar, atualizar e trabalhar enquanto estiverem offline ou conectados à internet. Windows Forms aplicativos podem acessar o hardware local e o sistema de arquivos do computador em que o aplicativo está sendo executado.

O que difere o Console Application para o Windows Form diz respeito às funcionalidades. Geralmente, o Console Application é mais utilizados para quem está iniciando na programação. Para quem busca aprender os princípios básicos de programação. Já no Windows Form, é mais utilizando a partir de uma vivência maior com programação. Seja ela C#, VB.NET, C++, etc. Logicamente esta apresenta mais métodos e classes. Onde pode-se desenvolver soluções mais completas e eficientes.

A linguagem é a mesma para Console, Windows Forms, ASP.NET, WPF, etc, aplicações Console são usadas mais em caso em que a interface é desnecessária, onde o software necessita muito de performance, ambientes de automação é comum ver aplicações assim por exemplo, porque a ação que o usuário necessita dar é apenas um "ENTER"...não precisaria de uma camada de apresentação bem elaborada, assim, inutilizando a ideia de usar uma interface gráfica.

6.3 EXEMPLO DE CÓDIGO

No programa, é solicitado que o usuário insira o CEP desejado, e enquanto o usuário não o inserir ele pede novamente. Após a digitação desse CEP, o código faz uma pesquisa na API do site https://viacep.com.br/ws/, e ele retorna as outras informações ao usuário baseadas no CEP digitado, que são: Logradouro; Complemento; Bairro; Localidade; UF; IBGE; Gia e Siaf.

Além disso, o código possibilita que o usuário faça uma nova consulta, e caso ele deseje realizar, o processo se repetirá, e caso não for de sua vontade, o programa finaliza.

```
// Learn more about F# at http://fsharp.org

open System
open FSharp.Data

//Declaração de valores fixos
[<Literal>]
let url = "https://viacep.com.br/ws/01001000/json/"
let api_url = "https://viacep.com.br/ws/"

type CEP = JsonProvider<url>

//Variáveis auxiliares
let mutable resp = ""
let mutable var = false

[<EntryPoint>]
let main argv =

   while resp <> "N" do

   //Entrada dos dados
   printfn "\nDigite o CEP:"
   let e_cep = Console.ReadLine()

   //Atribuições para variável
   let end_CEP = CEP.Load(api_url + e_cep + "/json/")
```

```
//Saída dos valores
printfn "\nCEP: %s" end_CEP.Cep
printfn "Logradouro: %s" end_CEP.Logradouro
printfn "Complemento: %s" end_CEP.Complemento
printfn "Bairro: %s" end_CEP.Bairro
printfn "Localidade: %s" end_CEP.Localidade
printfn "Uf: %s" end_CEP.Uf
printfn "IBGE: %i" end_CEP.Ibge
printfn "Gia: %i" end_CEP.Gia
printfn "Siafi: %i" end_CEP.Siafi

//Pergunta se o usuário deseja fazer uma nova consulta
printfn "\n\nDeseja fazer uma nova consulta? (S/N)"
let a = Console.ReadLine()

 //Verificação de nova consulta
  if a <> "S" && a <> "N" then

    var <- false

    while var <> true do
    //Mensagem para o usuário digitar apenas uma resp válida
       printfn "\nDigite uma resposta válida!"
       printfn "\nDeseja fazer uma nova consulta? (S/N)"
       let b = Console.ReadLine()
       resp <- b

       if resp = "S" || resp = "N" then
          var <- true
  else
     resp <- a

0 // return an integer exit code
```

- Depurando o código:

Console de Depuração do Microsoft Visual Studio

```
Digite o CEP:
11500250

CEP: 11500-250
Logradouro: Rua João Damaso
Complemento:
Bairro: Parque Fernando Jorge
Localidade: Cubatão
Uf: SP
IBGE: 3513504
Gia: 2835
Siafi: 6371


Deseja fazer uma nova consulta? (S/N)
11010020

Digite uma resposta válida!

Deseja fazer uma nova consulta? (S/N)
S

Digite o CEP:
11010020

CEP: 11010-020
Logradouro: Rua Riachuelo
Complemento: lado par
Bairro: Centro
Localidade: Santos
Uf: SP
IBGE: 3548500
Gia: 6336
Siafi: 7071


Deseja fazer uma nova consulta? (S/N)
N
```

REFERÊNCIAS

- MACORATTI, José. Visual Studio Code: **Apresentando o editor multiplataforma da Microsoft**. Disponível em: <https://imasters.com.br/desenvolvimento/visual-studio-code-apresentando-o-editor-multiplataforma-da-microsoft>. Acesso em: 18 mai. 2021.
- SYME, DON. **The Early History of F#. United Kingdom, 2020**. Disponível em: <https://fsharp.org/history/hopl-final/hopl-fsharp.pdf>. Acesso em: 19 mai. 2021.
- JETBRAINS. **AMBIENTE DE DESENVOLVIMENTO INTEGRADO: O que é IDE?** Disponível em: <https://www.jetbrains.com/pt-br/rider/>. Acesso em: 19 mai. 2021.
- Open Collective. **Ionide: a Visual Studio Code package.** Disponível em: <https://opencollective.com/ionide>. Acesso em: 19 mai. 2021.
- CIESLAK, Krzysztof. **Reflexões F#: Ionide - Uma Nova Esperança**. Disponível em: <http://kcieslak.io/Ionide-New-Hope>. Acesso em: 19 mai. 2021.
- ANDRADE, Ana Paula. **O que é F# (F Sharp)?** Disponível em: <https://www.treinaweb.com.br/blog/o-que-e-f-f-sharp/>. Acesso em: 20 mai. 2021.
- Microsoft. **O que é F#. Docs.Microsoft**. Disponível em: <https://docs.microsoft.com/pt-br/dotnet/fsharp/what-is-fsharp>. Acesso em: 20 mai. 2021.
- Código Logo. **F# para iniciantes – Hello World.** 2017. (1m57s). Disponível em: <https://www.youtube.com/watch?v=-s3vpSAXBDA> Acesso em: 20 mai. 2021.
- JETBRAINS. **Use F#.** Disponível em: <https://www.jetbrains.com/pt-br/rider/>. Acesso em: 20 mai. 2021.
- Github. **Ionide: ferramentas de desenvolvedor F#**. Disponível em: <https://github.com/ionide>. Acesso em: 20 mai. 2021.
- COMARTIN, Derek. **IDE .NET rápido e poderoso entre plataformas.** Disponível em: <https://www.jetbrains.com/pt-br/rider/>. Acesso em: 20 mai. 2021.
- GitHub. **Ionide: vscode-fsharp.** Disponível em: <https://github.com/ionide/ionide-vscode-fsharp>. Acesso em: 20 mai. 2021.
- Open Collective. **Ionida**. Disponível em: <https://opencollective.com/ionide>. Acesso em: 21 mai. 2021.
- FINCHER, Mitch. **Aprendendo F Sharp pelo Exemplo**. Disponível em: <https://www.fincher.org/tips/Languages/fsharp.shtml>. Acesso em: 15 jun. 2021.

- FSHARP.CORE. **Módulo de Operadores**. Disponível em: <https://fsharp.github.io/fsharp-core-docs/reference/fsharp-core-operators.html>. Acesso em: 15 jun. 2021.
- TELLES, Diego. **Operadores**. Disponível em: <https://medium.com/unicornacademy/operadores-3b7d4262c51a>. Acesso em: 15 jun. 2021.
- MICROSOFT. **Sobrecarga de operador**. Disponível em: <https://docs.microsoft.com/pt-br/dotnet/fsharp/language-reference/operator-overloading>. Acesso em: 15 jun. 2021.
- MICROSOFT. **Operadores Bit a Bit**. Disponível em: <https://docs.microsoft.com/pt-br/dotnet/fsharp/language-reference/symbol-and-operator-reference/bitwise-operators>. Acesso em: 15 jun. 2021.
- ALVES, Gustavo Furtado de Oliveira. **Conheça os operadores lógicos**. Disponível em: <https://dicasdeprogramacao.com.br/operadores-logicos/>. Acesso em: 15 jun. 2021.
- FONSECA, Elton. **Operadores lógicos**. Disponível em: <https://www.treinaweb.com.br/blog/operadores-logicos>. Acesso em: 15 jun. 2021.
- CRUZ, C. **Lógica de Programação - Operadores Lógicos**. Disponível em https://autociencia.blogspot.com/2016/08/logica-de-programacao-operadores-logicos.html. Acesso em: 15 jun. 2021.
- MICROSOFT. **Operadores bit a bit e de deslocamento (referência do C#)**. Disponível em: <https://docs.microsoft.com/pt-br/dotnet/csharp/language-reference/operators/bitwise-and-shift-operators>. Acesso em: 15 jun. 2021.
- REIS, Fábio dos. **Lógica de Programação – Operadores e Expressões Lógicas**. Disponível em: <http://www.bosontreinamentos.com.br/logica-de-programacao/09-logica-de-programacao-operadores-e-expressoes-logicas/>. Acesso em: 15 jun. 2021.
- MORAIS, ROMULO MACHADO. **OPERADORES ARITMÉTICOS | Lógica de Programação**. **Roteiro: Sharpax.** [*S. l.*: *s. n.*], 2019. Disponível em: <https://www.youtube.com/watch?v=DMiAvs2qCtY>. Acesso em: 15 jun. 2021.

- DEVMEDIA. **Sobrecarga de Operadores**. Disponível em: <https://www.devmedia.com.br/sobrecarga-de-operadores/1605>. Acesso em: 15 jun. 2021.
- MICROSOFT. **Referência de símbolo e operador**. Disponível em: <https://docs.microsoft.com/pt-br/dotnet/fsharp/language-reference/symbol-and-operator-reference/>. Acesso em: 18 jun. 2021.
- MICROSOFT. **Matrizes**. Disponível em: <https://docs.microsoft.com/pt-br/dotnet/fsharp/language-reference/arrays>. Acesso em: 18 jun. 2021.
- BANAS, Derek. **F# Tutorial**. Disponível em: <https://www.youtube.com/watch?v=c7eNDJN758U&t=2347s>. Acesso em: 18 jun. 2021.
- IMPACTA. **Você sabe o que é Visual Studio?** Disponível em: <https://www.impacta.com.br/blog/voce-sabe-o-que-e-visual-studio/>. Acesso em: 21 jun. 2021.
- MICROSOFT. **Loops: para ... na expressão**. Disponível em: <https://docs.microsoft.com/en-us/dotnet/fsharp/language-reference/loops-for-in-expression>. Acesso em: 21 jun. 2021.
- MICROSOFT. **Loops: expressão for...to**. Disponível em: <https://docs.microsoft.com/pt-br/dotnet/fsharp/language-reference/loops-for-to-expression>. Acesso em: 21 jun. 2021.
- MICROSOFT. **Loops: expressão for...in**. Disponível em: <https://docs.microsoft.com/pt-br/dotnet/fsharp/language-reference/loops-for-in-expression>. Acesso em: 21 jun. 2021.
- MICROSOFT. **Loops: expressão while...do**. Disponível em: <https://docs.microsoft.com/pt-br/dotnet/fsharp/language-reference/loops-while-do-expression>. Acesso em: 21 jun. 2021.
- TUTORIAISPOLNT. **F# - Correspondência de padrões**. Disponível em: <https://www.tutorialspoint.com/fsharp/fsharp_pattern_matching.htm>. Acesso em: 22 jun. 2021.
- MICROSOFT. **Formatação de texto sem formatação**. Disponível em: <https://docs.microsoft.com/pt-br/dotnet/fsharp/language-reference/plaintext-formatting>. Acesso em: 22 jun. 2021.

- MACORATTI, José Carlos. **4 formas distintas de calcular o fatorial**. Disponível em: <http://www.macoratti.net/20/05/c_3fatorial1.htm>. Acesso em: 22 jun. 2021.
- MICROSOFT. **Funções**. Disponível em: <https://docs.microsoft.com/pt-br/dotnet/fsharp/language-reference/functions/>. Acesso em: 22 jun. 2021.
- TUTORIALSPOINTS. **F# - Estrutura do Programa.** Disponível em: <https://www.tutorialspoint.com/fsharp/fsharp_program_structure.htm>. Acesso em: 22 jun. 2021.
- MICROSOFT. **Campos explícitos: a palavra-chave val.** Disponível em: <https://docs.microsoft.com/pt-br/dotnet/fsharp/language-reference/members/explicit-fields-the-val-keyword>. Acesso em: 22 jun. 2021.
- MICROSOFT. **switch (Referência em C#).** Disponível em: <https://docs.microsoft.com/pt-br/dotnet/csharp/language-reference/keywords/switch>. Acesso em: 22 jun. 2021.
- MICROSOFT. **if-else (Referência de C#).** Disponível em: <https://docs.microsoft.com/pt-br/dotnet/csharp/language-reference/keywords/if-else>. Acesso em: 22 jun. 2021.
- MICROSOFT. **Expressões condicionais: if...then...else.** Disponível em: <https://docs.microsoft.com/pt-br/dotnet/fsharp/language-reference/conditional-expressions-if-then-else>. Acesso em: 22 jun. 2021.
- CANALTECH (org.). **O que é GUI?** Disponível em: https://canaltech.com.br/produtos/O-que-e-GUI/. Acesso em: 14 jul. 2021.
- DEVMEDIA (org.). **Orientação a Objetos - simples assim!** Disponível em: https://www.devmedia.com.br/orientacao-a-objetos-simples-assim/3254. Acesso em: 14 jul. 2021.
- DOCS, Microsoft (org.). **Como adicionar um arquivo de configuração de aplicativo a um projeto C#**. 2016. Disponível em: https://docs.microsoft.com/pt-br/visualstudio/ide/how-to-add-app-config-file?view=vs-2019. Acesso em: 14 jul. 2021.
- DOCS, Microsoft (org.). **Como criar um manifesto de pacote**. 2016. Disponível em: https://docs.microsoft.com/pt-br/visualstudio/deployment/how-to-create-a-package-manifest?view=vs-2019. Acesso em: 14 jul. 2021.

- DOCS, Microsoft (org.). **Geração de manifesto no Visual Studio**. 2016. Disponível em: https://docs.microsoft.com/pt-br/cpp/build/manifest-generation-in-visual-studio?view=msvc-160. Acesso em: 15 jul. 2021.
- DOCS, Microsoft (org.). **High DPI support in Windows Forms**. 2017. Disponível em: https://docs.microsoft.com/en-us/dotnet/desktop/winforms/high-dpi-support-in-windows-forms?view=netframeworkdesktop-4.8. Acesso em: 15 jul. 2021.
- VAINIO, Ville M. **Creating a Windows Forms GUI with F#**. 2018. Disponível em: https://medium.com/@vivainio/creating-a-windows-forms-gui-with-f-968b3ae75a82. Acesso em: 15 jul. 2021.
- CANALTECH (org.). **O que é GUI?** Disponível em: https://canaltech.com.br/produtos/O-que-e-GUI/. Acesso em: 14 jul. 2021.
- W, Scott. **F# para diversão e lucro.** Disponível em: <https://fsharpforfunandprofit.com/posts/classes/>. Acesso em: 10 set. 2021.
- SOUZA, Ivan de. **Banco de dados: saiba o que é, os tipos e a importância para o site da sua empresa.** Disponível em: <https://rockcontent.com/br/blog/banco-de-dados/>. Acesso em: 06 out. 2021.
- ALVES, Gustavo Furtado de Oliveira. **A história dos bancos de dados**. Disponível em: <https://dicasdeprogramacao.com.br/a-historia-dos-bancos-de-dados/>. Acesso em: 10 set. 2021.
- ANGELO PUBLIO. **CRUD: o que é este conceito no Desenvolvimento de Sistemas**. Disponível em: <https://angelopublio.com.br/blog/crud/>. Acesso em: 10 set. 2021.
- ALMEIDA, Aloiso. **O que é CRUD? E porque você deveria aprender a criar um**. Disponível em: <https://devporai.com.br/o-que-e-crud-e-porque-voce-deveria-aprender-a-criar-um/>. Acesso em: 10 set. 2021.
- SAKURAI, Rafael Guimarães. **JPA – Exemplo de DAO (CRUD).** Disponível em: <http://www.universidadejava.com.br/materiais/jpa-exemplo-crud/>. Acesso em: 10 set. 2021.
- DEVMEDIA. **Como fazer um CRUD a partir de templates.** Disponível em: < https://www.devmedia.com.br/como-fazer-um-crud-a-partir-de-templates/33797>. Acesso em: 10 set. 2021.

- GABRIEL SCHADE. **Extraindo dados da web - JSON.** Disponível em: < https://gabrielschade.github.io/2017/12/28/web-data-json.html>. Acesso em: 05 nov. 2021.
- FSPROJECTS. **JSON Type Provider.** Disponível em: < https://fsprojects.github.io/FSharp.Data/library/JsonProvider.html>. Acesso em: 05 nov. 2021.
- TAKE BLIP. **API: conceito, exemplos de uso e importância da integração para desenvolvedores.** Disponível em: <https://www.take.net/blog/tecnologia/api-conceito-e-exemplos/>. Acesso em: 16 nov. 2021.
- CANALTECH. **O que é API?** Disponível em: <https://canaltech.com.br/software/o-que-e-api/>. Acesso em: 16 nov. 2021.
- CARTER, Phillip. **Provedores de tipos**. Disponível em: <https://docs.microsoft.com/pt-br/dotnet/fsharp/tutorials/type-providers/>. Acesso em: 17 nov. 2021.
- MICROSOFT. **Guia da área de trabalho (Windows Forms .net)**. Disponível em: <https://docs.microsoft.com/pt-br/dotnet/desktop/winforms/overview/?view=netdesktop-5.0>. Acesso em: 17 nov. 2021.
- TECHOPEDIA. **Console Aplication.** Disponível em: <https://www.techopedia.com/definition/25593/console-application-c>. Acesso em: 18 nov. 2021.