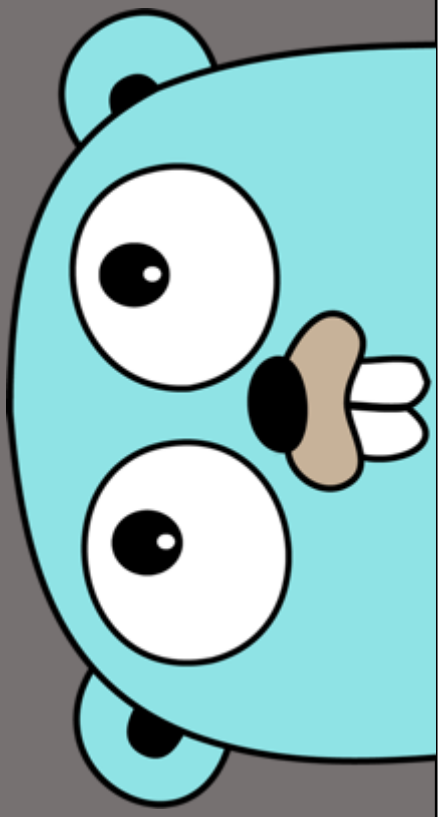
A linguagem de programação GO

Aprendendo uma nova linguagem
de um jeito leve e rápido



```
package main
Import “fmt”
Func main() {
fmt.Println ( DESENVOLVIDO POR:
-Brandon Oliveira Simões
-Camily Vitória da Silva Santos
-Giovanna da Silva de Oliveira
-Laiza dos Santos Reis
-Nathália Regina Vieira Teixeira
-Nicoly Lopes dos Santos
-Paloma dos Santos Rezende Andrade
-Samuel Leite Nunes)
```

CUBATÃO, 2021

SUMARIO

1 HISTÓRICO DA GO E SUAS PRINCIPAIS CARACTERÍSTICAS

Go é uma linguagem de programação que se originou na empresa da Google, em 2007, como resultado DA solução de alguns obstáculos encarados pela empresa ao desenvolver sistemas que operam em sua infraestrutura. Estes obstáculos apareceram com base no desempenho de processadores multinúcleos, sistemas divididos em redes, computação massiva e modelo de computação web, a despeito de transformar a computação demasiadamente mais forte, o que ocasionou uma série de complexidades.

Sucedeu-se muitas investidas fracassadas de solucionar alguns destes obstáculos com sistemas desenvolvidos em linguagens já existentes, como em C++, Java e Python. No entanto, ainda que qualquer uma dessas linguagens tenha atributos interessantes, o ideal era obter todos estes atributos exclusivamente uma linguagem.

Características como uma linguagem compilada, garbage-collector, tipagem estática, trabalhar com concorrência, sintaxe fácil e dentre outras facilidades foram agrupadas na linguagem GO.

Esses sistemas têm milhares de linhas de código e milhares de engenheiros de software trabalham que todos os dias, resultando em um processo de fusão e compilação muito complicado e demorado.

A linguagem Go foi criada como um projeto interno por Ken Thompson, Rob Pike e Robert Griessemer. Em 2008, Go passou de um projeto de meio período para um projeto de período integral no Google. É claro que, além do fundador, muitos outros engenheiros participaram do projeto. Em 2009, Go tornou-se Open Source e foi lançado como um código livre em novembro de 2009. Trata-se de uma versão inspirada na linguagem C e possui recursos avançados como garbage collection e abstração para estruturas de dados. Em 2010, a linguagem começou a ser adotada por desenvolvedores de fora do Google.

Go lançou a versão oficial 1 oficial em 2012. Em 2013, as versões 1.1 e 1.2 foram lançadas. Em meados de 2014, a versão 1.3 foi lançada e, finalmente, a versão 1.4 foi lançada. Seus desenvolvedores acreditam que seu pacote de software padrão é muito bom e completo, o que elimina a necessidade de dependências externas e simplifica seu gerenciamento. Atualmente, muitas empresas adotaram o uso da linguagem Go e, porventura, o Google é a primeira empresa a utilizá-la.

2 AMBIENTES DE DESENVOLVIMENTO INTEGRADO (IDE'S)

As linguagens de programação foram criadas para que o programador conseguisse transmitir informações para a máquina. E, através de regras específicas, códigos e símbolos, permite que o programador crie programas responsáveis pelo controle físico e lógico da máquina.

IDE, abreviação de *Integrated Development Environment*, que traduzindo para o português significa Ambiente de Desenvolvimento Integrado. É um software criado com a finalidade de se obter mais praticidade no desenvolvimento de aplicações, juntando as principais ferramentas de desenvolvimento, como escrever, compilar e testar as aplicações em uma única interface gráfica (GUI), facilitando o trabalho do programador. Ela também pode disponibilizar uma biblioteca de códigos, encontrar erros de digitação e identificar problemas no código-fonte (debug).

A linguagem de programação escolhida foi a GO, e a partir dela analisamos três opções de IDE.

Então entramos em um impasse: qual seria o melhor IDE para programar em Go? A resposta curta é que não há uma resposta clara. É uma questão profundamente subjetiva e depende do que o usuário deseja.

LiteIDE é um Go IDE simples e de código aberto. É notável por ser o primeiro IDE a direcionar diretamente para o passado em 2012. É um C ++ Qt, o que significa que se parece e se sente semelhante a outros compiladores, uma vez que foi projetado diretamente para Go. LiteIDE tem uma série de recursos úteis para desenvolvedores diretamente da caixa, incluindo comandos de construção configuráveis, um editor de código avançado e amplo suporte. Outros recursos incluem gerenciamento de código, um depurador gdb e Delve, preenchimento automático e temas com WordApi, sistema baseado em tipo MIME.

Visual Studio Code é bastante popular, pois também se trata de um IDE de código aberto da Microsoft e suporta a linguagem escolhida. Com a extensão vscode-go, os desenvolvedores ficam a par de mais recursos, incluindo integração com várias ferramentas Go, preenchimento inteligente com IntelliSense, integração Git embutida, a capacidade de depurar código direto do editor. O VScode é altamente extensível, com várias opções de personalização por meio de suas muitas extensões. Ele também oferece

suporte em dezenas de idiomas, o que torna compreensível por que foi classificado como a ferramenta de desenvolvedor mais popular na Pesquisa de Desenvolvedor de 2018 do Stack Overflow.

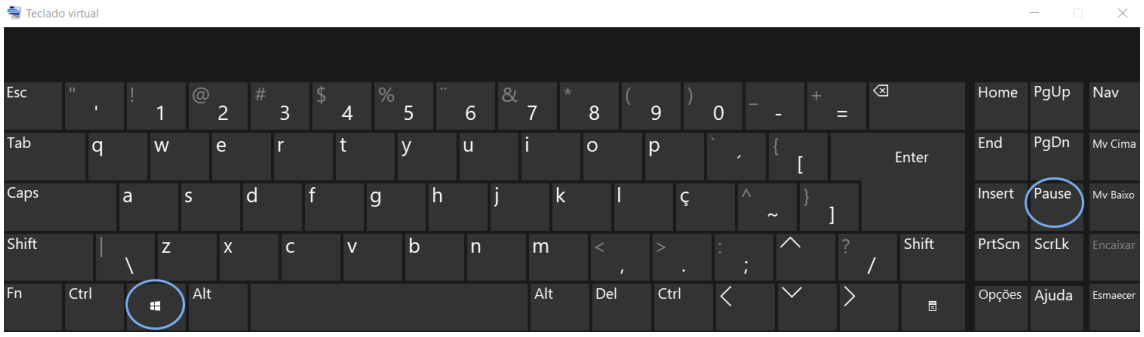
Goji é uma estrutura Golang rápida e leve para desenvolvimento web. É a combinação certa de capacidade de composição e empresas. Muitas das organizações que estão trabalhando em projetos da web preferem Goji. Ele tem um multiplexador de solicitação HTTP minimalista semelhante ao net / HTTP ServeMux. Inclui recursos de desligamento suave, pilha de middleware configurável e padrões de URL.

Dadas estas opções, a IDE escolhida foi Vscode (Visual Studio Code), pela sua simplicidade e usabilidade na hora de desenvolver.

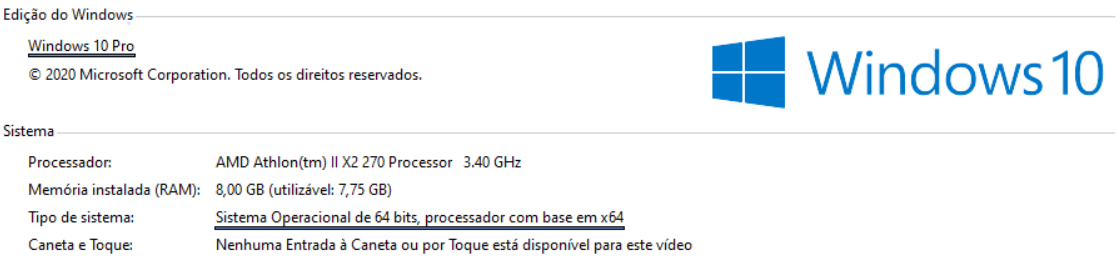
2.1 REQUISITOS PARA INSTALAÇÃO DA LINGUAGEM GO

Para que seja possível a utilização da linguagem Go, antes é necessário realizar a instalação de seus recursos. Do contrário, o computador não conseguirá identificar as palavras-chave necessárias para a escrita do código, muito menos prosseguirá com a execução dos programas elaborados. Isto acontece, pois, uma linguagem de programação é apenas um intermediário entre os seres humanos com o computador: a máquina em si só entende o sistema binário, composto por zeros e uns. O papel das linguagens de programação é fornecer comandos similares à forma humana de se comunicar, de modo que eles possam ser traduzidos para a linguagem de máquina que o computador compreende. No caso da linguagem GO, esse processo é feito por um compilador, capaz de converter nossos programas para o código de máquina – ou em uma outra linguagem mais próxima ao que o computador entende, como Assembly – e que será introduzido no decorrer da instalação.

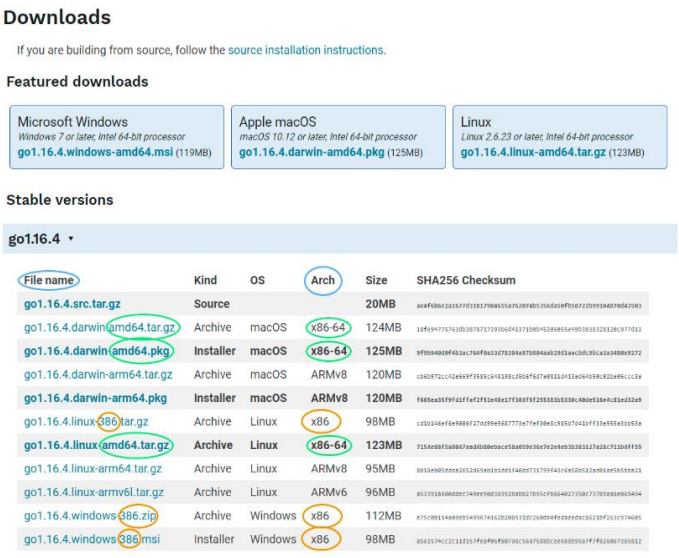
1º PASSO: Devido à variedade de instaladores, deve-se verificar qual o sistema operacional e em qual versão ele está sendo consumido em seu computador. No caso do Windows, basta pressionar as teclas **Windows + Pause/Break** e será aberta uma janela com as propriedades do sistema:



2º PASSO: Após a abertura da tela, procure pela edição e tipo do sistema:



3º PASSO: Com essas informações, acesse o **link** para a página oficial e escolha um instalador que coincida com as características do seu sistema. Se for de 32 bits, role a página para baixo e procure na tabela o arquivo que contenha "386" em seu "file name", ou x86 em "arch". Caso seja de 64 bits, basta escolher o instalador nas caixas azuis da parte superior da página, ou busque na tabela pelo instalador que tenha a nomenclatura "amd64" em seu nome ou "x86-64" em "arch". Em todos os casos, baixe o arquivo que contenha ".msi" ao final de seu nome.



4º PASSO: Feito o download do instalador, execute-o.

5º PASSO: Com a abertura do programa, irá aparecer a seguinte tela. Clique em "Next":

go1.16.4.windows-amd64.msi

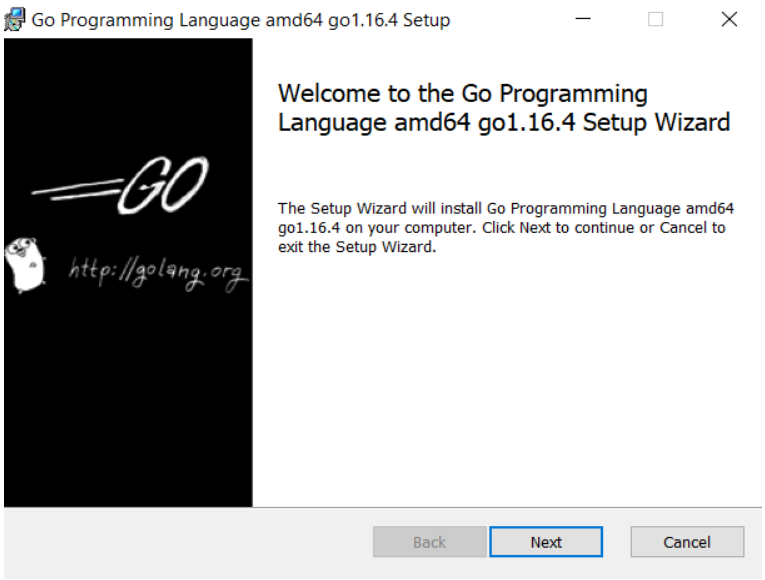
"pacote da linguagem Go na versão 1.16.14 para Windows, versão de 64 bits, com instalador (msi)."

go1.16.4.windows-386.zip

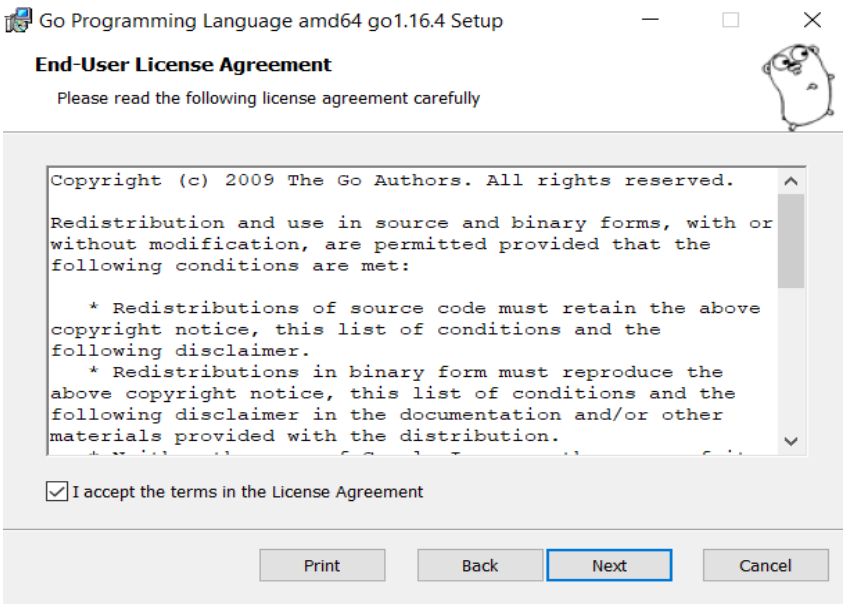
"pacote da linguagem Go na versão 1.16.4 para Windows, versão de 32 bits, compactado em um arquivo zip."

go1.16.4.linux-386.tar.gz

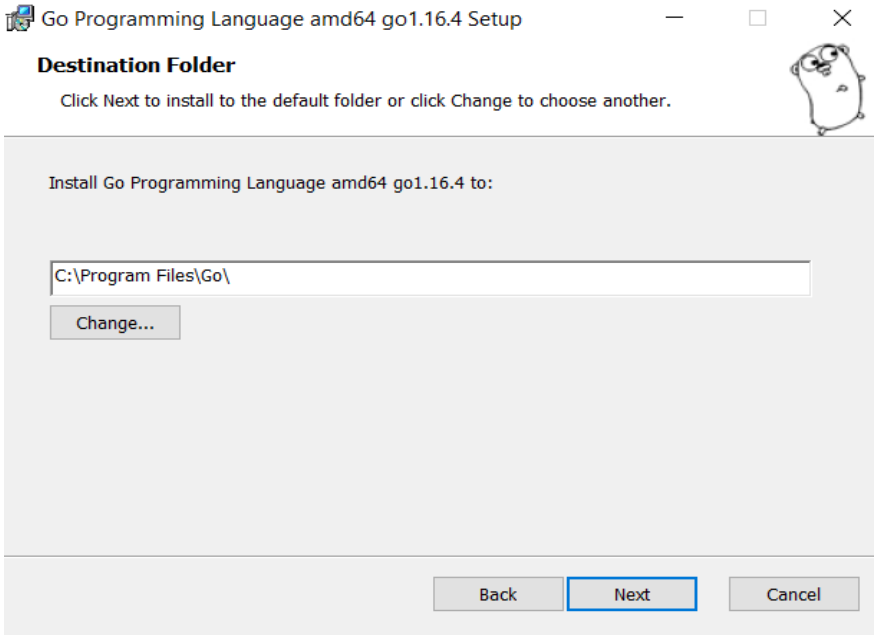
"pacote da linguagem Go na versão 1.16.14 para Windows, versão de 32 bits, compactado em .tar.gz."



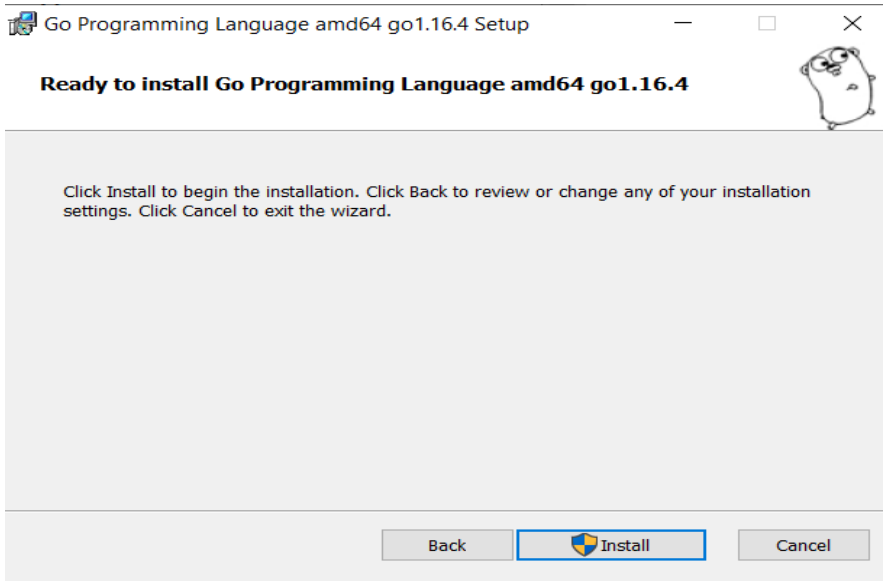
6º PASSO: De praxe, os desenvolvedores da linguagem solicitam a leitura e aceitação dos termos de uso. Marque a caixa "I accept the terms in the License Agreement" para aceitar os termos e clique novamente em "Next":

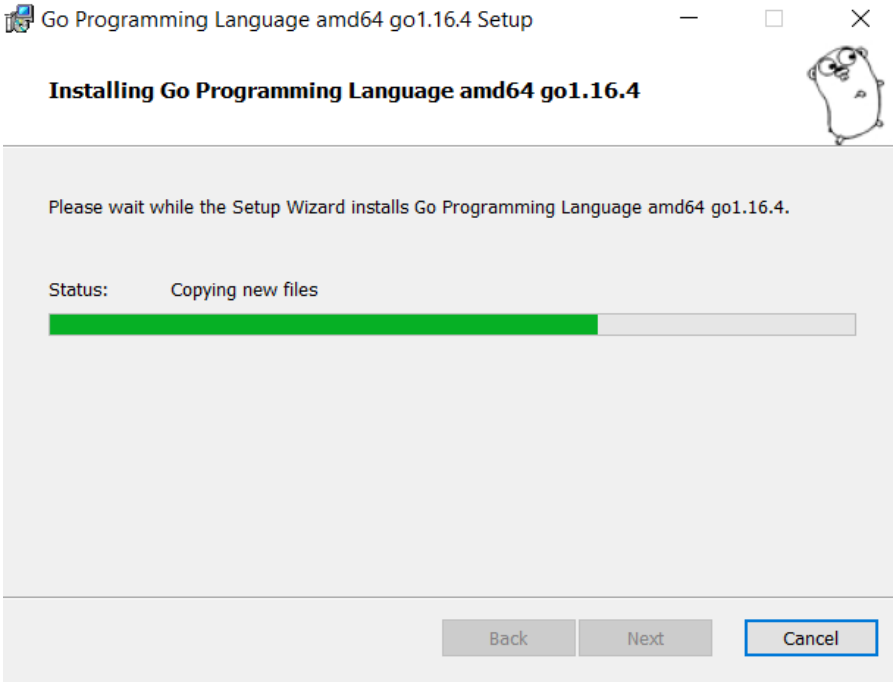


7º PASSO: Nesta próxima tela, é possível modificar o local em que os arquivos serão instalados. Caso não queira realizar nenhuma alteração, basta clicar em “Next” mais uma vez:



8º PASSO: Após selecionar as opções anteriores, basta clicar em “Install” e clicar em “sim" para dar início ao processo de instalação. Espere a instalação terminar.



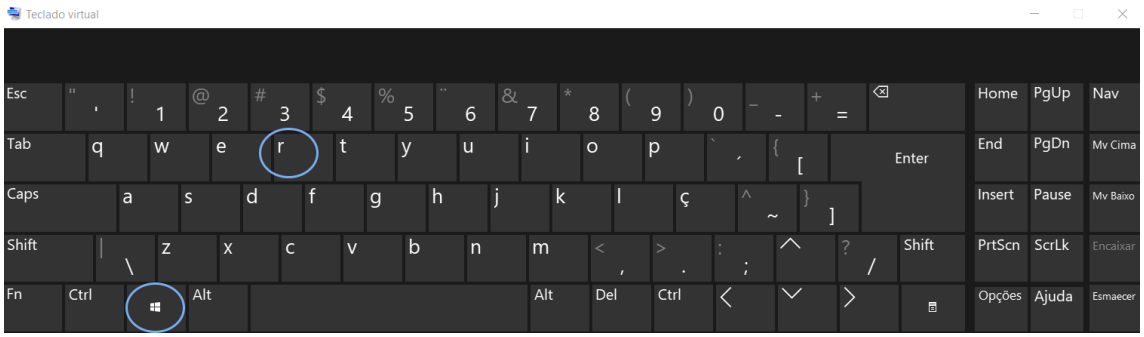


9º PASSO: Caso não esteja logado como administrador, irá aparecer uma janela de confirmação. Basta clicar em “Sim” e prosseguir.

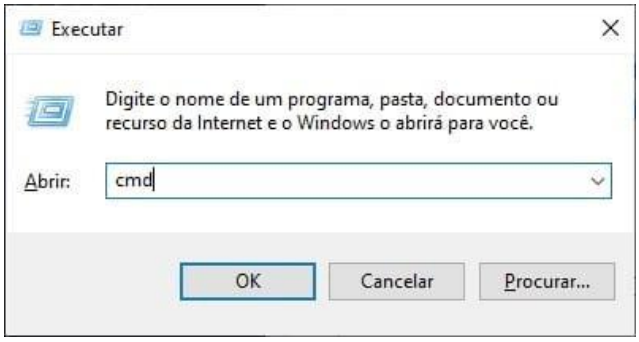
10º PASSO: Aguarde o término da instalação e clique em “Finish” quando terminar.

Pronto! Os recursos da linguagem Go agora estão instalados em sua máquina. Para averiguar o êxito da situação, basta abrir o terminal de comandos do seu sistema operacional. Com o Windows, basta os seguintes passos:

Pressione as teclas Windows + R:



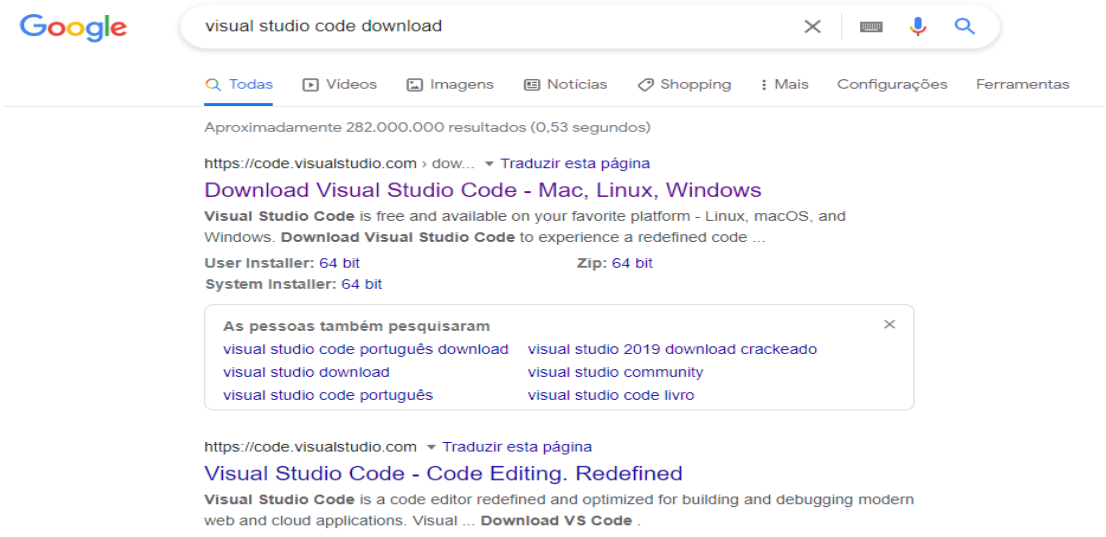
Essa tecla de atalho irá abrir a janela “Executar”. Digite "cmd" e clique em "ok" para abrir um prompt de comando.



Feito isso, digite “go version” na tela que se abrirá e aperte “enter”. Ao aparecer a versão do Go instalada após a execução do comando, significa que as etapas foram feitas corretamente.

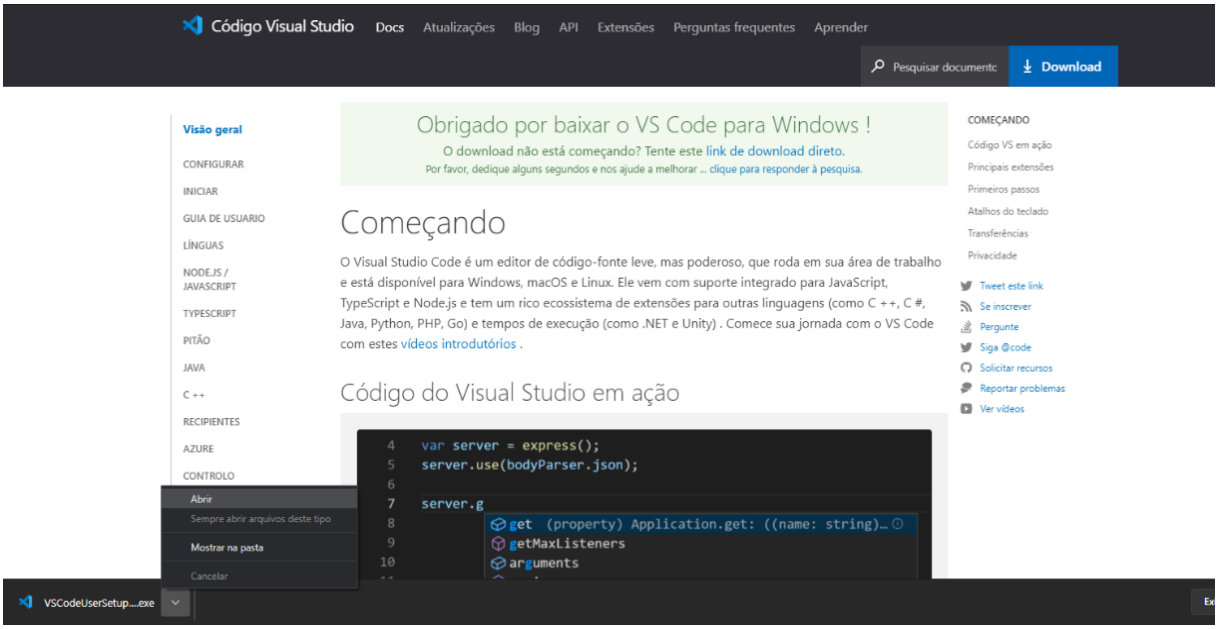
2.2 PROCESSO DE DOWNLOAD DO VISUAL STUDIO CODE

1° PASSO: Deve-se fazer a busca, na plataforma de pesquisa que preferir, na imagem abaixo foi usado o Google. No campo de busca escreve-se “visual studio code download”, o primeiro resultado é o link correto.

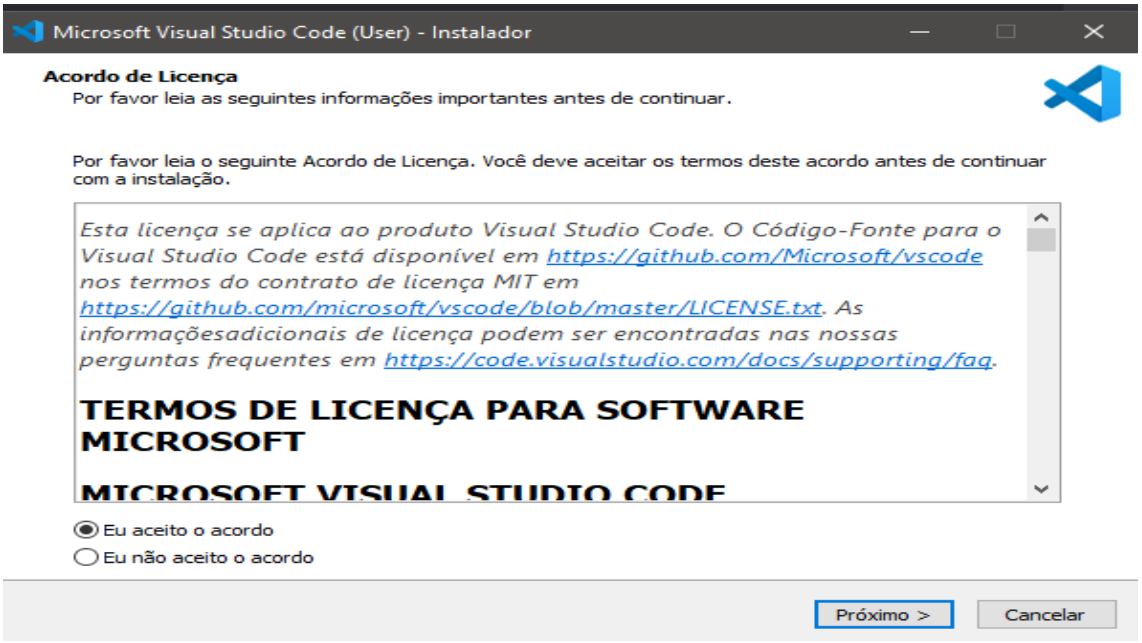


2° PASSO: Ao entrar no site você deverá selecionar qual o sistema operacional do seu computador. O download será iniciado imediatamente.

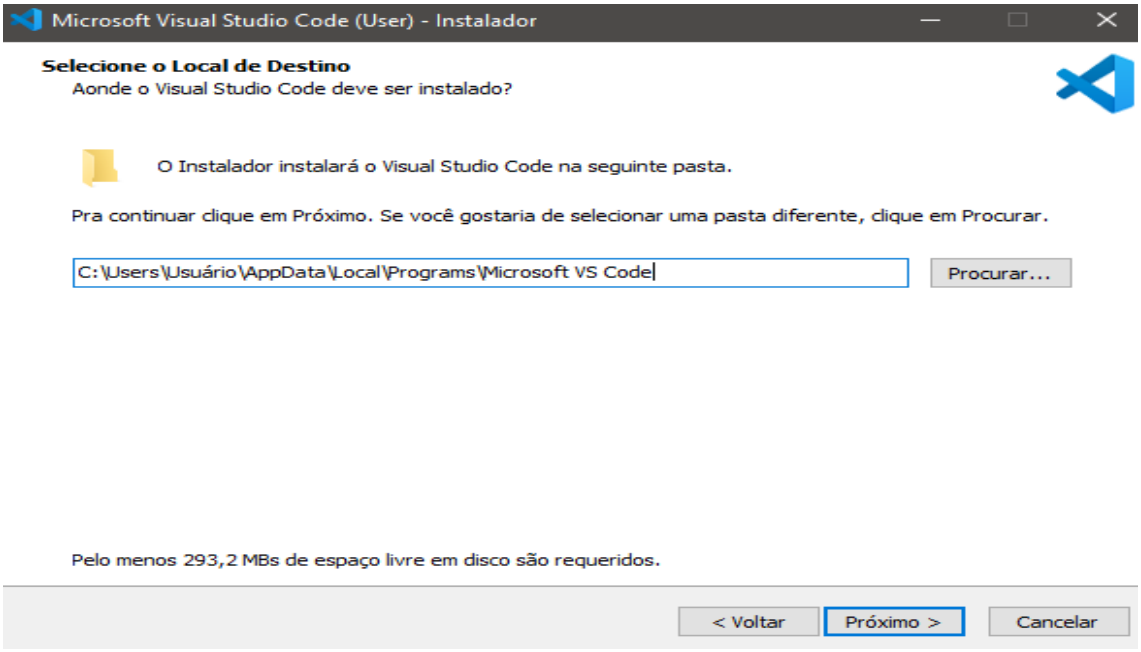
3° PASSO: Após o fim do download você deverá abrir o arquivo (ou ele abrirá sozinho)



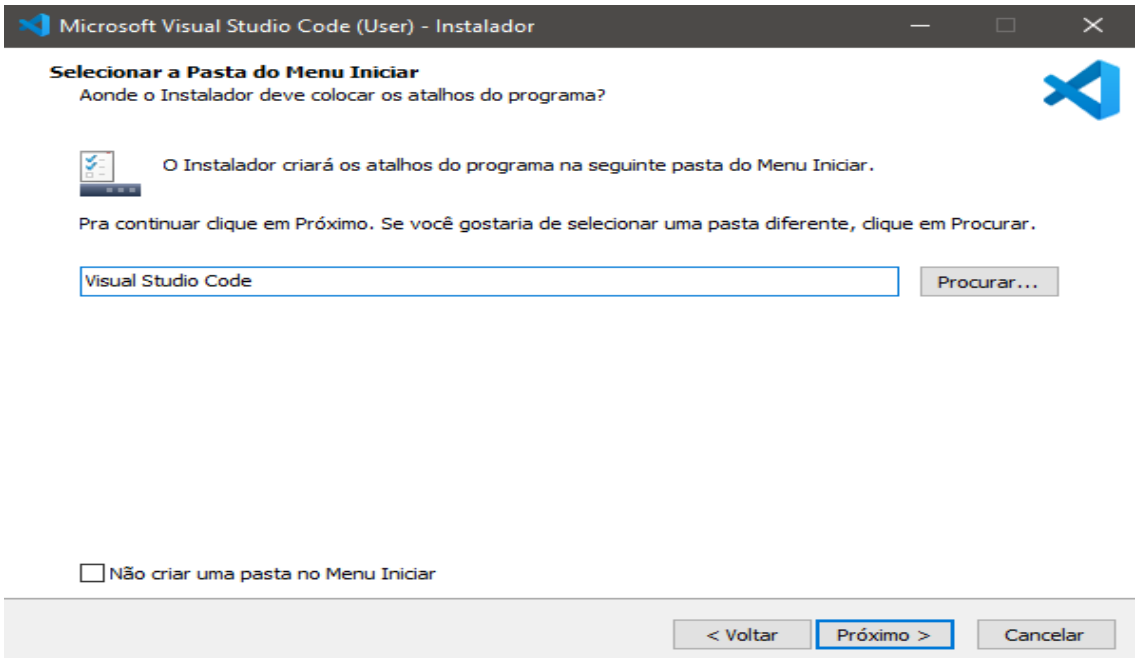
4° PASSO: Uma janela será aberta, nela terá duas opções “Eu aceito o acordo” e “Eu não aceito o acordo”, para iniciar-se a instalação deve-se aceitar o acordo e apertar “Próximo >”.



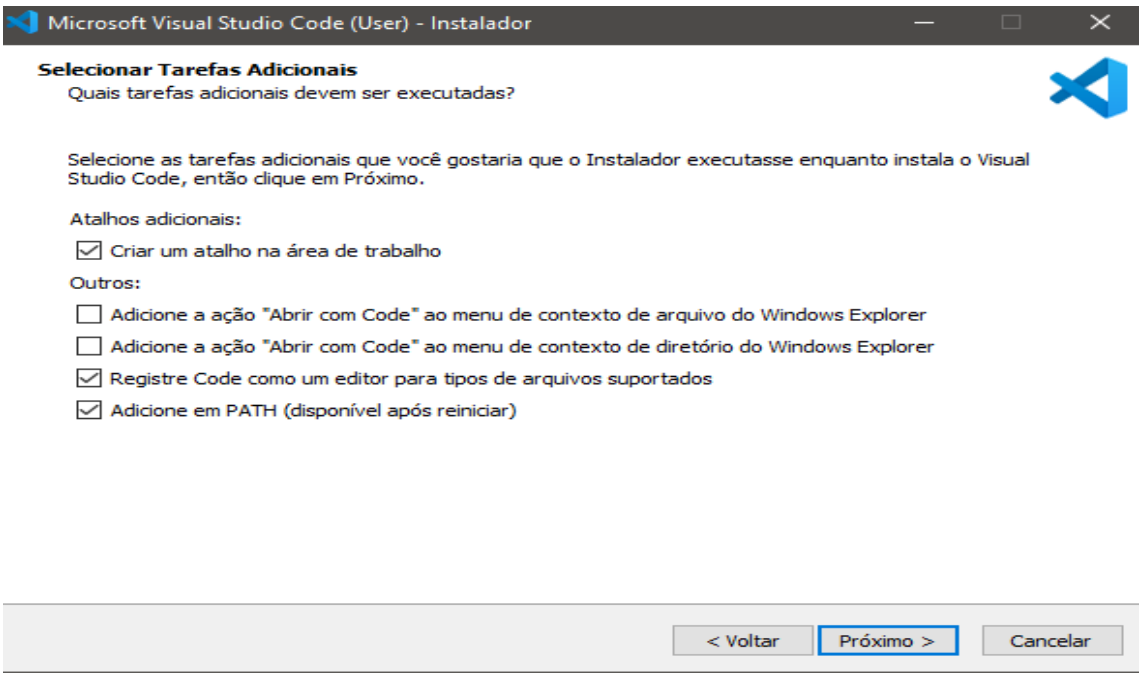
5° PASSO: Neste você precisa escolher o local de instalação da Visual Studio Code, um caminho já estará proposto assim como mostra a foto a seguir.



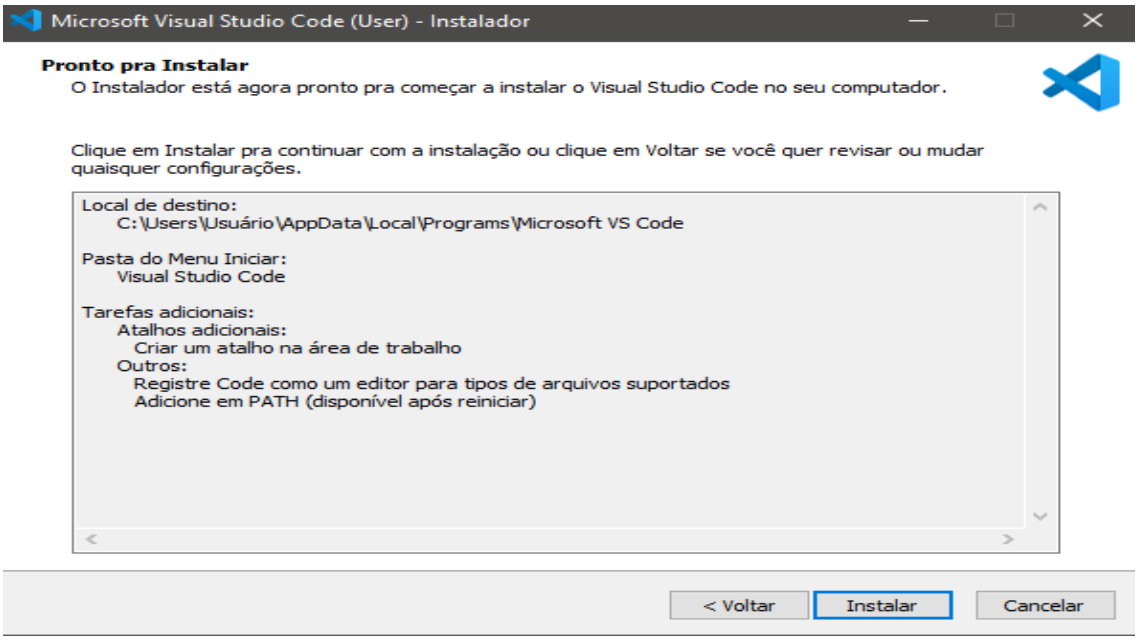
6° PASSO: Nesta etapa o aplicativo quer saber se você deseja que um atalho seja criado e onde ele deve ser criado. Caso você não queira um atalho basta selecionar “Não criar uma pasta no Menu Iniciar” e posteriormente “Próximo >”



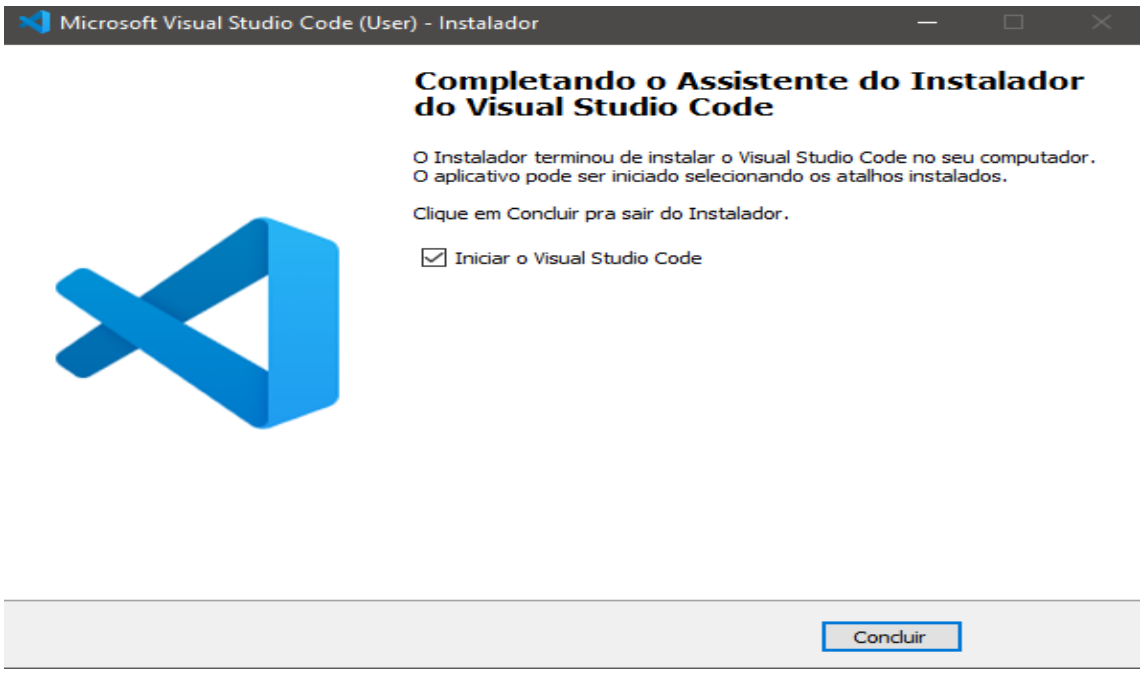
7° PASSO: Assim como criar atalhos, para facilitar o aplicativo deseja saber quais outras atividades você necessita, basta, então, selecioná-las.



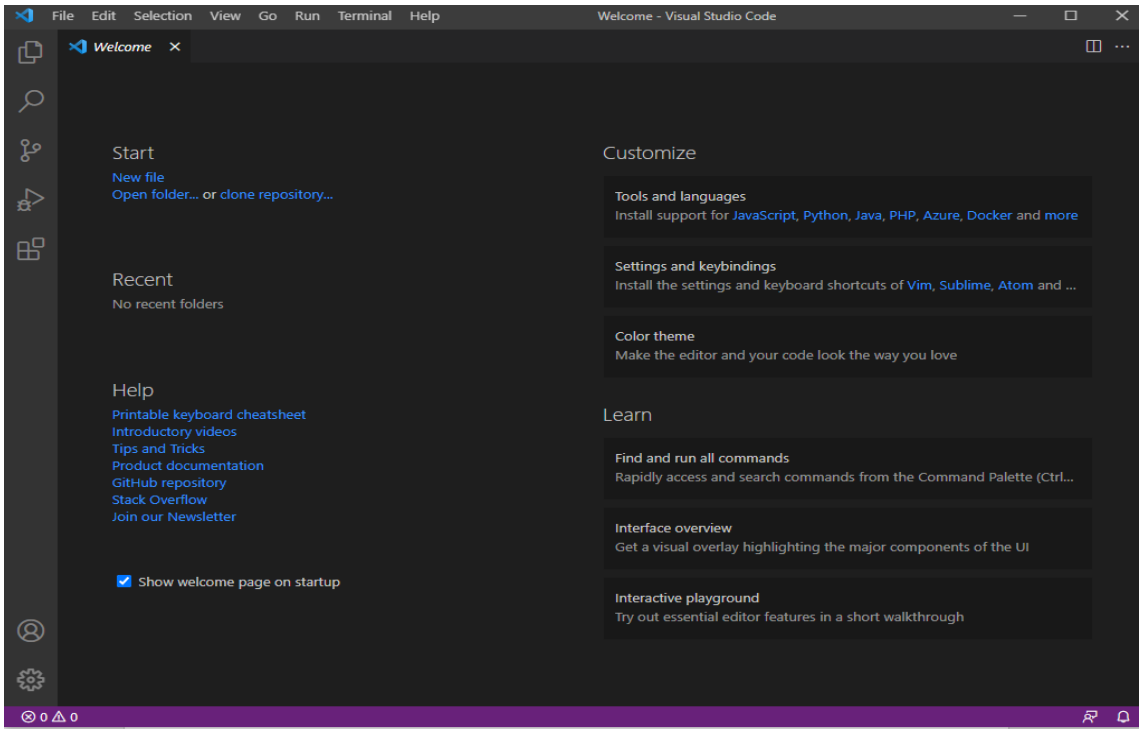
8° PASSO: Selecionada as atividades complementares necessárias o processo termina e agora é preciso apertar em “Instalar” para que a instalação inicie.



9° PASSO: Após a instalação aperta-se em “Concluir”.



10° PASSO: Logo após selecionar a opção “Concluir” o Visual Studio Code será aberto.

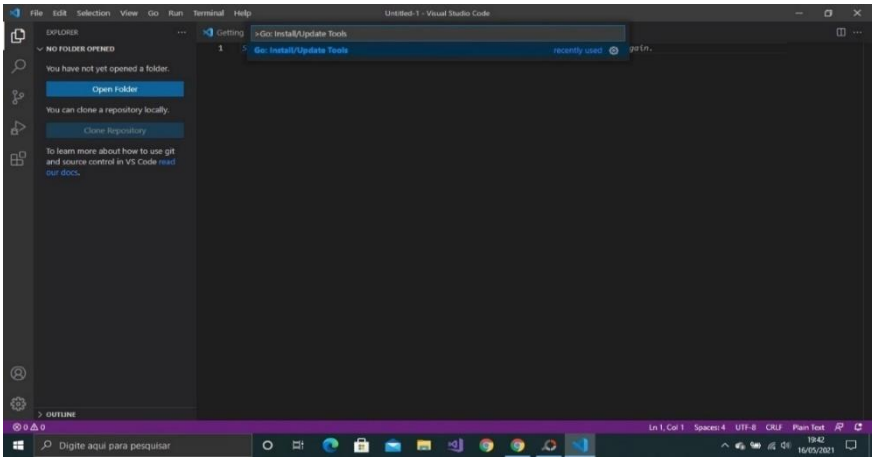


2.3 ABRINDO O VISUAL STUDIO CODE PELA PRIMEIRA VEZ

1º PASSO: Logo após precisamos configurar a linguagem (extensão) que usaremos e que já foi instalada.

Seguindo os seguintes comandos: pressione Ctrl + Shift + P e digite: >Go Install/Update Tools.

Caso o VSCode pergunte qual tool você quer instalar, selecione todas.



2º PASSO: No terminal, a tela preta que foi aberta, você deverá ter uma visão semelhante a esta:

```
Installing 17 tools at ~/Developer/go/bin
  gocode
  gopkgs
  go-outline
  go-symbols
  guru
  gorename
  dlv
  gocode-gomod
  godef
  goreturns
  golint
  gotests
  gomodifytags
  impl
  fillstruct
  goplay
  godoctor

Installing github.com/mdempsky/gocode SUCCEEDED
Installing github.com/uudashr/gopkgs/cmd/gopkgs SUCCEEDED
Installing github.com/ramya-rao-a/go-outline SUCCEEDED
Installing github.com/acroca/go-symbols SUCCEEDED
Installing golang.org/x/tools/cmd/guru SUCCEEDED
```

3º PASSO: Download finalizado, podemos iniciar a o desenvolvimento dos programas. O ambiente está pronto para compilar, testar, e debugar código em Go. Como teste criamos o arquivo helloworld.go, com o seguinte conteúdo:

helloworld.go

```go
package main

import "fmt"

func main() {
    fmt.Println("Hello World")
}
```

4º PASSO: Gerando, após executado, o seguinte output no terminal.

```
Hello World
```

3 EXERCICIOS PARA FIXAÇÃO

Neste capítulo você verá o desenvolvimento de alguns exercícios para que se familiarize com a linguagem, partindo do básico para fixar os principais comandos, evoluindo, então, gradativamente.

Você contará com duas listas, a primeira terá cinco exercícios, já a segunda conterá onze exercícios.

3.1 LISTA 1

Exercício 1

Neste código, a partir da digitação da altura e largura, será calculado a área de um triangulo. Observe:

```
package main
import "fmt"

func main() {
    //DECLARAÇÃO DAS VARIAVEIS
    var altura, base, area float32

    //ENTRADA DE DADOS
    fmt.Print("Valor da base do triângulo: ")
    fmt.Scan(&base)
    fmt.Print("Valor da altura do triângulo: ")
    fmt.Scan(&altura)
    //CALCULO DA ÁREA
    area = (base * altura) / 2
    //SAÍDA
    fmt.Printf("A área do triângulo é: %v", area)
}
```

O comando Scan(&variavel) serve para armazenar o dado digitado pelo usuário. Para exibir o resultado usa-se o comando Printf(“”).

Para executar o código você deve abrir um novo terminal – pode ser aberto através do atalho CTRL + SHIFT + ‘ -, e em seguida digitar “go run nomedoarquivo.go”. Assim como no exemplo abaixo.

```
PS C:\Users\Usuário> go run C:\Users\Usuário\EX01\ex01.go
Valor da base do triângulo: 9
Valor da altura do triângulo: 12
A área do triângulo é: 54
```

Exercício 2

Nesse exercício usamos as funções Math.Cos(x) e Math.Sin(x), que calculam, respectivamente, o cosseno e o seno do ângulo apresentado a ele dentro do parâmetro.

Após as importações necessárias, foi feita a declaração da variável ‘número’ e a entrada de dado pelo usuário, que digita o Ângulo que quer descobrir o seno e o cosseno.

Depois ocorre o cálculo e a impressão do cosseno e do seno do ângulo desejado, usando as funções math.Cos(angulo) e math.Sin(angulo) como parâmetros de impressão do fmt.PrintIn().

```go
package main


import (
"fmt"
"math"
)


func main() {


//declaração de variável
var angulo float64



//entrada de dado
```

```
fmt.Println("Digite o ângulo desejado: ")
fmt.Scan(&angulo)


//cálculo e impressão do cosseno
fmt.Println("Cosseno de", angulo, "é:", math.Cos(angulo))
//cálculo e impressão do seno
fmt.Println("Seno de", angulo, "é:", math.Sin(angulo))
}
```

Observe a execução.

```
Digite o ângulo desejado:
65
Cosseno de 65 é: -0.562453851238172
Seno de 65 é: 0.8268286794901034
```

Exercício 3

Neste exercício pede-se que o usuário entre com 4 notas bimestrais, feito isso, a média é calculada e exibida. Optamos por criar duas funções, a primeira, e que por regra sempre deverá aparecer, sendo a função main.

```
package main

// Importação do pacote fmt
import "fmt"

func main() {

    //Declaração das variáveis
    var n, bim, resultado, total float32
    bim = 1

    //Solitação ao usário para atriabuiçaõ das notas seguindo o critério de 0 A 10
    for bim >= 1 && bim <= 4 {
        fmt.Printf("Qual a nota do %vº Bimestre: \n", bim)
        fmt.Scan(&n)
        total += n
        bim++
    }

    for n < 0 || n > 10 {
        fmt.Printf("Digite a nota do %vº Bimestre (de 0 a 10): \n", bim)
        fmt.Scan(&n)
        total += n
    }
```

Voltando para a função main, o resultado é apresentado para o usuário.

```
    //Chamando e apresentando a função Media
    resultado = Media(total)
    fmt.Print("/n")
    fmt.Print(resultado)
}
```

Retorno dos dados no terminal:

```
PROBLEMS   OUTPUT   TERMINAL   DEBUG CONSOLE
PS C:\Users\Paloma Andrade\Documents\ex3> go run ex3.go
Qual a nota do 1º Bimestre:
7.5
Qual a nota do 2º Bimestre:
9
Qual a nota do 3º Bimestre:
5.5
Qual a nota do 4º Bimestre:
6
Aluno APROVADO com média 7
PS C:\Users\Paloma Andrade\Documents\ex3>
```

Exercício 4

Neste código será solicitado a digitação de dois números e um caractere (+, -, /, ou *) que significam, respectivamente, soma, subtração, divisão e multiplicação, o caractere digitado realizara a operação com os dois números escolhidos pelo usuário. Perceba que foi utilizada a expressão && e != que significa, nessa ordem, E diferente de, então, se o usuário não digitar um desses caracteres o programa repetirá a pergunta, até que seja favorável.

```
package main


import "fmt"


func main() {
    //DECLARAÇÃO DAS VARIAVEIS
    var numero1, numero2, resultado float32
    var operacao string = " "
```

```
    //ENTRADA DE DADOS (NÚMEROS)
    fmt.Print("Digite o primeiro número: ")
    fmt.Scan(&numero1)
    fmt.Print("Digite o segundo número: ")
    fmt.Scan(&numero2)
    //ENTRADA DE DADOS (CARACTERE)
    fmt.Print("Escolha o tipo de operação que será feita: ")
    fmt.Scan(&operacao)

    //VERIFICAÇÃO DO CARACTERE
    if operacao != "+" && operacao != "-
" && operacao != "*" && operacao != "/" {
        fmt.Println("Erro! Caractere inválido.")
        fmt.Println("Escolha o tipo de operação que será feita: ")
        fmt.Scan(&operacao)
    }
    //CONDIÇÃO E CÁLCULO ESCOLHIDO
    if operacao == "+" {
        resultado = numero1 + numero2
    } else if operacao == "-" {
        resultado = numero1 - numero2
    } else if operacao == "*" {
        resultado = numero1 * numero2
    } else {
        resultado = numero1 / numero2
    }
    //SAÍDA
    fmt.Printf("Resultado da operação: %v", resultado)
}
```

Observe que o usuário digitou "." e, por isso, o programa solicitou, novamente, o caractere. Após a correção a operação foi feita com sucesso.

```
Digite o primeiro número: 5
Digite o segundo número: 2
Escolha o tipo de operação que será feita: .
Erro! Caractere inválido.
Escolha o tipo de operação que será feita:
*
Resultado da operação: 10
```

Exercício 5

Este exercício é idêntico ao anterior, a diferença é que no exercício 4 usamos um conjunto de *if's* para fazer a verificação, neste usamos switch case.

Pedimos para o usuário escolher a operação desejada. Implementando também uma validação por *if* para verificar se ele realmente digitou uma operação válida. Com a estrutura de controle *Switch*, iremos denominar o tipo de operação a ser executada segundo o "símbolo" solicitado pelo usuário:

```go
package main

import "fmt"


func main() {
    //DECLARAÇÃO DAS VARIAVEIS
    var numero1, numero2, resultado float32
    var operacao string = " "


    //Entrada de dados
    fmt.Println("Digite o primeiro número: ")
    fmt.Scan(&numero1)
    fmt.Println("Digite o segundo número: ")
    fmt.Scan(&numero2)
```

```
    //Escolha da Operação
    fmt.Println("Escolha o tipo de operação que será feita: ")
    fmt.Scan(&operacao)
    if operacao != "+" && operacao != "-
" && operacao != "*" && operacao != "/" {
        fmt.Println("Erro! Caractere inválido.")
        fmt.Print("Escolha o tipo de operação que será feita: ")
        fmt.Scan(&operacao)
    }
    //Estrutura de controle Switch
    switch operacao {
    case "+":
        resultado = numero1 + numero2
    case "-":
        resultado = numero1 - numero2
    case "*":
        resultado = numero1 * numero2
    default:
        resultado = numero1 / numero2
    }
    //Exibição do resultado
    fmt.Printf("Resultado da operação: %v", resultado)
}
```

O retorno dos dados após a execução é feito pelo terminal, que apresenta os números escolhidos e o resultado da operação.

No terminal abaixo, temos o cálculo da divisão entre 5 e 3:

```
Digite o primeiro número:
5
Digite o segundo número:
3
Escolha o tipo de operação que será feita:
/
Resultado da operação: 1.6666666
```

3.2 LISTA 2

Exercício 1

Nesse exercício usamos o loop *for* para a impressão da tabuada desejada. Primeiramente fazemos as importações e declarações de variáveis necessárias, em seguida, a entrada de dados com o fmt.Scan(), e, então, construímos um loop que calcule e imprima a tabuada do número digitado pelo usuário.

```
package main


import "fmt"


func main() {
    //declaração das variáveis
    var numero, result int


    //entrada de dados
    fmt.Print("Escolha um número positivo e inteiro: ")
    fmt.Scan(&numero)
    //loop para impressão da tabuada
    for ii := 1; ii <= 10; ii++ {
        result = numero * ii
        fmt.Printf("%v X %v = %v\n", numero, ii, result)
    }
}
```

O laço *for* consiste em uma variável com seu valor inicial, seu valor máximo, e uma condição. Nesse caso a variável é ii, que começa com o valor 1 e precisa continuar sendo menor ou igual a 10, sendo que a cada execução do loop é adicionado uma unidade em seu valor.

Dentro do laço é feita uma multiplicação da variável número pela variável ii, o resultado é armazenado na variável result e exibido como parâmetro da impressão. Isso ocorre até que o valor de ii seja menor ou igual a 10, resultando na exibição de toda a tabuada do valor digitado pelo usuário.

```
Escolha um número positivo e inteiro: 7
7 X 1 = 7
7 X 2 = 14
7 X 3 = 21
7 X 4 = 28
7 X 5 = 35
7 X 6 = 42
7 X 7 = 49
7 X 8 = 56
7 X 9 = 63
7 X 10 = 70
```

Exercício 2

No exercício abaixo iremos exibir os "n" valores de uma série, onde a quantidade de termos exibidos serão digitalizados pelo usuário e guardados na variável "qtd". Em seguida, utilizaremos a estrutura de repetição *for* com uma variável auxiliar contadora "x", que será atribuída de valores até que os seus termos sejam feitos em uma quantidade menor ou igual a escolhida pelo usuário.

```go
package main
import "fmt"
func main() {
    //DECLARAÇÃO DAS VARIAVEIS
    var qtd int
    var serie int

    //Valor de "N"
    fmt.Print("Quantos valores serão exibidos: ")
    fmt.Scan(&qtd)
    fmt.Println("Termos da série:")
    //Exibição dos termos da série
    for x := 1; x <= qtd; x++ {
        serie = x*x + 1
        //Exibição dos valores
        fmt.Println(serie)
    }
}
```

A variável auxiliar toda vez que atribuída passará pelo cálculo da variável “serie”, sendo ele o termo geral esclarecido no enunciado, $x^2 + 1$. Após, teremos a linha que imprimimos os valores dos termos da série escolhido pelo usuário.

No terminal abaixo, fizemos a execução do algoritmo e escolhemos que fossem exibidos cinco números da série.

```
PROBLEMAS    SAÍDA    TERMINAL    CONSOLE DE DEPURAÇÃO

Termos da série:  26
PS C:\Users\Usuário> go run C:\Users\Usuário\EX05\ex05.go
Quantos valores serão exibidos: 5
2
5
10
17
26
PS C:\Users\Usuário>
```

Exercício 3

Neste programa temos uma sequência clássica, essa lista os termos da série Fibonacci.

```go
package main


import "fmt"
func main() {
    //DECLARAÇÃO DAS VARIAVEIS
    a, b := 0, 1
    //CONDIÇÃO E CÁLCULO DO FIBONACCI
    for a < 1000 {
        fmt.Print(a, " ")
        b = a + b
        a = b - a
    }
}
```

As variáveis ‘a’ e ‘b’ valem, nesta ordem, 0 e 1. A laço de repetição ‘for’, nesse caso vai ser repetido até que a variável ‘a’ seja menor que 1000.

A contagem começa de ‘a’, ou seja, 0, em seguida foi feita uma conta que seria capaz de alterar o valor de ‘a’, fazendo, agora ele valer 1. Temos agora que ‘a’ é igual a 1, assim como ‘b’, esses valores vão aumentando até que ‘a’ seja menor que 1000.

```
PS C:\Users\Usuário\EX01> go run C:\Users\Usuário\EX03\ex03-1.go
0 1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987
```

Nesse caso a sequência não chegou a 1000, pois, o número seguinte da sequência excederia esse valor.

Exercício 4

Neste exercicio o usuario deve digitar dois valores, onde o segundo deve, obrigadoriamente, ser maior que o primeiro.

```go
package main
import "fmt"
func main() {
    var primeiroNumero, segundoNumero int

    fmt.Println("Digite o primeiro número: ")
    fmt.Scan(&primeiroNumero)
    fmt.Println("Digite o segundo número (maior que o primeiro): ")
    fmt.Scan(&segundoNumero)

  for primeiroNumero > segundoNumero {
   fmt.Println("Erro! O segundo número deverá ser maior que o primeiro!")
        fmt.Println("Digite novamente o segundo número: ")
        fmt.Scan(&segundoNumero)
    }
```

```
    fmt.Printf("O número %v é maior que o número %v.", segundoNumero, pri
meiroNumero)
}
```

Temos a estrutura de laço, para fazer a verificação do segundo número digitado, se o número for novamente digitado da forma errada (menor que o primeiro), o programa irá pedir a digitação novamente. Veja.

```
Digite o primeiro número:
10
Digite o segundo número (maior que o primeiro):
6
Erro! O segundo número deverá ser maior que o primeiro!
Digite novamente o segundo número:
11
O número 11 é maior que o número 10.
```

Exercício 5

O programa a seguir pede que o usuário digite seu sexo, e aceita apenas como resposta os caracteres ‘F’ e ‘M’. Nele foi feito um loop que verifica o dado digitado pelo usuário, e, em seguida, continua pedindo até que o dado seja igual à um dos caracteres desejados.

```
package main
import (
    "fmt"
    "strings"
)
func main() {
    var sexo string

    fmt.Println("Digite o sexo do usuário (F/M): ")
    //loop da entrada de dados
    for ok := true; ok; ok = (sexo != "F" || sexo != "M") {
        fmt.Scan(&sexo)
```

```
        //conversão pra uppercase
        var sexoMais = strings.ToUpper(sexo)
        //verificação da entrada de dados
        if sexoMais != "F" && sexoMais != "M" {
            fmt.Println("Entrada inválida, digite apenas 'F' ou 'M': ")
        } else {
            fmt.Println("Obrigada!!")
            break
        }
    }
}
```

Esse loop foi feito a partir de um laço for, que continua enquanto a variável sexo for diferente de “F” ou diferente de “M”.

Dentro desse laço há o uso da função strings.ToUpper(), a qual converte o parâmetro de modo que todos os seus caracteres fiquem em letra maiúscula, (ou Upper Case). Fazendo então com que seja possível que o usuário digite “f” ou “m” em minúsculo e seja validada corretamente a sua inserção. Obs.: Para utilizar essa função é necessário importar o pacote strings. Veja a execução.

```
Digite o sexo do usuário (F/M):
k
Entrada inválida, digite apenas 'F' ou 'M':
f
Obrigada!!
```

Exercício 6

Nesse exercício vamos calcular o fatorial de um valor determinado pelo usuário, mas, antes, repetindo o pedido até que o mesmo entre com um valor válido, ou seja, inteiro e positivo. Depois de calculado e exibido o resultado, perguntaremos se o usuário deseja repetir o processo e, caso a resposta seja positiva, ocorrerá tudo novamente.

Para isso precisamos, primeiramente, de uma função que faça o cálculo do fatorial. Fazemos da seguinte forma:

```
//Função para cálculo do fatorial
func fat(n int) int {
    if n == 0 {
        return 1
    }
    return n * fat(n-1)
}
      fmt.Println("Digite um número inteiro e positivo: ")
      fmt.Scan(&numeroString)
```

Também precisamos saber como validar o dado digitado pelo usuário, para que o programa aceite apenas números inteiros e positivos. Primeiro fazemos a impressão do pedido e a entrada de dados em uma variável do tipo String:

Em seguida vem a validação:

```
      numeroInteiro, erro = strconv.Atoi(numeroString)

      for erro != nil || numeroInteiro <= 0 {
          fmt.Println("\nErro!") // \n => pula uma linha
          fmt.Println("Digite um número inteiro e positivo: ")
          fmt.Scan(&numeroString)
          numeroInteiro, erro = strconv.Atoi(numeroString)
      }
    fmt.Println("\nFatorial de", numeroInteiro, "= ", fat(numeroInteiro))
```

Na primeira linha, converte-se o numeroString para uma variável do tipo Int. Essa conversão produzirá um erro caso o valor não seja um número inteiro, nos possibilitando, então, de validar o dado. Esse erro produzido será guardado na variável ‘erro’ e usado como condição no laço *for* que vem logo em seguida.

Lê-se o laço *for* da seguinte maneira: “para ‘erro’ diferente de vazio, ou, ‘numeroInteiro’ menor ou igual a zero, faça”. E então é impressa a mensagem de erros e a entrada de dados ocorre novamente. Na última linha vemos a impressão e o cálculo.

O terminal, por ora, fica assim:

```
PROBLEMAS 10   SAÍDA   TERMINAL   CONSOLE DE DEPURAÇÃO
Digite um número inteiro e positivo:
-3

Erro!
Digite um número inteiro e positivo:
2,4

Erro!
Digite um número inteiro e positivo:
4

Fatorial de 4 =  24
```

Agora perguntamos se o usuário deseja continuar, e seguimos uma lógica parecida para fazer a validação da sua resposta, permitindo que o programa aceite apenas ‘S’ para sim e ‘N’ para não.

```
    fmt.Println("\nDeseja continuar? (S/N)")
    fmt.Scan(&resp)
    resp = strings.ToUpper(resp) //converte para caracteres maiúsculos


    for resp != "S" && resp != "N" {
        fmt.Println("\nErro!")
        fmt.Println("Digite apenas 'S' ou 'N': ")
        fmt.Scan(&resp)
        resp = strings.ToUpper(resp)
    }
```

Lê-se o laço *for* da seguinte maneira “para ‘resp’ diferente de S e ‘resp’ diferente de N, faça: ”. E então há a impressão da mensagem de erro e novamente a entrada de dados ocorre.

O terminal dessa parte ficará assim:

```
PROBLEMAS 10   SAÍDA   TERMINAL   CONSOLE DE DEPURAÇÃO

Fatorial de 4 =  24

Deseja continuar? (S/N)
r

Erro!
Digite apenas 'S' ou 'N':
p

Erro!
Digite apenas 'S' ou 'N':
n
PS C:\Users\natha\!! PJS\Capítulo 2>
```

É importante ressaltar que, para o uso das funções strconv.Atoi( ) e strings.ToUpper( ) é necessário importar, respectivamente, os pacotes “strconv” e “strings”.

Abaixo vemos como ficou, então, o corpo completo do código.

```
package main

import (
    "fmt"
    "strconv"
    "strings"
)

//Função para cálculo do fatorial
func fat(n int) int {
    if n == 0 {
        return 1
    }
    return n * fat(n-1)
}

func main() {
    var numeroString string
    var resp string = " "
    var numeroInteiro int
    var erro error

    for resp != "N" {

        //Entrada de dado
        fmt.Println("Digite um número inteiro e positivo: ")
        fmt.Scan(&numeroString)

        //Validação do dado
        numeroInteiro, erro = strconv.Atoi(numeroString)

        for erro != nil || numeroInteiro <= 0 {
            fmt.Println("\nErro!") // \n => pula uma linha
            fmt.Println("Digite um número inteiro e positivo: ")
            fmt.Scan(&numeroString)
            numeroInteiro, erro = strconv.Atoi(numeroString)
        }

        //impressão do fatorial
        fmt.Println("\nFatorial de", numeroInteiro, "= ", fat(numeroInteiro))
```

```
        //Entrada de dado
        fmt.Println("\nDeseja continuar? (S/N)")
         fmt.Scan(&resp)
         resp = strings.ToUpper(resp) //converte em caracteres maiusculos

         //Validação do dado
         for resp != "S" && resp != "N" {
             fmt.Println("\nErro!")
             fmt.Println("Digite apenas 'S' ou 'N': ")
             fmt.Scan(&resp)
             resp = strings.ToUpper(resp)
         }
    }
}
```

Exercício 7

Este exercício pede para o usuário digitar 10 números, após a digitação o programa exibe esses números na ordem inversa à digitação. Para isso, usamos um laço, onde armazenamos os valores digitados. Por fim, usamos outro laço, desta vez para que fosse possível inverter a ordem.

```
package main

import "fmt"


func main() {

    var cresc [10]int


    for ii := 0; ii < len(cresc); ii++ {

        fmt.Print("Digite o ", ii+1, "° número: ")

        fmt.Scan(&cresc[ii])

    }

    fmt.Println("Ordem inversa:")

    for ii := 9; ii >= 0;  ii-- {

        fmt.Println(cresc[ii])

    }
}
```

Observe a resolução.

```
Digite o 1º número: 5
Digite o 2º número: 4
Digite o 3º número: 3
Digite o 4º número: 2
Digite o 5º número: 1
Digite o 6º número: 0
Digite o 7º número: 1
Digite o 8º número: 2
Digite o 9º número: 3
Digite o 10º número: 4
Ordem inversa:
4
3
2
1
0
1
2
3
4
5
```

Exercício 8

Neste programa o usuário deve entrar com 10 valores. Após a digitação é feita a soma e a média de todos os números digitados.

```go
import "fmt"

func main() {
    //DECLARAÇÃO DAS VARIAVEIS
    var numeros [11]int
    var cont, soma, media int

    //LAÇO PARA ENTRADA DOS 10 NÚMEROS
    for cont = 1; cont <= 10; cont++ {
        fmt.Printf("Digite o %vº número: ", cont)
        fmt.Scan(&numeros[cont])
    }
    //LAÇO PARA A SOMA E MÉDIA DOS 10 NÚMEROS
    for cont := 0; cont <= 10; cont++ {
        soma = soma + numeros[cont]
        //CALCULANDO A MÉDIA
        media = soma / 10
    }
```

```
    //EXIBE A SOMA E A MÉDIA
    fmt.Printf("A soma foi: %v ", soma)
    fmt.Printf("e a média foi: %v", media)
}
```

Para a soma usou-se o laço for, onde cada número digitado anteriormente foi somado um a um. Já a média, simplesmente, foi feita a divisão pela quantidade de números digitados, no caso, 10. Veja como ficou após a compilação.

```
PS C:\Users\Usuário\EX08> go run ex08.go
Digite o 1º número: 5
Digite o 2º número: 5
Digite o 3º número: 5
Digite o 4º número: 5
Digite o 5º número: 5
Digite o 6º número: 5
Digite o 7º número: 5
Digite o 8º número: 1
Digite o 9º número: 0
Digite o 10º número: 5
A soma foi: 41 e a média foi: 4
```

Exercício 9

No exercício abaixo iremos entrar com dez números e armazená-los em um vetor, a qual nomeamos de “cresc”. Também precisaremos declarar inicialmente duas variáveis auxiliares, nomeadas de “i” e “aux”.

Após o pedido de digitalização, passaremos os dados por uma classificação direta, onde iremos ordená-los na ordem decrescente, passando o maior valor para o primeiro índice do vetor, depois o segundo maior para a segunda e assim sucessivamente até o último índice. Por último, imprimiremos os números em ordem decrescente.

```
package main

import "fmt"

func main() {
    //DECLARAÇÃO DAS VARIAVEIS
    var cresc [10]int
    var i, aux int
```

```
    // Entrada de dados
    for i := 0; i < len(cresc); i++ {
        fmt.Print(i+1, "º número: ")
        fmt.Scan(&cresc[i])
    }
    // Classificação por seleção direta
    for i := 0; i < len(cresc)-1; i++ {
        for j := i + 1; j < len(cresc); j++ {
            if cresc[i] < cresc[j] {
                aux = cresc[i]
                cresc[i] = cresc[j]
                cresc[j] = aux
            }
        }
    }
    fmt.Println("Exibição em ordem decrescente:")
    // Exibir os dados em ordem decrescente
    for i = 0; i < len(cresc); i++ {
        fmt.Println(cresc[i])
    }
}
```

Como exemplo temos esse caso com números aleatórios colocados quando executamos.

```
1º número: 5
2º número: 3
3º número: 4
4º número: 9
5º número: 6
6º número: 0
7º número: 1
8º número: 7
9º número: 2
10º número: 8
Exibição em ordem descrescente:
9
8
7
6
5
4
3
2
1
0
```

Exercício 10

Este programa consiste na digitação de 10 nomes de pessoas e suas respectivas idades. Após a digitação o programa pede para você pesquisar um nome qualquer, estando ou não na lista eventualmente criada. Caso o nome informado não esteja na lista, aparece uma mensagem, mas se o nome for encontrado, informará e mostrará idade desta pessoa. Após a primeira busca, encontrando ou não, o nome digitado, o programa pergunta se você deseja fazer uma nova consulta.

Primeiramente, devemos pedir ao usuário os dez valores e armazená-los em seus respectivos vetores, da seguinte forma:

```
package main
import (
    "fmt"
    "strings"
)
func main() {
    /*Declaração dos vetores e da constante
    que define a quantidade de seus índices.*/
    const quantidadeVetor int = 10
    var nomes [quantidadeVetor]string
    var idades [quantidadeVetor]int

    for i := 0; i < quantidadeVetor; i++ {
        fmt.Printf("Digite o %vº nome: ", i+1)
        fmt.Scan(&nomes[i])
        fmt.Printf("Digite a idade de %v: ", nomes[i])
        fmt.Scan(&idades[i])
        fmt.Println()
    }
```

Com os dados em memória, devemos iniciar o processo de pesquisa ao solicitar um nome e verificar se ele já foi digitado anteriormente:

```
var pesquisas []string
for consulta := " "; consulta != "N"; {
    var pesquisa string = " "
    consulta = " "

    fmt.Println("\n--CONSULTAS--")
    for pesquisa == " " {
        fmt.Println("Nome a ser consultado: ")
        fmt.Scan(&pesquisa)
       for _, nomeItem := range pesquisas {
            if pesquisa == nomeItem {
                fmt.Printf("\nNome já pesquisado anteriormente!\n")
                pesquisa = " "
                break
            }
        }
    }
```

A variável “pesquisa” é responsável por armazenar o conteúdo da consulta, e começa como uma string armazenando um espaço vazio. Enquanto ela ainda estiver vazia, continuaremos executando as mesmas linhas do programa que pedem novamente a digitação de um valor.

Para verificar se a pesquisa por um nome já foi realizada anteriormente, criou-se o vetor “pesquisas”, responsável por armazenar tais consultas. A cada vez que o usuário realizar uma pesquisa, a rotina do “for” será executada nesse vetor para verificar se há alguma correspondência. Caso o valor da pesquisa atual coincida com algum valor de “pesquisas”, a variável “pesquisa” receberá “ “ como valor para continuar no laço de repetição e o usuário terá de digitar um novo nome. Caso contrário, o comando “append” é utilizado para acrescentar ao vetor o valor da pesquisa atual.

No trecho de código acima, a palavra “break” foi utilizada para sair do bloco de código do “*if* pesquisa == nomeItem” no momento em que achar tal correspondência, evitando que o programa tenha que percorrer todas as posições do vetor ainda que já tenhamos atingido o

objetivo de nossa instrução condicional. Ademais, observe o uso da palavra-chave "range": ela serve para pesquisarmos por todos os valores contidos em um determinado vetor.

Agora que recebemos o valor da consulta e evitamos sua repetição, devemos prosseguir com a pesquisa dos nomes e de suas respectivas idades. Neste caso, nossa solução será muito parecida com a busca feita dentro do vetor "pesquisas":

```
      pesquisas = append(pesquisas, pesquisa)
      var ocorrencias int = 0
      for i, nomeItem := range nomes {
          if pesquisa == nomeItem {
            fmt.Printf("A pessoa \"%v\" foi encontrada e possui %v anos.\
n",
                  nomeItem, idades[i])
              ocorrencias++
          }
      }
```

A variável "ocorrencias" serve para nos dizer quantos resultados foram obtidos a partir dessa pesquisa. Dessa forma, poderemos exibir uma mensagem ao usuário caso nenhum resultado seja retornado da consulta:

```
        if ocorrencias == 0 {
            fmt.Println("\nPessoa não localizada.")
        } else {
            fmt.Printf("\nForam encontradas %v ocorrências do nome inform
ado.\n", ocorrencias)
        }
```

Com isto, resta perguntar ao usuário se ele deseja realizar uma nova consulta – verificando o valor desta entrada, que deve ser "S" ou "N":

```
for consulta != "S" && consulta != "N" {
            fmt.Println("\nDeseja realizar uma nova consulta? S/N")
            fmt.Scan(&consulta)
            consulta = strings.ToUpper(consulta)    }
```

Observe uma possível consulta.

```
Digite o 1º nome: Guilherme
Digite a idade de Guilherme: 18

Digite o 2º nome: Ana
Digite a idade de Ana: 19

Digite o 3º nome: Felipe
Digite a idade de Felipe: 21


--CONSULTAS--
Nome a ser consultado:
George

Pessoa não localizada.

Deseja realizar uma nova consulta? S/N
s
--CONSULTAS--
Nome a ser consultado:
Felipe
A pessoa "Felipe" foi encontrada e possui 21 anos.

Foram encontradas 1 ocorrências do nome informado.

Deseja realizar uma nova consulta? S/N
n
```

Exercício 11

No último exercício dessa lista o programa representa um determinado cinema, cujo possui 20 fileiras com 15 cadeiras. O programa deve solicitar, então, o nome do cliente e o lugar – fileira e cadeira – que tal deseja sentar-se, caso o assento já esteja reservado deve-se pedir para o cliente escolher outro lugar, a pergunta deve ser feita até que o lugar escolhido esteja livre.

Começamos a entrada dos dados com um for. O usuário entra com nome, a fileira e cadeira:

```go
package main


import "fmt"


func main() {
    //Criação de variáveis
    var nome string
    var cad, fil, cad1, fil1, esc int
    var lug [15][20]string
```

```
    for veri := 2; veri > 1; veri++ {
        //Entrada de dados
        fmt.Println("Informe seu nome:")
        fmt.Scanln(&nome)
        fmt.Println("Informe o número da sua fileira de 1 até 20:")
        fmt.Scanln(&fil)
        fmt.Println("Informe o número da sua cadeira de 1 até 15:")
        fmt.Scanln(&cad)

        cad1 = cad - 1
        fil1 = fil - 1
```

Com o *If* e *Else* verificamos a condição escolhida da cadeira e fileira, se está vazia ou não. Para uma nova reserva, optamos para que a condição seja retornada como número – 1 para sim, 0 para não.

```
        //Verificação
        if lug[cad1][fil1] == "" {
            fmt.Println("Seu assento foi reservado com sucesso!")
            lug[cad1][fil1] = nome
            fmt.Println("Você deseja fazer uma nova reserva? Para 'Sim' d
igite 1, para 'Não' digite 0.")
            fmt.Scanln(&esc)
            if esc == 0 {
                veri = 0
            }
        } else {
            fmt.Println("Este assento já foi reservado!")
            fmt.Println("Você quer tentar escolher um novo lugar? Para 'S
im' digite 1, para 'Não' digite 0.")
            fmt.Scanln(&esc)
            if esc == 0 {
                veri = 0
            }
```

```
        }

    }

    fmt.Println("Assentos: ", lug)

}
```

Se o cliente não quiser reservar mais nenhum assento, o programa exibe os assentos já preenchidos.

```
Informe seu nome:
GIOVANNA
Informe o número da sua fileira de 1 até 20:
20
Informe o número da sua cadeira de 1 até 15:
5
Seu assento foi reservado com sucesso!
Você deseja fazer uma nova reserva? Para 'Sim' digite 1, para 'Não' digite 0.
1
Informe seu nome:
NICOLY
Informe o número da sua fileira de 1 até 20:
20
Informe o número da sua cadeira de 1 até 15:
4
Seu assento foi reservado com sucesso!
Você deseja fazer uma nova reserva? Para 'Sim' digite 1, para 'Não' digite 0.
0
Assentos:  [[                ] [                ] [                ] [                NICOLY] [
] [                NICOLY] [                GIOVANNA] [                ] [                ] [
    ] [                ] [                ] [                ] [   [                ] [
                ] [                ] [                ] [
      ] [                ] [                ]]
```

4 FYNE FRAMEWORK

Fyne é um framework muito fácil de aprender, gratuita e de código aberto para a construção de gráficos para desktop, combinando a usabilidade e a simplicidade da linguagem de programação Go com uma biblioteca de widgets. Construído com os melhores procedimentos de design, teste e validação, com um alto nível de padrão de qualidade e design, atendendo às expectativas do usuário.

Recentemente criada, Fyne teve seu início em fevereiro de 2018, com um grande apoio dos primeiros usuários. De início, desenvolveram um site para ajudar os visitantes a entender do que se tratava o projeto e para acompanhar os progressos. Foi escolhida a linguagem Go como principal para desenvolvimento e API, pois, possui um desenvolvimento simples para qualquer nível de experiência.

A estrutura Fyne é projetada para usabilidade no núcleo e os widgets e layouts Fyne se adaptam perfeitamente ao contexto do usuário, permitindo que os desenvolvedores se concentrem na funcionalidade e também no teste da interface do usuário. Com uma API bem elaborada, a aparência limpa do Material Design e a documentação clara, o kit de ferramentas Fyne oferece suporte a uma nova geração de desenvolvimento de aplicativos de plataforma cruzada.

4.1 INSTALAÇÃO DO FRAMEWORK

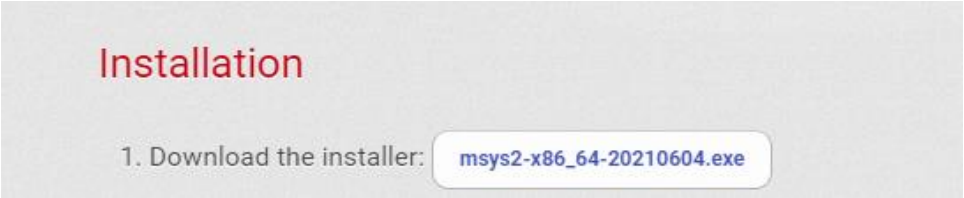
REQUISITOS TÉCNICOS:

Para que uma aplicação Fyne possa ser desenvolvida em sua máquina, você irá precisar de duas coisas:

- A linguagem GO instalada em sua máquina;
- Um compilador da linguagem C.

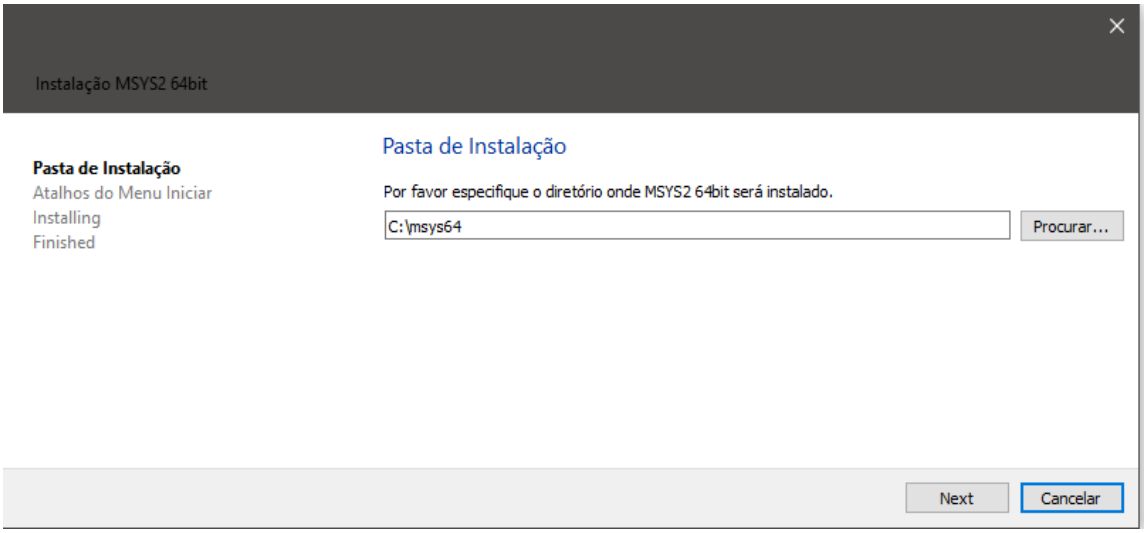
Na apostila, utilizaremos como compilador o Msys2 por questão de preferência, mas você pode optar por qualquer outro.

1. Para darmos início ao download devemos acessar o seguinte site: https://www.msys2.org/. Ao entrar aperte em “msys2-x89_64-20210604.exe”. Aguarde o fim do download;

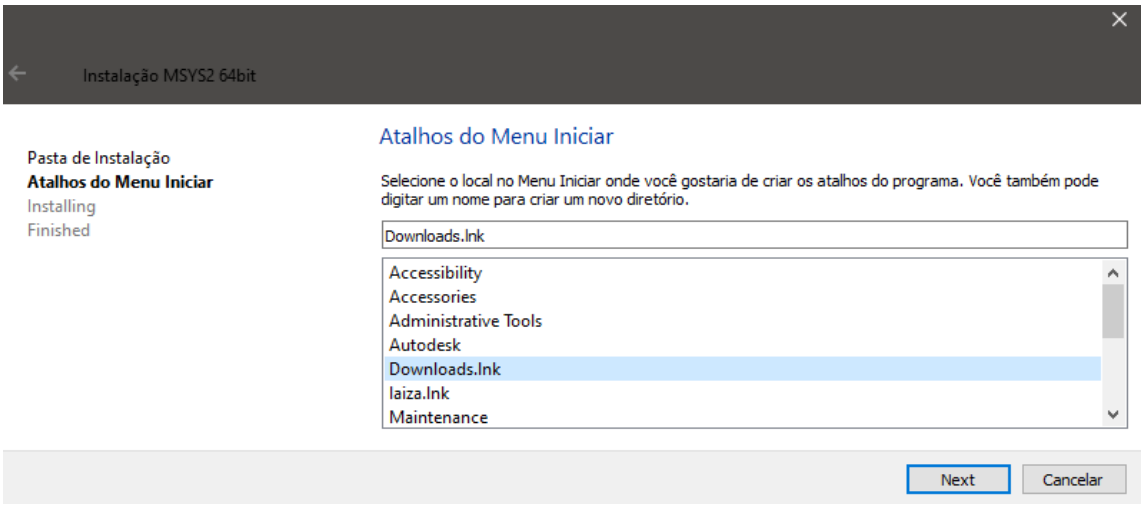


2. Execute. Uma janela se abrirá, para dar continuidade à instalação aperte em “Next”;

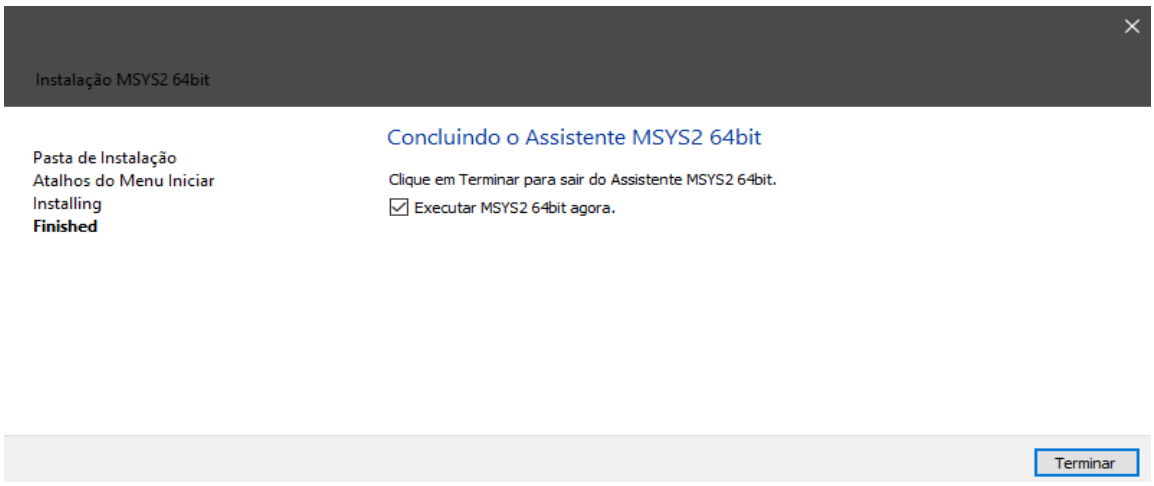
3. Mais uma vez aperte em “Next”



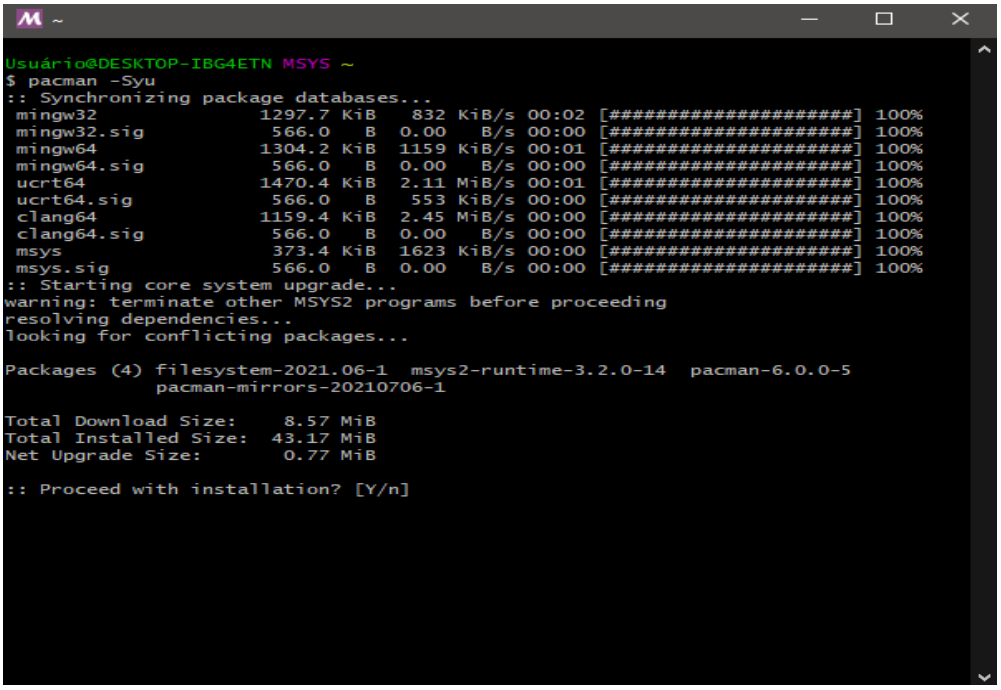
4. Nesta janela você deve escolher um local para a criação de um atalho para o programa que estamos instalando. Após fazer sua escolha, aperte em “Next”. **Recomenda-se que não altere a opção padrão do programa**;



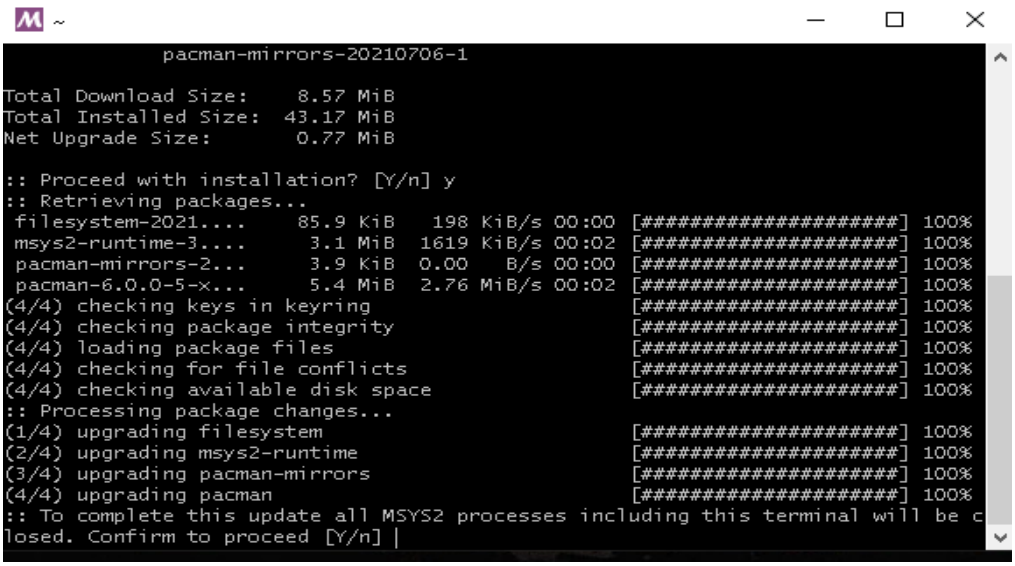
5. Aguarde a instalação. Aperte em “Terminar”.



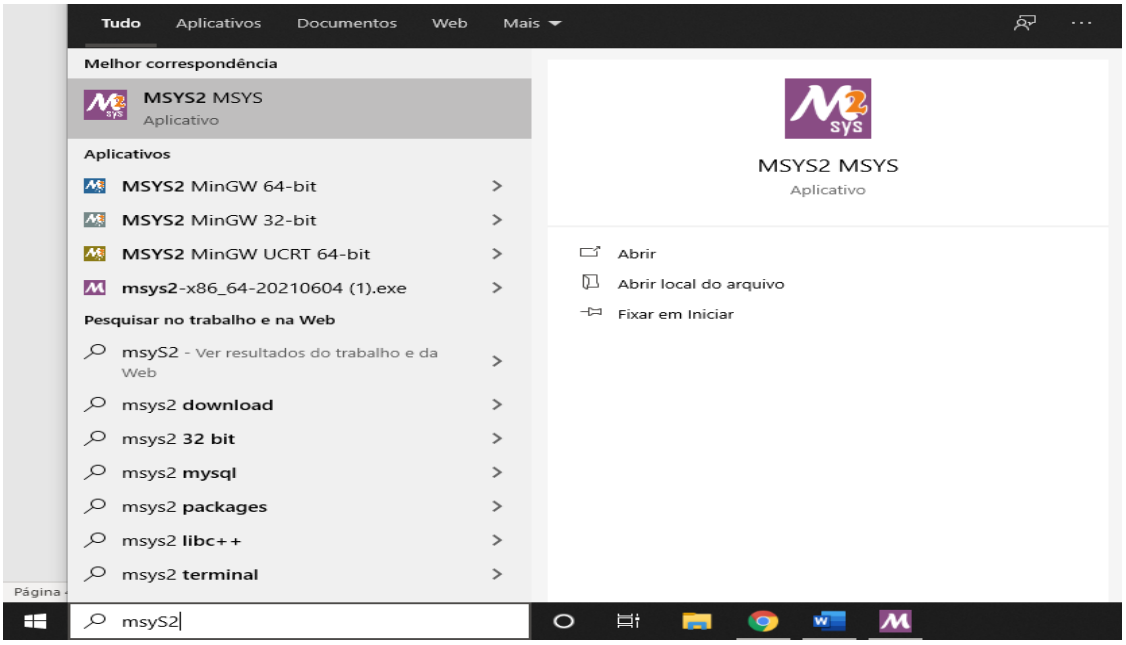
6. Uma CMD abrirá, nela você deve digitar “pacman –Syu”. Aguarde. Uma pergunta irá aparecer “Continuar com a instalação?”, pressione ‘Y’ e posteriormente Enter;



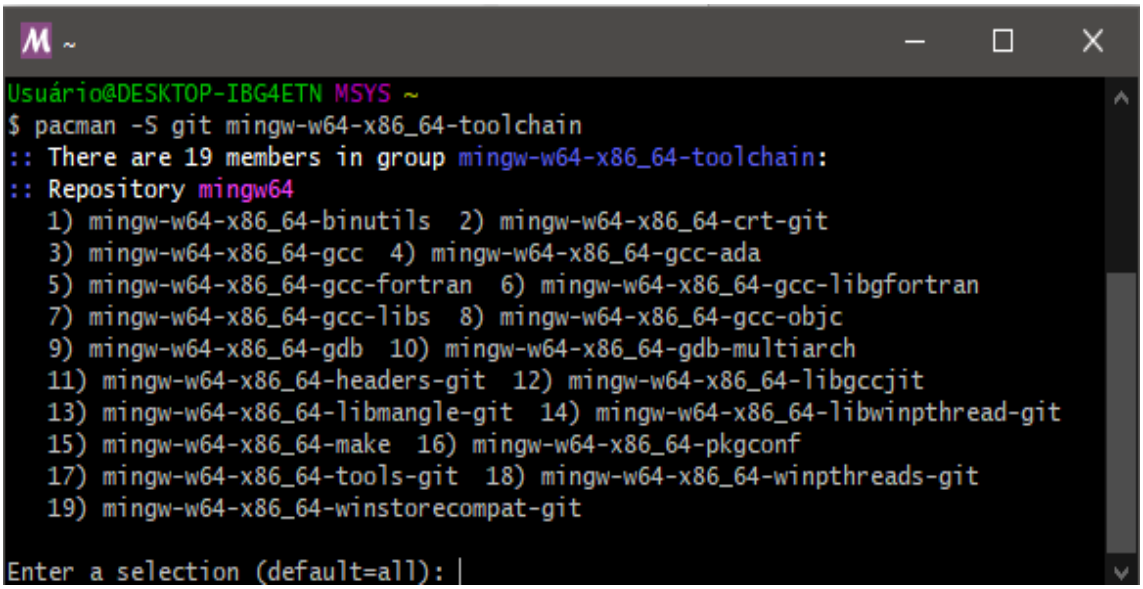
7. Quando a instalação terminar, aperte ‘Y’ e Enter novamente. O CMD irá fechar;



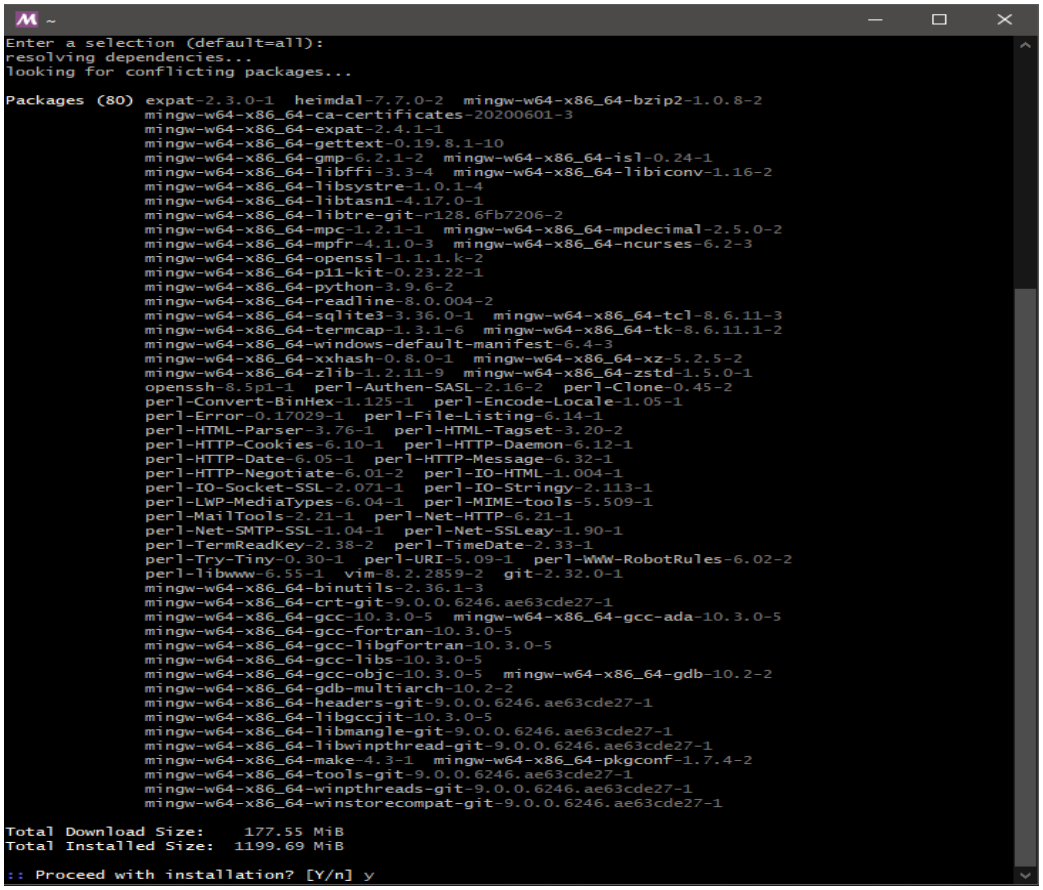
8. Na barra de pesquisa do Windows escreva “MSYS2” para, novamente, abrir o CMD;



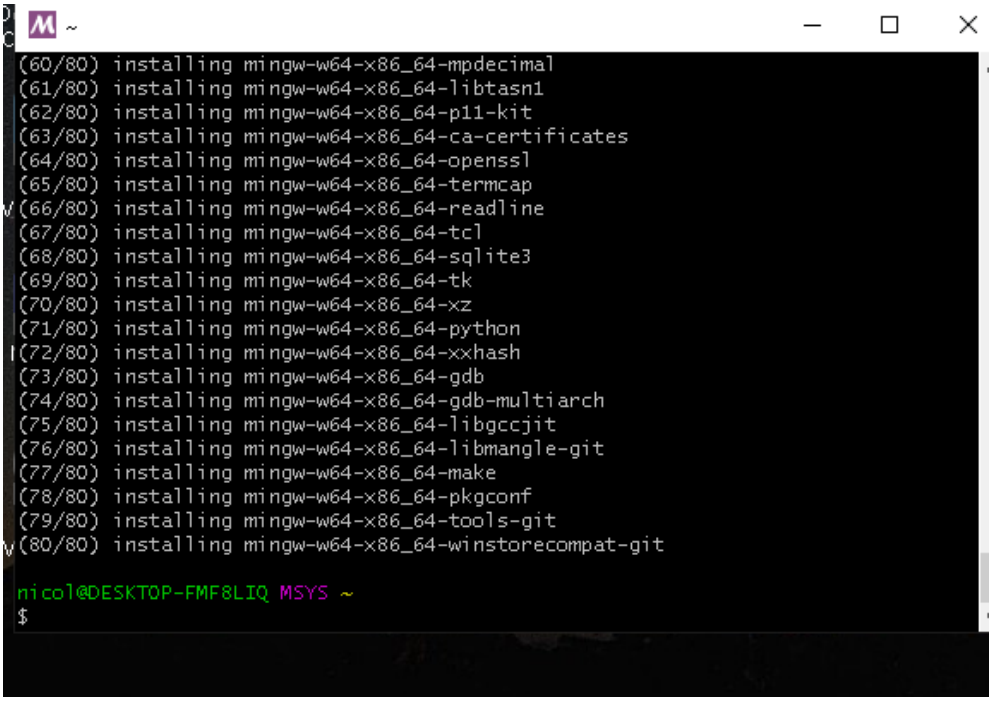
9. Agora digite “pacman -S git mingw-w64-x86_64-toolchain. Aperte em qualquer Tecla, para continuar;”



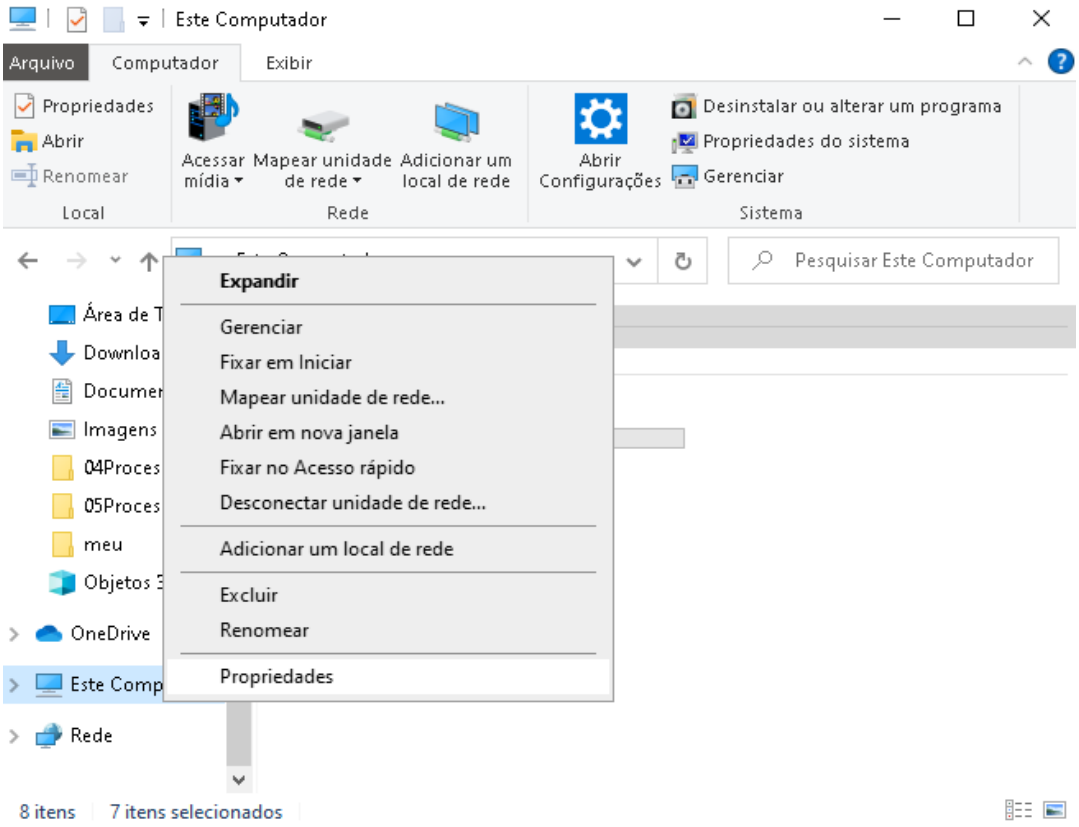
10. Pressione ‘Y’ para continuar a instalação;



11. Após encerrar, aparecerá essa pág., feche apertando no ‘X’;

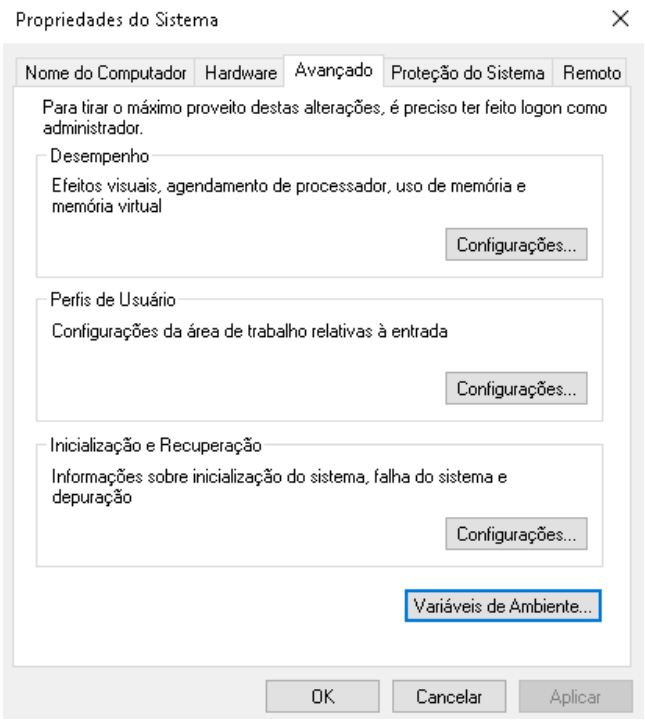


12. Depois, abra o explorador de arquivo, vá até “Este computador”, clique com o botão direito do mouse e clique em propriedades;

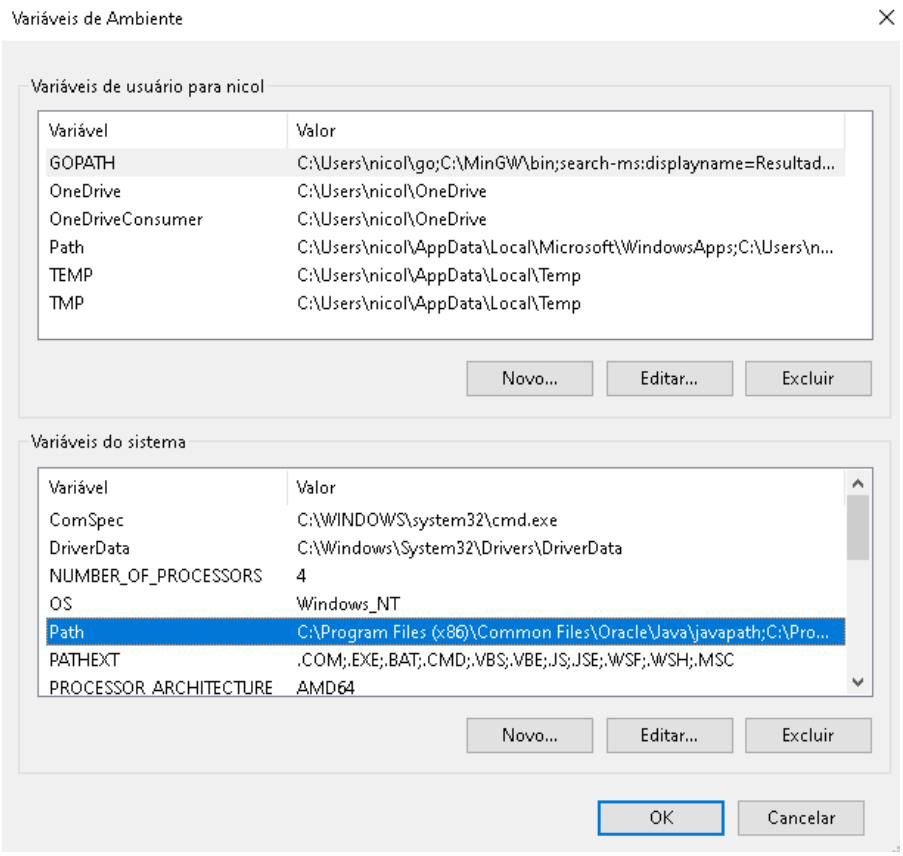


13. Procure e clique em “Configurações avançadas do sistema”;

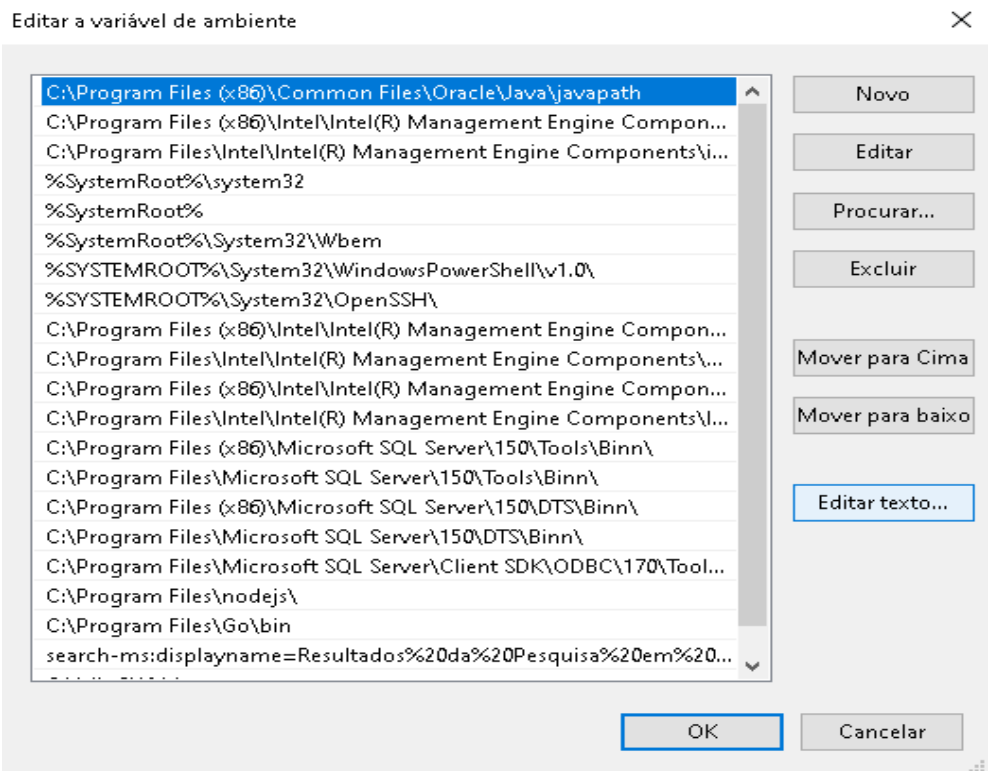
14. Clique em “Variáveis de Ambiente”;



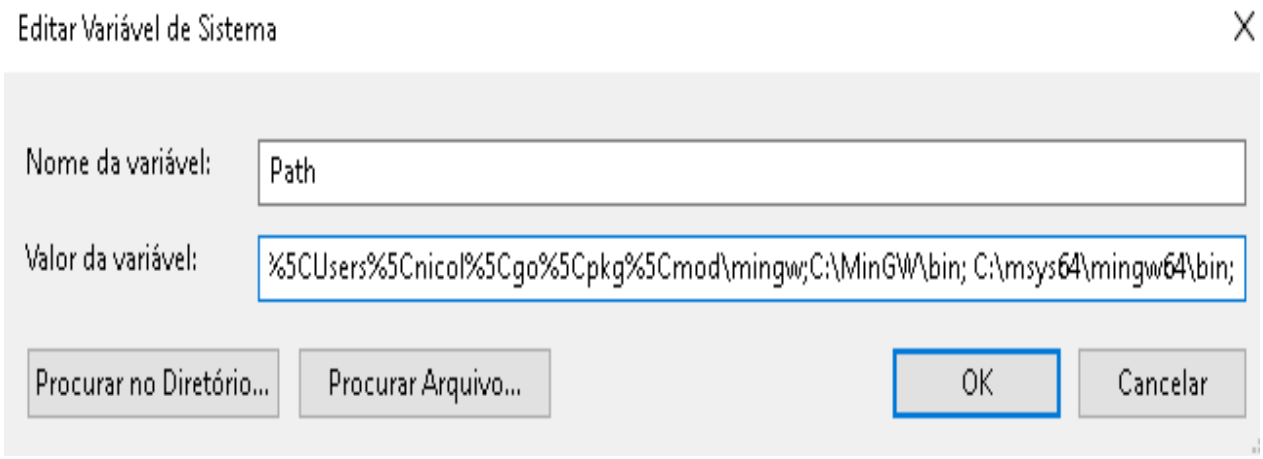
15. Vá até “Variáveis do sistema”, clique em “Path” e depois “Editar”;



16. Clique em “Editar texto”;



17. Ao final da linha “Valor da variável”, adicione “C:\msys64\mingw64\bin;”, como está na foto, e aperte em “Ok”;



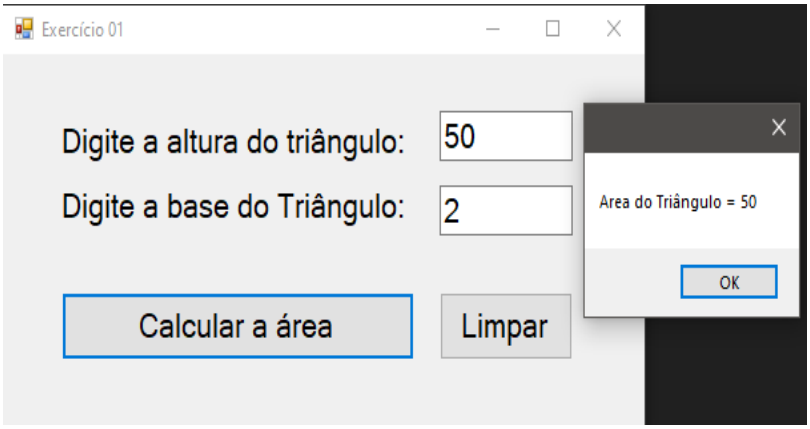
4.2 Windows Form APP

Agora que você já instalou o framework está pronto para desenvolver alguns códigos. Mostraremos a seguir uma lista com quatro (4) exercícios na linguagem C# e, sua recodificação para a linguagem estudada, ou seja, Golang. Vamos lá!

Exercício 1 – Neste primeiro exercício temos o cálculo da área de um triangulo, onde o usuário digita a altura *"alturatriangulo"* e a base *"basetriangulo"*, ao apertar o botão *"button1"* o programa calcula e exibe o resultado *"areatriangulo"* a área. Ao apertar o *"button2"* os dados digitados e o resultado são apagados e o usuário pode digitar novamente. **Lembre-se, este é um exemplo de código em C#.**

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace Exercicio01
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        private void button1_Click(object sender, EventArgs e)
        {
            float alturatriangulo = float.Parse(textBox1.Text);
            float basetriangulo = float.Parse(textBox2.Text);
            float areatriangulo = (alturatriangulo * basetriangulo) / 2;
            MessageBox.Show("Area do Triângulo = " + areatriangulo);
        }

        private void button2_Click(object sender, EventArgs e)
        {
            textBox1.Clear();
            textBox2.Clear();
            textBox1.Focus();
        }}}
```

Observe o form.



Agora que você já possui uma base de como ficaria o exercício 1 em C#, vamos ver o mesmo exercício na linguagem Go.

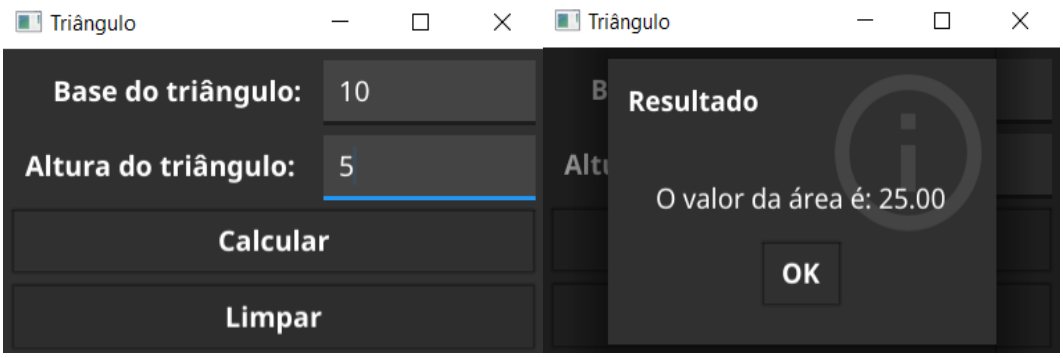
O usuário digita a base desejada em um text "*entradaBaseTriangulo"*, esse valor é armazenado em uma variável chamada 'valorBase'. O mesmo acontece na variável 'valorAltura'. Após a digitação o usuário pode apertar o botão "*botaoCalcular"*, assim, na variável, 'area' o cálculo é feito e o resultado é exibido na tela. Em Go precisamos usar os caracteres '%.2f' para exibirmos o resultado que está armazenado em uma variável, assim como no trecho: fmt.Sprintf("O valor da área é: %.2f", area), onde o caractere sitado representa a variável 'area'.

O *"botaoLimpar"*, assim como o "button2" que vimos em C#, também apaga todos os dados e o resultado e permite um novo cálculo.

```
package main

import (
    "fmt"
    "strconv"

    "fyne.io/fyne/v2"
    "fyne.io/fyne/v2/app"
    "fyne.io/fyne/v2/container"
    "fyne.io/fyne/v2/dialog"
    "fyne.io/fyne/v2/layout"
    "fyne.io/fyne/v2/widget"
)
```

```
func main() {
    aplicativo := app.New()
    janela := aplicativo.NewWindow("Triângulo")
    entradaBaseTriangulo := widget.NewEntry()
    entradaAlturaTriangulo := widget.NewEntry()
    areaResultado := widget.NewEntry()

    botaoLimpar := widget.NewButton("Limpar", func() {
        entradaBaseTriangulo.SetText("")
        entradaAlturaTriangulo.SetText("")
        areaResultado.SetText("")
    })

    botaoCalcular := widget.NewButton("Calcular", func() {
        valorBase, _ := strconv.ParseFloat(entradaBaseTriangulo.Text, 64)
        valorAltura, _ := strconv.ParseFloat(entradaAlturaTriangulo.Text,
 64)
        area := (valorBase * valorAltura) / 2
        dialog.ShowInformation("Resultado", fmt.Sprintf("O valor da área
é: %.2f", area), janela)
    })
    containerBotoes := container.New(layout.NewVBoxLayout(), botaoCalcula
r, botaoLimpar)
    areaResultado.Disable()
    form := widget.NewForm(
        widget.NewFormItem("Base do triângulo:", entradaBaseTriangulo),
        widget.NewFormItem("Altura do triângulo: ", entradaAlturaTriangul
o),
    )

    container := container.New(layout.NewVBoxLayout(), form, containerBot
oes)
    janela.SetContent(container)
    janela.Resize(fyne.NewSize(300, 160))

    janela.ShowAndRun()
}
```

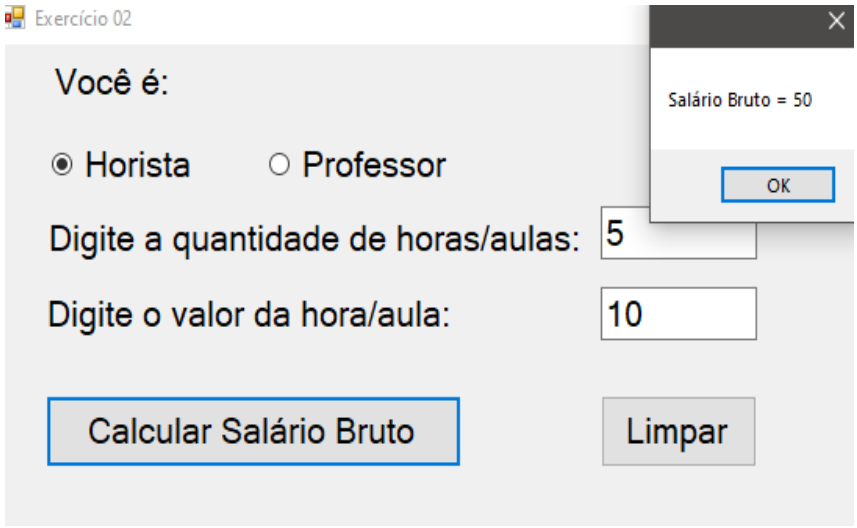
Agora veja como fica o form.



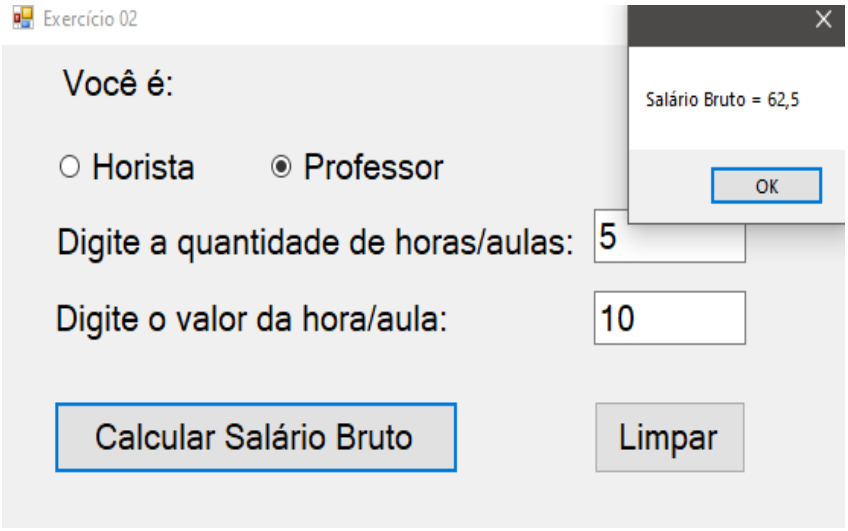
Exercício 2 – Neste exercício calculamos o salário bruto 'sb' do horista ou professor. Caso o usuário selecione a opção "horista" o cálculo do salário é: quantidade de horas 'qtd' vezes o valor da hora 'valor', já se o usuário selecionar a opção "professor" o cálculo do salário é: (quantidade de horas 'qtd' vezes o valor da hora 'valor', já se o usuário selecionar a opção "professor") vezes 1,25. Ao apertar "calcular" exibe-se o resultado em uma messageBox. O botão "limpar" limpa os dados digitados. **Lembre-se, este é um exemplo de código em C#.**

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace Exercicio02
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
        private void button1_Click(object sender, EventArgs e)
        {
            float qtd = float.Parse(textBox1.Text);
            float valor = float.Parse(textBox2.Text);
            float sb;

            if (radioButton1.Checked)
            {
                sb = qtd * valor;
            }
            Else {sb = (qtd * valor) * 1.25f;}
            MessageBox.Show("Salário Bruto = " + sb);
        }
        private void button2_Click(object sender, EventArgs e)
        {
            textBox1.Clear();
            textBox2.Clear();
            textBox1.Focus();
        }
    }}
```

Observe o form do Horista.



Agora na opção “Professor”.



O mesmo programa em Go ficaria assim:

```
package main

import (
    "fmt"
    "strconv"

    "fyne.io/fyne/v2"
    "fyne.io/fyne/v2/app"
```

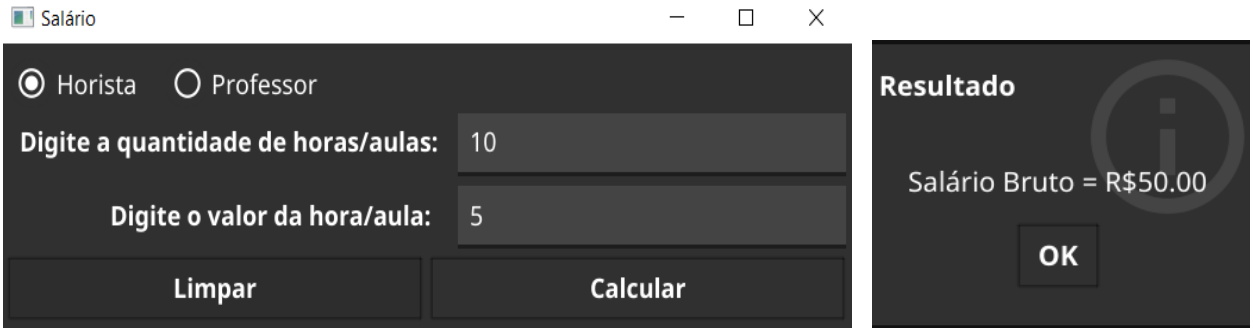
```
    "fyne.io/fyne/v2/container"
    "fyne.io/fyne/v2/dialog"
    "fyne.io/fyne/v2/layout"
    "fyne.io/fyne/v2/widget"
)

func main() {
    //Cria a janela e a nossa aplicação
    aplicativo := app.New()
    janela := aplicativo.NewWindow("Salário")
    var fatorMultiplicacao float64

    /*Criação dos radio buttons,
    os círculos que permitem selecionar uma única alternativa*/
    radio := widget.NewRadioGroup([]string{"Horista", "Professor"}, func(
value string) {
        //O salário será multiplicado conforme a opção selecionada
        if value == "Professor" {
            fatorMultiplicacao = 1.25
        } else {
            fatorMultiplicacao = 1
        }
    })
    /*Ordena os botões horizontalmente e
    coloca "horista" marcado por padrão.*/
    radio.Horizontal = true
    radio.SetSelected("Horista")

    //Criação dos widgets de entrada e do formulário
    quantidadeAulasHoras := widget.NewEntry()
    valorAulaHora := widget.NewEntry()
    form := widget.NewForm(
        widget.NewFormItem("Digite a quantidade de horas/aulas:", quantid
adeAulasHoras),
        widget.NewFormItem("Digite o valor da hora/aula: ", valorAulaHora
),
    )

    //Criação do botão limpar
    botaoLimpar := widget.NewButton("Limpar", func() {
        quantidadeAulasHoras.SetText("")
        valorAulaHora.SetText("")
    })

    /*Criação do botão calcular, que exibe
    o salário em uma caixa de diálogo.*/
    botaoCalcular := widget.NewButton("Calcular", func() {
        quantidade, _ := strconv.ParseFloat(quantidadeAulasHoras.Text, 64
)
```

```
        valor, _ := strconv.ParseFloat(valorAulaHora.Text, 64)
        salario := quantidade * valor * fatorMultiplicacao
        dialog.ShowInformation("Resultado", fmt.Sprintf("Salário Bruto =
R$%.2f", salario), janela)
    })

    //Criação de containers para juntar os elementos criados
    containerBotoes := container.New(layout.NewGridLayoutWithColumns(2),
botaoLimpar, botaoCalcular)
    container := container.New(layout.NewVBoxLayout(), radio, form, conta
inerBotoes)

    //Coloca o conteúdo na janela e altera seu tamanho
    janela.SetContent(container)
    janela.Resize(fyne.NewSize(550, 140))
    janela.ShowAndRun()
}
```

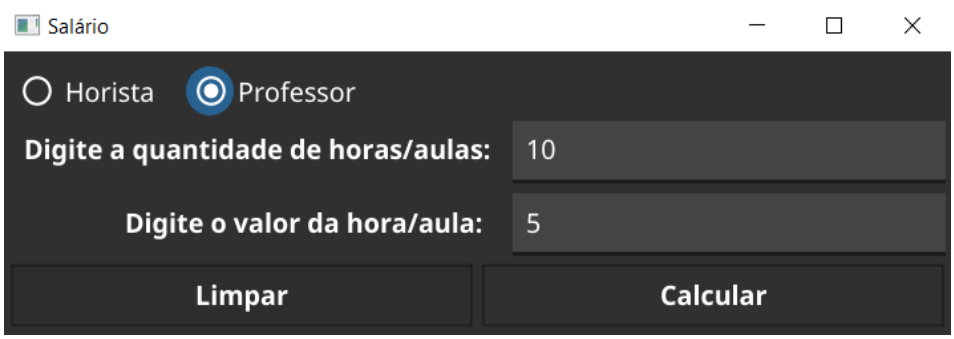
Agora a quantidade de aulas/horas e armazenada em uma variável chamada ‘quantidadedeAulasHoras’ e o valor da horam em ‘valorAulaHoa’. A equação utilizada é a mesma que vimos em C#. Para uma melhor compreensão, leia os comentários presentes no código.

Agora que você já leu todo o código vamos ver como ficou o Form.

Salário “horista”:



Salário “professor”:

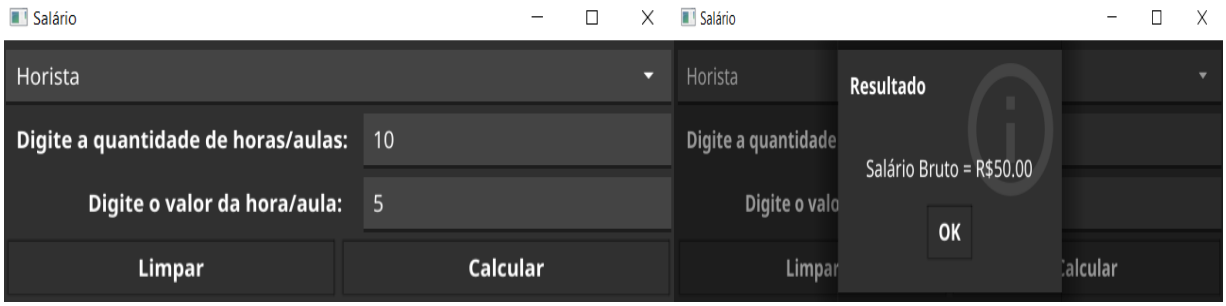


Exercício 3 –

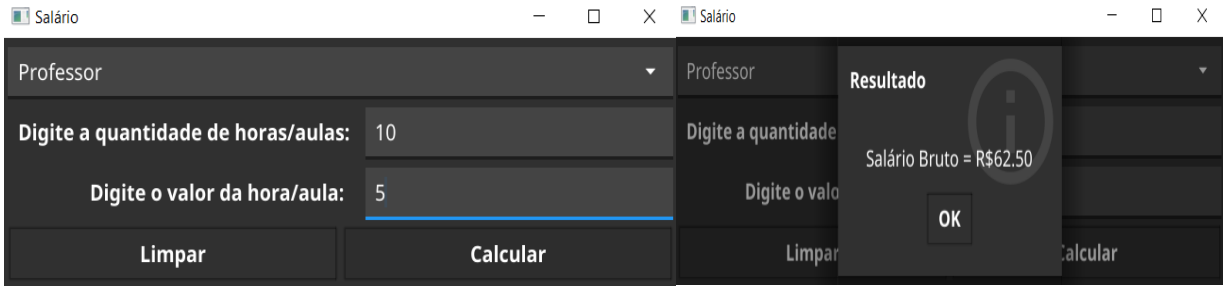
```
package main

import (
    "fmt"
    "strconv"

    "fyne.io/fyne/v2"
    "fyne.io/fyne/v2/app"
    "fyne.io/fyne/v2/container"
    "fyne.io/fyne/v2/dialog"
    "fyne.io/fyne/v2/layout"
    "fyne.io/fyne/v2/widget"
)

func main() {
    //Cria a janela e a nossa aplicação
    aplicativo := app.New()
    janela := aplicativo.NewWindow("Salário")
    var fatorMultiplicacao float64

    /*Criação de uma comboBox,
    a "caixinha" que permite selecionar uma única alternativa*/
    combo := widget.NewSelect([]string{"Horista", "Professor"}, func(valu
e string) {
        //O salário será multiplicado conforme a opção selecionada
        if value == "Professor" {
            fatorMultiplicacao = 1.25
        } else {
            fatorMultiplicacao = 1
        }
    })
```

```
    //Coloca "horista" marcado por padrão
    combo.SetSelected("Horista")

    //Criação dos widgets de entrada e do formulário
    quantidadeAulasHoras := widget.NewEntry()
    valorAulaHora := widget.NewEntry()
    form := widget.NewForm(
        widget.NewFormItem("Digite a quantidade de horas/aulas:", quantid
adeAulasHoras),
        widget.NewFormItem("Digite o valor da hora/aula: ", valorAulaHora
),
    )

    //Criação do botão limpar
    botaoLimpar := widget.NewButton("Limpar", func() {
        quantidadeAulasHoras.SetText("")
        valorAulaHora.SetText("")
    })

    /*Criação do botão calcular, que exibe
    o salário em uma caixa de diálogo.*/
    botaoCalcular := widget.NewButton("Calcular", func() {
        quantidade, _ := strconv.ParseFloat(quantidadeAulasHoras.Text, 64
)
        valor, _ := strconv.ParseFloat(valorAulaHora.Text, 64)
        salario := quantidade * valor * fatorMultiplicacao
        dialog.ShowInformation("Resultado", fmt.Sprintf("Salário Bruto =
R$%.2f", salario), janela)
    })

    //Criação de containers para juntar os elementos criados
    containerBotoes := container.New(layout.NewGridLayoutWithColumns(2),
botaoLimpar, botaoCalcular)
    container := container.New(layout.NewVBoxLayout(), combo, form, conta
inerBotoes)

    //Coloca o conteúdo na janela e altera seu tamanho
    janela.SetContent(container)
    janela.Resize(fyne.NewSize(550, 140))
    janela.ShowAndRun()
}
```

SALÁRIO HORISTA:



SALÁRIO PROFESSOR:

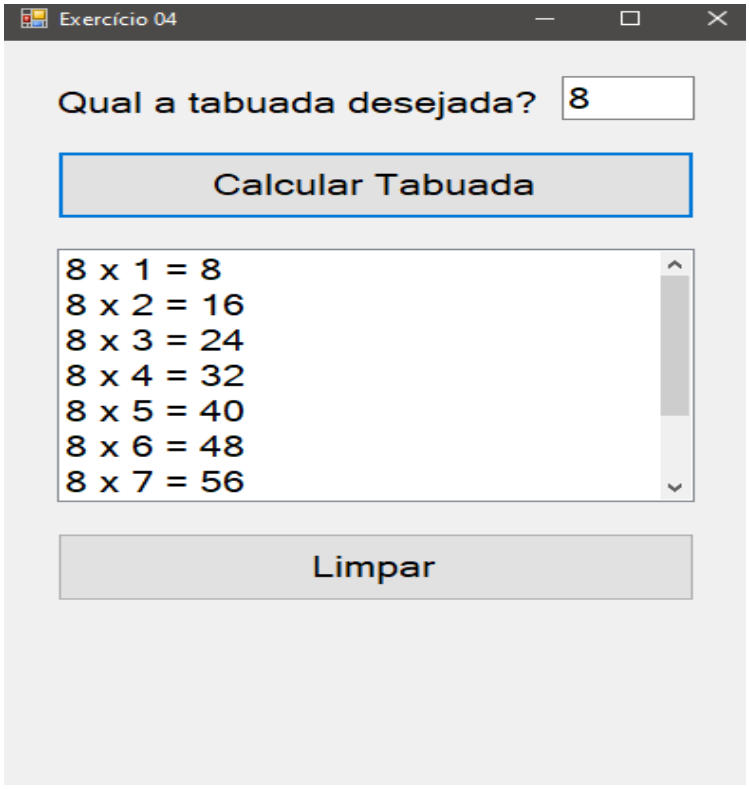


Exercício 4 – Nesse exercício o usuário digita um valor, o qual ele deseja saber a tabuada, em uma TextBox. Após a digitação ele aperta em "Calcular Tabuada" e o a tabuada do número escolhido é impressa em uma outra TextBox. **Lembre-se, este é um exemplo de código em C#.**

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace Exercicio04
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
```

```
        }

        private void button1_Click(object sender, EventArgs e)
        {
            int tabuada = int.Parse(textBox1.Text);

            for (int i = 1; i < 11; ++i)
            {
                listBox1.Items.Add(tabuada + " x " + i + " = " + tabuada
* i);
            }
        }

        private void button2_Click(object sender, EventArgs e)
        {
            textBox1.Clear();
            listBox1.Items.Clear();
            textBox1.Focus();
        }
    }
}
```

Veja como fica o Form.



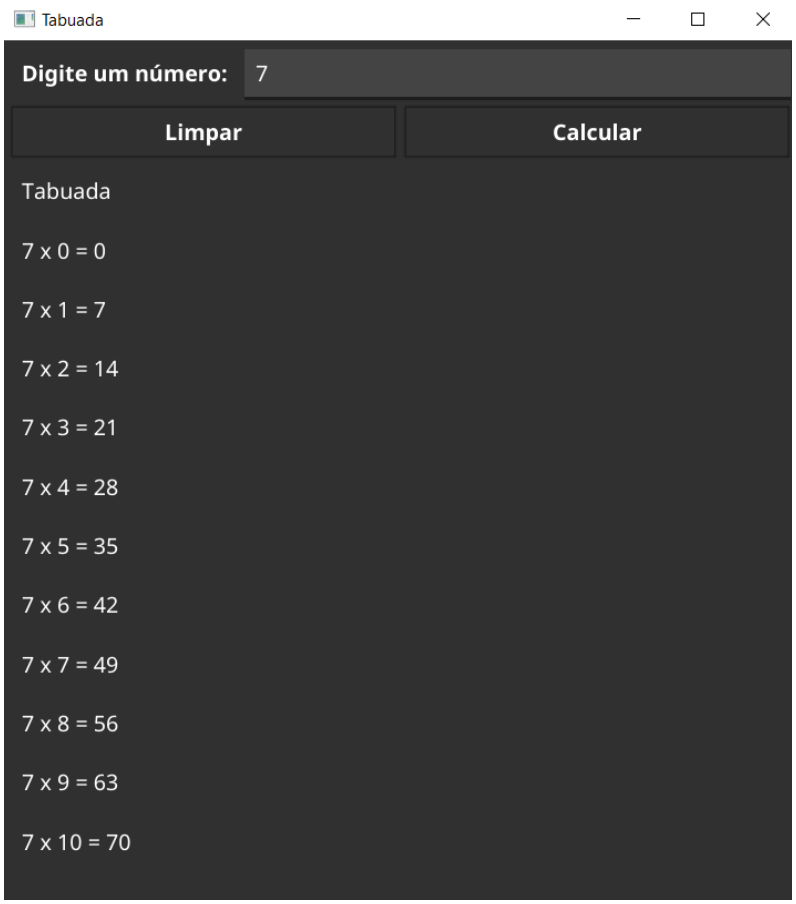
Assim como os outros programas já apresentados, o botão “Limpar” apaga todos os dados e o usuário pode digitar um novo valor.

Agora observe o mesmo programa na linguagem Go.

```
package main

import (
    "fmt"
    "strconv"

    "fyne.io/fyne/v2"
    "fyne.io/fyne/v2/app"
    "fyne.io/fyne/v2/container"
    "fyne.io/fyne/v2/layout"
    "fyne.io/fyne/v2/widget"
)

func main() {
    //Cria a janela e a nossa aplicação
    aplicativo := app.New()
    janela := aplicativo.NewWindow("Tabuada")
    /*Criação do vetor que irá conter os itens de exibição
    da nossa tabuada*/
    var tab [11]*widget.Label

    //Criação do widget de entrada
    numeroTabuada := widget.NewEntry()
    form := widget.NewForm(
        widget.NewFormItem("Digite um número:", numeroTabuada),
    )
    containerTabuada := container.New(layout.NewVBoxLayout(), widget.NewL
abel("Tabuada"))
    //Criação do botão limpar
    botaoLimpar := widget.NewButton("Limpar", func() {
        limpar(containerTabuada, tab)
    })

    /*Criação do botão calcular, que limpa os resultados anteriores e,
    em seguida, alimenta o array tab com as Labels e as adiciona em um co
ntainer.*/
    botaoCalcular := widget.NewButton("Calcular", func() {
        limpar(containerTabuada, tab)
        numTab, _ := strconv.ParseFloat(numeroTabuada.Text, 64)
        for i := 0; i < len(tab); i++ {
            tab[i] = widget.NewLabel(fmt.Sprintf("%v x %v = %v", numTab,
i, numTab*float64(i)))
            containerTabuada.Add(tab[i])
        }
    })

    //Criação de containers para juntar os elementos criados
```

```
    containerBotoes := container.New(layout.NewGridLayoutWithColumns(2),
botaoLimpar, botaoCalcular)
    container := container.New(layout.NewVBoxLayout(), form, containerBot
oes, containerTabuada)

    //Coloca o conteúdo na janela e altera seu tamanho
    janela.SetContent(container)
    janela.Resize(fyne.NewSize(550, 600))
    janela.ShowAndRun()
}

/*Remove do container as Labels armazenadas
  em um vetor */
func limpar(container *fyne.Container, vetor [11]*widget.Label) {
    for i := 0; i < len(vetor); i++ {
        container.Remove(vetor[i])
    }}
```

Para entender melhor leia os comentários deixados no corpo do código, após a leitura, observe o Form.



5 PROGRAMAÇÃO ORIENTADA À OBJETOS (POO)

Está na hora de darmos um passo maior, deixar as coisas um pouco mais “difíceis”. No capítulo 4 desta apostila você aprenderá como funciona a/o POO em Golang.

Go não tem classes, objetos, exceções e nem modelos. A omissão mais aparente, em relação a orientação a objetos, é que não há hierarquia de tipos em Go. Mas então Go não é uma linguagem orientada à objetos? É e não é! Go tem estruturas (tipos definidos pelo usuário) e métodos que permitem um estilo de programação orientado a objetos.

Para ajudar a fixar essa ideia vamos apresentar alguns exemplos de programas.

5.1 Para entender melhor

No capítulo anterior você viu alguns exercícios, nós utilizaremos o exercício 1 e 2 para ajudar você a entender melhor como funciona a linguem de programação orientada à objeto (POO) em Go e mais dois (2) exercícios para sua fixação.

Porém, antes de iniciarmos a apresentação, é importante ressaltarmos que em **Go NÃO EXISTE** classes, embora tenha Structs (tipos definidos pelos usuários).

OBS: o código e o diretório responsáveis pelos erros continuarão os mesmos em todos os códigos a seguir.

Exercício 1- Nesse exercício será calculada a área do triangulo.

```go
package erro

/*Como não existem classes, muito menos membros estáticos
em golang, a solução para se adequar ao exercício
é criar uma única "instância" de Erro e atualizá-la
em todas as etapas do programa.*/

/* CRIANDO A INSTANCIA ERRO */
type Erro struct {
    erro     bool
    mensagem string
}

func (e *Erro) GetErro() bool {
    return e.erro
}

func (e *Erro) GetMensagem() string {
    return e.mensagem
}
```

```
/*Em Go, não existe a sobrecarga de métodos.
Por isso, foram criados métodos set com nomes distintos.*/
func (e *Erro) SetErroBool(erro bool) {
    e.erro = erro
}

func (e *Erro) SetErroString(mensagem string) {
    e.erro = true
    e.mensagem = mensagem
}
package triangulo

import (
    "fmt"
    "strconv"
)
/*CRIANDO A ESTRUTURA PRINCIPAL DO PROGRAMA*/
type Triangulo struct {
    base   string
    altura string
}

/* *Triangulo significa que este método é um
receptor de ponteiro. Ele aponta para o struct
que chama este método, e é capaz de modificar seus
valores. */
func (t *Triangulo) SetBase(base string) {
    t.base = base
}

func (t *Triangulo) SetAltura(altura string) {
    t.altura = altura
}

func (t *Triangulo) GetAltura() string {
    return t.altura
}

func (t *Triangulo) GetBase() string {
    return t.base
}

func (t Triangulo) GetArea() string {
    baseFloat, _ := strconv.ParseFloat(t.base, 64)
    alturaFloat, _ := strconv.ParseFloat(t.altura, 64)
    area := fmt.Sprint((baseFloat * alturaFloat) / 2)
    return area
}
```

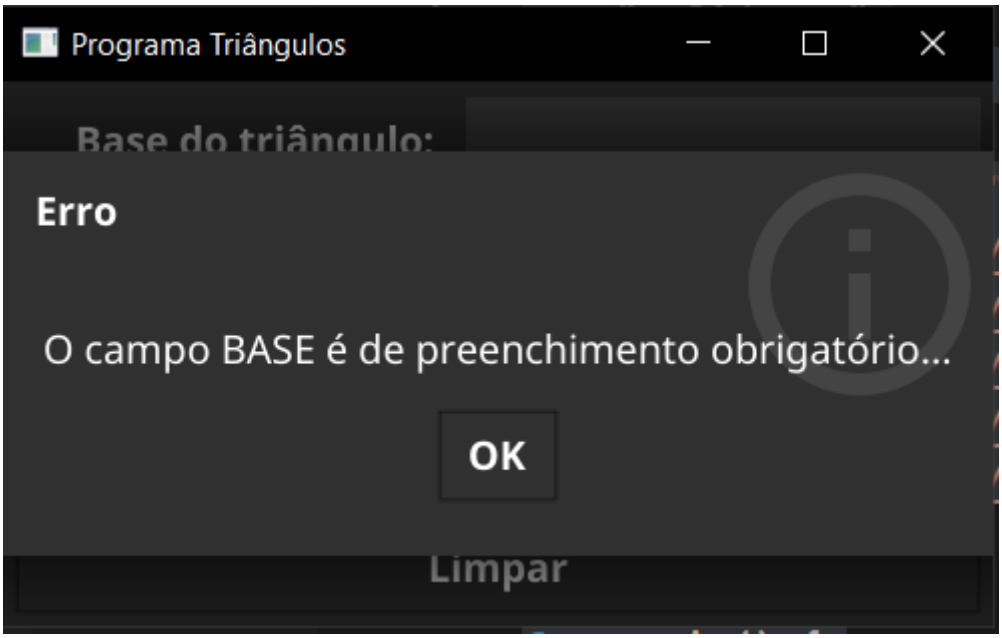
```
/*CRIANDO A VALIDAÇÃO DOS CAMPOS*/
package triangulo

import (
    "ex01/erro"
    "strconv"
)

/*Não existe uma representação de campos estáticos
para "tratar" o objeto triângulo, como feito
com a classe TrianguloBLL no C#. Para invocar
este código, chamaremos o nome do pacote.*/
func ValidaDados(e *erro.Erro, t *Triangulo) {
    e.SetErroBool(false)
    //Verificação campo vazio
    if len(t.GetBase()) == 0 {
        e.SetErroString("O campo BASE é de preenchimento obrigatório...")
        return
    }
    //Verificação campo não-numérico
    if _, err := strconv.ParseFloat(t.GetBase(), 64); err != nil {
        e.SetErroString("O campo BASE deve ser numérico...")
        return
        //Verificação intervalo número
    } else if num, _ := strconv.ParseFloat(t.GetBase(), 64); num <= 0 {
        e.SetErroString("O campo BASE deve ser maior que zero.")
        return
    }

    //Verificação campo vazio
    if len(t.GetAltura()) == 0 {
        e.SetErroString("O campo ALTURA é de preenchimento obrigatório...
")
        return
    }
    //Verificação campo não-numérico
    if _, err := strconv.ParseFloat(t.GetAltura(), 64); err != nil {
        e.SetErroString("O campo ALTURA deve ser numérico...")
        return
        //Verificação intervalo número
    } else if num, _ := strconv.ParseFloat(t.GetAltura(), 64); num <= 0 {
        e.SetErroString("O campo ALTURA deve ser maior que zero.")
        return
    }

}
package main
```

```
import (
    "ex01/erro"
    "ex01/triangulo"

    "fyne.io/fyne/v2"
    "fyne.io/fyne/v2/app"
    "fyne.io/fyne/v2/container"
    "fyne.io/fyne/v2/dialog"
    "fyne.io/fyne/v2/layout"
    "fyne.io/fyne/v2/widget"
)

func main() {
    aplicativo := app.New()
    janela := aplicativo.NewWindow("Programa Triângulos")
    entradaBaseTriangulo := widget.NewEntry()
    entradaAlturaTriangulo := widget.NewEntry()
    areaResultado := widget.NewEntry()
    //Aloca memória para um tipo triangulo, mas t continua zerada
    t := new(triangulo.Triangulo)
    erro := new(erro.Erro)

    botaoLimpar := widget.NewButton("Limpar", func() {
        entradaBaseTriangulo.SetText("")
        entradaAlturaTriangulo.SetText("")
        areaResultado.SetText("")
        entradaBaseTriangulo.FocusGained()
    })

    botaoCalcular := widget.NewButton("Calcular", func() {
        /*Por serem de pacotes diferentes, o main não consegue chamar
        os campos de t, somente seus métodos. O que faz eles serem
        públicos ou não é a presença do caractere maiúsculo no início
        do identificador. */
        t.SetBase(entradaBaseTriangulo.Text)
        t.SetAltura(entradaAlturaTriangulo.Text)
        triangulo.ValidaDados(erro, t)
        if erro.GetErro() {
            dialog.ShowInformation("Erro", erro.GetMensagem(), janela)
            return
        }
        areaResultado.SetText(t.GetArea())
    })
    containerBotoes := container.New(layout.NewVBoxLayout(), botaoCalcula
r, botaoLimpar)
    areaResultado.Disable()
    form := widget.NewForm(
     widget.NewFormItem("Base do triângulo:", entradaBaseTriangulo),
     widget.NewFormItem("Altura do triângulo: ", entradaAlturaTriangulo),
```

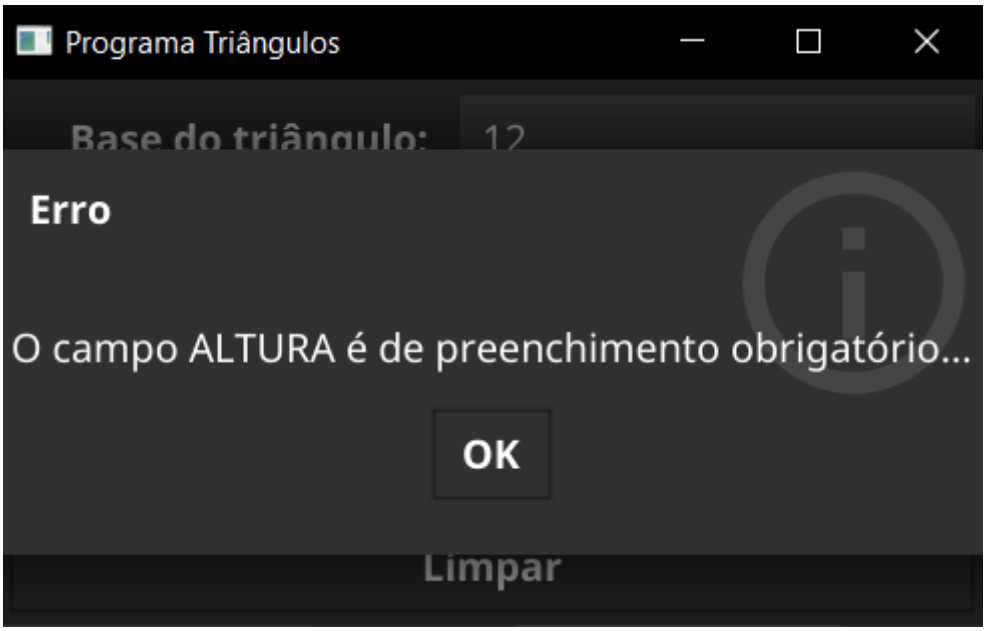
```
        widget.NewFormItem("Área: ", areaResultado),    )
    container := container.New(layout.NewVBoxLayout(), form, containerBot
oes)
    janela.SetContent(container)
    janela.Resize(fyne.NewSize(380, 175))
    janela.ShowAndRun()

}
```

E assim a gente finaliza o código. Observe agora os possíveis erros e suas mensagens:

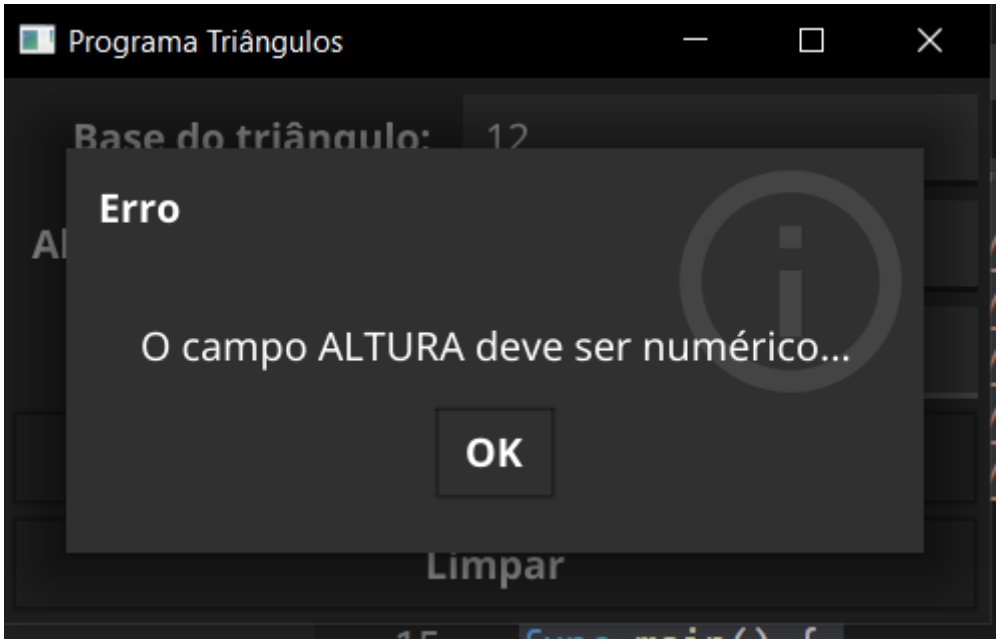
- Erro quando não for preenchido o campo Base:



- Erro quando não for preenchido o campo Altura:



- Erro quando o preenchimento dos campos (base ou altura) não forem valores numéricos:



Por fim, a execução sem nenhum erro:

Exercício 2 - Neste exercício calculamos o salário bruto do horista. O cálculo utilizado é idêntico ao exercício dois (2) do capítulo anterior.

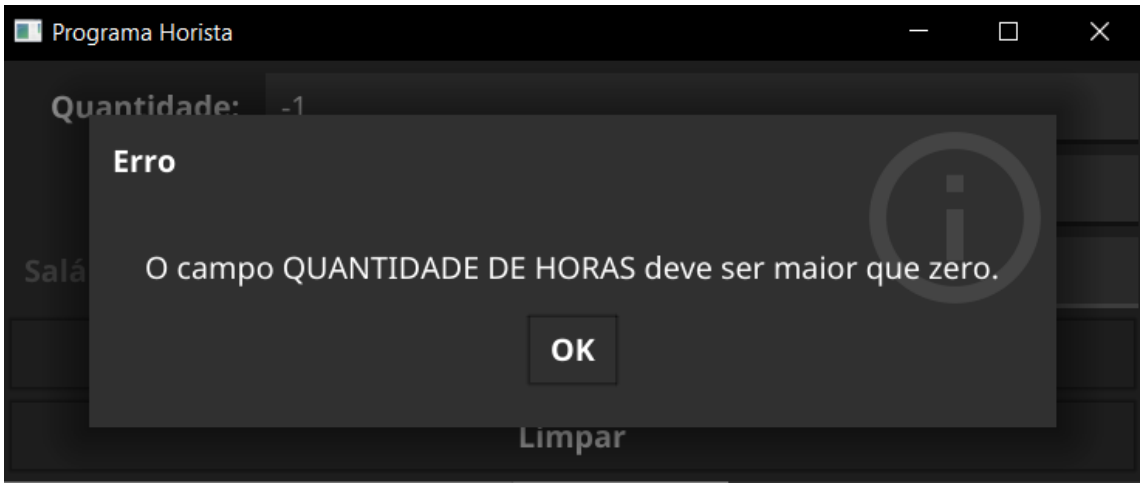
```
/*CRIAÇÃO DO ESTRUTURA DO HORISTA*/
package horista

import (
    "fmt"
    "strconv"
)

type Horista struct {
    qtd   string
    valor string
}

func (t *Horista) SetQtd(qtd string) {
    t.qtd = qtd
}

func (t *Horista) SetValor(valor string) {
    t.valor = valor
}

func (t *Horista) GetValor() string {
    return t.valor
}

func (t *Horista) GetQtd() string {
    return t.qtd
}

func (t Horista) GetSalarioBruto() string {
    qtdFloat, _ := strconv.ParseFloat(t.qtd, 64)
    valorFloat, _ := strconv.ParseFloat(t.valor, 64)
    salarioBruto := fmt.Sprint((qtdFloat * valorFloat) / 2)
    return salarioBruto
}
/*CRIANDO A VALIDAÇÃO HORISTA*/
package horista

import (
    "ex01/erro"
    "strconv"
)

func ValidaDados(e *erro.Erro, t *Horista) {
    e.SetErroBool(false)
```

```
    //Verificação campo vazio
    if len(t.GetQtd()) == 0 {
        e.SetErroString("O campo QUANTIDADE DE HORAS é de preenchimento o
brigatório...")
        return
    }
    //Verificação campo não-numérico
    if _, err := strconv.ParseFloat(t.GetQtd(), 64); err != nil {
        e.SetErroString("O campo QUANTIDADE DE HORAS deve ser numérico...
")
        return
        //Verificação intervalo número
    } else if num, _ := strconv.ParseFloat(t.GetQtd(), 64); num <= 0 {
        e.SetErroString("O campo QUANTIDADE DE HORAS deve ser maior que z
ero.")
        return
    }

    //Verificação campo vazio
    if len(t.GetValor()) == 0 {
        e.SetErroString("O campo VALOR DA HORA é de preenchimento obrigat
ório...")
        return
    }
    //Verificação campo não-numérico
    if _, err := strconv.ParseFloat(t.GetValor(), 64); err != nil {
        e.SetErroString("O campo VALOR DA HORA deve ser numérico...")
        return
        //Verificação intervalo número
    } else if num, _ := strconv.ParseFloat(t.GetValor(), 64); num <= 0 {
        e.SetErroString("O campo VALOR DA HORA deve ser maior que zero.")
        return
    }

}
/* CRIANDO MAIN PRINCIPAL*/
package main

import (
    "ex01/erro"
    "ex01/horista"

    "fyne.io/fyne/v2"
    "fyne.io/fyne/v2/app"
    "fyne.io/fyne/v2/container"
    "fyne.io/fyne/v2/dialog"
    "fyne.io/fyne/v2/layout"
    "fyne.io/fyne/v2/widget"
)
```

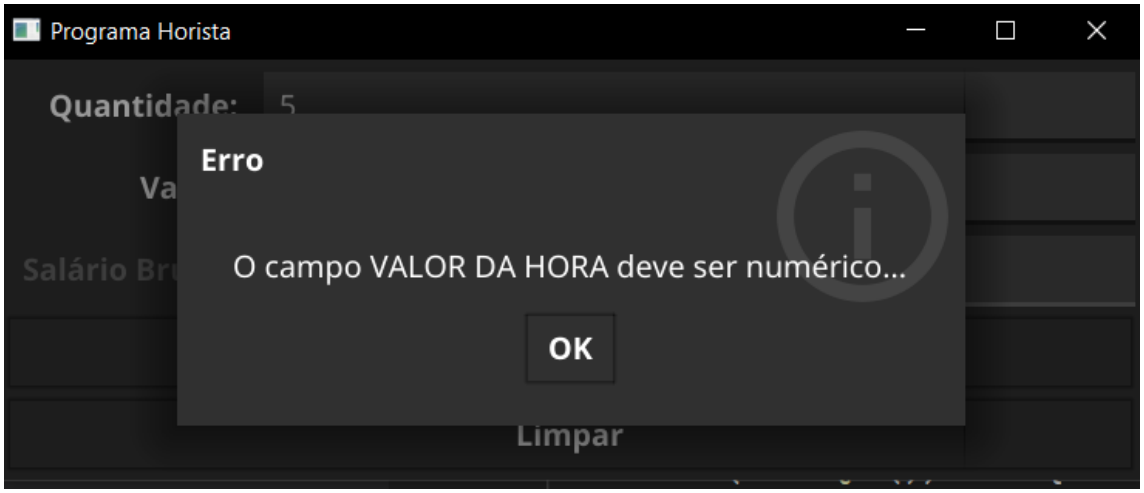
```
func main() {
    aplicativo := app.New()
    janela := aplicativo.NewWindow("Programa Horista")
    entradaQtd := widget.NewEntry()
    entradaValor := widget.NewEntry()
    salarioBruto := widget.NewEntry()
    //Aloca memória para um tipo horista, mas t continua zerada
    t := new(horista.Horista)
    erro := new(erro.Erro)

    botaoLimpar := widget.NewButton("Limpar", func() {
        entradaQtd.SetText("")
        entradaValor.SetText("")
        salarioBruto.SetText("")
        entradaQtd.FocusGained()
    })

    botaoCalcular := widget.NewButton("Calcular", func() {
        t.SetQtd(entradaQtd.Text)
        t.SetValor(entradaValor.Text)
        horista.ValidaDados(erro, t)
        if erro.GetErro() {
            dialog.ShowInformation("Erro", erro.GetMensagem(), janela)
            return
        }
        salarioBruto.SetText(t.GetSalarioBruto())
    })
    containerBotoes := container.New(layout.NewVBoxLayout(), botaoCalcula
r, botaoLimpar)
    salarioBruto.Disable()
    form := widget.NewForm(
        widget.NewFormItem("Quantidade:", entradaQtd),
        widget.NewFormItem("Valor: ", entradaValor),
        widget.NewFormItem("Salário Bruto: ", salarioBruto),
    )
    container := container.New(layout.NewVBoxLayout(), form, containerBot
oes)
    janela.SetContent(container)
    janela.Resize(fyne.NewSize(570, 200))
    janela.ShowAndRun()
}
```

E assim a gente finaliza o código. Observe agora os possíveis erros e suas mensagens:

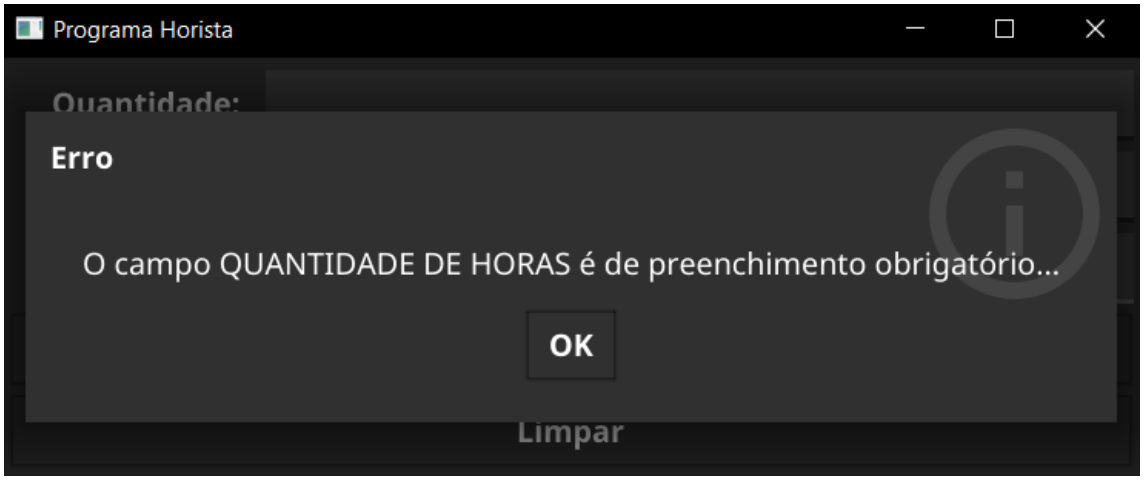
- Quando a quantidade de horas for inferior a 0:



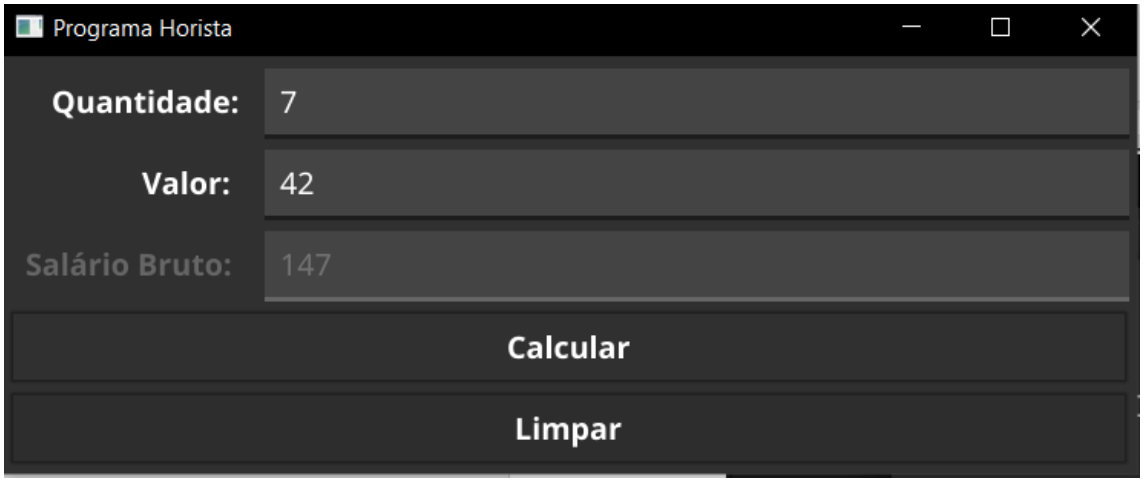
- Quando o valor da hora for diferente de um valor numérico:



- Quando o campo QUANTIDADE DE HORAS estiver vazio:



E por fim, a execução quando não houver erros:



Exercício 3 - Nesse exercício o usuário entrara com três (3) valores: A, B e C, que representarão os números que compõem uma equação do segundo grau "$ax^2+bx+c=0$". Após a digitação será exibido os possíveis resultados, seja os valores dos 'x' encontrados ou a mensagem de erro informando que não existe raiz para os números digitados. Observe o código.

```
package equacao

import (
    "fmt"
    "math"
    "strconv"
)

type Equacao struct {
    a string
    b string
    c string
}

func (t *Equacao) SetA(a string) {
    t.a = a
}

func (t *Equacao) GetA() string {
    return t.a
}

func (t *Equacao) SetB(b string) {
    t.b = b
}
```

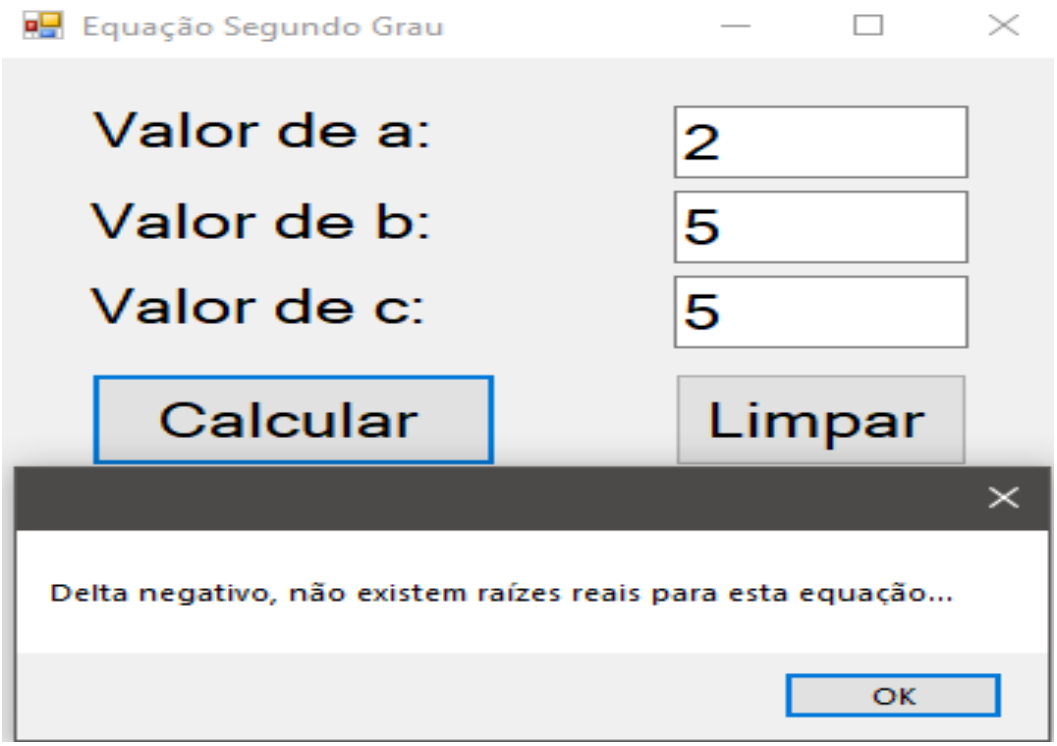
```
func (t *Equacao) GetB() string {
    return t.b
}

func (t *Equacao) SetC(c string) {
    t.c = c
}

func (t *Equacao) GetC() string {
    return t.c
}

func (t *Equacao) GetDelta() string {
    floatA, _ := strconv.ParseFloat(t.a, 64)
    floatB, _ := strconv.ParseFloat(t.b, 64)
    floatC, _ := strconv.ParseFloat(t.c, 64)
    return fmt.Sprint(((floatB * floatB) - (4 * floatA * floatC)))
}

func (t *Equacao) GetX1() string {
    floatA, _ := strconv.ParseFloat(t.a, 64)
    floatB, _ := strconv.ParseFloat(t.b, 64)
    delta, _ := strconv.ParseFloat(t.GetDelta(), 64)
    return fmt.Sprint(((-floatB + math.Sqrt(delta)) / (2 * floatA)))
}

func (t *Equacao) GetX2() string {
    floatA, _ := strconv.ParseFloat(t.a, 64)
    floatB, _ := strconv.ParseFloat(t.b, 64)
    delta, _ := strconv.ParseFloat(t.GetDelta(), 64)
    return fmt.Sprint(((-floatB - math.Sqrt(delta)) / (2 * floatA)))
}
/* VALIDAÇÃO DA EQUAÇÃO*/
package equacao

import (
    "ex03/erro"
    "strconv"
)
func ValidaDados(e *erro.Erro, t *Equacao) {
    e.SetErroBool(false)
    //Verificação campo vazio
    if len(t.GetA()) == 0 {
        e.SetErroString("O valor A é de preenchimento obrigatório...")
        return
    }
    //Verificação campo não-numérico
    if _, err := strconv.ParseFloat(t.GetB(), 64); err != nil {
```

```
        e.SetErroString("O valor B deve ser numérico...")
        return
    }
    //Verificação campo vazio
    if len(t.GetB()) == 0 {
        e.SetErroString("O valor B é de preenchimento obrigatório...")
        return
    }
    //Verificação campo não-numérico
    if _, err := strconv.ParseFloat(t.GetC(), 64); err != nil {
        e.SetErroString("O valor C deve ser numérico...")
        return
    }
    //Verificação campo vazio
    if len(t.GetC()) == 0 {
        e.SetErroString("O valor C é de preenchimento obrigatório...")
        return
    }
    //Verificação campo não-numérico
    if _, err := strconv.ParseFloat(t.GetC(), 64); err != nil {
        e.SetErroString("O valor C deve ser numérico...")
        return
    }
    if num, _ := strconv.ParseFloat(t.GetDelta(), 64); num < 0 {
        e.SetErroString("Delta negativo, não existem raízes reais para es
ta equação...")
        return
    }
}
package main

import (
    "ex03/equacao"
    "ex03/erro"

    "fyne.io/fyne/v2"
    "fyne.io/fyne/v2/app"
    "fyne.io/fyne/v2/container"
    "fyne.io/fyne/v2/dialog"
    "fyne.io/fyne/v2/layout"
    "fyne.io/fyne/v2/widget"
)

func main() {
    aplicativo := app.New()
    janela := aplicativo.NewWindow("Programa Equação")
    entradaA := widget.NewEntry()
    entradaB := widget.NewEntry()
    entradaC := widget.NewEntry()
```

```
    entradaX1 := widget.NewEntry()
    entradaX2 := widget.NewEntry()

    t := new(equacao.Equacao)
    erro := new(erro.Erro)

    botaoLimpar := widget.NewButton("Limpar", func() {
        entradaA.SetText("")
        entradaB.SetText("")
        entradaC.SetText("")
        entradaX2.SetText("")
        entradaX1.SetText("")
        entradaA.FocusGained()
    })

    botaoCalcular := widget.NewButton("Calcular", func() {
        //Recebimento de dados para a estrutura equacao
        t.SetA(entradaA.Text)
        t.SetB(entradaB.Text)
        t.SetC(entradaC.Text)
        //Validacao
        equacao.ValidaDados(erro, t)
        //Exibicao do Erro
        if erro.GetErro() {
            dialog.ShowInformation("Erro", erro.GetMensagem(), janela)
            return
        }
        entradaX1.SetText(t.GetX1())
        entradaX2.SetText(t.GetX2())
    })

    containerBotoes := container.New(layout.NewVBoxLayout(), botaoCalcula
r, botaoLimpar)
    form := widget.NewForm(
        widget.NewFormItem("Valor de A:", entradaA),
        widget.NewFormItem("Valor de B: ", entradaB),
        widget.NewFormItem("Valor de C: ", entradaC),
    )
    form2 := widget.NewForm(
        widget.NewFormItem("X1:", entradaX1),
        widget.NewFormItem("X2: ", entradaX2),
    )

    container := container.New(layout.NewVBoxLayout(), form, containerBot
oes, form2)
    janela.SetContent(container)
    janela.Resize(fyne.NewSize(420, 300))
    janela.ShowAndRun()
}
```

Veja como ficaria a execução desse programa caso não houvesse erros:

Agora se os números digitados não tivessem raiz reais:



Exercicio 4 – Esse programa funciona como uma livraria. O usuario seria, no caso, o bibliotecario. Sua função é inserir no "banco de dados" as informações dos livros, onde o botão SALVAR faz o cadastro do livro na biblioteca. O botão CONSULTAR serve para, como o nome já diz, consultar um livro, ou seja, pesquisar se o livro está cadastrado em sua biblioteca. Já o botão limpar, assim como nos outros exercicios, limpa os campos do form. Observe o código.

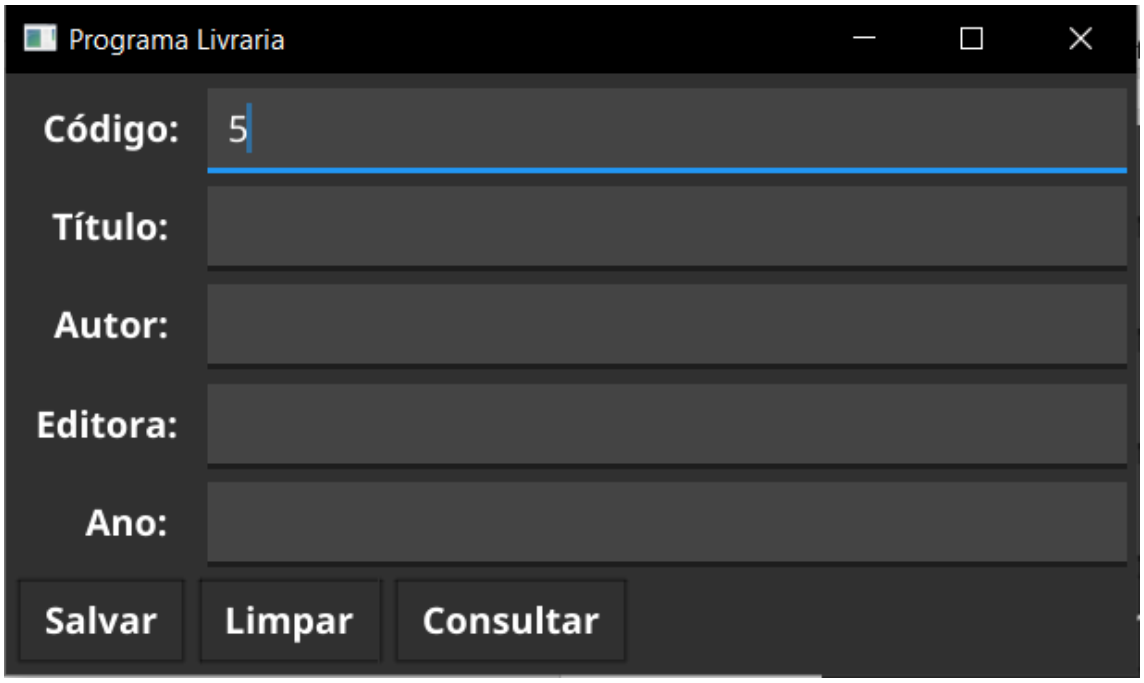
```
package livro

type Livro struct {
    codigo  string
    titulo  string
    autor   string
    editora string
    ano     string
}

func (t *Livro) SetCodigo(codigo string) {
    t.codigo = codigo
}

func (t *Livro) SetTitulo(titulo string) {
    t.titulo = titulo
}

func (t *Livro) SetAutor(autor string) {
    t.autor = autor
}

func (t *Livro) SetEditora(editora string) {
    t.editora = editora
}

func (t *Livro) SetAno(ano string) {
    t.ano = ano
}

func (t *Livro) GetCodigo() string {
    return t.codigo
}

func (t *Livro) GetTitulo() string {
    return t.titulo
}

func (t *Livro) GetAutor() string {
```

```
    return t.autor
}

func (t *Livro) GetEditora() string {
    return t.editora
}

func (t *Livro) GetAno() string {
    return t.ano
}
/* VALIDAÇÃO DO LIVRO*/
package livro

import (
    "ex04/erro"
    "strconv"
)

func ValidaCodigo(e *erro.Erro, t *Livro) {
    e.SetErroBool(false)
    if len(t.GetCodigo()) == 0 {
        e.SetErroString("O código é de preenchimento obrigatório!")
        return
    }
}

func ValidaDados(e *erro.Erro, t *Livro) {
    ValidaCodigo(e, t)
    if len(t.GetTitulo()) == 0 {
        e.SetErroString("O título é de preenchimento obrigatório!")
        return
    }
    if len(t.GetAutor()) == 0 {
        e.SetErroString("O autor é de preenchimento obrigatório!")
        return
    }
    if len(t.GetEditora()) == 0 {
        e.SetErroString("A editora é de preenchimento obrigatório!")
        return
    }
    if len(t.GetAno()) == 0 {
        e.SetErroString("O ano é de preenchimento obrigatório!")
        return
    }

    if _, erroConv := strconv.Atoi(t.GetAno()); erroConv != nil {
        e.SetErroString("O valor do ano deve ser numérico!")
        return
    } else if num, _ := strconv.Atoi(t.GetAno()); num <= 0 {
```

```
        e.SetErroString("O valor do ano deve ser positivo!")
        return
    }
}
/* MAIN*/
package main

import (
    "ex04/erro"
    "ex04/livro"

    "fyne.io/fyne/v2"
    "fyne.io/fyne/v2/app"
    "fyne.io/fyne/v2/container"
    "fyne.io/fyne/v2/dialog"
    "fyne.io/fyne/v2/layout"
    "fyne.io/fyne/v2/widget"
)

func main() {
    aplicativo := app.New()
    janela := aplicativo.NewWindow("Programa Livraria")
    entradaCodigo := widget.NewEntry()
    entradaTitulo := widget.NewEntry()
    entradaAutor := widget.NewEntry()
    entradaEditora := widget.NewEntry()
    entradaAno := widget.NewEntry()

    t := new(livro.Livro)
    erro := new(erro.Erro)

    botaoLimpar := widget.NewButton("Limpar", func() {
        entradaCodigo.SetText("")
        entradaTitulo.SetText("")
        entradaAutor.SetText("")
        entradaAno.SetText("")
        entradaEditora.SetText("")
        entradaCodigo.FocusGained()
    })

    botaoSalvar := widget.NewButton("Salvar", func() {
        //Recebimento de dados para a estrutura livro
        t.SetCodigo(entradaCodigo.Text)
        t.SetTitulo(entradaTitulo.Text)
        t.SetAutor(entradaAutor.Text)
        t.SetEditora(entradaEditora.Text)
        t.SetAno(entradaAno.Text)
        //Validacao
        livro.ValidaDados(erro, t)
```

```
        if erro.GetErro() {
            dialog.ShowInformation("Erro", erro.GetMensagem(), janela)
            return
        } else {
            dialog.ShowInformation("Mensagem", "Dados inseridos com suces
so!", janela)
        }

    })

    botaoConsultar := widget.NewButton("Consultar", func() {
        livro.ValidaCodigo(erro, t)
        //Exibicao do Erro
        if erro.GetErro() {
            dialog.ShowInformation("Erro", erro.GetMensagem(), janela)
            return
        } else {
            entradaCodigo.SetText(t.GetCodigo())
            entradaTitulo.SetText(t.GetTitulo())
            entradaAutor.SetText(t.GetAutor())
            entradaEditora.SetText(t.GetEditora())
            entradaAno.SetText(t.GetAno())
        }

    })

    containerBotoes := container.New(layout.NewHBoxLayout(), botaoSalvar,
 botaoLimpar, botaoConsultar)
    form := widget.NewForm(
        widget.NewFormItem("Código:", entradaCodigo),
        widget.NewFormItem("Título: ", entradaTitulo),
        widget.NewFormItem("Autor: ", entradaAutor),
        widget.NewFormItem("Editora:", entradaEditora),
        widget.NewFormItem("Ano: ", entradaAno),
    )

    container := container.New(layout.NewVBoxLayout(), form, containerBot
oes)
    janela.SetContent(container)
    janela.Resize(fyne.NewSize(470, 250))
    janela.ShowAndRun()

}
```

Observe como funciona o botão CONSULTAR. OBS: Primeiramente, fez-se o cadastro do livro.


Programa Livraria
Código:
5
Título:
Autor:
Editora:
Ano:
Salvar
Limpar
Consultar


Programa Livraria
Código:
5
Título:
A vida de Go
Autor:
Golangueiro Golador
Editora:
Gol
Ano:
2005
Salvar
Limpar
Consultar

Chegou a hora de conversamos um pouco sobre o acesso ao banco de dados e a linguagem GO. Para darmos início a está etapa precisamos deixar claro que usaremos o sistema de gerenciamento de banco de dados MySQL.

6 UM POUCO SOBRE O MYSQL

Para quem não sabe o MySQL é um sistema de gerenciamento de banco de dados (SGBD), que utiliza a linguagem SQL (Linguagem de Consulta Estruturada, do inglês Structured Query Language) como interface. É atualmente um dos bancos de dados mais populares, com mais de 10 milhões de instalações pelo mundo.

Entre as suas principais características podemos citar:

É open source que se trata de um software já muito disseminado; É confiável e tem, comprovadamente, alta disponibilidade, inclusive para uso na web; Seu código é estável e também confiável; Tem uma grande comunidade de usuários e de administradores; Existe uma grande coleção de informações publicadas a seu respeito, na web e na mídia impressa.

Essas características conferem ao MySQL uma série de vantagens que o tornam ainda mais poderoso. Por ser bastante popular, por exemplo, existe uma enorme quantidade de informações online; incluindo toda a documentação do SGBD, tutoriais, fóruns de discussão e mailing lists. Com isso, é mais fácil obter suporte, tirar dúvidas e até mesmo aprender como utilizar o MySQL.

Outra grande vantagem do MySQL é o seu código aberto. Ele está disponível para o download de qualquer pessoa que se interessar, sem qualquer licença ou autorização e, inclusive, gratuitamente. No entanto, a versão gratuita, a Community Edition, só está disponível para uso não comercial. Apenas as versões pagas do MySQL podem ser usadas comercialmente.

MySQL	
LICENÇA	Open Source e gratuito para uso não comercial
SEGURANÇA	Recursos avançados de encriptação, MySQL Enterprise Firewall, ProxySQL data masking, MySQL Enterprise Audit Plugin e gerenciamento de senhas
PERFORMANCE	Todos os três SGBDs relacionais não diferem muito em relação à performance
ANALYTICS	Sem recursos de Analytics próprios
BACKUP	mysqlbackup
PARTICIONAMENTO	Padrão
ESCALABILIDADE	MySQL Enterprise Threadpool, MySQL Group Replication,
PRINCIPAIS MECANISMOS DE ARMAZENAMENTO	InnoDB(default), XtraDB, Federated

A imagem a seguir representa o modo como o MySQL funciona.

Estrutura básica de um modelo cliente-servidor. Um ou mais dispositivos (clientes) conectam com o servidor através de uma rede específica.

Cada cliente pode fazer a solicitação através de uma interface gráfica do usuário (IGU) em suas telas. E o servidor vai produzir o melhor resultado, desde que ambos entendam as instruções. Sem entrar muito nos méritos técnicos, os principais processos que acontecem em um ambiente MySQL são os mesmos.

- O mysql cria um banco de dados para armazenamento e manipulação de dados, definindo a relação de cada tabela;
- Clientes podem fazer solicitações digitando comandos sql específicos no mysql;
- A aplicação do servidor responde com a informação solicitada fazendo aparecer no cliente.

6.1 Usando a linguagem GO com MYSQL

Neste exemplo de CRUD em GO com MySQL, estamos fazendo um cadastro de livros, onde é feita a entrada de dados como: nome do livro, autor, editora e ano. Como podemos ver na foto, pode ser feito o cadastro do livro, alterar um cadastro, visualizar ou excluir.

Para realizar esse CRUD, o primeiro passo é definir a estrutura do livro.

```
package livro

type Livro struct {
 codigo string
 titulo string
 autor string
 editora string
 ano string
}
func (t *Livro) SetCodigo(codigo string) {
 t.codigo = codigo
}
func (t *Livro) SetTitulo(titulo string) {
 t.titulo = titulo
}
func (t *Livro) SetAutor(autor string) {
 t.autor = autor
}
func (t *Livro) SetEditora(editora string) {
 t.editora = editora
}
func (t *Livro) SetAno(ano string) {
```

```
 t. ano = ano
}
func (t *Livro) GetCodigo() string {
 return t.codigo
}
func (t *Livro) GetTitulo() string {
 return t.titulo
}
func (t *Livro) GetAutor() string {
 return t.autor
}
func (t *Livro) GetEditora() string {
 return t.editora
}
func (t *Livro) GetAno() string {
 return t.ano
}
```

Neste "*package* livro", temos criação, entrada e saída de dados, sendo que no "type Livro *struct*" são criadas as seguintes *strings*: codigo; titulo; autor; editora; e ano. Na entrada e saída de dados, temos o *get* e *set*. No *set*, as *strings* criadas são colocadas como valor de uma variável, onde tem o nome da *string* acompanhado do prefixo "t.". Já no *get*, ocorre o retorno dessas respectivas variáveis.

Agora fazemos a validação, para chamar o banco de dados.

```
package livro

import (
 "bdd/erro"
 "strconv"
)
func ValidaCodigo(e *erro.Erro, t *Livro, op string) {
 e.SetErroBool(false)
 if len(t.GetCodigo()) == 0 {
 e.SetErroString("O código é de preenchimento obrigatório!")
 return
 }
 if op == "c" {
     consultaUmLivro(t, e)
 } else {
     excluiUmLivro(t)
 }
}


func ValidaDados(e *erro.Erro, t *Livro, op string) {
```

```
 ValidaCodigo(e, t, op)
 if len(t.GetTitulo()) == 0 {
 e.SetErroString("O título é de preenchimento obrigatório!")
 return
 }
 if len(t.GetAutor()) == 0 {
 e.SetErroString("O autor é de preenchimento obrigatório!")
 return
 }
 if len(t.GetEditora()) == 0 {
 e.SetErroString("A editora é de preenchimento obrigatório!")
 return
 }
 if len(t.GetAno()) == 0 {
 e.SetErroString("O ano é de preenchimento obrigatório!")
 return
 }
 if _, erroConv := strconv.Atoi(t.GetAno()); erroConv != nil {
 e.SetErroString("O valor do ano deve ser numérico!")
 return
 } else if num, _ := strconv.Atoi(t.GetAno()); num <= 0 {
    e.SetErroString("O valor do ano deve ser positivo!")
    return
    }
if op == "i" {
    inseriUmLivro(t)
} else {
    atualizaUmLivro(t)
}
   }
```

Iniciamos esse código realizando funções das validações necessárias para que "*package* de livro" seja executado somente dentro das qualificações obrigatórias, onde, todas as funções são de preenchimento obrigatórios e o valor do ano deve ser númerico e positivo. Após serem realizadas as validações, chamamos o banco de dados, para que todas as informações sejam inclusas.

Agora vamos para o código do banco de dados.

```
package livro

import (
    "bdd/erro"
    "database/sql"
    _ "github.com/go-sql-driver/mysql"
)
```

```
/**
    Golang não possui membros estáticos. Portanto, como solução,
    preferimos encapsular os dados de conexão com o banco (a string de
    conexão do exemplo passado) neste método.
**/

func conecta() (db *sql.DB) {
    dbDriver := "mysql"
    dbUsuario := "root"
    dbSenha := "root"
    dbNome := "crudgo"

    db, err := sql.Open(dbDriver, dbUsuario+":"+dbSenha+"@/"+dbNome)
    if err != nil {
        panic(err.Error())
    }
    return db
}

func inseriUmLivro (livro *Livro) {
    db := conecta()
    sqlInserir, err := db.Prepare("INSERT INTO livro (codigo, titulo,
autor, editora, ano) VALUES (?, ?, ?, ?, ?)")
    if err != nil {
        panic(err.Error())
    }
    //Exec retorna algum erro pra ajudar?
    sqlInserir.Exec(livro.GetCodigo(), livro.GetTitulo(),
livro.GetAutor(), livro.GetEditora(), livro.GetAno())
    defer db.Close()
}

func excluiUmLivro (livro *Livro) {
    db := conecta()
    sqlDeletar, err := db.Prepare("DELETE FROM livro WHERE codigo=?")
    if err != nil {
        panic(err.Error())
    }
    sqlDeletar.Exec(livro.GetCodigo())
    defer db.Close()
}

func atualizaUmLivro (livro *Livro) {
    db := conecta()
    sqlInserir, err := db.Prepare("UPDATE livro SET titulo=?, autor=?,
editora=?, ano=? WHERE id=?")
    if err != nil {
        panic(err.Error())
    }
```

```
    sqlInserir.Exec(livro.GetTitulo(), livro.GetAutor(),
livro.GetEditora(), livro.GetAno())
    defer db.Close()
}

func consultaUmLivro (livro *Livro, e *erro.Erro) {
    var titulo, autor, editora, ano string
    db := conecta()
    row := db.QueryRow("SELECT titulo, autor, editora, ano FROM livro
where codigo = ?", livro.GetCodigo())
    err := row.Scan(&titulo, &autor, &editora, &ano);

    if err != nil {
        e.SetErroString("Livro não encontrado.")
    } else {
        livro.SetTitulo(titulo)
        livro.SetAutor(autor)
        livro.SetEditora(editora)
        livro.SetAno(ano)
    }
}
```

Antes de tudo fizemos as importações necessárias para o banco de dados. E, então, podemos fazer a conexão do nosso projeto com o mysql, banco de dados utilizado, definindo valores para usuário, senha, nome e driver.

Após isso, descrevemos as funções que queremos no banco de dados. Nesse caso criamos inseriUmLivro, excluiUmLivro, atualizaUmLivro e consultaUmLivro. Os nomes são autoexplicativos sobre o objetivo de cada uma delas, elas são responsáveis por, respectivamente, inserir, excluir, atualizar e consultar livros dentro do nosso banco de dados.

Por último, mas não menos importante, o main:

```
package main

import (
    "bdd/erro"
    "bdd/livro"

    "fyne.io/fyne/v2"
    "fyne.io/fyne/v2/app"
    "fyne.io/fyne/v2/container"
    "fyne.io/fyne/v2/dialog"
    "fyne.io/fyne/v2/layout"
    "fyne.io/fyne/v2/widget"
)
```

```
func main() {
    aplicativo := app.New()
    janela := aplicativo.NewWindow("Programa Livraria")
    entradaCodigo := widget.NewEntry()
    entradaTitulo := widget.NewEntry()
    entradaAutor := widget.NewEntry()
    entradaEditora := widget.NewEntry()
    entradaAno := widget.NewEntry()

    t := new(livro.Livro)
    erro := new(erro.Erro)

    botaoLimpar := widget.NewButton("Limpar", func() {
        entradaCodigo.SetText("")
        entradaTitulo.SetText("")
        entradaAutor.SetText("")
        entradaAno.SetText("")
        entradaEditora.SetText("")
        entradaCodigo.FocusGained()
    })

    botaoSalvar := widget.NewButton("Salvar", func() {
        //Recebimento de dados para a estrutura livro
        t.SetCodigo(entradaCodigo.Text)
        t.SetTitulo(entradaTitulo.Text)
        t.SetAutor(entradaAutor.Text)
        t.SetEditora(entradaEditora.Text)
        t.SetAno(entradaAno.Text)
        //Validacao
        livro.ValidaDados(erro, t, "i")
        if erro.GetErro() {
            dialog.ShowInformation("Erro", erro.GetMensagem(), janela)
            return
        } else {
            dialog.ShowInformation("Mensagem", "Dados inseridos com
sucesso!", janela)
        }

    })

    botaoConsultar := widget.NewButton("Consultar", func() {
        t.SetCodigo(entradaCodigo.Text)
        livro.ValidaCodigo(erro, t, "c")
        //Exibicao do Erro
        if erro.GetErro() {
            dialog.ShowInformation("Erro", erro.GetMensagem(), janela)
            return
        } else {
```

```
            entradaCodigo.SetText(t.GetCodigo())
            entradaTitulo.SetText(t.GetTitulo())
            entradaAutor.SetText(t.GetAutor())
            entradaEditora.SetText(t.GetEditora())
            entradaAno.SetText(t.GetAno())
        }

    })

    botaoExcluir := widget.NewButton("Excluir", func() {
        t.SetCodigo(entradaCodigo.Text)
        livro.ValidaCodigo(erro, t, "e")
        if erro.GetErro() {
            dialog.ShowInformation("Erro", erro.GetMensagem(), janela)
        } else {
            dialog.ShowInformation("Exclusão", "livro excluído!", janela)
        }
    })

    containerBotoes := container.New(layout.NewHBoxLayout(), botaoSalvar,
botaoLimpar, botaoConsultar, botaoExcluir)
    form := widget.NewForm(
        widget.NewFormItem("Código:", entradaCodigo),
        widget.NewFormItem("Título: ", entradaTitulo),
        widget.NewFormItem("Autor: ", entradaAutor),
        widget.NewFormItem("Editora:", entradaEditora),
        widget.NewFormItem("Ano: ", entradaAno),
    )

    container := container.New(layout.NewVBoxLayout(), form,
containerBotoes)
    janela.SetContent(container)
    janela.Resize(fyne.NewSize(470, 250))
    janela.ShowAndRun()

}
```

No "package main" temos a criação da interface gráfica do programa feito com fyne, onde foi criado os campos, local em que o usuário coloca os dados, os botões de limpar, salvar, consultar, excluir e alterar, como foi visto na foto anterior.

7 A CONSTRUÇÃO DA API NA LINGUAGEM GO

Como último desafio dessa apostila, temos a missão de consumir uma API através da linguagem GO.

Uma API nada mais é do que um ponto de comunicação entre plataformas, em que há uma série de protocolos e padrões restringindo como dois ou mais softwares podem interagir. Nesse caso, iremos utilizar a API "ViaCep" e, através dela, podemos criar um formulário que peça um número de CEP e retornar o endereço correspondente.

Basicamente, o serviço funciona da seguinte forma:

Através da URL viacep.com.br/ws/01001000/json/, é informado um CEP no espaço em laranja, que será recebido pelo serviço. Ao capturar essa informação, ele irá verificar em sua base de dados se há um endereço condizente a esse número e, através do método HTTP GET, irá retornar uma resposta. Caso ele encontre algo, teremos uma resposta para a requisição feita e será retornado o código 200 para a operação feita. Caso contrário, obteremos dados nulos e o código 400 de resposta.

Em casos de sucesso, a resposta será retornada em JSON, um formato que se popularizou principalmente no desenvolvimento WEB, em que a troca de informação entre servidores é constante. Sua estrutura é similar à utilizada com objetos e structs, além de ser leve e empregado em diversos sistemas.

7.1 O código final

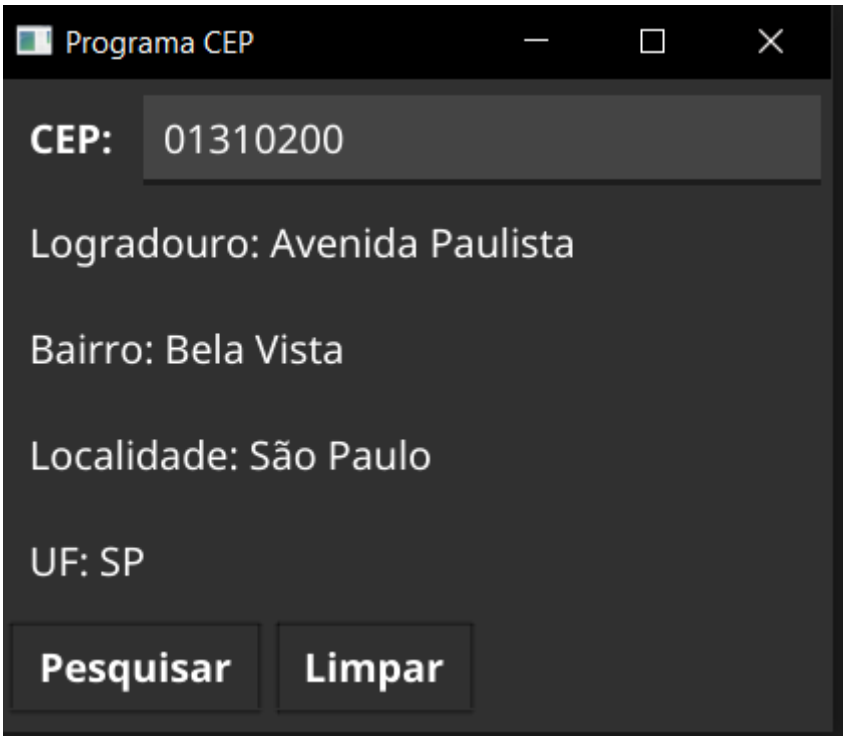
Acompanhe o código a seguir:

```
package main

import (
    "encoding/json"
    "io/ioutil"
    "net/http"
    "regexp"

    "fyne.io/fyne/v2"
    "fyne.io/fyne/v2/app"
    "fyne.io/fyne/v2/container"
    "fyne.io/fyne/v2/dialog"
    "fyne.io/fyne/v2/layout"
    "fyne.io/fyne/v2/widget"
)

/*
    Criação da estrutura que irá receber os dados.
    As tags "json:" servem para mapear o nome dos campos com
    os títulos em Json, recebidos da api.
```

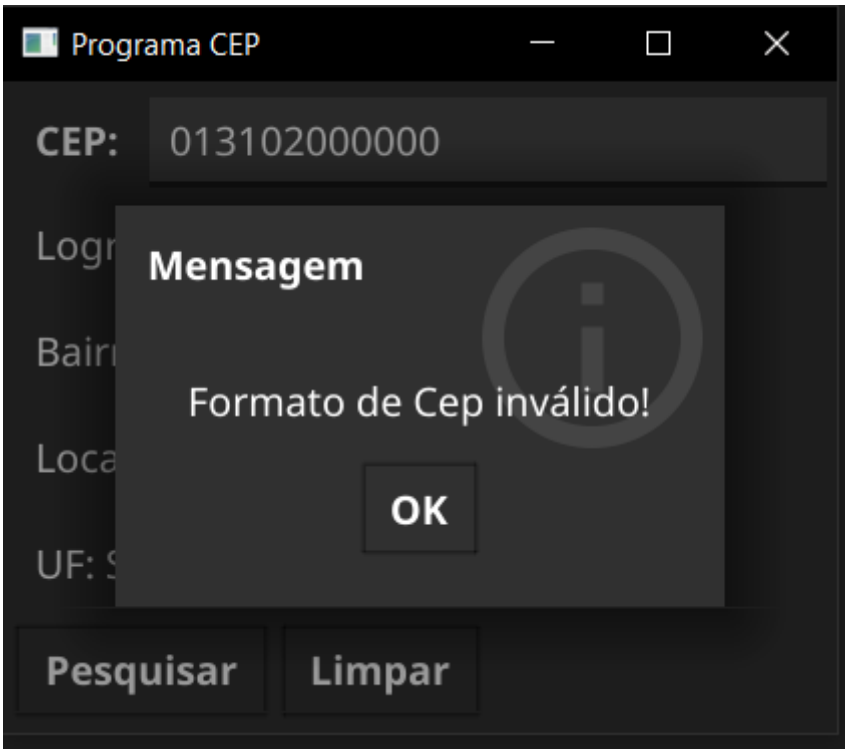
```
    Obs: todos eles estão em letra maiúscula para que códigos externos
    possam manipulá-los, como a função json.Unmarshal
*/
type Cep struct {
    Cep        string `json:"cep"`
    Logradouro string `json:"logradouro"`
    Bairro     string `json:"bairro"`
    Localidade string `json:"localidade"`
    Uf         string `json:"uf"`
}

/*
    Função responsável pela validação do Cep, através de um recurso
    chamado expressão regular, que verifica se uma string
    segue um determinado padrão de caracteres.
    Nesse caso, é aceito o padrão 000000-000 ou 00000000
*/
func validarCep(cep string) bool {
    match, _ := regexp.MatchString("(^[0-9]{5})-?([0-9]{3}$)", cep)
    return match
}

func main() {

    aplicativo := app.New()
    janela := aplicativo.NewWindow("Programa CEP")
    entradaCep := widget.NewEntry()
    lblLogradouro := widget.NewLabel("Logradouro...")
    lblBairro := widget.NewLabel("Bairro...")
    lblLocalidade := widget.NewLabel("Localidade...")
    lblUf := widget.NewLabel("Uf...")

    botaoPesquisar := widget.NewButton("Pesquisar", func() {
        cep := new(Cep)
        cep.Cep = entradaCep.Text
        //Url utilizada para requisitar os dados pelo método GET
        url := "https://viacep.com.br/ws/" + cep.Cep + "/json/unicode"
        respostaByte, _ := http.Get(url)
        //Dados recebidos em byte no body da requisição, lidos em string
        respostaString, _ := ioutil.ReadAll(respostaByte.Body)
        //String formatada em json convertida para o struct de Cep
        json.Unmarshal(respostaString, &cep)

        if !validarCep(cep.Cep) {
            dialog.ShowInformation("Mensagem", "Formato de Cep
inválido!", janela)
            return
        }
```

```
        if cep.Bairro == "" {
            dialog.ShowInformation("Mensagem", "Cep não encontrado!",
janela)
            return
        }
        lblLogradouro.SetText("Logradouro: " + cep.Logradouro)
        lblBairro.SetText("Bairro: " + cep.Bairro)
        lblLocalidade.SetText("Localidade: " + cep.Localidade)
        lblUf.SetText("UF: " + cep.Uf)

    })

    botaoLimpar := widget.NewButton("Limpar", func() {
        entradaCep.SetText("")
        lblLogradouro.SetText("Logradouro...")
        lblBairro.SetText("Bairro...")
        lblLocalidade.SetText("Localidade...")
        lblUf.SetText("Uf...")
        entradaCep.FocusGained()
    })

    containerBotoes := container.New(layout.NewHBoxLayout(),
botaoPesquisar, botaoLimpar)
    form := widget.NewForm(
        widget.NewFormItem("CEP:", entradaCep),
    )
    container := container.New(layout.NewVBoxLayout(),
        form,
        lblLogradouro,
        lblBairro,
        lblLocalidade,
        lblUf,
        containerBotoes)
    janela.SetContent(container)
    janela.Resize(fyne.NewSize(320, 250))
    janela.ShowAndRun()

}
```

Agora veja a execução caso o CEP seja valído:



Ou caso CEP seja invalído:



Nós, Brandon; Camilly; Giovanna; Laiza; Nicoly; Nathália; Paloma; Samuel agradecemos você que acompanhou nossa pesquisa, que se aprofundou com a gente em um novo desafio, em uma nova linguagem, você que começou do zero e embarcou nessa insana experiência conosco.

Ficamos gratos em saber que nosso trabalho de alguma forma ajudou você leitor, estudante ou curioso a tirar suas dúvidas e pedimos desculpas caso tenhamos te deixado com a pulga atrás da orelha em algum momento.

Nosso trabalho foi realizado com muito carinho para você aprender de uma forma menos estressante, com o layout fora do normal para você se sentir confortável e talvez, até mesmo, esquecer que só está lendo porque algum professor está te obrigando a fazer isso.

A linguagem do google (GOLANG), não é uma linguagem fácil de aprender, muito menos de encontra registros que nos ajude, por isso caso você esteja desesperado por não encontrar o que deseja não desanime, se nós conseguimos você também vai conseguir, apenas foque, ligue para o papa se preciso for. NÓS ACREDITAMOS EM VOCÊ.

Obrigada por ler até aqui!

9 REFERÊNCIAS

- DA SILVA, Carlos. **5 dicas para se tornar um desenvolvedor de Google Go**. Código Fonte. 2017. Disponível em: https://www.codigofonte.com.br/artigos/5-dicas-para-se-tornar-um-desenvolvedor-de-google-go.
- Qual é o melhor IDE para desenvolver em Golang. ICHI.PRO. Disponível em: https://ichi.pro/pt/qual-e-o-melhor-ide-para-desenvolver-em-golang-167668448086404.
- SOUZA, Evandro. **GoLang — Simplificando a complexidade**. Medium. 2018. Disponível em: https://medium.com/trainingcenter/golang-d94e16d4b383.
- Marla Vitória. **O que é a linguagem de programação Go?**. Locaweb. 2018. Disponível em: https://blog.locaweb.com.br/temas/codigo-aberto/o-que-e-a-linguagem-de-programacao-go/#:~:text=A%20linguagem%20Go%20%C3%A9%20open,e%20na%20hist%C3%B3ria%20da%20computa%C3%A7%C3%A3o.
- CARVALHO, Suelen Goulart. **Go Lang**: A linguagem do Google. 2015. Disponível em: https://www.ime.usp.br/~amaris/mac-5742/reports/GoLang.pdf
- SEYFAN, Gigi. **VAMOS LA: PROGRAMAÇÃO ORIENTADA A OBJETOS EM GOLANG.** Envato Tuts+. 2016. Disponível em: https://code.tutsplus.com/pt/tutorials/lets-go-object-oriented-programming-in-golang--cms-26540.
- ONE, Host. **Qual banco de dados usar: MySQL, MariaDB ou Percona?** Host One. 2019. Disponível em: https://blog.hostone.com.br/mysql-mariadb-ou-percona/
- PACIEVITCH, Yuri. **MySQL.** InfoEscola. 2011. Disponível em: https://www.infoescola.com/informatica/mysql/
- OFICINA, Redação. **O que é MySQL.** Oficina da Net. 2010. Disponível: https://www.oficinadanet.com.br/artigo/2227/mysql_-_o_que_e