Da história à aplicação, em uma simples leitura.

PYTHON, UMA FORMA SIMPLES DE APRENDER

#DESENVOLVIDO POR
PRINT('BRUNA COSTA PINHEIRO DE SOUZA')
PRINT('JONATHAN MACEDO CASTRO')
PRINT('KETHLYN DA SILVA LOPES')
PRINT('LUCAS SANTOS COVA RAMOS')
PRINT('MARIA EDUARDA PUPO BONIFACIO')
PRINT('MITSUO LUAN DE ANDRADE MIYAZATO')
PRINT('PEDRO CESAR ANTUNES DE AMIGO')
PRINT('THAYNÁ DE ALMEIDA MELO ARAUJO')

Sumário

Capítulo 1 - Introdução a Python

1 - História

A linguagem Python foi criada nos anos 80, por Guido Van Hossum. O desenvolvedor da linguagem ABC trabalhava, no Instituto de Pesquisa Nacional para Matemática e Ciência da Computação (CWI), em um sistema operacional chamado AMOEBA.

Ao criar a linguagem Python, Guido pretendia desenvolver uma linguagem intuitiva e de fácil compreensão, uma vez que a linguagem C se mostrava complexa no desenvolvimento de determinados softwares e era necessário a ajuda de programadores experientes para compreender certos programas.

O nome da linguagem foi inspirado no programa Monty Python's Flying Circus, o programa favorito de Van Hossum.

A serpente píton só foi associada à linguagem no primeiro livro de programação em Python, produzido pela editora O'Reilly. A editora tem o costume de colocar um animal na capa de seus livros e escolheu a serpente píton para fazer parte da capa da linguagem Python.

2 – Características principais da linguagem

Python é uma linguagem de programação imperativa, interpretada e de alto nível com tipagem forte e dinâmica. É considerada a linguagem de múltiplos paradigmas, pois aceita diferentes formas de programação. Características funcionais principais de Python são: lambda (função sem nome); map (aplica uma função a cada item de uma lista); reduce (faz um somatório de uma lista); filter: (aplica uma condição em cada item da lista); zip (agrupa elementos entre listas).

Um dos principais recursos do Python é sua notação especial. E leia o código de acordo. Lá não existem separadores convencionais, que consistem em chaves e são usados, por exemplo, em linguagens como C, C ++, C # e Java. O ponto-e-vírgula usado como final de um comando nas linguagens, é opcional em Python. Apenas essas duas funções distinguem claramente o Python dos outros em termos de legibilidade e fonte. Outro recurso interessante do Python é que qualquer pessoa pode estendê-lo facilmente. As extensões de suporte podem ser escritas em qualquer idioma. As extensões podem adicionar funções, variáveis e tipos de objetos ao interpretador Python. Isso, junto com sua funcionalidade e implantação de código

aberto, justifica sua escolha por muitas organizações governamentais. É preferível que eles não usem tecnologia proprietária sobre a qual não tenham controle total. Python é uma solução que contém código-fonte aberto e pode ser estendida para vários fins.

Outra característica marcante em Python é sua aprendizagem marcante nos alunos que estão iniciando, ela é uma linguagem mais simples tendo assim uma possibilidade de entendimento e ensinamento melhores para quem está em busca de iniciar na programação, e aos poucos ele vai aprendendo e aperfeiçoando sua técnica em Python para obter saídas mais complexas em termos de processamento de dados.

Não adianta que uma linguagem seja fácil de aprender se não for procurado no mercado de trabalho. Algumas linguagens projetadas especificamente para fins de ensino, como Pascal e Basic, nunca foram totalmente aceitos no mercado e tiveram que se adaptar por algum tempo para contornar idiomas que são difíceis de aprender, mas muito mais poderosos, como C + + e Java, com a introdução de interessantes frameworks e suporte a linguagens em diversos servidores web, o Python tem se destacado nos últimos anos com tamanha relevância e tem se destacado no mercado, com Python, pode-se programar essencialmente em qualquer sistema.

Essa linguagem ela também é intuitiva e de fácil manutenção pode ser usado no sistema operacional Windows ou Linux, contendo também uma vasta biblioteca padrão. Esta biblioteca fornece recursos para executar as tarefas mais importantes para o desenvolvimento de várias tarefas. O interpretador Python é escrito nas linguagens C e C ++ para que possa ser portado para qualquer plataforma que possua compiladores da linguagem, levando em consideração que C ++ é a linguagem mais utilizada e a base para a prática de todas as informações tecnologia de formulários. Usada em várias aplicações: Web, desktop, sistemas embarcados, sistemas 3D.

3 - O que é uma IDE e qual utilizar?

Um IDE é um pacote de software que consolida as ferramentas básicas necessárias para escrever e testar softwares. Basicamente um IDE, ou ambiente de desenvolvimento integrado, é um software que une ferramentas de desenvolvimento em uma única interface gráfica do usuário (GUI). A seguir, mostraremos alguns IDEs possíveis de trabalhar com a linguagem Python.

VISUAL STUDIO

Visual Studio foi lançado em fevereiro de 2002, entretanto sua versão beta foi criada ano anterior da versão final. Esse IDE é uma das mais populares no mercado dos programadores, de acordo com pesquisa do *Stack Overflow*, não somente na linguagem Python como em outras propriedades que o software proporciona.

A Microsoft criou esse IDE com o intuito de fornecer recursos para o programador, além da criação de códigos, que depois serão compilados e executados, na linguagem Python há alternativa de fazer desenvolvimentos *WEB* – apesar que anos depois a própria Microsoft criou um IDE somente para esse serviço –, também é possível criar códigos em C# ou C++. O Visual Studio pode-se encontrar em três formatos diferentes, a *Community*, *Enterprise* e *Professional*; sendo suportado em dois Sistemas Operacionais: Windows e macOS.

PYCHARM

Lançado em fevereiro de 2010 pela empresa JetBrains, o PyCharm é um ótimo IDE para desenvolvedores em Python. Sendo desenvolvido para multiplataforma, como Linux, macOS, e versões do Windows, conta com a *Community Edition*, lançada sob a licença Apashe, que permite com que seus usuários utilizem o software para qualquer finalidade, sendo possível a modificação e distribuição do mesmo sem a necessidade de pagamento de qualquer taxa. Além dessa, existe também a *Professional Edition*, lançada sob uma licença proprietária conta com recursos extras.

Atualmente oferece aos usuários uma série de recursos, como seu sistema de multitecnologias, que torna possível o suporte para JavaScript, TypeScript, SQL, HTML/CSS entre outras, *Frameworks* de desenvolvimento *WEB*, assistência inteligente, recursos para desenvolvimento remoto entre outros.

VISUAL STUDIO CODE

Lançado em 2015 pela Microsoft, o Visual Studio Code, ou VSCode, como é comumente conhecido, é uma alternativa viável para programar a linguagem Python. A sua criação teve como foco principal a edição de código para aplicações *WEB*, entretanto, baixando extensões é possível fazer a excursão em Python. Existe a possiblidade de fazer os códigos do VSCode em três Sistemas Operacionais distintos são: Windows; Linux; macOS.

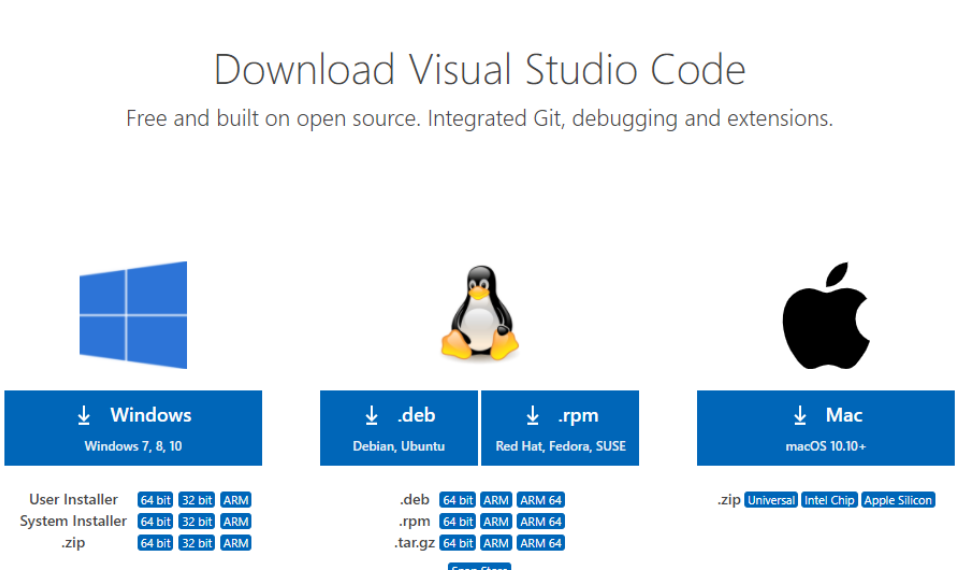
Existe a comparação com o Visual Studio, também lançado pela Microsoft, porém o VSCode tem um formato menor para instalação, é um software novo no mercado de programadores e não há uma variedade de conteúdos que o Visual Studio fornece para ser usado de outras formas, como usar a linguagem C#.

IDE ESCOLHIDA

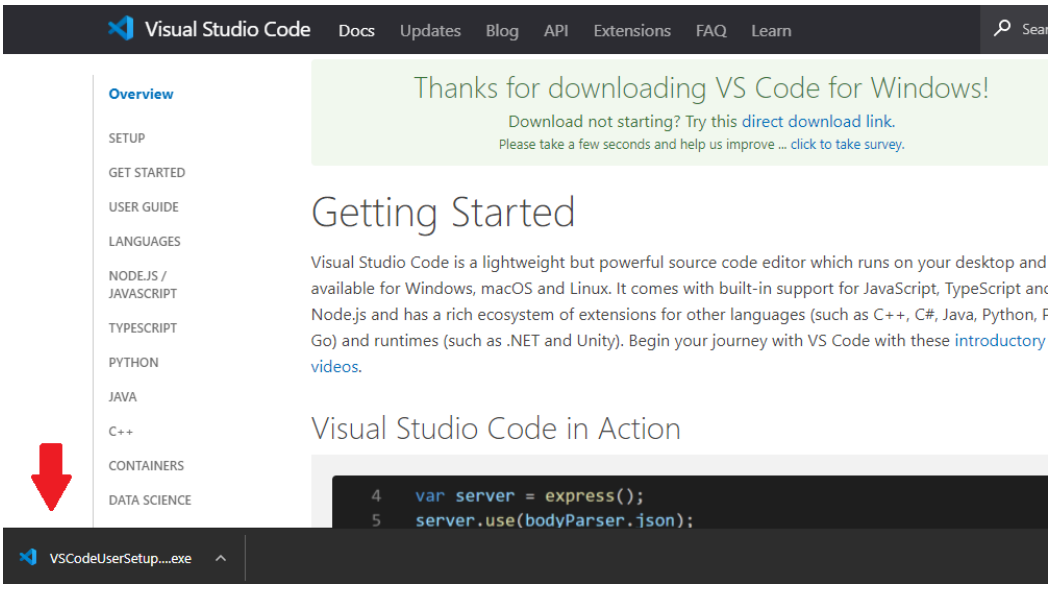
O grupo optou por utilizar o Visual Studio Code, por ser o mais recomendado pelos usuários e estarmos em uma instituição que utiliza uma IDE bem parecida – Visual Studio - em seu primeiro e segundo ano, assim criando uma familiaridade com a mesma.

4 - Processo de download e instalação da linguagem e da IDE - Baixando o Visual Studio Code

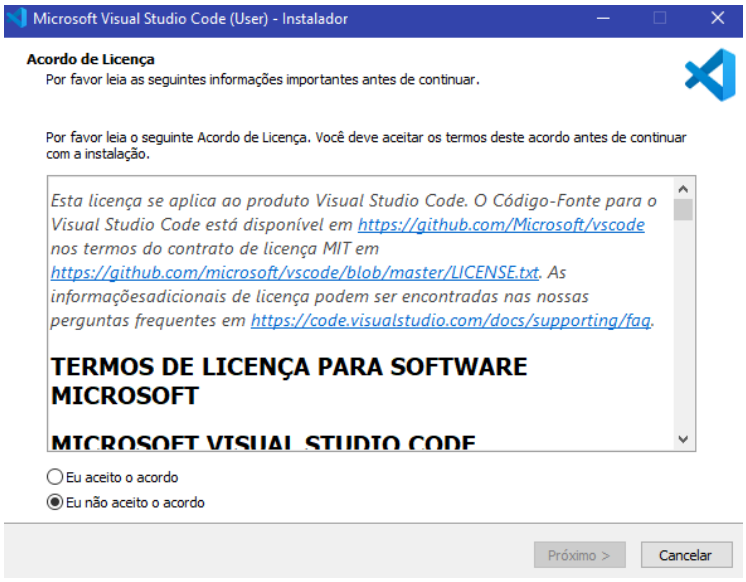
Inicialmente, iremos baixar o Visual Studio Code, para isso, basta acessar o endereço "https://code.visualstudio.com/download". Nesse site, obteremos os links para baixar o Code:



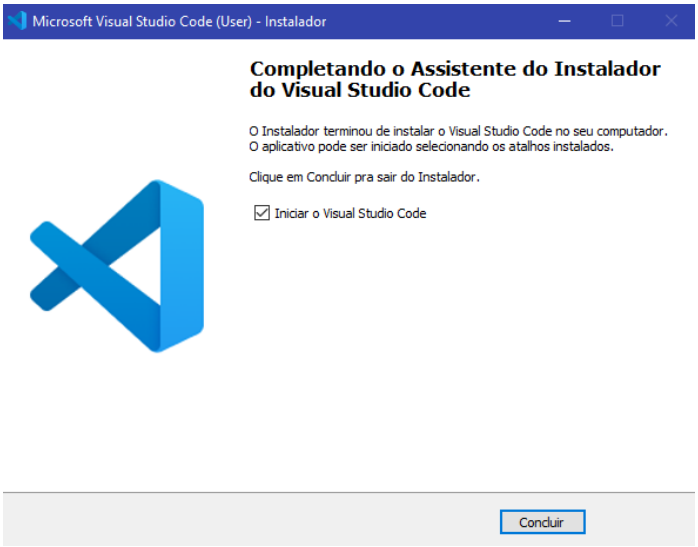
Deve-se selecionar a opção mais adequada para as configurações de seu computador. Neste caso, utilizaremos o link do Windows e selecionaremos “User Installer - 64 bit”. Ao clicar, o arquivo já será baixado em seu computador.



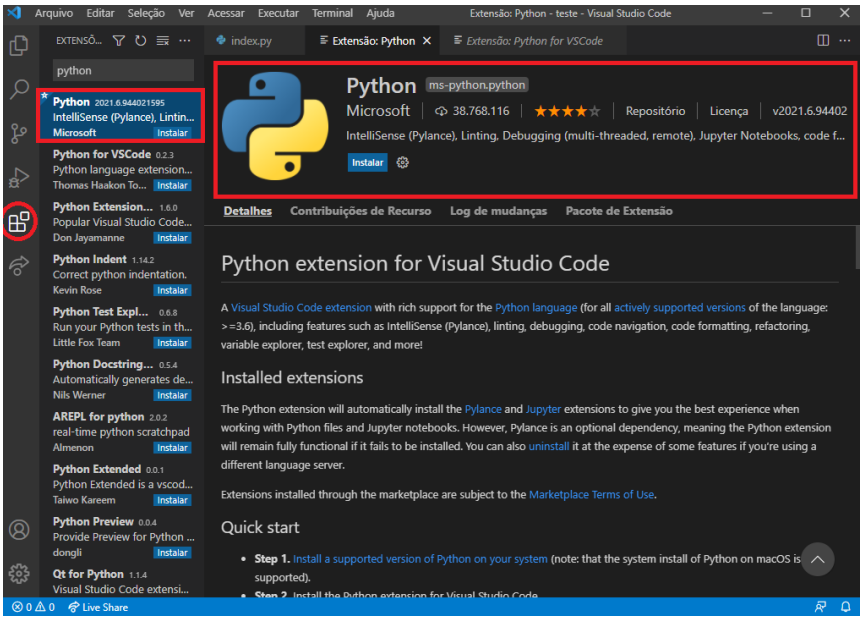
Posteriormente, clica-se no arquivo para poder executá-lo, assim, o instalador do Visual Studio Code irá ser inicializado. Segue-se as instruções iniciais de instalação, configura-se o local e os atalhos do Software.



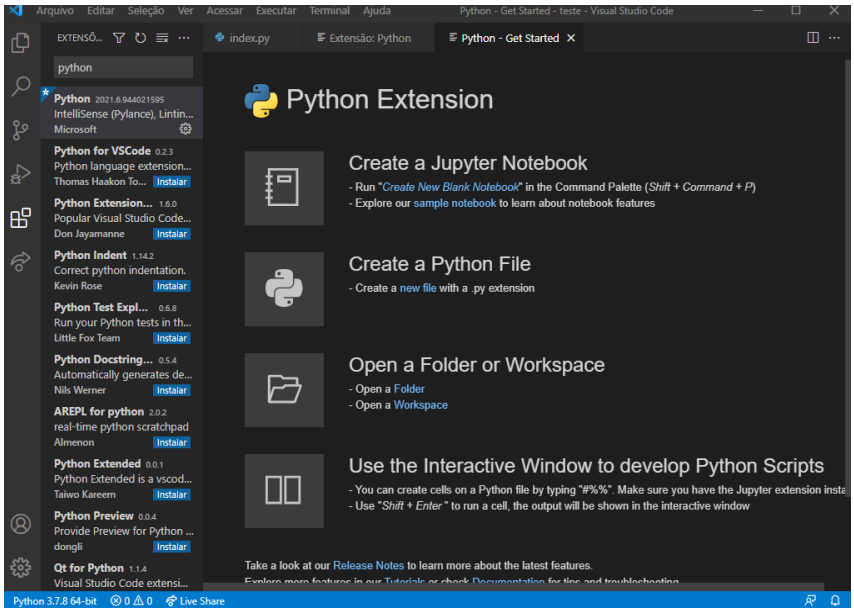
Após isso, o Microsoft Visual Studio Code para o usuário já está instalado e pronto para ser usado.

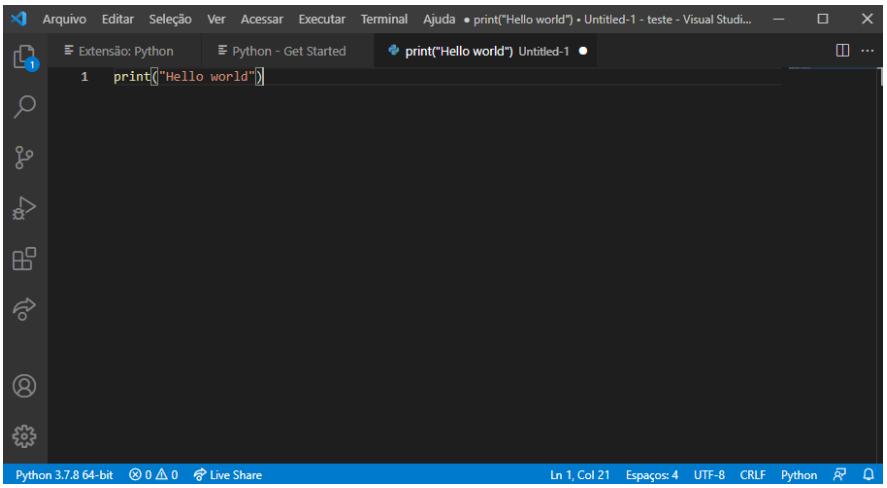


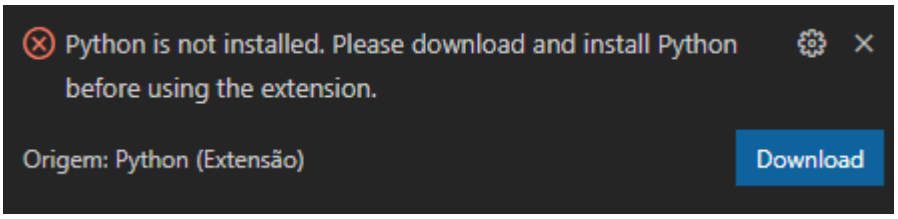
Agora, já com a instalação do Code, iremos instalar a extensão para que seja possível a utilização da linguagem Python. Deste modo, no Code, clica-se em “Extensões” localizando no canto esquerdo, pesquisa-se por “Python”, e procura-se por essa opção (marcada por uma estrela). Com isso, clica-se em “Instalar”:



Depois da instalação da extensão ser concluída, abrirá uma nova aba no Code. Ao clicar em “Create a Python File”, já será possível codificar a partir de um novo arquivo na linguagem Python.



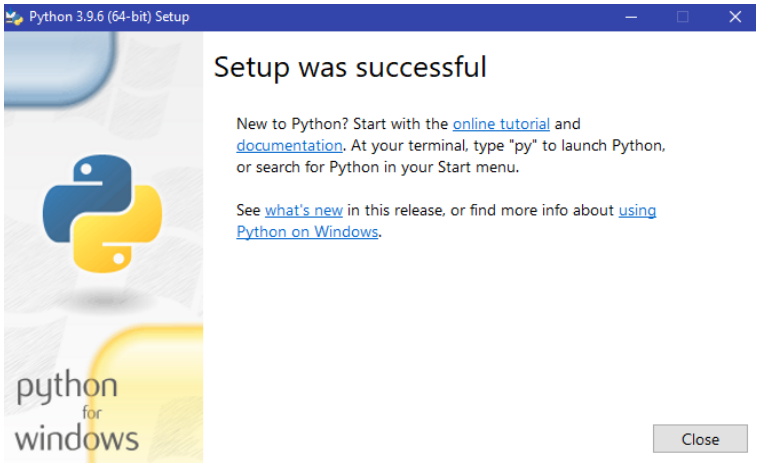




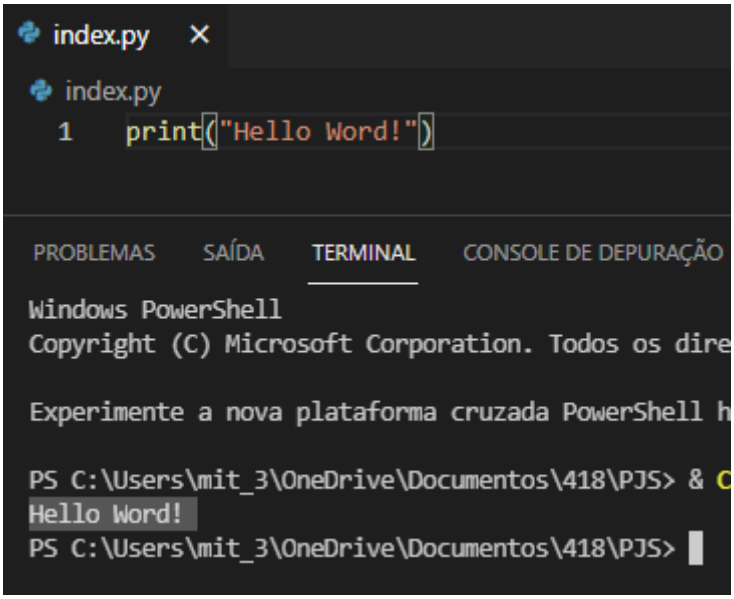
Apesar de ser possível criar arquivos Python (.py), não será possível executá-lo no Visual Studio Code. Você receberá a mensagem acima.

Para solucionar esse problema, basta clicar em “Download” que você será encaminhado para o endereço de site ”https://www.python.org/downloads/“. Lá, é possível instalar a linguagem Python. Cliquemos em “Download Python” que um arquivo será baixado, após isso, basta executá-lo que uma guia para instalar a linguagem em seu computador aparecerá.

Posteriormente, clique em “Install Now” que o instalador completará a instalação do Python. Caso a instalação seja um sucesso, aparecerá a seguinte mensagem:



E assim, já será possível codificar em Python e executá-lo no Visual Studio Code sem problemas.



Capítulo 2 – Apresentando a programação em Python

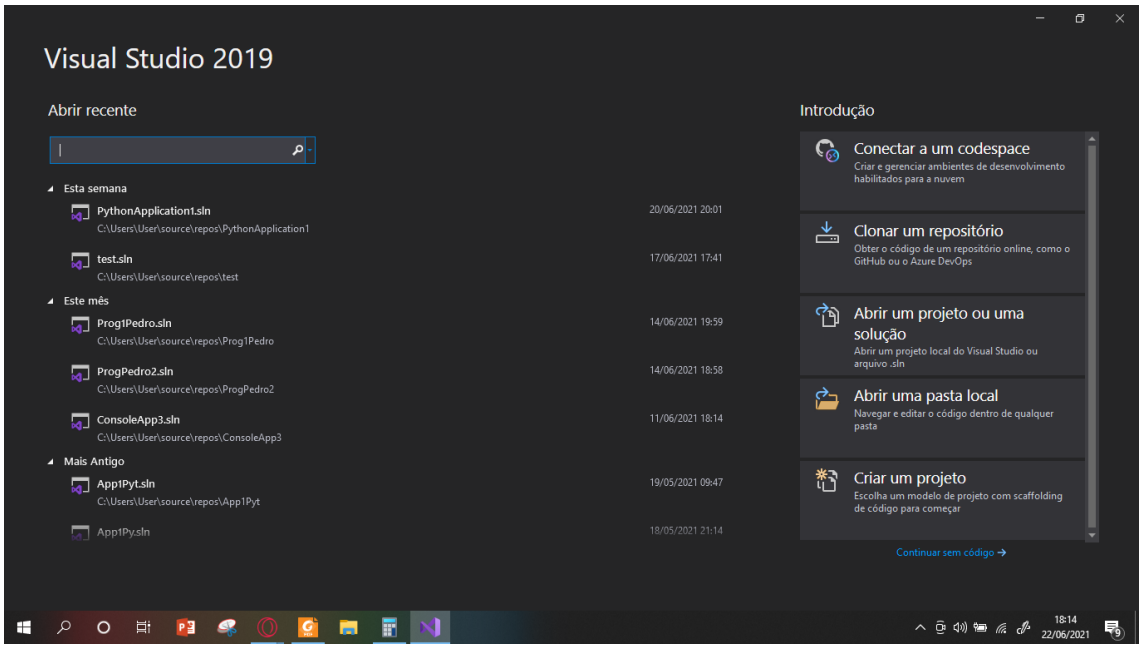
1 – Exercícios Resolvidos, entendendo na pratica.

A seguir, veremos 16 exercícios resolvidos na linguagem Python, assim demonstrando na pratica conceitos que a linguagem pode lhe oferecer.

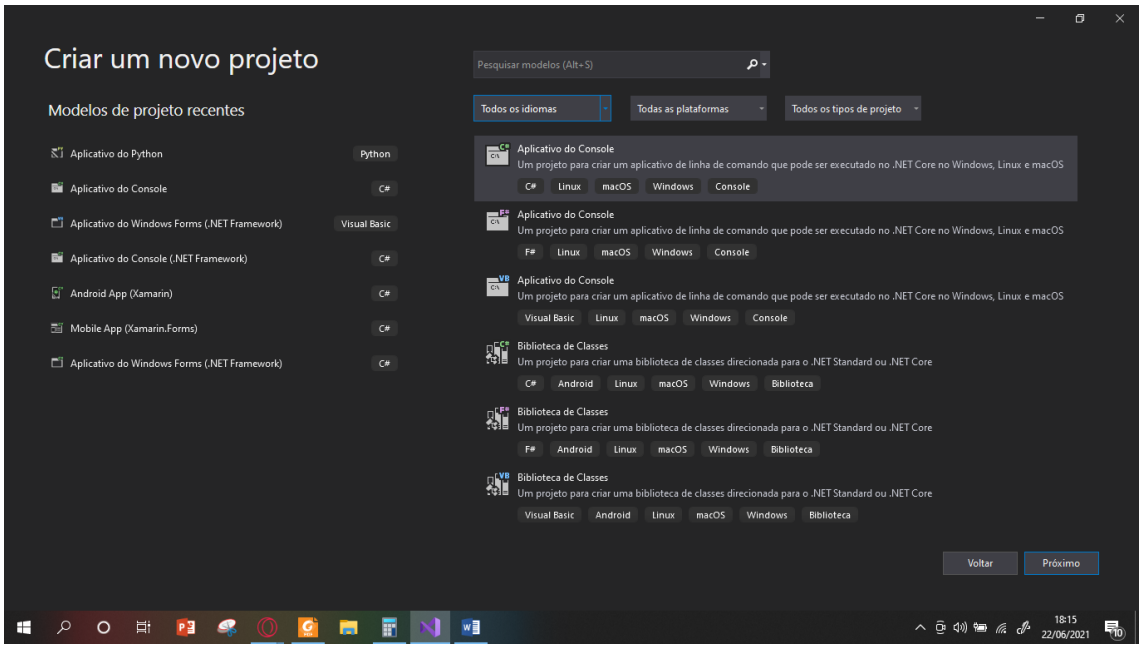
1.0 – Dando os primeiros passos

Antes de desenvolver qualquer aplicação, é necessário ter ao menos um conhecimento mínimo para conseguir se localizar dentro da IDE (Ambiente de Desenvolvimento Integrado).

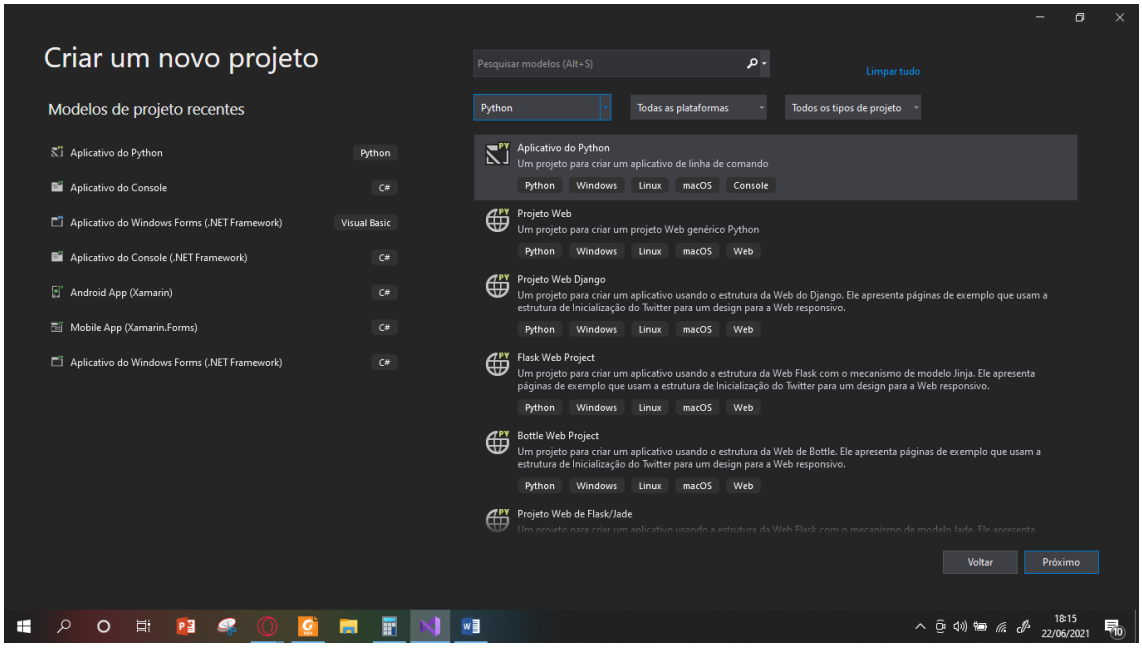
Ao abrir o Visual Studio 2019, o desenvolvedor se depara com esta tela inicial, em seguida, deverá clicar na opção “Criar um projeto”, que está localizada ao inferior do canto esquerdo da tela.



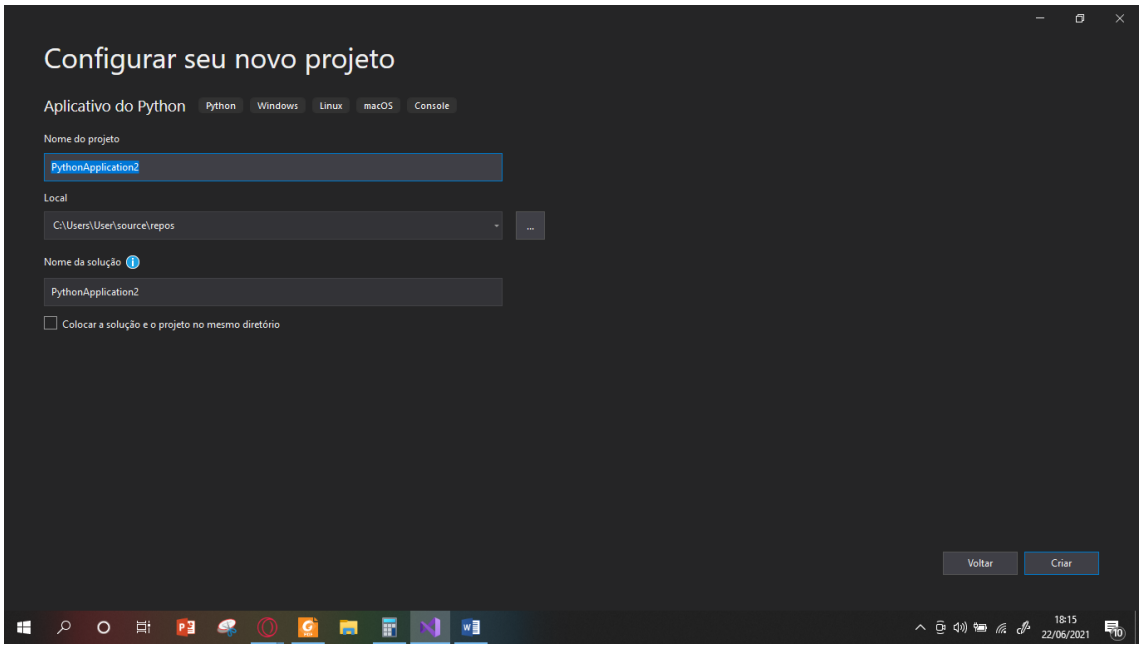
Ao seguir o passo anterior, o programador será direcionado à esta tela de busca, onde na opção em que se encontra “Todos os idiomas” irá pesquisar por “Python”.



Ao filtrar as opções oferecidas para a linguagem Python, deverá selecionar a opção "Aplicativo do Python".



Por fim, o desenvolvedor será conduzido para esta tela, onde poderá alterar o nome do projeto e o local em que ele ficará salvo.



Após ter feito este processo, será direcionado para o ambiente no qual poderá desenvolver suas aplicações na linguagem Python.

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

Separando a aplicação em três passos: Entrada de dados, operação aritmética e saída de dados.

Passo 1: Entrada de dados

```
#Entrada de dados
base = int(input('Digite o valor da base: '))
altura = int(input('Digite o valor da altura: '))
```

Neste primeiro passo, serão solicitados ao usuário os valores da base e da altura do triângulo, que serão armazenados respectivamente nas variáveis denominadas como *base* e *altura*.

Posteriormente, é declarado que a entrada de dados é do tipo *int*, ou seja, só aceita a entrada de números inteiros como, por exemplo, 1, 5 -12. Na linguagem Python é possível utilizar:

Tipo	Descrição	Exemplo

str	String (Texto)	'Olá mundo'
bool	Valor Booleano	True ou False
int	Número Inteiro	2, 8, -15
long	Número Inteiro de Valor Arbitrário	5416541658
float	Número Real	5.0, 7.2, -12.63
list	Lista	[1, 4, 10]

Por último, é utilizado o comando input, que serve para ler o que o usuário digitar no prompt.

Passo 2: Operação Aritmética

```
#Operação Aritmética
area = (base*altura)/2
```

Neste segundo passo, serão utilizados conceitos básicos da matemática para calcular a área do triangulo. Nesta etapa, é declarada a variável *area*, que vai armazenar o valor da operação. A linguagem Python tem os seguintes operadores aritméticos:

Operador	Descrição	Exemplo
+	Adição	2 + 3
-	Subtração	2 - 2
*	Multiplicação	2 * 4
/	Divisão	10 / 2
//	Divisão Inteira	4 // 3
**	Exponenciação	2 ** 100
%	Resto da Divisão	10 % 2

Passo 3: Saída de dados

```
#Saída de dados
print('A área do triangulo é: {}'.format(area))
```

Neste passo, será apresentado ao usuário o resultado da operação. É utilizada a função *print*, que apresenta ao usuário os dados de maneira legível. Em seguida é utilizado o método *format()*, que serve criar uma string que contém campos entre chaves e são substituídos pelos

argumentos do format, neste caso, o argumento foi a variável *area*, que continha o resultado da operação.

1.2 – Crie um programa que vai calcular o seno e cosseno de um ângulo digitado pelo usuário. Vejamos como fica a linha de código logo abaixo:

```
#Importação da Biblioteca Math
import math

#Declaração de variável
grau = float(input('Digite o ângulo:'))

#Cálculo do seno e cosseno
seno = math.sin(grau)
cosseno = math.cos(grau)

#Exibição para o usuário
print('Cosseno do ângulo desejado: {}' .format(cosseno))
print('Seno do ângulo desejado: {}' .format(seno))
```

Nas linguagens C# e Java, a biblioteca **math** não precisamos importar ela no código como é visto, logo abaixo, na linguagem Python, ou seja, nelas só precisamos colocar o comando **math.** e ele já vai mostrar todos os comandos que possíveis de serem executados pela biblioteca.

```
#Importação da Biblioteca Math
import math
```

Nas próximas linhas é observado a criação da variável para armazenar o valor que o usuário vai digitar e os cálculos desse número. Usamos somente dois comandos da biblioteca de Matemática, seno e cosseno, porém existe o comando, caso queira, para calcular a tangente que seria **math.tan()**.

```
#Declaração de variável
grau = float(input('Digite o ângulo:'))

#Cálculo do seno e cosseno
seno = math.sin(grau)
cosseno = math.cos(grau)
```

Por fim, para exibir as variáveis que armazenaram os resultados das operações, como já é de costume, usamos **print(‘’)**, depois utilizamos **{** e **}** para decidir o local onde será mostrado o valor e o comando **.format(variável)** para finalizar a linha de código.

```
#Exibição para o usuário
print('Cosseno do ângulo desejado: {}' .format(cosseno))
print('Seno do ângulo desejado: {}' .format(seno))
```

Caso queira ser um programador que utiliza poucas linhas de código, existe uma alternativa para executar o que foi pedido pelo enunciado. Contudo, a linha de apresentação é diferente do que estamos acostumados.

```
print(f'Cosseno do ângulo desejado: {math.cos(grau): .2f}')
print(f'Seno do ângulo desejado: {math.sin(grau): .2f}')
```

Invés de usarmos **.format(variável)** no print, podemos simplificar usar a letra **f** ou **F** antes das aspas simples, também é possível observar que dentro das chaves já está sendo efetuado o cálculo do cosseno e seno. Logo após a execução da biblioteca math, utilizamos o comando para exibir somente duas casas decimais.

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

Separando a aplicação em três passos: Entrada de dados, operação aritmética e verificação da situação do aluno.

Passo 1: Entrada de dados

```python
#Entrada de dados
nota1 = float(input('Nota do 1º bimestre: '))
nota2 = float(input('Nota do 2º bimestre: '))
nota3 = float(input('Nota do 3º bimestre: '))
nota4 = float(input('Nota do 4º bimestre: '))
```

Neste primeiro passo, serão solicitados ao usuário os valores das notas dos 4 bimestres, que serão armazenados respectivamente nas variáveis denominadas *nota1*, *nota2* *nota3* e *nota 4*.

Passo 2: Operação Aritmética

```python
#Operação aritmética
media = (nota1+nota2+nota3+nota4)/4
print("Sua média é {}" .format(media))
```

Neste segundo passo, serão utilizados conceitos básicos da matemática para calcular a média aritmética entre as 4 notas. Nesta etapa, é declarada a variável *media*, que vai armazenar o valor da operação. Posteriormente através na função *print* e do método *format()*, o resultado é apresentado ao usuário.

Passo 3: Verificação da situação do aluno

```python
#Verificação da situação do aluno
if media >= 6:
    print('Você está aprovado!')
elif media < 3:
    print('Você está reprovado!')
else:
    print('Você está de exame')
```

Neste passo será verificado, por meio da estrutura if-elif -else, a situação do aluno. O if é uma estrutura condicional, que permite avaliar determinada expressão e de acordo com o resultado, executar determinada ação. Caso exista mais de uma condição alternativa que precise ser verificada, deve-se utilizar o elif, como mostra no exemplo acima. Por último, o else é utilizado para casos onde nenhuma das condições postas anteriormente são satisfeitas.

As condições para comparação podem utilizar de alguns operadores como:

Operador	Descrição	Exemplo
>	Maior que	3 > 2
<	Menor que	2 < 5
>=	Maior igual que	2 >= 2
<=	Menor igual que	2 <= 1
!=	Diferente de	True != False
==	Igual a	True == 1

Caso o valor da variável *media* satisfaça a condição, será exibido ao usuário o resultado correspondente.

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

Inicialmente, o programa exige a digitação de dois números juntamente com o caractere desejado, iniciando-se assim a entrada de dados.

```
n1 = int(input('Um valor: '))
c1 = str(input('Caractere: '))
n2 = int(input('Outro valor: '))
```

Posteriormente, são declaradas as operações aritméticas a fim de realizar os respectivos cálculos.

```
so = n1 + n2;
su = n1 - n2;
m = n1 * n2;
d = n1 / n2;
```

A seguir, tem-se um exemplo de uso de If, que por sua vez tem como objetivo verificar se o operador aritmético corresponde a um dos quatro da questão. Em caso positivo, o resultado do cálculo será exibido na tela, caso não houver correspondência será impresso uma mensagem de erro. É notório que foi aplicado o uso de um "elif", que é basicamente uma condição de if junto com else.

```
if c1 == "+":
    print('A soma vale {} '.format(so))
elif c1 == "-":
    print('A subtração vale {}'.format(su))
elif c1 == "*":
    print('A multiplicação vale {}'.format(m))
elif c1 == "/":
    print('A divisão vale {}'.format(d))
else:
    print("ERRO VOCÊ NÃO DIGITOU UM DOS CARACTERES VÁLIDOS")
```

1.5 – **Idem ao exercício anterior, porém agora a solução deverá ser desenvolvida usando um switch-case.**

Neste caso, não será possível desenvolver o exercício da forma pedida, pois a linguagem de programação Python não tem switch/case. Entretanto, a maneira mais comum de fazer um código similar é usando if/else, como foi aplicado no exercício 4.

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

```
# Criar e declarar o tipo da variável
numero = int(input('Digite um número positivo e inteiro: '))

# Criar travessões
print('-' * 20)

# Criação de um laço de repetição
for c in range(1, 11):
        # Exibição para o usuário
        print(f'{numero} x {c} = {numero * c}')
# Criar travessões
print('-' * 20)
```

É feita a criação da variável para armazenar o valor que será digitado pelo usuário e tem a necessidade de ser um número inteiro, logo após foi feito (para exibir melhor) um print que vai mostrar 20 travessões em seguida.

```
# Criar e declarar o tipo da variável
numero = int(input('Digite um número positivo e inteiro: '))

# Criar travessões
print('-' * 20)
```

Para fazer a execução do cálculo vamos utilizar um laço de repetição, em específico for. Esse comando em outras linguagens tem uma linha mais extensa a ser digitada, entretanto em Python o programador não precisa de muito tempo escrevendo e outra característica da linguagem é que ela conta como se fosse um ser humano normal, ou seja, o Python começa a contar do número 1 em diante.

```
# Criação de um laço de repetição
for c in range(1, 11):
        # Exibição para o usuário
        print(f'{numero} x {c} = {numero * c}')
# Criar travessões
print('-' * 20)
```

O papel do c é somente para ser um contador no laço acima e como foi dito no exercício que pedia um programa que calculava o seno e cosseno, usamos novamente o f antes das aspas simples. A função **range()** é a forma mais simples para incrementar no ciclo.

1.7 – **O programa deverá exibir na tela os "n" primeiros termos da série: 2, 5, 10, 17, 26... onde o valor de "n" deverá ser inicialmente digitado. Em tempo esclareço que tal série tem como termo geral (x2 + 1) onde x = {1, 2, 3, 4, 5 ...}.**

```
n = int(input("Entre com o valor de 'n': "))
impar = 1
cont = 0
valor = 1
```

Na imagem acima, podemos ver o início do programa, definindo suas variáveis e seus devidos valores, pode-se ver também que a variável **'n'** foi definida para receber um valor do usuário.

```
while (n <= 0):
    n = int(input("Somente valores positivos: "))
    break
```

Nesta segunda parte, temos basicamente um identificador de números negativos ou iguais a 0, usando o **while**, fazendo com que o valor de 'n' seja pedido novamente, enquanto não for digitado corretamente e usando o **break** para por um fim no loop.

```
print("Sequência: ")
for i in range (0,n):
    cont += 1
    valor += impar
    impar += 2
    print (valor)
```

Aqui temos um laço ***for*** para que a sequência seja feita de forma correta, com a função ***range*** dentro do ***for***, podemos pegar todos os valores do intervalo exigido no parâmetro entre parênteses, ou seja, tudo entre 0 e ***'n'***. Em termos matemáticos, temos o contador que soma 1 em todo loop, iniciando em 0 indo até 'n', impar é igualado a ímpar + 2, ou seja, a cada loop ele soma 2 em seu valor, sendo sempre um número ímpar (1, 3, 5, 7, 9...), enquanto valor é somado ao impar toda vez, assim atualizando o número e agregando um novo valor da sequência, por fim, exibindo a lista exigida.

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

```
print("Sequência de fibonacci até 1000:")

a=1
b=1
```

Primeiramente, definição de variáveis simples para o desenvolvimento da lógica, e um **print**, apenas para sinalizar o início da sequência na execução.

```
while(a < 1000):
        print(a)

        aux = a
        a = b
        b = aux + b
```

Por fim temos um laço de repetição ***while***, onde usamos ***('a' < 1000)*** para que se repita enquanto o valor da lista não for maior que 1000, poderia ser ***('a'<= 1000)***, porém na sequência vem um número maior, logo, não é necessário. Primeiro usamos o ***print*** para exibir ***'a'*** (ou seja,

1), assim iniciando a sequência, logo depois temos ***'aux'*** como um auxiliar para fazer o sistema o igualamos com **'a'**, depois **'a'** foi igualado a **'b'**, por fim, como última operação **'b = aux + b'**, assim redefinindo o valor de **'a'** e o exibindo na lista, até q o laço se finalize, retornando a com valores diferentes, fazendo com que a sequência seja possível.

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

```
n1 = int(input('Digite o primeiro número: '))
```

Primeiramente, foi definida a primeira variável (n1), de tipo inteiro (int), e foi solicitado que o usuário digitasse o primeiro valor.

```
while True:
    n2 = int(input('Digite um valor, maior ou igual ao primeiro, para o segundo número: '))
```

Em seguida, foi criado um laço de repetição (while) onde, enquanto condição for verdadeira, a segunda variável (n2) é definida e é solicitado que o usuário digite o segundo valor.

```
    if not n2<n1:
        break
```

Na terceira e última etapa foi utilizada uma estrutura de decisão (if) para impor a condição de parada. Neste caso, o operador not é utilizado na estrutura de decisão para indicar que a parada só será feita se a condição (n2<n1) for falsa.

Enquanto o valor de n1 for maior que o valor de n2 o laço de repetição é executado, só parando o looping quando o valor de n2 for maior ou igual ao de n1.

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

```
sexo = input('Digite o sexo (F ou M): ')
```

Na primeira etapa a variável (sexo) foi definida e foi solicitado que o usuário digitasse se o sexo era feminino (F) ou masculino (M).

```
while sexo.upper() != 'M' and sexo.upper() != 'F':
    sexo = input('Digite o sexo (F ou M): ')
```

Na segunda etapa foi criado um laço de repetição (while), que será executado enquanto o sexo for diferente de F e M. Nessa etapa é utilizado o método upper() com o objetivo de deixar o texto digitado pelo usuário em caixa alta, para que facilite a validação feita no laço de repetição.

Enquanto o sexo digitado pelo usuário for diferente de F e de M, o laço é executado, solicitando que o usuário digite novamente.

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

Inicialmente, receberemos numa variável um valor, logo após, realiza-se um controle através do "while not numero.isdigit():" que irá verificar se o valor é inteiro e positivo (note que o tipo do valor recebido pela variável não é especificado, se tivesse sido o caso, não seria possível colocar uma mensagem de erro de números que não fossem inteiros, apenas seria mostrado erro no programa). Caso o valor não satisfaça a condição, entrará no laço de repetição até que um número inteiro e positivo seja digitado. A variável resposta posteriormente será usada para perguntar se o usuário pretende realizar o fatorial de outro número. Por isso recebe "S", ou seja, enquanto resposta for "S", fará o processo de cálculo de fatorial.

```
resposta = 'S'

while resposta.upper() == 'S':

    numero = input('Digite um número inteiro e positivo: ')

    while not numero.isdigit():
        numero = input('Digite um número inteiro e positivo: ')

    fatorial = 1
    i = 2
```

A variável “fatorial” iniciará o fatorial do número selecionado; o “i” funcionará como um contador auxiliando nas multiplicações do fatorial.

```
fatorial = 1
i = 2

while i <= int(numero):
    fatorial = fatorial * i
    i = i + 1

print('O fatorial de %d é:' %int(numero), fatorial)
```

Cria-se um “while” do qual o contador, como foi dito, será a variável “i”. Assim, fatorial receberá o produto anterior multiplicado pelo contador “i”. Então, por exemplo, fatorial (1) * i (2) será igual a 2; posteriormente, “i” soma 1, e passa a valer 3. Consequentemente, fatorial receberá fatorial (2) * i (3), passando a valer 6. Esse processo terminará quando “i” for igual “int(numero). Note que foi usado “int(numero)” ao invés de apenas “número” pois no momento em que fora atribuído valor a essa variável, não havia sido especificado o seu tipo, como já foi dito. Assim, obtêm-se o fatorial do número que o usuário escolher. Posteriormente, através do comando print, mostra-se o resultado (também usando-se de “int(numero) ao invés de apenas numero).

```
resposta = input('Deseja fazer o fatorial de outro número? (S ou N) ')

while resposta.upper() != 'S' and resposta.upper() != 'N':
    resposta = input('ERRO! Deseja fazer o fatorial de outro número? (S ou N) ')
```

Como foi dito, a variável “resposta” servirá para saber se o usuário quer ou não fazer o fatorial de outro número. O laço “while” controla para que o usuário digite apenas “S” ou “N”. O comando “.upper()” em frente a string faz com que, mesmo que o usuário digite apenas “s” ou “n”, funcione da mesma forma, ou seja, “resposta” receberá strings em maiúsculo apenas.

Caso a resposta do usuário seja “S”, o programa se repetirá, caso seja “N”, finalizará. Para isso, todo o programa deve estar dentro do “while” como mostra a figura a seguir:

```
resposta = 'S'

while resposta.upper() == 'S':

    numero = int(input('Digite um número inteiro e positivo: '))

    while numero < 0:
        numero = int(input('Digite um número inteiro e positivo: '))

    fatorial = 1
    i = 2

    while i <= numero:...

    print('O fatorial de %d é:' %numero, fatorial)

    resposta = input('Deseja fazer o fatorial de outro número? (S ou N) ')

    while resposta.upper() != 'S' and resposta.upper() != 'N':
        resposta = input('ERRO! Deseja fazer o fatorial de outro número? (S ou N) ')
```

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

Para esse exercício, inicialmente cria-se um vetor, no caso será “y”. A variável “j” será usada apenas para efeito de design.

```
y = []

j = 1

for i in range(10):
    y.append(input('Digite o %d° número: ' %j))
    j = j + 1
```

Cria-se um “for” com *range* 10, ou seja, o laço se repetirá 10 vezes. Com o comando “.append” adiciona-se valor ao vetor.

```
print()

print('Ordem inversa:')

for x in reversed(y):
    print(x)
```

Após colocar os valores no vetor, basta usar um “for” com o comando “*reversed*” que o vetor será mostrado na ordem inversa.

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

Iniciamos o programa declarando um vetor (vet), que na linguagem de programação Python não se comporta igual na linguagem de programação C. Em Python o vetor não existe, e ele é como se fosse uma lista que armazena os números. Ao declararmos, colocamos também outra variavel para fazer a somatória no final. Dentro do laço for, pedimos a digitação de 10 números, que será armazenado e mostrado logo em seguida.

```
vet =[];
s = 0;


for i in range (0, 10):
    i = float (input ('Digite o numero:'))
    vet.append(i);

print (vet)
```

Em seguida vamos entrar em um outro laço de repetição, onde todos os números que estão armazenados serão somados e atribuídos a variavel ‘s’ somatória. Ao longo que todos os números foram somados, ele irá sair do laço e fará uma média dos números inseridos, para que em seguida seja mostrado na tela

```
for n in vet:
    s=s+n

m=s/10
print('A media é:'+ str(m))
```

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

Iniciamos o programa declarando um vetor (vet), que na linguagem de programação Python não se comporta igual na linguagem de programação C. Em Python o vetor não existe, e ele é como se fosse uma lista que armazena os números. Ao declararmos, colocamos também outra variavel para fazer a somatória no final. Dentro do laço for, pedimos a digitação de 10 números, que será armazenado e mostrado logo em seguida.

```
vet = []

for i in range(1,11):
    vet.append(float(input("digite o " + str(i) +"º número:")))
```

Para a verificação do maior e menor número, criamos um laço de repetição (for), cujo o primeiro número a ser inserido pela pessoa, possuirá automaticamente a posição de menor número e assim que o segundo número for colocado o laço irá verificar se esse número é maior ou menor que o primeiro já inserido fazendo essa verificação com todos e apresentando na tela.

```
minimo = vet[0]
maximo = minimo

for n in vet:
    if n < minimo:
        minimo = n
    else:
        if n > maximo:
            maximo = n

print('mínimo é: ' + str(minimo))
print('máximo é: ' + str(maximo))
```

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

A seguir temos o código completo, que será destrinchado para melhor entendimento:

```
#Cria a matriz para nome e idade das pessoas solicitadas
pessoas = [[0, 0, 0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]]

#Laço de alimentação da matriz
for c in range(10):
    pessoas[0][c] = input('Nome da Pessoa: ' )
    pessoas[1][c] = input('Idade da Pessoa: ' )

#While criado para que sirva como o do...while do C#, assim permitindo que a pesquisa seja repetida
while True:
   salvar = ""
   pergunta = input('Qual nome deseja pesquisar?')

   #Laço para a pesquisa dentro da matriz
   for c in range(10):
    #Condicionante para que a pesquisa seja feita
     if pergunta == pessoas[0][c]:
        salvar = pessoas[1][c]

   #Condicionante para que seja analisado se houve uma pessoa encontrada na pesquisa feita no laço anterior
   if salvar != "":
      print("A idade da pessoa pesquisada é: " + salvar)
   else:
      print("Pessoa não identificada")

   #Pergunta se o usuario deseja ou não fazer uma nova pesquisa
   resposta = input('Deseja fazer outra pesquisa? (Digite somente S para fazer uma nova pesquisa): ')

   #Caso a resposta não seja Sim (S) finaliza a tarefa, caso contrario continuamos refazemos a pesquisa.
   if not resposta == "S":
    break
```

Há uma diferença enorme entre a criação de matrizes em C# e a criação de matrizes em Python. A matriz em Python deve ser criada com todas as casas que serão utilizadas, não somente colocando a quantidade de casas que utilizaremos. Podemos ver a seguir a criação de uma matriz em nosso programa:

```
#Cria a matriz para nome e idade das pessoas solicitadas
pessoas = [[0, 0, 0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]]
```

Continuando a analisar o código, veremos o laço de alimentação da matriz, que pedirá para o usuário digitar o Nome e a Idade da pessoa a quantidade de vezes solicitadas no programa e armazenar esses dados na matriz:

```
#Laço de alimentação da matriz
for c in range(10):
    pessoas[0][c] = input('Nome da Pessoa: ' )
    pessoas[1][c] = input('Idade da Pessoa: ' )
```

Após isso é criado um While, que servirá como o do...while do C#, para que seja feita uma nova pesquisa. Em seguida será feita solicitado ao usuário o nome da pessoa que ele deseja pesquisar:

```
#While criado para que sirva como o do...while do C#, assim permitindo que a pesquisa seja repetida
while True:
    salvar = ""
    pergunta = input('Qual nome deseja pesquisar?')
```

Posteriormente ao usuário ter digitado o nome, será feito um laço para que seja varrido todas as casas da matriz. Logo abaixo do laço vemos a condicionante, onde somente será inserido na variável salvar a idade da pessoa pesquisada:

```
#Laço para a pesquisa dentro da matriz
for c in range(10):
 #Condicionante para que a pesquisa seja feita
  if pergunta == pessoas[0][c]:
     salvar = pessoas[1][c]
```

Logo abaixo, vemos a condicionante para que a idade da pessoa seja apresentada na tela, caso a variável salvar não esteja vazia, é printado a idade da pessoa, caso ela esteja, será printada uma mensagem:

```
#Condicionante para que seja analisado se houve uma pessoa encontrada na pesquisa feita no laço anterior
if salvar != "":
   print("A idade da pessoa pesquisada é: " + salvar)
else:
   print("Pessoa não identificada")
```

A seguir no código, o usuário é perguntado se ele deseja ou não fazer uma nova pesquisa, caso sua resposta seja afirmativa, a condicionante a baixo será falsa e retornaremos ao início da pesquisa, caso contrário, o programa se encerra:

```
#Pergunta se o usuario deseja ou não fazer uma nova pesquisa
resposta = input('Deseja fazer outra pesquisa? (Digite somente S para fazer uma nova pesquisa): ')

#Caso a resposta não seja Sim (S) finaliza a tarefa, caso contrario continuamos refazemos a pesquisa.
if not resposta == "S":
 break
```

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

A seguir temos o código completo, que será destrinchado para melhor entendimento:

```
#Cria matriz 15 X 20
assento = [[1, 2, 3 , 4, 5, 6,7 , 8, 9, 10, 11, 12, 13, 14, 15],
           [1, 2, 3 , 4, 5, 6,7 , 8, 9, 10, 11, 12, 13, 14, 15],
           [1, 2, 3 , 4, 5, 6,7 , 8, 9, 10, 11, 12, 13, 14, 15],
           [1, 2, 3 , 4, 5, 6,7 , 8, 9, 10, 11, 12, 13, 14, 15],
           [1, 2, 3 , 4, 5, 6,7 , 8, 9, 10, 11, 12, 13, 14, 15],
           [1, 2, 3 , 4, 5, 6,7 , 8, 9, 10, 11, 12, 13, 14, 15],
           [1, 2, 3 , 4, 5, 6,7 , 8, 9, 10, 11, 12, 13, 14, 15],
           [1, 2, 3 , 4, 5, 6,7 , 8, 9, 10, 11, 12, 13, 14, 15],
           [1, 2, 3 , 4, 5, 6,7 , 8, 9, 10, 11, 12, 13, 14, 15],
           [1, 2, 3 , 4, 5, 6,7 , 8, 9, 10, 11, 12, 13, 14, 15],
           [1, 2, 3 , 4, 5, 6,7 , 8, 9, 10, 11, 12, 13, 14, 15],
           [1, 2, 3 , 4, 5, 6,7 , 8, 9, 10, 11, 12, 13, 14, 15],
           [1, 2, 3 , 4, 5, 6,7 , 8, 9, 10, 11, 12, 13, 14, 15],
           [1, 2, 3 , 4, 5, 6,7 , 8, 9, 10, 11, 12, 13, 14, 15],
           [1, 2, 3 , 4, 5, 6,7 , 8, 9, 10, 11, 12, 13, 14, 15],
           [1, 2, 3 , 4, 5, 6,7 , 8, 9, 10, 11, 12, 13, 14, 15],
           [1, 2, 3 , 4, 5, 6,7 , 8, 9, 10, 11, 12, 13, 14, 15],
           [1, 2, 3 , 4, 5, 6,7 , 8, 9, 10, 11, 12, 13, 14, 15],
           [1, 2, 3 , 4, 5, 6,7 , 8, 9, 10, 11, 12, 13, 14, 15],
           [1, 2, 3 , 4, 5, 6,7 , 8, 9, 10, 11, 12, 13, 14, 15]]

while True:
    Nome = input('Seu nome: ')
    Fileira = int(input('Fileira que deseja reservar: '))
    Cadeira = int(input('Cadeira que deseja reservar: '))

    #O método split() vai quebrar a string pelo separador informado, como não informou ele vai quebrar pelo espaço e usar o índice zero para pegar o primeiro elemento.
    if str(assento[Fileira - 1][Cadeira - 1]).split(",")[0] == "Ocupado":
        print('Desculpe, cadeira já reservada!')
    else:
        assento[Fileira - 1][Cadeira - 1] = "Ocupado, " + Nome
        print('Sua cadeira foi reservada com sucesso!')

    resposta = input("Deseja fazer uma nova reserva? (Digite somente S para uma nova reserva): ")

    if not resposta == "S":
        break
```

Como explicado no exercício anterior, a matriz tem que ser feita uma a uma de suas casas. Neste trecho do código vemos a criação da matriz:

```
#Cria matriz 20 X 15
assento = [[1, 2, 3 , 4, 5, 6,7 , 8, 9, 10, 11, 12, 13, 14, 15],
           [1, 2, 3 , 4, 5, 6,7 , 8, 9, 10, 11, 12, 13, 14, 15],
           [1, 2, 3 , 4, 5, 6,7 , 8, 9, 10, 11, 12, 13, 14, 15],
           [1, 2, 3 , 4, 5, 6,7 , 8, 9, 10, 11, 12, 13, 14, 15],
           [1, 2, 3 , 4, 5, 6,7 , 8, 9, 10, 11, 12, 13, 14, 15],
           [1, 2, 3 , 4, 5, 6,7 , 8, 9, 10, 11, 12, 13, 14, 15],
           [1, 2, 3 , 4, 5, 6,7 , 8, 9, 10, 11, 12, 13, 14, 15],
           [1, 2, 3 , 4, 5, 6,7 , 8, 9, 10, 11, 12, 13, 14, 15],
           [1, 2, 3 , 4, 5, 6,7 , 8, 9, 10, 11, 12, 13, 14, 15],
           [1, 2, 3 , 4, 5, 6,7 , 8, 9, 10, 11, 12, 13, 14, 15],
           [1, 2, 3 , 4, 5, 6,7 , 8, 9, 10, 11, 12, 13, 14, 15],
           [1, 2, 3 , 4, 5, 6,7 , 8, 9, 10, 11, 12, 13, 14, 15],
           [1, 2, 3 , 4, 5, 6,7 , 8, 9, 10, 11, 12, 13, 14, 15],
           [1, 2, 3 , 4, 5, 6,7 , 8, 9, 10, 11, 12, 13, 14, 15],
           [1, 2, 3 , 4, 5, 6,7 , 8, 9, 10, 11, 12, 13, 14, 15],
           [1, 2, 3 , 4, 5, 6,7 , 8, 9, 10, 11, 12, 13, 14, 15],
           [1, 2, 3 , 4, 5, 6,7 , 8, 9, 10, 11, 12, 13, 14, 15],
           [1, 2, 3 , 4, 5, 6,7 , 8, 9, 10, 11, 12, 13, 14, 15],
           [1, 2, 3 , 4, 5, 6,7 , 8, 9, 10, 11, 12, 13, 14, 15],
           [1, 2, 3 , 4, 5, 6,7 , 8, 9, 10, 11, 12, 13, 14, 15]]
```

Assim como no código anterior, utilizaremos o while True para fazermos o processo quantas vezes o usuário precisar:

```
#While criado para que sirva como o do...while do C#, assim permitindo que a pesquisa seja repetida
while True:
```

No código a seguir, é pedido ao usuário que digite o seu nome, fileira e cadeira que deseja se sentar:

```
    Nome = input('Seu nome: ')
    Fileira = int(input('Fileira que deseja reservar: '))
    Cadeira = int(input('Cadeira que deseja reservar: '))
```

Logo após, podemos observar a condicionante onde deveremos destrincha-la em algumas partes:

1 – Vemos que utilizamos “Fileira - 1” e “Cadeira – 1”, isto ocorre, pois, as matrizes começam no número 0 e vão até o número escolhido -1. Por isso que será utilizado o Cadeira/Fileira – 1;

2 – É utilizado o comando .split, o qual tem a finalidade de dividir a string. A que utilizamos, solicitamos que a string seja dividida a cada “,”, podemos ver isso dentro dos parênteses. Ademais, podemos ver dentro dos colchetes o número 0, que se refere a qual parte da string iremos utilizar, e neste caso utilizaremos a parte 0, a 1º parte;

3 – Ao passarmos por este processo inteiro, se a 1° parte da string for igual a Ocupado, retornaremos a mensagem que a cadeira está ocupada, caso contrário, retornamos a mensagem de sucesso na reserva.

```
#O método split() vai quebrar a string pelo separador informado, como não informou ele vai quebrar pelo espaço e usar o índice zero para pegar o primeiro elemento.
if str(assento[Fileira - 1][Cadeira - 1]).split(",")[0] == "Ocupado":
    print('Desculpe, cadeira já reservada!')
else:
    assento[Fileira - 1][Cadeira - 1] = "Ocupado, " + Nome
    print('Sua cadeira foi reservada com sucesso!')
```

A seguir no código, o usuário é perguntado se ele deseja ou não fazer uma nova pesquisa, caso sua resposta seja afirmativa, a condicionante a baixo será falsa e retornaremos ao início da pesquisa, caso contrário, o programa se encerra:

```
    resposta = input("Deseja fazer uma nova reserva? (Digite somente S para uma nova reserva): ")

    if not resposta == "S":
        break
```

Capítulo 3 – Apresentando a interface gráfica

1 – O que é PyQt5 e como ele funciona?

Para entender o que é PyQt, primeiramente é necessário entender o que é o Qt.

O Qt é um framework utilizado para criar interfaces gráficas. A princípio, era utilizado somente em C++, porém, devido a necessidade de seus usuários de programar em outras linguagens e da afinidade que esses tinham com a plataforma, foram desenvolvidas algumas versões da ferramenta voltadas para a programação em linguagens diferentes, como por exemplo em Python e Java.

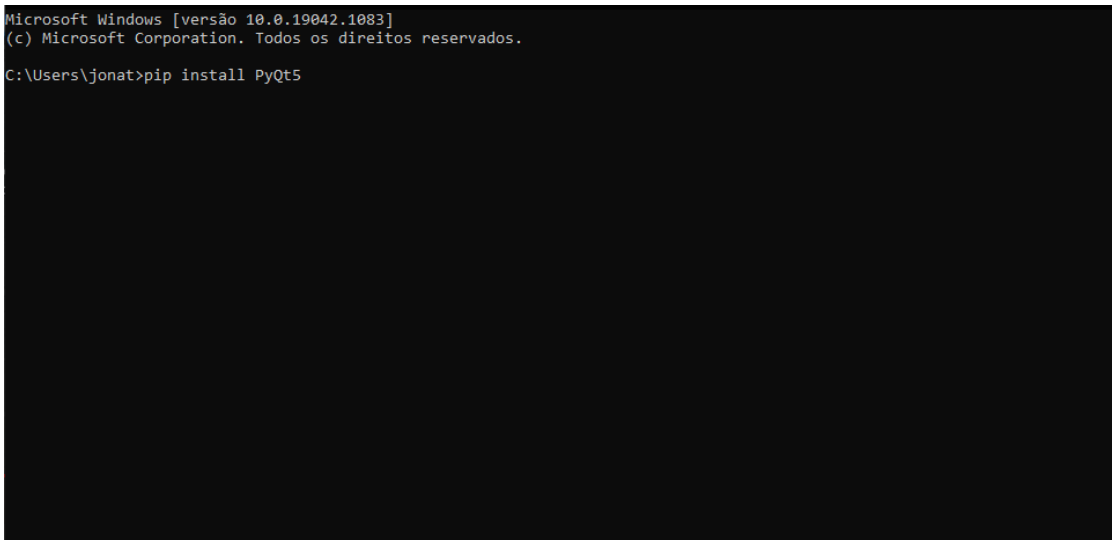
O PyQt5, portanto, é a quinta versão da ferramenta Qt voltada para a programação em Python

Primeiramente para se obter PyQt5 precisamos ter o Python instado no computador e obter a instalação do PyQt5, em seguida abriremos o pront de comando e quando o console for aberto chamaremos a plataforma para ser aberta.

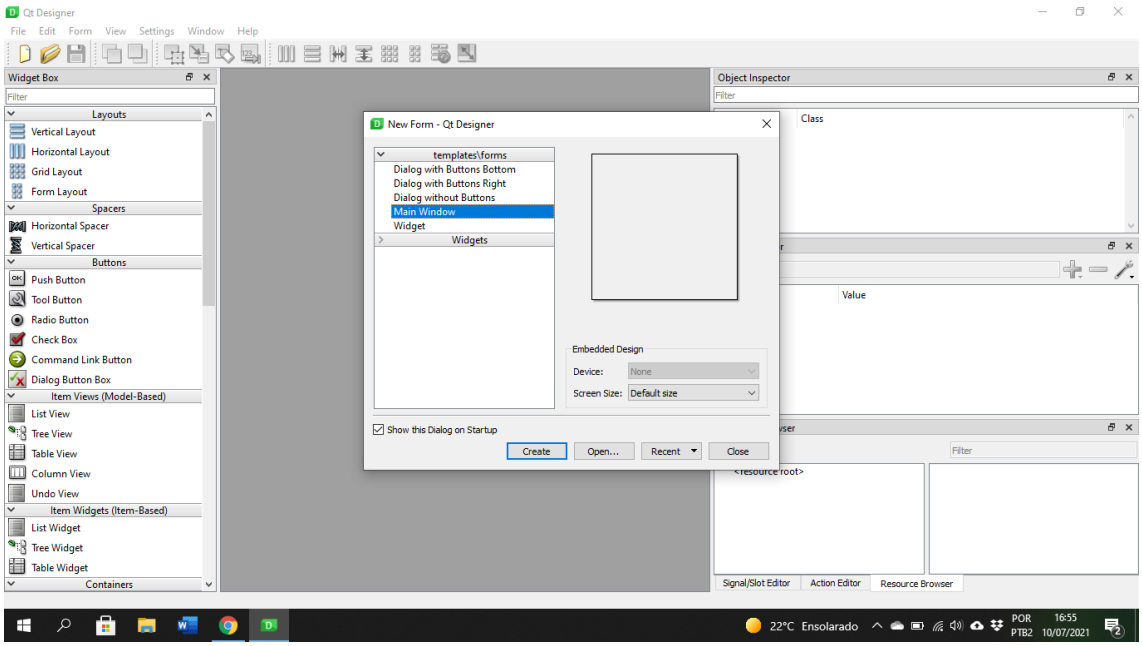
Para termos o bom desenvolvimento da plataforma para o designer, temos que instalar uma ferramenta chamada QtDesigner, essa ferramenta ela permite a criação de interfaces utilizando somente recursos Drag-and-Drop ou seja, não serão mais necessárias várias linhas de código para criar uma janela, com essa ferramenta nós podemos simplesmente clicar e arrastar os widgets para nossa interface, sendo somente necessário aplicar a lógica para interação e eventos.

Em seguida, reinicie o pront de comandos, digite "designer" e pressione enter, e já conseguiremos utilizar a plataforma corretamente.

2 – Instalando e conhecendo o PyQt5

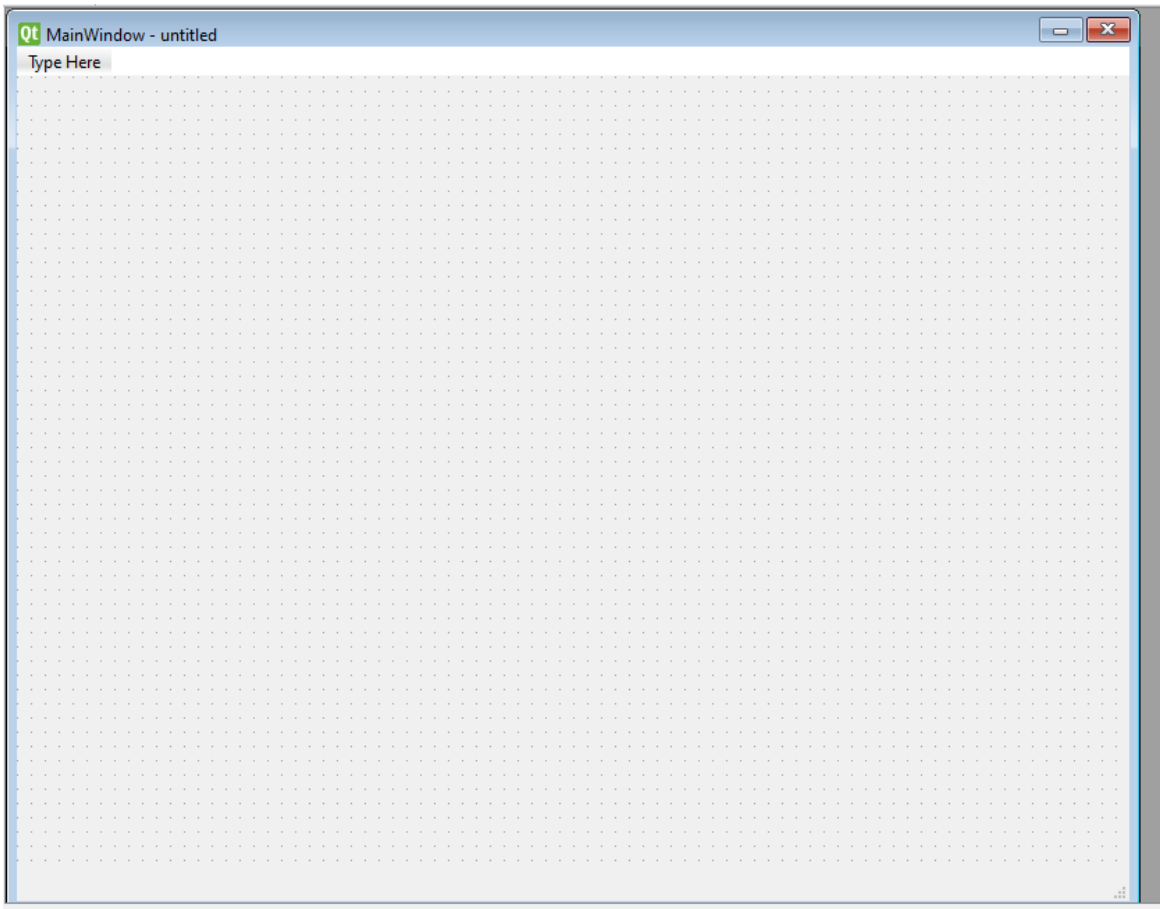


Para a instalação do PyQt5 é necessário acessar o prompt de comando do Windows (CMD), executá-lo como administrador e digitar a seguinte linha de comando: “pip install pyqt5”, onde “pip” é o gerenciador de pacote do Python. Após esse procedimento, a instalação do PyQt5 será iniciada. Caso o comando não seja executado, verifique se seu sistema operacional tem o Python instalado, caso não tenha, é preciso ir até o site da linguagem que é: https://www.python.org/downloads/ e baixar sua última versão. Linux e Mac Os já vem de fábrica com o Python instalado.

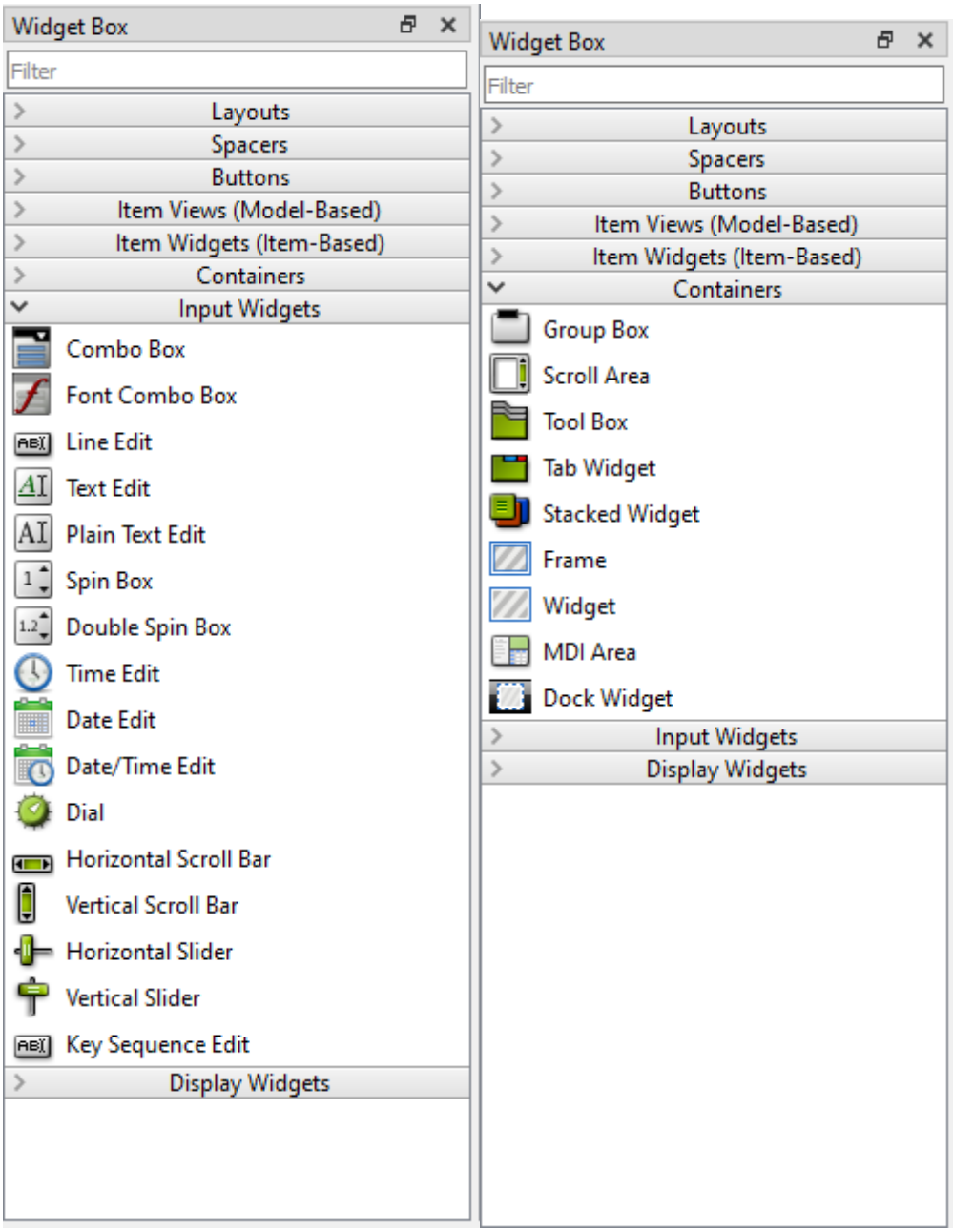


Assim que estiver instalado, ao clicar para entrar no PyQt5 aparecerá uma janela com as opções para dar início a primeira parte do processo de criação da interface gráfica. Nesse caso, será usado o “Main Window”, que cria apenas a janela.

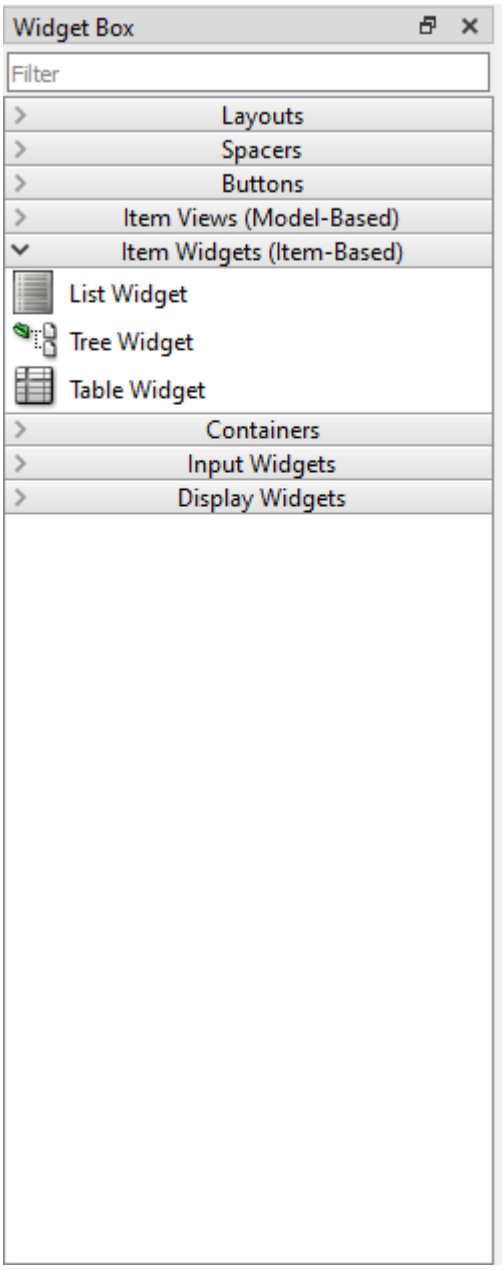
Após realizar os procedimentos citados acima a janela será criada e estará pronta para uso, sendo possível escolher e posicionar os elementos na interface gráfica. Não é preciso detalhar cada um dos componentes na tela, somente clicar no elemento desejado e arrastá-lo para o local escolhido.

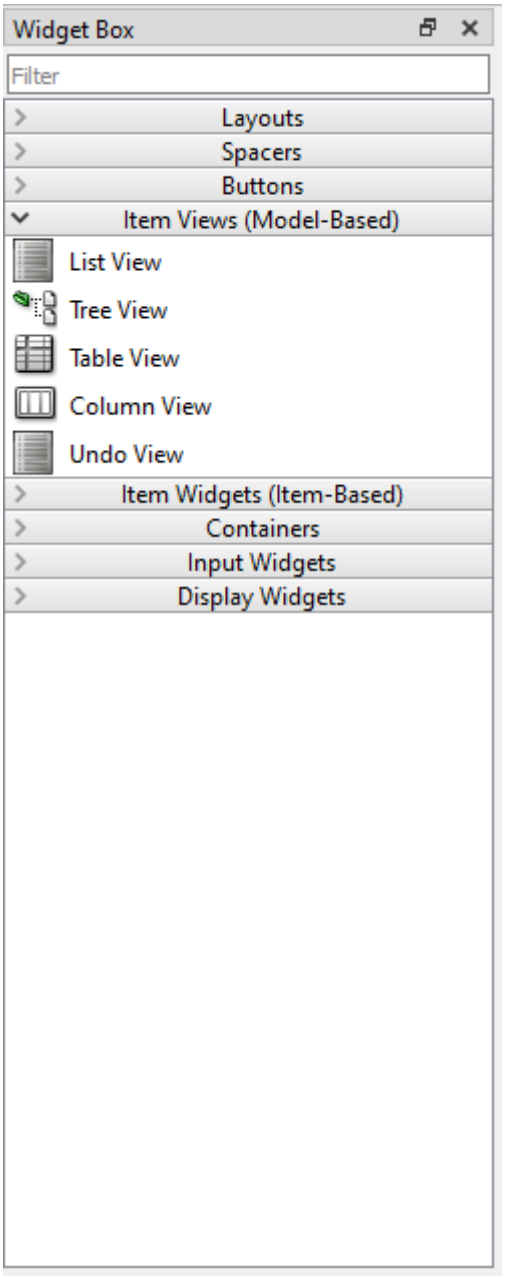


No lado esquerdo da tela terão vários menus, os quais disponibilizarão os recursos responsáveis para a montagem da interface gráfica, conforme segue abaixo nos menus “Input Widget” e “Containers”:

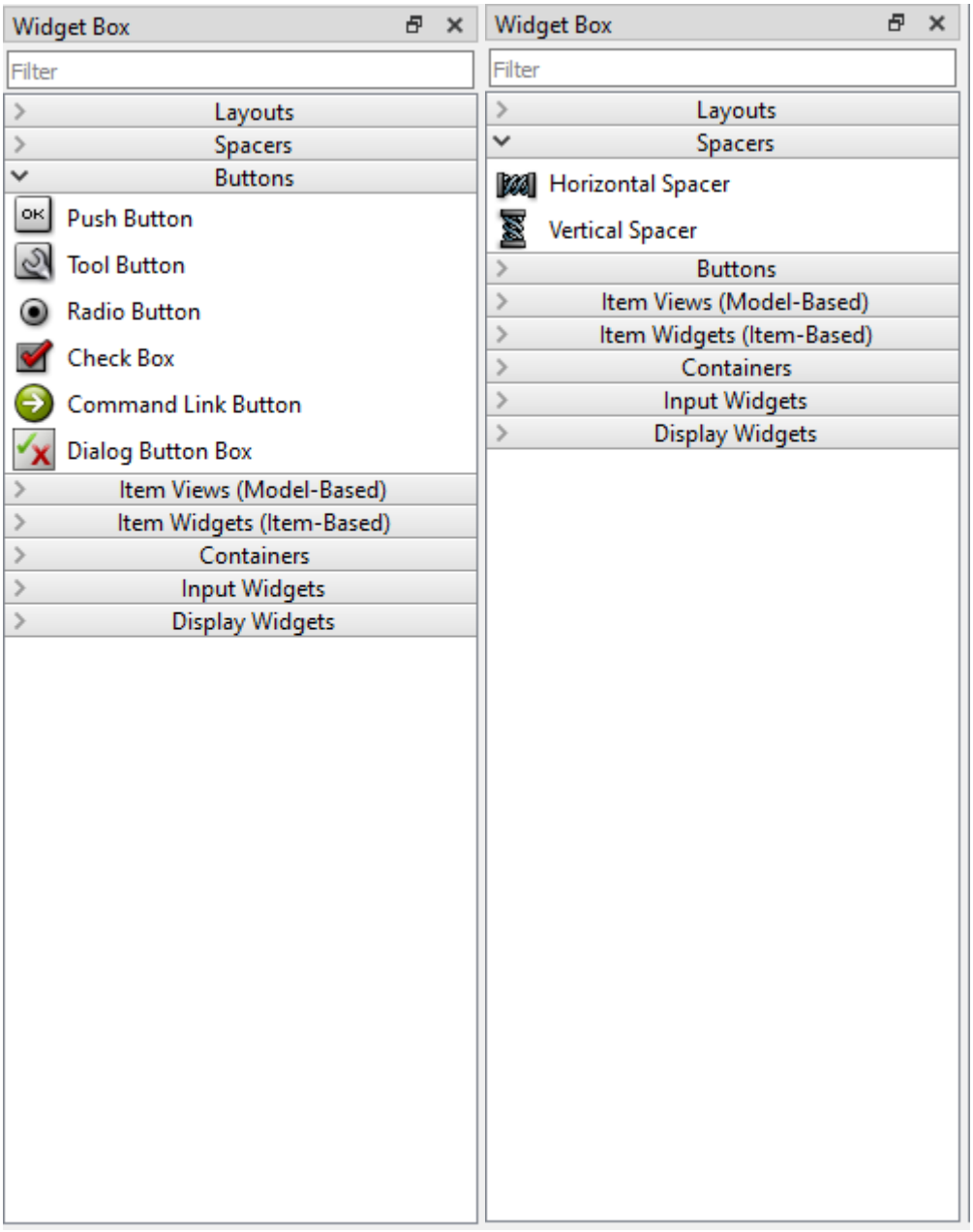


Nos menus “Item Widgets” e “Item Views” encontram-se as ferramentas para organizar as informações em listas, tabelas, árvores e colunas.

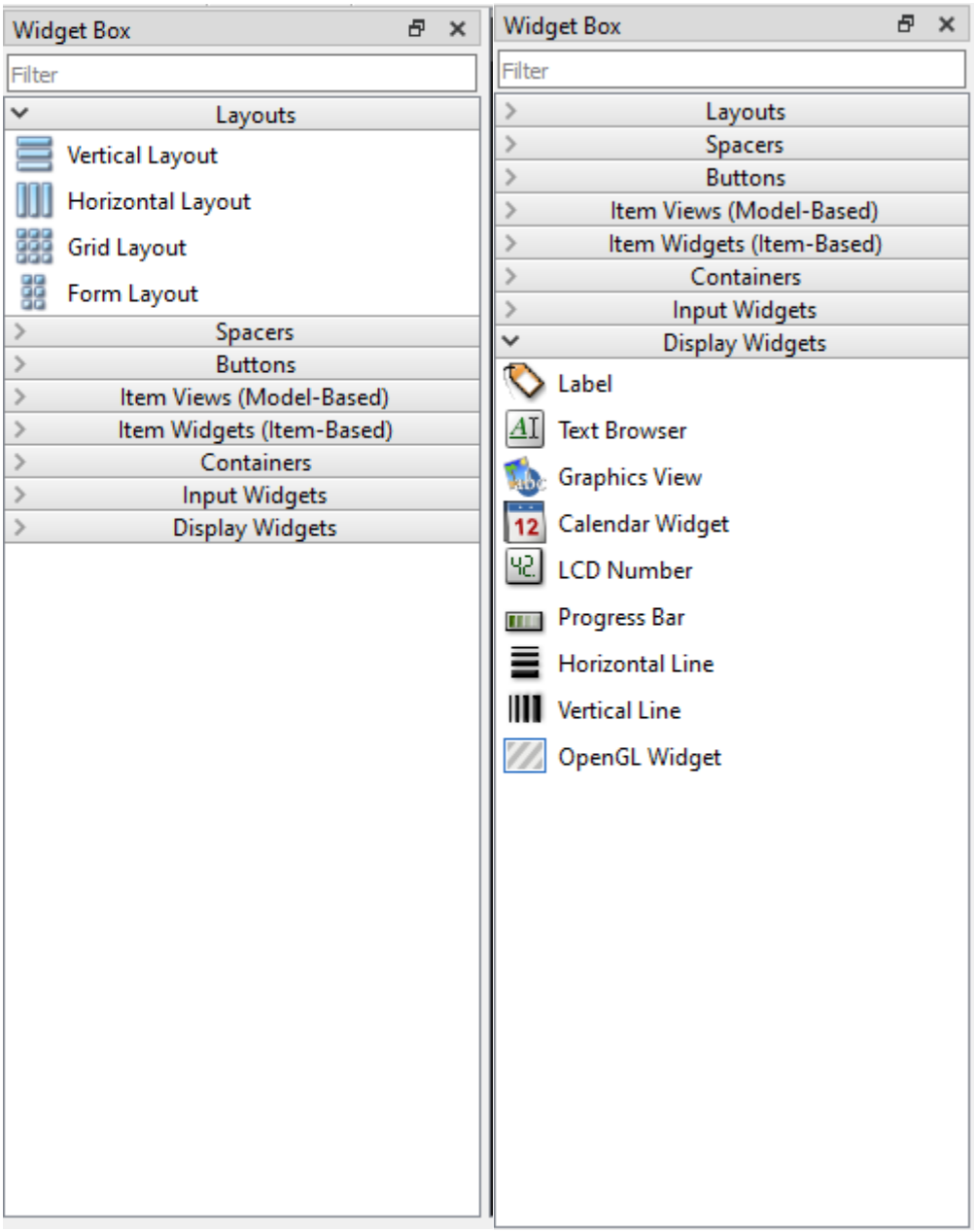




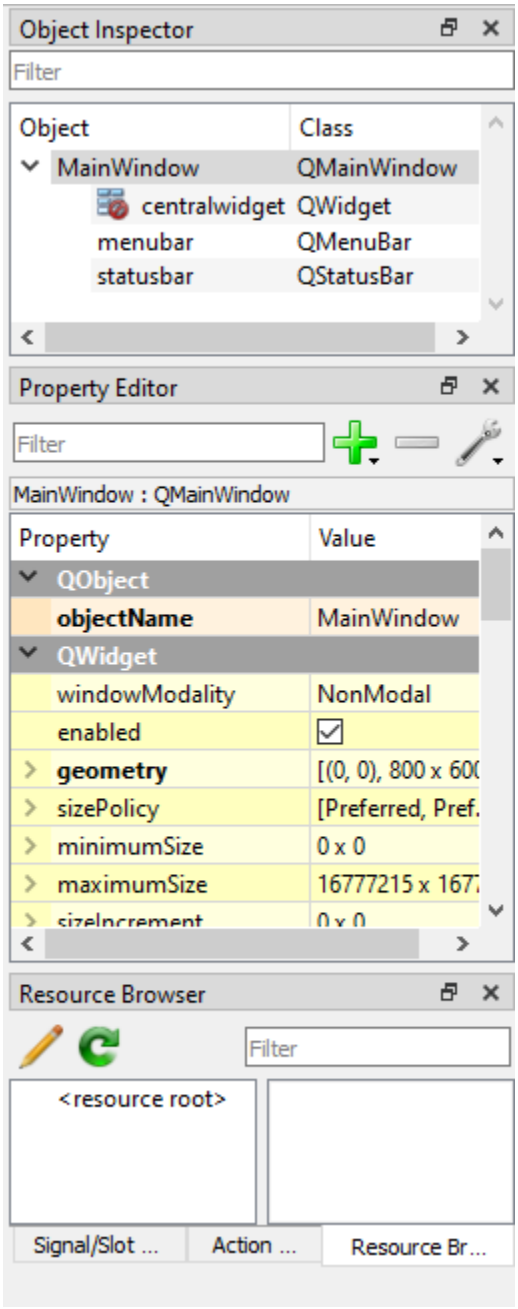
No menu “Buttons” encontram-se vários tipos de botões com diversas funcionalidades. Enquanto no menu “Spacers” disponibilizam ferramentas que auxiliam no espaçamento tanto na vertical quanto na horizontal.



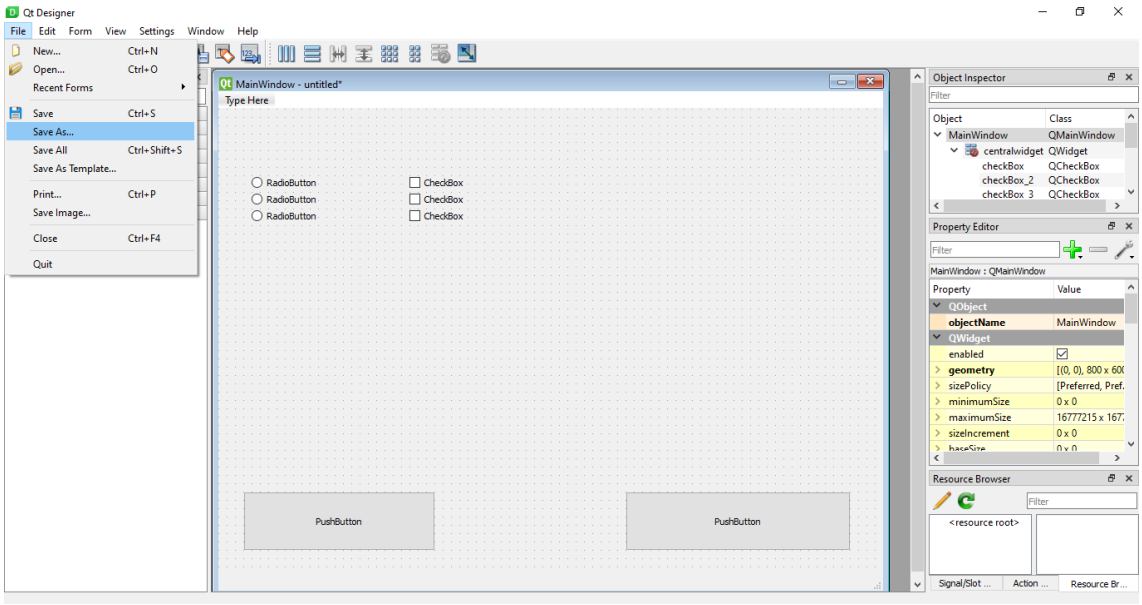
No menu “Layouts” existem diversas opções para a personalização do layout da interface gráfica. Enquanto no menu “Display Widgets” disponibilizam várias formas de exibição na tela, como barra de progressão, calendário, gráfico e entre outras ferramentas.



Na barra lateral localizada no canto direito da tela, estão as propriedades relativas ao elemento selecionado, assim, pode-se personalizar o nome, a fonte, definir o tamanho, a cor e entre outras permissões.



Ao finalizar a construção da interface gráfica, será necessário salvar esse projeto para posteriormente gerar o mesmo arquivo em Python e continuar editando-o.

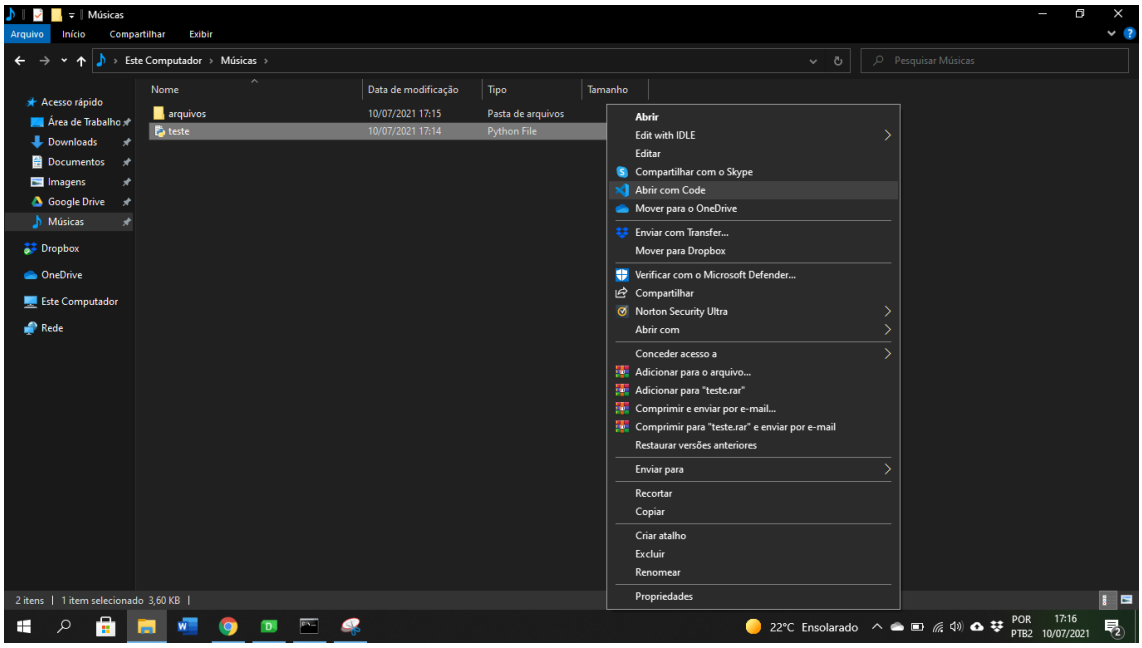


Após clicar em salvar, o arquivo será salvo com a extensão “ui”, conforme o exemplo abaixo:

Em seguida será preciso acessar o terminal do Windows e digitar a linha de comando “pyuic5 –x teste.ui –o teste.py”, onde “ui” localizado no comando “pyuic5” é a extensão do arquivo anteriormente salvo; o “-x” seguido do nome do projeto Qt Designer, o qual será gerado em Python, que nesse caso é “teste.ui”, e por fim, inserir o “-o” seguido do mesmo nome do projeto ou um novo nome e é necessário estar acompanhado da extensão “py”. Em seguida, aperte a tecla “Enter”.

```
C:\Users\jonat\Music>pyuic5 -x teste.ui -o teste.py

C:\Users\jonat\Music>
```

Caso não apareça nenhum tipo de divergência ao pular para a linha de baixo o arquivo foi gerado com sucesso, conforme o exemplo abaixo:



Logo após a criação do arquivo de extensão “.py”, o mesmo será aberto na IDE escolhida, no caso Visual Studio Code (VSCode), e sua apresentação será da seguinte forma:

```python
# -*- coding: utf-8 -*-
# Form implementation generated from reading ui file 'teste.ui'
#
# Created by: PyQt5 UI code generator 5.15.4
#
# WARNING: Any manual changes made to this file will be lost when pyuic5 is
# run again.  Do not edit this file unless you know what you are doing.


from PyQt5 import QtCore, QtGui, QtWidgets
class Ui_MainWindow(object):
    def setupUi(self, MainWindow):
        MainWindow.setObjectName("MainWindow")
        MainWindow.resize(800, 600)
        self.centralwidget = QtWidgets.QWidget(MainWindow)
        self.centralwidget.setObjectName("centralwidget")
        self.pushButton = QtWidgets.QPushButton(self.centralwidget)
        self.pushButton.setGeometry(QtCore.QRect(30, 460, 231, 71))
        self.pushButton.setObjectName("pushButton")
        self.pushButton_2 = QtWidgets.QPushButton(self.centralwidget)
        self.pushButton_2.setGeometry(QtCore.QRect(490, 460, 271, 71))
        self.pushButton_2.setObjectName("pushButton_2")
        self.radioButton = QtWidgets.QRadioButton(self.centralwidget)
        self.radioButton.setGeometry(QtCore.QRect(40, 80, 82, 17))
        self.radioButton.setObjectName("radioButton")
        self.radioButton_2 = QtWidgets.QRadioButton(self.centralwidget)
        self.radioButton_2.setGeometry(QtCore.QRect(40, 100, 82, 17))
        self.radioButton_2.setObjectName("radioButton_2")
        self.radioButton_3 = QtWidgets.QRadioButton(self.centralwidget)
        self.radioButton_3.setGeometry(QtCore.QRect(40, 120, 82, 17))
        self.radioButton_3.setObjectName("radioButton_3")
        self.checkBox = QtWidgets.QCheckBox(self.centralwidget)
        self.checkBox.setGeometry(QtCore.QRect(230, 80, 70, 17))
        self.checkBox.setObjectName("checkBox")
        self.checkBox_2 = QtWidgets.QCheckBox(self.centralwidget)
        self.checkBox_2.setGeometry(QtCore.QRect(230, 100, 70, 17))
        self.checkBox_2.setObjectName("checkBox_2")
        self.checkBox_3 = QtWidgets.QCheckBox(self.centralwidget)
        self.checkBox_3.setGeometry(QtCore.QRect(230, 120, 70, 17))
        self.checkBox_3.setObjectName("checkBox_3")
        MainWindow.setCentralWidget(self.centralwidget)
        self.menubar = QtWidgets.QMenuBar(MainWindow)
        self.menubar.setGeometry(QtCore.QRect(0, 0, 800, 21))
        self.menubar.setObjectName("menubar")
        MainWindow.setMenuBar(self.menubar)
        self.statusbar = QtWidgets.QStatusBar(MainWindow)
        self.statusbar.setObjectName("statusbar")
        MainWindow.setStatusBar(self.statusbar)

        self.retranslateUi(MainWindow)
        QtCore.QMetaObject.connectSlotsByName(MainWindow)

    def retranslateUi(self, MainWindow):
        _translate = QtCore.QCoreApplication.translate
        MainWindow.setWindowTitle(_translate("MainWindow", "MainWindow"))
        self.pushButton.setText(_translate("MainWindow", "PushButton"))
        self.pushButton_2.setText(_translate("MainWindow", "PushButton"))
        self.radioButton.setText(_translate("MainWindow", "RadioButton"))
        self.radioButton_2.setText(_translate("MainWindow", "RadioButton"))
        self.radioButton_3.setText(_translate("MainWindow", "RadioButton"))
        self.checkBox.setText(_translate("MainWindow", "CheckBox"))
        self.checkBox_2.setText(_translate("MainWindow", "CheckBox"))
        self.checkBox_3.setText(_translate("MainWindow", "CheckBox"))

if __name__ == "__main__":
    import sys
    app = QtWidgets.QApplication(sys.argv)
    MainWindow = QtWidgets.QMainWindow()
    ui = Ui_MainWindow()
    ui.setupUi(MainWindow)
    MainWindow.show()
    sys.exit(app.exec_())
```

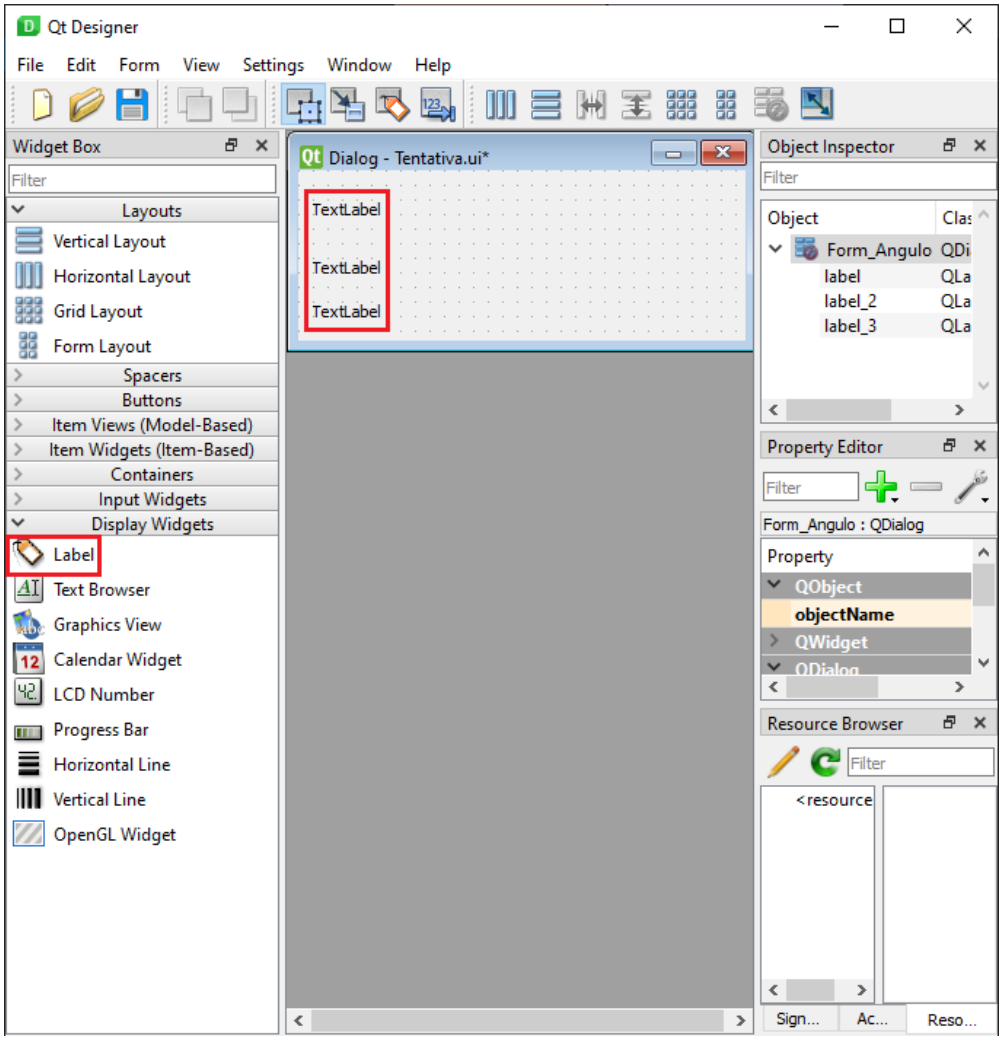
3 – Criando programas em modo gráfico

A seguir, iremos fazer em modo gráfico, três aplicações já explicadas anteriormente.

1° Programa: Aplicação que calcula o seno e o cosseno de um ângulo.

Passo 1: Construindo o formulário

Adicionando *Label's*

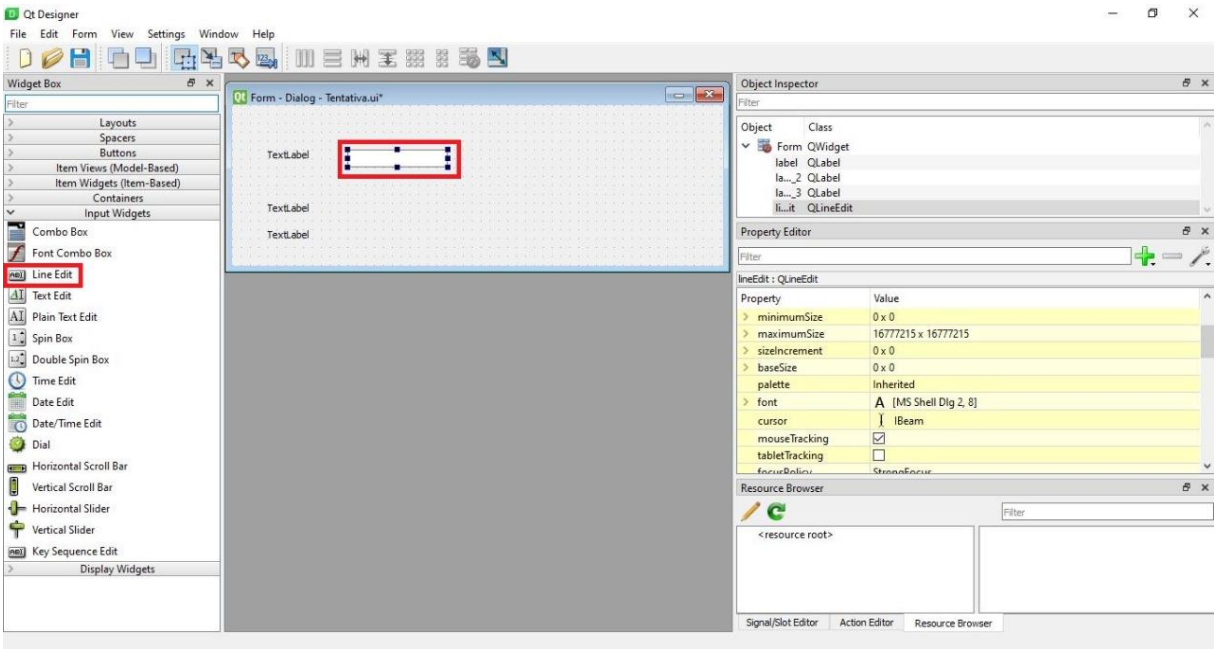


A princípio, clicando no ícone *Label* do menu disposto na lateral esquerda, foram adicionadas 3 *label's. Label*, que traduzido para português seria equivalente a palavra rótulo,

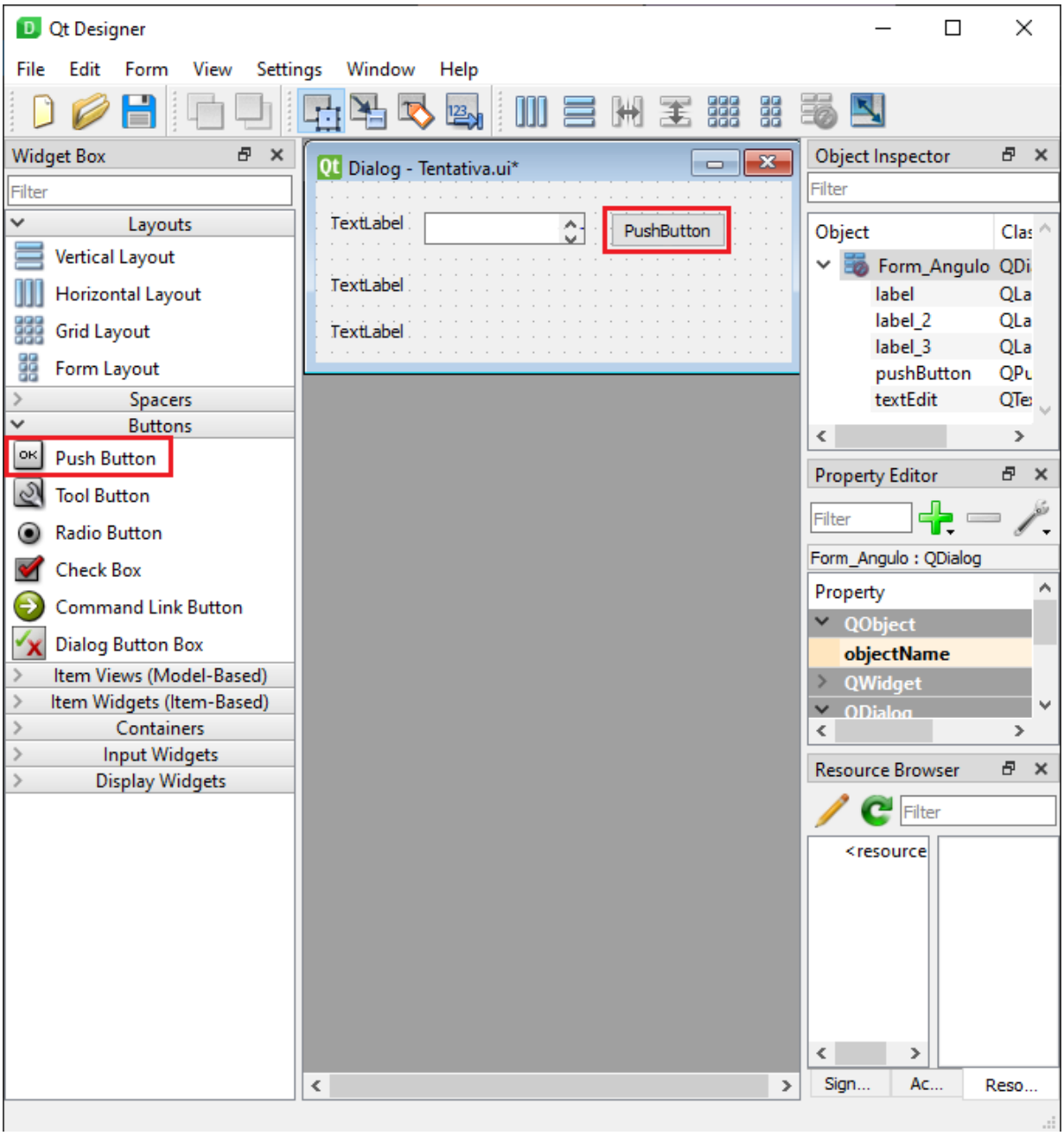
é usada como principal método para adicionar textos não editáveis, como títulos fixos ou *label's* dinâmicas, que recebem algum método como resposta.

Adicinando *Line Edit's*

Em seguida, foi adicionada 1 *Line Edit*, espaço editável que servirá de entrada de dados para o usuário.

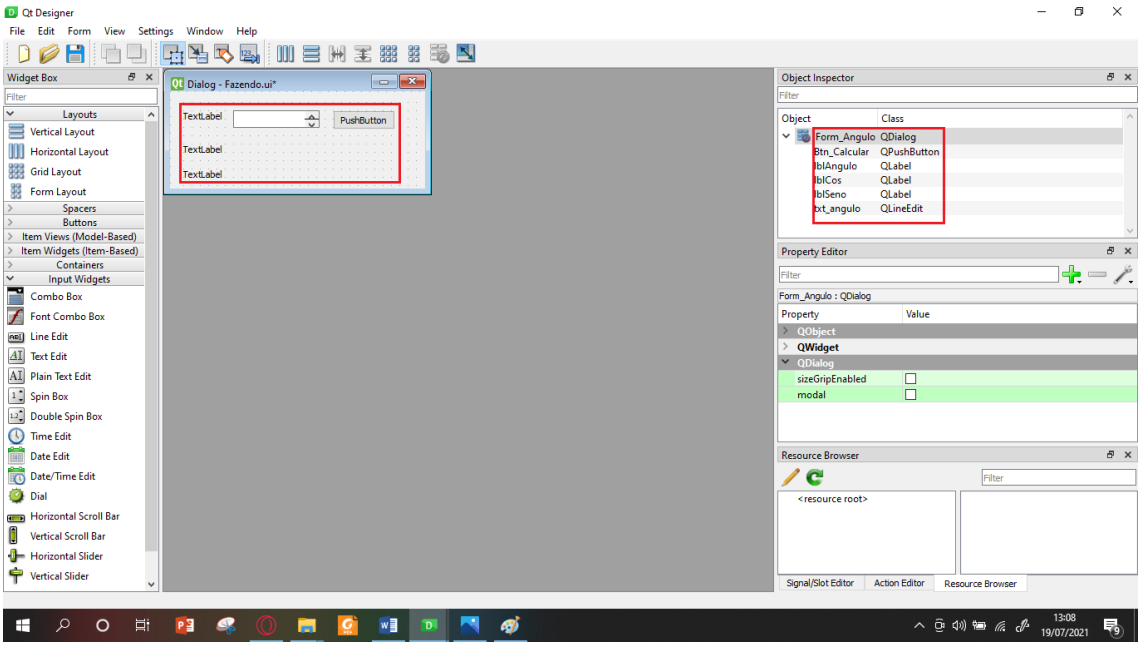


Adicionando *Push Button*



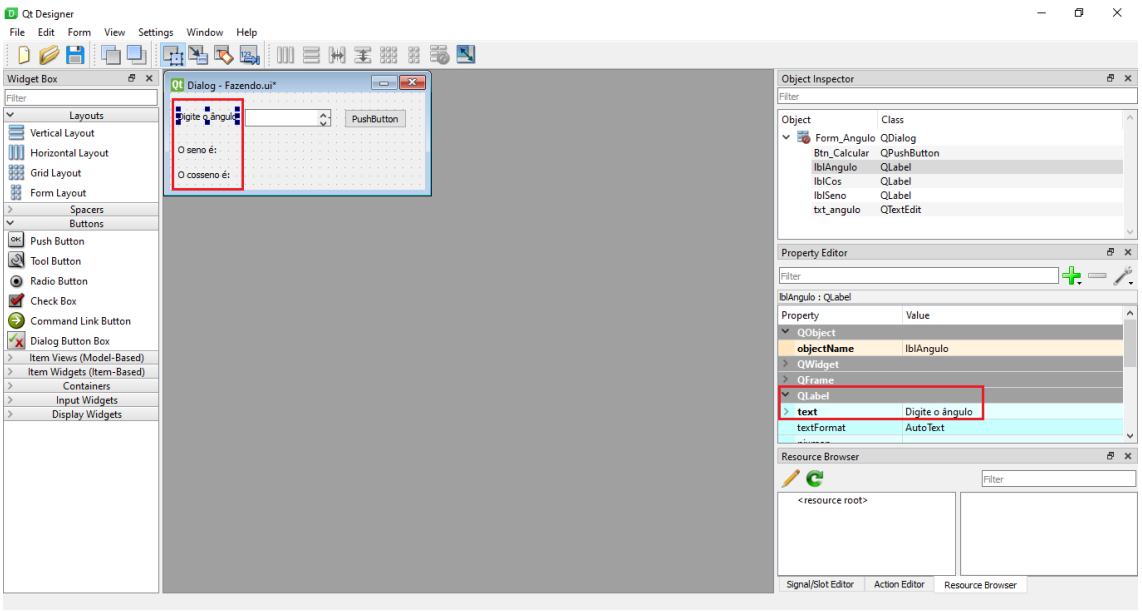
Seguindo na construção do formulário, foi adicionado um Push Button, que nada mais é do que um botão clicável.

Identificando Função



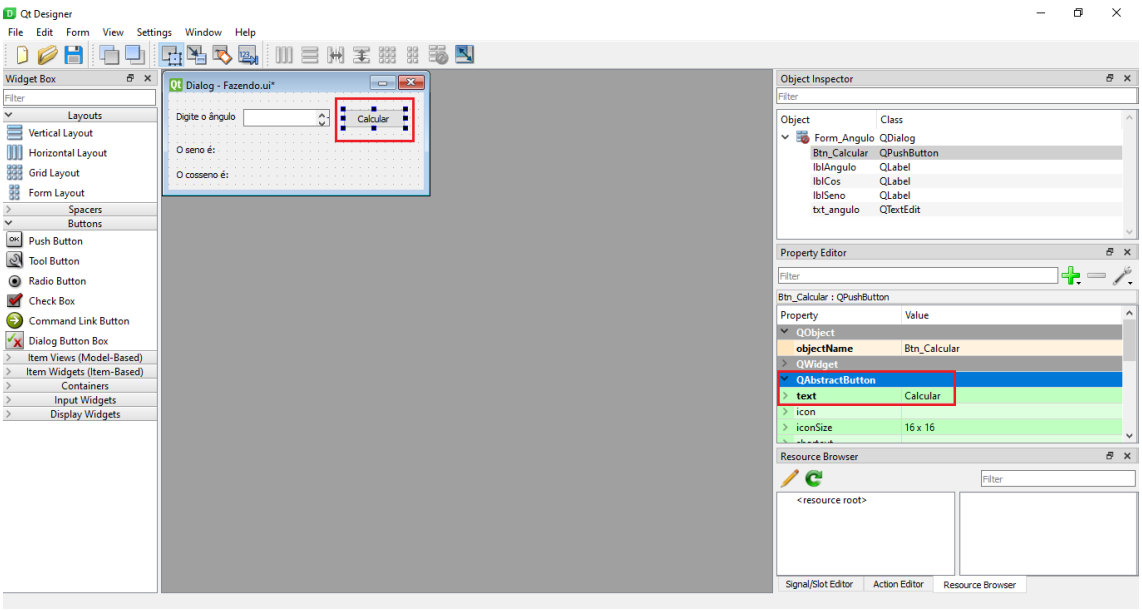
Após a montagem, no campo “Dialog” do menu disposto na lateral direita, foi feita uma identificação dos componentes adicionados no formulário. Está parte é importante, pois a programação da aplicação utilizará desta identificação em sua construção, portanto, certifique-se de identificar todos os componentes utilizados.

Adicionando texto nas *Label’s*



Clicando no ícone *QLabel* do menu disposto na lateral direita, foi alterado o componente *text*, que corresponde ao texto que fica visível para o usuário.

Adicionando texto no *Push Button*



Seguindo na alteração do componente *text*, foi clicado no Ícone QAbstractButton, onde foi alterado o conteúdo que ficaria visível para o usuário.

Obs: Ao final de todas estas etapas, não se esqueça de salvar seu projeto!

Vale ressaltar que a parte gráfica pode ser feita de diferentes maneiras utilizando de todos os recursos disponíveis além dos apresentados nesta aplicação.

Passo 2: Construindo a lógica da aplicação

Transformando ui em py

```
C:\Users\User\Desktop\Tentantiva>pyuic5 -x Fazendo.ui -o Programa1.py
```

Utilizando de conceitos já explicados anteriormente neste capítulo, foi feito uma transformação do arquivo para tornar possível trabalhar na programação com a IDE selecionada.

Importando Biblioteca

```
import math
```

Para tornar possível a utilização e realização de algumas tarefas matemáticas, a linguagem Python possui um conjunto de funções matemáticas integradas, basta apenas importar, como feito neste passo utilizando o *import math*.

Criando um Método

```
def Calcular(self):
    ang = float(self.txt_angulo.text())
    self.lblCos.setText("O cosseno do ângulo é: " + str(math.cos(ang)))
    self.lblSeno.setText("O seno do ângulo é: " + str(math.sin(ang)))
```

Na parte da programação foi criado o método *Calcular()*, que terá a mesma lógica deste mesma aplicação apresentada no capítulo anterior utilizando as funções disponibilizadas pela biblioteca math, havendo apenas mudanças de sintaxe por estar sendo feita no modo gráfico.

Aqui surgem nomenclaturas como *def*, que serve para declarar o método, *self*, que permite acessar os atributos do método, setText, que serve para apresentar a resposta ao usuário.

Chamando o Método

```
self.Btn_Calcular.clicked.connect(self.Calcular)
```

Em seguida, é programado quando o método deve ser chamado, ou seja, quando ele vai funcionar. Neste caso, sempre que o usuário clicar no botão para calcular.

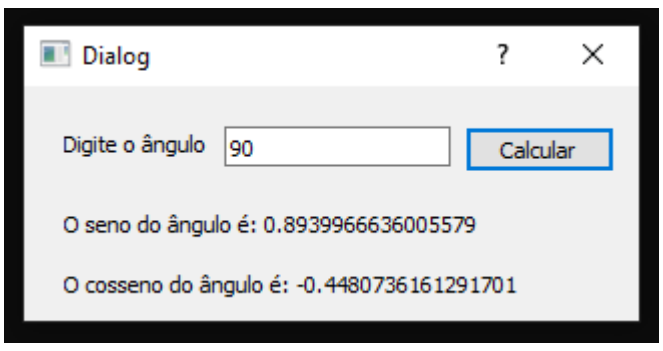
Programa Inteiro

Veja abaixo o visual do programa inteiro, juntando o que foi importado da ui.

```python
import math
from PyQt5 import QtCore, QtGui, QtWidgets


class Ui_Form_Angulo(object):
    def Calcular(self):
        ang = float(self.txt_angulo.text())
        self.lblCos.setText("O cosseno do ângulo é: " + str(math.cos(ang)))
        self.lblSeno.setText("O seno do ângulo é: " + str(math.sin(ang)))

    def setupUi(self, Form_Angulo):
        Form_Angulo.setObjectName("Form_Angulo")
        Form_Angulo.resize(307, 117)
        self.lblAngulo = QtWidgets.QLabel(Form_Angulo)
        self.lblAngulo.setGeometry(QtCore.QRect(20, 20, 71, 16))
        self.lblAngulo.setObjectName("lblAngulo")
        self.lblSeno = QtWidgets.QLabel(Form_Angulo)
        self.lblSeno.setGeometry(QtCore.QRect(20, 60, 271, 16))
        self.lblSeno.setObjectName("lblSeno")
        self.lblCos = QtWidgets.QLabel(Form_Angulo)
        self.lblCos.setGeometry(QtCore.QRect(20, 90, 271, 16))
        self.lblCos.setObjectName("lblCos")
        self.Btn_Calcular = QtWidgets.QPushButton(Form_Angulo)
        self.Btn_Calcular.setGeometry(QtCore.QRect(220, 20, 75, 23))
        self.Btn_Calcular.setObjectName("Btn_Calcular")
        self.txt_angulo = QtWidgets.QLineEdit(Form_Angulo)
        self.txt_angulo.setGeometry(QtCore.QRect(100, 20, 113, 20))
        self.txt_angulo.setObjectName("txt_angulo")

        self.retranslateUi(Form_Angulo)
        QtCore.QMetaObject.connectSlotsByName(Form_Angulo)

        self.Btn_Calcular.clicked.connect(self.Calcular)

    def retranslateUi(self, Form_Angulo):
        _translate = QtCore.QCoreApplication.translate
        Form_Angulo.setWindowTitle(_translate("Form_Angulo", "Dialog"))
        self.lblAngulo.setText(_translate("Form_Angulo", "Digite o ângulo"))
        self.lblSeno.setText(_translate("Form_Angulo", "O seno é:"))
        self.lblCos.setText(_translate("Form_Angulo", "O cosseno é:"))
        self.Btn_Calcular.setText(_translate("Form_Angulo", "Calcular"))


if __name__ == "__main__":
    import sys
    app = QtWidgets.QApplication(sys.argv)
    Form_Angulo = QtWidgets.QDialog()
    ui = Ui_Form_Angulo()
    ui.setupUi(Form_Angulo)
    Form_Angulo.show()
    sys.exit(app.exec_())
```

Programa Rodando

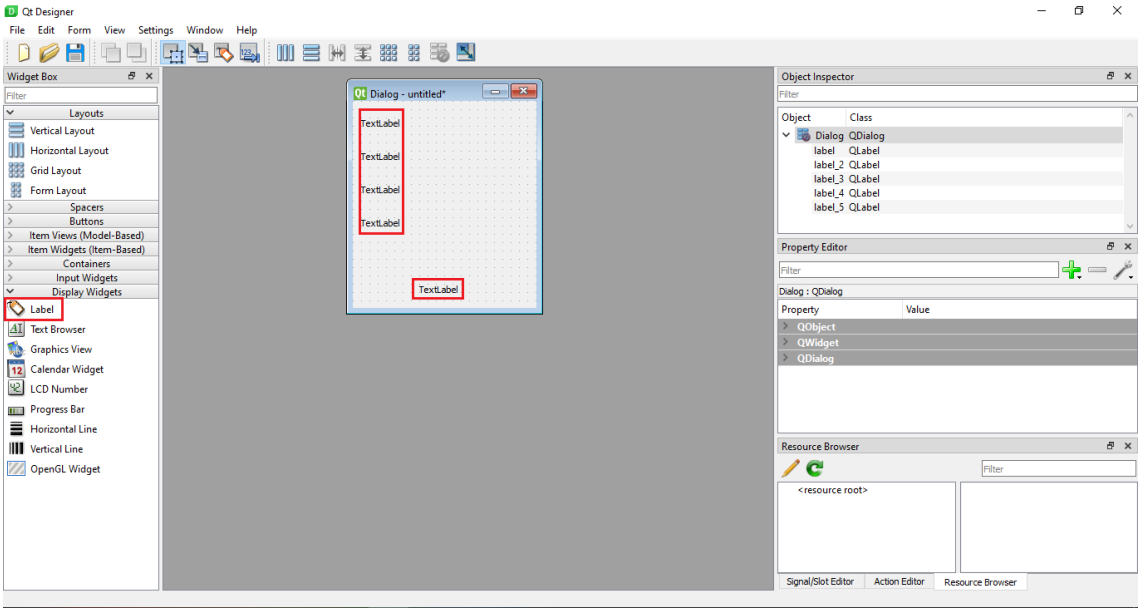
Visual do programa para o usuário.



2° Programa: Aplicação que calcula a média e verificação da situação do usuário.

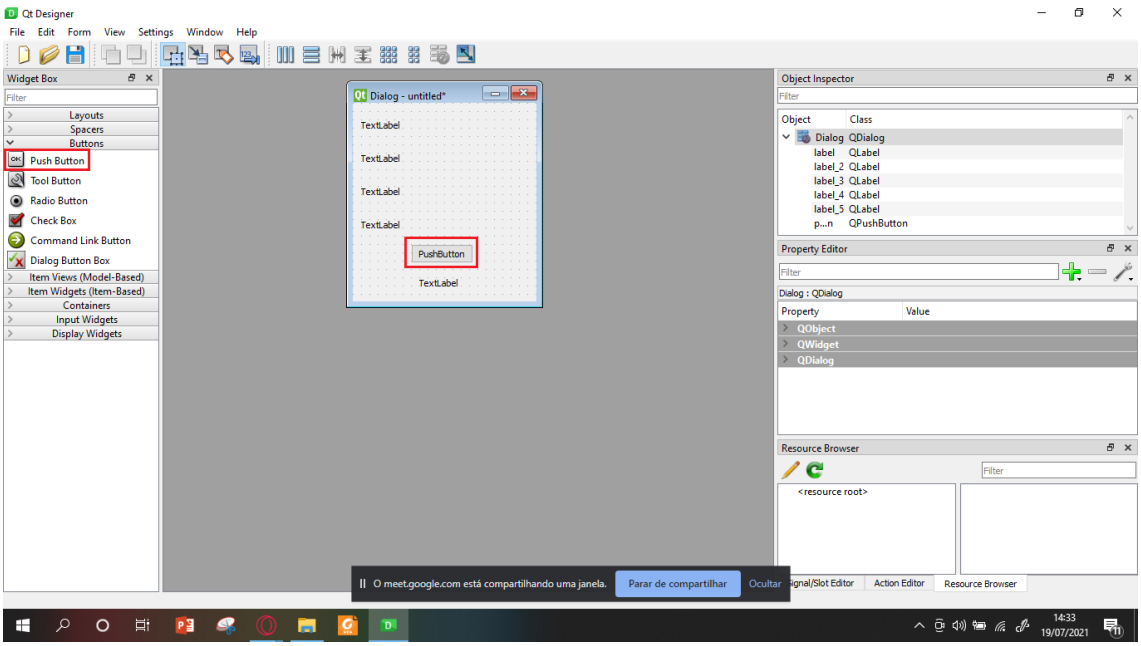
Passo 1: Construindo o formulário

Adicionando *Label's*



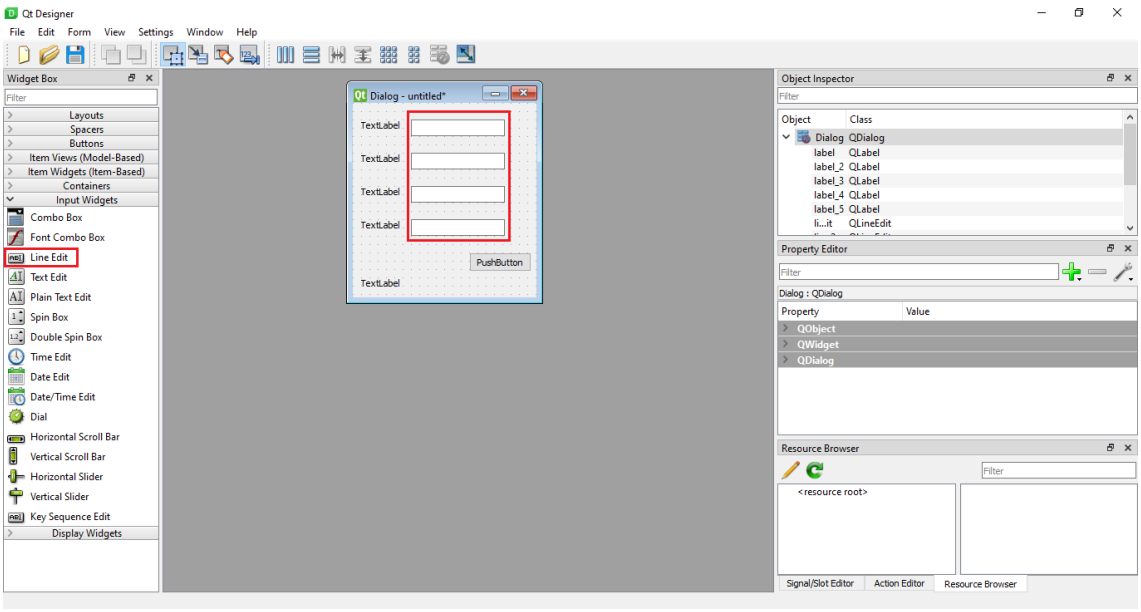
A princípio, clicando no ícone *Label* do menu disposto na lateral esquerda, foram adicionadas 5 *label's.*

Adicionando *Push Button*



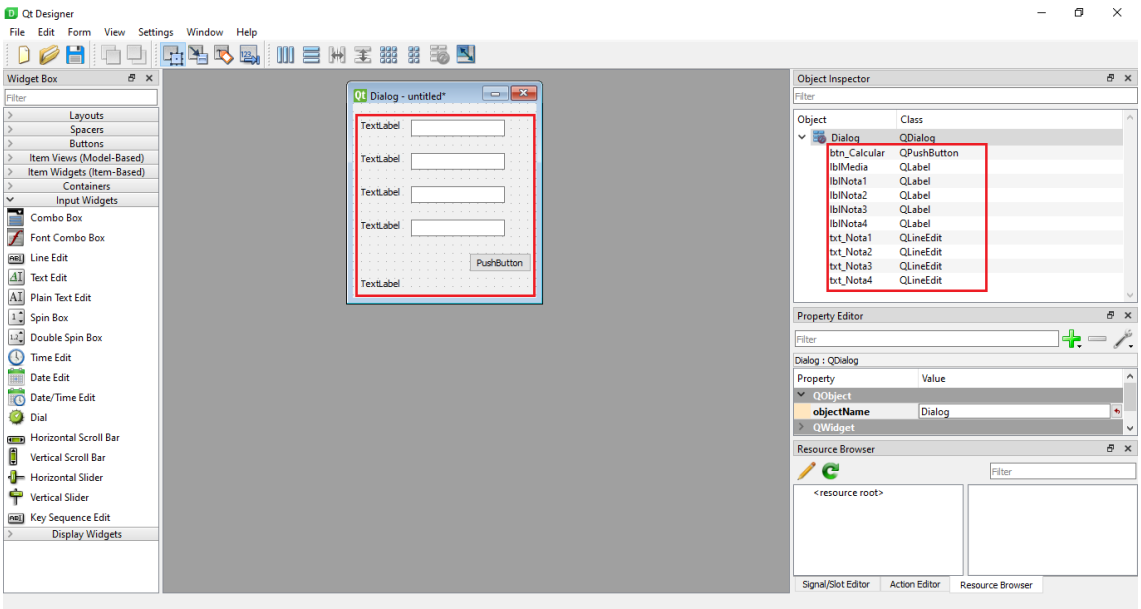
Após isto, foi adicionado um Push Button.

Adicinando *Line Edit's*



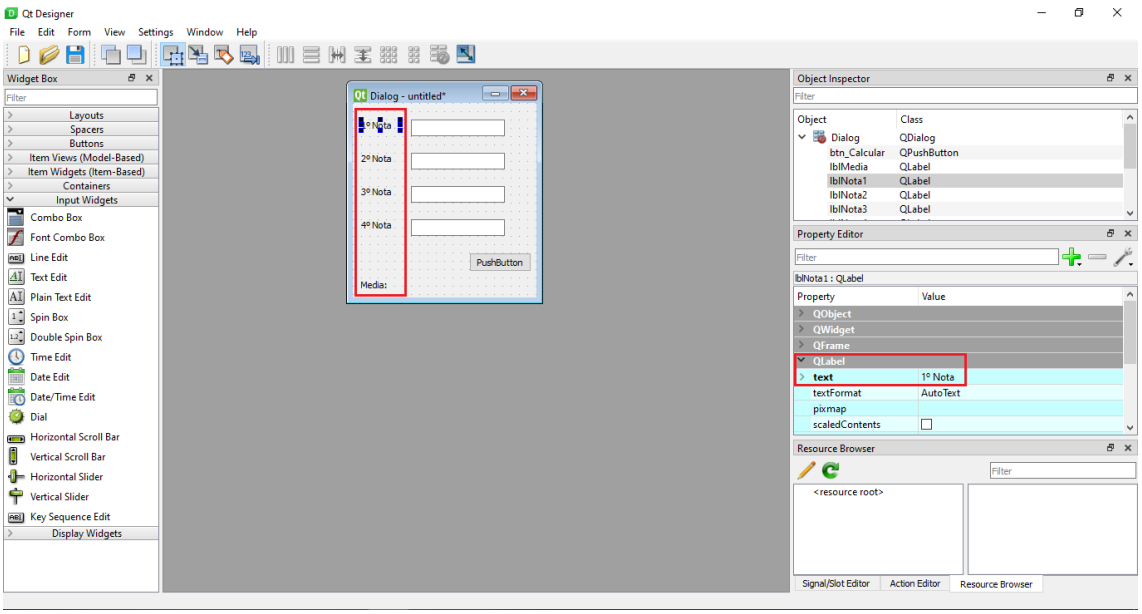
Seguindo na construção do formulário, foram adicionados 4 *Line Edit's*.

Identificando Função



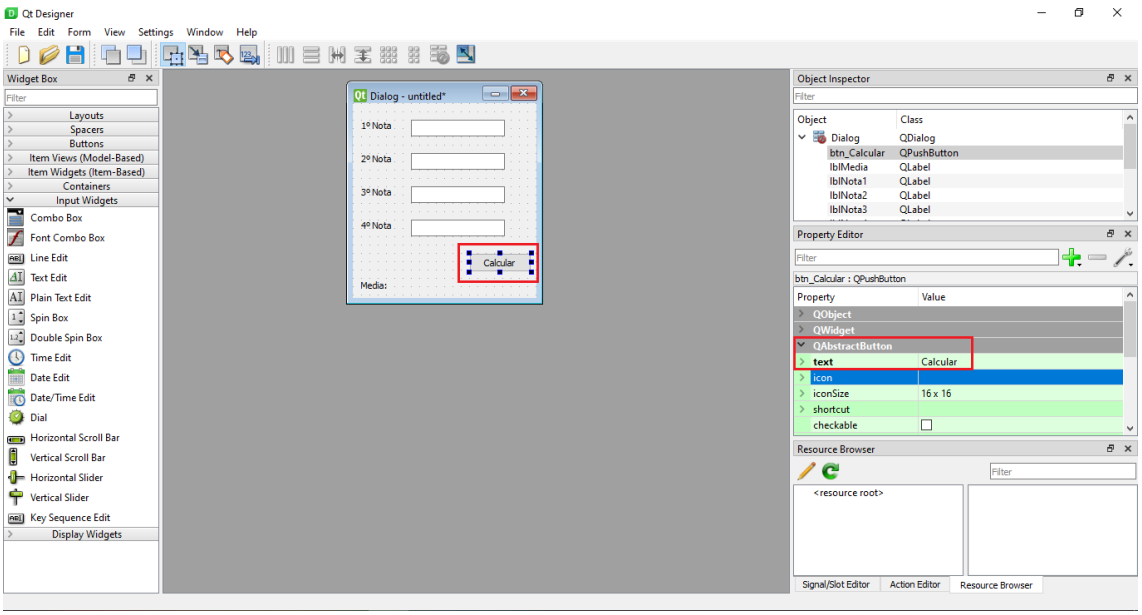
Em seguida, foi feita uma identificação dos componentes adicionados no formulário.

Adicionando texto nas Label's



Clicando no ícone *QLabel* do menu disposto na lateral direita, foi alterado o componente *text.*

Adicionando texto no *Push Button*



Seguindo na alteração do componente *text*, foi clicado no Ícone QAbstractButton, onde foi alterado o conteúdo que ficaria visível para o usuário.

Passo 2: Construindo a lógica da aplicação

Transformando ui em py

```
C:\Users\User\Desktop\Tentantiva>pyuic5 -x FazendoProg2.ui -o Prog2.py
```

Utilizando de conceitos já explicados anteriormente neste capítulo, foi feito uma transformação do arquivo.

Criando um Método

```
def CalcularMed (self):
    Nota1 = float(self.txt_Nota1.text())
    Nota2 = float(self.txt_Nota2.text())
    Nota3 = float(self.txt_Nota3.text())
    Nota4 = float(self.txt_Nota4.text())
    Media = (Nota1 + Nota2 + Nota3 + Nota4) / 4
    if Media > 6:
        self.lblMedia.setText("Sua média é " + str(Media) + " - Você esta aprovado!")
    elif Media < 3:
        self.lblMedia.setText("Sua média é " + str(Media) + " - Você esta reprovado!")
    else:
        self.lblMedia.setText("Sua média é " + str(Media) + " - Você esta de exame...")
```

Na parte da programação foi criado o método *CalcularMed()*, que terá a mesma lógica deste mesma aplicação apresentada no capítulo anterior, utilizando a mesma estrutura condicional if-elif-else , havendo apenas mudanças de sintaxe por estar sendo feita no modo gráfico.

Chamando o Método

```
self.btn_Calcular.clicked.connect(self.CalcularMed)
```

Após a construção do método, é programado quando o mesmo deve ser chamado. Neste caso, sempre que o usuário clicar no botão para calcular.

Programa Inteiro

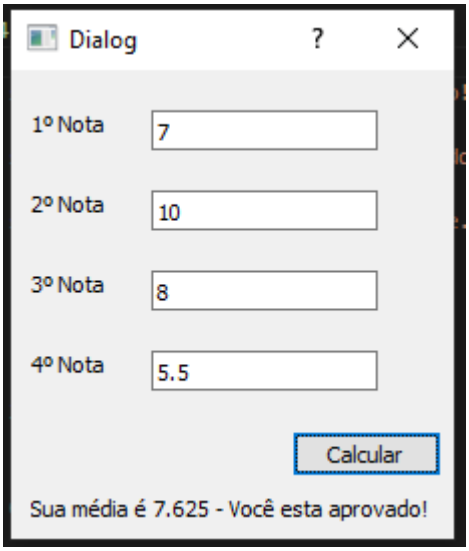
Visual do programa inteiro, juntando o que foi importado da ui.

```
class Ui_Dialog(object):
    def CalcularMed (self):
        Nota1 = float(self.txt_Nota1.text())
        Nota2 = float(self.txt_Nota2.text())
        Nota3 = float(self.txt_Nota3.text())
        Nota4 = float(self.txt_Nota4.text())
        Media = (Nota1 + Nota2 + Nota3 + Nota4) / 4
        if Media > 6:
            self.lblMedia.setText("Sua média é " + str(Media) + " - Você esta aprovado!")
        elif Media < 3:
            self.lblMedia.setText("Sua média é " + str(Media) + " - Você esta reprovado!")
        else:
            self.lblMedia.setText("Sua média é " + str(Media) + " - Você esta de exame...")

    def setupUi(self, Dialog):
        Dialog.setObjectName("Dialog")
        Dialog.resize(221, 234)
        self.lblNota1 = QtWidgets.QLabel(Dialog)
        self.lblNota1.setGeometry(QtCore.QRect(10, 20, 47, 13))
        self.lblNota1.setObjectName("lblNota1")
        self.lblNota2 = QtWidgets.QLabel(Dialog)
        self.lblNota2.setGeometry(QtCore.QRect(10, 60, 47, 13))
        self.lblNota2.setObjectName("lblNota2")
        self.lblNota3 = QtWidgets.QLabel(Dialog)
        self.lblNota3.setGeometry(QtCore.QRect(10, 100, 47, 13))
        self.lblNota3.setObjectName("lblNota3")
        self.lblNota4 = QtWidgets.QLabel(Dialog)
        self.lblNota4.setGeometry(QtCore.QRect(10, 140, 47, 13))
        self.lblNota4.setObjectName("lblNota4")
        self.lblMedia = QtWidgets.QLabel(Dialog)
        self.lblMedia.setGeometry(QtCore.QRect(10, 210, 200, 16))
        self.lblMedia.setObjectName("lblMedia")
        self.btn_Calcular = QtWidgets.QPushButton(Dialog)
        self.btn_Calcular.setGeometry(QtCore.QRect(140, 180, 75, 23))
        self.btn_Calcular.setObjectName("btn_Calcular")
        self.txt_Nota1 = QtWidgets.QLineEdit(Dialog)
        self.txt_Nota1.setGeometry(QtCore.QRect(70, 20, 113, 20))
        self.txt_Nota1.setObjectName("txt_Nota1")
        self.txt_Nota2 = QtWidgets.QLineEdit(Dialog)
        self.txt_Nota2.setGeometry(QtCore.QRect(70, 60, 113, 20))
        self.txt_Nota2.setObjectName("txt_Nota2")
        self.txt_Nota3 = QtWidgets.QLineEdit(Dialog)
        self.txt_Nota3.setGeometry(QtCore.QRect(70, 100, 113, 20))
        self.txt_Nota3.setObjectName("txt_Nota3")
        self.txt_Nota4 = QtWidgets.QLineEdit(Dialog)
        self.txt_Nota4.setGeometry(QtCore.QRect(70, 140, 113, 20))
        self.txt_Nota4.setObjectName("txt_Nota4")
```

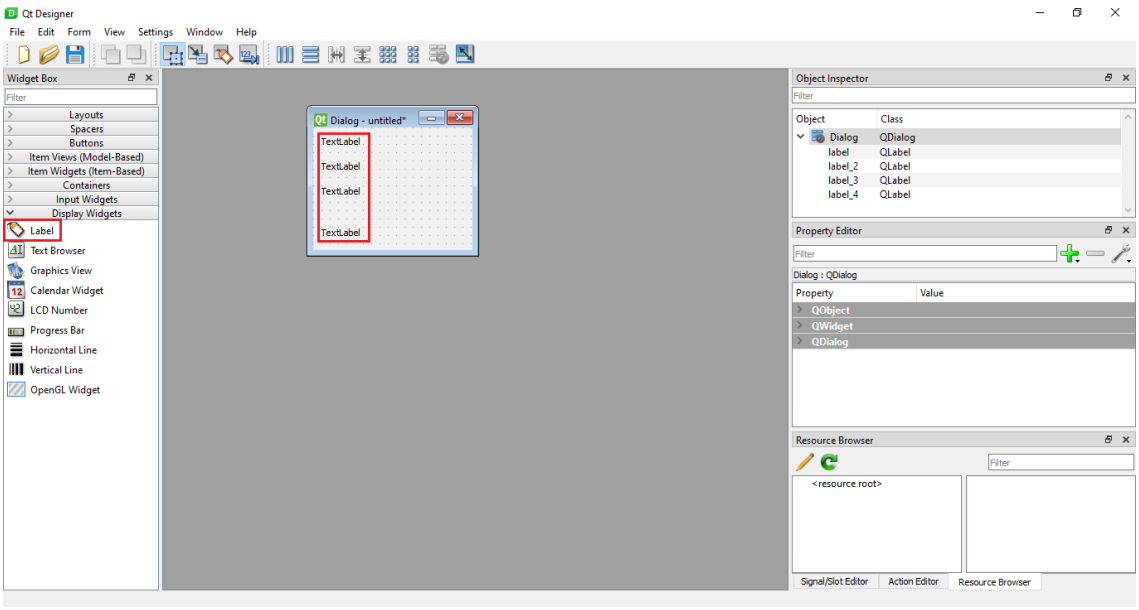
```
        self.retranslateUi(Dialog)
        QtCore.QMetaObject.connectSlotsByName(Dialog)

        self.btn_Calcular.clicked.connect(self.CalcularMed)

    def retranslateUi(self, Dialog):
        _translate = QtCore.QCoreApplication.translate
        Dialog.setWindowTitle(_translate("Dialog", "Dialog"))
        self.lblNota1.setText(_translate("Dialog", "1º Nota"))
        self.lblNota2.setText(_translate("Dialog", "2º Nota"))
        self.lblNota3.setText(_translate("Dialog", "3º Nota"))
        self.lblNota4.setText(_translate("Dialog", "4º Nota"))
        self.lblMedia.setText(_translate("Dialog", "Média: "))
        self.btn_Calcular.setText(_translate("Dialog", "Calcular"))


if __name__ == "__main__":
    import sys
    app = QtWidgets.QApplication(sys.argv)
    Dialog = QtWidgets.QDialog()
    ui = Ui_Dialog()
    ui.setupUi(Dialog)
    Dialog.show()
    sys.exit(app.exec_())
```

Programa Rodando

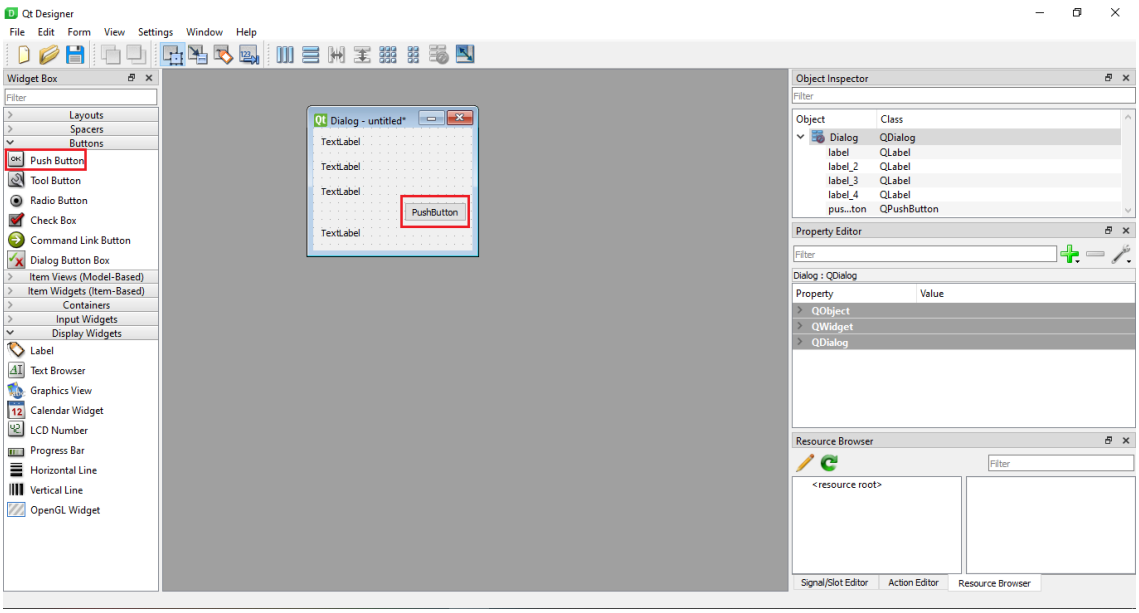
Conteúdo visível para o usuário.



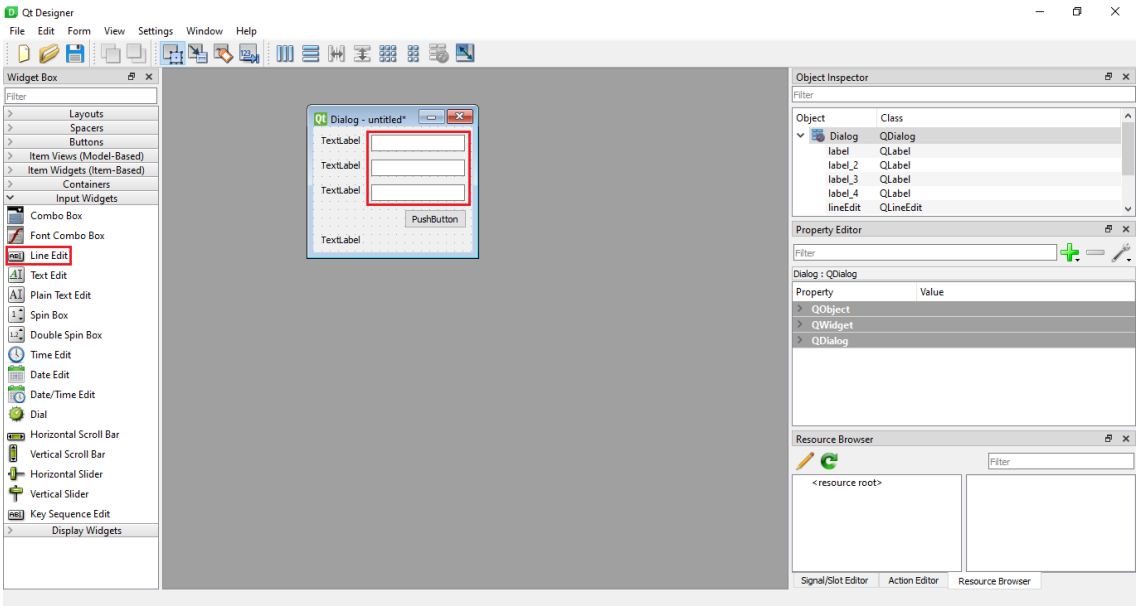
3° Programa: Criando uma calculadora



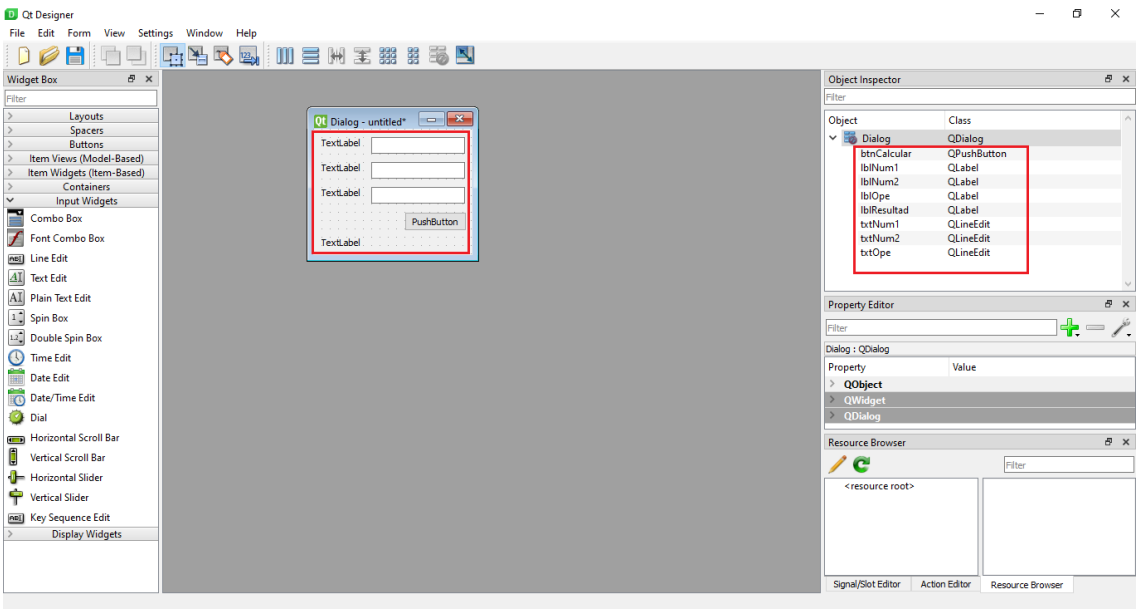
Para começarmos a interface, definimos ***"labels"*** básicas, na aba de ***"Display Widgets"***, apenas para definir onde ficará cada coisa.



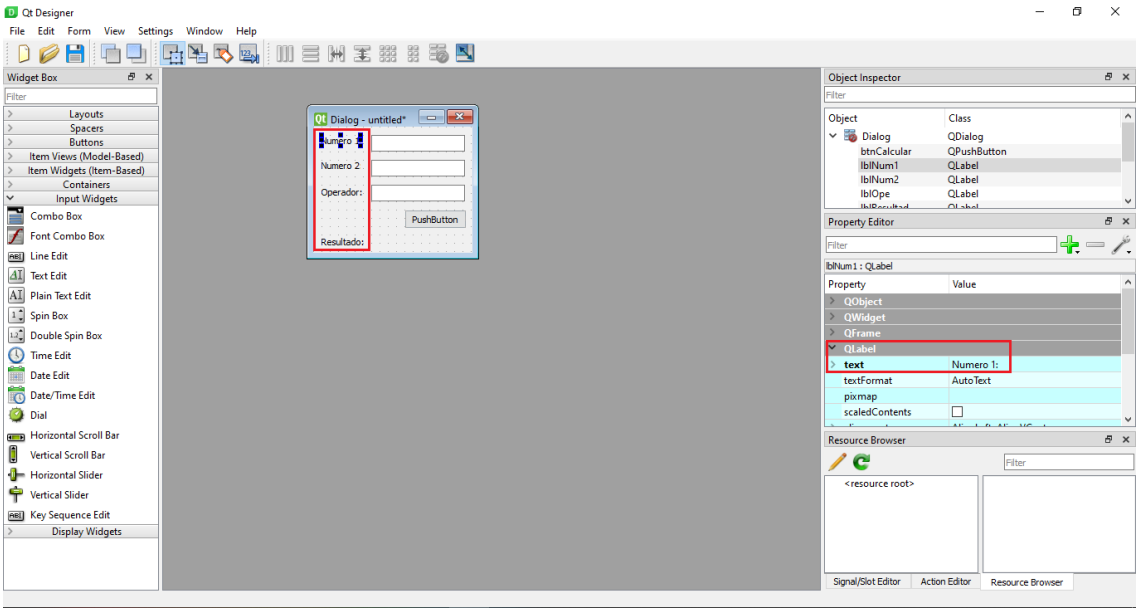
Logo depois, adicionamos um botão ou ***"Push Button"***, na aba de ***"Buttons"***, apenas como uma futura forma de confirmar os dados digitados.



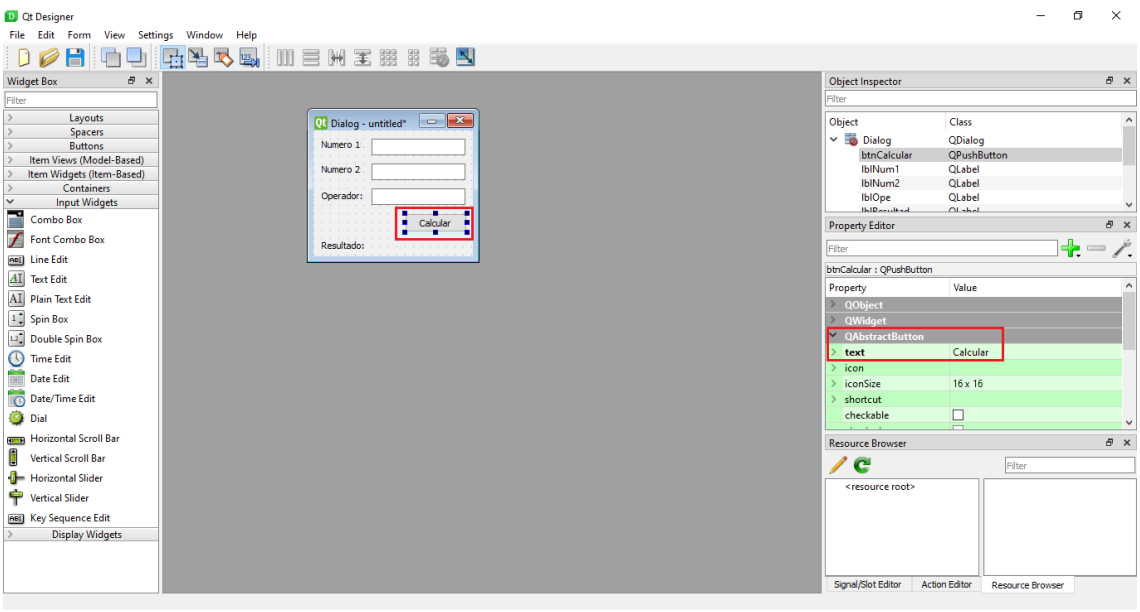
Para inserir os dados, usamos o ***"Line Edit"***, presente na aba de ***"Input Widgets"***, assim deixando claro onde os dados vão ficar.



Antes de continuar com a formatação, precisa-se nomear cada um desses elementos, para que fiquem com uma fácil detecção dentro do código, ou seja, para que fique fácil de ver onde está cada coisa, seja lá para arrumar um problema ou até mesmo só como mérito de organização.



Com tudo devidamente nomeado, podemos mudar a ***"string"*** das label ou simplesmente o que fica escrito na label, assim, dando um significado real e visual para cada uma. Aqui deixamos na ordem de Número 1, Número 2, Operador e o Resultado. Pode-se realizar tal tarefa apenas clicando em cima e vendo seus dados ao lado, assim como na imagem.



Assim como o último passo das *Labels*, pode-se realizar a mesma tarefa com o botão, apenas como mérito de organização.

```
C:\Users\User\Desktop\Tentantiva>pyuic5 -x FazendoProg3.ui -o Prog3.py
```

Ao salvar, o seu arquivo ficará salvo como um arquivo ***".ui"***. Para converte-lo para o formato que queremos, que no caso é ***".py"***, precisamos deste comando acima.

```
class Ui_Dialog(object):
    def Calcular(self):
        if self.txtOpe.text() == "+":

            self.lblResultad.setText("Sua adição deu: " + str(float(self.txtNum1.text()) + float(self.txtNum2.text())))

        elif self.txtOpe.text() == "-":

            self.lblResultad.setText("Sua subtração deu: " + str(float(self.txtNum1.text()) - float(self.txtNum2.text())))

        elif self.txtOpe.text() == "*":

            self.lblResultad.setText("Sua multiplicação deu: " + str(float(self.txtNum1.text()) * float(self.txtNum2.text())))

        elif self.txtOpe.text() == "/":

            self.lblResultad.setText("Sua divisão deu: " + str(float(self.txtNum1.text()) / float(self.txtNum2.text())))

    def setupUi(self, Dialog):
        Dialog.setObjectName("Dialog")
        Dialog.resize(192, 152)
        self.lblNum1 = QtWidgets.QLabel(Dialog)
        self.lblNum1.setGeometry(QtCore.QRect(10, 10, 47, 13))
        self.lblNum1.setObjectName("lblNum1")
        self.lblNum2 = QtWidgets.QLabel(Dialog)
        self.lblNum2.setGeometry(QtCore.QRect(10, 40, 47, 13))
        self.lblNum2.setObjectName("lblNum2")
        self.lblOpe = QtWidgets.QLabel(Dialog)
        self.lblOpe.setGeometry(QtCore.QRect(10, 70, 51, 16))
        self.lblOpe.setObjectName("lblOpe")
        self.lblResultad = QtWidgets.QLabel(Dialog)
        self.lblResultad.setGeometry(QtCore.QRect(10, 130, 181, 16))
        self.lblResultad.setObjectName("lblResultad")
        self.btnCalcular = QtWidgets.QPushButton(Dialog)
        self.btnCalcular.setGeometry(QtCore.QRect(110, 100, 75, 23))
        self.btnCalcular.setObjectName("btnCalcular")
        self.txtNum1 = QtWidgets.QLineEdit(Dialog)
        self.txtNum1.setGeometry(QtCore.QRect(70, 10, 113, 20))
        self.txtNum1.setObjectName("txtNum1")
        self.txtNum2 = QtWidgets.QLineEdit(Dialog)
        self.txtNum2.setGeometry(QtCore.QRect(70, 40, 113, 20))
        self.txtNum2.setObjectName("txtNum2")
        self.txtOpe = QtWidgets.QLineEdit(Dialog)
        self.txtOpe.setGeometry(QtCore.QRect(70, 70, 113, 20))
        self.txtOpe.setObjectName("txtOpe")

        self.retranslateUi(Dialog)
        QtCore.QMetaObject.connectSlotsByName(Dialog)
```

Ao abrir o arquivo, você irá se deparar com essas linhas de código, porém a parte que você irá programar é apenas a parte lógica, ou seja, os cálculos com operadores.

```
def Calcular(self):
    if self.txtOpe.text() == "+":

        self.lblResultad.setText("Sua adição deu: " + str(float(self.txtNum1.text()) + float(self.txtNum2.text())))

    elif self.txtOpe.text() == "-":

        self.lblResultad.setText("Sua subtração deu: " + str(float(self.txtNum1.text()) - float(self.txtNum2.text())))

    elif self.txtOpe.text() == "*":

        self.lblResultad.setText("Sua multiplicação deu: " + str(float(self.txtNum1.text()) * float(self.txtNum2.text())))

    elif self.txtOpe.text() == "/":

        self.lblResultad.setText("Sua divisão deu: " + str(float(self.txtNum1.text()) / float(self.txtNum2.text())))
```

Aqui é a parte dos cálculos, começando com um ***"if"*** para detectar qual operador está sendo usado na situação, cada operador tendo sua variação, soma (+), subtração (-), multiplicação (*) e divisão (/), por fim, ao realizar o cálculo, exibirá a mensagem da determinada operação. Pode-se observar que estamos usando os dados inseridos nas caixas de textos que colocamos na parte visual ***"txtNum1"*** e ***"txtNum2"***.

```
        self.btnCalcular.clicked.connect(self.Calcular)

    def retranslateUi(self, Dialog):
        _translate = QtCore.QCoreApplication.translate
        Dialog.setWindowTitle(_translate("Dialog", "Dialog"))
        self.lblNum1.setText(_translate("Dialog", "Numero 1:"))
        self.lblNum2.setText(_translate("Dialog", "Numero 2:"))
        self.lblOpe.setText(_translate("Dialog", "Operador:"))
        self.lblResultad.setText(_translate("Dialog", "Resultado: "))
        self.btnCalcular.setText(_translate("Dialog", "Calcular"))


if __name__ == "__main__":
    import sys
    app = QtWidgets.QApplication(sys.argv)
    Dialog = QtWidgets.QDialog()
    ui = Ui_Dialog()
    ui.setupUi(Dialog)
    Dialog.show()
    sys.exit(app.exec_())
```

Aqui temos a continuação do programa, mas o que realmente importa é a parte do cálculo, ou seja, a operação em si.

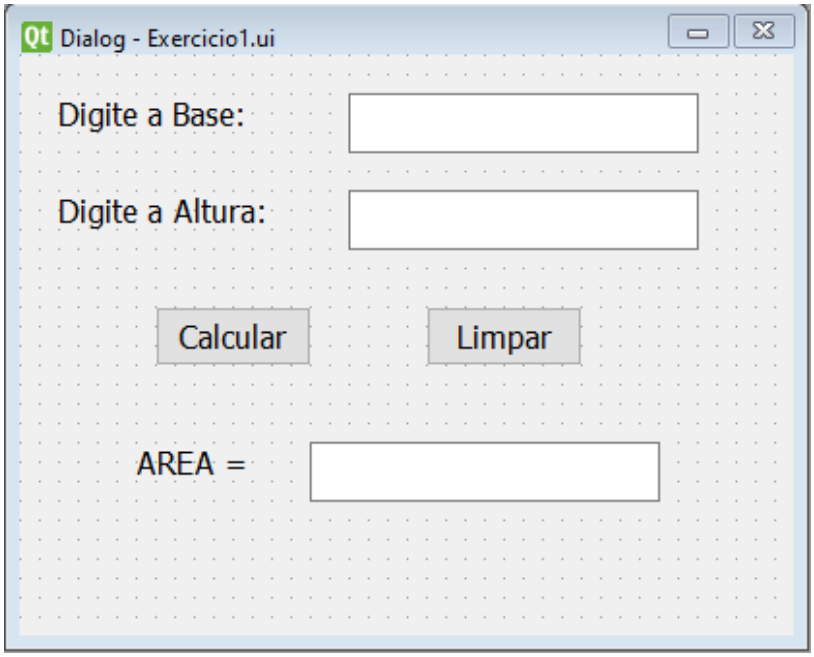
```
self.btnCalcular.clicked.connect(self.Calcular)
```

Essa linha define que, quando pressionado o botão, o cálculo seja feito e todo o resto seja efetivamente realizado, assim mostrando tudo corretamente no fim.

Capítulo 4 – Exemplos de programas com a interface gráfica

1 – Calcular a área do triângulo

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



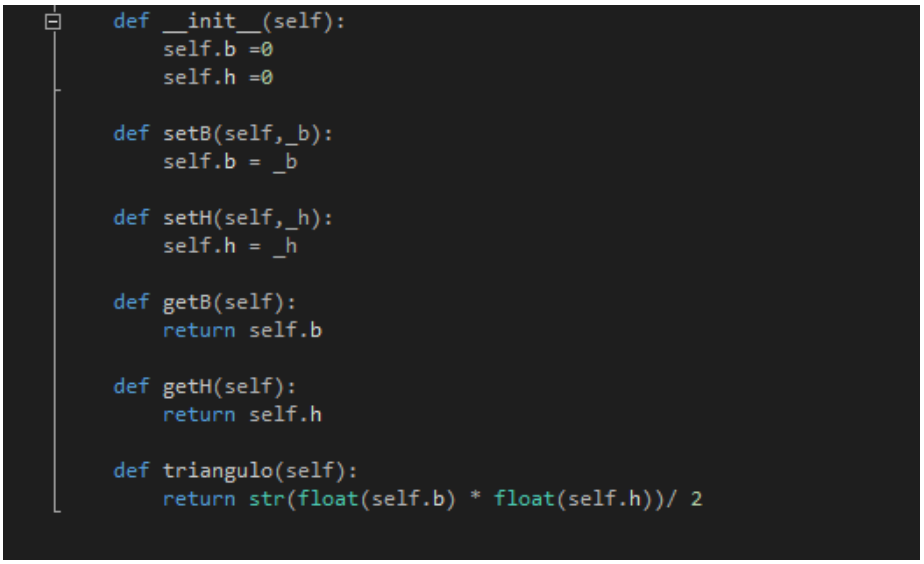
Primeiramente, o formulário foi construído adicionando *Label's*, *Line Edit's* e *Push Button's*. Em seguida, foram adicionados textos para cada uma das *Label's* e dos *Push Button's*, a fim de facilitar a compreensão por parte do usuário. Também foi feita a identificação de cada um dos componentes adicionados no formulário, no campo "*Dialog*", para que cada componente pudesse ser identificado durante a programação.

Após a construção do formulário, o arquivo do mesmo foi convertido para ".py", para que pudesse ser utilizado na IDE escolhida.

Classe Triângulo

```
class triangulo():
```

Dando início à parte de programação, foi criada a classe triangulo().



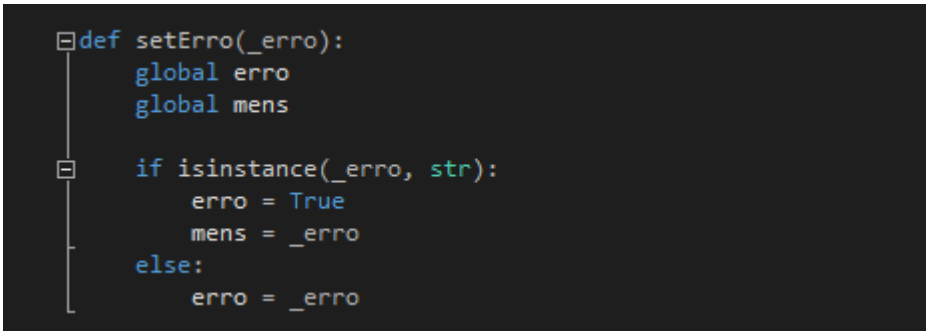
```
def __init__(self):
    self.b =0
    self.h =0

def setB(self,_b):
    self.b = _b

def setH(self,_h):
    self.h = _h

def getB(self):
    return self.b

def getH(self):
    return self.h

def triangulo(self):
    return str(float(self.b) * float(self.h))/ 2
```

Nesta classe foram criados os métodos: _init_(self), que inicializa as variáveis, atribuindo a elas o valor “0”; setB(self, _b), que define a variável b como receptora do parâmetro _b (base); setH(self, _h), que define a variável h como receptora do parâmetro _h (altura); getB(self), que retorna o valor da variável b; getH(self), retorna o valor da variável h; e triangulo(self), que, utilizando os valores atribuídos à b e h, calculam a área do triangulo e retornam o resultado do cálculo.

Classe Erro

```
erro = False
mens = ''
```

Nesta classe, a princípio foi atribuído “false” à variável erro e uma string vazia à variável mens.



```
def setErro(_erro):
    global erro
    global mens

    if isinstance(_erro, str):
        erro = True
        mens = _erro
    else:
        erro = _erro
```

Em seguida, foi desenvolvido o método setErro(_erro), no qual foram declaradas as variáveis globais erro e mens. Este tem como objetivo verificar se ocorre alguma digitação indevida durante a inserção dos valores da base e da altura. Caso seja identificado alguma digitação indevida, a variável erro recebe true e mens recebe _erro.

```
    def getErro():
        return erro

    def getMens():
        return mens
```

Os métodos getErro() e getMens() têm como objetivo retornar, respectivamente, as variáveis erro e mens.

BLL Triangulo

```
import Erro


def validaDados(triangulo):

    Erro.setErro(False)

    if len(triangulo.getB())==0:
        Erro.setErro("A base tem que ter preenchimento obrigatório.")
        return None
    else:
        try:
            float(triangulo.getB())
        except:
            Erro.setErro("A base deve ser numérico.")
            return None

    if len(triangulo.getH())==0:
        Erro.setErro("A altura tem que ter preenchimento obrigatório.")
        return None
    else:
        try:
            float(triangulo.getH())
        except:
            Erro.setErro("A altura deve ser numérico.")
            return None
```

Ao iniciar a BLL a classe Erro foi importada e foi criado o método validaDados(triangulo), que tem como objetivo identificar se houve algum erro. Em caso de erro, através de um laço de decisão, é feita a verificação do tipo de erro, que pode ser: base não preenchida, base com valor não numérico, altura não preenchida e altura com valor não

numérico. Após a verificação do erro, é retornada ao usuário uma mensagem comunicando a existência da falha.

IHM Triangulo

```
# -*- coding: utf-8 -*-

# Form implementation generated from reading ui file 'Exercicio1.ui'
#
# Created by: PyQt5 UI code generator 5.15.4
#
# WARNING: Any manual changes made to this file will be lost when pyuic5 is
# run again.  Do not edit this file unless you know what you are doing.


from PyQt5 import QtCore, QtGui, QtWidgets
from Triangulo import triangulo
import TrianguloBLL
import Erro
```

Nesta classe, foi construída a interação entre o formulário e o programa. Primeiramente, são importadas as classes criadas anteriormente.

```
class Ui_Dialog(object):
    def setupUi(self, Dialog):
        Dialog.setObjectName("Dialog")
        Dialog.resize(278, 253)
        self.pushButton = QtWidgets.QPushButton(Dialog)
        self.pushButton.setGeometry(QtCore.QRect(40, 110, 81, 31))
        font = QtGui.QFont()
        font.setPointSize(12)
        self.pushButton.setFont(font)
        self.pushButton.setObjectName("pushButton")
        self.pushButton_2 = QtWidgets.QPushButton(Dialog)
        self.pushButton_2.setGeometry(QtCore.QRect(160, 110, 81, 31))
        font = QtGui.QFont()
        font.setPointSize(12)
        self.pushButton_2.setFont(font)
        self.pushButton_2.setObjectName("pushButton_2")
        self.label = QtWidgets.QLabel(Dialog)
        self.label.setGeometry(QtCore.QRect(30, 20, 121, 16))
        font = QtGui.QFont()
        font.setPointSize(12)
        self.label.setFont(font)
        self.label.setObjectName("label")
        self.label_2 = QtWidgets.QLabel(Dialog)
        self.label_2.setGeometry(QtCore.QRect(20, 60, 111, 16))
        font = QtGui.QFont()
        font.setPointSize(12)
        self.label_2.setFont(font)
        self.label_2.setObjectName("label_2")
        self.label_3 = QtWidgets.QLabel(Dialog)
        self.label_3.setGeometry(QtCore.QRect(10, 170, 141, 21))
        font = QtGui.QFont()
        font.setPointSize(12)
        self.label_3.setFont(font)
        self.label_3.setObjectName("label_3")
        self.textobase = QtWidgets.QLineEdit(Dialog)
```

```
        self.textobase.setGeometry(QtCore.QRect(130, 20, 141, 20))
        self.textobase.setObjectName("textobase")
        self.textoaltura = QtWidgets.QLineEdit(Dialog)
        self.textoaltura.setGeometry(QtCore.QRect(130, 60, 141, 20))
        self.textoaltura.setObjectName("textoaltura")
        self.areadotriangulo = QtWidgets.QLineEdit(Dialog)
        self.areadotriangulo.setGeometry(QtCore.QRect(160, 170, 113, 20))
        self.areadotriangulo.setObjectName("areadotriangulo")

        self.retranslateUi(Dialog)
        QtCore.QMetaObject.connectSlotsByName(Dialog)
```

A classe Ui_Dialog(object) é onde se encontram especificados cada um dos componentes adicionados no formulário. Esta classe é gerada automaticamente quando se faz a conversão do formulário e o insere-se na IDE escolhida.

```
        self.pushButton.clicked.connect(self.event_calcular)
        self.pushButton_2.clicked.connect(self.Limpar)
```

```
    def retranslateUi(self, Dialog):
        _translate = QtCore.QCoreApplication.translate
        Dialog.setWindowTitle(_translate("Dialog", "Dialog"))
        self.label.setText(_translate("Dialog", "Digite a Base:"))
        self.textobase.setText(_translate("Dialog", "0"))
        self.pushButton.setText(_translate("Dialog", "Calcular"))
        self.label_2.setText(_translate("Dialog", "Digite a Altura:"))
        self.label_3.setText(_translate("Dialog", "Área do triângulo ="))
        self.textoaltura.setText(_translate("Dialog", "0"))
        self.areadotriangulo.setText(_translate("Dialog", "0"))
        self.pushButton_2.setText(_translate("Dialog", "Limpar"))
```

Após a especificação dos componentes, foi feita a conexão dos métodos de eventos aos botões.

```
def Limpar(self):
    _translate = QtCore.QCoreApplication.translate
    self.textobase.setText(_translate("Dialog", "0"))
    self.textoaltura.setText(_translate("Dialog", "0"))
    self.areadotriangulo.setText(_translate("Dialog", "0"))
```

O método Limpar(self) redefine os campos editáveis (Line Edit's) para seus valores iniciais.

```
    def event_calcular (self):
        Calcular = triangulo()

        Calcular.setB(self.textobase.text())
        Calcular.setH(self.textoaltura.text())

        TrianguloBLL.validaDados(triangulo)


        if Erro.getErro():
            msgErro = QtWidgets.QMessageBox()
            msgErro.setText(Erro.getMens())
            msgErro.exec()
        else:
            self.areadotriangulo.setText(triangulo.calcular())



if __name__ == "__main__":
    import sys
```

O método event_calcular(self) faz com que, ao clicar no botão “calcular”, os valores inseridos nas Line Edit’s textobase e textoaltura sejam capturados e, após o tratamento de erros, a área do triangulo seja exibida no campo areadotriangulo.

2 – Calcular Salário Horista

```python
class Salario:
    def _init_(self):
        self.qtd = "0"
        self.valor = "0"

    def setQtd(self, _qtd):
        self.qtd = _qtd

    def setVal(self, _valor):
        self.valor = _valor

    def getQtd(self):
        return self.qtd

    def getVal(self):
        return self.valor

    def getSalarioBruto(self):
        return str((int(self.qtd) * int(self.valor)) / 2)
```

O programa inicia-se criando a classe Salario e definindo o valor de cada aspecto.

```python
erro = False
mens = ''

def setErro(_erro):
    global erro
    global mens

    if isinstance(_erro, str):
        erro = True
        mens = _erro
    else:
        erro = _erro

def getErro():
    return erro

def getMens():
    return mens
```

Então, pode-se definir a mensagem de erro e o que pode causa-la durante o processo.

```
import Erro

def validaDados(Salario):

    Erro.setErro(False)

    if len(Salario.getQtd()) == 0:
        Erro.setErro("O campo QUANTIDADE DE HORAS é de preenchimento obrigatório ... ")
        return None
    else:
        try:
            int(Salario.getQtd())
        except:
            Erro.setErro("O campo QUANTIDADE DE HORAS deve ser numérico ... ")
            return None

    if int(Salario.getQtd()) <= 0:
        Erro.setErro("O campo QUANTIDADE DE HORAS deve ser maior que zero.");
        return None

    if len(Salario.getVal()) == 0:
        Erro.setErro("O campo VALOR DA HORA é de preenchimento obrigatório ... ")
        return None
    else:
        try:
            int(Salario.getVal())
        except:
            Erro.setErro("O campo VALOR DA HORA deve ser numérico ... ")
            return None

    if int(Salario.getVal()) <= 0:
        Erro.setErro("O campo VALOR DA HORA deve ser maior que zero.");
        return None
```

Assim, importa-se o Erro para a validação dos dados usando if e else para exibir as mensagens adequadas.

```
from PyQt5 import QtCore, QtGui, QtWidgets
import Erro
import Ex2BLL
from Ex2 import Salario
```

Para a parte principal, todas as classes previamente abertas são importadas para que se complementem.

```
def Calcular(self):
    msg = QtWidgets.QMessageBox()
    salario = Salario()

    salario.setQtd(self.qtdH.text())
    salario.setVal(self.valH.text())

    Ex2BLL.validaDados(salario)
    if Erro.getErro():
        msg.setText(Erro.getMens())
        msg.exec()
    else:
        self.ValSalario.setText(salario.getSalarioBruto())

        self.qtdH.setEnabled(False)
        self.valH.setEnabled(False)
        self.ValSalario.setEnabled(False)



def Limpar(self):
    self.qtdH.setText("")
    self.valH.setText("")
    self.ValSalario.setText("")

    self.qtdH.setEnabled(True)
    self.valH.setEnabled(True)
    self.ValSalario.setEnabled(True)
```

Aqui, constrói-se os botões de “calcular” e “limpar” funcionem apropriadamente, definindo o que cada um faz.

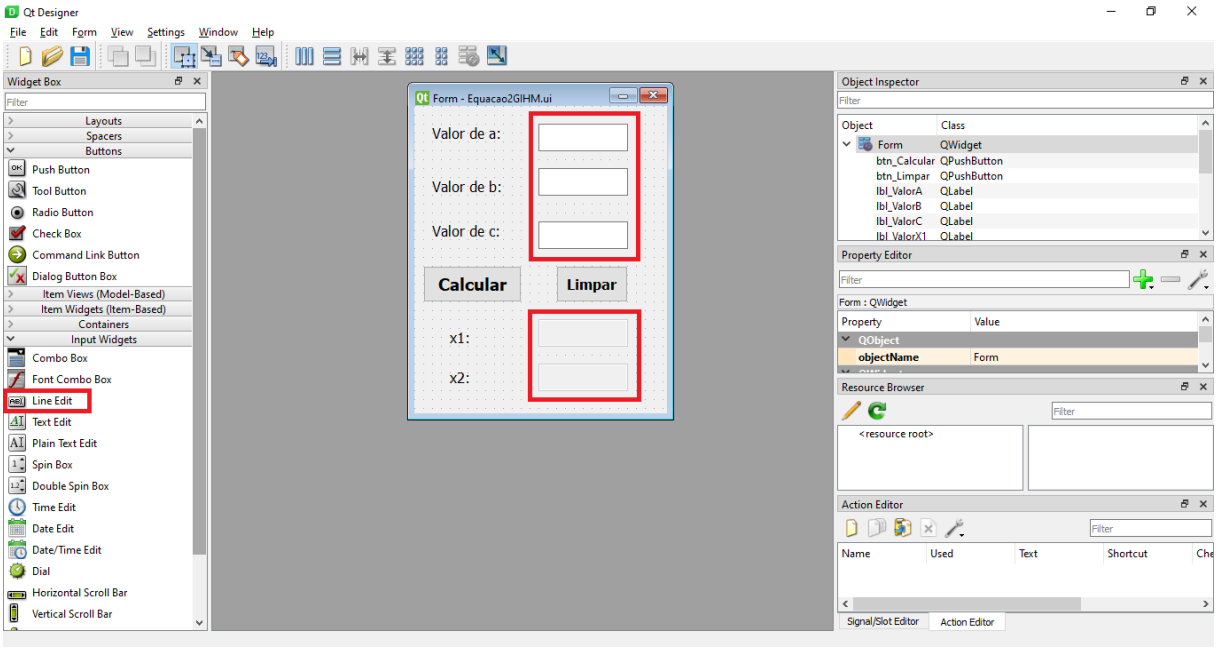
```
self.calcButton.clicked.connect(self.Calcular)
self.cleanButton.clicked.connect(self.Limpar)
```

Finaliza-se conectando a lógica dos botões com os botões propriamente ditos.

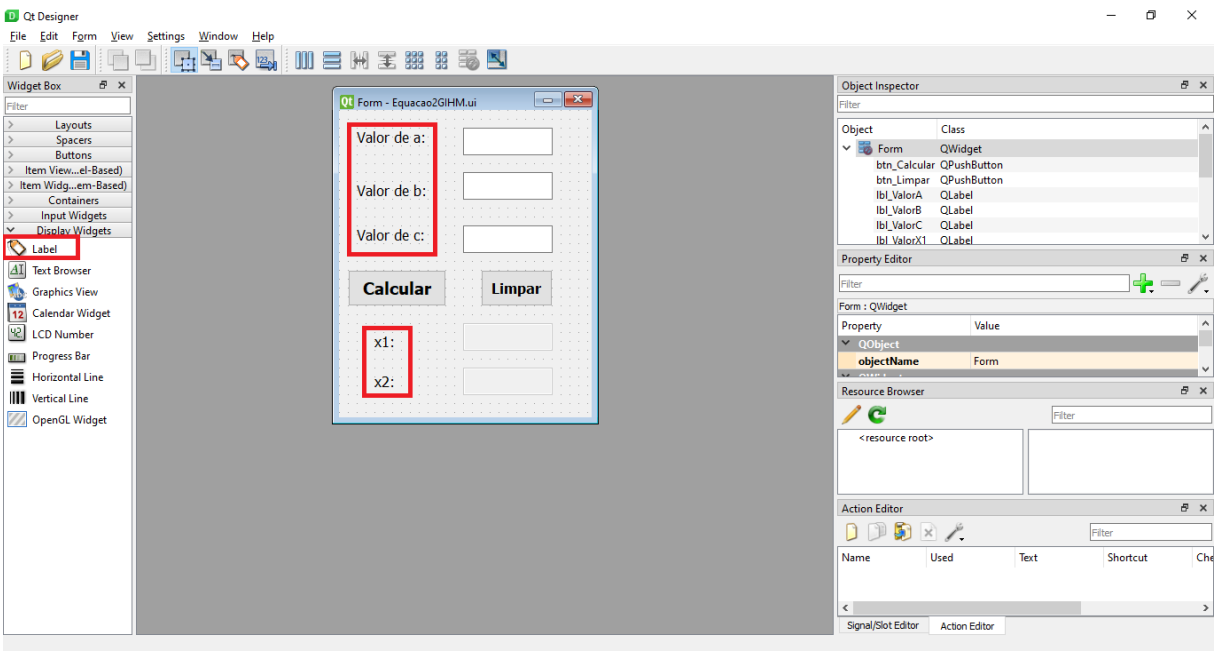
3 – Equação de Segundo Grau

Neste projeto, o PyQt5 Design foi utilizado para o desenvolvimento gráfico da interface do usuário.

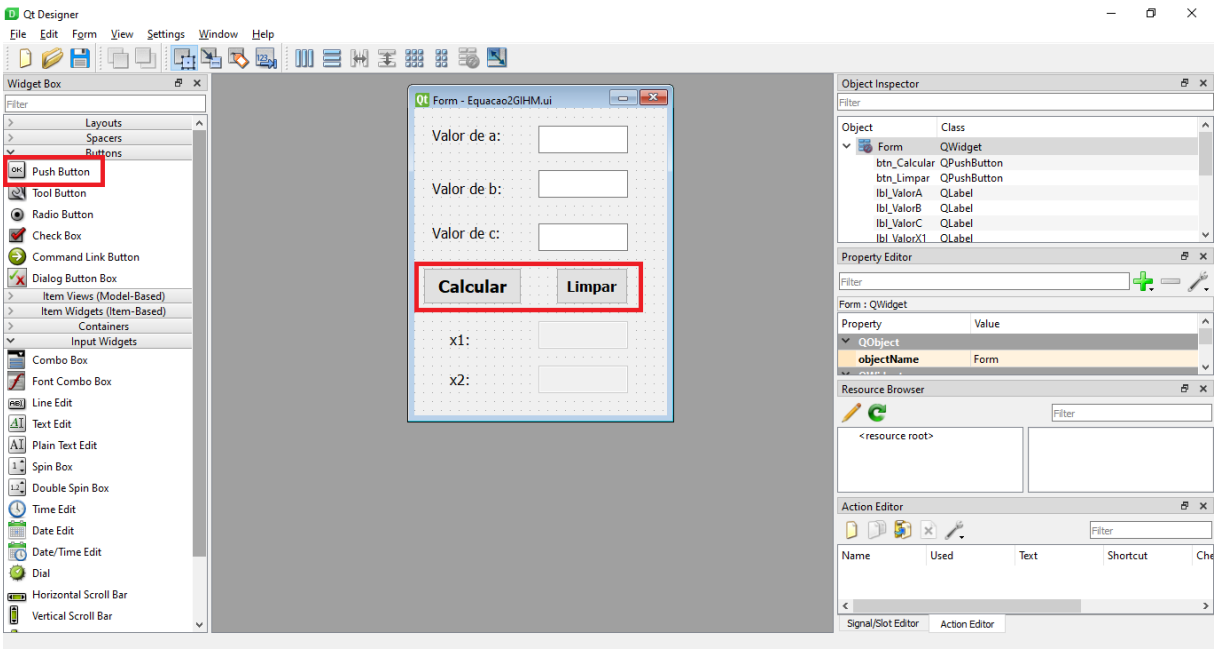
Segue abaixo os componentes necessários para construí-la:



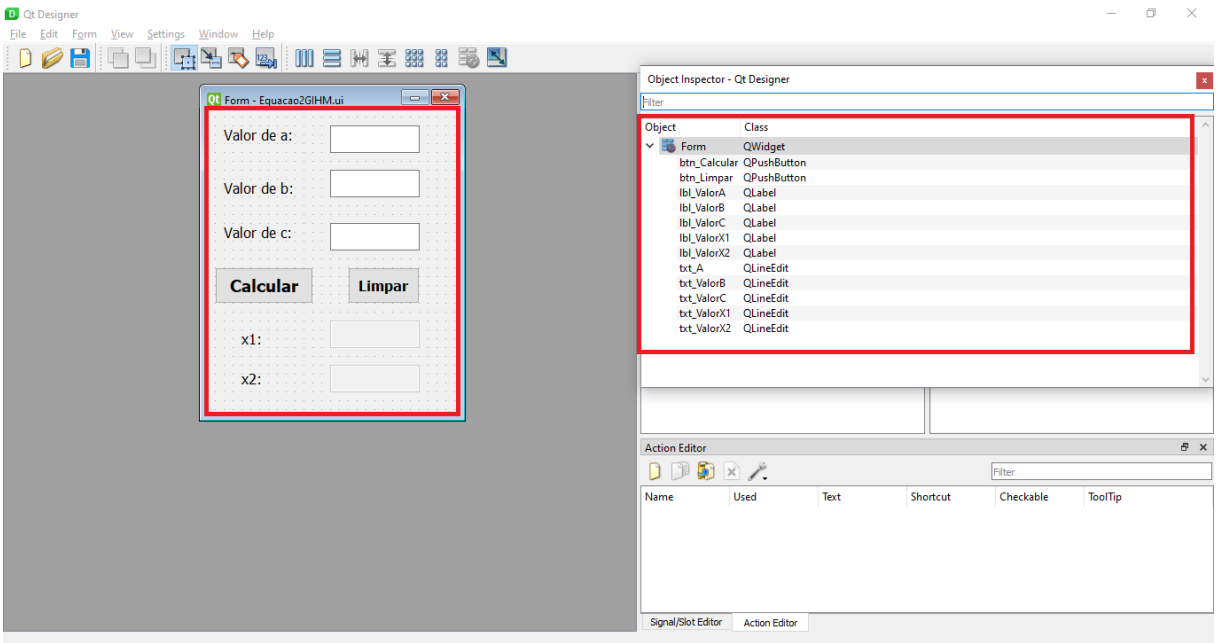
Os "***Line Edit***" foram inseridos para possibilitar a digitação dos números escolhidos pelo usuário e exibir o resultado.



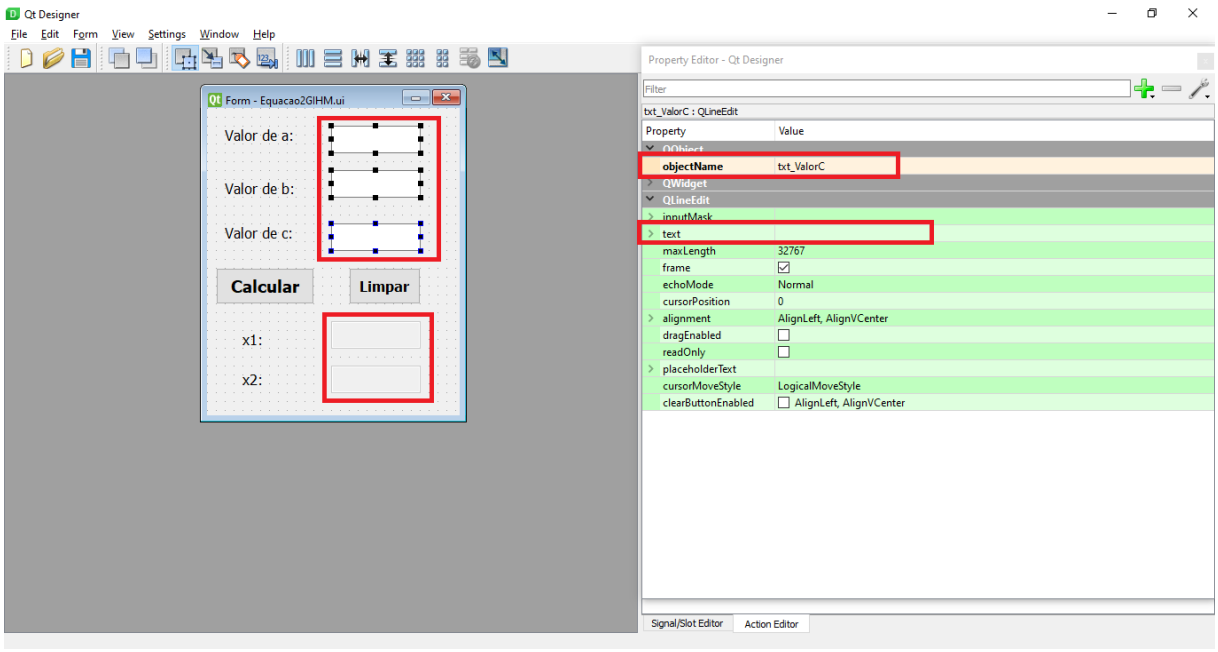
As “***Labels***” foram inseridas com o intuito de identificar cada campo.

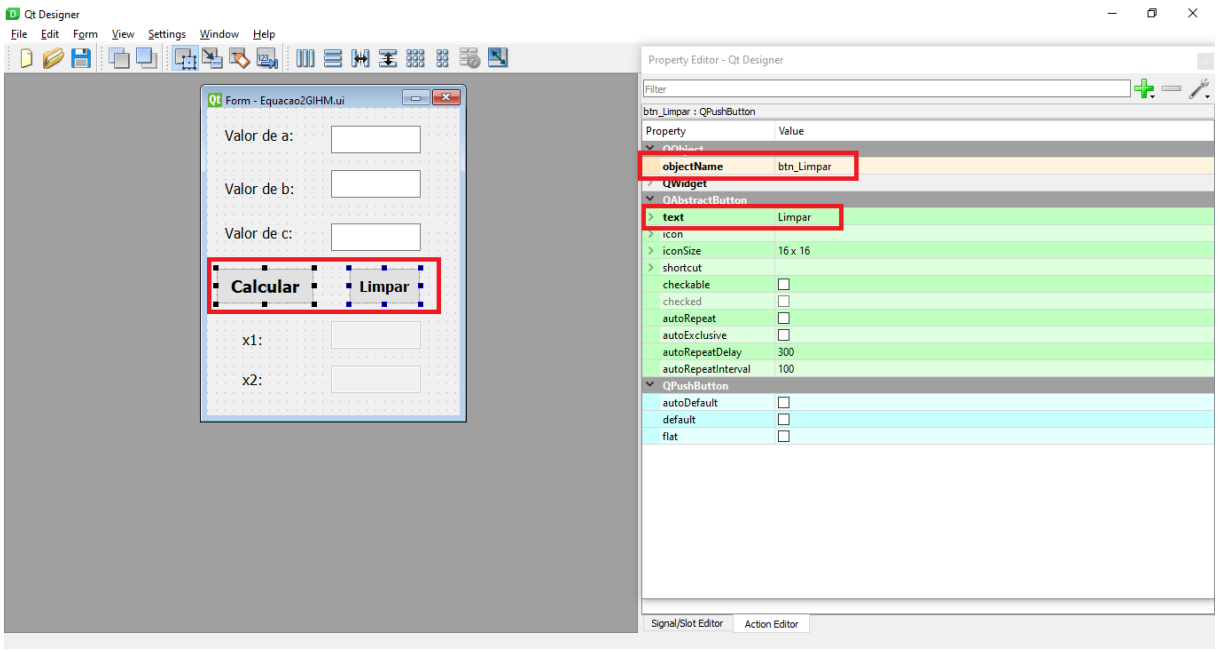


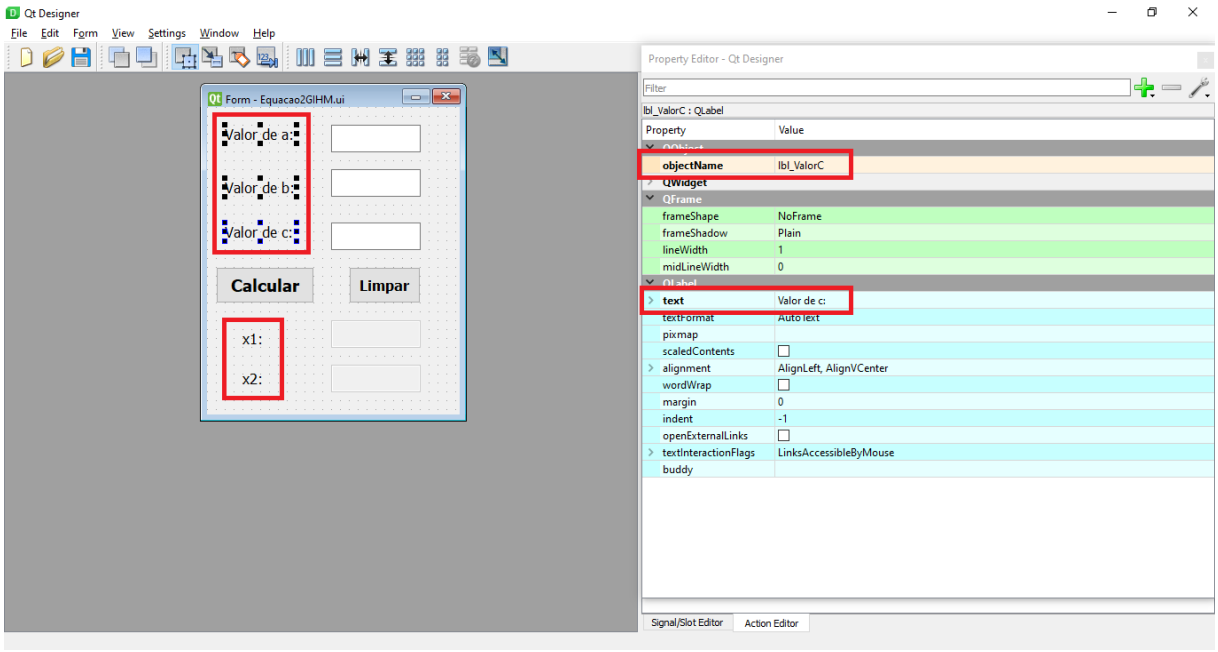
Os “***Push Button***” são os botões que farão as ações necessárias para a progressão do projeto.

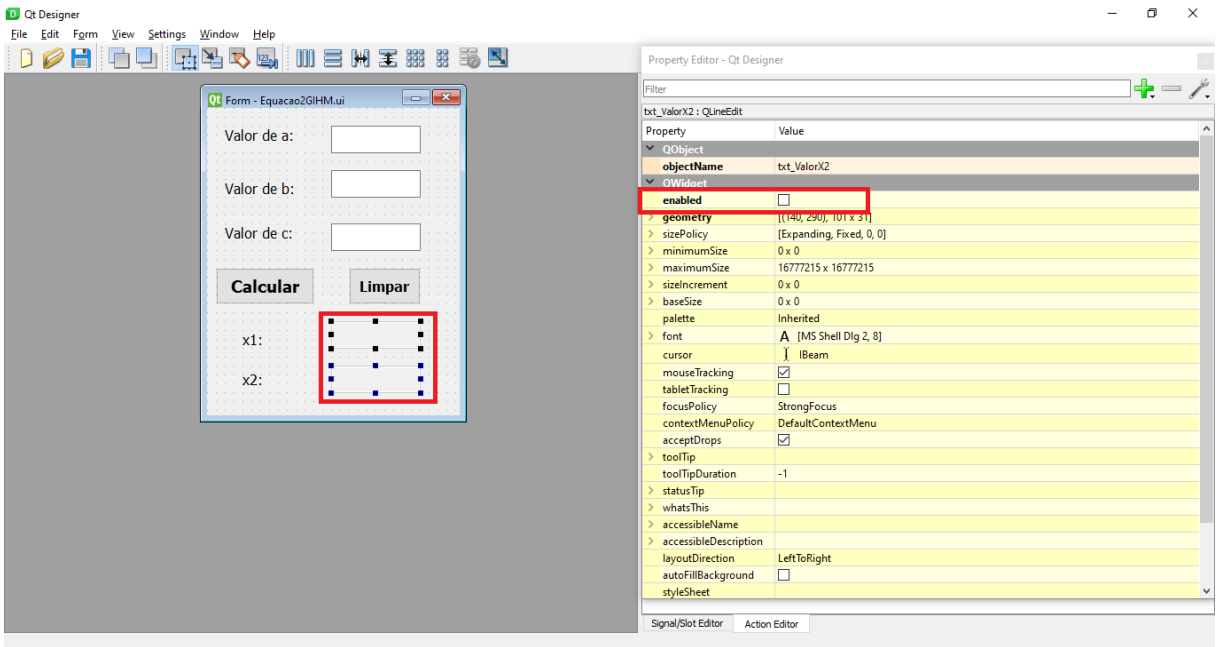


Nas ilustrações abaixo seguem todos os componentes e seus respectivos **“*objectName*”**. Nesta parte, foi feita a inserção dos nomes que aparecerão para o usuário na parte gráfica e dos “***objectName***” para facilitar posteriormente na programação.


Qt Designer
File Edit Form View Settings Window Help
Form - Equacao2GIHM.ui
Valor de a:
Valor de b:
Valor de c:
Calcular
Limpar
x1:
x2:
Property Editor - Qt Designer
Filter
txt_ValorC : QLineEdit
Property
Value
objectName
txt_ValorC
QWidget
QLineEdit
inputMask
text
maxLength
32767
frame
echoMode
Normal
cursorPosition
0
alignment
AlignLeft, AlignVCenter
dragEnabled
readOnly
placeholderText
cursorMoveStyle
LogicalMoveStyle
clearButtonEnabled
AlignLeft, AlignVCenter
Signal/Slot Editor
Action Editor


Qt Designer
File Edit Form View Settings Window Help
Form - Equacao2GIHM.ui
Valor de a:
Valor de b:
Valor de c:
Calcular
Limpar
x1:
x2:
Property Editor - Qt Designer
Filter
btn_Limpar : QPushButton
Property
Value
objectName
btn_Limpar
QWidget
QAbstractButton
text
Limpar
icon
iconSize
16 x 16
shortcut
checkable
checked
autoRepeat
autoExclusive
autoRepeatDelay
300
autoRepeatInterval
100
QPushButton
autoDefault
default
flat
Signal/Slot Editor
Action Editor

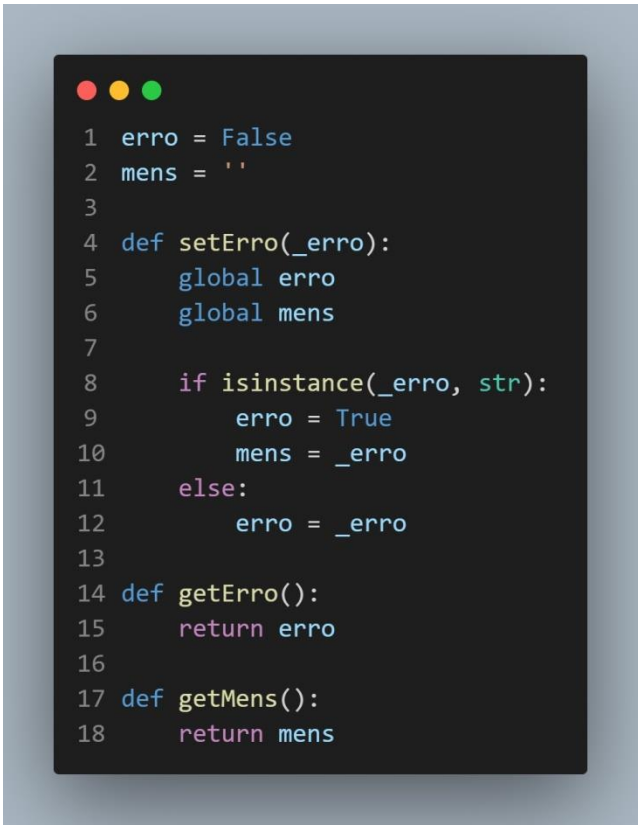




Os ***"Line Edit"*** selecionados, como mostrados acima, foram desmarcados como impossibilitados para a edição, ou seja, ***enabled = false***.

1.1 - CLASSES

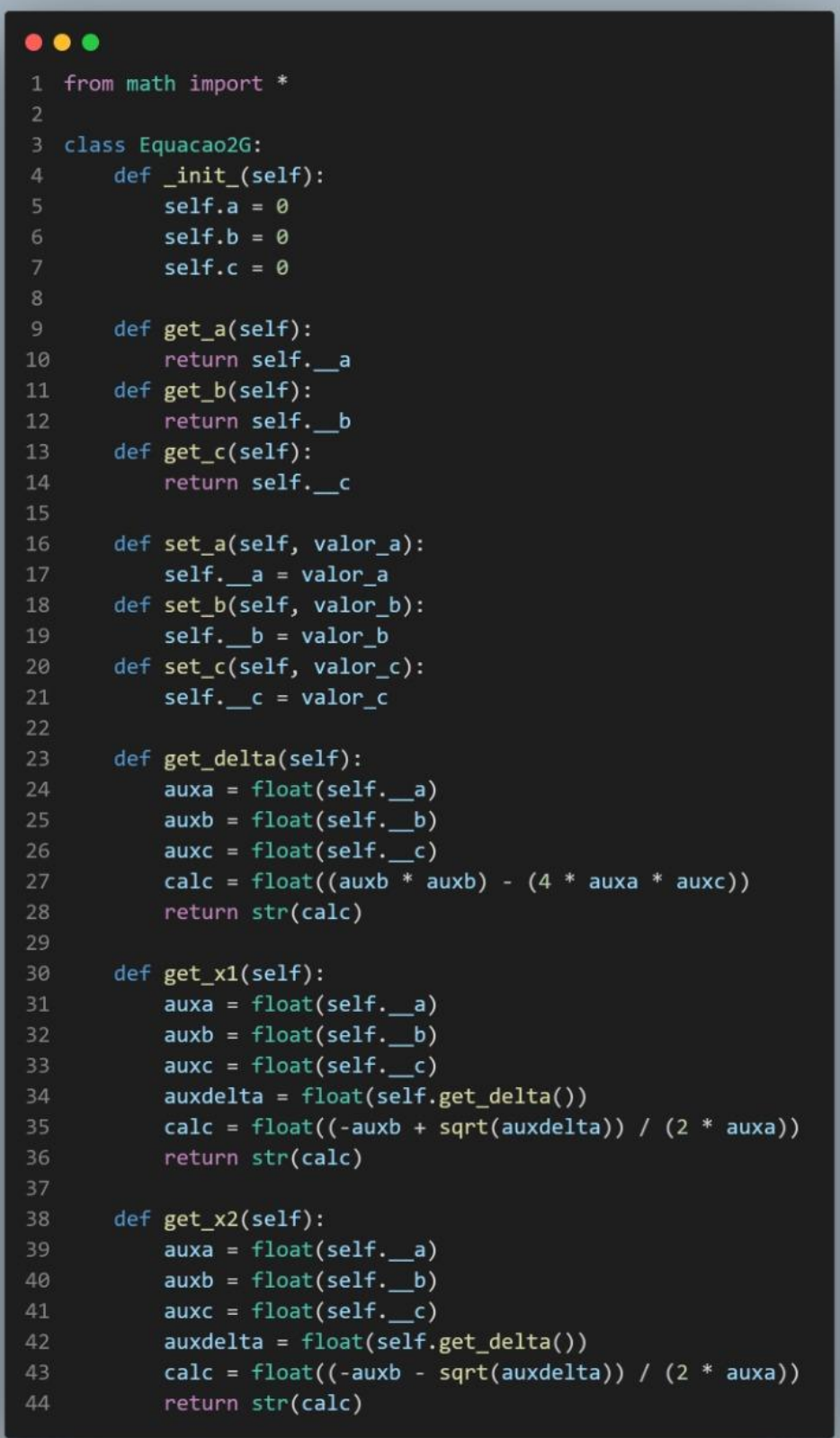
Nesse projeto, também foi usada a classe denominada “Erro”, conforme segue abaixo:



```python
erro = False
mens = ''

def setErro(_erro):
    global erro
    global mens

    if isinstance(_erro, str):
        erro = True
        mens = _erro
    else:
        erro = _erro

def getErro():
    return erro

def getMens():
    return mens
```

Na classe “Equação2G”, foi importada a biblioteca **“*Math*”** para auxiliar no cálculo da equação do segundo grau, na qual a fórmula utilizada foi:

$$\Delta = \boldsymbol{b^2 - 4 * a * c}$$



```python
from math import *

class Equacao2G:
    def _init_(self):
        self.a = 0
        self.b = 0
        self.c = 0

    def get_a(self):
        return self.__a
    def get_b(self):
        return self.__b
    def get_c(self):
        return self.__c

    def set_a(self, valor_a):
        self.__a = valor_a
    def set_b(self, valor_b):
        self.__b = valor_b
    def set_c(self, valor_c):
        self.__c = valor_c

    def get_delta(self):
        auxa = float(self.__a)
        auxb = float(self.__b)
        auxc = float(self.__c)
        calc = float((auxb * auxb) - (4 * auxa * auxc))
        return str(calc)

    def get_x1(self):
        auxa = float(self.__a)
        auxb = float(self.__b)
        auxc = float(self.__c)
        auxdelta = float(self.get_delta())
        calc = float((-auxb + sqrt(auxdelta)) / (2 * auxa))
        return str(calc)

    def get_x2(self):
        auxa = float(self.__a)
        auxb = float(self.__b)
        auxc = float(self.__c)
        auxdelta = float(self.get_delta())
        calc = float((-auxb - sqrt(auxdelta)) / (2 * auxa))
        return str(calc)
```

Para o cálculo das raízes da equação do segundo grau, foram utilizadas as fórmulas abaixo:

$$x_1 = \frac{-a - \sqrt{\Delta}}{2 * a}$$

$$x_2 = \frac{-a + \sqrt{\Delta}}{2 * a}$$

```python
def get_delta(self):
    auxa = float(self.__a)
    auxb = float(self.__b)
    auxc = float(self.__c)
    calc = float((auxb * auxb) - (4 * auxa * auxc))
    return str(calc)

def get_x1(self):
    auxa = float(self.__a)
    auxb = float(self.__b)
    auxc = float(self.__c)
    auxdelta = float(self.get_delta())
    calc = float((-auxb + sqrt(auxdelta)) / (2 * auxa))
    return str(calc)

def get_x2(self):
    auxa = float(self.__a)
    auxb = float(self.__b)
    auxc = float(self.__c)
    auxdelta = float(self.get_delta())
    calc = float((-auxb - sqrt(auxdelta)) / (2 * auxa))
    return str(calc)
```

Na classe “Equacao2GBLL”, será feita a validação dos dados que foram inseridos, levando em consideração as devidas regras para o cálculo.

Há casos em que mesmo que os dados forem válidos, o valor de delta (Δ) pode resultar em negativo, nesse caso não haverá valores para x_1 e x_2, dessa forma exibirá uma mensagem informando esta particularidade.

```python
import Erro

def validaDados(equacao2G):
    Erro.setErro(False)

    if len(equacao2G.get_a()) == 0:
        Erro.setErro("O valor de A é de preenchimento obrigatório...")
        return None
    else:
        try:
            float(equacao2G.get_a())
        except:
            Erro.setErro("O valor de A deve ser numérico...")
            return None

    if len(equacao2G.get_b()) == 0:
        Erro.setErro("O valor de B é de preenchimento obrigatório...")
        return None
    else:
        try:
            float(equacao2G.get_b())
        except:
            Erro.setErro("O valor de B deve ser numérico...")
            return None

    if len(equacao2G.get_a()) == 0:
        Erro.setErro("O valor de C é de preenchimento obrigatório...")
        return None
    else:
        try:
            float(equacao2G.get_c())
        except:
            Erro.setErro("O valor de C deve ser numérico...")
            return None

    if float(equacao2G.get_delta()) < 0:
        Erro.setErro("Delta negativo, não existem raízes reais para esta equação..")
```

Na classe “Equacao2GIHM”, encontram-se os métodos que fornecerão os resultados esperados e, além disso, irão armazenar toda a codificação gráfica.

```python
from PyQt5 import QtCore, QtGui, QtWidgets
import Erro
from Equacao2G import Equacao2G
import Equacao2GBLL

class Ui_Form(object):
    def setupUi(self, Form):
        Form.setObjectName("Form")
        Form.resize(285, 346)
        self.lbl_ValorA = QtWidgets.QLabel(Form)
        self.lbl_ValorA.setGeometry(QtCore.QRect(20, 20, 81, 21))
        font = QtGui.QFont()
        font.setPointSize(12)
        self.lbl_ValorA.setFont(font)
        self.lbl_ValorA.setObjectName("lbl_ValorA")
        self.lbl_ValorB = QtWidgets.QLabel(Form)
        self.lbl_ValorB.setGeometry(QtCore.QRect(20, 80, 81, 21))
        font = QtGui.QFont()
        font.setPointSize(12)
        self.lbl_ValorB.setFont(font)
        self.lbl_ValorB.setObjectName("lbl_ValorB")
        self.lbl_ValorC = QtWidgets.QLabel(Form)
        self.lbl_ValorC.setGeometry(QtCore.QRect(20, 130, 81, 21))
        font = QtGui.QFont()
        font.setPointSize(12)
        self.lbl_ValorC.setFont(font)
        self.lbl_ValorC.setObjectName("lbl_ValorC")
        self.btn_Calcular = QtWidgets.QPushButton(Form)
        self.btn_Calcular.setGeometry(QtCore.QRect(10, 180, 111, 41))
        font = QtGui.QFont()
        font.setPointSize(14)
        font.setBold(True)
        font.setWeight(75)
        self.btn_Calcular.setFont(font)
        self.btn_Calcular.setObjectName("btn_Calcular")
        self.btn_Limpar = QtWidgets.QPushButton(Form)
        self.btn_Limpar.setGeometry(QtCore.QRect(160, 180, 81, 41))
        font = QtGui.QFont()
        font.setPointSize(12)
        font.setBold(True)
        font.setWeight(75)
        self.btn_Limpar.setFont(font)
        self.btn_Limpar.setObjectName("btn_Limpar")
        self.lbl_ValorX1 = QtWidgets.QLabel(Form)
        self.lbl_ValorX1.setGeometry(QtCore.QRect(40, 250, 47, 21))
        font = QtGui.QFont()
        font.setPointSize(12)
        self.lbl_ValorX1.setFont(font)
        self.lbl_ValorX1.setObjectName("lbl_ValorX1")
        self.lbl_ValorX2 = QtWidgets.QLabel(Form)

```

```
        self.lbl_ValorX2.setGeometry(QtCore.QRect(40, 290, 61, 31))
        font = QtGui.QFont()
        font.setPointSize(12)
        self.lbl_ValorX2.setFont(font)
        self.lbl_ValorX2.setObjectName("lbl_ValorX2")
        self.txt_ValorA = QtWidgets.QLineEdit(Form)
        self.txt_ValorA.setGeometry(QtCore.QRect(140, 20, 101, 31))
        self.txt_ValorA.setObjectName("txt_ValorA")
        self.txt_ValorB = QtWidgets.QLineEdit(Form)
        self.txt_ValorB.setGeometry(QtCore.QRect(140, 70, 101, 31))
        self.txt_ValorB.setObjectName("txt_ValorB")
        self.txt_ValorC = QtWidgets.QLineEdit(Form)
        self.txt_ValorC.setGeometry(QtCore.QRect(140, 130, 101, 31))
        self.txt_ValorC.setObjectName("txt_ValorC")
        self.txt_ValorX1 = QtWidgets.QLineEdit(Form)
        self.txt_ValorX1.setEnabled(False)
        self.txt_ValorX1.setGeometry(QtCore.QRect(140, 240, 101, 31))
        self.txt_ValorX1.setObjectName("txt_ValorX1")
        self.txt_ValorX2 = QtWidgets.QLineEdit(Form)
        self.txt_ValorX2.setEnabled(False)
        self.txt_ValorX2.setGeometry(QtCore.QRect(140, 290, 101, 31))
        self.txt_ValorX2.setObjectName("txt_ValorX2")

        self.retranslateUi(Form)
        QtCore.QMetaObject.connectSlotsByName(Form)

        self.btn_Calcular.clicked.connect(self.Calcular)
        self.btn_Limpar.clicked.connect(self.Limpar)

    def retranslateUi(self, Form):
        _translate = QtCore.QCoreApplication.translate
        Form.setWindowTitle(_translate("Form", "Form"))
        self.lbl_ValorA.setText(_translate("Form", "Valor de a:"))
        self.lbl_ValorB.setText(_translate("Form", "Valor de b:"))
        self.lbl_ValorC.setText(_translate("Form", "Valor de c:"))
        self.btn_Calcular.setText(_translate("Form", "Calcular"))
        self.btn_Limpar.setText(_translate("Form", "Limpar"))
        self.lbl_ValorX1.setText(_translate("Form", "x1:"))
        self.lbl_ValorX2.setText(_translate("Form", "x2:"))

    def Calcular(self):
        equacao2G = Equacao2G()
        msg = QtWidgets.QMessageBox()

        equacao2G.set_a(self.txt_ValorA.text())
        equacao2G.set_b(self.txt_ValorB.text())
        equacao2G.set_c(self.txt_ValorC.text())

        Equacao2GBLL.validaDados(equacao2G)

        if(Erro.getErro()):
            msg.setText(Erro.getMens())
            msg.exec()
        else:
            self.txt_ValorX1.setText(equacao2G.get_x1())
            self.txt_ValorX2.setText(equacao2G.get_x2())

            self.txt_ValorA.setEnabled(False)
            self.txt_ValorB.setEnabled(False)
            self.txt_ValorC.setEnabled(False)
    def Limpar(self):
        self.txt_ValorA.setText("")
        self.txt_ValorB.setText("")
        self.txt_ValorC.setText("")
        self.txt_ValorX1.setText("")
        self.txt_ValorX2.setText("")

        self.txt_ValorA.setEnabled(True)
        self.txt_ValorB.setEnabled(True)
        self.txt_ValorC.setEnabled(True)

if __name__ == "__main__":
    import sys
    app = QtWidgets.QApplication(sys.argv)
    Form = QtWidgets.QWidget()
    ui = Ui_Form()
    ui.setupUi(Form)
    Form.show()
    sys.exit(app.exec_())
```

Foram importadas algumas classes e bibliotecas para o funcionamento dos métodos necessários para quando o usuário estiver navegando pela interface gráfica, conforme segue o exemplo abaixo:

```
1 from PyQt5 import QtCore, QtGui, QtWidgets
2 import Erro
3 from Equacao2G import Equacao2G
4 import Equacao2GBLL
```

A primeira importação foi de uma biblioteca do próprio PyQt5 utilizada para exibir uma mensagem na tela quando o usuário realizar alguma ação. As outras três importações foram de classes criadas e já citadas acima.

```python
def Calcular(self):
    equacao2G = Equacao2G()
    msg = QtWidgets.QMessageBox()

    equacao2G.set_a(self.txt_ValorA.text())
    equacao2G.set_b(self.txt_ValorB.text())
    equacao2G.set_c(self.txt_ValorC.text())

    Equacao2GBLL.validaDados(equacao2G)

    if(Erro.getErro()):
        msg.setText(Erro.getMens())
        msg.exec()
    else:
        self.txt_ValorX1.setText(equacao2G.get_x1())
        self.txt_ValorX2.setText(equacao2G.get_x2())

        self.txt_ValorA.setEnabled(False)
        self.txt_ValorB.setEnabled(False)
        self.txt_ValorC.setEnabled(False)

```

O método acima foi criado para quando o usuário clicar no botão “Calcular”, nele é possível observar as importações feitas. Assim que o botão “Calcular” for clicado, serão desabilitados os campos de ***“Line Edit”*** para impedir a digitação de novos coeficientes, posteriormente serão exibidos os resultados na tela, com base nos valores escolhidos pelo usuário.

```python
def Limpar(self):
    self.txt_ValorA.setText("")
    self.txt_ValorB.setText("")
    self.txt_ValorC.setText("")
    self.txt_ValorX1.setText("")
    self.txt_ValorX2.setText("")

    self.txt_ValorA.setEnabled(True)
    self.txt_ValorB.setEnabled(True)
    self.txt_ValorC.setEnabled(True)
```

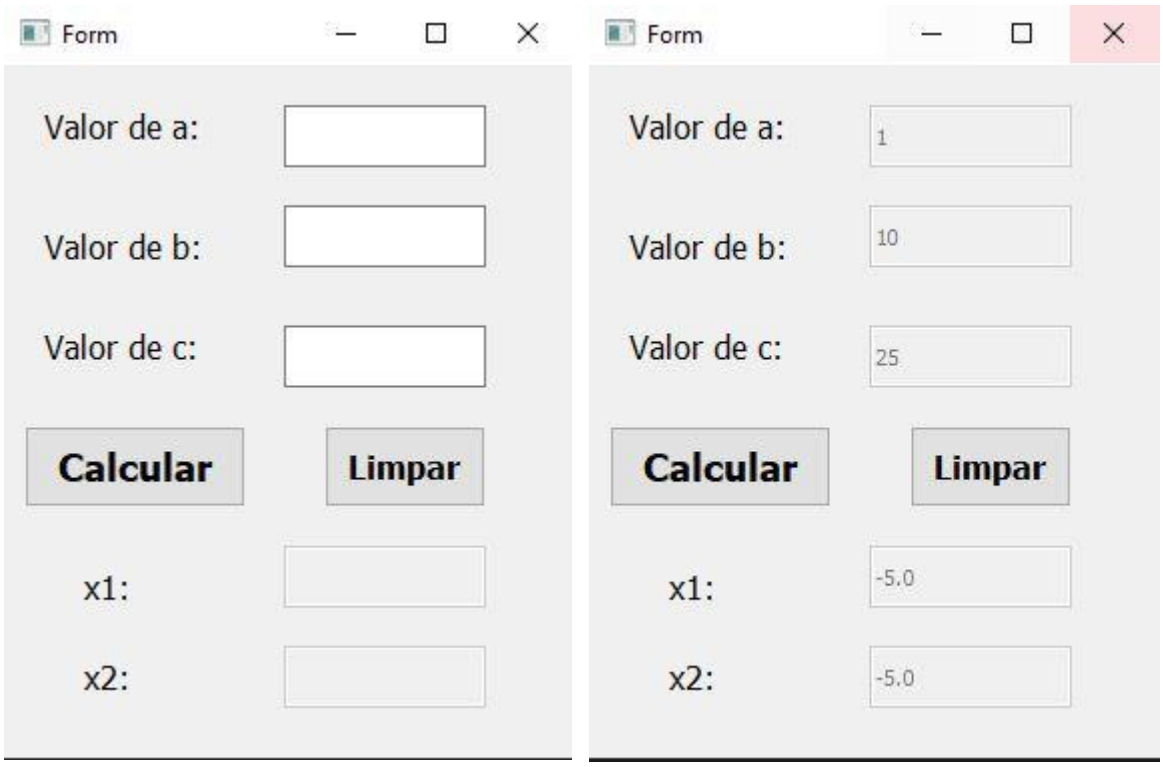
O método acima foi criado para quando o usuário clicar no botão “Limpar”. Basicamente, ao clicar nesse botão, será feita a limpeza de todos os campos, possibilitando que o usuário faça outra consulta.

```python
self.btn_Calcular.clicked.connect(self.Calcular)
self.btn_Limpar.clicked.connect(self.Limpar)
```

Essas duas linhas serão responsáveis por unir os botões aos métodos criados e citados acima.

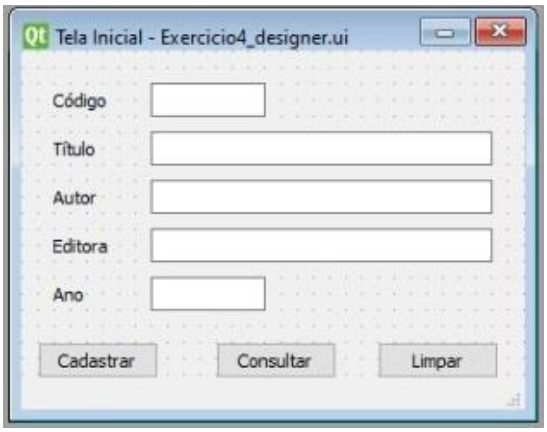
1.2 EXECUÇÃO

As ilustrações abaixo demonstram o projeto em execução:



4 – Cadastro livro

- Montando o design



Não se esqueça de salvar e transformar o design em python!

- Construindo a lógica

A princípio, foi elaborada uma classe denominada Livro, que por intermédio do método set será responsável por armazenar os dados inseridos pelo usuário, como o código, o título, o autor, a editora e o ano do livro. Além disso, por intermédio do método get, a classe apresenta os respectivos dados "setados" para o usuário.

```
class Livro:
    def _init_(self):
        self.codigo = ""
        self.titulo = ""
        self.autor = ""
        self.editora = ""
        self.ano = ""


    def getCodigo(self):
        return self.__codigo
    def getTitulo(self):
        return self.__titulo
    def getAutor(self):
        return self.__autor
    def getEditora(self):
        return self.__editora
    def getAno(self):
        return self.__ano


    def setCodigo(self, valor_codigo):
        self.__codigo = valor_codigo
    def setTitulo(self, valor_titulo):
        self.__titulo = valor_titulo
    def setAutor(self, valor_autor):
        self.__autor = valor_autor
    def setEditora(self, valor_editora):
        self.__editora = valor_editora
    def setAno(self, valor_ano):
        self.__ano = valor_ano
```

Posterior a isso, foi criado outro arquivo .py denominado erro.py, que tem como propósito lidar com os erros do programa, utilizando de mesmos métodos já citados anteriormente, o método get e o set. Neste trecho, é possível notar o uso de variáveis globais, que nada mais são do que variáveis acessíveis em todos os escopos do programa. Intercalado aos métodos, tem-se uma estrutura condicional if/else para determinar se é um erro ou não.

```
erro = False
mens = ''

def setErro(_erro):
    global erro
    global mens

    if isinstance(_erro, str):
        erro = True
        mens = _erro
    else:
        erro = _erro

def getErro():
    return erro

def getMens():
    return mens
```

Após isto, foi criado o arquivo Exercício4BLL.py. Nele foi importado o elemento erro criado anteriormente.

```
import Erro
```

Então, foi feito o trecho do código que irá checar se o usuário preencheu todos os campos. Nesta parte, foi utilizado o método len() que retorno o número de elementos da lista, no programa, ele recebe como parâmetro os dados inseridos pelo usuários e posteriormente é feito a comparação com o número 0, pois se for igual é sinal de que o usuários deixou o campo em branco.

```
def validaCodigo(livro):
    Erro.setErro(False)

    if len(livro.getCodigo) == 0:
        Erro.setErro("O código é de preenchimento obrigatório!")
        return

def validaDados(livro):

    Erro.setErro(False)

    if len(livro.getCodigo()) == 0:
        Erro.setErro("O código é de preenchimento obrigatório!")
        return None

    if len(livro.getTitulo()) == 0:
        Erro.setErro("O título é de preenchimento obrigatório!")
        return None

    if len(livro.getAutor()) == 0:
        Erro.setErro("O autor é de preenchimento obrigatório!")
        return None

    if len(livro.getEditora()) == 0:
        Erro.setErro("O editora é de preenchimento obrigatório!")
        return None

    if len(livro.getAno()) == 0:
        Erro.setErro("O ano é de preenchimento obrigatório!")
        return None
```

Ainda na checagem de dados, é analisado se o usuário utilizou algo além de números para preencher o campo “ano” ou se o número inserido é um valor positivo. Para isto foi utilizado a estrutura try/except, que serve para tratamento de exceções, logo, se o programa identificar alguma exceção, algo além de número no campo “ano”, irá retornar ao usuário. Para checar se o valor é positivo, o ano inserido pelo usuário, por intermédio de um if, será comparado como valor igual ou menor que 0, caso for, irá retornar ao usuário.

```
try:
    int(livro.getAno())
except:
    Erro.setErro("O valor do ano deve ser numérico!")
    return None

if int(livro.getAno()) <= 0:
    Erro.setErro("O valor do ano deve ser numérico e positivo!")
    return None
```

Por fim, foi criado o arquivo denominado Exercicio4IHM.py, onde nele fica armazenado umas das principais informações sobre o programa. A princípio foram importados os arquivos necessários.

```
from PyQt5 import QtCore, QtGui, QtWidgets
#Import classes
import Erro
import Exercicio4BLL
from Exercicio4 import Livro
```

A classe livro foi adicionada ao def setupUI (def gerada automaticamente com a conversão do design) para poder ser utilizada como global e ficar acessível em todos os escopos do programa.

```
self.livro = Livro()
```

Em seguida, os botões foram configurados e tornados em clicáveis.

```
self.btnCadastrar.clicked.connect(self.Cadastrar)
self.btnConsultar.clicked.connect(self.Consultar)
self.btnLimpar.clicked.connect(self.Limpar)
```

O botão “Cadastrar” foi programado para inserir os valores na classe Livro, neste trecho o arquivo Exercicio4BLL é chamado para fazer a validação dos dados como foi programado anteriormente. A estrutura condicional if/else foi utilizada para apresentar a mensagem de erro caso tenha alguma inconsistência nos dados inseridos ou para apresentar a mensagem informando que os dados foram inseridos com sucesso.

```
def Cadastrar(self):
    msg = QtWidgets.QMessageBox()

    #Inserindo valores na classe livro
    self.livro.setCodigo(self.txtCodigo.text())
    self.livro.setTitulo(self.txtTitulo.text())
    self.livro.setAutor(self.txtAutor.text())
    self.livro.setEditora(self.txtEditora.text())
    self.livro.setAno(self.txtAno.text())

    Exercicio4BLL.validaDados(self.livro)

    if Erro.getErro():
        msg.setText(Erro.getMens())
        msg.exec()
    else:
        msg.setText("Dados inseridos com sucesso!")
        msg.exec()
```

O botão “Consultar” foi programado a partir do código inserido pelo usuário, ele utiliza de uma falsa base de dados, neste caso o setCodigo, para averiguar se aquele dado havia sido colocado pelo usuário, além de fazer a checagem dos dados por meio do arquivo Exercicio4BLL. Caso tenha algum problema, a estrutura condicional retornará uma mensagem de erro, caso contrário irá retornar com os dados solicitados.

```
def Consultar(self):
    msg = QtWidgets.QMessageBox()

    self.livro.setCodigo(self.txtCodigo.text())
    Exercicio4BLL.validaCodigo(self.livro)

    if Erro.getErro():
        msg.setText(Erro.getMens())
        msg.exec()
    else:
        self.txtCodigo.setText(self.livro.getCodigo())
        self.txtTitulo.setText(self.livro.getTitulo())
        self.txtAutor.setText(self.livro.getAutor())
        self.txtEditora.setText(self.livro.getEditora())
        self.txtAno.setText(self.livro.getAno())
```

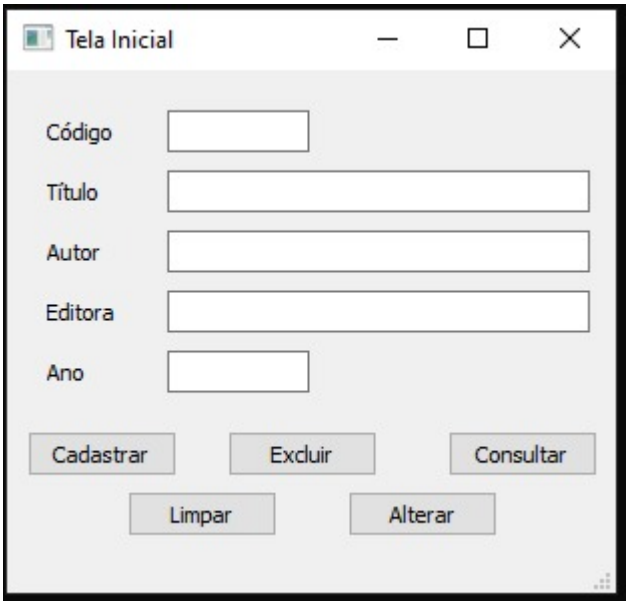
Por fim, o botão “limpar” só limpará os campos utilizados.

```
def Limpar(self):
    self.txtCodigo.setText("")
    self.txtTitulo.setText("")
    self.txtAutor.setText("")
    self.txtEditora.setText("")
    self.txtAno.setText("")
```

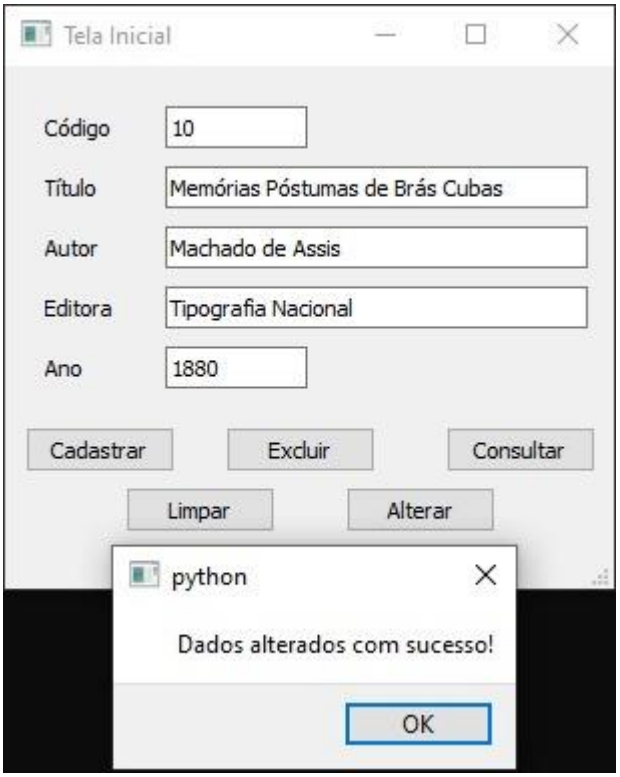
Capítulo 5 – Ligando My Sql ao Python

1 – Acesso ao banco de dados

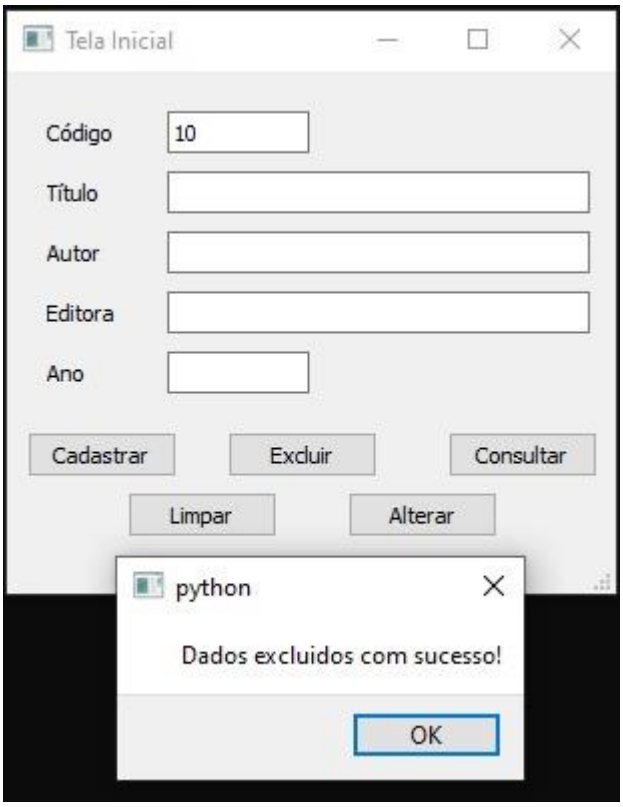
Design do programa:



A partir da digitação do livro, o programa deverá cadastra-lo e, em seguida, emitir uma mensagem de sucesso no monitor.



Ele também mostrará quando o livro cadastrado for excluído.

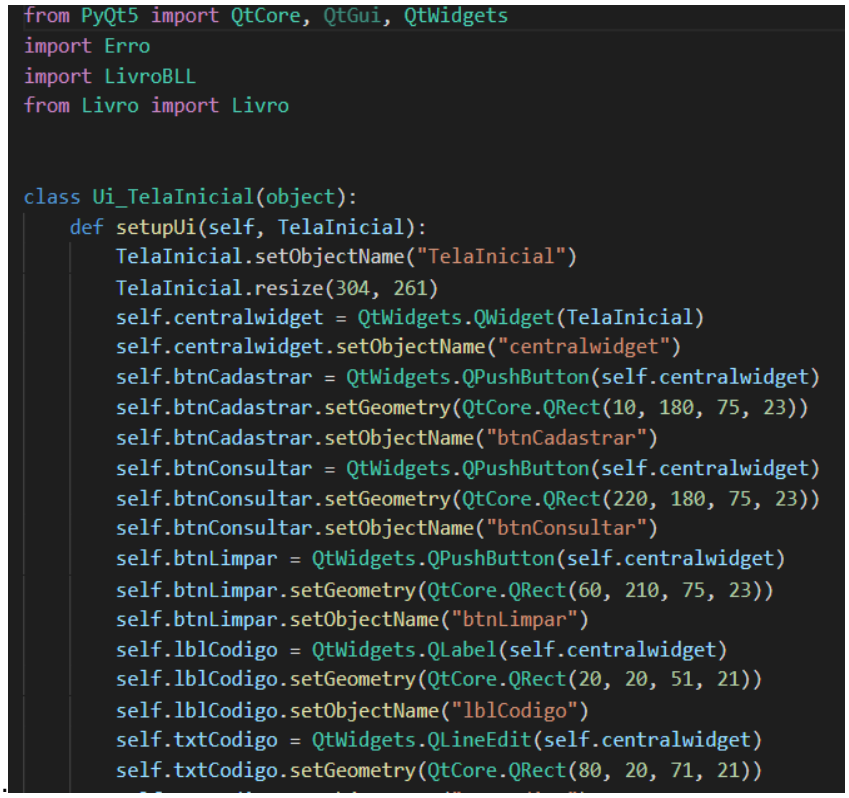


2 – Construindo a lógica

A base que fora utilizada foi o programa 4 do capítulo anterior, porém as únicas classes que permaneceram foram: livro e erro.

2.1 – Classe IMH

Na classe IMH, foi construída a interação entre o formulário e o programa. Primeiramente, foram importadas as classes erro, livro e BLL.



```
from PyQt5 import QtCore, QtGui, QtWidgets
import Erro
import LivroBLL
from Livro import Livro


class Ui_TelaInicial(object):
    def setupUi(self, TelaInicial):
        TelaInicial.setObjectName("TelaInicial")
        TelaInicial.resize(304, 261)
        self.centralwidget = QtWidgets.QWidget(TelaInicial)
        self.centralwidget.setObjectName("centralwidget")
        self.btnCadastrar = QtWidgets.QPushButton(self.centralwidget)
        self.btnCadastrar.setGeometry(QtCore.QRect(10, 180, 75, 23))
        self.btnCadastrar.setObjectName("btnCadastrar")
        self.btnConsultar = QtWidgets.QPushButton(self.centralwidget)
        self.btnConsultar.setGeometry(QtCore.QRect(220, 180, 75, 23))
        self.btnConsultar.setObjectName("btnConsultar")
        self.btnLimpar = QtWidgets.QPushButton(self.centralwidget)
        self.btnLimpar.setGeometry(QtCore.QRect(60, 210, 75, 23))
        self.btnLimpar.setObjectName("btnLimpar")
        self.lblCodigo = QtWidgets.QLabel(self.centralwidget)
        self.lblCodigo.setGeometry(QtCore.QRect(20, 20, 51, 21))
        self.lblCodigo.setObjectName("lblCodigo")
        self.txtCodigo = QtWidgets.QLineEdit(self.centralwidget)
        self.txtCodigo.setGeometry(QtCore.QRect(80, 20, 71, 21))
```

Na classe Ui_TelaInicial(object) é onde se encontram especificados cada um dos componentes adicionados no formulário.

```
self.txtCodigo.setGeometry(QtCore.QRect(80, 20, 71, 21))
self.txtCodigo.setObjectName("txtCodigo")
self.txtTitulo = QtWidgets.QLineEdit(self.centralwidget)
self.txtTitulo.setGeometry(QtCore.QRect(80, 50, 211, 21))
self.txtTitulo.setObjectName("txtTitulo")
self.lblTitulo = QtWidgets.QLabel(self.centralwidget)
self.lblTitulo.setGeometry(QtCore.QRect(20, 50, 51, 21))
self.lblTitulo.setObjectName("lblTitulo")
self.txtAutor = QtWidgets.QLineEdit(self.centralwidget)
self.txtAutor.setGeometry(QtCore.QRect(80, 80, 211, 21))
self.txtAutor.setObjectName("txtAutor")
self.lblAutor = QtWidgets.QLabel(self.centralwidget)
self.lblAutor.setGeometry(QtCore.QRect(20, 80, 51, 21))
self.lblAutor.setObjectName("lblAutor")
self.txtEditora = QtWidgets.QLineEdit(self.centralwidget)
self.txtEditora.setGeometry(QtCore.QRect(80, 110, 211, 21))
self.txtEditora.setObjectName("txtEditora")
self.lblEditora = QtWidgets.QLabel(self.centralwidget)
self.lblEditora.setGeometry(QtCore.QRect(20, 110, 51, 21))
self.lblEditora.setObjectName("lblEditora")
self.txtAno = QtWidgets.QLineEdit(self.centralwidget)
self.txtAno.setGeometry(QtCore.QRect(80, 140, 71, 21))
self.txtAno.setText("")
self.txtAno.setObjectName("txtAno")
self.lblAno = QtWidgets.QLabel(self.centralwidget)
self.lblAno.setGeometry(QtCore.QRect(20, 140, 51, 21))
```

```
        self.lblAno.setGeometry(QtCore.QRect(20, 140, 51, 21))
        self.lblAno.setObjectName("lblAno")
        self.btnExcluir = QtWidgets.QPushButton(self.centralwidget)
        self.btnExcluir.setGeometry(QtCore.QRect(110, 180, 75, 23))
        self.btnExcluir.setObjectName("btnExluir")
        self.btnAlterar = QtWidgets.QPushButton(self.centralwidget)
        self.btnAlterar.setGeometry(QtCore.QRect(170, 210, 75, 23))
        self.btnAlterar.setObjectName("btnAlterar")
        TelaInicial.setCentralWidget(self.centralwidget)
        self.statusbar = QtWidgets.QStatusBar(TelaInicial)
        self.statusbar.setObjectName("statusbar")
        TelaInicial.setStatusBar(self.statusbar)
        self.livro = Livro()
        LivroBLL.conecta(self)
```

```
        self.retranslateUi(TelaInicial)
        QtCore.QMetaObject.connectSlotsByName(TelaInicial)

        self.btnCadastrar.clicked.connect(self.Cadastrar)
        self.btnConsultar.clicked.connect(self.Consultar)
        self.btnExcluir.clicked.connect(self.Excluir)
        self.btnAlterar.clicked.connect(self.Alterar)
        self.btnLimpar.clicked.connect(self.Limpar)

    def retranslateUi(self, TelaInicial):
        _translate = QtCore.QCoreApplication.translate
        TelaInicial.setWindowTitle(_translate("TelaInicial", "Tela Inicial"))
        self.btnCadastrar.setText(_translate("TelaInicial", "Cadastrar"))
        self.btnConsultar.setText(_translate("TelaInicial", "Consultar"))
        self.btnLimpar.setText(_translate("TelaInicial", "Limpar"))
        self.lblCodigo.setText(_translate("TelaInicial", "Código"))
        self.lblTitulo.setText(_translate("TelaInicial", "Título"))
        self.lblAutor.setText(_translate("TelaInicial", "Autor"))
        self.lblEditora.setText(_translate("TelaInicial", "Editora"))
        self.lblAno.setText(_translate("TelaInicial", "Ano"))
        self.btnExcluir.setText(_translate("TelaInicial", "Excluir"))
        self.btnAlterar.setText(_translate("TelaInicial", "Alterar"))
```

Após a especificação dos componentes, foi feita a conexão dos métodos de eventos aos botões.

```
def Cadastrar(self):
    msg = QtWidgets.QMessageBox()

    self.livro.setCodigo(self.txtCodigo.text())
    self.livro.setTitulo(self.txtTitulo.text())
    self.livro.setAutor(self.txtAutor.text())
    self.livro.setEditora(self.txtEditora.text())
    self.livro.setAno(self.txtAno.text())

    LivroBLL.validaDados(self.livro, "i")

    if Erro.getErro():
        msg.setText(Erro.getMens())
        msg.exec()
    else:
        msg.setText("Dados inseridos com sucesso!")
        msg.exec()
```

A def Cadastrar foi criada para que, quando o usuário pressionar o botão Cadastrar, os dados necessários serão enviados para a BLL.

```
def Consultar(self):
    msg = QtWidgets.QMessageBox()

    self.livro.setCodigo(self.txtCodigo.text())
    LivroBLL.validaCodigo(self.livro, "c")

    if Erro.getErro():
        msg.setText(Erro.getMens())
        msg.exec()
    else:
        self.txtCodigo.setText(self.livro.getCodigo())
        self.txtTitulo.setText(self.livro.getTitulo())
        self.txtAutor.setText(self.livro.getAutor())
        self.txtEditora.setText(self.livro.getEditora())
        self.txtAno.setText(self.livro.getAno())
```

A def Consultar foi criada para que, assim que o usuário pressionar o botão Consultar, os dados necessários para a BLL serão enviados.

```
self.livro.setCodigo(0)
self.livro.setTitulo("")
self.livro.setAutor("")
self.livro.setEditora("")
self.livro.setAno(0)
```

Após a exibição dos dados, são excluídos todos os dados que estão na memória do programa, evitando qualquer tipo de interferência.

```
def Limpar(self):
    self.txtCodigo.setText("")
    self.txtTitulo.setText("")
    self.txtAutor.setText("")
    self.txtEditora.setText("")
    self.txtAno.setText("")
```

A def Limpar foi criada para que, assim que o usuário pressionar o botão limpar, todas as informações presentes nos campos serão apagadas.

```
def Excluir(self):
    msg = QtWidgets.QMessageBox()

    self.livro.setCodigo(self.txtCodigo.text())
    LivroBLL.validaCodigo(self.livro, "d")

    if Erro.getErro():
        msg.setText(Erro.getMens())
        msg.exec()
    else:
        msg.setText("Dados excluidos com sucesso!")
        msg.exec()
```

```
    def Alterar(self):
        msg = QtWidgets.QMessageBox()

        self.livro.setCodigo(self.txtCodigo.text())
        self.livro.setTitulo(self.txtTitulo.text())
        self.livro.setAutor(self.txtAutor.text())
        self.livro.setEditora(self.txtEditora.text())
        self.livro.setAno(self.txtAno.text())

        LivroBLL.validaDados(self.livro, "A")

        if Erro.getErro():
            msg.setText(Erro.getMens())
            msg.exec()
        else:
            msg.setText("Dados alterados com sucesso!")
            msg.exec()

if __name__ == "__main__":
    import sys
    app = QtWidgets.QApplication(sys.argv)
    TelaInicial = QtWidgets.QMainWindow()
    ui = Ui_TelaInicial()
    ui.setupUi(TelaInicial)
    TelaInicial.show()
    sys.exit(app.exec_())
```

As defs Excluir e Alterar foram criadas para, assim que o usuário pressionar o botão excluir ou o alterar, enviar-se-á os dados necessários para a BLL. Os dados serão validados, verificando se está tudo correto. Se não estiver correto, a mensagem de erro da classe erro será gerada. E, caso tudo esteja correto, será apresentada a mensagem de que foi inserido com sucesso.

2.2 – Classe BLL

Para começar, é necessário importar as classes Erro e DAL, em seguida define-se três defs, uma para chamar a pagina DAL e conectar com o banco de dados, outra para chamar a pagina DAL e desconectar com o banco de dados e a última para verificar o método validaCodigo(livro, op), que tem como objetivo identificar se houve algum erro. O último If é utilizado para fazer a análise de casos: caso seja c, consulta o livro, caso seja outra letra, deleta o livro.

```
import Erro
import LivroDAL


def conecta(self):
    LivroDAL.conecta(self)

def desconecta(self):
    LivroDAL.desconecta()


def validaCodigo(livro, op):
    Erro.setErro(False)

    if len(livro.getCodigo()) == 0:
        Erro.setErro("O código é de preenchimento obrigatório!")
        return

    if op == "c":
        LivroDAL.ConsultarLivro(livro)
    else:
        LivroDAL.ExcluirLivro(livro)
```

Em caso de erro, através de um laço de decisão, é feita a verificação do tipo de erro, que pode ser: código, título, autor, editora não preenchido corretamente, entre outros. Após a verificação do erro, é retornada ao usuário uma mensagem comunicando a existência da falha.

```
def validaDados(livro, op):

    Erro.setErro(False)

    if len(livro.getCodigo()) == 0:
        Erro.setErro("O código é de preenchimento obrigatório!")
        return None

    if len(livro.getTitulo()) == 0:
        Erro.setErro("O título é de preenchimento obrigatório!")
        return None

    if len(livro.getAutor()) == 0:
        Erro.setErro("O autor é de preenchimento obrigatório!")
        return None

    if len(livro.getEditora()) == 0:
        Erro.setErro("O editora é de preenchimento obrigatório!")
        return None

    if len(livro.getAno()) == 0:
        Erro.setErro("O ano é de preenchimento obrigatório!")
        return None
```

```
    try:
        int(livro.getAno())
    except:
        Erro.setErro("O valor do ano deve ser numérico!")
        return None

    if int(livro.getAno()) <= 0:
        Erro.setErro("O valor do ano deve ser numérico e positivo!")
        return None
```

Aqui, o If é utilizado para fazer uma análise de casos: caso seja i, insere o livro, caso seja outra letra, atualiza o livro.

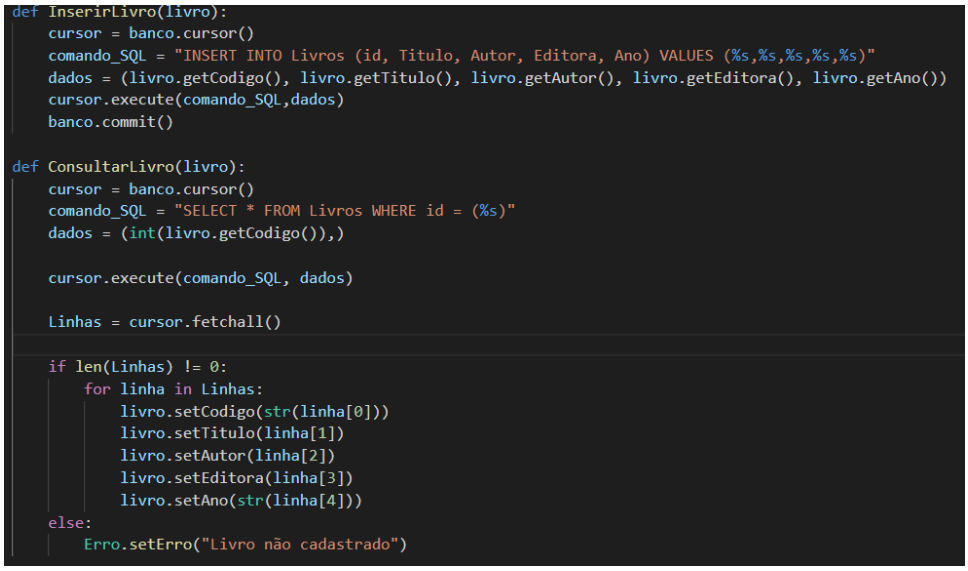
```
if op == "i":
    LivroDAL.InserirLivro(livro)
else:
    LivroDAL.AtualizarLivro(livro)
```

2.3 – Classe DAL

Nessa classe, importa-se a biblioteca mysql.connector e a classe Erro, em seguida cria-se a variável que recebe a instancia do connector e faz a conexão com o banco. O primeiro def criado tem o objetivo de fazer a conexão entre o programa e o banco, nesse If percebe-se que, ao conectar o banco, é exibido no prompt uma mensagem informando que foi conectado e em qual banco foi conectado. O segundo def foi criado para desconectar do banco de dados.

```
import mysql.connector
import Erro

banco = mysql.connector.connect(host='localhost',database='CadLivros',user='root',password='Pedro123!')

def conecta(self):

    if mysql.connector.connect():
        db_info = banco.get_server_info()
        print("Conectado ao servidor MySQL versão ", db_info)
        cursor = banco.cursor()
        cursor.execute("select database();")
        linha = cursor.fetchone()
        print("Conectado ao bando de dados ", linha)

def desconectar(self):
    cursor.close()
    con.close()
    print("Conexão ao MySQL encerrada")
```

Nessa segunda parte da DAL foram criadas duas defs, a primeira foi criada para inserir um livro, com o cursor e o comando SQL e a segunda criada para consultar um livro, com o cursor e o comando SQL. Nota-se na segunda que para que seja possível fazer a consulta do livro selecionado pelo usuário, utiliza-se o código inserido anteriormente e, para ter certeza de que o livro consultado está cadastrado no banco, utiliza-se o If.



```
def InserirLivro(livro):
    cursor = banco.cursor()
    comando_SQL = "INSERT INTO Livros (id, Titulo, Autor, Editora, Ano) VALUES (%s,%s,%s,%s,%s)"
    dados = (livro.getCodigo(), livro.getTitulo(), livro.getAutor(), livro.getEditora(), livro.getAno())
    cursor.execute(comando_SQL,dados)
    banco.commit()

def ConsultarLivro(livro):
    cursor = banco.cursor()
    comando_SQL = "SELECT * FROM Livros WHERE id = (%s)"
    dados = (int(livro.getCodigo()),)

    cursor.execute(comando_SQL, dados)

    Linhas = cursor.fetchall()

    if len(Linhas) != 0:
        for linha in Linhas:
            livro.setCodigo(str(linha[0]))
            livro.setTitulo(linha[1])
            livro.setAutor(linha[2])
            livro.setEditora(linha[3])
            livro.setAno(str(linha[4]))
    else:
        Erro.setErro("Livro não cadastrado")
```

Na terceira e última parte do DAL, tem-se mais duas defs: a def Excluir que exclui um livro com o cursor e o comando SQL e a def Atualizar, que atualiza um livro com o cursor e o comando SQL. Para que se apague/atualize o livro selecionado pelo usuário deve-se utilizar o código, inserido anteriormente.

```
def ExcluirLivro(livro):
    cursor = banco.cursor()
    comando_SQL = "DELETE FROM Livros WHERE id = %s"
    dados = (int(livro.getCodigo()),)
    cursor.execute(comando_SQL,dados)
    banco.commit()

def AtualizarLivro(livro):
    cursor = banco.cursor()
    comando_SQL = "UPDATE Livros set id = %s, Titulo = %s, Autor=%s, Editora=%s, Ano=%s WHERE id=%s;"
    dados = (livro.getCodigo(), livro.getTitulo(),  livro.getAutor(), livro.getEditora(), livro.getAno(), int(livro.getCodigo()))
    cursor.execute(comando_SQL, dados)
    banco.commit()
```

Capítulo 6 – Aplicando Firebase Google no Python

1 - CONSTRUINDO A LÓGICA

Foi reaproveitada a aplicação feita no capítulo anterior, portanto as classes *Erro* e *Livro* continuam idênticas neste projeto, já as demais classes foram devidamente modificadas.

1 - CLASSE DAL

Nessa classe foram importadas as classes: *Erro* e *Pyrebase* as quais serão importantes para a validação de todos os componentes do formulário, além de fazer o acesso com a API.

```
import pyrebase
import Erro
```

Foi implementado a configuração padrão do Firebase de Google para que ao longo do desenvolvimento do projeto haja a criação de aplicações de alta qualidade e performance. Foi criada também a variável *dados* com o Firebase.

```
config = {
    "apiKey": "AIzaSyCljr18iqOvHELNzHPZiRohhBeTRoEO-sw",
    "authDomain": "crudtcc-bd856.firebaseapp.com",
    "databaseURL": "https://crudtcc-bd856-default-rtdb.firebaseio.com",
    "projectId": "crudtcc-bd856",
    "storageBucket": "crudtcc-bd856.appspot.com",
    "messagingSenderId": "646646144344",
    "appId": "1:646646144344:web:a0e950d1cc0d8d1179d92d"
}

firebase = pyrebase.initialize_app(config)

#Variavel com o Firebase
dados = firebase.database()
```

Na def InserirLivro foi usada a classe child que serve para aceder à sub-nós dentro de uma referência. Nesse caso, será colocado o livro no Firebase o qual ficará dentro da classe Livros e será colocado na subclasse com seu devido código ID.

```
def InserirLivro(livro):
    dados.child("Livros").child(livro.getCodigo()).set({"Id": livro.getCodigo(), "Titulo": livro.
    getTitulo(), "Autor": livro.getAutor(), "Editora": livro.getEditora(), "Ano": livro.getAno()})
```

Para a construção dessa classe foi criada a def ConsultarLivro, com o intuito de fazer a consulta de um livro que anteriormente já foram inseridas suas devidas informações. Para verificar se realmente o livro está cadastrado no Firebase foi criado o try except, dessa forma ao fazer a procura do livro e não encontrá-lo será exibido a mensagem que o livro não foi cadastrado.

```
def ConsultarLivro(livro):
    try:
        Dados = dados.child("Livros").child(livro.getCodigo()).get()
        livro.setCodigo(str(Dados.val()['Id']))
        livro.setTitulo(Dados.val()['Titulo'])
        livro.setAutor(Dados.val()['Autor'])
        livro.setEditora(Dados.val()['Editora'])
        livro.setAno(str(Dados.val()['Ano']))
    except:
        Erro.setErro("Livro não cadastrado!")
```

A def ExcluirLivro foi criada para fazer a exclusão do cadastro de um livro.

```
def ExcluirLivro(livro):
    dados.child("Livros").child(livro.getCodigo()).remove()
```

A def AtualizarLivro foi criada para fazer a atualização do cadastro de um livro, possibilitando a alteração dos campos código, título, autor, editora e ano.

```
def AtualizarLivro(livro):
    dados.child("Livros").child(livro.getCodigo()).update({"Id": livro.getCodigo(), "Titulo": livro.
    getTitulo(), "Autor": livro.getAutor(), "Editora": livro.getEditora(), "Ano": livro.getAno()})
```

2 - CLASSE IMH

Nessa classe foi importado mais duas classes: *Erro* e *LivroBLL*. Elas serão importantes para validação de todos os componentes do formulário e fazer o acesso com a API.

```
import Erro
import LivroBLL
```

Uma *def* foi criada para que, ao usuário clicar no botão, os dados necessários serem enviados para a *BLL*. Apresenta um *if* para mostrar a mensagem de erro e o *else*, uma mensagem de sucesso.

```
def Cadastrar(self):
    msg = QtWidgets.QMessageBox()

    self.livro.setCodigo(self.txtCodigo.text())
    self.livro.setTitulo(self.txtTitulo.text())
    self.livro.setAutor(self.txtAutor.text())
    self.livro.setEditora(self.txtEditora.text())
    self.livro.setAno(self.txtAno.text())

    LivroBLL.validaDados(self.livro, "i")

    if Erro.getErro():
        msg.setText(Erro.getMens())
        msg.exec()
    else:
        msg.setText("Dados inseridos com sucesso!")
        msg.exec()
```

Logo depois, mais uma *def* para o botão consultar. Assim que o usuário clicar no botão, os dados equivalentes e necessários serão enviados para a *BLL*. Apresenta um *if* para importar a mensagem de erro e o *else* sendo responsável por exibir os dados corretamente.

```
def Consultar(self):
    msg = QtWidgets.QMessageBox()

    self.livro.setCodigo(self.txtCodigo.text())
    LivroBLL.validaCodigo(self.livro, "c")

    if Erro.getErro():
        msg.setText(Erro.getMens())
        msg.exec()
    else:
        self.txtCodigo.setText(self.livro.getCodigo())
        self.txtTitulo.setText(self.livro.getTitulo())
        self.txtAutor.setText(self.livro.getAutor())
        self.txtEditora.setText(self.livro.getEditora())
        self.txtAno.setText(self.livro.getAno())
```

Após a exibição dos dados, serão excluídos todos os dados que estão na memória do programa, assim evitando alguma interferência.

```
self.livro.setCodigo(0)
self.livro.setTitulo("")
self.livro.setAutor("")
self.livro.setEditora("")
self.livro.setAno(0)
```

Uma *def* para quando o botão limpar for pressionado todos os dados sejam limpos.

```
def Limpar(self):
    self.txtCodigo.setText("")
    self.txtTitulo.setText("")
    self.txtAutor.setText("")
    self.txtEditora.setText("")
    self.txtAno.setText("")
```

Def criada para assim que o usuário pressionar o botão excluir enviar os dados necessários para a *BLL*. Novamente usando o *if* para uma mensagem de *erro* e else para uma mensagem de sucesso.

```
def Excluir(self):
    msg = QtWidgets.QMessageBox()

    self.livro.setCodigo(self.txtCodigo.text())
    LivroBLL.validaCodigo(self.livro, "d")

    if Erro.getErro():
        msg.setText(Erro.getMens())
        msg.exec()
    else:
        msg.setText("Dados excluidos com sucesso!")
        msg.exec()
```

Essa última *def* foi criada para quando o usuário apertar o botão alterar, as informações ou dados necessários sejam enviadas para a BLL. Com um *if* para importar a mensagem de erro e o *else* sendo responsável por exibir os dados corretamente.

```
def Alterar(self):
    msg = QtWidgets.QMessageBox()

    self.livro.setCodigo(self.txtCodigo.text())
    self.livro.setTitulo(self.txtTitulo.text())
    self.livro.setAutor(self.txtAutor.text())
    self.livro.setEditora(self.txtEditora.text())
    self.livro.setAno(self.txtAno.text())

    LivroBLL.validaDados(self.livro, "A")

    if Erro.getErro():
        msg.setText(Erro.getMens())
        msg.exec()
    else:
        msg.setText("Dados alterados com sucesso!")
        msg.exec()
```

3 - CLASSE BLL

Nessa classe foi importado mais duas classes: *Erro* e *LivroDAL*. Elas serão importantes para validação de todos os componentes do formulário e fazer o acesso com a API.

```
import Erro
import LivroDAL
```

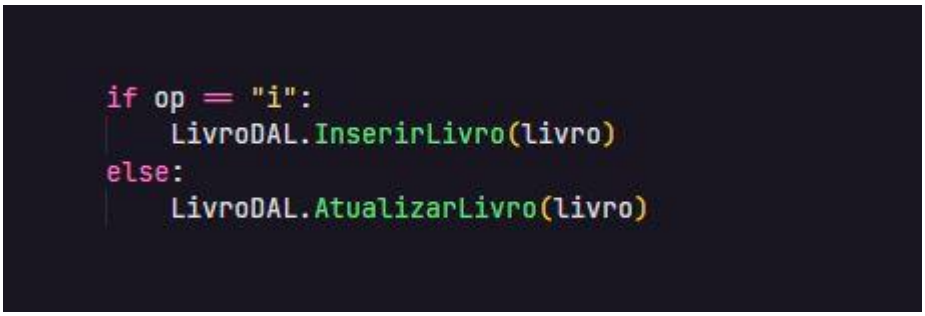
Foram necessários somente duas *def* para fazer a construção dessa classe. A primeira *def* é responsável por validar todo o código e possui duas estruturas condicionais, a primeira condição é para avisar ao usuário da aplicação sobre algum erro e a segunda condição é utilizada para fazer a análise de casos: caso seja c, consulta o livro, caso seja outra letra, deleta o livro.

```
def validaCodigo(livro, op):
    Erro.setErro(False)

    if len(livro.getCodigo()) == 0:
        Erro.setErro("O código é de preenchimento obrigatório!")
        return
    if op == "c":
        LivroDAL.ConsultarLivro(livro)
    else:
        LivroDAL.ExcluirLivro(livro)
```

Em caso de erro, através de um laço de decisão, é feita a verificação do tipo de erro, que pode ser: código, título, autor, editora não preenchidos corretamente, entre outros. Após a verificação do erro, é retornada ao usuário uma mensagem comunicando a existência da falha.

```
def validaDados(livro, op):
    Erro.setErro(False)

    if len(livro.getCodigo()) == 0:
        Erro.setErro("O código é de preenchimento obrigatório!")
        return None

    if len(livro.getTitulo()) == 0:
        Erro.setErro("O título é de preenchimento obrigatório!")
        return None

    if len(livro.getAutor()) == 0:
        Erro.setErro("O autor é de preenchimento obrigatório!")
        return None

    if len(livro.getEditora()) == 0:
        Erro.setErro("O editora é de preenchimento obrigatório!")
        return None

    if len(livro.getAno()) == 0:
        Erro.setErro("O ano é de preenchimento obrigatório!")
        return None

    try:
        int(livro.getAno())
    except:
        Erro.setErro("O valor do ano deve ser numérico!")
        return None

    if int(livro.getAno()) <= 0:
        Erro.setErro("O valor do ano deve ser numérico e positivo!")
        return None
```

Outra estrutura condicional foi feita para caso a opção seja verdadeira, portanto, o usuário que vai decidir se quer inserir um novo livro ou se irá atualizar alguma informação já existente.



```
if op == "i":
    LivroDAL.InserirLivro(livro)
else:
    LivroDAL.AtualizarLivro(livro)
```

Fazendo uma comparação com o capítulo anterior onde foi feita a conexão com o banco de dados, neste projeto foi feito a conexão com uma API, não sendo necessário usar as *defs* conectando e desconectando.

Referências

Monteiro, Leandro Pinho. "Python: características, noções e guia de estudo"; *universidade da tecnologia*. Disponível em <https://universidadedatecnologia.com.br/estudo-linguagem-python-2018/> Acesso em 05 de maio de 2021.

Brito, Edivaldo. "Conheça as características da linguagem Python"; *blog do Edivaldo*. Disponível em <https://www.edivaldobrito.com.br/conheca-as-caracteristicas-da-linguagem-python/> Acesso em 05 de maio de 2021.

Brito, Edivaldo. "Características do Python"; *eXcript*. Disponível em <http://excript.com/python/caracteristica-python.html> Acesso em 05 de maio de 2021.

SILVA, I. R. S; SILVA, R. O. Linguagem de Programação Python. Revista Tecnologias em Projeção, v. 10, n. 1, p. 56-57. 2019.