Instituto Federal de Educação Ciência e Tecnologia de São Paulo Campus Cubatão

Curso Técnico de Informática integrado ao ensino médio

Turma 448

DANIELA MANGUEIRA NASCIMENTO – GIULIA MOURA DA SILVA – JULIANA RIBEIRO DO NASCIMENTO – KAUE KENJI UEMA – MARIA EDUARDA MENDES DOS SANTOS - MARIANA BRANDÃO TRINDADE – RAQUEL DA SILVA PEREIRA

LINGUAGEM DE PROGRAMAÇÃO: PYTHON

CUBATÃO

2021

DANIELA MANGUEIRA NASCIMENTO – GIULIA MOURA DA SILVA – JULIANA RIBEIRO DO NASCIMENTO – KAUE KENJI UEMA – MARIA EDUARDA MENDES DOS SANTOS - MARIANA BRANDÃO TRINDADE – RAQUEL DA SILVA PEREIRA

LINGUAGEM DE PROGRAMAÇÃO: PYTHON

Apostila apresentada à disciplina de PJS do curso técnico em informática integrado ao ensino médio, a ser utilizado como método de avaliação.

Prof. Mauricio Neves Asenjo.

CUBATÃO

2021

Sumário

Introdução

Python é uma linguagem de programação de alto nível, sendo dinâmica, interpretada, modular, multiplataforma e orientada a objetos — uma forma específica de organizar softwares onde, a grosso modo, os procedimentos estão submetidos às classes, o que possibilita maior controle e estabilidade de códigos para projetos de grandes proporções.

Por ser uma linguagem de sintaxe relativamente simples e de fácil compreensão, ganhou popularidade entre profissionais da indústria tecnológica que não são especificamente programadores, como engenheiros, matemáticos, cientistas de dados, pesquisadores e outros.

Histórico e características da Linguagem

Idealizada e desenvolvida por Guido Van Rossum, um importante programador e pesquisador do Instituto Nacional de Matemática e Ciência da Holanda, o Python foi criado com o objetivo de otimizar a leitura de códigos e estimular a produtividade de quem os cria, seja este um programador ou qualquer outro profissional.

A ideia surgiu, como quase todas as boas ideias, de uma necessidade: a de economizar tempo no desenvolvimento e melhorar a eficiência em um projeto desenvolvido no instituto onde Guido era pesquisador. Para que esta melhoria pudesse ser feita de forma mais rápida e eficaz, Guido desenvolveu uma linguagem muito descomplicada e flexível: o Python.

Ela prioriza a legibilidade do código sobre a velocidade ou expressividade. Combina uma sintaxe concisa e clara com os recursos poderosos de sua biblioteca padrão e por módulos e frameworks desenvolvidos por terceiros.

Python é uma linguagem de propósito geral de alto nível, multiparadigma, suporta o paradigma orientado a objetos, imperativo, funcional e procedural. Possui tipagem dinâmica e uma de suas principais características é permitir a fácil leitura do código e exigir poucas linhas de código se comparado ao mesmo programa em outras linguagens. Devido às suas características, ela é utilizada, principalmente, para processamento de textos, dados científicos e criação de CGIs para páginas dinâmicas para a web.

Um de seus maiores atrativos é possuir um grande número de bibliotecas, nativas e de terceiros, tornando-a muito difundida e útil em uma grande variedade de setores dentro de desenvolvimento web, e também em áreas como análise de dados, machine learning e IA.

Foi feita com base na linguagem ABC, possui parte da sintaxe derivada do C, compreensão de listas, funções anônimas e função map de Haskell. Os iteradores são baseados na Icon, tratamentos de exceção e módulos da Modula-3, expressões regulares de Perl. Uma vez que esta linguagem passou a possibilitar a criação desde scripts muito simples até sistemas extremamente poderosos, profissionais de várias áreas começaram a progressivamente utilizá-la cada vez mais. Hoje, além dos desenvolvedores de software, temos biólogos, contadores, físicos e outros profissionais potencializando suas habilidades através dela.

A Linguagem Python também influenciou várias linguagens, algumas delas foram Boo e Cobra, que usa a indentação como definição de bloco e Go, que se baseia nos princípios de desenvolvimento rápido de Python. Atualmente, Python é um dos componentes padrão de vários sistemas operacionais, entre eles estão a maioria das distribuições do Linux, AmigaOS 4, FreeBSD, NetBSD, OpenBSD e OS X. A linguagem se tornou a padrão no curso de ciências da computação do MIT em 2009.

O Python é uma linguagem muito popular nas áreas da tecnologia relacionadas à análise de dados, pesquisa, desenvolvimento de algoritmos e IA. Podendo ser utilizada em: Scripting e automação, Desenvolvimento web, Enquadramento de testes, Big Data, Ciência de dados, Computação gráfica e Inteligência artificial. As vantagens de se dedicar ao aprendizado de Python são muitas, e entre elas está o fato de que os profissionais especializados nesta linguagem de programação no mercado são escassos. Alguns benefícios do Python são: Fácil de aprender, multiplataforma e licença de uso público. Algumas empresas que utilizam Python e têm parte de seus serviços desenvolvidos nesta linguagem são Dropbox, Spotify, Airbnb e Uber. As redes sociais Facebook, Instagram e Pinterest também têm algumas de suas funcionalidades escritas em Python. Até mesmo a NASA utiliza esta linguagem de programação. O Python é uma linguagem de programação de compreensão bastante

acessível, com uma sintaxe simples e legibilidade clara, além de ter uma aprendizagem bastante rápida. Para quem já tem alguma bagagem intelectual em lógica de programação, é possível aprender Python em apenas algumas semanas

Estes são alguns motivos que tem feito o uso do Python crescer consideravelmente nos últimos anos em detrimento de outras linguagens.

Alguns outros benefícios do Python são:

1. Simples e fácil de aprender: a curva de aprendizado de um estudante de Python é, de modo geral, relativamente baixa. A linguagem, por ter uma sintaxe muito acessível e ter sido criada em prol da agilidade e da produtividade de quem a utiliza, é absorvida rápida e facilmente.
2. Portátil, extensível e multiplataforma: por ser uma linguagem portátil e multiplataforma, o Python roda com tranquilidade em diversos sistemas operacionais, desde que seu interpretador esteja instalado. Além disso, o Python também é conhecido por suas propriedades extensíveis, tendo à sua disposição mais de 125.000 bibliotecas super versáteis.
3. Licença de uso público: em outras palavras: o Python é totalmente gratuito! Para instalar, utilizar e desenvolver em Python, basta simplesmente fazê-lo.

Além disso, a maior pesquisa realizada na área da programação, a StackOverflow Survey, perguntou este ano para desenvolvedores do mundo inteiro em qual linguagem eles mais gostam de programar e Python ficou em 1° lugar!

IDEs mais populares utilizadas

IDE, do inglês: Integrated Development Environment ou Ambiente de Desenvolvimento Integrado, é um pacote de software que juntas às ferramentas necessárias para testar e escrever softwares, tendo dito isso, as melhores IDEs para programar em Python são:

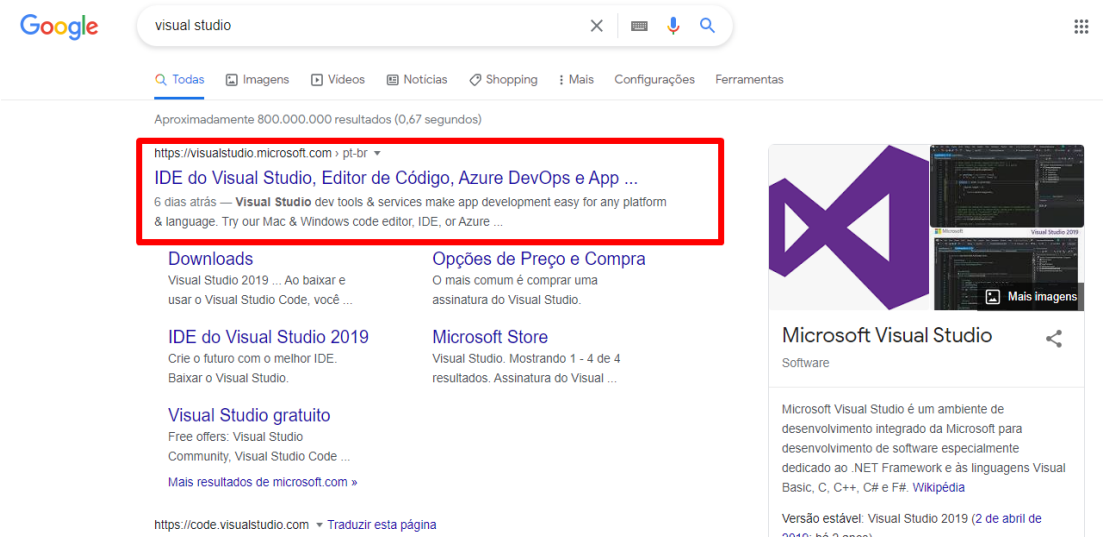
- **Visual Studio:** Essa IDE desenvolvida pela Microsoft é personalizável e possui um abundante conjunto de expressões que podem ser usados para melhorar as funcionalidades.
- **Jupyter:** Tem um ambiente de código aberto baseado na web para ajudar profissionais iniciantes. Devido à facilidade de uso ele é umas das IDEs mais utilizadas, sem contar que ela também serve como uma ferramenta de demonstrações, com isso o usuário pode ver e acessar o seu código rapidamente.
- **Pycharm:** Usado especialmente para a programação em Python, trata-se de ferramentas de análise de códigos, ferramentas de teste e opções de controle de versão. Utilizando essas ferramentas os desenvolvedores podem criar seus próprios processos como plugins, por exemplo, com a ajuda das APIs disponíveis na plataforma.
- **Atom:** Possui uma interface amigável e por conta disso é um das IDEs mais populares do mercado. Tem um código aberto para a linguagem Python, além de conter suporte integrado ao Git e funciona em diversas plataformas.

Para a realização desse projeto, decidimos utilizar, como IDE, a plataforma Visual Studio. Vários motivos nos levaram a tomar essa decisão, um deles é que: todos os participantes do grupo já conhecem e estão familiarizados com essa IDE, o que nos fornece mais tempo para

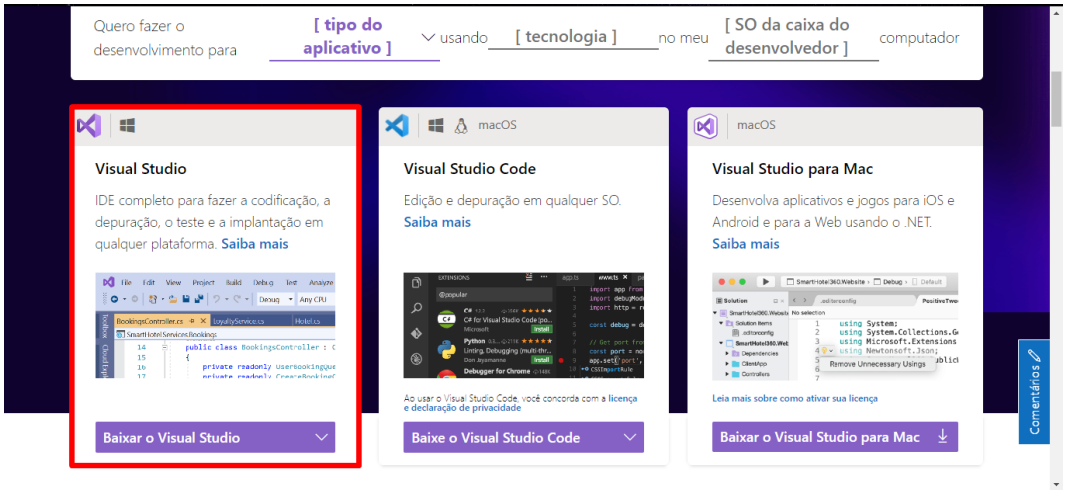
aprendermos a linguagem e desenvolver o projeto, sem ter que nos preocupar em aprender a usar uma plataforma diferente.

Outro motivo, e podemos dizer que o principal, é que essa plataforma é considerada e reconhecida como a melhor IDE existente no mercado, atualmente. É uma IDE muito completa e que nos dá as melhores ferramentas e opções para o desenvolvimento do sistema na linguagem escolhida, Phyton. A plataforma fornece um suporte ótimo e de primeira linha para a linguagem e também é a mais utilizada para programação em vários ambientes e locais.

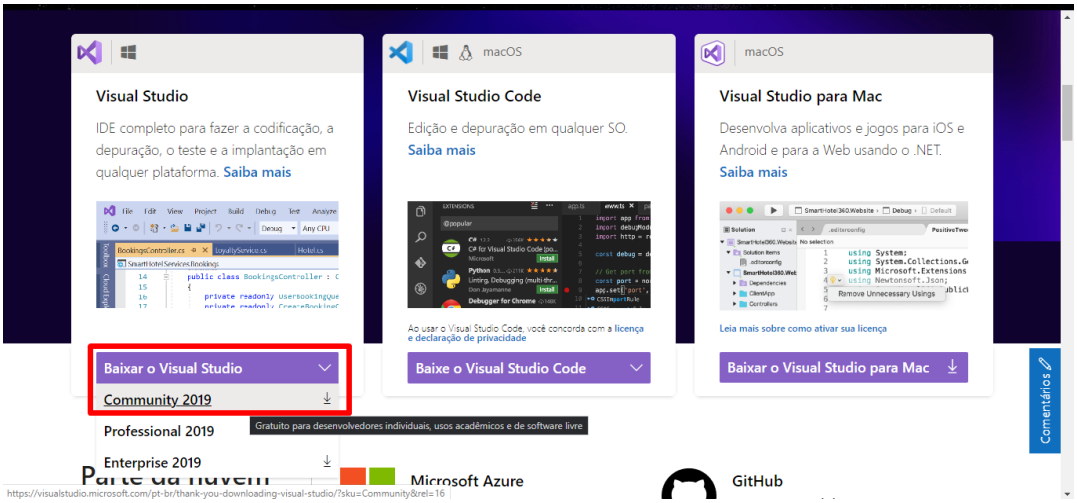
Processo de Instalação da IDE escolhida



Ao pesquisar o nome da IDE Visual Studio, clique no primeiro link que está destacado na Imagem 1

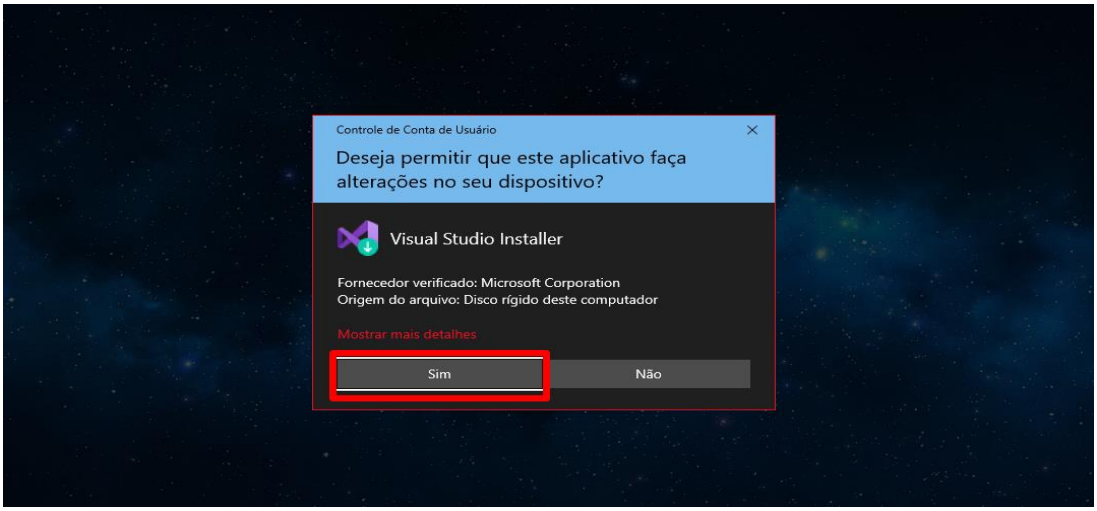


Ao entrar na página para o download, o aplicativo certo a ser instalado será o destacado na Imagem 2 para Windows, caso o computador seja de sistema MAC, o instalado será o aplicativo da direita.

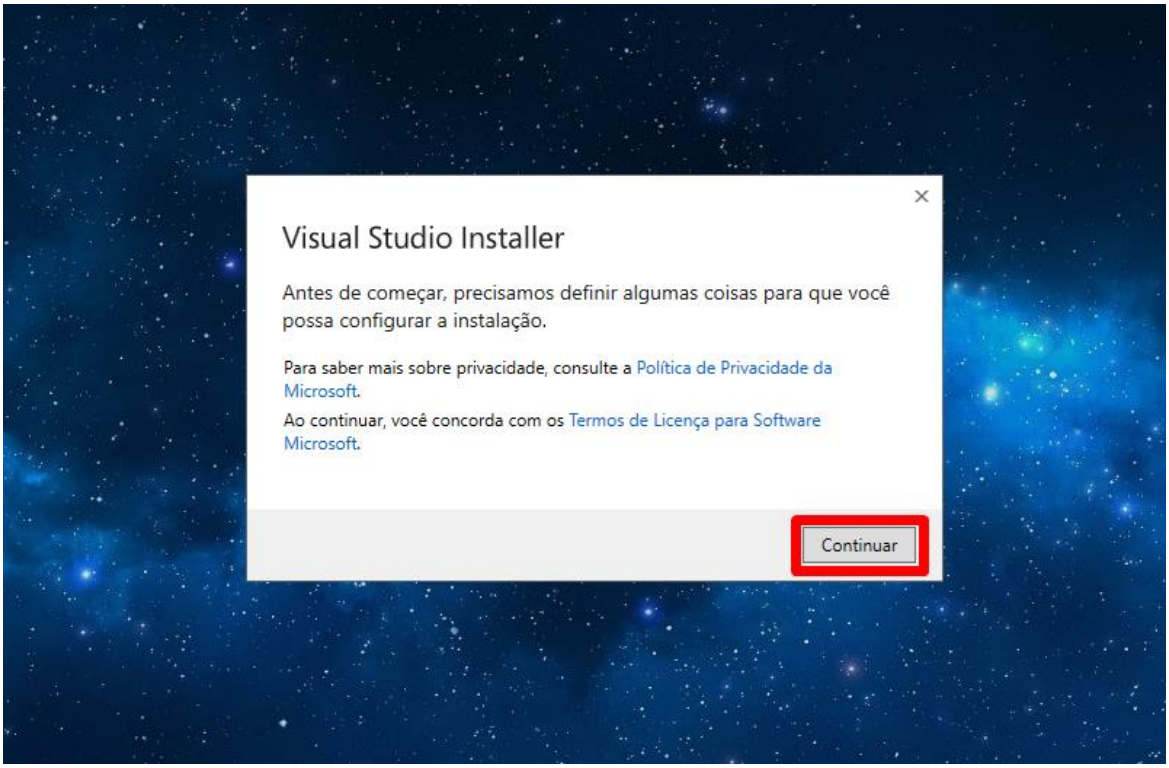


Ao arrastar o mouse no botão de Download, selecione o Visual Studio Community 2019 destacado na Imagem 3, já que é a ultima versão lançada gratuitamente para desenvolvedores independentes, uso acadêmico e de software livre.

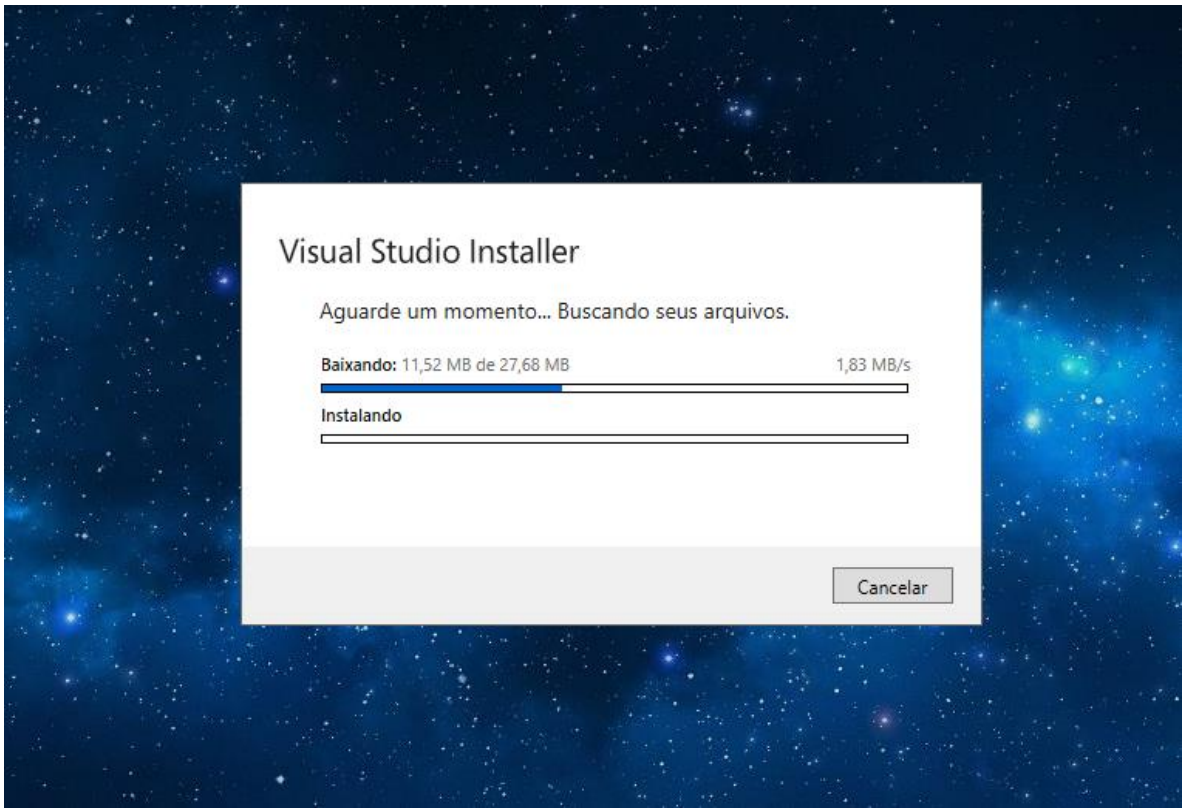
Após fazer o download, esta página será aberta, então, dê um duplo clique no arquivo vs_community_12...exe destacado na Imagem 4 para executar o instalador.



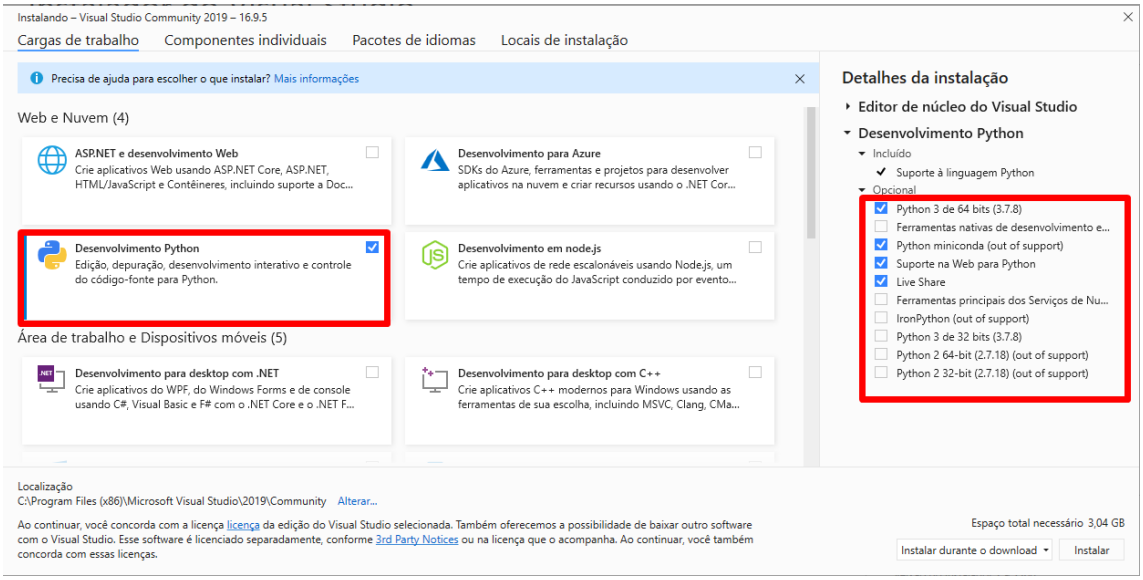
Permita as alterações no dispositivo referentes à imagem 5.



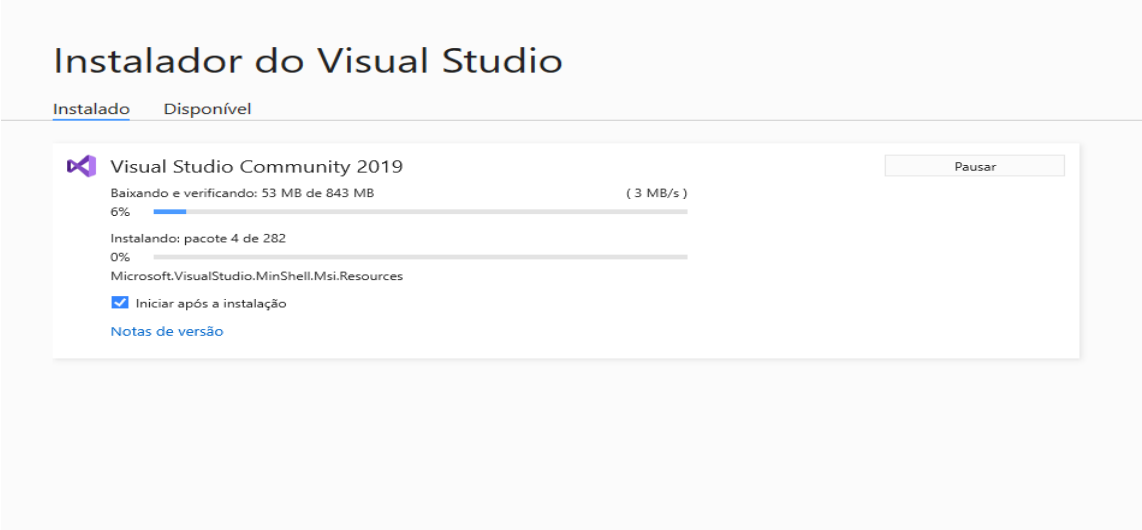
Esta janela do instalador será aberta, clique em Continuar de acordo com a Imagem 6.



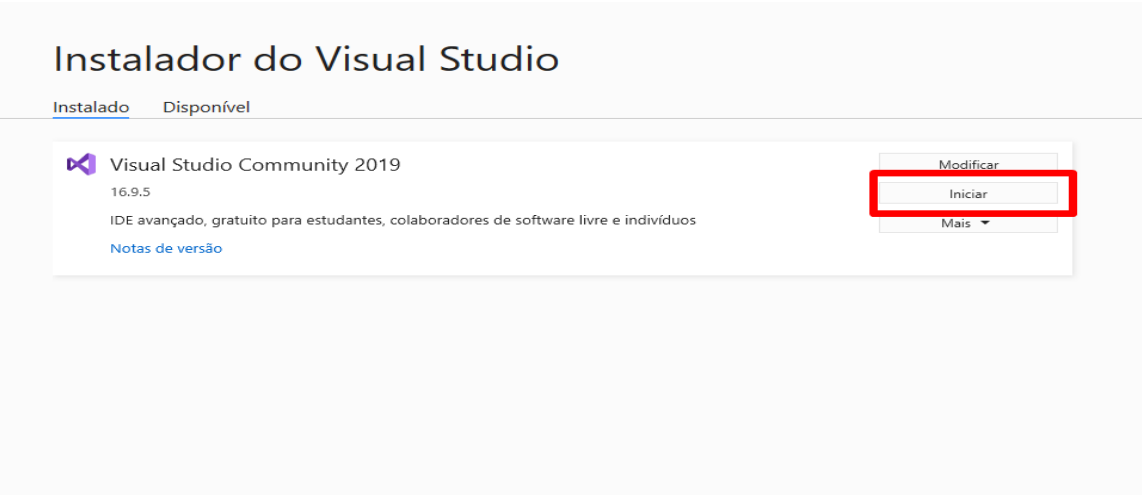
Aguarde a instalação dos arquivos representado na Imagem 7.



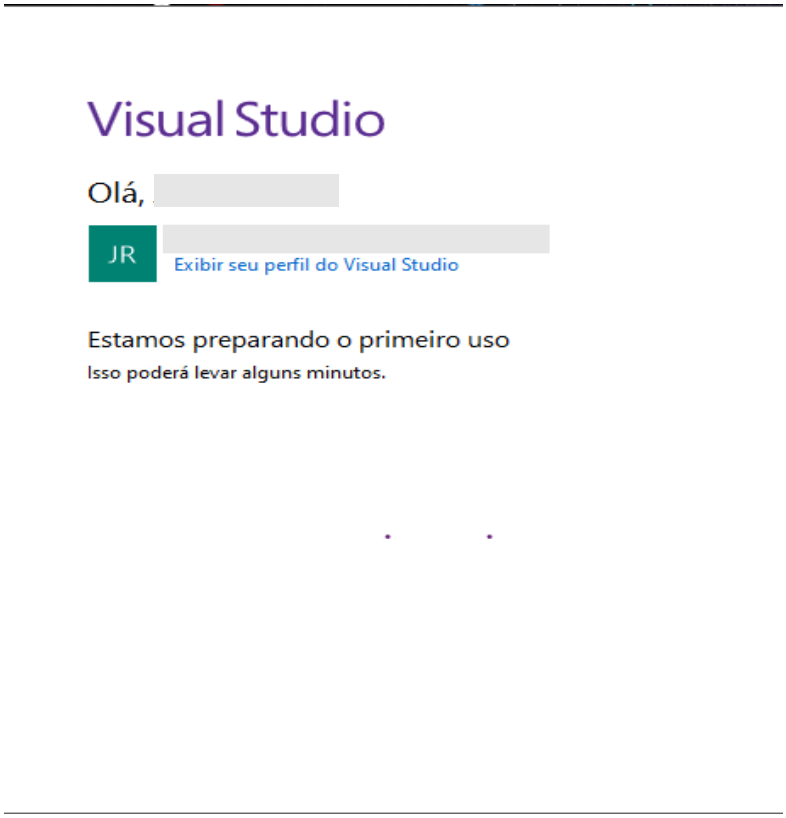
Após a instalação, a página da Imagem 8 será aberta, certifique-se que as configurações destacadas estejam devidamente selecionadas e clique em instalar.



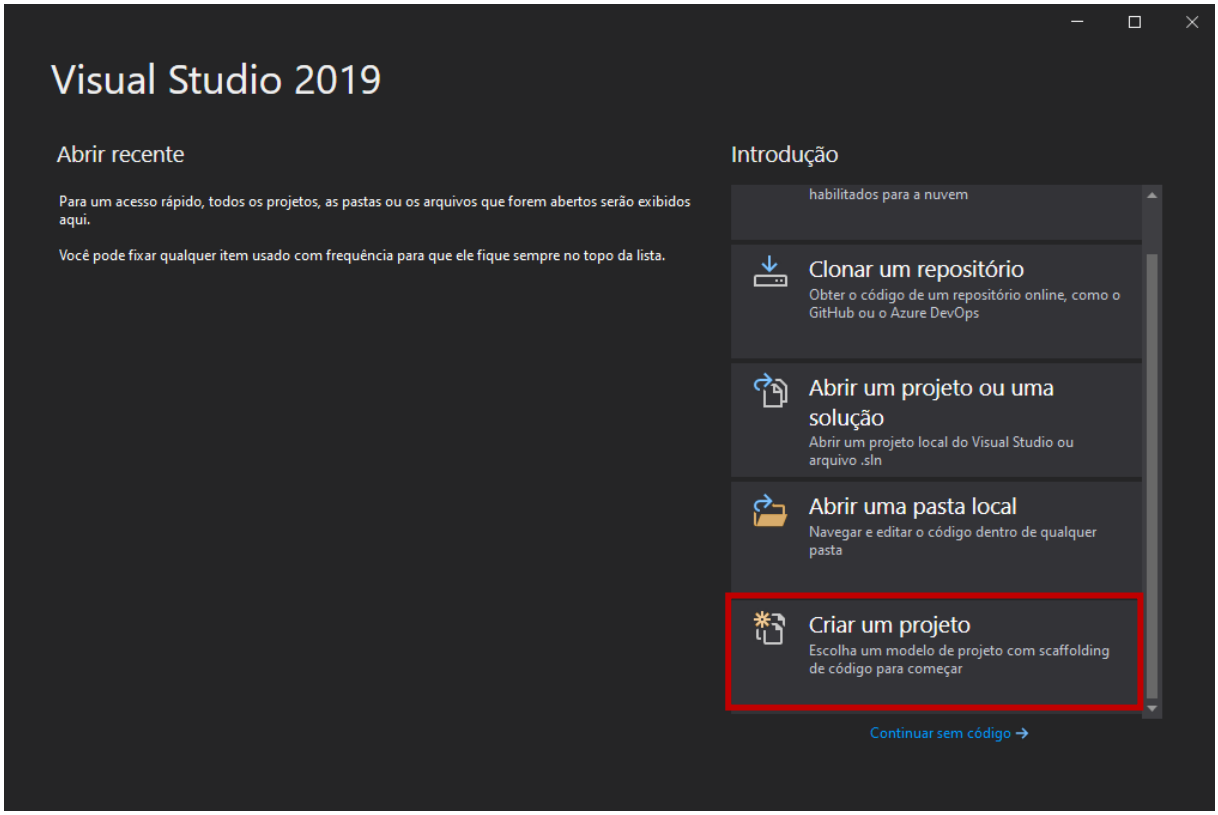
Aguarde a instalação das configurações necessárias, representadas na Imagem 9, esta ação pode demorar um pouco.



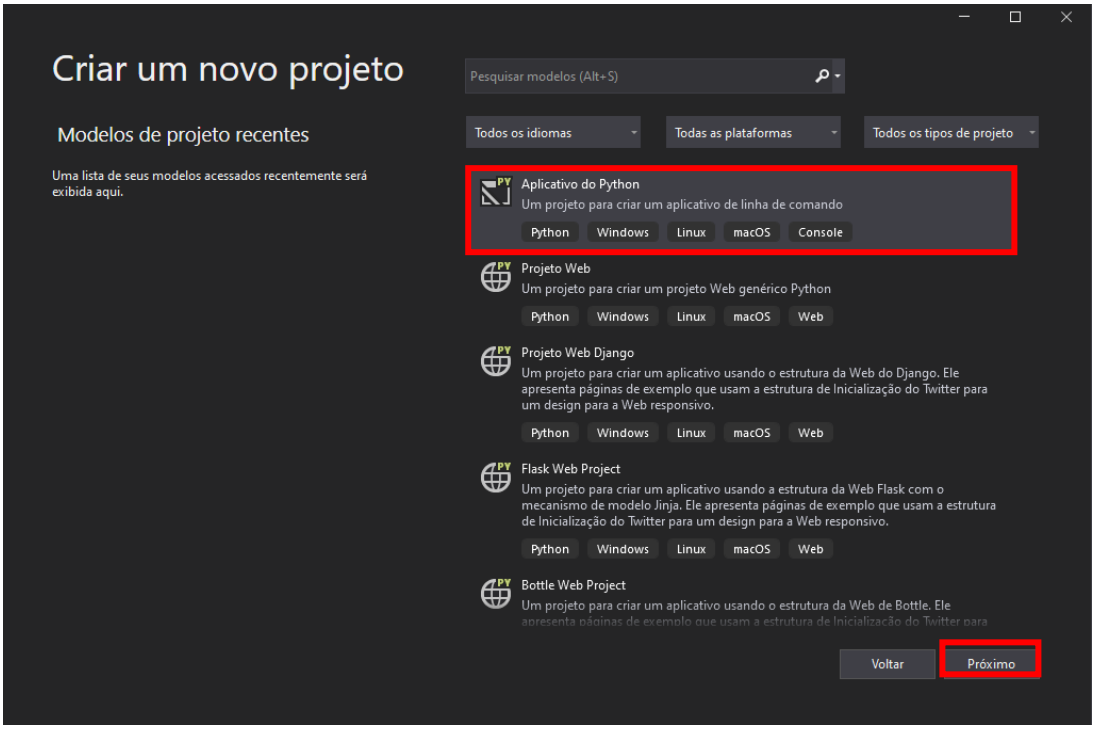
Após a instalação, clique em Iniciar o programa, de acordo com a Imagem 10.



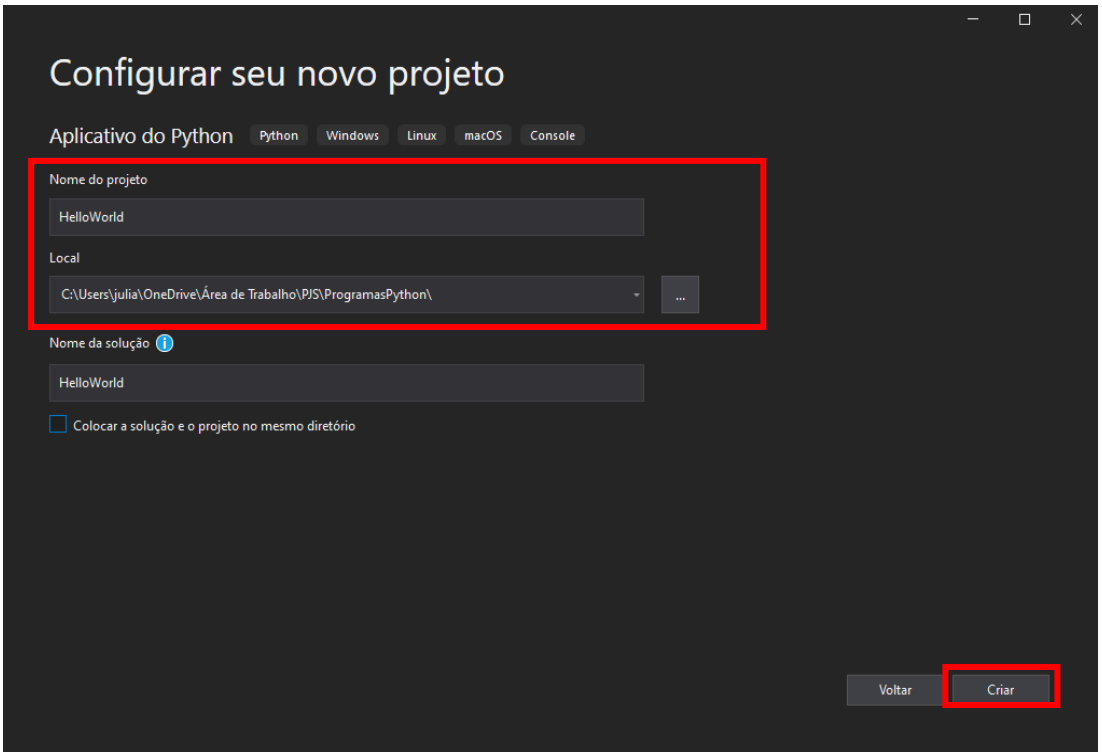
Após a inicialização do programa, a tela representada na Imagem 11 será exibida, onde você se conectará à sua conta Microsoft (a nossa já estava conectada) e, logo após inseri-la, aguarde.



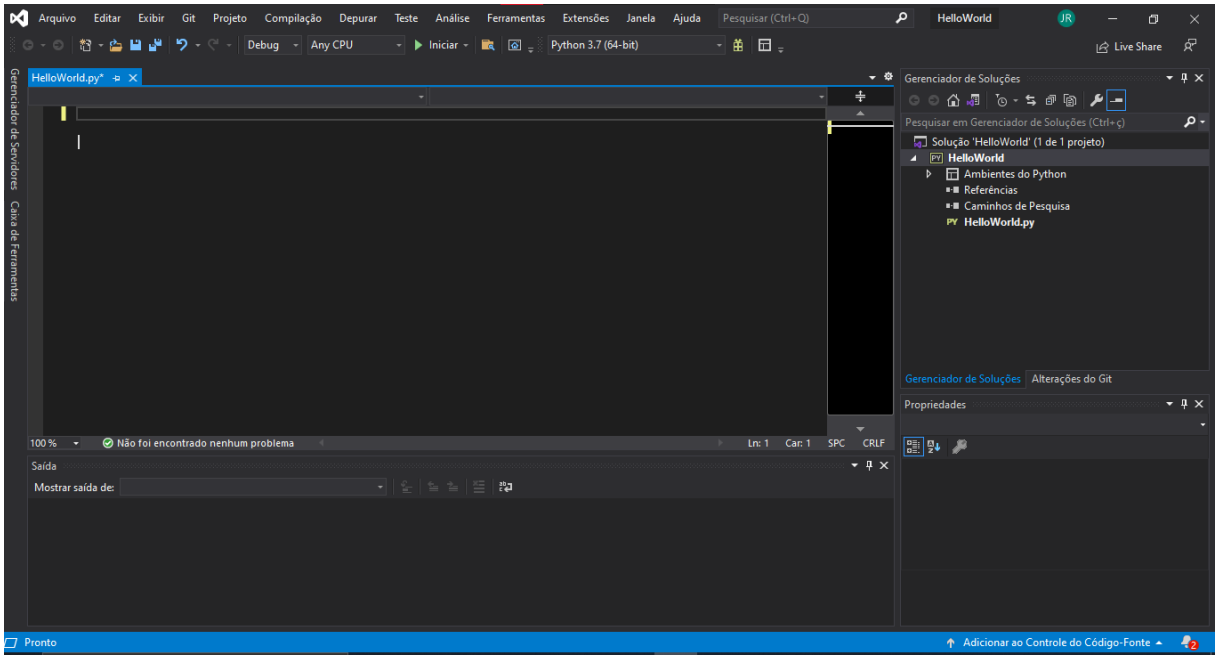
A Imagem 12 nos mostra a tela de inicialização do Visual Studio. Para começar a programar, basta clicar em "Criar um projeto", cujo está destacado em vermelho.



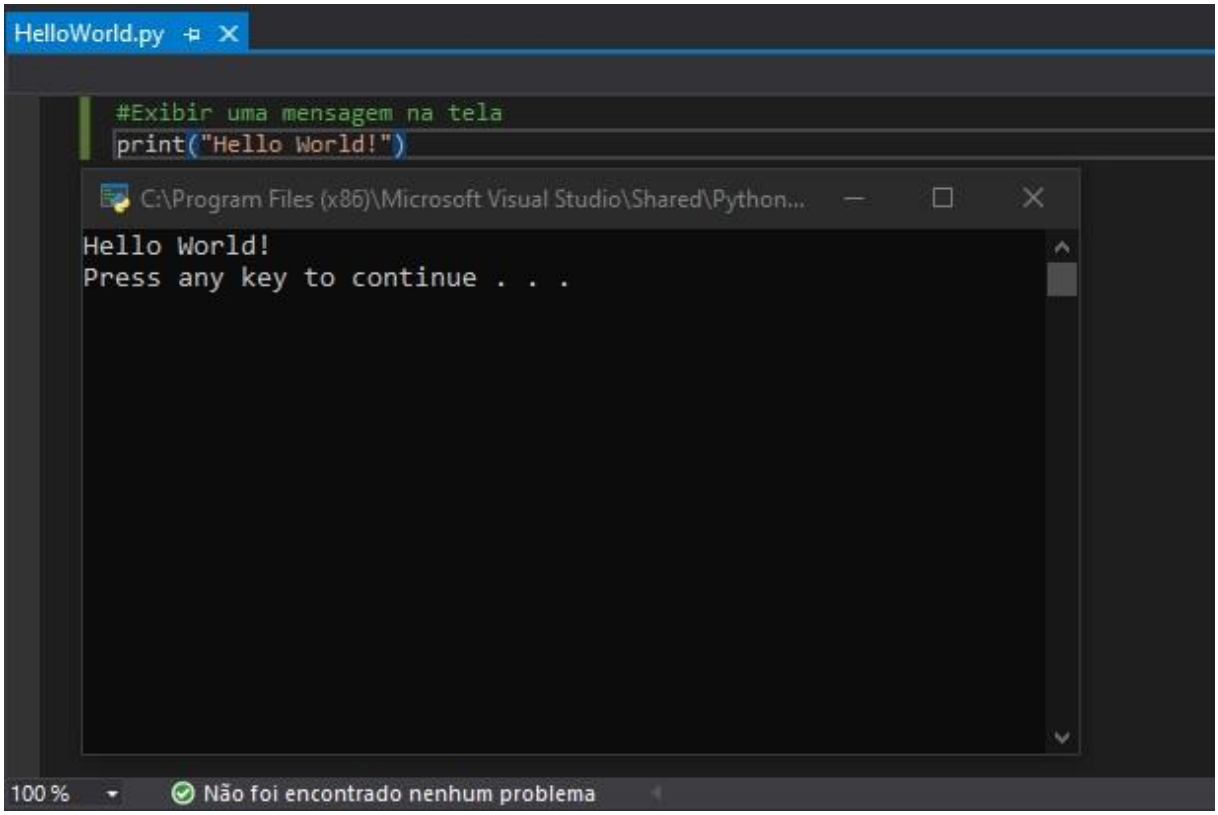
Logo após selecionar “Criar um projeto”, escolha a opção Aplicativo do Python, que está em destaque na Imagem 13, e clique em Próximo.



Logo após, configure seu novo projeto com o Nome e o Local onde ele ficará armazenado em seu computador de acordo com a Imagem 14. Depois, clique em Criar.



Após a Criação do Projeto, a Imagem 15 representa a página que será aberta, onde podemos começar a programar.



Na imagem 16, foi criado um projeto chamado 'Hello World', código esse que é uma "regra" informal no mundo da informática como primeiro projeto.

Comandos de Entrada e Saída via console

Agora que já vimos sobre o processo de instalação e configuração da IDE escolhida, falaremos sobre comandos de E/S via console, que são os comandos mais básicos e essenciais para quem está começando a vida na programação. Os comandos de entrada e saída de um programa disponíveis nas linguagens de programação possuem a função de comunicação com o mundo exterior, isto é, o comando de entrada permite que o usuário envie dados do mundo exterior (p.ex. teclado) para uma variável do programa (variável associada ao comando) e o comando de saída permite exibir (p. ex. na tela do computador) mensagens, expressões e/ou valores armazenados em variáveis do programa. Na linguagem C#, os comandos de E/S eram: Console.Write e Console.Writeline, para exibir algo na tela, e Console.Read, para ler algum dado que o usuário digitou e atribuí-lo à uma variável.

Comandos de Entrada em Python

O comando (função) de entrada do Python é nomeado como: input. Este comando interrompe a execução do programa para esperar que o usuário digite o valor solicitado no teclado e, ao final da digitação, pressione a tecla "Enter". Após isso, o valor digitado será movido para variável associada ao input.

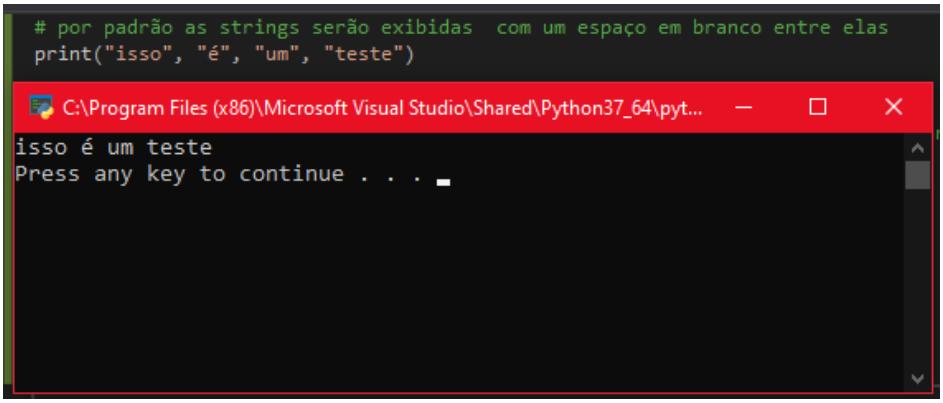
Forma Geral: **variável = input('Mensagem para o usuário')**

Comandos de Saída em Python

Esse comando (função) permite exibir na tela do usuário mensagens, expressões e/ou valores armazenados em variáveis do programa.

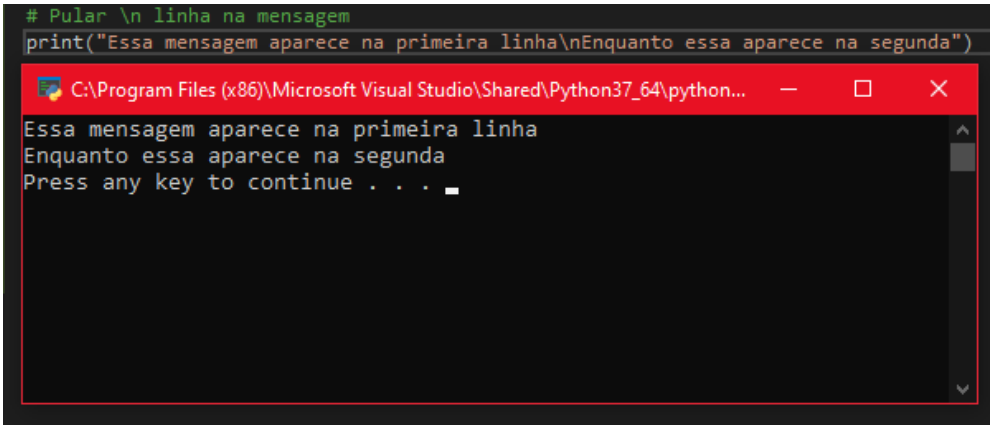
Forma Geral: **print (valores, [sep = ‘ ‘, end = ’ ‘, file = sys.stdout])**

- Valores: lista de mensagens e/ou variáveis (caso haja mais que uma, separar por vírgulas)

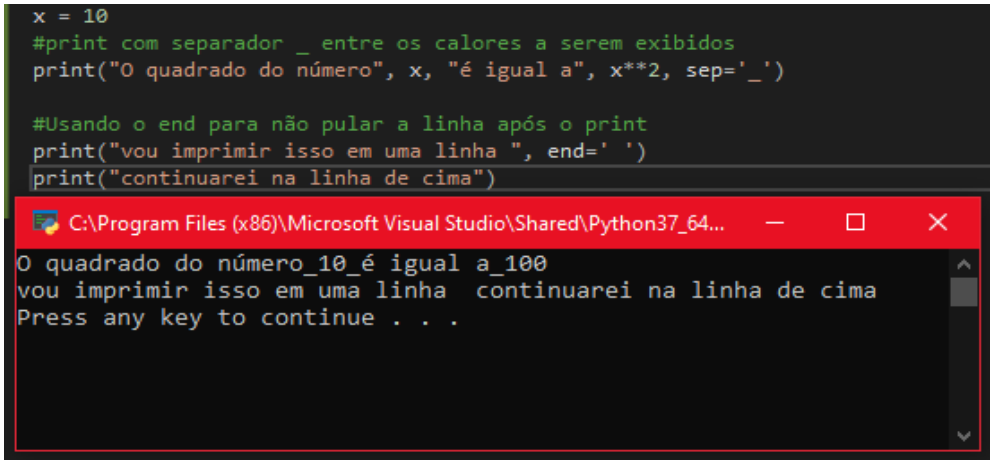


Itens opcionais:

- Sep: indica o tipo de separador que pode aparecer entre as strings e/ou valores;



- End: indica a string que será concatenada ao final do print. Padrão: Enter('\n'))



- File: indica onde serão enviados os valores do print. Padrão: Console(sys.stdout).

Variáveis

Variáveis servem para dar nome a posições de memória que irão armazenar dados de um programa. Por exemplo, o comando de atribuição:

NotaP1 = 7.5

Cria-se uma variável de nome **NotaP1** que armazena o valor real **7.5**. A partir do momento em que foi criada uma variável ela pode ser utilizada em expressões aritméticas e atribuídas a outras variáveis, por exemplo:

NotaP2 = 8.0
Media = (NotaP1+NotaP2)/2

Há também regras específicas para a criação de variáveis dentro da linguagem Python. Sendo elas:

- O nome das variáveis deve obrigatoriamente começar com uma letra;
- Deve-se utilizar nomes significativos dentro do contexto do programa;

  Exemplos de variáveis válidas:

  - Idade, Contador, PesoDoCarro, Usuario_1, CorDaPagina, RaioDoCirculo

  Exemplo de variáveis inválidas:

  - salario medio: não pode conter espaços;
  - 1a: não pode começar com números;

- Python é uma linguagem *case-sensitive*, ou seja, faz diferença entre nomes com letras maiúsculas e nomes com letras minúsculas. **Peso** e **peso** são variáveis diferentes;
- Pode-se usar letras maiúsculas e minúsculas para separar palavras dentro do nome de uma variável: "PesoDoCarro";
- O nome da variável deve ser diferente dos comandos da linguagem (if, for, ...).

Tipos de Variáveis

As variáveis em Python têm um **tipo**, que é definido no momento em que a variável é criada por um comando de atribuição. Cada tipo define os valores que a variável pode armazenar e ocupa uma certa quantidade de memória. Inicialmente, trataremos de 4 tipos de variáveis:

- **Inteiros**: 45, 30 -1034
- **Reais**: 18.34, 90.33, -2345.6543
- **Strings**: armazenam sequências de caracteres como nomes de pessoas ou lugares, frases, mensagens. As strings são identificadas no código por serem escritas entre aspas ("Uma string") ou apóstrofos ('Joao da Silva').
- **Lógico**: estas variáveis podem armazenar os valores especiais **True** e **False**, e servem para armazenar o resultado de uma condição como: **Teste = (3<2)**, que resulta em **False**. Maiores detalhes sobre estas variáveis serão apresentados quando tratarmos de comandos de decisão ou seleção.

Por padrão, o comando input move valores do tipo String para as variáveis associadas ao comando. Mas, caso o usuário queira que um valor numérico seja movido para a variável, deverá ser explicitado no comando, conforme exemplos, a seguir:

- Para solicitar um número inteiro: nome da variável = int(input('Texto a ser mostrado '))
- Para solicitar um número real: nome da variável = float(input('Texto a ser mostrado '))

Operadores Aritméticos e Funções Matemáticas

Os operadores aritméticos são utilizados na execução de operações matemáticas, tais como a soma e a subtração, por exemplo. Vejamos na **Tabela 1** a lista deles.

Operador	Conceito	Exemplo
+ (Adição ou sinal positivo)	- Realiza a soma entre operandos - Adiciona o sinal de positivo ao número	- 10 + 7 - +4
- (Subtração ou sinal negativo)	- Realiza a subtração entre operandos - Adiciona o sinal de negativo ao número	- 10 - 7 - -4
* (Multiplicação)	Realiza a multiplicação entre operandos	3 * 4
/ (Divisão)	Realiza a divisão entre operandos	10 / 5
// (Divisão inteira)	Realiza a divisão entre operandos e a parte decimal do resultado	10 // 6
% (Módulo)	Retorna o resto da divisão entre operandos	4 % 2

Operador	Conceito	Exemplo
** (Exponenciação)	Retorna um número elevado a potência de outro	4 ** 2

Uma característica importante a ser observada quando falamos dos operadores matemáticos é a precedência. Essa característica é relativa à ordem da execução deles e acontece quando mais de um operador está presente numa expressão. Segue a precedência dos operadores no Python.

“Nos exemplos abaixo usaremos a função print() para exibir os resultados das expressões.”

1. As expressões contidas em parênteses têm a precedência maior na linguagem Python. Isso permite que uma expressão execute antes de outra. Ex.:

```
print((2 + 5) * 3) # O resultado será 21
```

2. Após os parênteses, o próximo operador com maior precedência é o de exponenciação. Ex.:

```
print( 1 + 5**2) # O resultado será 2
```

3. Multiplicação e divisão têm precedência sobre a adição e subtração: como já é conhecido na matemática, divisão e multiplicação são executadas antes das operações de adição e subtração. Ex.:

```
print(5 * 3 + 8) # O resultado será 23
```

4. Ordem de precedência é avaliada da esquerda para a direita. Portanto, após os operadores anteriores, a sequência da execução será da esquerda para a direita. Ex.:

```
print(8 + 5 - 10) # O resultado será 3
```

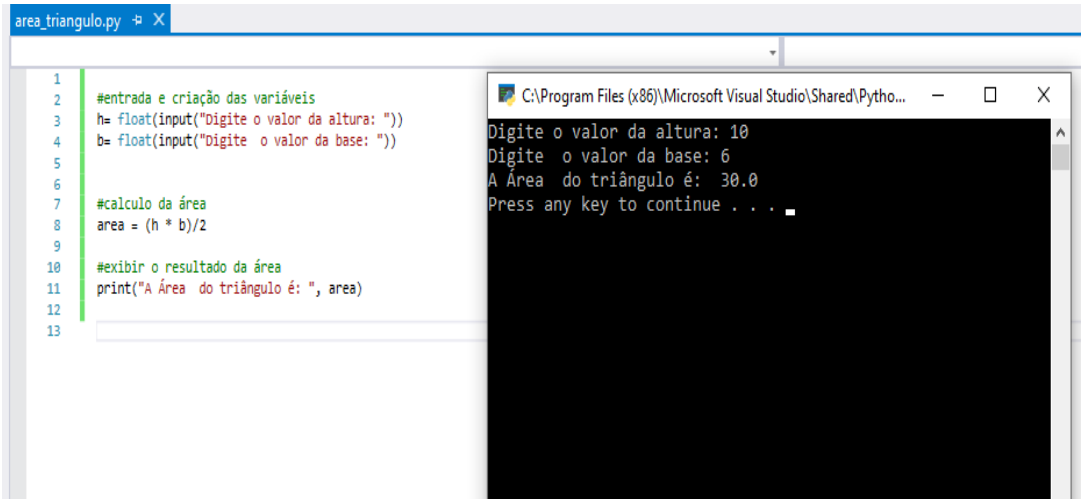
Ainda falando de Matemática, dentro do Python temos uma biblioteca diversa de funções matemáticas, e para utilizá-las, é necessário que a biblioteca Math esteja ativa no programa, para isso, utilize o import math em seu código antes das operações desejadas.

Segue abaixo uma lista das funções Math. mais utilizadas:

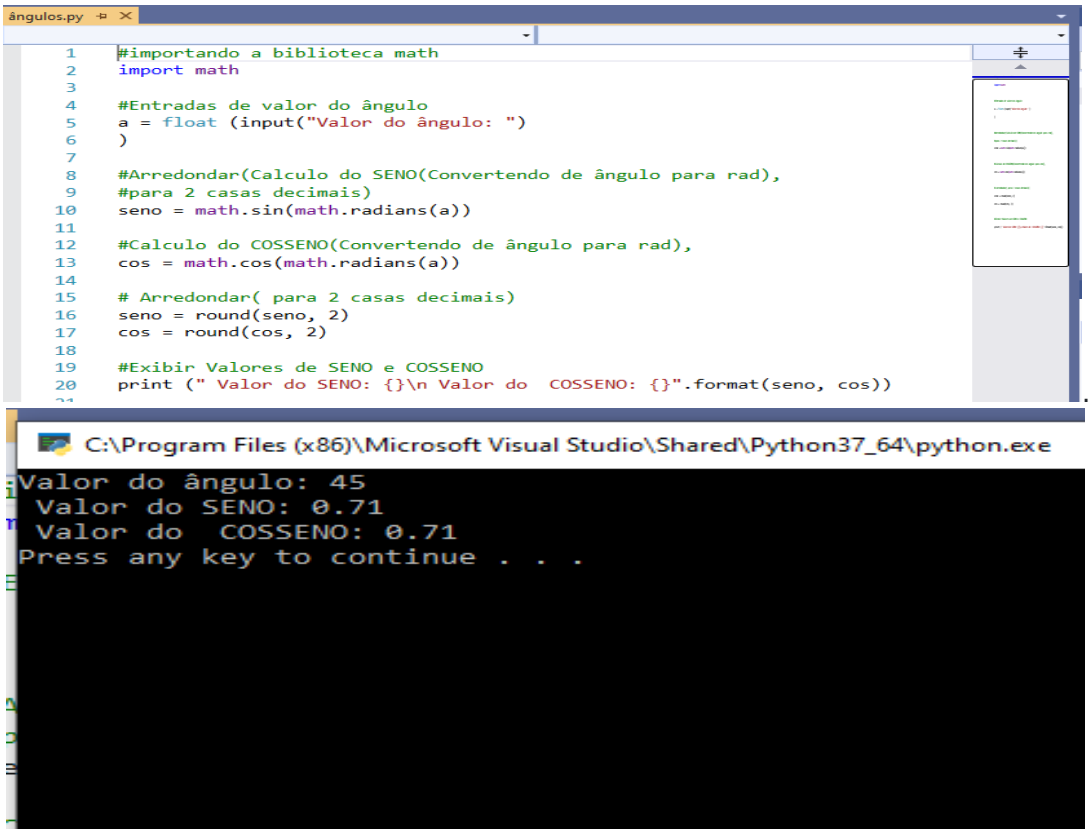
- math.sqrt(numero): Retorna a raiz quadrada do número
- math.cos(numero): Retorna o cosseno do número em radianos.
- math.sin(numero): Retorna o seno do número em radianos.
- math.tan(numero): Retorna a tangente do número em radianos.
- math.radians(numero): converte o ângulo ‘numero’ de graus para radianos.
- math.pi: Não é bem uma função, está mais para uma variável com o número pi.
- math.hypot(x,y): Retorna a hipotenusa dos números (catetos) fornecidos.

Exemplos de Fixação comandos de E/S e Operações Aritméticas.

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



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



Operadores Lógicos e de Comparação

O tipo de dados primitivo mais simples é o chamado booleano (ou lógico). Pra quem não conhece esse tipo de dados, um dado booleano só pode assumir dois valores (verdadeiro ou falso). As operações lógicas trabalham sobre valores booleanos, tanto os valores de entrada como o de saída são desse tipo. Exemplo:

Operador	Conceito	Exemplo
and	Retorna True se todas as condições forem verdadeiras, caso contrário retorna False	x > 1 and x < 5
or	Retorna True se uma das condições for verdadeiras, caso contrário retorna False	x > 1 or x < 5
not	Inverte o resultado: se o resultado da expressão for True, o operador retorna false	not(x > 1 and x < 5)

Os operadores de comparação são usados para comparar valores, o que vai retornar true ou false, dependendo da condição. Usando a comparação que é feita com esses operadores, podemos criar condições para os códigos. Exemplo:

Operador	Conceito	Exemplo
>(Maior que)	Verifica se um valor é maior que outro	x > 5
<(Menor que)	Verifica se um valor é menor que outro	x < 5
== (Igual a)	Verifica se um valor é igual a outro	x == 5
!= (Diferente de)	Verifica se um valor é diferente de outro	x != 5
>= (Maior ou igual a)	Verifica se um valor é maior ou igual a outro	x >= 5
<= (Menor ou igual a)	Verifica se um valor é menor ou igual a outro	x <= 5

Ao usar os operadores no Python, podemos atribuir valores a variáveis, criar condições, fazer comparações etc. Essas características fazem parte dos princípios básicos para montar algoritmos nessa linguagem. É importante dominar esses conceitos sobre os operadores, pois eles são amplamente usados no dia a dia do desenvolvedor.

Estruturas de Decisão

Comando de Seleção IF: este comando, traduzido como ‘se’, seleciona qual parte do código será executada a seguir, estabelecendo uma condição para que ele seja realizado. Por exemplo, o programa a seguir informa se o aluno está aprovado por média se a nota for maior igual a 7.

```
nota = float(input("Digite sua nota: "))

if nota >=7.0:
    print ("Você está aprovado por média.")

if nota < 7.0:
    print ("Você não está aprovado por média.")
```

Nota-se que o comando que está logo abaixo do IF está devidamente indentado, o que é obrigatório em Python, e, caso não seja feito, um erro será gerado como no exemplo a seguir:

```
a = input("Digite sua nota: ")
nota = float (a)

if nota >=7.0:
print ("Você está aprovado por média.")

if nota  < 7.0:
   print ("Você não está aprovado por média.")
```

A linha destacada em vermelho, não contém a indentação em seu início, feito com a tecla TAB, o que irá gerar um erro na execução, como este:

```
File "Decisao1.py", line 7
   print ("Você está aprovado por média.")
      ^
IndentationError: expected an indented block
```

Comando de Seleção IF-ELSE: Para que a sequencia de IFs não fique muito longa, é possível adicionar o comando else ao comando if. Como o modelo abaixo:

```
a = input("Digite sua nota: ")
nota = float (a)

if nota >=7.0:
   print ("Você está aprovado por média.")
else:
   print ("Você não está aprovado por média.")
```

Utilizando o If-Else o código executado após o teste, irá depender do valor da condição dada após o IF. Caso seja verdadeira, será executado o que está na linha seguinte. Note que após o comando e a condição, há um sinal de dois pontos para que sejam identificados o bloco de códigos. Além do símbolo de dois pontos no fim da linha, deve-se colocar todos os códigos pertencentes aquele bloco deslocados/alinhados à direita, indentados.
Quando é preciso tomar uma decisão, após outra ser tomada, usa-se o aninhamento de IFs. No exemplo abaixo, caso a nota do aluno seja menor do que 7, precisa-se testar se ele tem uma nota maior ou igual a 4 para que possa ser realizada a prova G2.

```
a = input("Digite sua nota: ")
nota = float (a)

if nota >=7.0:
   print ("Você está aprovado por média.")
   if nota>9.0:  # IF ANINHADO
      print ("Parabéns!")  # se nota > 9
   print ("Boas Ferias!")  #  se nota >=7
else:
   if nota>=4: # IF ANINHADO
      print ("Você pode fazer G2.");
      print ("Venha na próxima semana")
   else:
      print ("Você está reprovado!")
      print ("Você não pode fazer G2.")
```

Comando ELIF: Para evitar o aninhamento excessivos de blocos de IF's, é possível utilizar o comando ELIF, como no exemplo a seguir. É notável que todos os ELIFs ficam alinhados com o primeiro IF, e o ELSE no fim, é executado caso nenhuma das condições anteriores sejam verdadeiras.

```
a = input("Digite a idade da pessoa: ")
idade = int (a)

if idade <=1:
   print ("Recém nascido")
elif idade < 13:
      print ("Criança")
elif idade < 18:
   print ("Adolescente")
elif idade < 60:
   print ("Adulto")
elif idade < 80:
   print ("Idoso")
else:
   print ("Longevo")

print ("Acabou.")
```

Switch-Case: Python não tem uma construção como switch/case, em vez disso, a forma mais simples de se substituir esse comando é utilizando as opções acima. Porém, está sendo feita uma pesquisa para implementá-lo nas próximas versões de Python.

Exercícios de Fixação: Operadores Lógicos, Operadores de Comparação e Estruturas de Decisão.

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 está aprovado, menor que três reprovado, entre 3 e 6 de exame.

```
#Atribuindo os dados que o usuário digitar a uma variável
#o float(input()) converte uma variável de string para real
n1 = float(input("Digite a sua nota do primeiro bimestre: \n"))
n2 = float(input("Digite a sua nota do segundo bimestre: \n"))
n3 = float(input("Digite a sua nota do terceiro bimestre: \n"))
n4 = float(input("Digite a sua nota do quarto bimestre: \n"))

#calculando a média do aluno, atribuindo-a à uma variável
#coloca-se as quatro notas em parenteses para que a soma seja feita antes da divisão
final = (n1+n2+n3+n4)/4

#executa-se uma estrutura de repetição if para verificar se o aluno
#foi aprovado,reprovado ou se está de exame
if final < 3:
    print("Sua nota foi: ", final, "\n Você foi Reprovado!" )
else:
    if 6 > final > 3:
        print("Sua nota foi: ", final, "\n Você está de exame!" )
    else:
        if final >= 6:
            print("Sua nota final foi: ", final, "\n Você foi Aprovado!" )
#obs. a Identação no Python é extremamente necessária para identificar um bloco de códigos.
```

```
C:\Program Files (x86)\Microsoft Visual Studio\Shared\Python37_64\python.exe
Digite a sua nota do primeiro bimestre:
5
Digite a sua nota do segundo bimestre:
7
Digite a sua nota do terceiro bimestre:
8
Digite a sua nota do quarto bimestre:
4
Sua nota final foi:  6.0
 Você foi Aprovado!
Press any key to continue . . .
```

Teste em que a nota do aluno foi suficiente para passar

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.

```python
#Pedindo para que o usuário entre com dois números e um caractere
a= float(input("Digite o primeiro numero: \n"))
b= float(input("Digite o segundo numero: \n"))
c= input("Digite a operação (+, -, *, /): \n")

#utilizando de um ninho de ifs para verificar qual caractere foi utilizado
#dependendo do caractere utilizado, utiliza-se a variável resultado para armazenar o valor
if c == '+':
    resultado = a+b
    #Exibindo o valor final com um print
    print("O resultado é: ", resultado)
else:
    if c == '-':
        resultado = a-b
        print("O resultado é: ", resultado)
    else:
        if c == '*':
            resultado = a*b
            print("O resultado é: ", resultado)
        else:
            if c == '/':
                resultado = a/b
                print("O resultado é: ", resultado)
            else:
                #caso nenhum dos caracteres seja digitado, exibir mensagem de erro
                print("Erro! Operação Inválida!")
```

```
C:\Program Files (x86)\Microsoft Visual Studio\Shared\Python37_64\python.exe
Digite o primeiro numero:
5
Digite o segundo numero:
4
Digite a operação (+, -, *, /):
+
O resultado é:  9.0
Press any key to continue . . .
```

Teste com a operação soma

5. Idem ao exercício anterior, porém agora a solução deverá ser desenvolvida usando um elif, já que o switch case ainda não é suportado em Python.

```python
#Pedindo para que o usuário entre com dois números e um caractere
a= float(input("Digite o primeiro numero: \n"))
b= float(input("Digite o segundo numero: \n"))
c= input("Digite a operação (+, -, *, /): \n")

#utilizando elif para verificar qual caractere foi utilizado
#dependendo do caractere utilizado, utiliza-se a variável resultado para armazenar o valor
if c == '+':
    resultado = a+b
    #Exibindo o valor final com um print
    print("O resultado é: ", resultado)
elif c == '-':
     resultado = a-b
     print("O resultado é: ", resultado)
elif c == '*':
     resultado = a*b
     print("O resultado é: ", resultado)
elif c == '/':
     resultado = a/b
     print("O resultado é: ", resultado)
else:
     #caso nenhum dos caracteres seja digitado, exibir mensagem de erro
     print("Erro! Operação Inválida!")
```

```
C:\Program Files (x86)\Microsoft Visual Studio\Shared\Python37_64\python.exe
Digite o primeiro numero:
4
Digite o segundo numero:
3
Digite a operação (+, -, *, /):
2
Erro! Operação Inválida!
Press any key to continue . . .
```

Teste com operação inválida

Laços de Repetição: Estrutura For

A estrutura de repetição “for” também conhecida como laço de repetição, é responsável por executar um bloco de código por determinada quantidade de vezes, e ele não é o único laço de repetição disponível para uso na linguagem, o “while” também se encaixa nessa categoria.

A forma mais simples do comando for é a seguinte:

For variável in lista:
    <comandos>

Enquanto percorremos a lista de valores, a variável indicada no for receberá um item da coleção. Assim podemos executar algum comando com esse elemento. No código abaixo, percorremos a lista nomes e imprimimos cada elemento.

```
nomes = ['Pedro', 'João', 'Leticia']

for n in nomes:

    print(n)
```

A variável definida na primeira linha é uma lista com uma sequencia de valores do tipo string. O for percorre todos esses elementos e atribui o valor do item à variável n, que é impressa em seguida. O resultado, então é a impressão de todos os nomes contidos na lista. É possível também definir com números, quantas vezes aquele código dentro do for será executado. Para isso utilizamos um comando chamado range(x,y).

A sintaxe do for utilizando o range ficaria então assim: for n in range (x,y), onde o x é o numero onde começa a ser feito o loop, e y é onde o loop vai parar.

```
# Aqui repetimos o print 3 vezes
 for n in range(0, 3):
...     print(n)
...
0
1
2
```

Na linguagem Python, a escolhida por nós, é possível e permitido diferentes variações dentro de um laço, e não só de um em um, como aconteceu nos programas acima. No entanto, para fazer a variação diferente de 1, ou seja, mudar o incremento ou decremento do laço, é mais aconselhado fazer o uso da estrutura de repetição while. Podemos ver isso com mais clareza no exemplo abaixo:

```
Lista = 0
While (lista<=10):

    Lista += 2

    Print(lista)
```

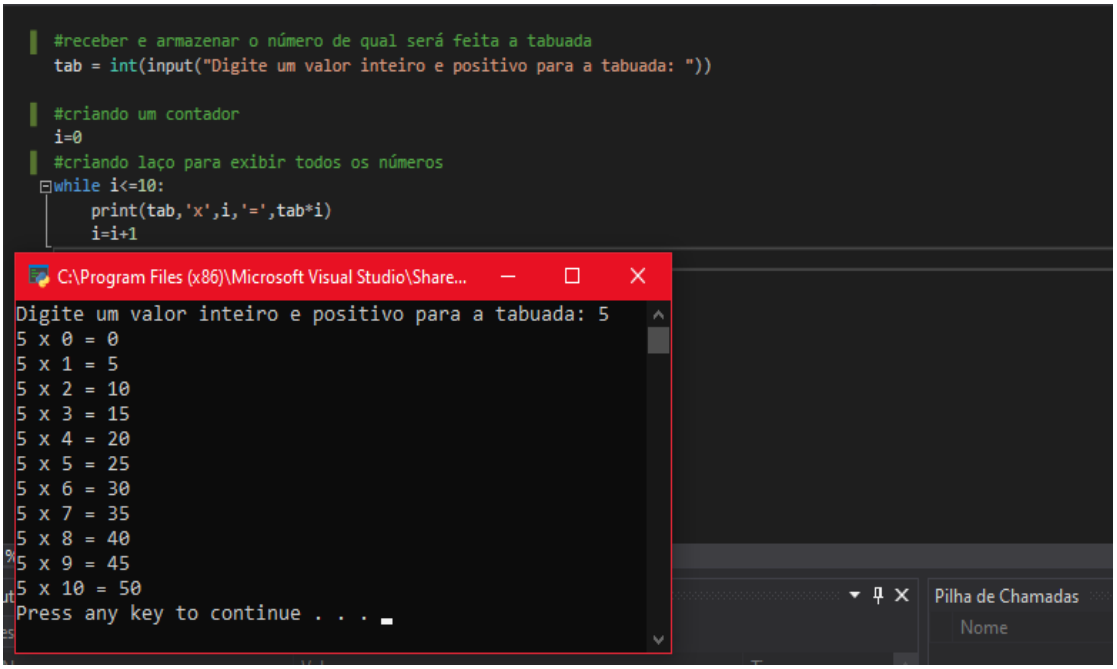
Ao final do for, podemos adicionar também a estrutura else. Isso faz com que um código ou bloco de códigos seja realizado ao final da repetição. Observe o exemplo abaixo:

```
nomes = ['Pedro', 'João', 'Leticia']
for n in nomes:
    print(n)
else:
    print("Todos os nomes foram listados com sucesso")
```

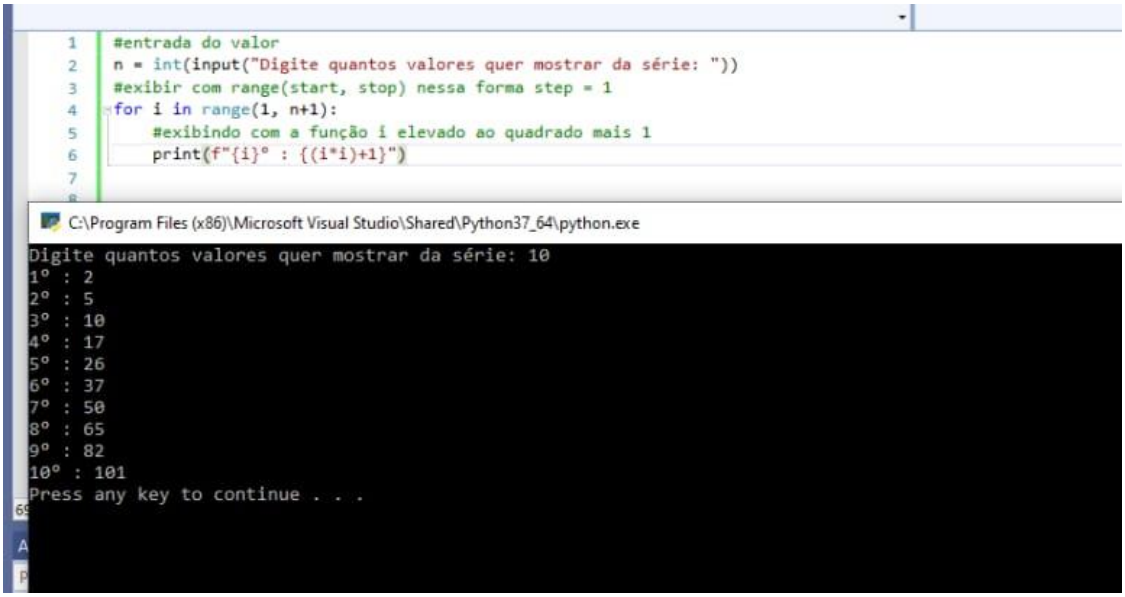
Na primeira linha, foi definida uma variável que armazenará uma lista de nomes, e após isso a estrutura for percorre todos esses elementos e atribui-os à variável n, que será exibida, e, após esse loop se encerrar, o código da instrução else será executado, imprimindo a mensagem na tela.

Exercícios de fixação: estrutura de repetição ‘For’

1. 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.



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



Laços de Repetição: Estrutura While

O comando while faz com que um bloco de instruções seja executado enquanto uma condição é atendida, e, quando o resultado dessa condição passa a ser falso, o loop é interrompido, como mostra o exemplo a seguir:

```
contador = 0
while (contador < 5):
    print(contador)
    contador   = contador + 1
```

Neste código, enquanto a variável contador, iniciada em 0, for menor que 5, o bloco de comandos será executado. Nota-se também, que na ultima linha, incrementa-se o valor da variável contador, de forma que após determinado tempo, o seu valor será igual ao número 5, e quando isso acontecer, o laço será interrompido.

Ao final do while, podemos também utilizar a instrução else. O propósito disso, é executar algum comando ou bloco de código após o loop terminar, como podemos ver no exemplo abaixo:

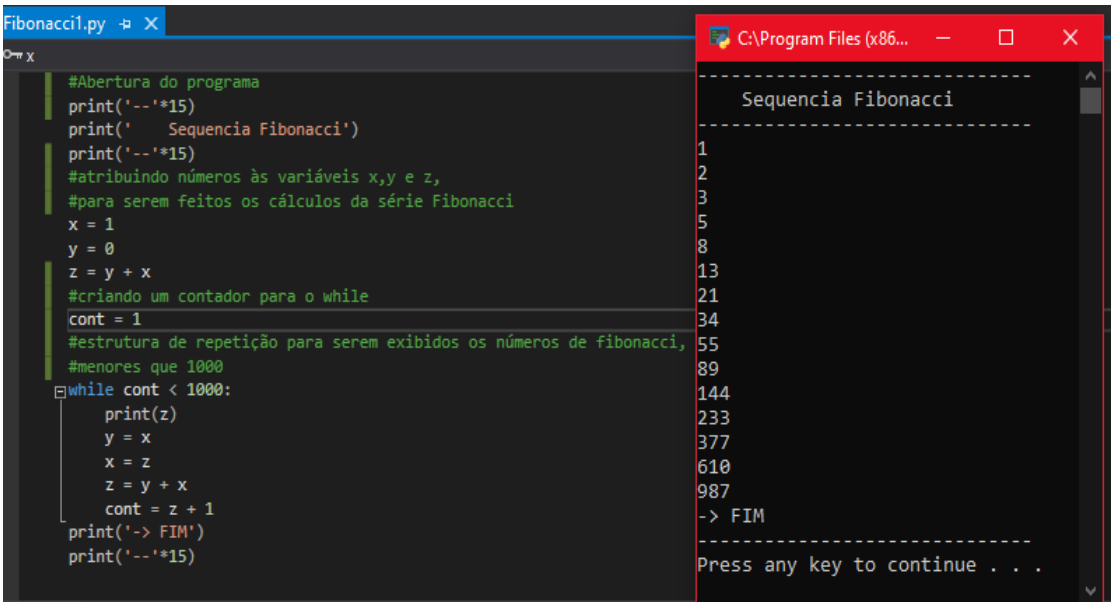
```
contador = 0
while (contador < 5):
    print(contador)
    contador = contador + 1
else:
    print("O loop while foi encerrado com sucesso!")
```

Dentro da repetição, podemos também utilizar um break, caso ele seja executado, o loop será encerrado sem executar o conjunto else.

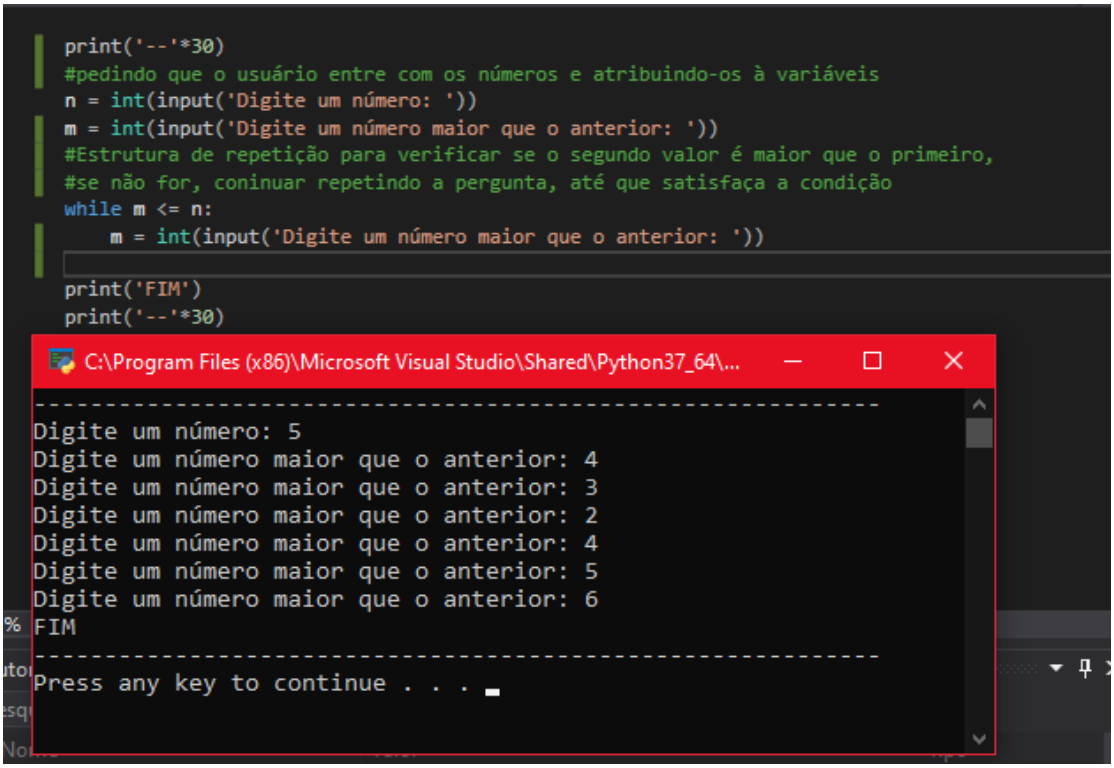
```
x = 0
while x < 10:
   print(x)
   x += 1
   if x == 6:
      print("x é igual a 6")
      break
else:
   print("fim while")
```

Exercícios de Fixação: estrutura de repetição ‘While’

3. 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.



4. 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.



5. 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.

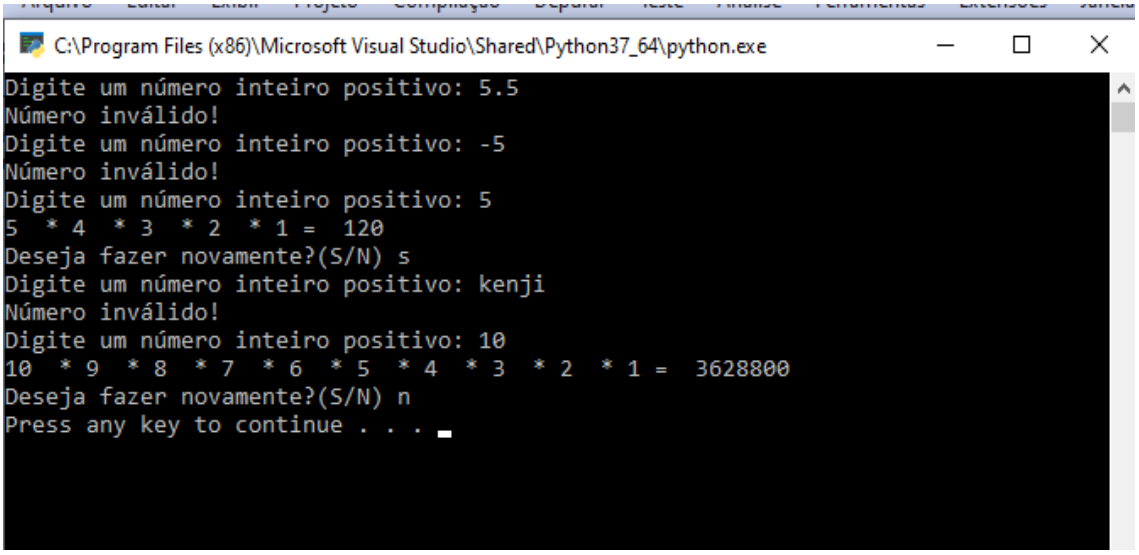
```
print('--'*15)
#pedindo que o usuário entre com o seu sexo
n = input('Digite seu sexo (M/F): ')
#enquanto n diferente de F e M repetir a pergunta
#fazendo verificação entre maiuscula e minuscula também
while n != 'F' and n!= 'M'and n != 'f' and n!= 'm':
    n = input('Digite seu sexo (M/F): ')
print('FIM')
print('--'*15)
```

```
C:\Program Files (x86)\Micro...
------------------------------
Digite seu sexo (M/F): a
Digite seu sexo (M/F): b
Digite seu sexo (M/F): c
Digite seu sexo (M/F): d
Digite seu sexo (M/F): e
Digite seu sexo (M/F): f
FIM
------------------------------
Press any key to continue . . .
```

6. 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.

fatoreal.py*

```
#while de continuar no prog
while 1==1:
    #while de validar o número no prog
    while 1==1:
        try:
            num = int(input("Digite um número inteiro positivo: "))
            if num <0:
                print("Número inválido!")
            else:
                break
        #caso não consiga converter pra int
        except:
            print("Número inválido!")
    #fazer o calculo
    fat = 1
    while num >0:
        fat *= num
        if num == 1:
            #exibir sem quebra de liha
            print(num, end="")
        else:
            print(num, " * ", end="")
        num = num -1
    #exibir o resultado
    print(" = ", fat)
    #perguntar se o usuário deseja continuar
    continuar = input("Deseja fazer novamente?(S/N) ").upper()
    #verificando resposta
    while continuar != "S" and continuar != "N":
        continuar = input("Resposta inválida!! \nDeseja fazer novamente?(S/N) ").upper()
    if continuar == "N":
        #sair do while do prog
        break
```


C:\Program Files (x86)\Microsoft Visual Studio\Shared\Python37_64\python.exe
Digite um número inteiro positivo: 5.5
Número inválido!
Digite um número inteiro positivo: -5
Número inválido!
Digite um número inteiro positivo: 5
5 * 4 * 3 * 2 * 1 = 120
Deseja fazer novamente?(S/N) s
Digite um número inteiro positivo: kenji
Número inválido!
Digite um número inteiro positivo: 10
10 * 9 * 8 * 7 * 6 * 5 * 4 * 3 * 2 * 1 = 3628800
Deseja fazer novamente?(S/N) n
Press any key to continue . . .

Variáveis Indexadas (Arrays)

Uma variável indexada é o conjunto de variáveis do mesmo tipo e nome, individualizadas por um índice. Podem ter um ou mais índices e recebe o nome de dimensão o número de índices necessários para a localização do elemento dentro da variável. Variáveis de uma dimensão são chamadas de vetores, já as variáveis de duas dimensões são chamadas de matrizes.

Listas de valores, como são chamadas em Python, são importantes estruturas de dados na linguagem Python, seus índices sempre começam por zero e terminam em n-1. Por exemplo: uma variável de tamanho 10, começará seu índice no número 0 e terminará no número 9. A diferença para com outras linguagens, é que as listas em Python podem conter elementos de tipos diferentes.

Exemplos:

```
#strings
cabelo = [“loiro”, “moreno”, “castanho”, “branco”, “calvo”]

#números
pares = [2, 4, 6, 8, 10]

#lista vazia
usuário = [ ]

#tipos diferentes
misturada = [“joão”, 23, -5, 2.5, [10,20]]

#lista com listas
nova_Lista = [pares, misturada]

#imprimir uma lista
print(cabelo)
print(pares)


#imprime o comprimento das listas
print(len(cabelo))
print(len(pares))
```

Para acessar os elementos das listas, utilizamos sua posição, seu índice dentro da lista. O índice do primeiro elemento é zero e qualquer expressão com resultado inteiro pode ser usada como índice. Índices negativos são considerados da direita para a esquerda. O valor do índice tem que estar entre 0 e len(lista) – 1, ou entre –len(lista) e -1.

Exemplos:

```
print("dois elementos de cabelo = ", cabelo[2], cabelo[-2])

print("ultimo elemento de pares = ", primos[len(pares) - 1])

for i in range(len(pares)):
    print("pares[", i, "] = ", pares[i])
```

Exercícios de Fixação: Variáveis Indexáveis

7. 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.

```
#criamos um vetor/array/lista em branco com 10 indices
numeros = [""] * 10

#pedimos ao usuário que digite os 10 valores
print('Digite 10 valores para serem exibidos inversamente')
#criamos uma estrutura de repetição para que cada dado seja armazenado...
#...em uma posição diferente
for i in range(10):
    numeros [i]= float(input())

#usando while para inverter a lista
#define-se um contador do tamanho do vetor inicial menos 1, para indicar...
#a posição da ultima casa do vetor, tamanho= 10, o primeiro indice sendo 0 e o ultimo 9
i = len(numeros) - 1
#criando um novo vetor para armazenar os valores invertidos
inverso = []

while (i>=0):
    #incrementando o valor da lista numero na lista inversa
    inverso.append(numeros[i])
    i = i - 1
#exibindo a lista inversa
print(inverso)
```

Utilizando estrutura while

```
#criamos um vetor/array/lista em branco com 10 indices
numeros = [""] * 10

#pedimos ao usuário que digite os 10 valores
print('Digite 10 valores para serem exibidos inversamente')
#criamos uma estrutura de repetição para que cada dado seja armazenado...
#...em uma posição diferente
for i in range(10):
    numeros [i]= float(input())

#python nos dá a função de inverter a lista sem precisar de um laço de repetição
#utilizando a função list(reversed(vetor))
inverso = list(reversed(numeros))
print(inverso)
```

Há também a opção de se utilizar a função list(reversed(lista)), que diminui a linha de código, além de ser mais prática a sua utilização

```
C:\Program Files (x86)\Microsoft Visual Studio\S...
Digite 10 valores para serem exibidos inversamente
1
2
3
4
5
6
7
8
9
10
[10.0, 9.0, 8.0, 7.0, 6.0, 5.0, 4.0, 3.0, 2.0, 1.0]
Press any key to continue . . .
```

8. 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.

```python
#criamos um vetor/array/lista em branco com 10 indices
numeros = [""] * 10

#pedimos ao usuário que digite os 10 valores
print('Digite 10 valores para serem somados')
#criamos uma estrutura de repetição para que cada dado seja armazenado...
#...em uma posição diferente
for i in range(10):
    numeros [i]= float(input())

# definimos uma variável soma para que os valores sejam armazenados
soma = 0
#criamos um laço para que sejam somados todas as posições do vetor
for i in numeros:
    #o sinal de += soma a variável com o indice e armazena o valor novo no mesmo local
    soma += i
#exibindo o resultado
print(soma)
```

Utilizando o laço de repetição For

```python
#criamos um vetor/array/lista em branco com 10 indices
numeros = [""] * 10

#pedimos ao usuário que digite os 10 valores
print('Digite 10 valores para serem somados')
#criamos uma estrutura de repetição para que cada dado seja armazenado...
#...em uma posição diferente
for i in range(10):
    numeros [i]= float(input())

# definimos uma variável soma para que os valores sejam armazenados
#python nos da a função soma, para que possa somar todas as posições da lista
soma = sum(numeros)
print(soma)
```

Python também nos dá a função de somar todos os elementos de uma lista com a função sum(), que deixa o código menor e mais prático.

```
C:\Program Files (x86)\Microsoft Visual Studio\Shared\Python37_64\pytho...
Digite 10 valores para serem somados
4
3
2
5
3
53
23
4
3
2
102.0
Press any key to continue . . .
```

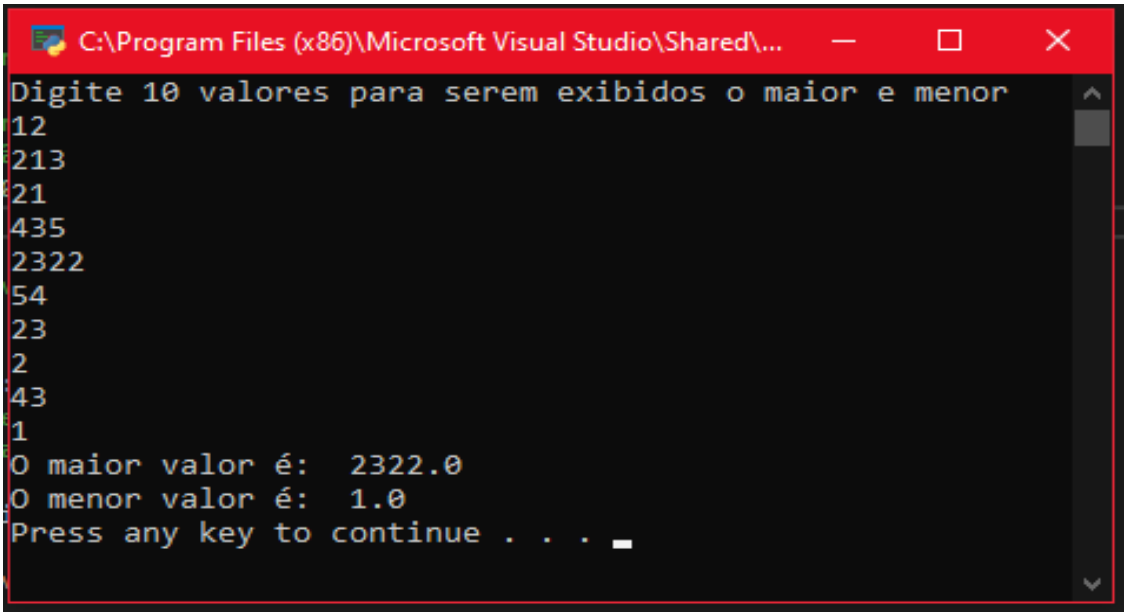
9. O programa deverá nos permitir digitar e armazenar dez números na memória do computador. Feito isso criar um laço capaz de identificar o maior e o menor dos números digitados e exibi-los ao final.

```
#criamos um vetor/array/lista em branco com 10 indices
numeros = [""] * 10

#pedimos ao usuário que digite os 10 valores
print('Digite 10 valores para serem exibidos o maior e menor')
#criamos uma estrutura de repetição para que cada dado seja armazenado...
#...em uma posição diferente
for i in range(10):
    numeros [i]= float(input())

#Atribuindo uma variável sem valor definido para armazenar os numeros
maior = None
#Fazendo um laço de repetição para procurar o maior numero
for i in numeros:
    #se a variável for fazia ou se o elemento for maior que ela
    #será feita a substituição de valor
    if (maior == None or i > maior):
        maior = i

print ('O maior valor é: ', maior)
#a mesma coisa será feita para a variável menor
menor = None
for i in numeros:
    #se a variável for fazia ou se o elemento for menor que ela
    #será feita a substituição de valor
    if (menor == None or i < maior):
        menor = i

print ('O menor valor é: ', menor)
```

Utilizando for para fazer a comparação dos valores da lista.

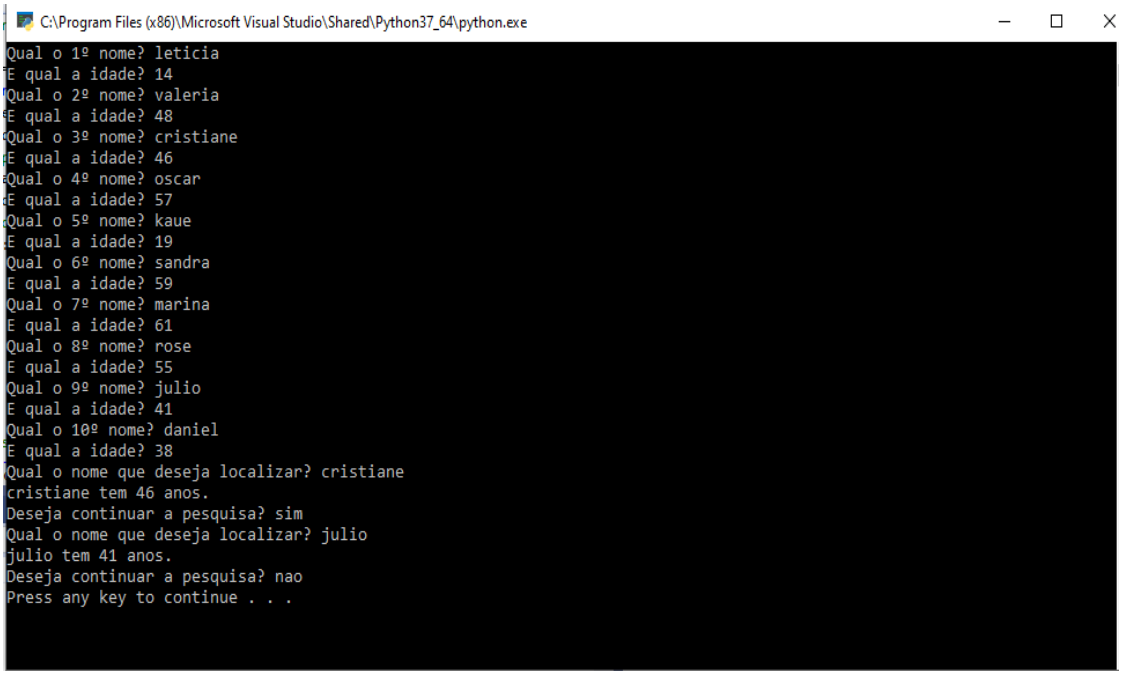
```
#criamos um vetor/array/lista em branco com 10 indices
numeros = [""] * 10

#pedimos ao usuário que digite os 10 valores
print('Digite 10 valores para serem exibidos inversamente')
#criamos uma estrutura de repetição para que cada dado seja armazenado...
#...em uma posição diferente
for i in range(10):
    numeros [i]= float(input())

#Python nos dá a função de buscar o maior e menor valor de uma lista

menor = min(numeros)
maior = max(numeros)

print('o menor é ', menor)
print('o maior é ', maior)
```

Python também nos dá a função de pesquisar quais são os valores maior e menor de uma lista.



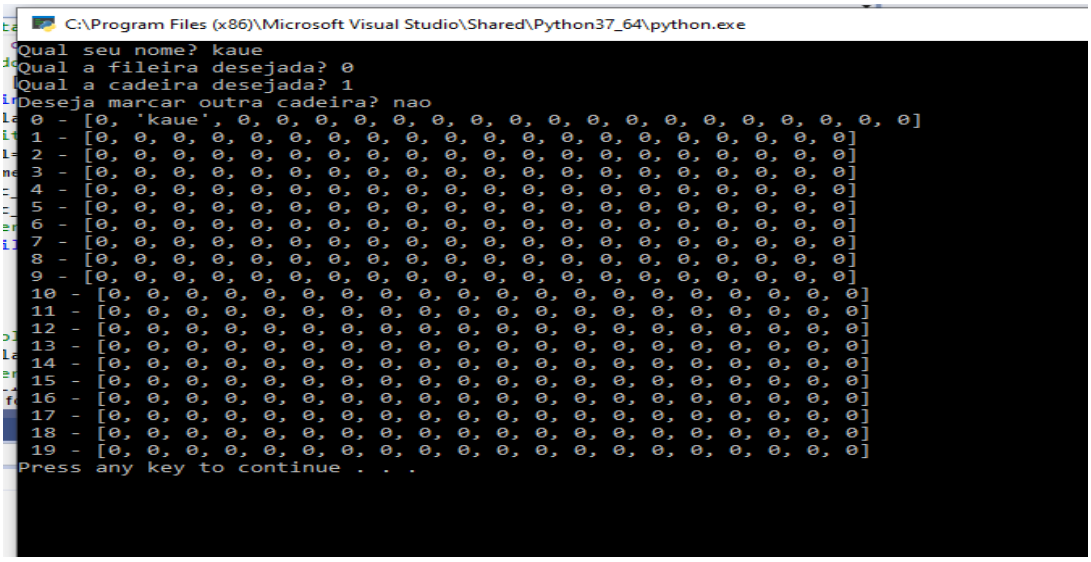
10. O programa deverá nos permitir digitar e armazenar o nome e idade de dez pessoas. Feito isso deverá nos solicitar a digitação de um nome e então proceder a pesquisa e informar a idade do sujeito pesquisado caso ele se encontre armazenado, caso contrário informar o fato através de mensagem “Pessoa 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.

```python
#inserindo os nomes e idades
nomes = []
idades = []
for i in range(10):
    nomes.append(input(f"Qual o {i+1}º nome? ") )
    idades.append(input("E qual a idade? "))
#while principal para usuário decidir se quer continuar as pesquisas
continuar = True
while continuar:
    #bloco try, se der erro em alguma parte desse bloco vai executar o except
    try:
        #pesquisa da idade
        num_index = nomes.index(input("Qual o nome que deseja localizar? "))
        print(f"{nomes[num_index]} tem {idades[num_index]} anos.")
        cont = input("Deseja continuar a pesquisa? ")
        #comparando a 1ª letra em maiúsculo se é diferente de 'S'
        if cont[0].upper() != "S":
            #para o while principal
            break
    #caso der erro na pesquisa do nome(não encontrar o nome)
    except:
        #while para localizar um novo nome ou sair do prog
        while 1==1:
            cont1= input("Pessoa não localizado. \nDeseja pesquisar novamente? ")
            if cont1[0].upper() == "S":
                #parar esse while e continua no while principa
                break
            else:
                continuar = False
                print("Xau :)")
                break
```



11. 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.

```
#importando biblioteca os
import os;
#criando matriz
sala = []
for i in range(20):
    sala.append([0]* 20)
#solicitando informações
while 1==1:
    nome = input("Qual seu nome? ")
    esc_fileira = int(input("Qual a fileira desejada? "))
    esc_cadeira = int(input("Qual a cadeira desejada? "))
    #verificando cadeira
    while sala[esc_fileira][esc_cadeira] != 0:
        os.system('cls')
        print("Cadeira já escolhida, digite novamente outra opção: ")
        esc_fileira = int(input("Qual a fileira desejada? "))
        esc_cadeira = int(input("Qual a cadeira desejada? "))
    #colocando nome na cadeira
    sala[esc_fileira][esc_cadeira] = nome
    #verificando se deseja marcar outra cadeira
    cont= input("Deseja marcar outra cadeira? ")
    if cont.upper() in ("SIM", "S"):
        #limpando a tela
        os.system('cls')
    else:
        break
#mostrando sala
for i in range(20):
    print(f" {i} -", sala[i])
```


C:\Program Files (x86)\Microsoft Visual Studio\Shared\Python37_64\python.exe
Qual seu nome? kaue
Qual a fileira desejada? 0
Qual a cadeira desejada? 1
Deseja marcar outra cadeira? nao
0 - [0, 'kaue', 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
1 - [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
2 - [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
3 - [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
4 - [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
5 - [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
6 - [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
7 - [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
8 - [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
9 - [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
10 - [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
11 - [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
12 - [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
13 - [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
14 - [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
15 - [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
16 - [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
17 - [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
18 - [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
19 - [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
Press any key to continue . . .

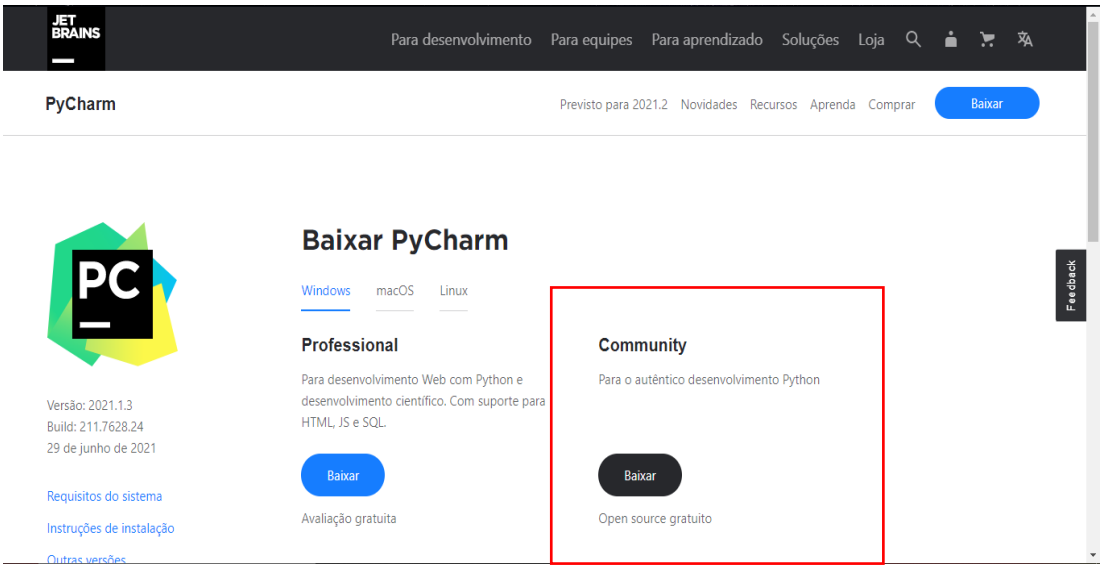
Interface Gráfica

A Interface gráfica do usuário (abreviadamente, GUI do inglês *Graphical User Interface)* é um tipo de interface que permite a integração com dispositivos digitais por meio de elementos gráficos e que o usuário interaja com o programa de forma mais acessível, trabalhando com símbolos, ícones e outros indicadores visuais. Essa interação acontece muitas vezes por meio de eventos do mouse e do teclado, cujo o usuário pode manipular os elementos a fim de obter algum resultado.

Utilizando Interface Gráfica no Python

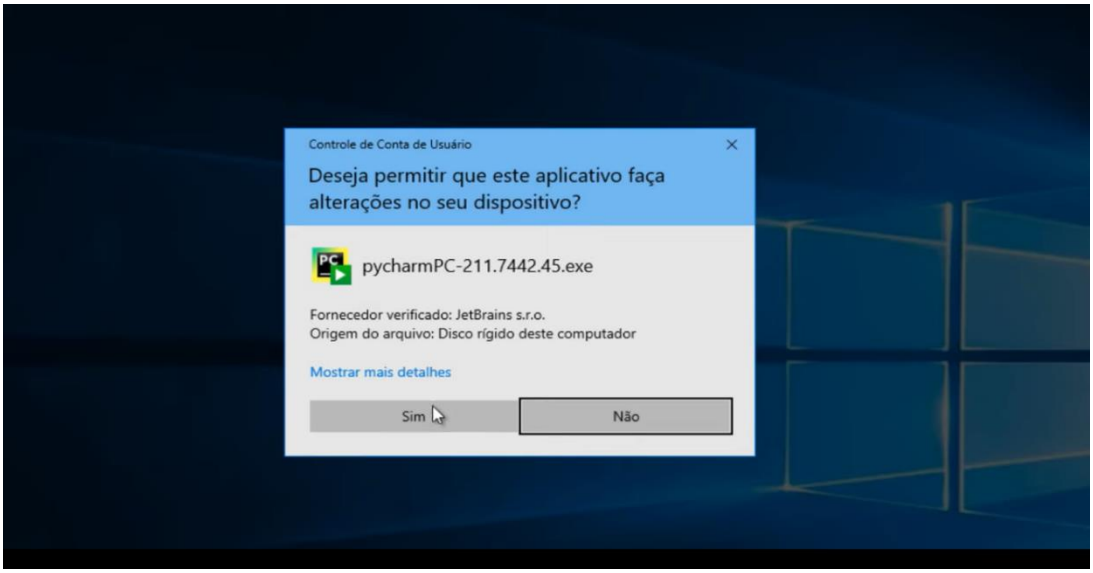
Instalando o PyCharm

Para esta etapa, instalaremos a IDE PyCharm, que irá auxiliar-nos a fazer a conversão dos elementos da interface gráfica para o nosso código.

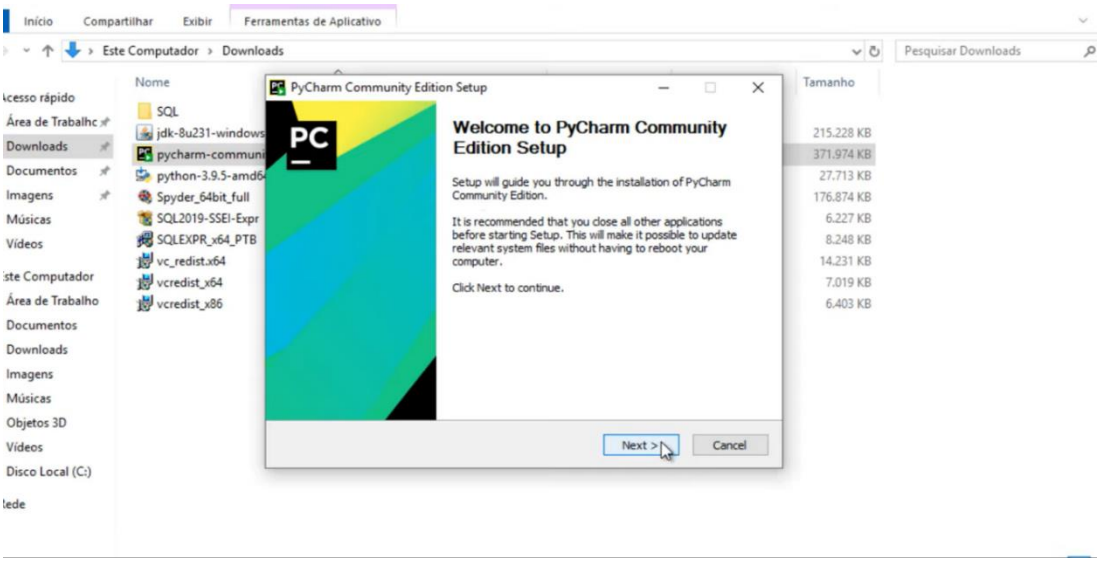


Ao procurar por PyCharm e clicar no primeiro link no google, chegaremos à essa página, onde iremos instalar a versão Community.

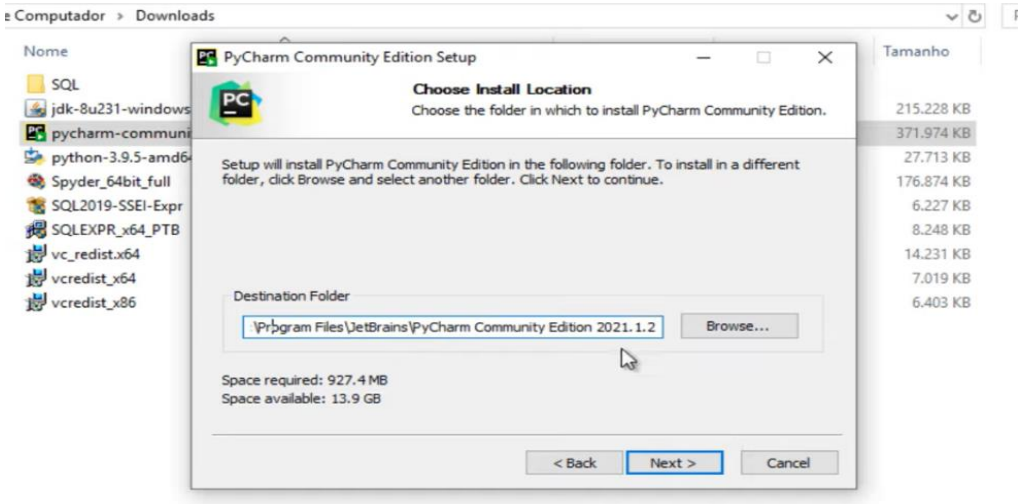
Após instalar, clique no arquivo executável para começarmos a instalação



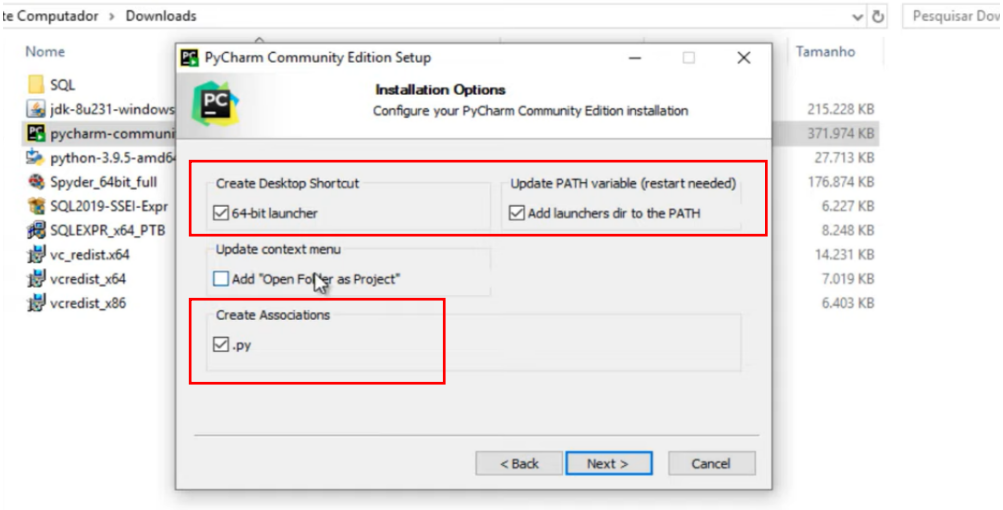
Permita que o arquivo faça alterações no dispositivo



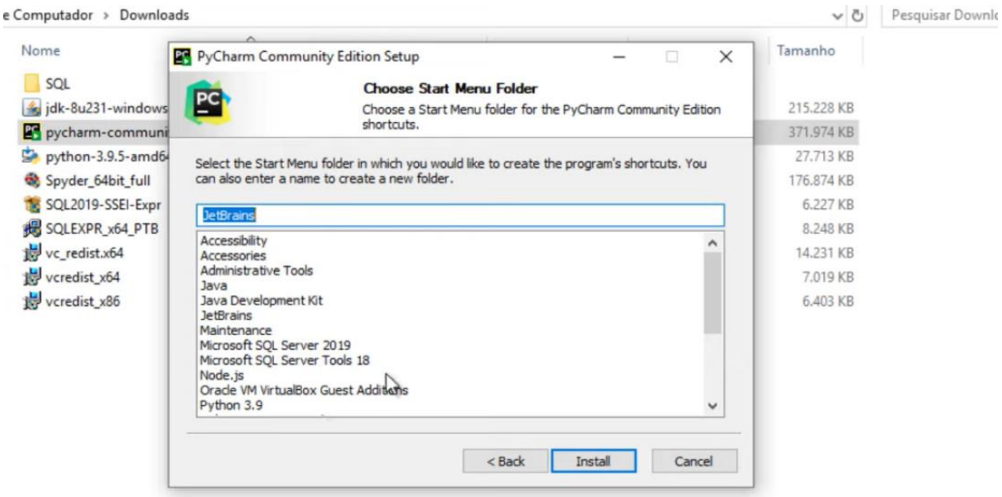
A tela do instalador será aberta, clique em *next* para começar a instalação.



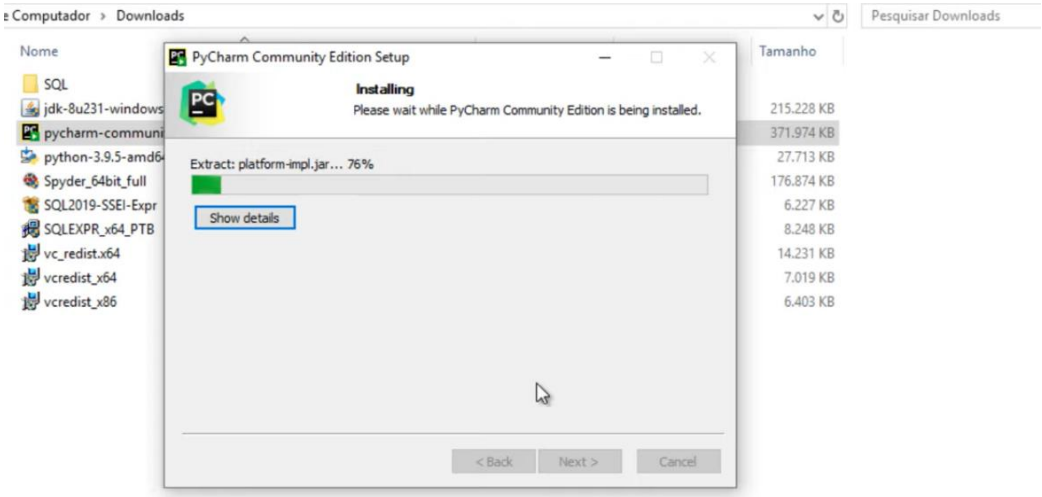
Clique em Next novamente.



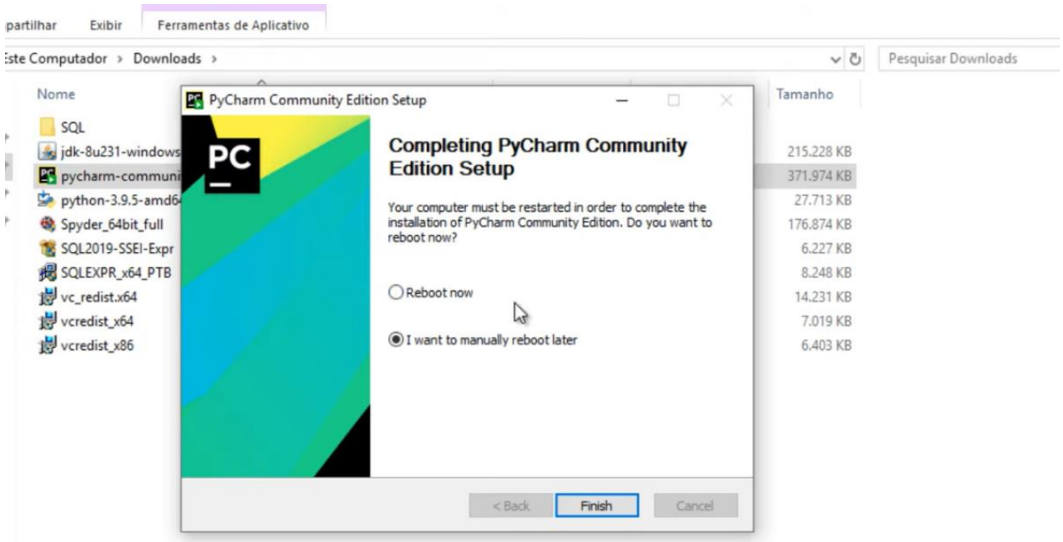
Assinale as alternativas destacadas e clique em Next novamente.



A tela irá mostrar onde tudo será instalado, clique em next novamente.



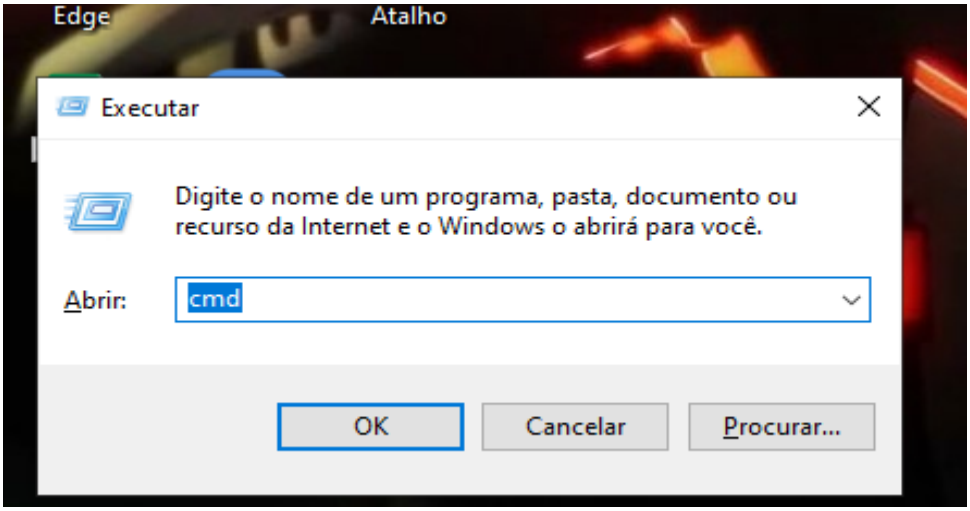
Aguarde todos os componentes instalarem devidamente, e clique em next.



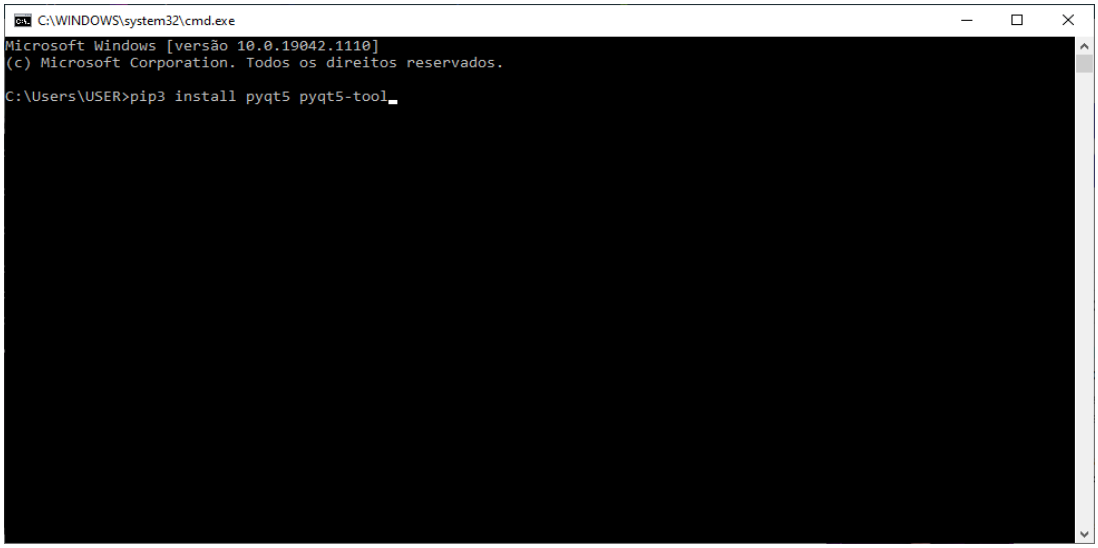
Após finalizar, reinicie o computador. Pronto, instalamos o PyCharm, porém não abriremos agora, ele será usado futuramente.

Instalando o PyQt5

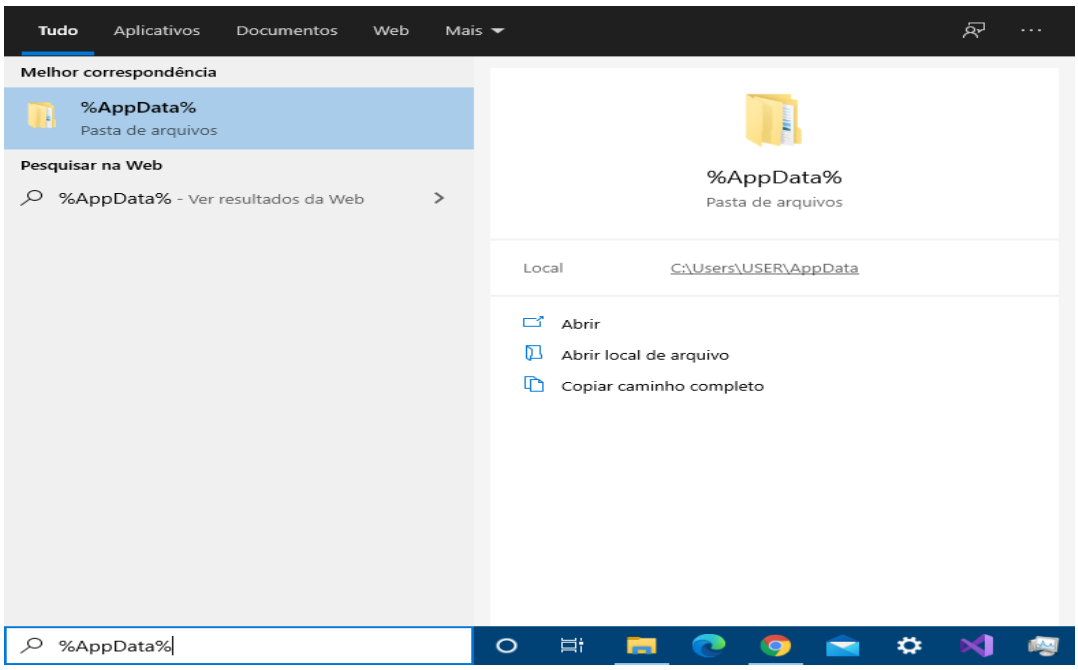
PyQt5 é um dos módulos mais utilizados para construir aplicativos com interface gráfica em Python pois é bem simples de ser utilizado. Outro método que incentiva os desenvolvedores a usar o PyQt5 é o seu designer, que torna muito fácil desenvolver aplicativos complexos em pouco tempo.



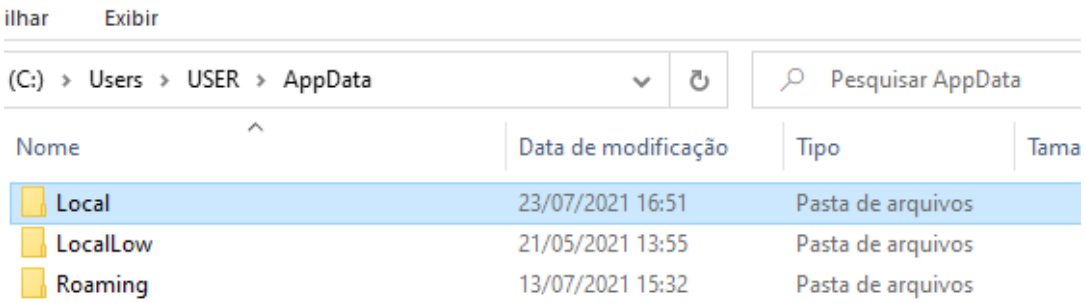
Primeiramente, clique nas teclas Windows + R para abrir o menu Executar e digite cmd para abrir o prompt de comando.



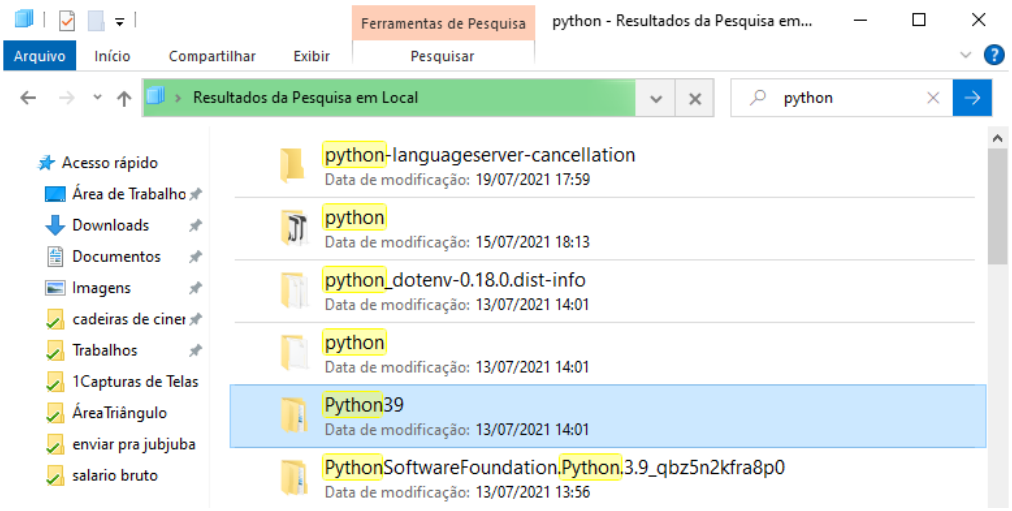
Após o prompt de comando ser aberto, iremos instalar o pyqt5 de acordo com a linha de comando logo após o C:\ Users\User> e dê enter.



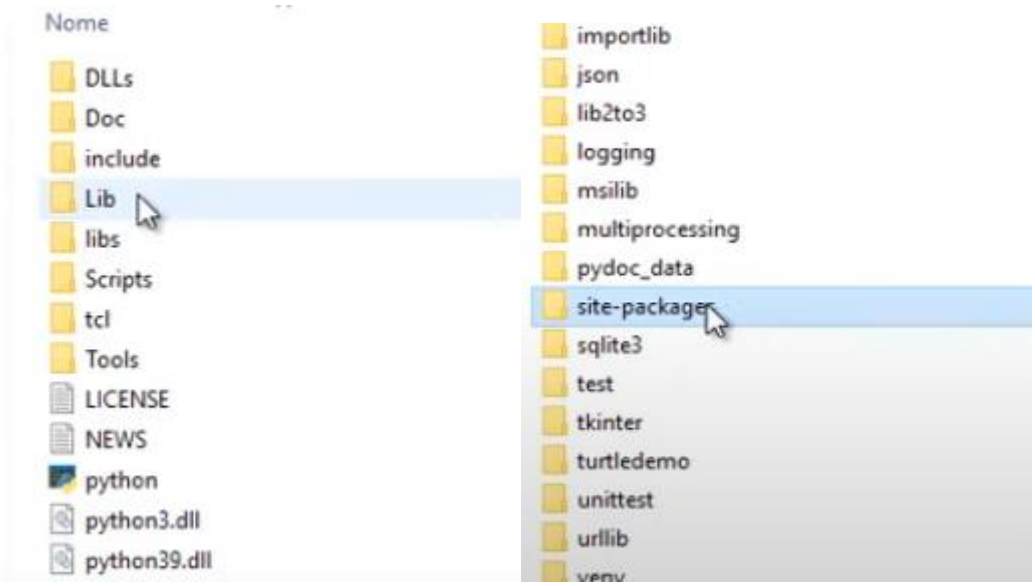
Logo após, feche o Prompt de comando e pesquise a pasta %AppData% em sua barra de pesquisa do Windows e selecione para abrir.



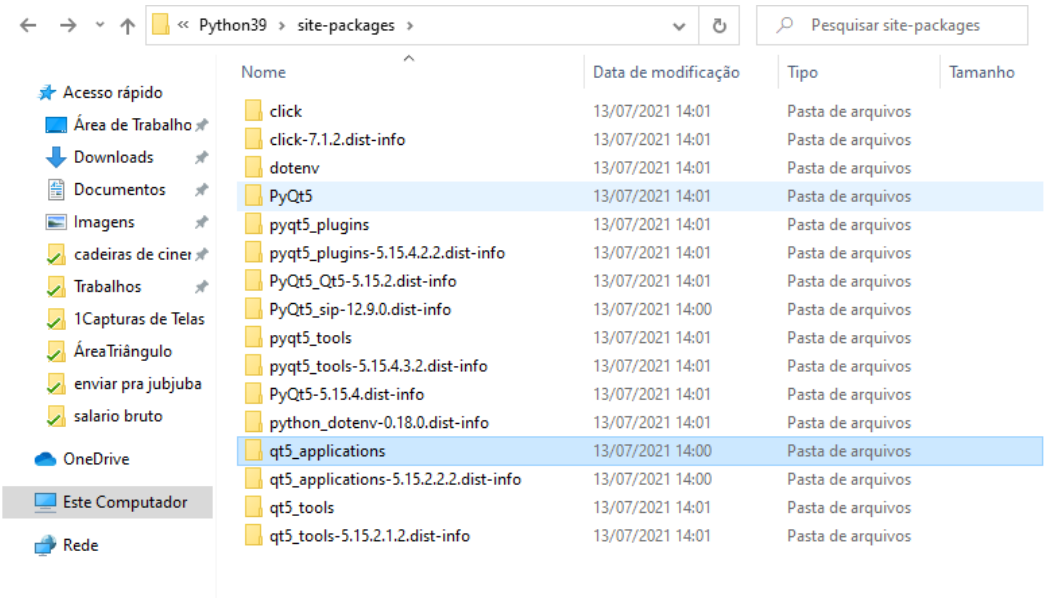
Procure e entre na pasta Local, dentro da pasta AppData.



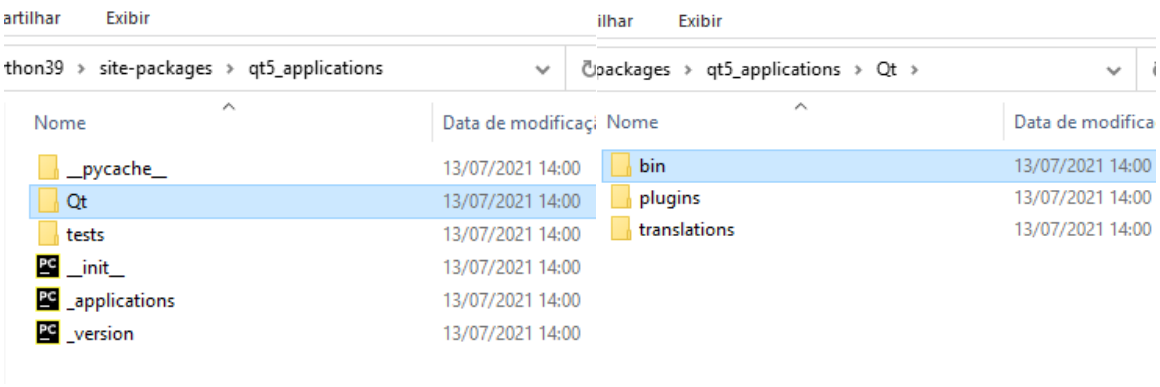
Dentro da pasta Local, procure pela pasta Python39 e abra-a.



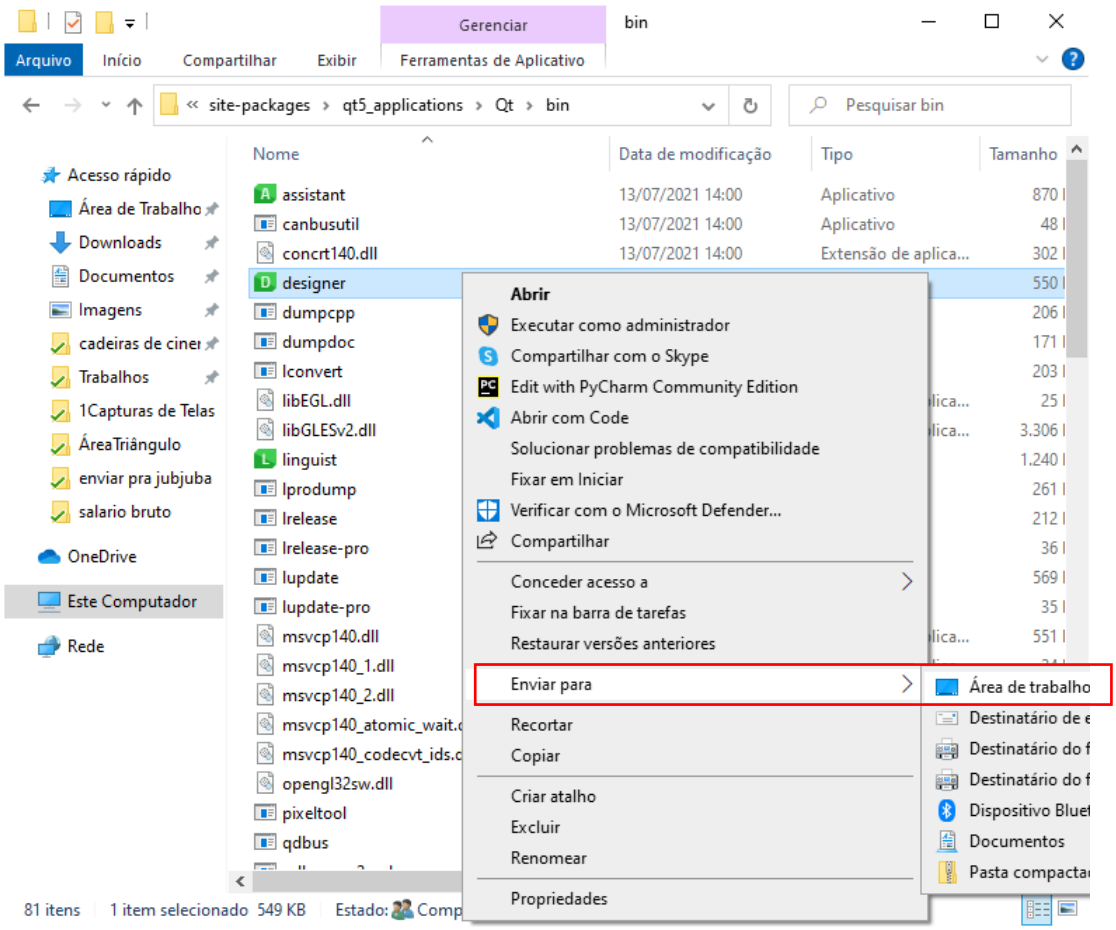
Após isso, selecione a pasta Lib e, logo após, a pasta site-package.



Dentro da pasta site-packages, abra a pasta qt5_applications.

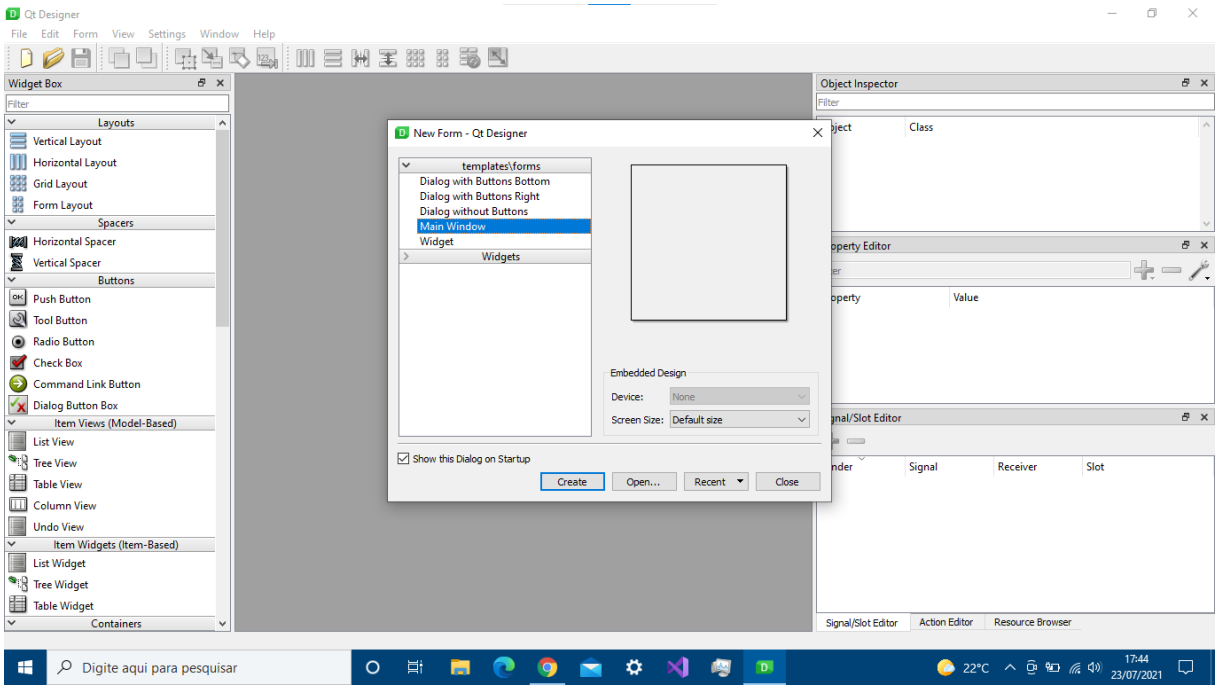


Abra a pasta Qt e logo após a pasta bin.

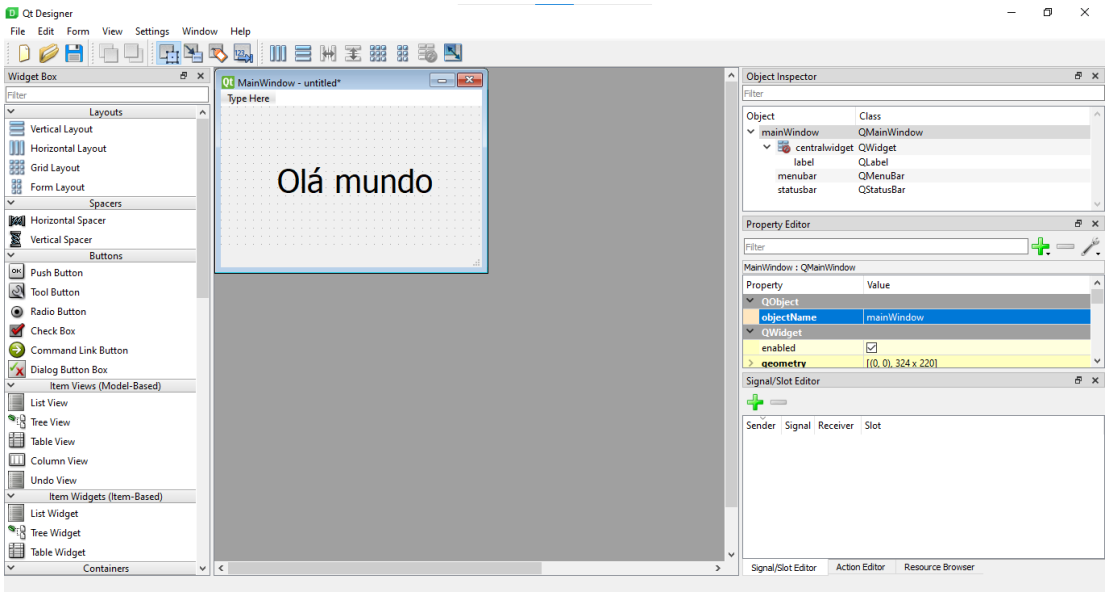


Dentro da pasta bin, localize o aplicativo designer, clique com o botão direito do mouse e envie-o para a área de trabalho.

Logo após, o aplicativo será adicionado na sua área de trabalho e não precisaremos mais acessar toda a estrutura de pastas, então feche o explorador de arquivos e dê um duplo clique no aplicativo para abri-lo.

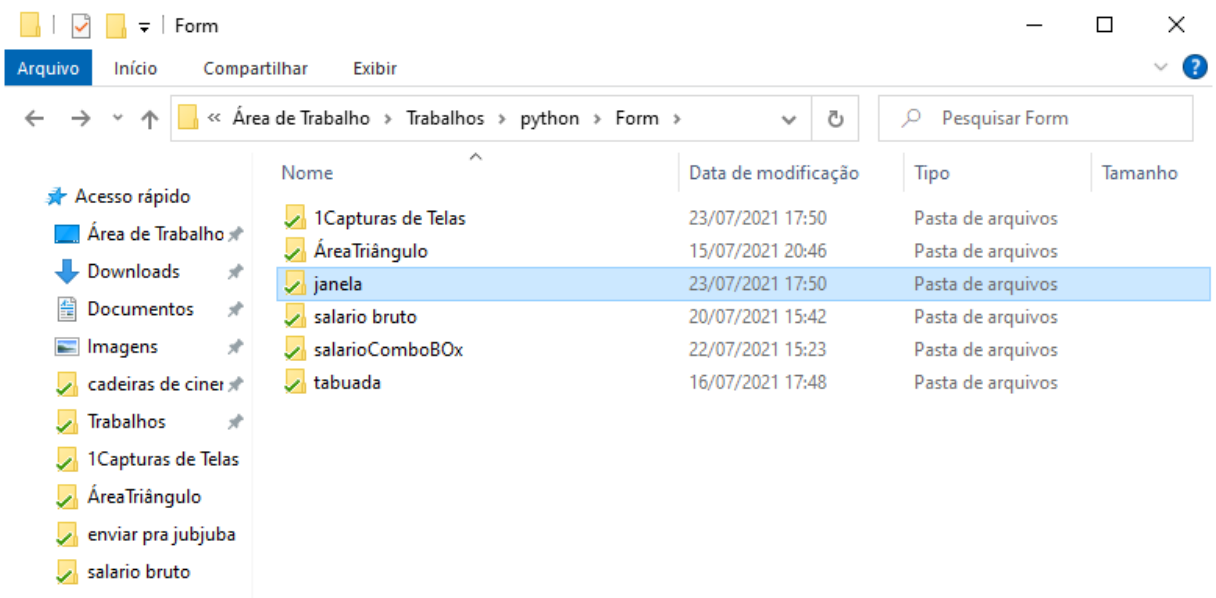


Crie um novo arquivo e selecione-o como Main Window, para a janela criada ser a principal no nosso projeto.

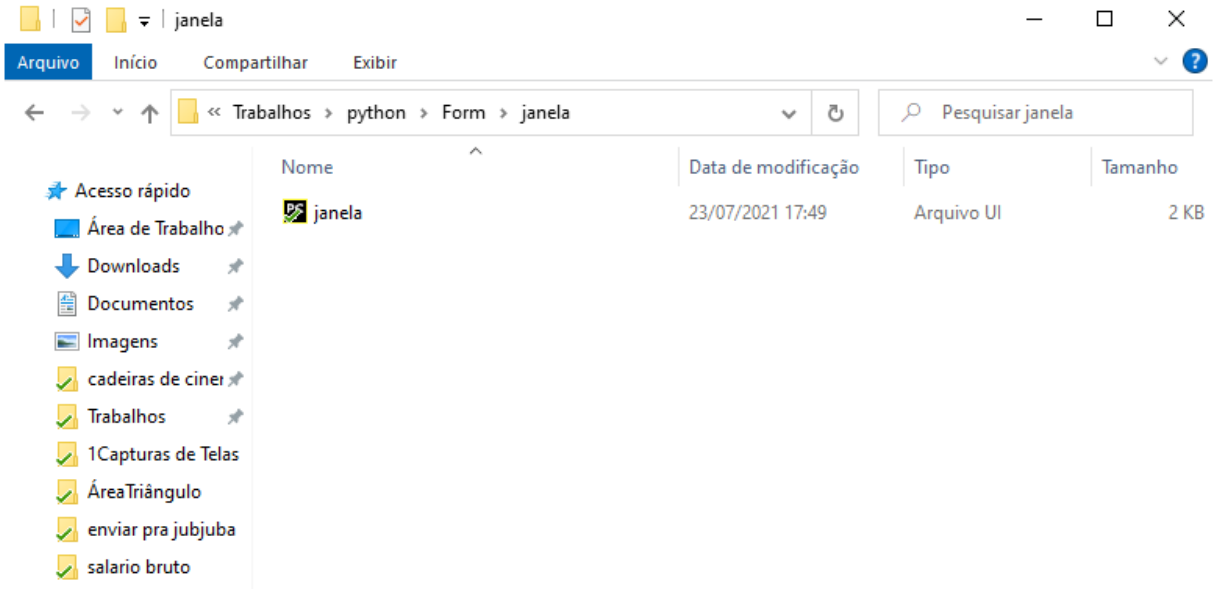


Segue um exemplo de “olá mundo” que foi feito utilizando somente um Label. Clique e arraste os objetos para o Form para fazer o que deseja.

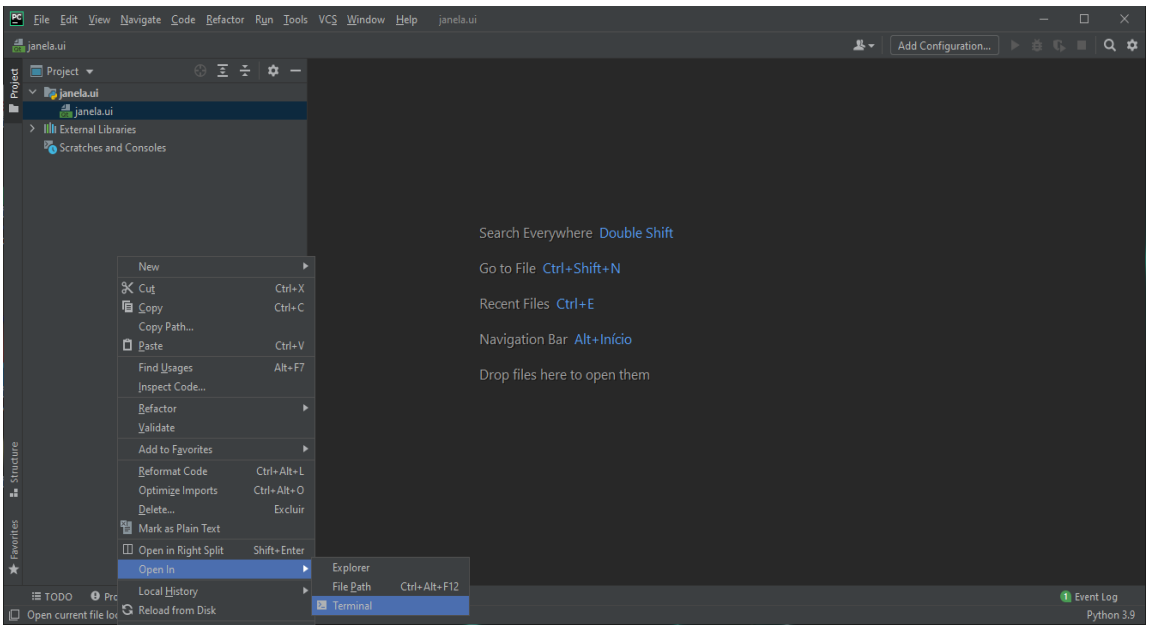
Agora, vamos aprender como utilizar esse design como código Python para que as funções correspondentes sejam programadas.



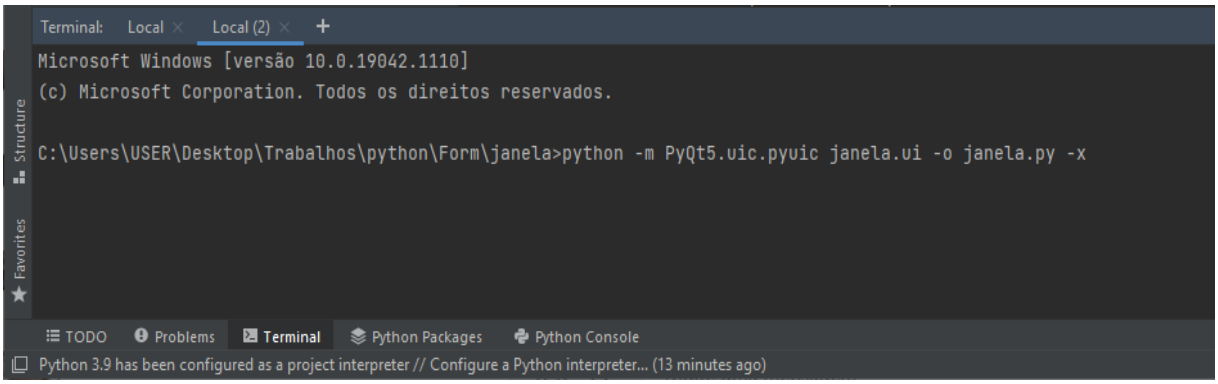
Salve seu design em uma pasta.



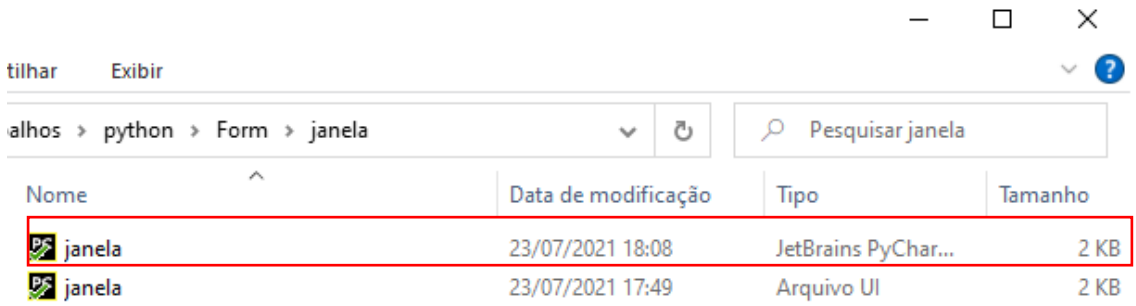
Entre na pasta que você salvou o design, clique com o botão direito e abra-a com o PyCharm.



Quando abrir o PyCharm, clique com o botão direito sob o arquivo, e selecione para abrir no terminal.



No terminal, digite a linha de comando após ...janela>, isso fará com que o design feito no PyQt seja convertido para código Python.



Abrindo a pasta novamente, nota-se que agora teremos um arquivo Python além do design já feito.

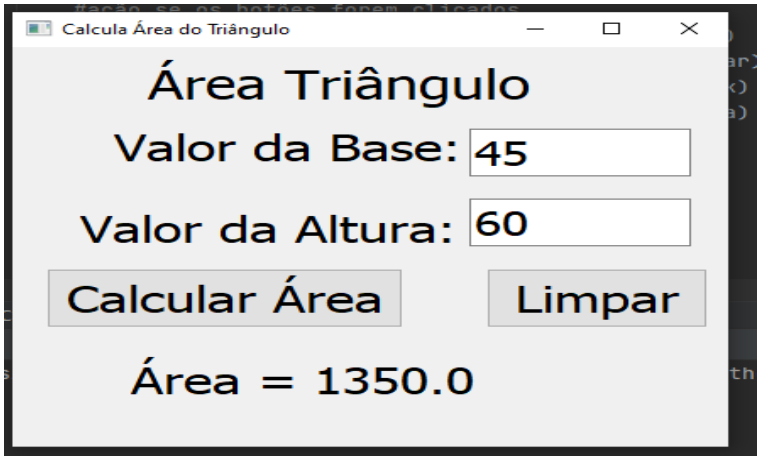
Esse processo serve para que o programador não precise codificar todo o formulário "na raça", pois Python não possui mais a funcionalidade em IDE's que permita que o design do formulário e o código sejam feitos no mesmo ambiente, e, observando isso, foi encontrado este modo, de utilizarmos um designer para montar o visual do formulário e convertê-lo, como se tivesse sido todo codificado desde o início.

Exercícios de fixação Interface gráfica

1)

```
68          #ação se os botões forem clicados
69          self.limpar.clicked.connect(self.valorBase.clear)
70          self.limpar.clicked.connect(self.valorAltura.clear)
71          self.calcular.clicked.connect(self.calcular_click)
72          QtCore.QMetaObject.connectSlotsByName(CalculaArea)
73      #ação do botão
74      def calcular_click(self):
75            b = float(self.valorBase.text())
76            h = float(self.valorAltura.text())
77            area =(b * h) / 2
78            result = "Área = " + str(area)
79            #editar texto da label
80            self.Resultado.setText(result)
```

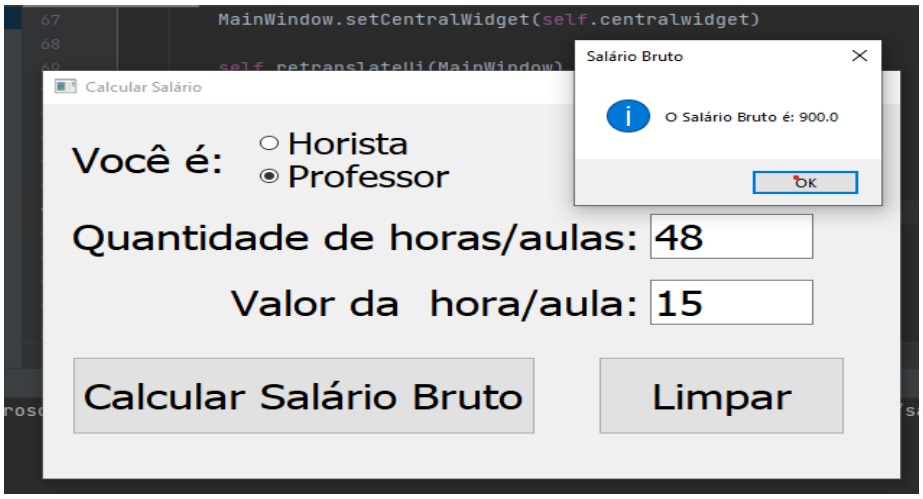
Neste primeiro exemplo, fizemos um formulário que calcula a área de um triângulo utilizando-se da base e da altura que o usuário digitará por meio de uma textBox. A primeira parte do código se refere à ação que o botão limpar irá realizar ao ser clicado: ele limpará as textBox's que foram nomeadas de valorBase e valorAltura. Após isso, tem-se o evento do botão calcular, que quando clicado, buscará o valor da Área, que foi definido logo abaixo, onde converte-se o valor textual para float e atribui-os às variáveis b e h e calcula-se com a fórmula matemática. Logo depois, é armazenado em uma variável um texto junto com o resultado da fórmula para ser exibido. Executando o código, então, ficará como a imagem abaixo:



Os objetos utilizados neste formulário foram: Labels (utilizados para colocar texto no formulário), textBox (as caixas brancas, onde o usuário irá digitar os respectivos valores) e os buttons (botões que, quando clicados, farão todo o processo de buscar no código o que ele deverá fazer com cada dado).

2)

```
70          # ação se os botões forem clicados
71          self.limpar.clicked.connect(self.qtdhr.clear)
72          self.limpar.clicked.connect(self.vlhr.clear)
73          self.botaocalc.clicked.connect(self.calcular)
74          QtCore.QMetaObject.connectSlotsByName(MainWindow)
75
76      #ação do botão
77      def calcular(self):
78          qtd = float(self.qtdhr.text())
79          vlr = float(self.vlhr.text())
80          sala = qtd * vlr
81         #verificar se o radiobutton do professor está selecionado
82          if (self.radioprof.isChecked()):
83              sala = sala * 1.25
84          #messagebox(título,mensagem)
85          messagebox.showinfo(title="Salário Bruto", message=f"O Salário Bruto é: {sala}")
```

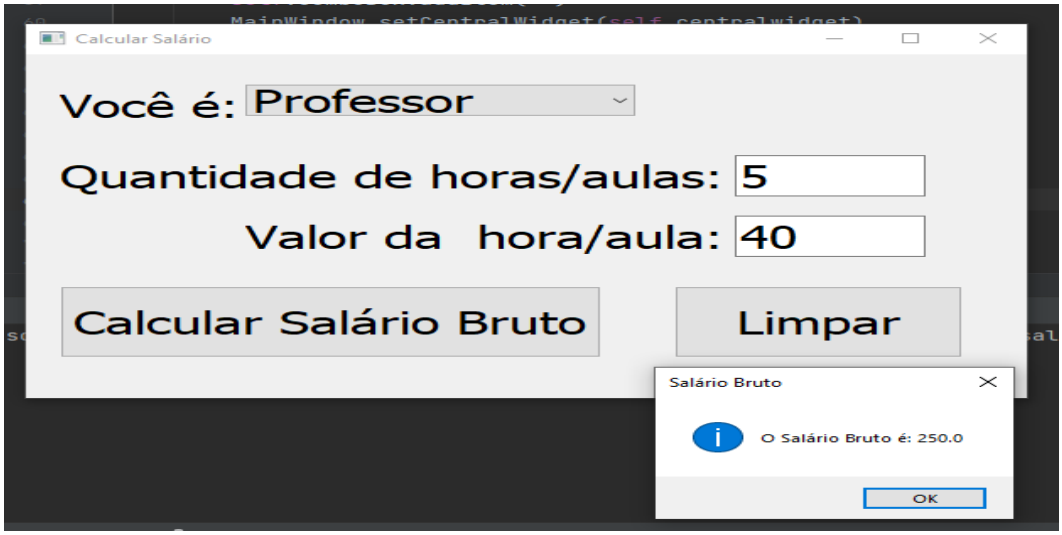


Neste programa, é calculado o valor do salário bruto de um horista ou professor, de acordo com o valor da hora ou da aula por quantas horas ele trabalhou ou deu aula. Este programa

é muito semelhante com o anterior, com somente algumas mudanças na formula utilizada e que agora temos que selecionar uma opção ou outra de acordo com um radioButton (campo onde você seleciona somente uma opção, muito utilizada em formulários que perguntam sobre o sexo de uma pessoa). Também foi utilizada uma messageBox, ao invés de um Label, para exibir o resultado em uma outra janela, método muito utilizado em mensagens de erro em alguns programas.

3)

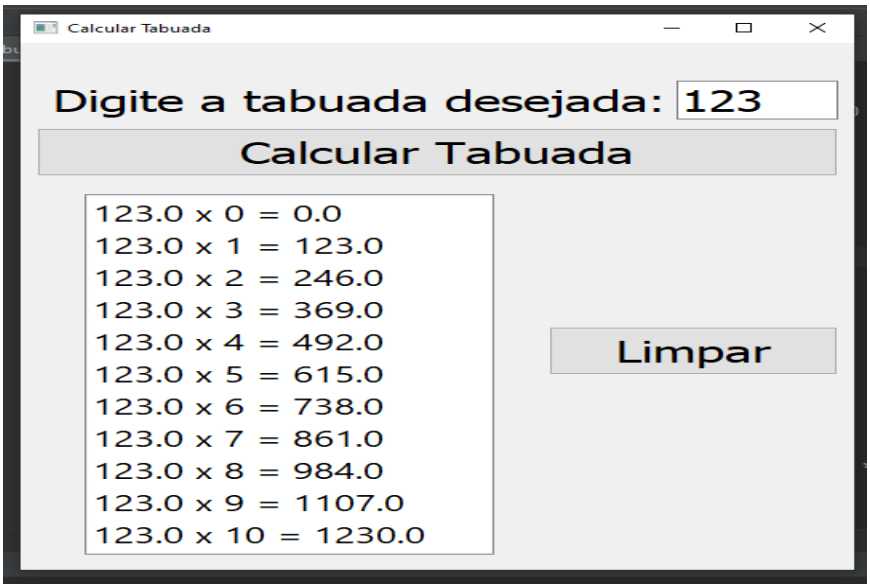
```
63          # ação se os botões forem clicados
64          self.limpar.clicked.connect(self.qtdhr.clear)
65          self.limpar.clicked.connect(self.vlhr.clear)
66          self.botaocalc.clicked.connect(self.calcular)
67          QtCore.QMetaObject.connectSlotsByName(MainWindow)
68
69          # ação do botão
70
71      def calcular(self):
72          qtd = float(self.qtdhr.text())
73          vlr = float(self.vlhr.text())
74          sala = qtd * vlr
75          # verificar se o combobox do professor está selecionado
76          if (self.comboBox.currentText() == "Professor"):
77              sala = sala * 1.25
78          # messagebox(título,mensagem)
79          messagebox.showinfo(title="Salário Bruto", message=f"O Salário Bruto é: {sala}")
```



Neste exercício, utilizamos a mesma função do exercício anterior, porém, ao invés de utilizarmos um radioButton, para selecionarmos dentre as opções de Horista e Professor, utilizamos uma comboBox, muito utilizada em formulários onde é necessário selecionar o país ou o idioma desejado.

4)

```
45          #ação quando os botões são clicados
46          self.limpar.clicked.connect(self.numero.clear)
47          self.limpar.clicked.connect(self.lista.clear)
48          self.Calcular.clicked.connect(self.tabuada_lista)
49          QtCore.QMetaObject.connectSlotsByName(CalcularTabuada)
50      #função da tabuada
51      def tabuada_lista(self):
52          #pegar número
53          num = float(self.numero.text())
54          #repetição para adicionar os itens na lista
55          for item in range(0,11, +1):
56              #adicionando itens na lista (linha, texto)
57              self.lista.insertItem(item, f"{num} x {item} = {num * item}")
58
```



Nesse projeto, utilizamos uma listBox para exibir os itens de uma lista, que seria a tabuada do número que o usuário digitou. Para fazer essa exibição, no programa foi utilizado o método de exibir em lista com o método de repetição for...in...range (inicio, fim), método esse utilizado em JavaScript.

Arquitetura de programação em Camadas

A arquitetura em camadas é um sistema cliente-servidor no qual a camada de apresentação, a camada de processamento de aplicativos e a camada de gerenciamento de dados são separadas. A arquitetura em camadas pode ser definida como o processo de decompor um sistema complexo em camadas, de modo a compreendê-lo e facilitar a manutenção desse sistema. Mostra também que essa tecnologia é emprestada da arquitetura de computadores, não que usam camadas de chamada ao sistema operacional, drivers e etc.

Programação Orientada a Objetos (POO)

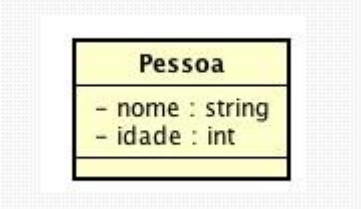
A Programação Orientada a Objetos (POO) é um padrão de desenvolvimento de softwares largamente utilizado em muitas linguagens de programação atuais, como Java, C#, PHP, Python, C++, entre outras.

Nesse processo de programação, são criadas coleções de objetos com estrutura e comportamentos próprios. Tais objetos interagem entre si e executam as ações solicitadas. Portanto, o objetivo da POO é aproximar o mundo real do mundo virtual e promover, também, a unificação de dados e processos, o agrupamento e a reutilização de códigos.

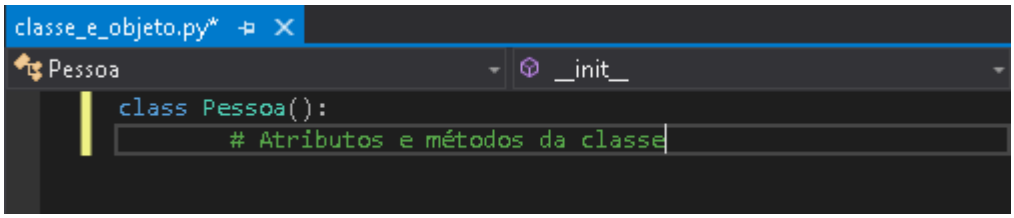
Classes

Para criar e declarar uma classe na linguagem Python é bem simples, com a criação dessas classes é possível definir os métodos e atributos que ela irá possuir. É através da definição da classe que teremos as propriedades e atributos que o objeto irá possuir as classes proporcionam uma maneira de organizar dados e funcionalidades juntos. Ao criar e declarar uma nova classe é criado um novo tipo de objeto, que permite que novas instâncias desse tipo de objeto sejam feitas. Cada instância da classe pode ter atributos anexados a ela, para manter seu estado.

Vamos ver um exemplo de como se declara uma classe na linguagem Python, seguindo a UML abaixo:



O nome da Classe é “Pessoa” e nome e idade são seus atributos. Para criar essa classe primeiro nós usamos a palavra *class*, seguida pelo nome da classe e dois pontos.

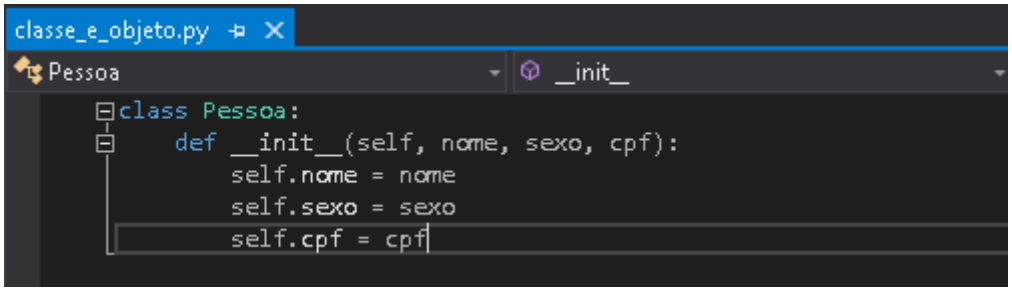


No Python, todas as classes devem possuir nomes que comece com a primeira letra maiúscula.

Métodos e Atributos

Agora vamos criar o construtor da classe, que basicamente serve para inicializar, eles são “chamados” quando os objetos dessa classe são criados:

Para declarar um atributo usamos o __init__, definindo como uma nova pessoa será criada. Para definir atributos basta passá-los como parâmetro, como acima.



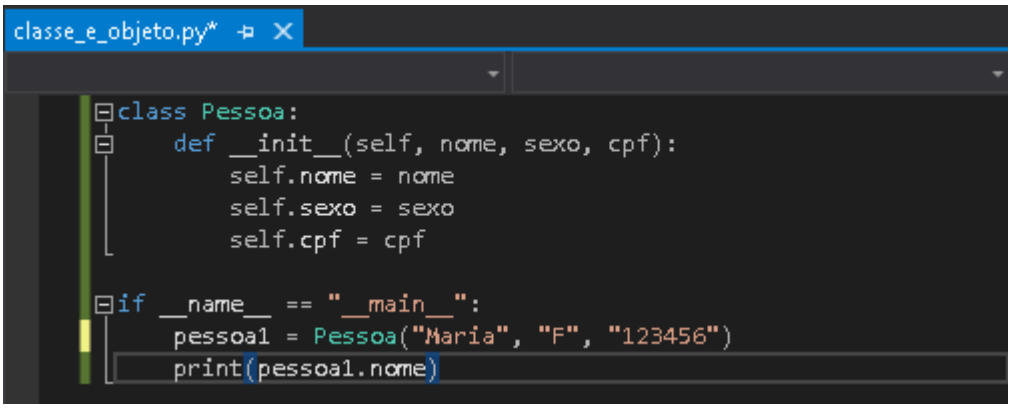
É válido ressaltar que o parâmetro *self* é obrigatório em todos os métodos.

Após o construtor é hora de criar os métodos get e set de todos os atributos:

```
def setNome(self, nome):
        self.nome = nome

def setIdade(self, idade):
        self.idade = idade

def getNome(self):
        return self.nome

def getIdade(self):
        return self.idade
```

As classes representam um elemento do mundo real, no entanto, elas são apenas o modelo desses elementos. Dizemos que estamos instanciando um objeto quando precisamos criar algo baseado na classe, instanciar um objeto significa que estamos criando a representação de uma classe em nosso programa.

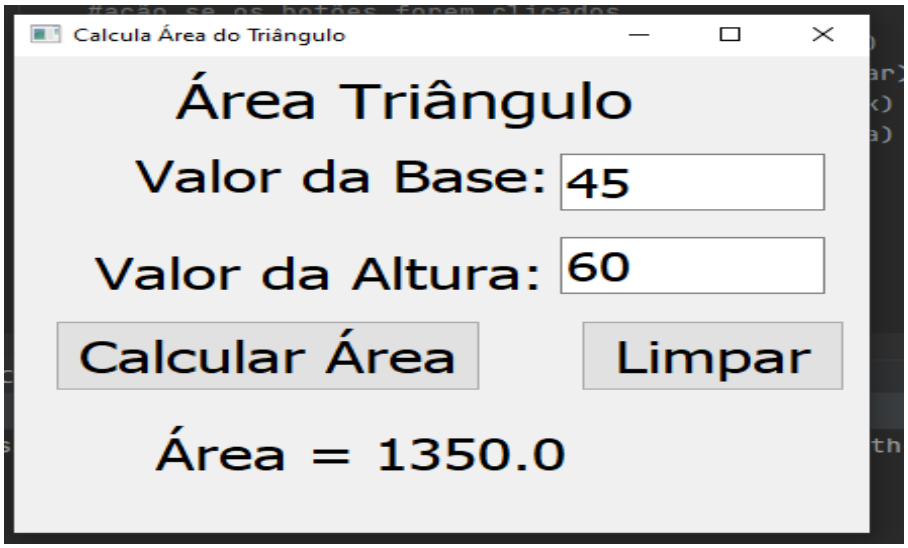
Na linguagem Phyton, para instanciar um objeto com base em uma classe já declarada, é só indicar a classe que desejamos utilizar como base e informar os valores referentes aos seus atributos:



Ao executar a linha pessoa1 = Pessoa(“Maria”, “F”, “123456”) estamos criando um objeto do tipo pessoa com nome “Maria”, sexo “F” e CPF “123456”.

Exercícios de Fixação POO

1) Área do Triângulo



Podemos definir a parte gráfica como a IHM (UIL), pois será ela que irá permitir a interação do usuário com o programa descrito.

janela.py | classe1.py

classe1.py > CalcularArea > __init__

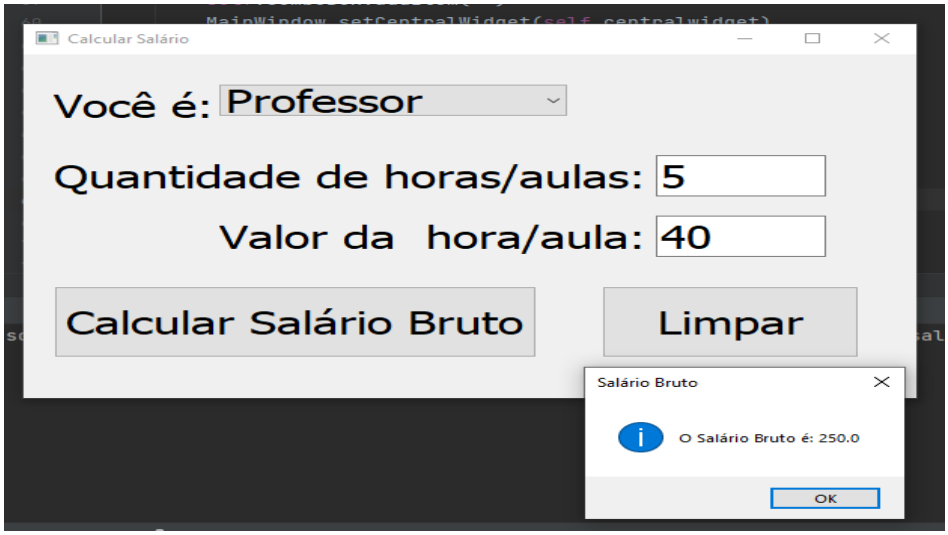
```python
#nome da classe
class CalcularArea:
    #início da classe com (valor da base, valor da altura
    def __init__(self, base, altura):
        #recebendo valores colocado nos parênteses
        self.b = base
        self.h = altura
        #calculando a área
        self.area = self.b * self.h/2
        #finalizar
        pass
```

```python
#importando a classe
from classe1 import CalcularArea
from PyQt5 import QtCore, QtGui, QtWidgets
```

```python
#ação do botão
def calcular_click(self):
    b = float(self.valorBase.text() )
    h = float(self.valorAltura.text() )

    #criando a classe
    calc = CalcularArea(b,h)
    #colocando calc.area que é a área do triângulo
    result = "Área = " + str(calc.area)
    #editar texto da label
    self.Resultado.setText(result)
```

Podemos definir a programação do botão e das classes como a BLL, pois será ela que irá permitir que toda a função do programa seja feita corretamente. Nesse caso, a BLL irá armazenar os valores da base e altura do triângulo para a área do mesmo ser calculada.

2) Salário Horista.



A IHM (UIL) deste programa, é baseada na utilização de uma combobox para ser selecionado a opção Horista ou Professor, para assim serem calculados o valor da hora/aula de cada um.

salarioCombo.py | classe.py

classe.py > CalcularSalario

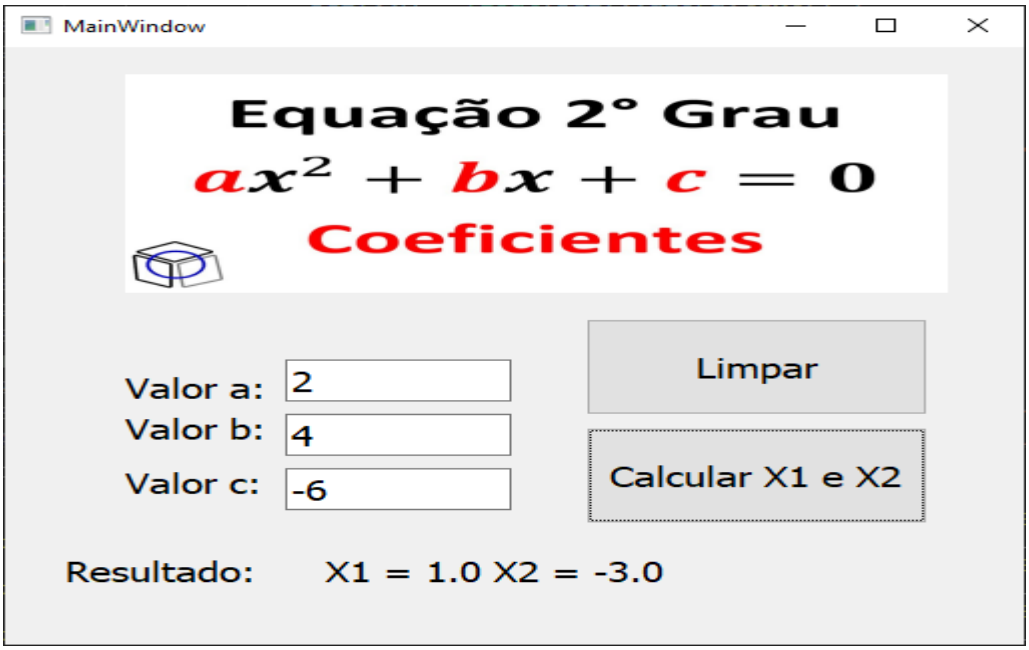
```python
#nome da classe
class CalcularSalario:
    #início da classe com valores
    def __init__(self, quanthora, valorhora):
        self.qtd = quanthora
        self.vl = valorhora
        pass

        #método horista
    def SalarioHori(self):
        self.sal = self.qtd * self.vl
        return self.sal
        #método professor
    def SalarioProf(self):
        self.sal = self.qtd * self.vl *1.25
        return self.sal
```

```python
#importar a classe
from classe import CalcularSalario
from PyQt5 import QtCore, QtGui, QtWidgets
#importando messagebox
from tkinter import messagebox
```

```
71
72      def calcular(self):
73          qtd = float(self.qtdhr.text())
74          vlr = float(self.vlhr.text())
75          sala = qtd * vlr
76          #criando o objeto
77          salario = CalcularSalario(qtd, vlr)
78          salario_calculado = salario.SalarioHori()
79         #verificar se o radiobutton do professor está selecionado
80         # verificar se o combobox do professor está selecionado
81          if (self.comboBox.currentText() == "Professor"):
82              sala = sala * 1.25
83          #messagebox(título,mensagem)
84          messagebox.showinfo(title="Salário Bruto", message=f"O Salário Bruto é: { salario_calculado}")
```

Nesse caso, a BLL irá verificar qual das opções está selecionada, para assim o valor da hora do horista, ou do professor, ser calculado. Assim retornando com o valor do salário bruto do mesmo que está selecionado.

3) Equação Segundo Grau.



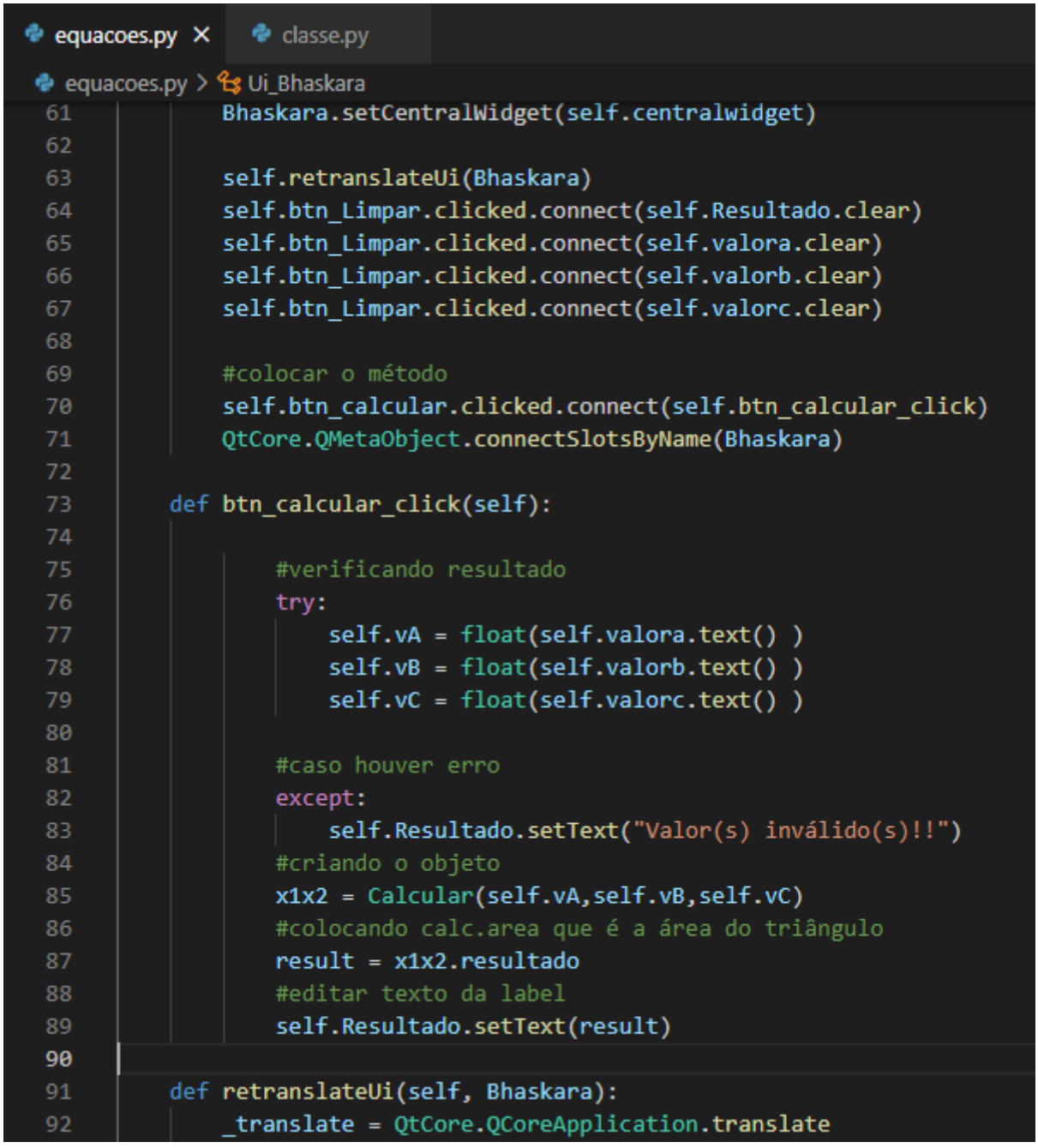
A IHM (UIL) deste programa nos traz três valores, os coeficientes, que deverão ser informados para assim, a equação de segundo grau ser formada, com a opção de limpar os campos e/ou calcular.

equacoes.py | classe.py

classe.py > Calcular > __init__

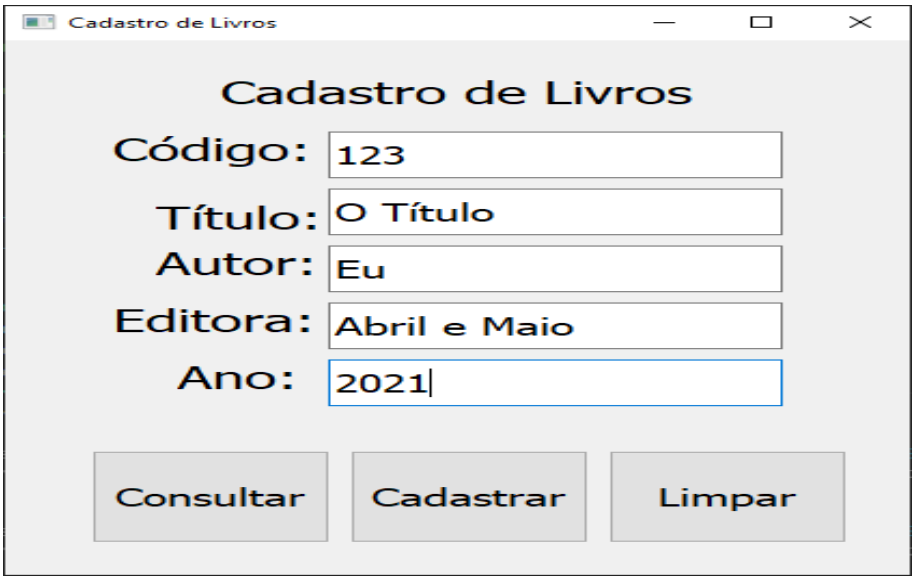
```python
#importar classe math para fazer raiz quadrada
import math
#nome da classe
class Calcular:
    #início da classe com (valor de A, valor de B, valor de C)
    def __init__(self, valorA, valorB, valorC):
        #recebendo valores colocado nos parênteses
        self.a = valorA
        self.b = valorB
        self.c = valorC
        self.delta = self.b ** 2 -4*self.a*self.c

        if self.delta < 0:
            self.resultado = "Não possui raiz!!"
        else:
        #calculando X1 eX2
            self.x1 = (-(self.b) + math.sqrt(self.delta) ) / (2* self.a)
            self.x2 = (-(self.b) - math.sqrt(self.delta) ) / (2* self.a)
            #arredondar
            self.x1 = round( self.x1 , 2)
            self.x2 = round( self.x2 , 2)
            self.resultado = f"X1 = {self.x1} X2 = {self.x2}"
        #finalizar
        pass
```

```python

#importar classes
from classe import Calcular
from PyQt5 import QtCore, QtGui, QtWidgets

```



Nesse caso, a BLL irá verificar os valores dos coeficientes, para que os valores da equação sejam calculados corretamente utilizando Bhaskara.

4) Cadastro Livro (sem banco de dados).



A IHM (UIL) deste programa nos traz alguns campos que deverão ser preenchidos para o cadastro ou consulta de um livro (que por enquanto será fake, por conta de não estarmos utilizando banco de dados).

Classe.py × CadastrarLivro.py

Classe.py > Consultar > __init__

```python
#lista
Livros = []
cod = []

#nome da classe
class Cadastrar:
    #início da classe com (codigo, titulo, autor, editora,ano)
    def __init__(self, codigo, titulo, autor, editora,ano):
        cod.append(codigo)
        Livros.append([codigo, titulo, autor, editora,ano])

        print(Livros)
        pass

class Consultar:

        #início da classe com (codigo)
    def __init__(self, codigo):
        index =0
        #comparando codigo digitado com os codigos digitados na lista
        for c in cod:
                if c == codigo :
                    self.titulo = Livros[index][1]
                    self.autor = Livros[index][2]
                    self.editora = Livros[index][3]
                    self.ano = Livros[index][4]
                index += 1
        pass

```

```
9
10    #importar classe
11    from Classe import Cadastrar, Consultar
12    from PyQt5 import QtCore, QtGui, QtWidgets
```

```
97          #funções dos botões
98          self.btn_Cadastrar.clicked.connect(self.Cadastrar_click)
99          self.btn_Consultar.clicked.connect(self.Consulta_click)
100         QtCore.QMetaObject.connectSlotsByName(CadastroLivros)
101
102     #métodos
103     def Cadastrar_click(self):
104         #Passando conteúdo dos campos
105         cod = self.Codigo.text()
106         titu = self.Titulo.text()
107         anos = self.Ano.text()
108         aut = self.Autor.text()
109         edit =  self.Editora.text()
110         #Cadastrar livros
111         livro = Cadastrar(cod, titu, aut, edit,anos)
112         #Limpar campos
113         self.Codigo.clear()
114         self.Titulo.clear()
115         self.Ano.clear()
116         self.Autor.clear()
117         self.Editora.clear()
118
119     def Consulta_click(self):
120          #Passando conteúdo do campo codigo
121         cod = self.Codigo.text()
122         #Consultar livro
123         livro = Consultar(cod)
124         #colocando dados nos campos
125         self.Titulo.setText(livro.titulo)
126         self.Ano.setText(livro.ano)
127         self.Autor.setText(livro.autor)
128         self.Editora.setText(livro.editora)
```

Nesse caso, a BLL irá armazenar todos os dados do livro que foram digitados e guardar na memória, para quando fizermos uma consulta, os mesmos dados serem consultados.

Classe Erro

Em todos os programas (alguns foram mostrados e outros não), podemos implementar uma classe erro, que nos retorna um aviso, caso os dados digitados não satisfaçam às condições para o programa funcionar corretamente. A classe erro é detalhada logo nas imagens abaixo utilizando-se do programa de calcular a área do triângulo:

janela.py | classe1.py | erro.py

erro.py > Erro

```python
#nome da classe
class Erro:
    #início da classe com (se tem erro, mensagem de erro)
    def __init__(self,erro, msg):
        #recebendo valores colocado nos parênteses
        self.erro = erro
        self.mensagem = msg

        #finalizandode erro
        pass
```

```python
#importando a classe
from classe1 import CalcularArea
from erro import Erro
from PyQt5 import QtCore, QtGui, QtWidgets
```

```python
    #ação do botão
    def calcular_click(self):
        #criando o objeto
        DeuErro =  Erro( False,"Houve Erro")
        #tentando fazer os comandos
        try:
            self.b = float(self.valorBase.text() )
            self.h = float(self.valorAltura.text() )
            if self.b<1 or self.h<1:
                DeuErro.erro = True
                self.Resultado.setText(DeuErro.mensagem)
        #caso houver erro
        except:
            DeuErro.erro = True
            self.Resultado.setText(DeuErro.mensagem)
        #se não houver erro
        if  DeuErro.erro == False:
            #criando o objeto
            calc = CalcularArea(self.b,self.h)
            #colocando calc.area que é a área do triângulo
            result = "Área = " + str(calc.area)
            #editar texto da label
            self.Resultado.setText(result)
```

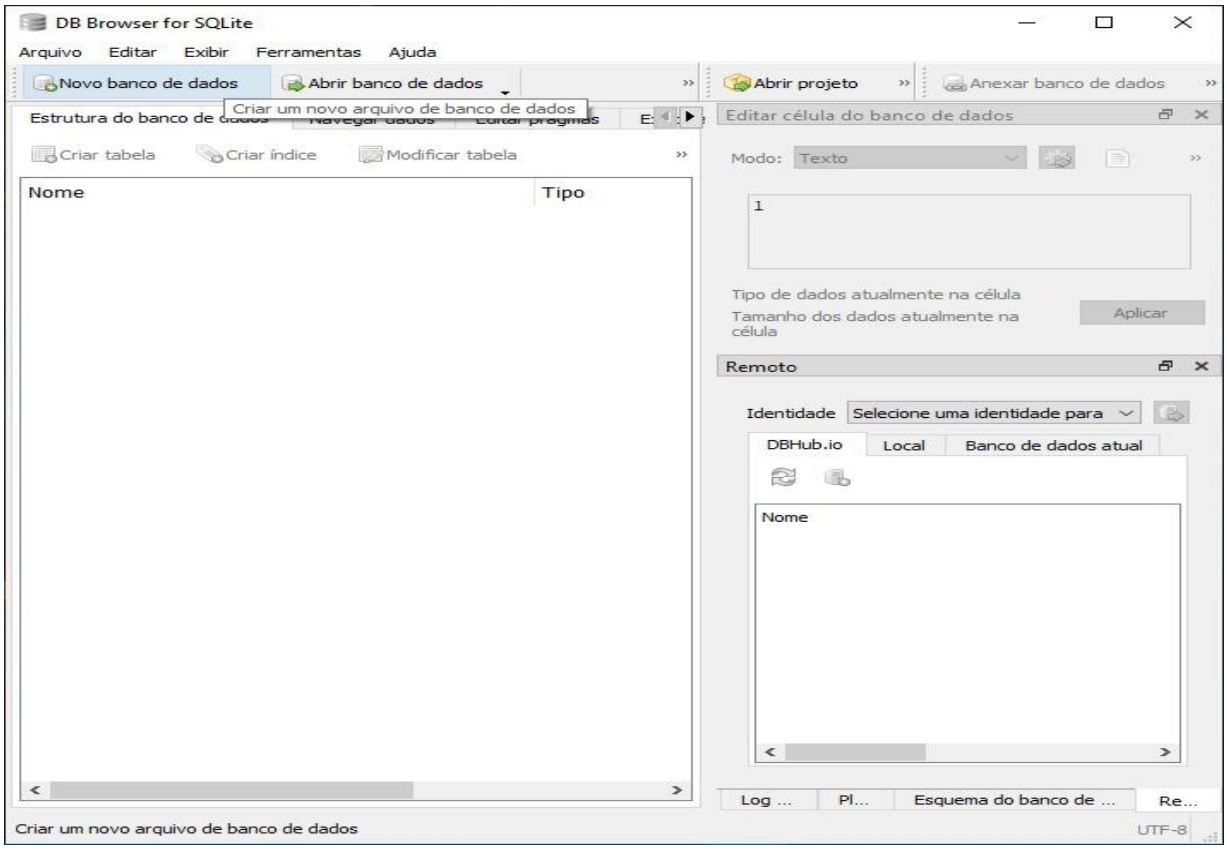
Banco de dados

Um banco de dados é um conjunto de dados que têm relação entre si, representando informações sobre algo ou alguém, ou seja, todos os tipos de informações que se relacionem, possam ser agrupadas e se tratarem de um mesmo assunto podemos dizer que temos um banco de dados. Podemos ter como exemplo de bancos de dados: listas telefônicas, catálogo de CDs ou até um sistema de controle de RH de uma empresa.

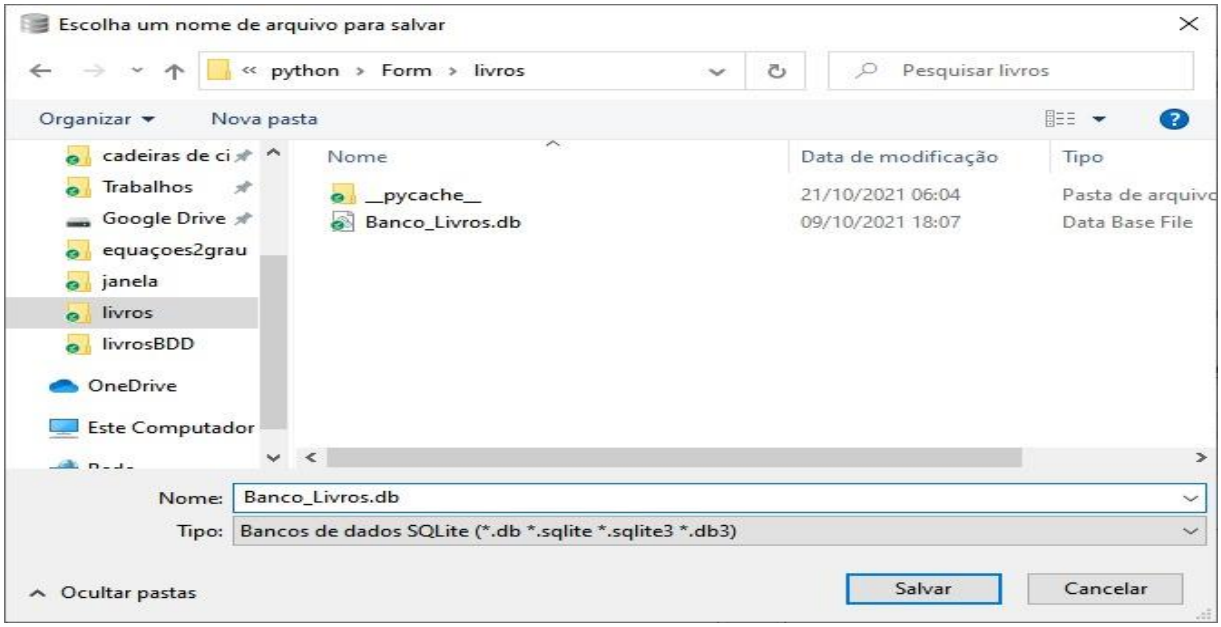
Um sistema de gerenciamento de banco de dados (SGBD) é um software que possui recursos capazes de manipular as informações do banco de dados e interagir com o usuário. Exemplos de SGBDs são: Oracle, SQL Server, DB2, PostgreSQL, MySQL, o próprio Access ou Paradox, entre outros.

Projeto CRUD

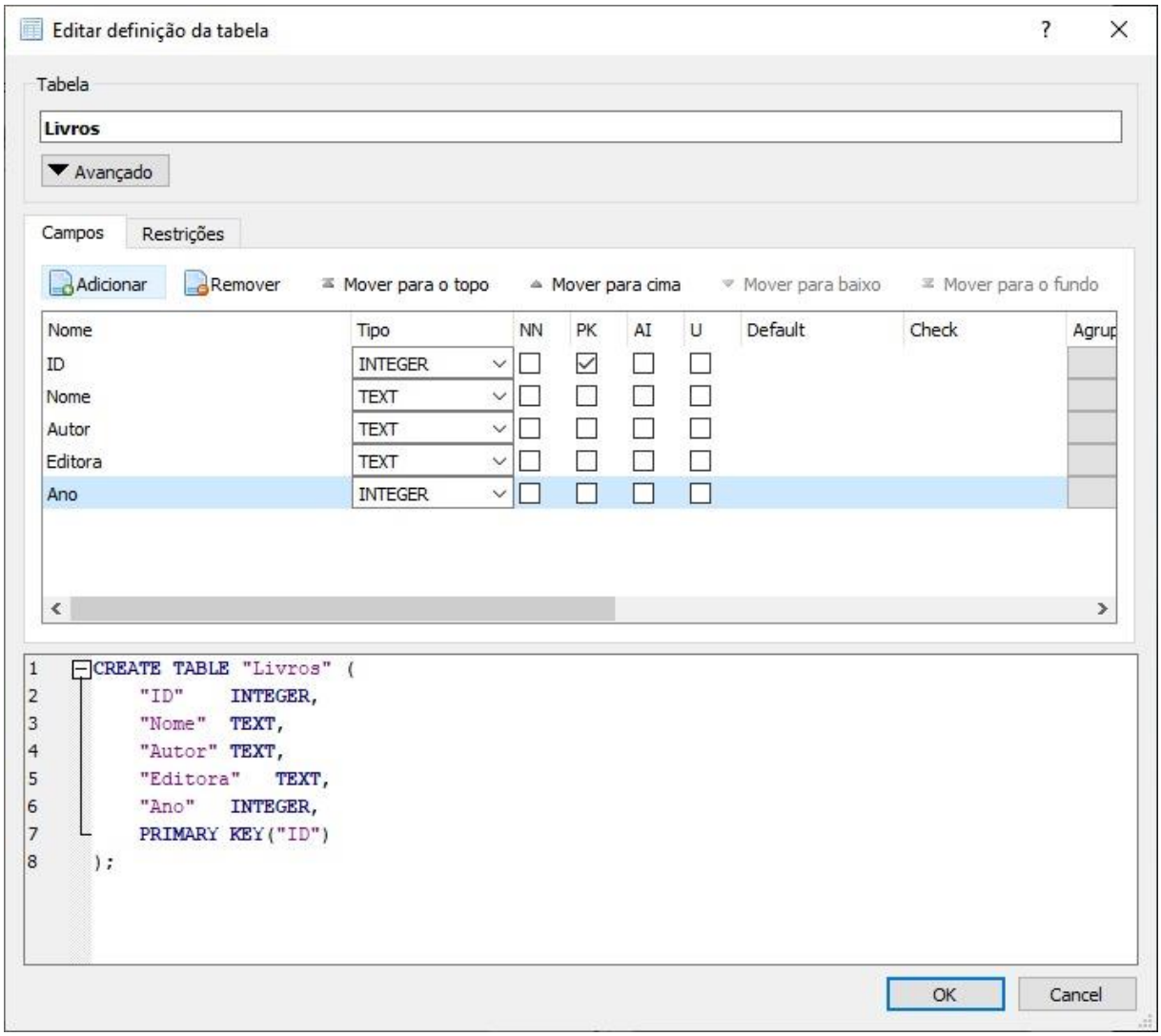
Neste momento, estaremos refazendo o exercício de fixação 4, primeiramente apresentado no tópico de POO. Desta vez, porém, o exercício será completo com a implementação de um banco de dados, que no caso, será utilizado o SQL Lite.



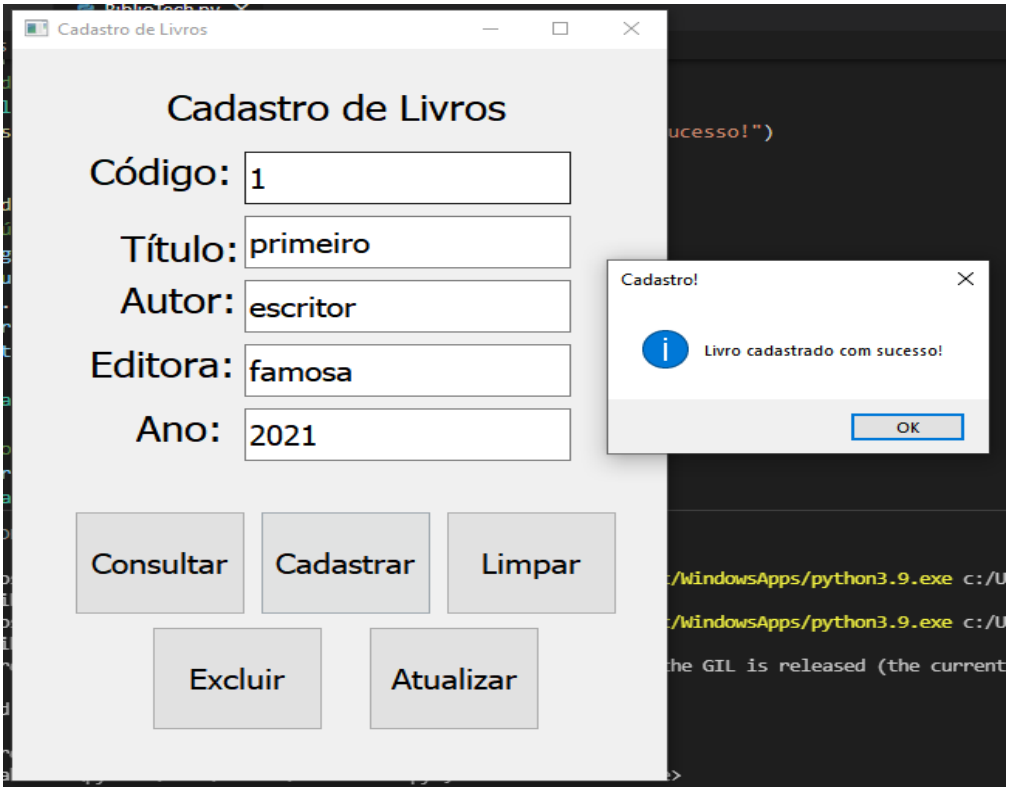
Abrindo o aplicativo SQL Lite, iremos clicar em novo banco de dados.



Selecionamos a pasta desejada e criamos o nome do banco de dados para que fique fácil localizá-lo.



Adicionamos cada coluna com seu respectivo texto, e para o caso do ID selecionamos como Primary Key, já que não poderá se repetir.



A interface (IHM/ULL) nos dá alguns campos a serem preenchidos e os botões com as ações que poderão ser feitas.

Classes Principais (CRUD)

ClasseFuncoes.py | erro.py | BiblioTech.py

ClasseFuncoes.py > ...

```python
#importando sqlite3
import sqlite3

#nome da classe cadastrar
class Cadastrar:
    #início da classe com (codigo, titulo, autor, editora,ano)
    def __init__(self, codigo0, titulo, autor, editora,ano):

        #conexão
        banco = sqlite3.connect("Banco_Livros.db")
        cursor = banco.cursor()
        #add no banco
        cursor.execute("INSERT INTO Livros VALUES('"+str(codigo0)+"','"+titulo+"','"+autor+"','"+editora+"','"+str(ano)+"')")
        banco.commit()
        #fecha banco
        banco.close()
        pass

#classe Consulta
class Consultar:
        #início da classe com (codigo)
    def __init__(self, codigo0):
        #conexão
        banco = sqlite3.connect("Banco_Livros.db")
        cursor = banco.cursor()
        # pegando resultados
        cursor.execute("SELECT titulo, autor, editora, ano FROM Livros WHERE codigo = "+codigo0+"")
        #passando dados pegos para uma lista
        dados = cursor.fetchone()

        #add no campo titulo
        self.titulo = str(dados[0])
```

```
33          #add no campo autor
34          self.autor = str(dados[1])
35          #add no campo editora
36          self.editora = str(dados[2])
37          #add no campo ano
38          self.ano = str(dados[3])
39          #fecha banco
40          banco.close()
41          pass
42
43
44  #classe Excluir
45  class Excluir:
46          #início da classe com (codigo)
47      def __init__(self, codigo0):
48          #conexão
49          banco = sqlite3.connect("Banco_Livros.db")
50          cursor = banco.cursor()
51
52          # pegando resultados
53          cursor.execute("DELETE FROM Livros WHERE codigo = "+codigo0+"")
54          banco.commit()
55          #fecha banco
56          banco.close()
57          pass
58
```

```
59  #classe Atualiza
60  class Atualizar:
61          #início da classe com (codigo)
62      def __init__(self,codigo0, titulo, autor, editora,ano):
63
64          Excluir(codigo0)
65          Cadastrar(codigo0, titulo, autor, editora,ano)
66          pass
```

Criamos as classes para as funções: Cadastrar, Consultar, Excluir e Atualizar os dados, e em cada uma delas é feita a conexão com o banco de dados.

Classe Erro (CRUD)

ClasseFuncoes.py | erro.py | BiblioTech.py

erro.py > ValidaCampos > __init__

```python
#importando messagebox
from tkinter import messagebox
from ClasseFuncoes import Consultar
#importando sqlite3
import sqlite3



#nome da classe principal
class Erro:
    #início da classe com (se tem erro, mensagem de erro)
    def __init__(self,erro, msg):
        #recebendo valores colocado nos parênteses
        self.erro = erro
        self.mensagem = msg

        if self.erro:
            #messagebox(título,mensagem)
            messagebox.showinfo(title="Erro!", message=msg)

        #finalizandode erro
        pass

#Verificar a conexão ao BDD
class VerificarConexão:
    #início da classe
    def __init__(self):
        #Bandeira erro
        self.TemErroDeConexão = False
        #Verificando conexão ao banco
        try:
            banco = sqlite3.connect("Banco_Livros.db")
            cursor = banco.cursor()
        except:
            Erro(True, "Banco de Dados não conectado")
            self.TemErroDeConexão = True

```

```python
#classe validar Código
class ValidaCampoCódigo:
        def __init__(self, codteste):
         #Bandeira erro
         self.TemErro = False
         conexao = VerificarConexão()
         if conexao.TemErroDeConexão == False:

             #Verificando campo Código
             try:
                 cod = codteste
                 livro = Consultar(cod)
             except:
                 Erro(True, "Código não encontrado")
                 self.TemErro = True
         else:
             Erro(True, "Banco de Dados não conectado")
         pass


#classe validar todos os campos
class ValidaCampos:
        def __init__(self, codteste, tituteste, autteste, editteste,anosteste):
        #Bandeira erro
            self.TemErro = False
            conexao = VerificarConexão()
            if conexao.TemErroDeConexão == False:

                #Verificando campo Código podendo colocar números e letras
                try:
                    cod = codteste
                    if cod == "":
                        Erro(True, "Campo Código está digitado incorretamente")
                        self.TemErro = True
                except:
                        Erro(True, "Campo Código está digitado incorretamente")
                        self.TemErro = True

                #Verificando campo Título
                try:
                    titu = tituteste
                    if titu == "":
                        Erro(True, "Campo Título está digitado incorretamente")
                        self.TemErro = True
                except:
                    Erro(True, "Campo Título está digitado incorretamente")
                    self.TemErro = True
```

```
                    Erro(True, "Campo Título está digitado incorretamente")
                    self.TemErro = True

                #Verificando campo Ano
                try:
                    anos = int(anosteste)
                    if anos < 0:
                        Erro(True, "Campo Ano está digitado incorretamente")
                        self.TemErro = True
                except:
                    Erro(True, "Campo Ano está digitado incorretamente")
                    self.TemErro = True

                #Verificando campo Autor
                try:
                    aut = autteste
                    if aut == "":
                        Erro(True, "Campo Autor está digitado incorretamente")
                        self.TemErro = True
                except:
                    Erro(True, "Campo Autor está digitado incorretamente")
                    self.TemErro = True

                    self.TemErro = True

                #Verificando campo Editora
                try:
                    edit = editteste
                    if edit == "":
                        Erro(True, "Campo Editora está digitado incorretamente")
                        self.TemErro = True
                except:
                    Erro(True, "Campo Editora está digitado incorretamente")
                    self.TemErro = True
        else:
            Erro(True, "Banco de Dados não conectado")

        pass
```

Criamos também uma classe erro para que sejam verificadas todas as possibilidades de erros do usuário que impeçam que o programa funcione corretamente.

Classe principal

```
#importar classes
from PyQt5 import QtCore, QtGui, QtWidgets
from ClasseFuncoes import Cadastrar, Consultar, Excluir, Atualizar
from erro import Erro, ValidaCampoCódigo, ValidaCampos, VerificarConexão
#importar messagebox
from tkinter import messagebox
```

Começamos importando as classes secundárias para a classe principal.

```
        self.retranslateUi(CadastroLivros)
        self.btn_Limpar.clicked.connect(self.Codigo.clear)
        self.btn_Limpar.clicked.connect(self.Titulo.clear)
        self.btn_Limpar.clicked.connect(self.Ano.clear)
        self.btn_Limpar.clicked.connect(self.Editora.clear)
        self.btn_Limpar.clicked.connect(self.Autor.clear)
        #funções dos botões
        self.btn_Cadastrar.clicked.connect(self.Cadastrar_clicado)
        self.btn_Consultar.clicked.connect(self.Consulta_clicado)
        self.btn_Excluir.clicked.connect(self.Excluir_clicado)
        self.btn_Atualizar.clicked.connect(self.Atualizar_clicado)

        QtCore.QMetaObject.connectSlotsByName(CadastroLivros)
```

Definimos as funções dos botões ao serem clicados. No caso de o botão limpar, todos os campos serão limpos; no caso dos outros, irão levar à função necessária.

```
132
133     #métodos
134     def Cadastrar_clicado(self):
135         #Passando conteúdo dos campos
136
137         cod = self.Codigo.text()
138         titu = self.Titulo.text()
139         anos = self.Ano.text()
140         aut = self.Autor.text()
141         edit = self.Editora.text()
142         #validar
143         validar = ValidaCampos(cod, titu, aut, edit,anos)
144
145         #Verificar se pode salvar no BDD
146         if validar.TemErro == False:
147             #Cadastrar livros
148             livro = Cadastrar(cod, titu, aut, edit,anos)
149             messagebox.showinfo(title="Cadastro!", message="Livro cadastrado com sucesso!")
150
151
152         #Limpar campos
153         self.Codigo.clear()
154         self.Titulo.clear()
155         self.Ano.clear()
156         self.Autor.clear()
157         self.Editora.clear()
```

```
158
159     def Consulta_clicado(self):
160         #Passando conteúdo do campo codigo
161         cod = self.Codigo.text()
162         #validar
163         validar = ValidaCampoCódigo(cod)
164
165         if validar.TemErro == False:
166             #Consultar livro
167             livro = Consultar(cod)
168             #colocando dados nos campos
169             self.Titulo.setText(livro.titulo)
170             self.Ano.setText(livro.ano)
171             self.Autor.setText(livro.autor)
172             self.Editora.setText(livro.editora)
173
174     def Excluir_clicado(self):
175         #Passando conteúdo do campo codigo
176         cod = self.Codigo.text()
177         #validar
178         validar = ValidaCampoCódigo(cod)
179
180         if validar.TemErro == False:
181             #Excluir dados
182             livro = Excluir(cod)
183             messagebox.showinfo(title="Excluir!", message="Livro excluído com sucesso!")
184
```

```
184
185
186     def Atualizar_clicado(self):
187         #Passando conteúdo do campo codigo
188         cod = self.Codigo.text()
189         titu = self.Titulo.text()
190         anos = self.Ano.text()
191         aut = self.Autor.text()
192         edit = self.Editora.text()
193         #validar
194         validar = ValidaCampos(cod, titu, aut, edit,anos)
195
196         #Verificar se pode salvar no BDD
197         if validar.TemErro == False:
198             livro = Atualizar(cod, titu, aut, edit,anos)
199             messagebox.showinfo(title="Atualizar!", message="Livro atualizado com sucesso!")
200
```

Passando por todos os métodos podemos, assim, trazer tudo o que foi definido nas classes para as ações dos botões e validarmos tudo.

Projeto final

Com base em tudo o que foi aprendido anteriormente, implementaremos agora uma API/Serviços de terceiros, em nosso projeto. No geral, o projeto consiste em pesquisarmos, pelo CEP, o nome de uma rua, seu bairro, sua cidade e estado. Serviço comumente utilizado pelos correios ou por pessoas que fazem entregas.

```
pesquisa_cep.py > Procurar > __init__
1   #importar biblioteca para usar API
2   import requests
3
4   class Procurar:
5         #início da classe com (cep)
6       def __init__(self,codigo):
7         #Pegar cep digitado
8         cep = codigo
9         #Verificar cep
10        if len(cep) != 8:
11            self.loug = "CEP inválido"
12            self.bar = "-"
13            self.cid = "-"
14            self.est = "-"
15        #caso não houver erro executar:
16        else:
17            #usando API
18            request = requests.get(f'https://viacep.com.br/ws/{cep}//json/')
19            adress = request.json()
20            #Verificar cep
21            if 'erro' not in adress:
22                self.loug = f"{adress['logradouro']}"
23                self.bar = f"{adress['bairro']}"
24                self.cid = f"{adress['localidade']}"
25                self.est = f"{adress['uf']}"
26            #caso não houver erro executar:
27            else:
28                self.loug = "CEP inválido"
29                self.bar = "-"
30                self.cid = "-"
31                self.est = "-"
```

Iniciamos criando uma classe para que a procura do CEP seja feita, para isso importamos a biblioteca requests e pedimos para verificar no sistema se o CEP indicado é válido, caso não seja, uma mensagem de erro será exibida, caso esteja tudo certo, todos os dados serão implementados para serem exibidos na classe principal.

main.py > ...

```python
#importar bibliotecas
from PyQt5 import uic, QtWidgets, QtCore, QtGui
from pesquisa_cep import Procurar

#funções
def Procurar_click(self):
    cep_digitado = tela.Txt_cep.text()
    ende = Procurar (cep_digitado)
    tela.Logradouro.setText(ende.loug)
    tela.Bairro.setText(ende.bar)
    tela.Cidade.setText(ende.cid)
    tela.Estado.setText(ende.est)

def Limpar_click(self):
    tela.Logradouro.setText("-")
    tela.Bairro.setText("-")
    tela.Cidade.setText("-")
    tela.Estado.setText("-")
#preparando comandos
app = QtWidgets.QApplication([])
tela = uic.loadUi("CEP_Form.ui")

#colocar o método
tela.Procurar.clicked.connect(Procurar_click)
tela.Limpar.clicked.connect(Limpar_click)

#apresentar janela
tela.show()
app.exec()
```

Na classe principal, definimos as funções dos botões procurar e limpar, e pedimos para exibir a tela.

Por fim, essa será a tela exibida, que poderá ser customizada de acordo com o que o cliente exigir, respeitando as limitações da Framework em Python.

Bibliografia

AULA 9 – COMANDOS DE REPETIÇÃO FOR E DO WHILE. **Aula 9 – Comandos de Repetição For e Do While**. Disponível em: https://www.ime.usp.br/~mms/mac1101s2019/aula10%20-%20Python%20-%20o%20comando%20for%20de%20repeticao.pdf. Acesso em: 23 jun. 2021.

AULA14 VARIÁVEIS INDEXADAS VETORES. **Aula14 variáveis indexadas vetores**. Disponível em: https://www.ime.usp.br/~mms/mac1101s2019/aula16%20-%20Python%20-%20Listas%20-%20variaveis%20indexadas%20ou%20vetores.pdf. Acesso em: 23 jun. 2021.

BLOG VOITTO. **O que é Python e pra que serve?**. Disponível em: https://www.voitto.com.br/blog/artigo/python. Acesso em: 15 mai. 2021.

CODINGAME. **Comandos de Entrada e Saída em Python**. Disponível em: https://www.codingame.com/playgrounds/34774/introducao-a-programacao-python---prof--marco-vaz/comandos-de-entrada-e-saida. Acesso em: 6 jun. 2021.

CURSO INTRODUTÓRIO DE PYTHON. **Estruturas de repetição**. Disponível em: http://curso.grupysanca.com.br/pt/latest/repeticao.html. Acesso em: 19 jun. 2021.

DEV MEDIA. **For Python: Estrutura de repetição for**. Disponível em: https://www.devmedia.com.br/for-python-estrutura-de-repeticao-for/38513. Acesso em: 12 jun. 2021.

DEV MEDIA. **Operadores no Python**. Disponível em: https://www.devmedia.com.br/operadores-no-python/40693. Acesso em: 8 jun. 2021.

DEV MEDIA. **Python: Estrutura de repetição while**. Disponível em: https://www.devmedia.com.br/python-estrutura-de-repeticao-while/38546. Acesso em: 12 jun. 2021.

DICAS DE PROGRAMAÇÃO. **Conheça os operadores lógicos!**. Disponível em: https://dicasdeprogramacao.com.br/operadores-logicos/#:~:text=As%20opera%C3%A7%C3%B5es%20l%C3%B3gicas%20trabalham%20sobre,E%20N%C3%83O%2DOU%2DEXCLUSIVO. Acesso em: 13 jun. 2021.

FIC INTRODUÇÃO A PROGRAMAÇÃO. **FIC Introdução a Programação - aula 15 - Vetor e Matriz**. Disponível em: http://docente.ifsc.edu.br/edilson.hipolito/materiais/2016-2/FIC-introducao-a-programacao-computadores-python/FIC%20Introdu%C3%A7%C3%A3o%20%C3%A0%20Programa%C3%A7%C3%A3o%20-%20aula%2016%20-%20Vetor%20e%20Matriz.pdf. Acesso em: 23 jun. 2021.

GEEK HUNTER. **10 Melhores IDEs e Editores de Código em Python para 2021**. Disponível em: https://blog.geekhunter.com.br/ides-e-editores-de-codigo-em-python-para-2021/. Acesso em: 16 mai. 2021.

KENZIE ACADEMY BRASIL. **O que é Python, para que serve e por que aprender?**. Disponível em: https://kenzie.com.br/blog/o-que-e-python/. Acesso em: 18 mai. 2021.

O ESTATÍSTICO. **10 IDEs para Programar em Python**. Disponível em: https://oestatistico.com.br/10-ides-para-programar-em-python/. Acesso em: 17 mai. 2021.

PUCRS - FACULDADE DE INFORMÁTICA - PROGRAMAÇÃO PARA CIENCIAS BIOLÓGICAS. **Variáveis em Python**. Disponível em: https://www.inf.pucrs.br/pinho/PCB/Variaveis/Variaveis.html. Acesso em: 10 jun. 2021.

PUCRS - PROGRAMAÇÃO EM PYTHON. **Comandos de Decisão**. Disponível em: https://www.inf.pucrs.br/pinho/PCB/ComandosDeDecisao/Decisao.htm. Acesso em: 19 jun. 2021.

WIKIPÉDIA, A ENCICLOPÉDIA LIVRE. **Python**. Disponível em: https://pt.m.wikipedia.org/wiki/Python. Acesso em: 19 mai. 2021.

YOUTUBE. **Python - Aula 02 - Instalar e configurar o Visual Studio para criar aplicações Python**. Disponível em: https://www.youtube.com/watch?v=r-utOIIjlWs&t=170s. Acesso em: 17 mai. 2021.

DEVMEDIA. **Como criar minha primeira classe em Python**. 2020. Disponível em: https://www.devmedia.com.br/como-criar-minha-primeira-classe-em-python/38912. Acesso em: 02 set. 2021.

SILVA, Regis da. **Introdução a Classes e Métodos em Python (básico)**. 2014. Disponível em: http://pythonclub.com.br/introducao-classes-metodos-pythonbasico.html. Acesso em: 02 set. 2021.

DEVMEDIA. **Conceitos Fundamentais de Banco de Dados.** Disponível em: https://www.devmedia.com.br/conceitos-fundamentais-de-banco-de-dados/1649. Acesso em: 09 out. 2021.