**Projeto Integrado de Síntese - Dados**

Instituição de Ensino



**Aluno:** Tiago Fernando Piveta **Contato:** [tiago.piveta@outlook.com](mailto:tiago.piveta@outlook.com)  
**Matricula:** 38470002  
**Data da elaboração:** 18/04/2024  
**Data da entrega:** 03/05/2024

1.Introdução:

* Este portfolio é destinado a Instituição de Ensino Faculdade Anhanguera, para a apreciação do tutor responsável e do professor(a) da matéria.  
  Para a elaboração do código em Python, foi utilizado o programa Visual Studio Code (Microsoft), o objetivo do código é a criação de uma GUI (Graphical User Interface) ou Interface Gráfica do Usuário, onde está GUI irá desempenhar o seguinte papel:  
  Cadastro de Cliente -> Consulta de Cliente -> Alteração de Cadastro  
  Cadastro de Livros -> Consulta de Livros (Título do livro, Nome do autor, Ano de publicação) -> Alteração de cadastro do Livro. Ainda no campo responsável pela Biblioteca teremos a janela separada pra o empréstimo e devolução de exemplares.  
  Peço a gentileza do tutor e professor(a) responsável que acessem o link <https://youtu.be/kpUNs0l8SAw> este é um vídeo privado que subi no Youtube onde demonstro o funcionamento do programa.

2. O que é Gui:

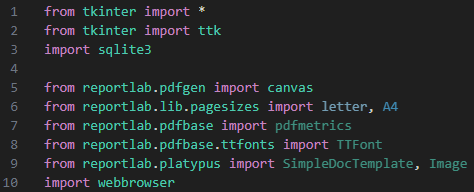
* **GUI (Interface Gráfica do Usuário): é uma forma de interação entre os usuários e os sistemas computacionais. Em vez de digitar comandos em um terminal, a GUI permite que os usuários interajam com o sistema por meio de elementos visuais, como botões, menus e ícones.  
  No Python, existem várias bibliotecas que você pode usar para criar GUIs. Vou falar sobre três delas: Tkinter, PySimpleGUI e PyQt.**
* **Tkinter: é a biblioteca padrão do Python para criar interfaces gráficas. Ela é simples e fácil de usar, tornando-a uma ótima opção para iniciantes. Com o Tkinter, você pode criar janelas, botões, menus, caixas de texto e outros widgets. Além disso, o Tkinter é uma biblioteca leve, o que significa que ela não tornara seu programa desnecessariamente pesado.**
* **PysimpleGUI:** é uma biblioteca Python para criar GUIs, como o nome sugere, o PySimpleGUI é projetado para ser simples e fácil de usar, ele oferece uma abordagem mais moderna e “pythonica” para criação de GUIs, tornando-a uma boa escolha se você estiver procurando algo um pouco mais atualizado que o Tkinter.
* **PyQT:** é uma biblioteca que permite criar GUIs usando o QT, um popular kit de ferramentas de GUI, o PyQT é extremamente poderoso e flexível, permitindo que você crie GUIs complexas com uma grande variedade de recursos, No entanto, essa flexibilidade vem com uma curva de aprendizado mais acentuada, então o PyQT pode ser mais adequado para desenvolvedores mais experientes.

**3.** Explicação do código:

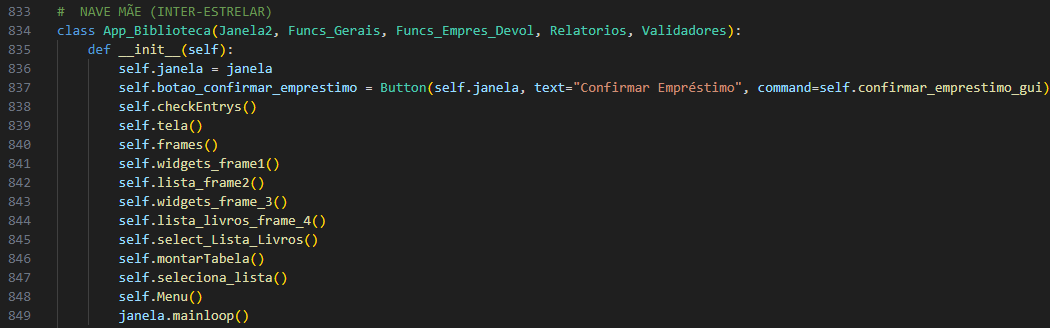
* O primeiro passo foi importar as minhas bibliotecas, para poder desempenhar com eficiência a proposta que atividade exige.  
  As bibliotecas importadas foram Tkinter(responsável pela Interface Gráfica), ttk(um modulo dentro da biblioteca gráfica Tkinter), para armazenamento de dados utilizei a biblioteca sqlite3(biblioteca de fácil instalação e não é necessário aplicativo externo para gerar conexão como é o caso do MySQL), o Reportlab(gera arquivos .pdf esta biblioteca será a responsável por gerar nossos relatórios quando solicitado), por último a biblioteca webrowser(sera responsável por abri uma página web com o nosso .pdf)
* **TTK:** O ‘ttk’ é um modulo dentro da biblioteca de interface gráfica ‘Tkinter’ para Python, que fornece acesso a um conjunto de widgets com estilo melhorado. TTK significa ‘Themed Tk’ e foi introduzido para superar limitações estéticas do Tkinter padrão, permitindo que os widgets sejam estilizados de maneira mais moderna e atraente que se adapte ao sistema do usuário.

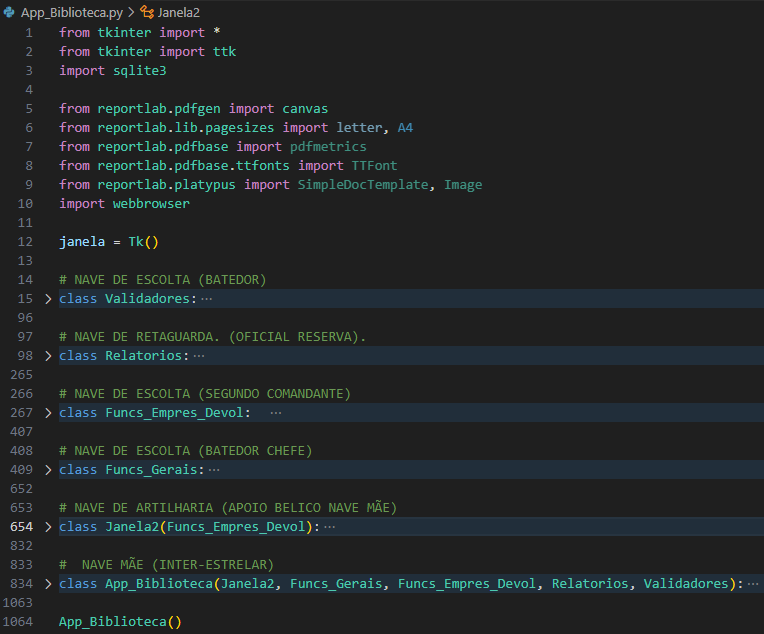
**Principais Características do ttk.**

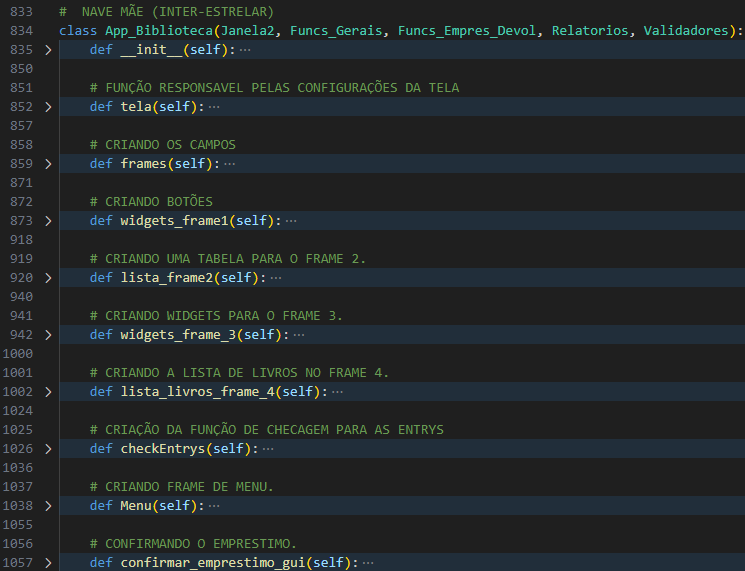
* **Tema Baseado**: Os widgets ttk são construídos com suporte a temas que permitem uma aparência consistente com o sistema operacional ou permitem a personalização através de temas definidos pelo usuário.
* **Widgets Melhorados**: Ele oferece versões melhoradas de muitos dos widgets padrão encontrados no Tkinter, como **Button**, **Label**, **Checkbutton**, **Radiobutton**, **Frame**, e muitos outros.
* **Facilidade de Uso**: Ele é usado de maneira muito similar ao Tkinter, facilitando a migração de projetos Tkinter existentes para ttk para melhorar a interface gráfica.
* **Portabilidade**: Assim como o Tkinter, ttk é altamente portátil e funciona em diferentes sistemas operacionais sem a necessidade de alterar o código.

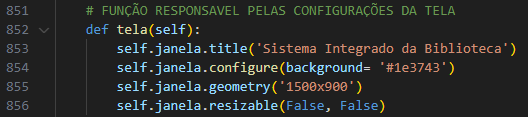


O próximo passo que foi dado foi passar os parâmetros de janela, configuração de janela, inserir o mainloop para manter a janela aberta, para abrigar todas as funcionalidades iniciais foi criada a class App\_Biblioteca (esta será a classe principal de todo código), em seguida irei apresentar uma visão geral do código e com o decorrer da documentação entrarei em mais detalhes.

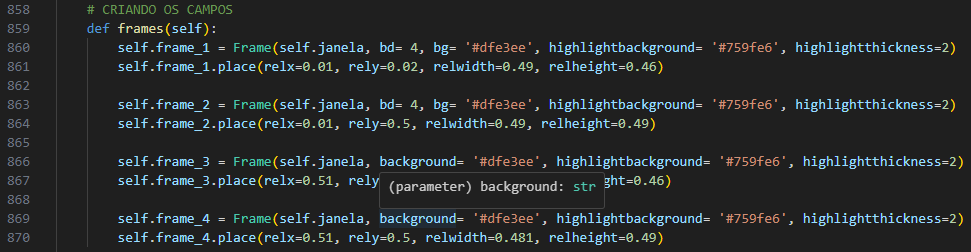
  
Aqui temos o **\_\_init\_\_** o nosso construtor responsável por chamar as funções necessárias, para o funcionamento do programa. Após a importação das bibliotecas defini a variável **janela = Tk()** padrão para o Tkinter e dentro do \_\_init\_\_ defini o **self.janela = janela** e **janela.mainloop()** e fora das classes chamei a classe principal, já garantindo que a janela ficara aberta pelo tempo que achar necessário.



**4.** Detalhamento da classe App\_Biblioteca:  


Para executar a tarefa primeiro criei a **def tela**, esta função contem toda a configuração da janela, como tamanho padrão, se irá mudar de tamanho ou não, o titulo da nossa janela e a cor de fundo da janela.

Na sequência já defini quantos frames iria utilizar e a distribuição que iria fazer por toda a janela. O **Frame 1** é o campo onde terá os textos como código, nome cliente, telefone e cidade, neste frame também irá conter os campos para preenchimento com os dados solicitados. O **Frame 2** será onde teremos nossa primeira Treeview, após o cadastro do cliente sea possível ver o cadastrado logo abaixo ‘importante salientar que a ordenação da Treeview será por nome do cliente em ordem alfabética.’  
O **Frame 3** irá conter os textos como titulo do livro, nome do autor, ano de publicação, assim como os campos para preenchimentos dos dados e os botões de confirmar cadastro do livro, alterar informações do livro entre outros. O **Frame 4** será a nossa segunda Treeview, após o cadastro do livro será possível vê-lo logo abaixo, a ordenação também se dará por ordem alfabética.



Este trecho de código é responsável por configurar a interface de usuário para a gestão de clientes, incluindo botões para diversas ações (limpar campos, buscar, adicionar, alterar e excluir clientes) e campos de entrada para inserir informações como código, O código inclui a criação e posicionamento de vários widgets dentro de self.frame\_1. A configuração visual de cada widget é detalhada com cores, fontes e tamanhos específicos.

**Botões:**

Limpar todos os campos (bt\_limpar):

**Função:** Limpa todos os campos de entrada na interface.

**Estilização:** Botão com fundo azul (#107db2), texto branco, fonte Verdana tamanho 8 e negrito.

**Posicionamento:** Localizado no topo, à esquerda do frame.

Buscar Cliente (bt\_buscar):

**Função:** Inicia uma busca pelo cliente com os dados fornecidos nos campos de entrada.

**Estilização:** Semelhante ao botão de limpar.

**Posicionamento:** Ao lado do botão de limpar.

Confirmar Novo Cadastro de Cliente (bt\_novo):

**Função:** Adiciona um novo cliente ao sistema com as informações inseridas.

**Estilização:** Semelhante aos botões anteriores.

**Posicionamento:** Próximo ao centro do frame, mas mais para a direita.

Alterar Cadastro do Cliente (bt\_alterar):

**Função:** Modifica o cadastro de um cliente existente.

**Estilização e Posicionamento:** Localizado mais abaixo no frame.

Excluir Cliente (bt\_apagar):

**Função:** Remove o cadastro de um cliente do sistema.

**Estilização e Posicionamento:** Localizado na parte inferior do frame.

**Campos de Entrada e Labels:**

Código (lb\_codigo e codigo\_entry):

**Descrição:** Campo para inserir o código de identificação do cliente.

**Validação:** validate='key' com uma função de validação self.vcmd1 para restringir entradas.   
OBS: A VALIDAÇÃO FAZ PARTE DE UMA CLASSE SEPARADA, E PARA COMPOR A VALIDAÇÃO FOI NECESSARIO CRIAR UMA DEF E SO ENTÃO CRIAR AS VARIAVEIS QUE ESTÃO NAS ENTRYS.

Nome (lb\_nome e nome\_entry):

**Descrição:** Campo para inserir o nome completo do cliente.

**Validação:** Similar ao campo de código, mas usa self.vcmd3.

Telefone (lb\_telefone e telefone\_entry):

**Descrição:** Campo para inserir o número de telefone do cliente, incluindo o DDD.

**Validação:** Usa self.vcmd2.

Cidade (lb\_cidade e cidade\_entry):

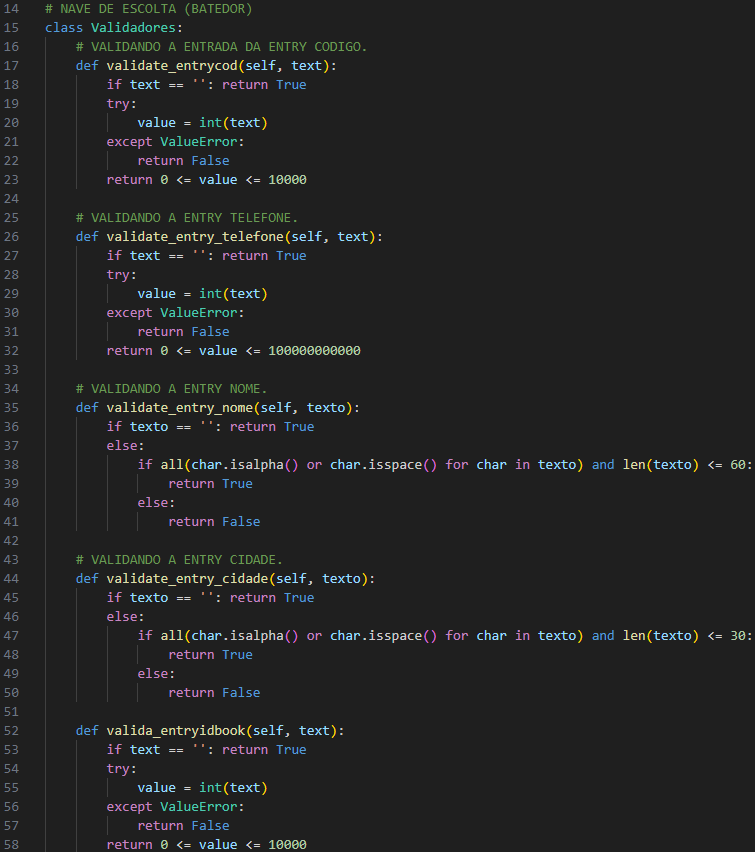
**Descrição:** Campo para inserir o nome da cidade do cliente.

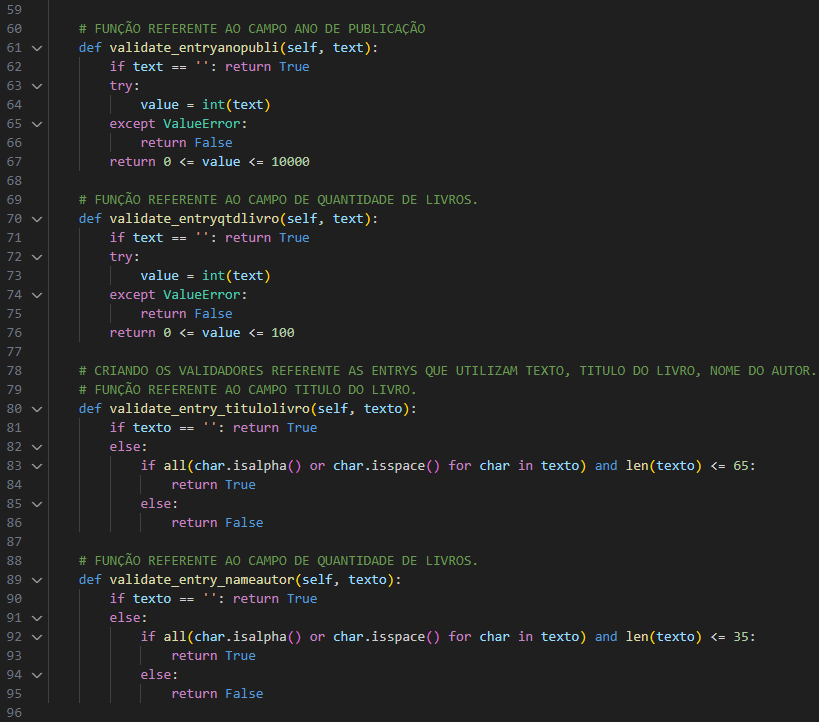
**Validação:** Usa self.vcmd4.

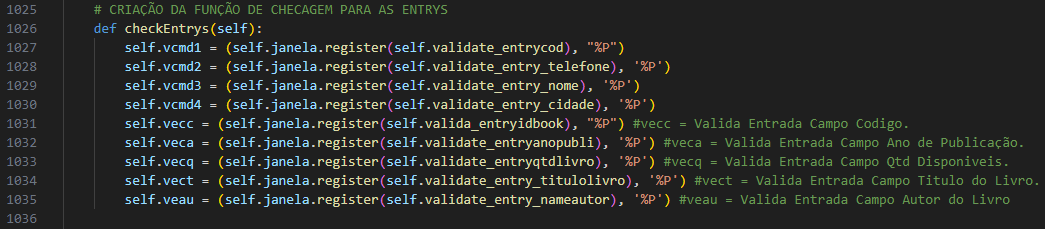
**Estilização e Layout:**

Todos os widgets são cuidadosamente estilizados com cores específicas para fundo e texto, e a fonte Verdana é usada consistentemente para manter a uniformidade visual. O uso de place para posicionamento permite um layout preciso baseado em coordenadas relativas dentro do frame\_1.

**Validação:**

Os campos de entrada utilizam a validação Tkinter validate='key', que permite que os eventos de tecla sejam interceptados e validados por uma função específica antes de serem aceitos no campo. Isso é útil para garantir que apenas caracteres O código inclui a criação e posicionamento de vários widgets dentro de self.frame\_1. A configuração visual de cada widget é detalhada com cores, fontes e tamanhos específicos.  
  
**5.** Explicação da Classe Validadores:  
Para maior contextualização será necessário a demonstração da **classe Validadores** que também é responsável pela entrys do frame 2, responsável pela biblioteca, esta classe é a responsável por limitar a inserção de dados errados, como inserir texto no campo telefone ou numero no campo de cidade. Após a criação da classe Validadores foi necessário criar uma def afim de criar as variáveis que utilizei nas entrys como vcmd1, vcmd2, vecc, veca. 



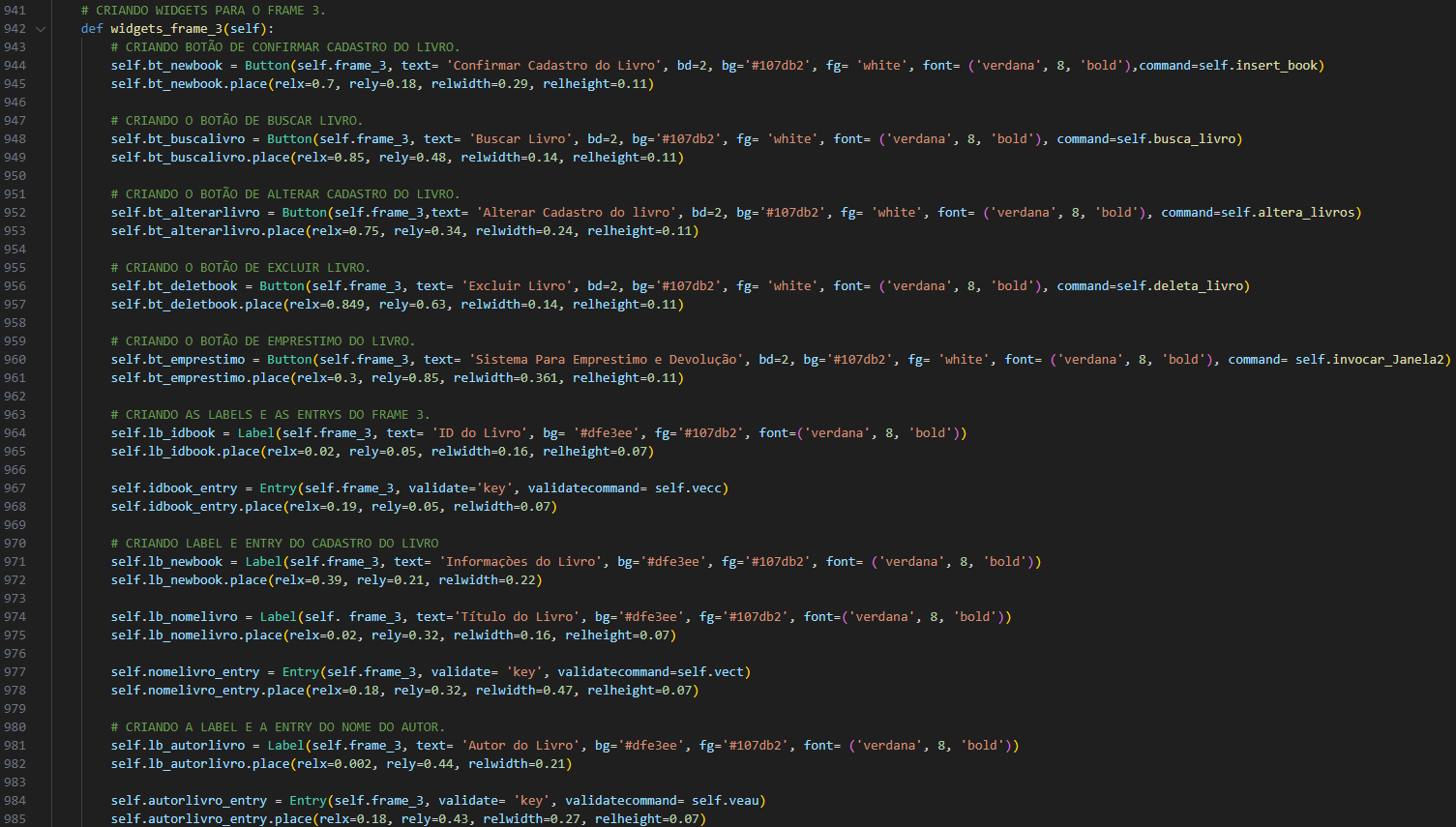
  
Na def responsável pela entry telefone, perceba que no final tem uma sequência de 0(zeros), quando estava configurando esta def tentei definir a quantidade de caracteres de números como <=11 ou char[:11] e essas formas não deram certo, a explicação que encontrei não foi oficial e também não consta na documentação oficial do tkinter, o número 1 é um tipo de inicializador e é tem a obrigação de ser informado, a sequência de zeros é a quantidade de números que será permitido a entrada. Após a criação da classe Validadores criei a **def checkEntrys** nesta função criei as variáveis que serão vinculadas as entrys tanto de cadastro de cliente quanto para o cadastro de livros, para a configuração é necessário informar qual janela a checagem ira desempenhar sua função através do **self.janela.register** e entre parênteses registro a def criada para cada entry dentro da classe Validadores, que ficara da seguinte forma: **variável=(self.janela.register(self.validate\_entrycod), ‘%P’)** o nosso ‘%P’ é o nosso caractere coringa que é inserido automaticamente nas entrys.  
  
**6.** Explicação da Criação da nossa Primeira Treeview:

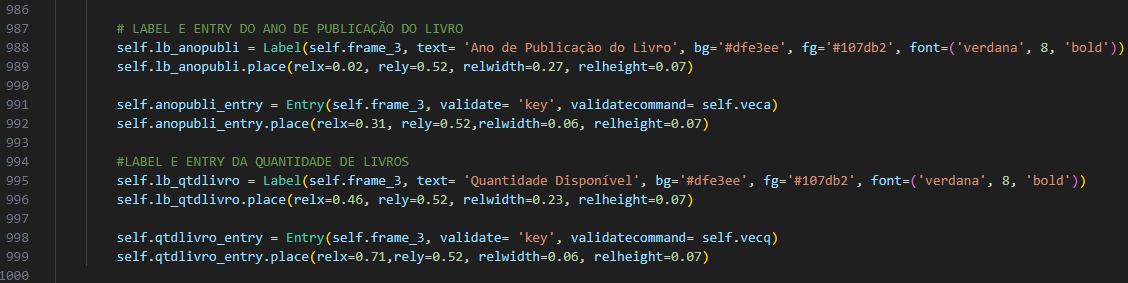
Para a criação da Treeview ou apenas lista foi necessário informar o frame que ela será inserida, o tamanho dela **height = 3** e definir as colunas pelo método **column=(‘col1’.....)**, após isto comecei chamando nossa treeview de listaCliente já que ela ira armazenar nossos dados de clientes, a configuração da coluna tem o parâmetro obrigatório **#0, text=’’** este parâmetro ele define um espaço da margem esquerda, após informar o parâmetro obrigatório foi so configurar a coluna através do método **heading**. Após definir o heading, comecei a definir a largura de cada coluna através do método **column** que também tem por obrigação a passagem do parâmetro **#0, width=1**, após passar o parâmetro obrigatório, executei a configuração da largura das colunas que realmente nos interessa, após essa tarefa foi só definir o local onde o nosso heading iria se acomodar para isto usei o método **.place** que dá uma liberdade muito grande para o posicionamento de elementos na tela. Após definir o local inseri ao lado da nossa lista um scroll de rolagem, informando o frame onde ira ficar, sua orientação se é vertical ou horizontal e informar o seu posicionamento no frame através do .place, após configurar o scroll de rolagem, inseri um evento bind chamado de clique\_duplo\_cli esta é uma def criada para quando fazermos o duplo clique no cliente ou livro, as informações sobem automaticamente para as entrys, podendo assim fazer alteração de cadastro ou exclusão.

Em breve irei entrar em mais detalhes sobre o evento de clique duplo quando for detalhar a classe em que esta função está inserida.

**7.** Explicação de widgets\_frame\_3:

Nesta função basicamente teremos o mesmo conteúdo da função widgets\_frame1, contudo este frame irá receber as informações de cadastro dos nossos livros, as configurações de botões seguem o padrão dos botões relacionados ao cadastro de usuários, mantendo assim um visual mais limpo e harmônico. Nesta função também recebem os validadores nas entrys impossibilitando digitar números nos campos de texto e vice-versa.



Continuação da def.  


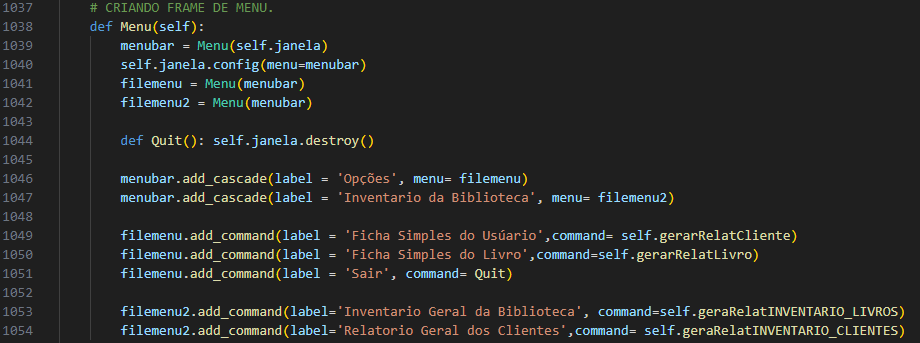
**8.** Explicação lista\_livros\_frame\_4:

Nesta função será configurado a nossa segunda Treeview, esta que será responsável por receber e mostras as informações do livro cadastrado, assim como ocorreu na criação da primeira treeview, temos dois que informar os valores obrigatórios para o heading e column, nesta treeview chamei de listaLivros, também inseri um scroll bar ao lado da nossa lista informando sua orientação e o seu posicionamento dentro do frame, temos aqui presente o outro evento clique\_duplo (em breve irei detalhar o motivo de criar dois eventos diferentes)

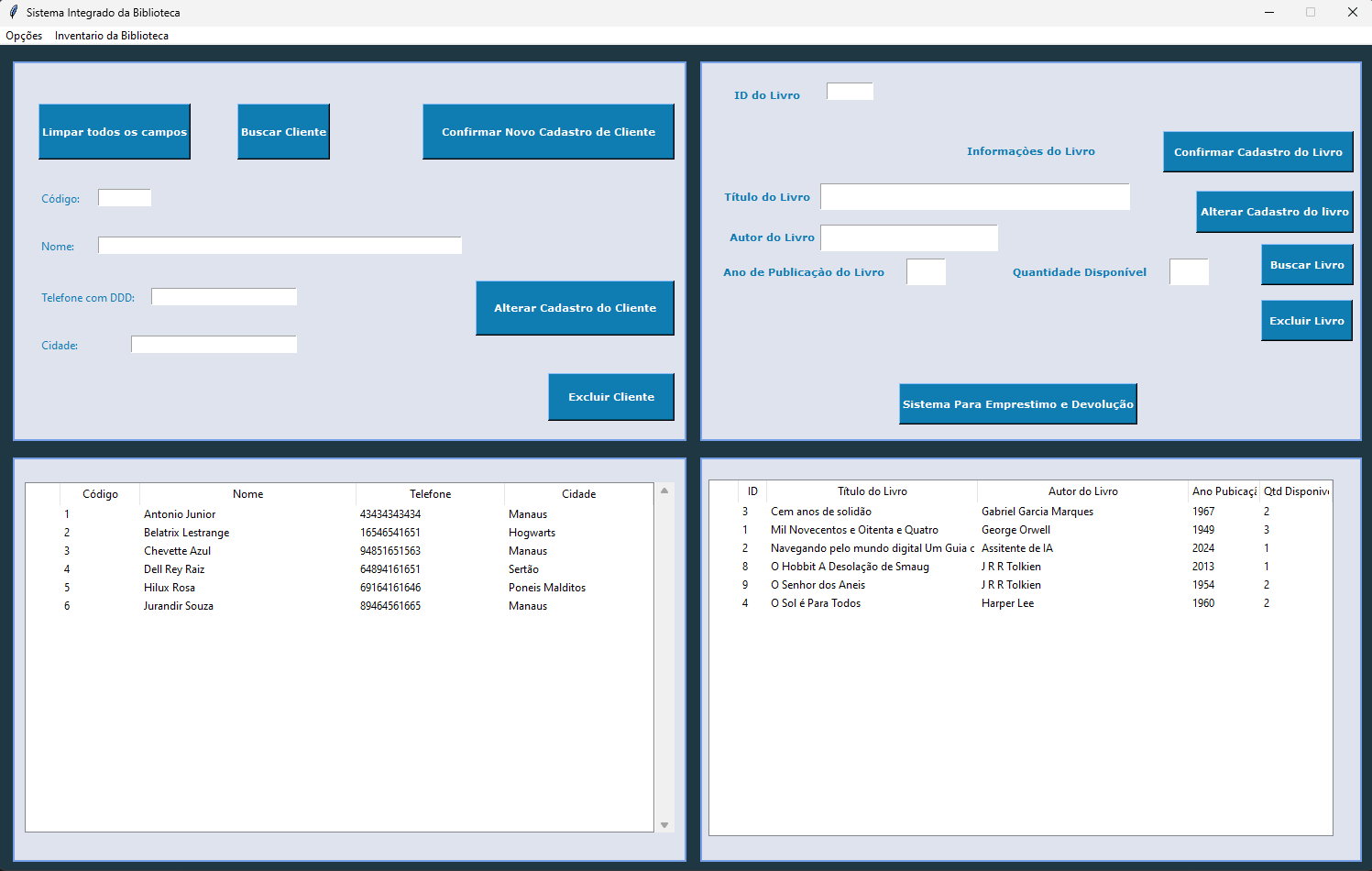


**9.** Explicação sobre a criação do menubar:

Esta função será a responsável pela nossa barra de menus na parte superior da janela, dentro do menu teremos as opções de gerar um relatório simples de um usuário especifico em .pdf dando a liberdade de imprimir e ter uma documentação física para arquivo. Neste menubar também haverá outros três geradores de relatório, Relatório simples de um livro, e Inventario da Biblioteca que irá apresentar um pdf com todos os livros cadastrados e por ultimo o Inventario de clientes onde também ira mostrar todos os clientes cadastrados.

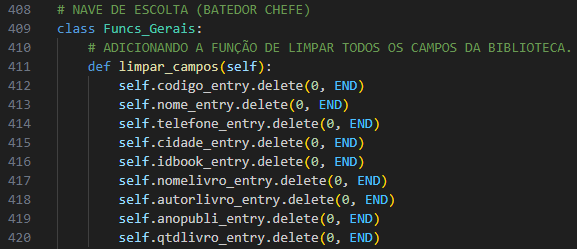


Após descrever todas as defs contidas na classe principal abaixo será demostrado como é a janela principal de cadastro de clientes e cadastro de livros.

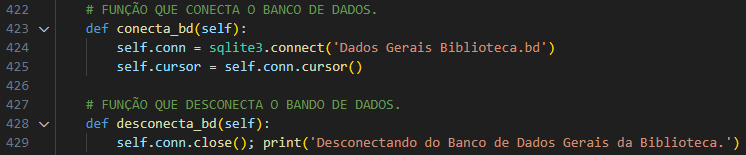


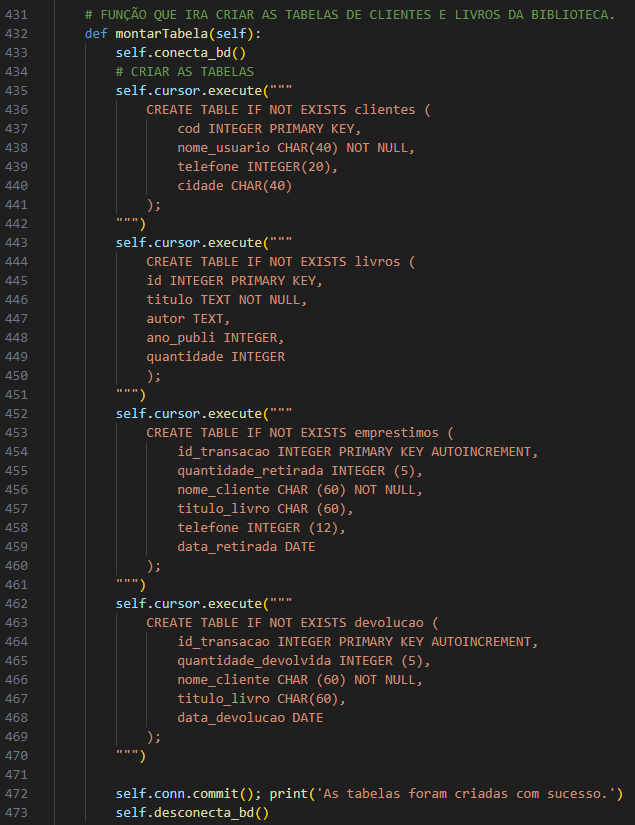
Após descrever a classe principal App\_Biblioteca e a classe de Validadores, agora será descrito a classe **Funcs\_Gerais** esta classe é aonde está comportada todas as funções para o pleno funcionamento da classe principal, entre as funções é a de clique duplo, de adicionar livros e clientes, exclusão de livros de clientes.

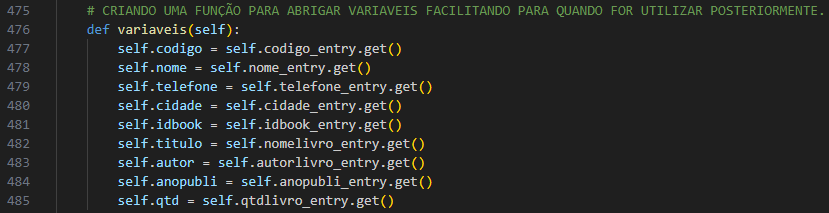
**10.** Classe Funcs\_Gerais**:**A primeira função da classe é de limpar\_campos, esta função limpara todas as entrys que estão no cadastro de clientes e no cadastro de livro, esta função foi uma das mais simples para ser escrita, já que apenas leva o nome das entrys seguida de delete, um exemplo é **self.codigo\_entry.delete(0, END)** esta simples linha já nos garante que a entry responsável pelo campo código terá seu conteúdo apagado.



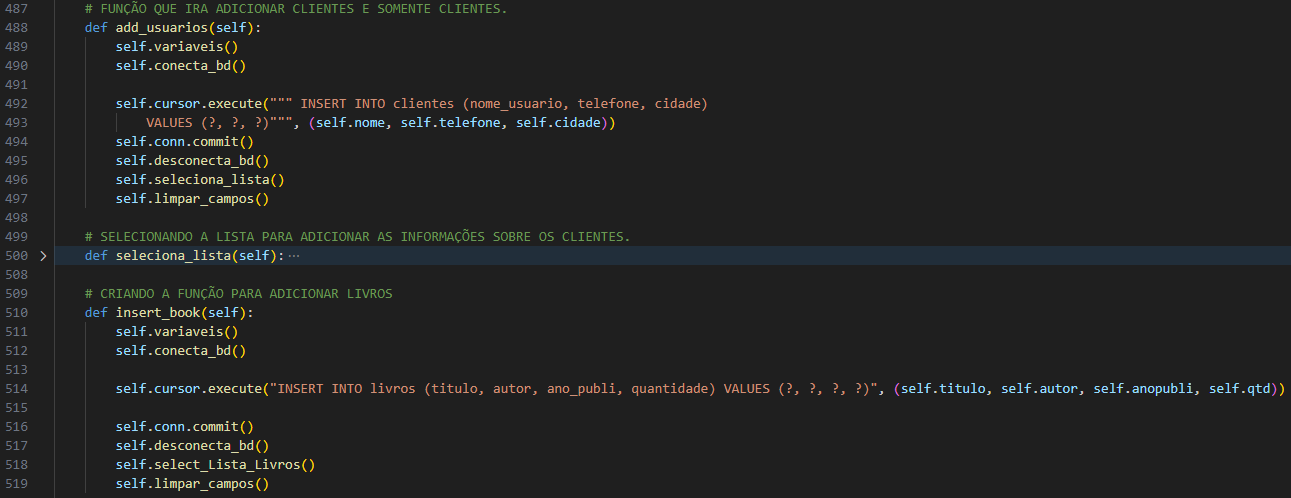
Após criar a função de limpar\_campos, chegou o momento de criarmos o nosso banco de dados utilizando a linguagem SQL e a biblioteca do sqlite3, as próximas funções será sobre criar o banco de dados e caso já tenho o banco de dados a função não será executada, criaremos também a função por fazer a desconexão com o banco de dados e também sera apresentado a função responsavel pela criação das nossas 4 tabelas, que serão, clientes, livros, empréstimo e ‘devolucao’.

  
Aqui é aonde se cria o nosso banco de dados com o nome de Dados Gerais Biblioteca, em seguida já foi criada a desconexão com o banco de dados, tanto a função de conexão tanto a de desconexão tem um print para saber se o programa está inicializando sem erros.

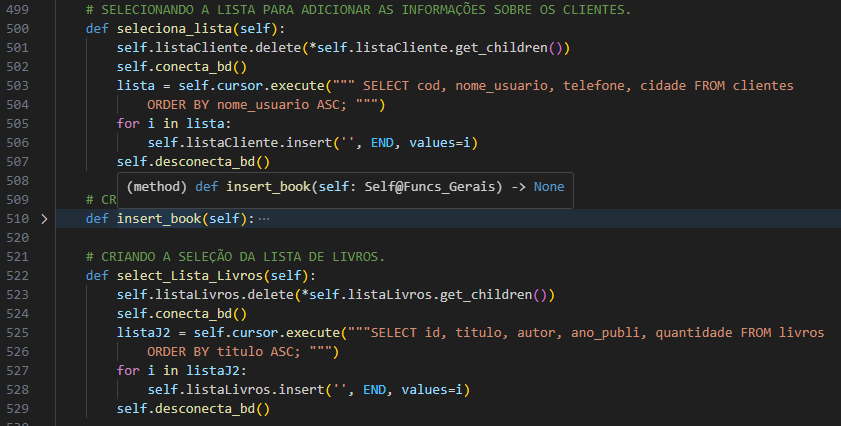


Aqui está a função montar\_Tabela, nesta função esta concentrado a criação das nossas 4 tabelas dentro no nosso banco de dados, por boas praticas para manutenção do código e revisão concentrar a criação das tabelas nesta função foi uma decisão assertiva modéstia parte falando. Perceba que logo após criar a def já chamo a nossa função de conexão com o banco de dados e na ultima linha esta a nossa função de desconexão.  
  


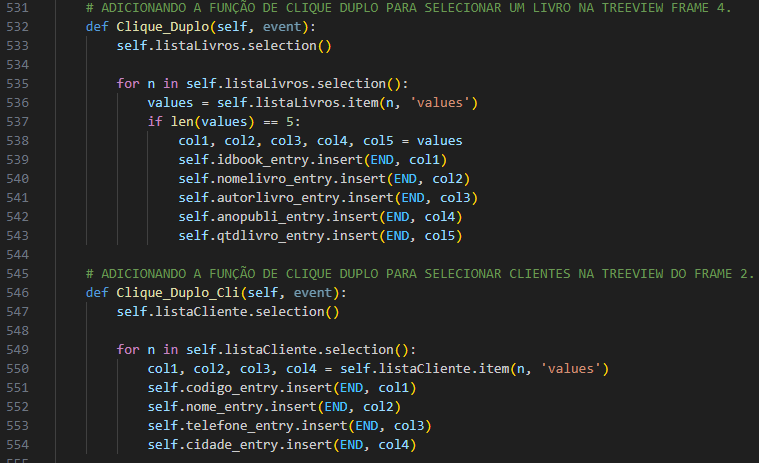
A def variáveis será muito útil mais pra frente no código, principalmente quando for criada as funções de adicionar clientes ou livros, nesta função concentrei todas as variáveis entrys com o **.get**.

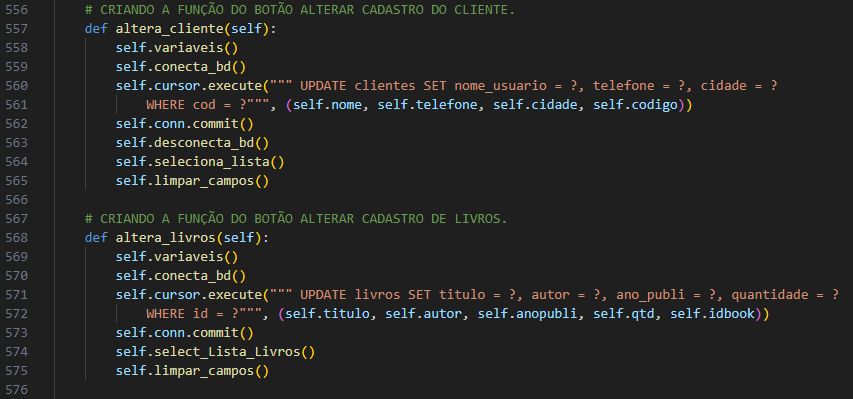


Neste momento irei explicar as duas funções de inserção de dados que é adicionar cliente e adicionar livro, estas funções são de todas as formas gêmeas se alterando apenas a lista que será feita a adição de dados. Após a criação das defs já chamo a def variáveis facilitando a escrita e a compreensão do código, em seguida já chamo a nossa conexão com o banco de dados seguidos pelo comando SQL de INSERT na tabela clientes nas colunas de nome\_usuario, telefone e cidade com os valores de indefinidos e insiro as variáveis das entrys que também foram definidas na função variáveis. Em seguida salvo as alterações com o commit() faço a desconexão com o banco de dados, faço a seleção da lista(Treeview) onde será adicionada as informações, e por último a função de limpar todos os campos deixando tudo pronto para um novo cadastro ou consulta.

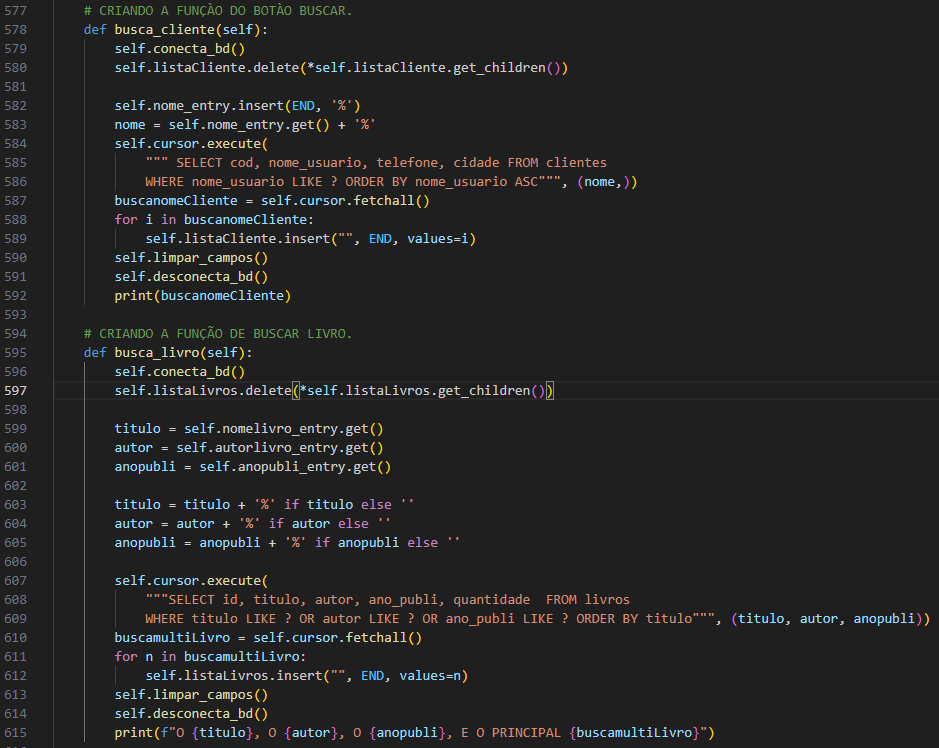


Para que a adição de informações seja possível tem que selecionar um lista onde será armazenado os dados e onde será exibido os dados contidos nas tabelas, assim como as funções de adição de informações são gêmeas estas defs também são gêmeas alterando apenas as listas onde será feita armazenada a informação e em qual tabela do nosso banco de dados terá alteração.

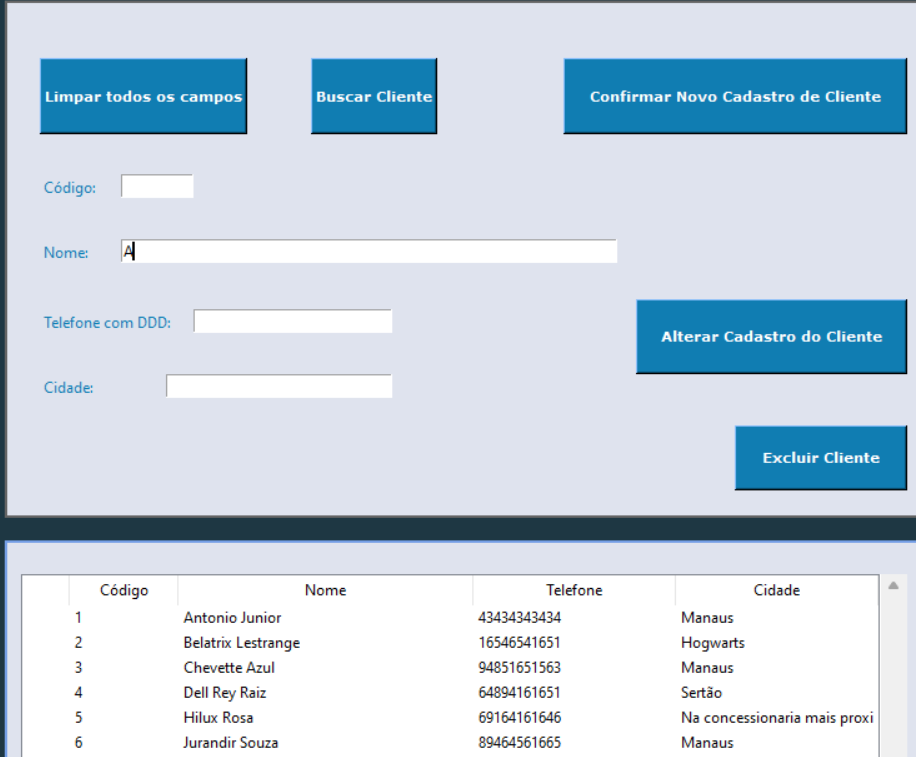
  
Chegou o momento de falarmos um pouco das defs de clique duplo. A configuração é parecida com a pequena adição do **if len(values)==5**, que de alguma forma o programa dava erro que faltava um argumento a ser passado, no mais a configuração é bem parecida. A função de clique duplo é a responsável por conseguirmos selecionar um livro ou um cliente, para fazermos alteração em suas informações ou a exclusão do cadastro.

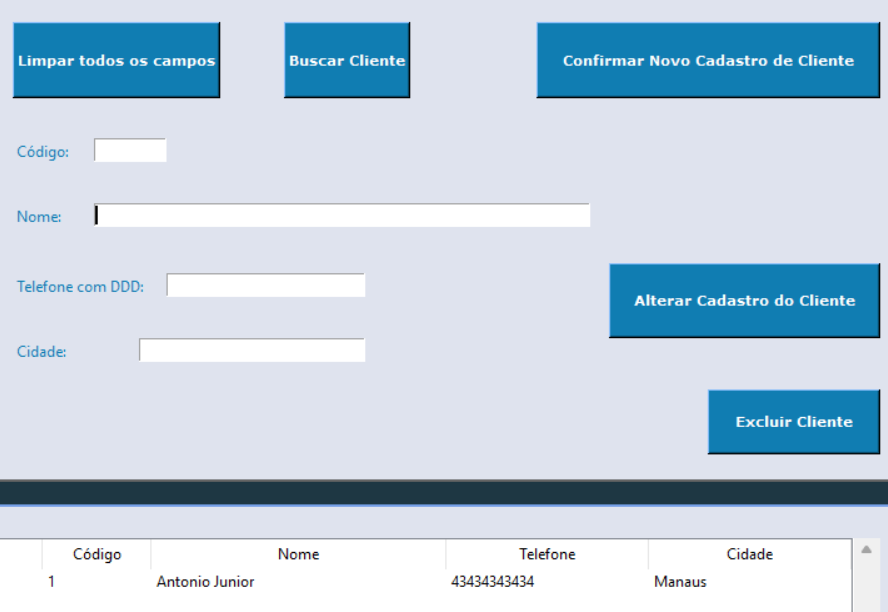


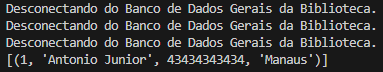
Esta é a função responsável por fazer a alteração no cadastro de cliente e livros, perceba que assim que crio a def, já chamo a função variáveis e em seguida já crio a conexão com o banco de dados, e faço o comando SQL solicitando update nas tabelas de clientes e livros, após isso salvo os dados com o commit(), faço a seleção de lista onde também terá as informações salvas e limpo todos os campos.

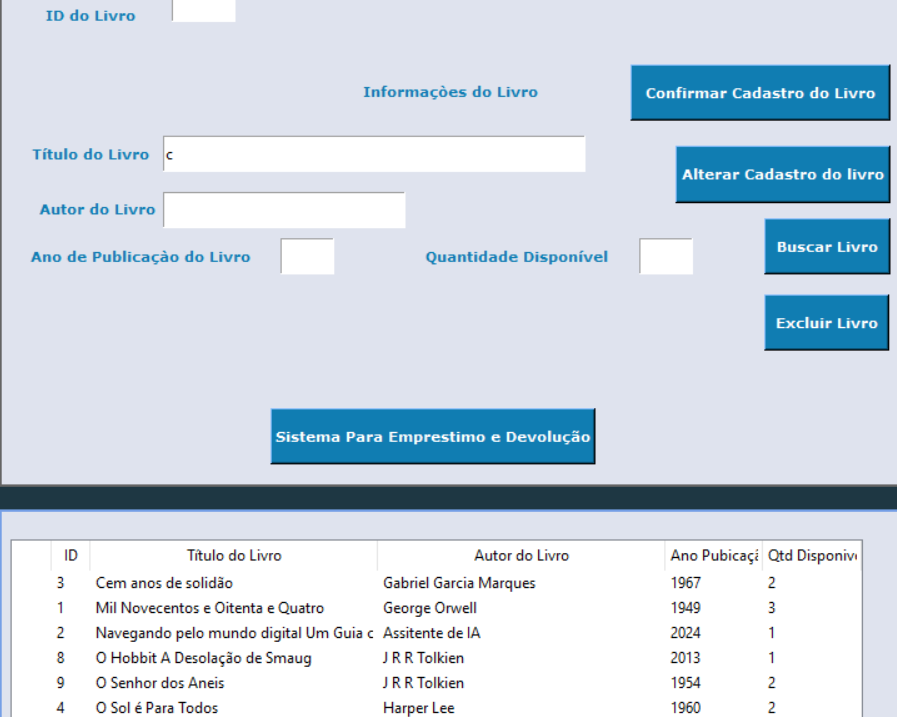


Nesta parte do codigo já defino as funções que irei vincular aos botões de buscar cliente e buscar livro. Para configurar a busca por cliente foi necessário fazer a conexão com o banco de dados e selecionar diretamente a listaCliente e fazer uma limpeza dessa lista antes de iniciar uma busca ou uma nova busca, a preparação da entrada com a linha **self.nome\_entry.insert(END, ‘%’)**, insere o caractere coringa ‘%’ ao final do conteúdo atual do campo de entrada, este caractere coringa é usado em SQL para as buscas que utilizam **LIKE**, significando que qualquer texto atual da entry nome já incluindo o ‘%’, ainda é adicionado outro ’%’ ao final para as buscas SQL. O comando execute fara a busca pelo nome ou pela inicial do nome o ‘?’ é substituído pela variável nome no final da consulta SQL, após isso o fetchall recupera todos os registros que satisfazem a pesquisa, a iteração for i in buscanomeCliente, fara a iteração para todos os nomes e em seguida o listaCliente.insert adiciona o resultado na nossa treeview, após isso foi preciso solicitar a função de limpar campos, desconexão com o banco de dados e solicito um print para que eu possa acompanhar também pela output do VSCode o funcionamento do código. Veja um exemplo abaixo onde apenas insiro a letra A, o resultado esperado é Antonio Junior.

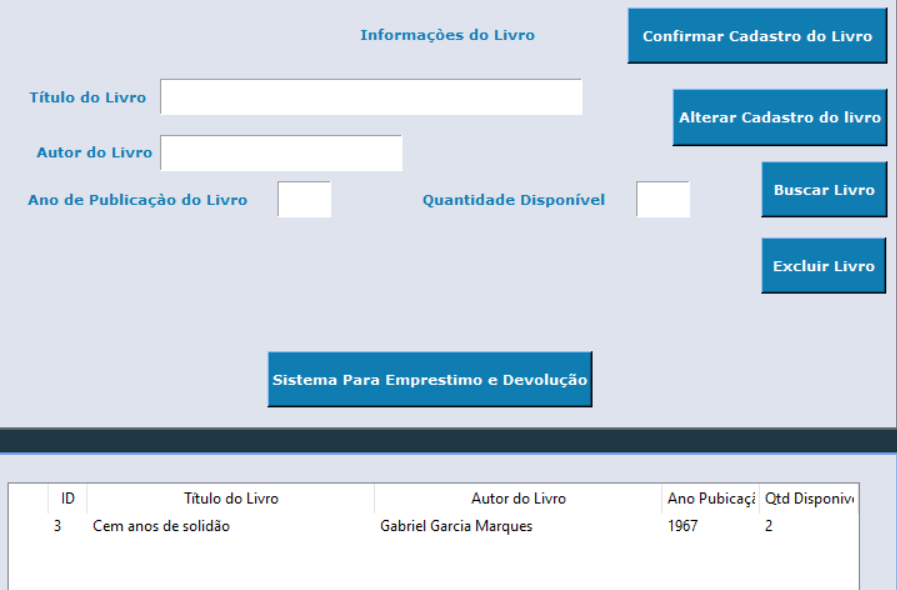


Após clicar em buscar cliente, o resultado foi como o esperado, veja abaixo:  


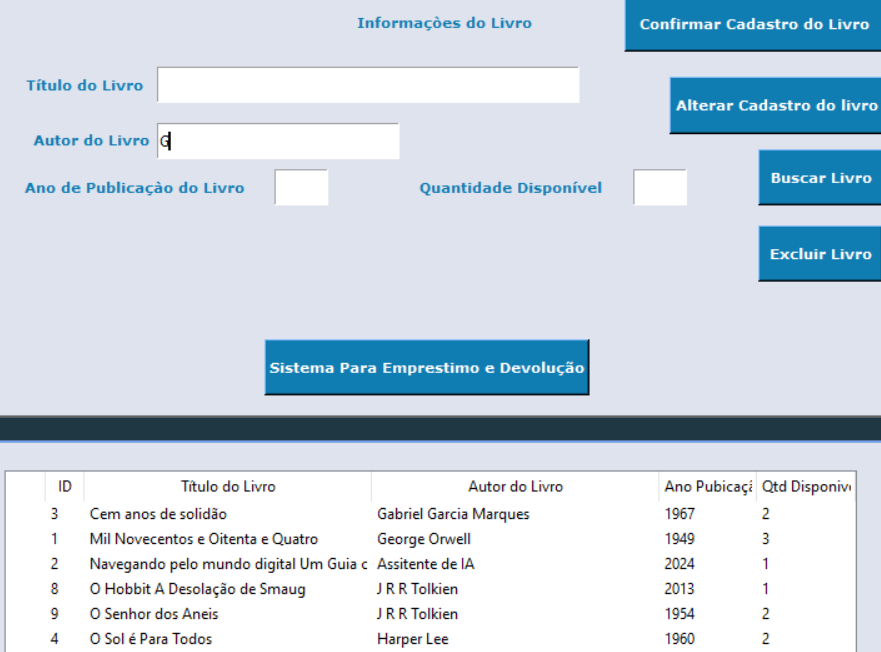
Veja que o print solicitado também nos retorna o cliente Antonio.  
  
  
Agora em especial irei detalhar a busca\_livro, esta função teve uma “exigência”, descrita no campo de atividades do portfólio, que a busca de livros devera ser feita pelo titulo do livro, nome do autor ou pelo ano de publicação. Diferente da busca do nome do cliente a busca por livro nos coloca um desafio. A solução que encontrei foi definir três variáveis titulo, autor e anopubli, que estarão nas linhas 599,600 e 601 após isso chamei as três variáveis recém criadas e concatenei utilizando o sinal de ‘+’ mais o caractere coringa ‘%’, após isso faço o execute selecionando a tabela livros e substituindo ‘?’ pelas três variáveis criadas dentro da função, veja abaixo os três exemplos de busca de livros.

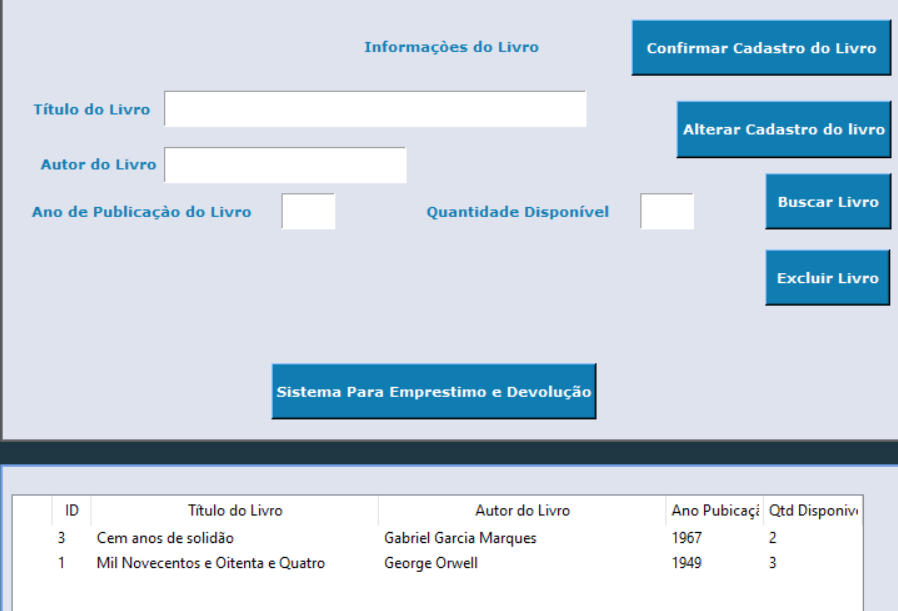


Neste exemplo irei buscar utilizando letra minúscula e pelo livro Cem anos de solidão, vejamos o resultado da busca abaixo.



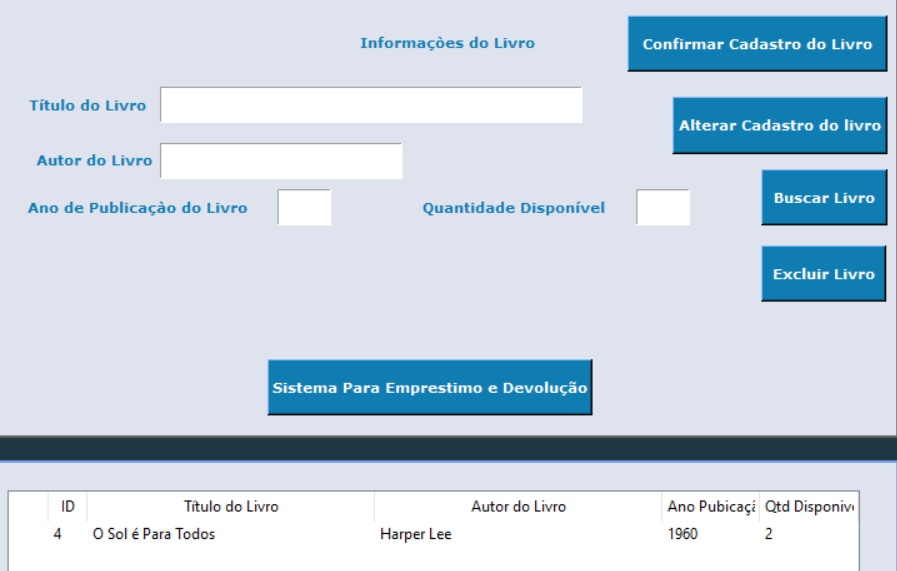
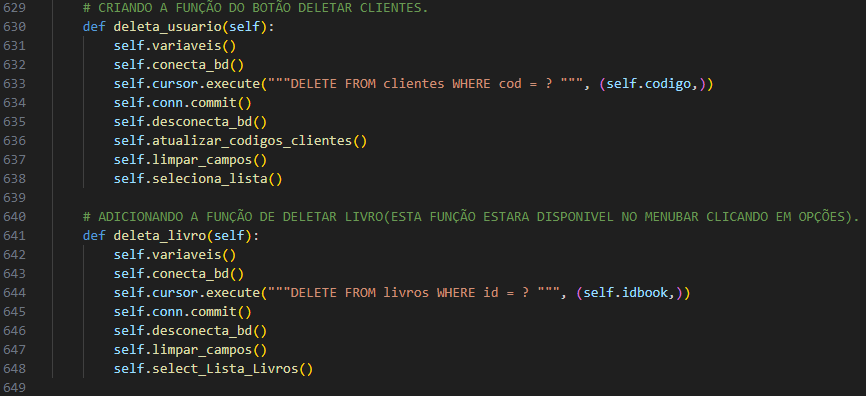
A busca nos retornou o esperado, veja também o output do print:  


Agora irei demonstrar a busca por autor:  


Farei a busca usando a letra G, onde temos dois autores Gabriel e George, vejamos o resultado da busca, seguido pelo output no terminal.  
  

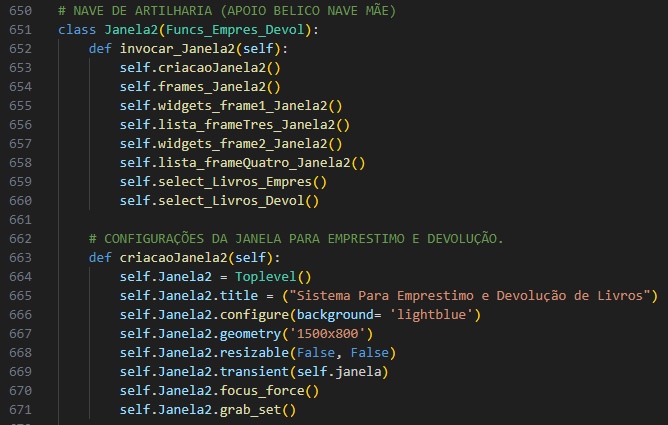

O resultado da busca foi o esperado e por ultimo será a busca pelo ano de publicação.  

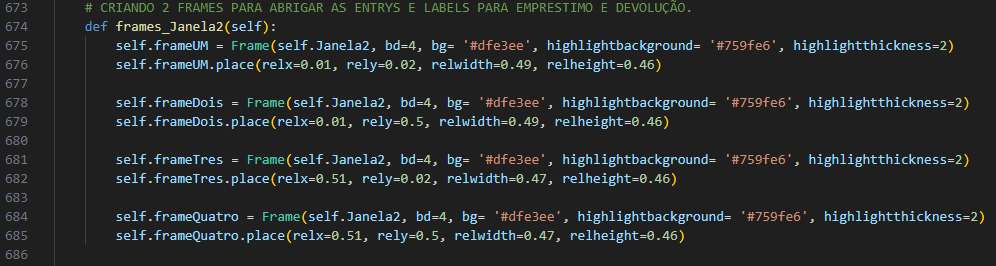

Iremos buscar pelo livro de 1960 O Sol é Para Todos de Harper Lee, vejamos o resultado da busca e o resultado do output no terminal.

  
  
O resultado mais uma vez foi satisfatório, entregando com êxito o pedido feito nas atividades propostas.  
  
Dando sequencia teremos nossas duas ultimas funções que será de excluir cliente e excluir livro.  


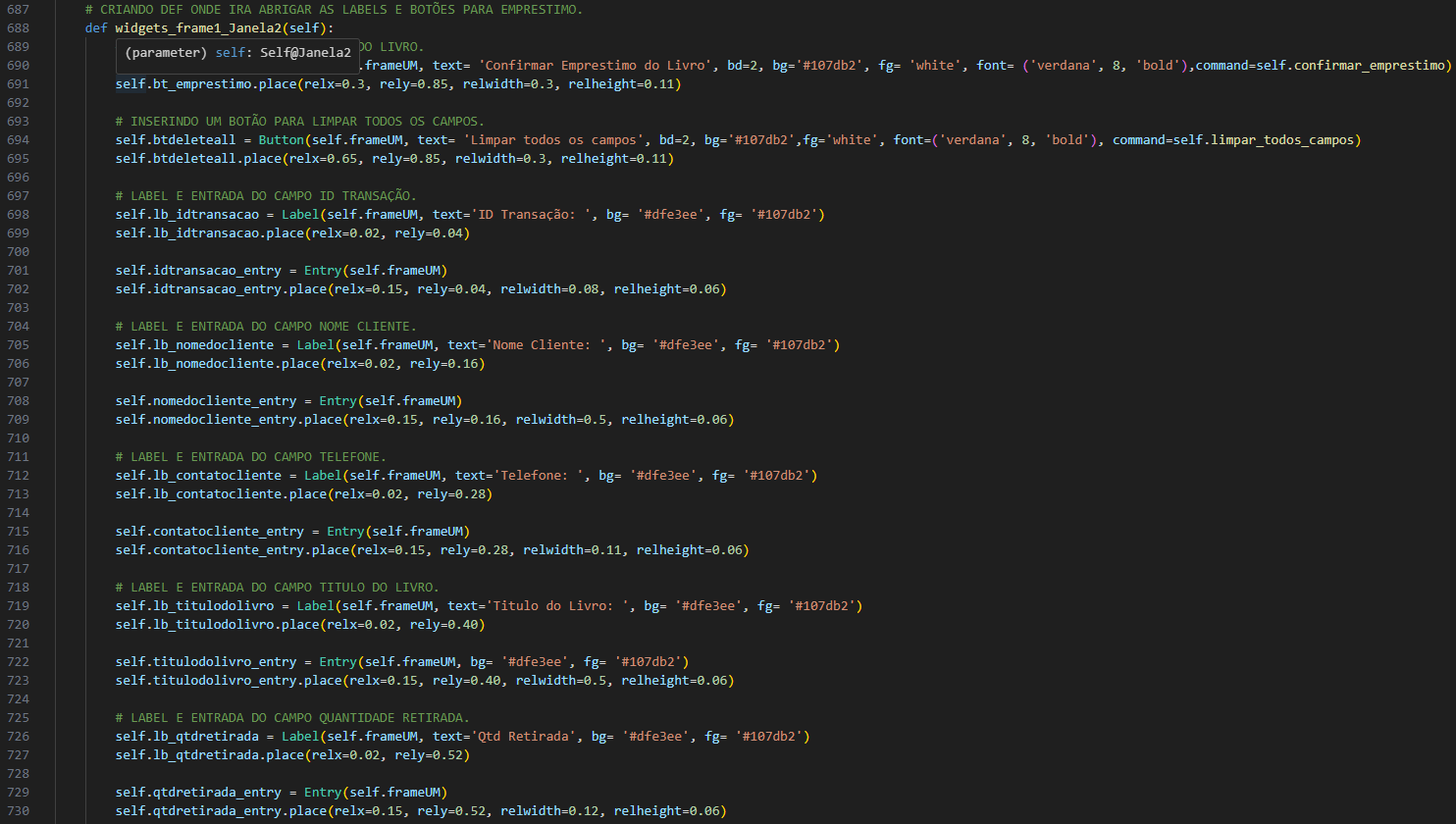
As duas funções têm sua configuração basicamente iguais, alterando apenas a tabela que será feita a ação de deletar. Com isso encerro a explicação da terceira classe.

**12.** Classe responsável pela Janela 2.  
Esta classe abriga toda a configuração da segunda janela que será a responsável pelo empréstimo e devolução de livros, nesta classe contamos com a primeira def que é a responsável por invocar a janela 2 quando clicamos no botão “Sistema de Empréstimo e Devolução”, nesta def também contamos com as invocações das defs contidas dentro da classe. A segunda def criada foi criacaojanela2, nesta função coloquei todas as configurações da janela definindo ela como **toplevel()** e usando os métodos **focus\_force** e **grab\_set** estes métodos impedem o usuário do sistema de mexer na janela 1.

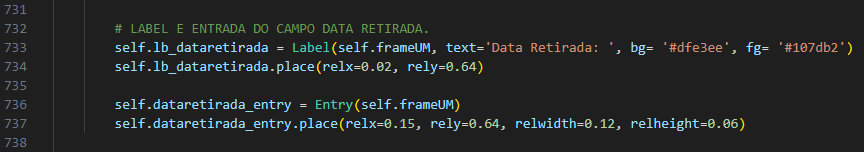


A seguir foi criada a def de criação dos frames e definição da posição deles na janela2.  


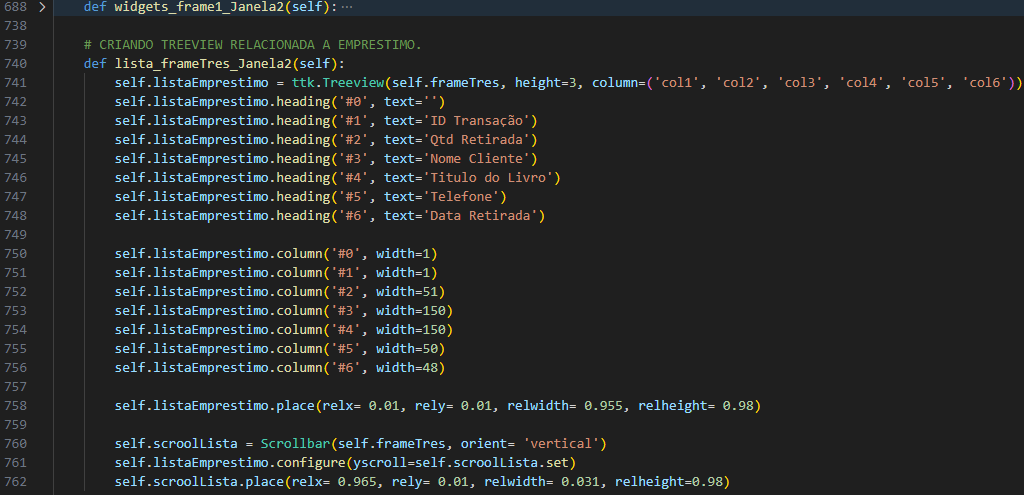
Após a criação da def frames parti para a configuração dos botões e labels do frame 1, onde abrigara os campos referentes ao empréstimo de livros, neste frame teremos apenas dois botões, “confirmar empréstimo” e “limpar todos campos”.



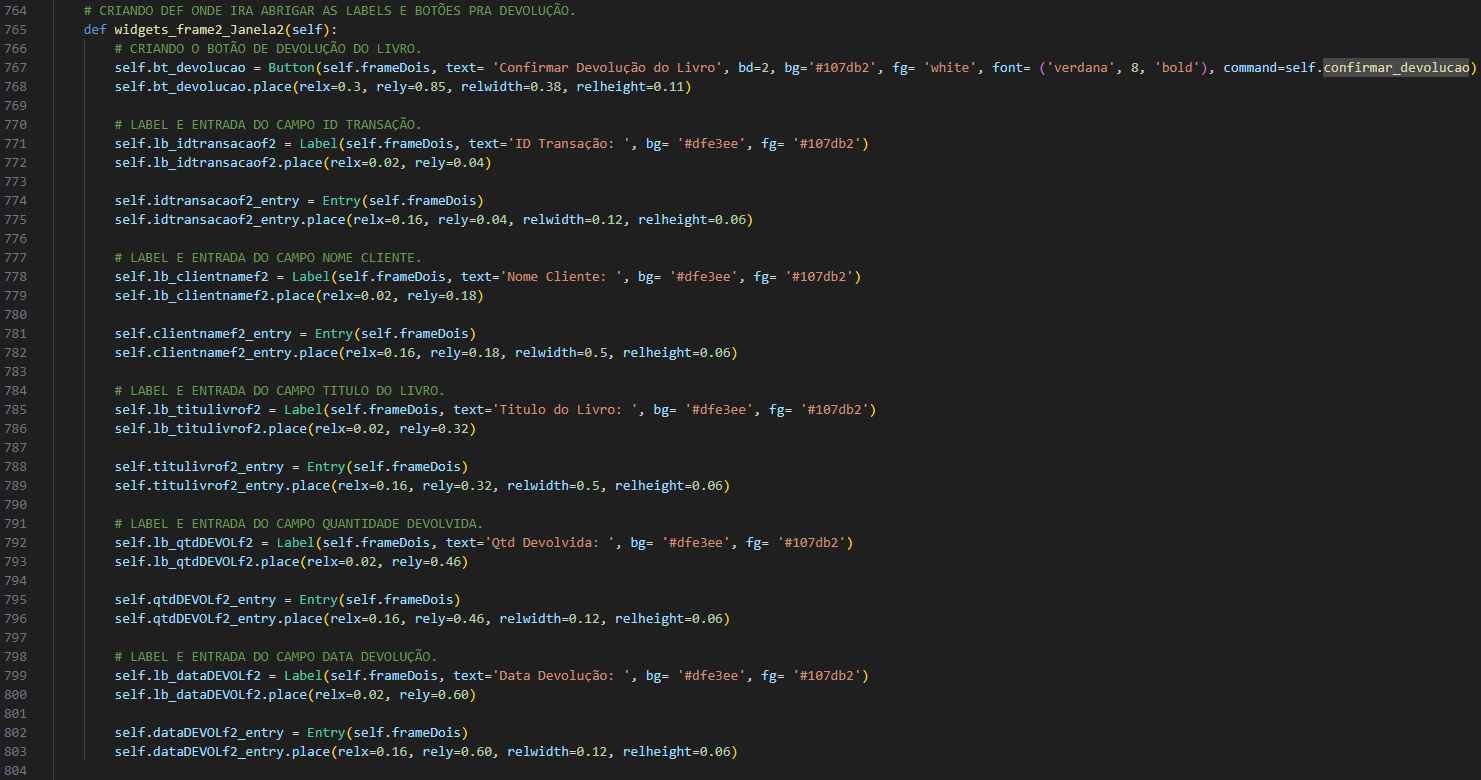
Continuação do código:



A sequencia de defs nesta janela é idêntica as configurações da janela 1, onde criei a treeview relacionada há empréstimos de livros chamada de lista\_framTres\_Janela2, a próxima def criada foi a de widgets relacionada a devolução de livros onde inseri apenas um botão que é “confirmar\_devolucao” e a criação e posicionamento das labels e entrys dentro do frame e por ultimo foi adicionado a treeview relacionada a devolução.



Continuação do código:



Continuação do código:  
  
  
Estas funções são semelhantes as funções contidas na janela 1 por este motivo estou apenas recapitulando algumas particularidades contidas nas funções.

**13.** Classe Funcs\_Empres\_Devol:

Esta classe concentra todas as funções responsáveis pelo funcionamento da janela2, apesar de concentrar todas as funções muitas das funções são gêmeas das funções contidas na classe Funcs\_Gerais, entre as funções gêmeas estão **limpar\_todos\_campos** esta função como sua ‘Irma gêmea’ também limpara todas as entrys do campo empréstimo e devolução, **conecta\_bd** função responsável por criar a conexão com o nosso banco de dados, **desconecta\_bd** fara a desconexão com o banco de dados quando solicitarmos, **variáveis\_Empres\_Devol** concentra todas as variáveis entrys dos frames de empréstimos e devolução, **add\_Livros\_Empres** adiciona livros a nossa tabela empréstimos contida no nosso banco de dados, **select\_Livros\_Empres** seleciona a treeview relacionada a empréstimos para fazer a adição, **add\_Livros\_Devol** adiciona as informações de devolução a nossa tabela de devolução, **select\_Livros\_Devol** seleciona a treeview relacionada a devolução para inserir informações.  
Temos a def **confirmar\_emprestimo** que é necessário explica-la com mais calma diferente das anteriores, esta função foi difícil acertar a configuração pela complexidade que foi além do esperado e por acontecer uma infinidade de erros que tanto por confirmar empréstimos aparecendo na treeview e na tabela SQL não constar nada ate o fechamento da interface gráfica.

**13.1 .** Explicação Detalhadada função confirmar\_emprestimo:

A função **confirmar\_emprestimo**: lida com a verificação da disponibilidade do livro, a atualização do estoque e o registro do empréstimo na base de dados. Ela utiliza uma abordagem de tratamento de erros robusta para garantir que todas as operações sejam feitas automaticamente.

Sequência de passos da Função  
*1. Conexão com o Banco de Dados*:  
self.conecta\_bd(): Este método estabelece a conexão com o banco de dados.

*2. Inicialização das Variáveis Necessárias*:  
self.variaveis\_Empres\_Devol(): Inicializa as variáveis necessárias para o processo de empréstimo e devolução.

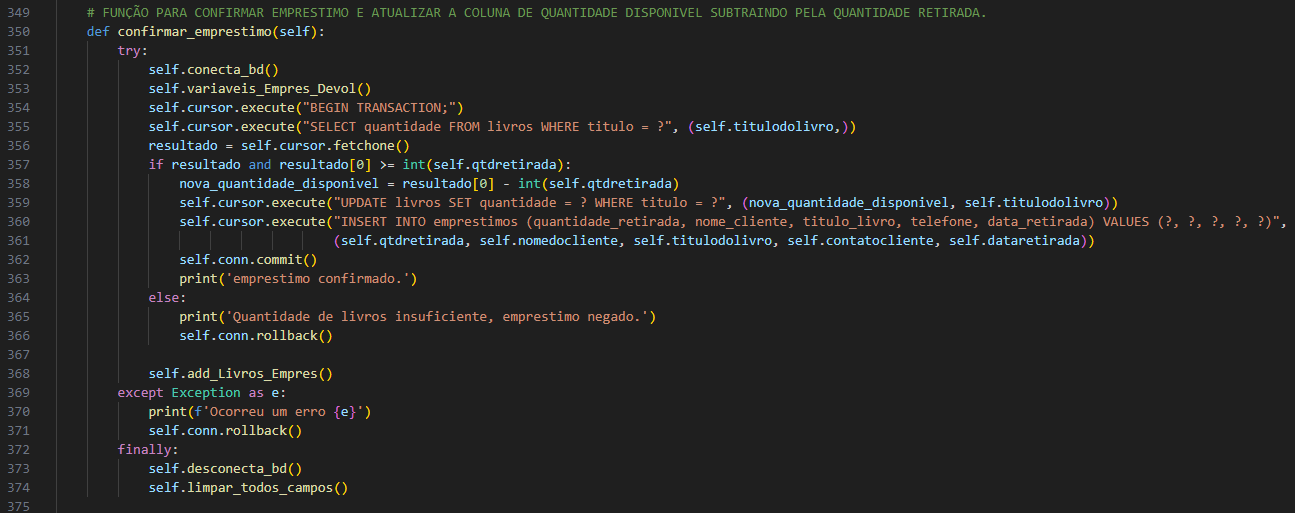
1. **Início da Transação:**
   * **self.cursor.execute("BEGIN TRANSACTION;")**: Inicia uma transação SQL para garantir que todas as operações subsequentes sejam parte de uma única transação atômica.
2. **Consulta de Disponibilidade:**
   * **Consulta SQL**: A função consulta a quantidade disponível do livro especificado.
   * **Resultado**: O resultado é buscado e verificado. Se a quantidade disponível do livro é suficiente para o empréstimo **(resultado[0] >= int(self.qtdretirada))**, a operação prossegue.
3. **Atualização do Estoque:**
   * **Cálculo da Nova Quantidade**: A nova quantidade disponível é calculada subtraindo a quantidade retirada da quantidade atual.
   * **Atualização no Banco de Dados**: A quantidade do livro no banco de dados é atualizada para refletir a nova quantidade disponível.
4. **Registro do Empréstimo:**
   * **Inserção na Tabela de Empréstimos**: Um novo registro é adicionado na tabela de empréstimos com os detalhes do cliente e do empréstimo.
5. **Confirmação ou Rejeição do Empréstimo:**
   * **Commit**: Se todas as operações forem bem-sucedidas, a transação é confirmada (commit).
   * **Rollback**: Se a quantidade de livros for insuficiente, a transação é desfeita (rollback), e uma mensagem de erro é mostrada.
   * **Tratamento de Exceções**: Qualquer erro durante o processo resulta em um rollback da transação.
6. **Encerramento da Conexão:**
   * **self.desconecta\_bd()**: Encerra a conexão com o banco de dados.
7. **Limpeza dos Campos:**

* **self.limpar\_todos\_campos()**: Limpa os campos de entrada na interface gráfica para preparar para novas entradas.

**Tratamento de Exceções:** A função lida com exceções de forma eficaz para evitar que erros inesperados causem problemas de dados inconsistentes.

**Mensagens de Feedback:** Fornece feedback adequado ao usuário através de mensagens de confirmação ou erro, dependendo do resultado das operações.

Esta função é crucial para garantir a integridade das operações de empréstimo de livros em uma biblioteca, garantindo que as mudanças no estoque e os registros de empréstimos sejam feitos de forma confiável e segura.

E como falei no inicio esta função foi ofereceu inúmeros desafios e erros ao longo do código, porem depois do esforço codando o que foi descrito acima a função ainda assim não funcionava direito, foi necessário fazer mais uma nova função dentro da nossa classe principal a App\_Biblioteca e fazer uma pequena adaptação técnica no init da classe principal.  


**Descrição Geral**

A função **confirmar\_emprestimo\_gui** serve como uma camada intermediária entre a interface gráfica do usuário (GUI) e a lógica de negócios para confirmar empréstimos de livros. Ela é responsável por preparar e passar dados de entrada coletados pela GUI para a função de lógica de negócios **confirmar\_emprestimo**.

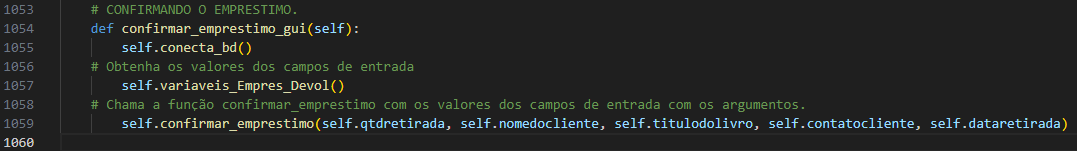
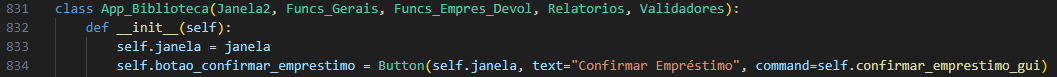
**Passos Executados pela Função**

1. **Conexão com o Banco de Dados:**
   * **self.conecta\_bd()**: Este método é chamado para estabelecer uma conexão com o banco de dados, assegurando que todas as operações subsequentes tenham acesso ao banco.
2. **Coleta de Dados de Entrada:**
   * **self.variaveis\_Empres\_Devol()**: Este método é responsável por coletar e preparar as variáveis que serão usadas na transação de empréstimo. Isso inclui obter valores de campos de entrada como a quantidade de livros retirada, nome do cliente, título do livro, contato do cliente e a data da retirada.
3. **Chamada da Função de Emprestimo:**
   * **self.confirmar\_emprestimo(...)**: Após a coleta de dados, esta função é chamada com os argumentos necessários para processar o empréstimo. Os argumentos passados são aqueles coletados no passo anterior: **self.qtdretirada**, **self.nomedocliente**, **self.titulodolivro**, **self.contatocliente**, e **self.dataretirada**. Esta função efetivamente executa as operações de verificação de disponibilidade, atualização de estoque e registro de empréstimo no sistema.

**Finalidade e Importância**

* **Integração GUI e Lógica de Negócios**: **confirmar\_emprestimo\_gui** funciona como um facilitador para integrar a interface gráfica com a lógica de negócios do sistema de empréstimos, garantindo que a interação do usuário seja processada corretamente no backend.
* **Abstração e Organização**: Ao manter a lógica de conexão com o banco e a coleta de dados separadas da execução específica do empréstimo, o código fica mais organizado, modular e fácil de manter.

Este método é essencial para a eficiência operacional do sistema de empréstimos da biblioteca, assegurando que todas as etapas do processo de empréstimo sejam corretamente executadas e que os dados do usuário sejam manuseados de forma segura e eficaz.

Após fazer a nova def e colocar esta adaptação dentro do \_\_init\_\_ da nossa classe principal a função emprestar livro funcionou muito bem e não apresentou erros.

A última def dessa classe é confirmar\_devolucao e também será preciso fazer um detalhamento para esclarecer seu funcionamento.

**13.2.** Detalhando a def confirmar\_devolucao:

**Descrição Geral**

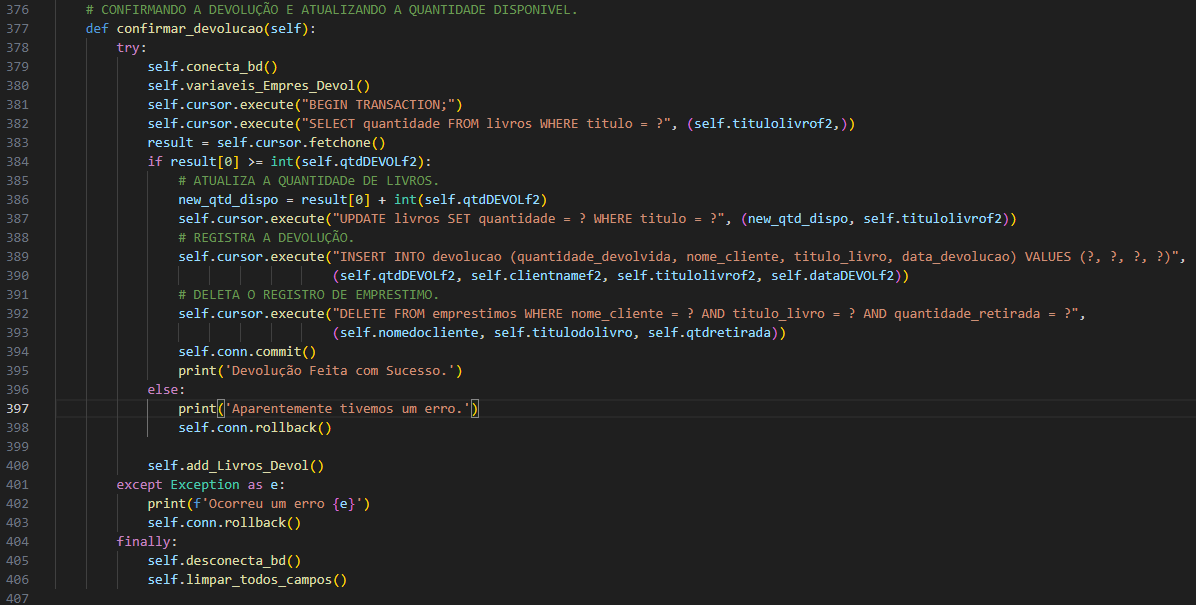
Esta função realiza várias operações críticas relacionadas à devolução de livros, incluindo a atualização do inventário de livros, o registro da devolução em uma tabela específica e a exclusão do registro de empréstimo correspondente. A função também utiliza transações para garantir a integridade dos dados durante essas operações.

**Etapas Detalhadas da Função**

1. **Conexão com o Banco de Dados:**
   * **self.conecta\_bd()**: Estabelece uma conexão com o banco de dados.
2. **Inicialização de Variáveis:**
   * **self.variaveis\_Empres\_Devol()**: Inicializa variáveis relevantes para a operação de devolução.
3. **Início da Transação:**
   * **self.cursor.execute("BEGIN TRANSACTION;")**: Inicia uma transação para garantir que todas as operações subsequentes sejam atômicas.
4. **Verificação da Quantidade Disponível:**
   * Consulta a quantidade atual do livro especificado.
   * **result = self.cursor.fetchone()**: Busca o resultado da consulta.
5. **Atualização do Estoque de Livros:**
   * Verifica se a quantidade no registro é suficiente para a devolução.
   * Atualiza a quantidade disponível de livros no banco de dados se a condição for satisfeita.
6. **Registro da Devolução:**
   * Insere um registro na tabela de devolução com detalhes sobre a devolução.
7. **Exclusão do Registro de Empréstimo:**
   * Deleta o registro de empréstimo correspondente para limpar o registro que não é mais necessário.
8. **Commit ou Rollback:**
   * **self.conn.commit()**: Confirma todas as mudanças se todas as operações forem bem-sucedidas.
   * **self.conn.rollback()**: Reverte todas as operações se houver alguma falha ou se a condição para a devolução não for atendida.
9. **Tratamento de Exceções:**
   * Captura e imprime exceções que possam ocorrer durante as operações de banco de dados, e faz rollback para manter a consistência dos dados.
10. **Encerramento da Conexão e Limpeza:**
    * **self.desconecta\_bd()**: Encerra a conexão com o banco de dados.
    * **self.limpar\_todos\_campos()**: Limpa todos os campos de entrada para novas operações.

**Finalidade e Importância**

Esta função é essencial para manter a integridade dos registros de livros e empréstimos da biblioteca, permitindo que a biblioteca atualize seu inventário e registros de maneira precisa e confiável após a devolução de livros. A utilização de transações garante que todas as operações sejam concluídas com sucesso ou que nenhuma mudança seja aplicada, evitando assim inconsistências nos dados.



**14.** Classe Relatórios:

A classe **Relatorios** é parte de uma aplicação de gerenciamento de biblioteca, projetada para gerar e exibir relatórios diversos sobre clientes e livros. Ela utiliza o módulo **canvas** do ReportLab para a criação de documentos PDF e o módulo **webbrowser** para abrir os arquivos PDF gerados. A classe é integrada com o SQLite para manipulação de dados do banco de dados. Abaixo está uma explicação detalhada dos métodos e suas funcionalidades dentro da classe:

**Métodos de Visualização de PDF**

* **printCliente()**: Abre o PDF de cliente usando **webbrowser.open('Cliente.pdf')**.
* **printLivro()**: Abre o PDF de informações detalhadas sobre um livro específico.
* **printInventario()**: Abre o PDF do inventário geral dos livros.
* **printInvent\_CLIENTES()**: Abre o PDF do inventário geral dos clientes.

**Métodos de Geração de Relatório**

* **gerarRelatCliente()**: Gera um relatório detalhado para um cliente específico. Extrai informações de campos de entrada da GUI (código, nome, telefone, cidade) e organiza essas informações em um documento PDF formatado.
* **gerarRelatLivro()**: Similar ao método de cliente, mas focado em livros. Gera um relatório para um livro específico com detalhes como código, título, autor, ano de publicação e quantidade disponível.
* **geraRelatINVENTARIO\_LIVROS()**: Gera um relatório geral de todos os livros na biblioteca. Utiliza dados do banco de dados, extraídos por uma consulta SQL, para listar todos os livros com seus detalhes em um PDF.
* **geraRelatINVENTARIO\_CLIENTES()**: Funciona de maneira similar ao inventário de livros, mas para clientes. Gera um relatório detalhando todos os clientes da biblioteca, listando informações como ID, nome, telefone e cidade.

**Métodos de Conexão com o Banco de Dados**

* **callSQL()** e **call\_to\_SQL()**: São métodos utilizados para conectar-se ao banco de dados e executar consultas SQL para buscar dados necessários para os relatórios de livros e clientes, respectivamente. Após a execução das consultas, os dados são retornados e a conexão é encerrada.

**Estrutura e Layout dos Relatórios**

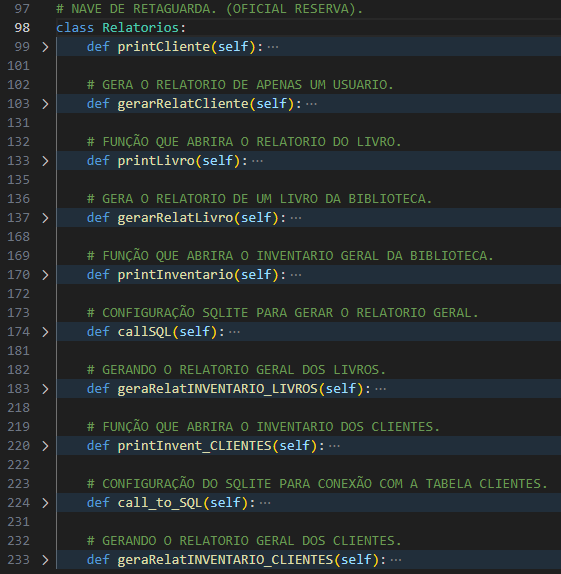
Os métodos que geram relatórios utilizam o **canvas** do ReportLab para definir o tipo e o tamanho da fonte, bem como a posição do texto no documento. Cada método configura o layout do relatório, escrevendo títulos e valores em posições específicas, e gerenciando a paginação quando necessário.

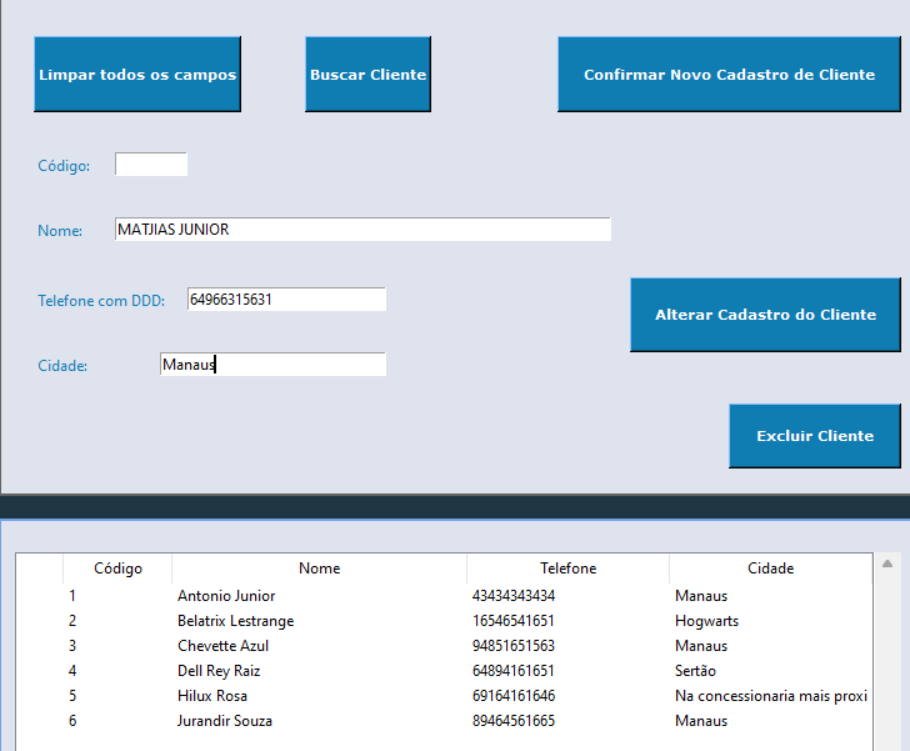
**Finalidade da Classe**

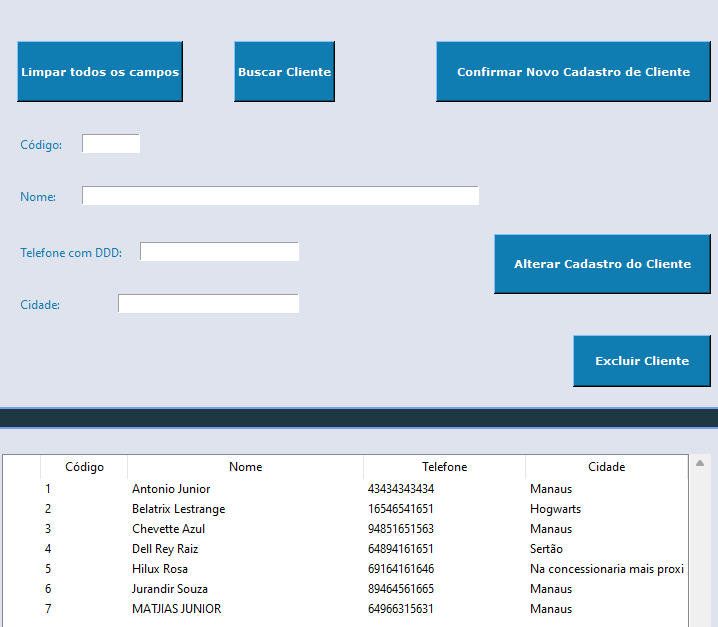
A classe **Relatorios** serve como um componente central para a geração de relatórios na aplicação de gerenciamento de biblioteca, facilitando a visibilidade e o acesso a informações estruturadas tanto para os administradores da biblioteca quanto para os clientes. A classe abstrai complexidades relacionadas à geração de PDFs e gerenciamento de dados, proporcionando uma interface clara e direta para a criação de documentos informativos.

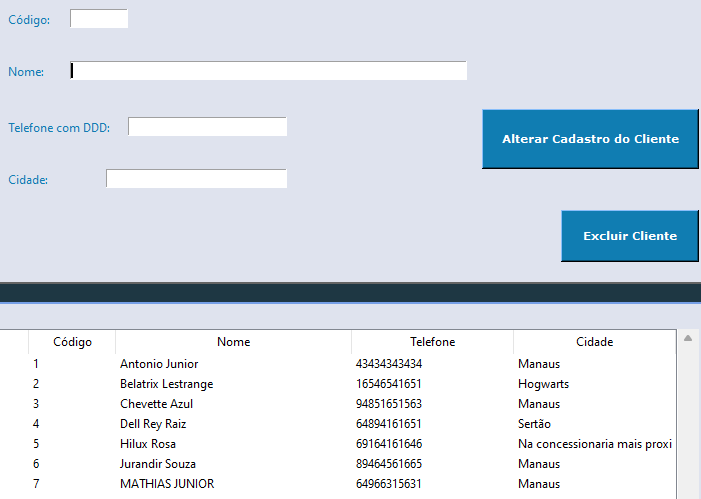
**Utilização da Classe**

A classe pode ser utilizada em um ambiente onde os administradores da biblioteca precisam gerar relatórios frequentemente para monitoramento de inventário ou para fornecer informações específicas aos clientes sobre seus empréstimos ou detalhes cadastrais.

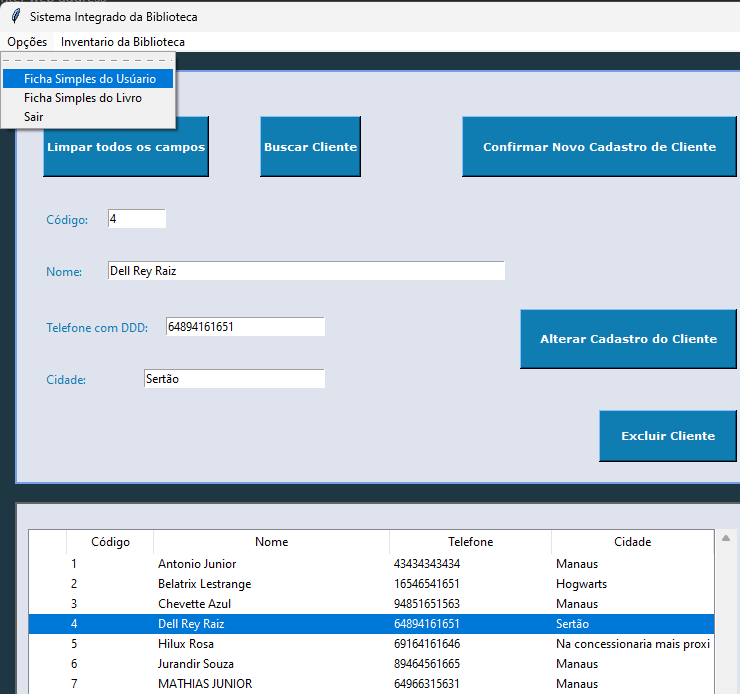


Demonstração de Resultados.  


Supondo que o operador do sistema, fosse cadastrar o nome Mathias, mas acabou digitando errado e não percebeu o erro e confirmou o cadastro.  


Para fazer a alteração no cadastro do Mathias o operador do sistema terá que, dar o clique duplo no cadastro do Mathias, fazer a correção e clicar em alterar cadastro do cliente.  
  
Veja que após a ação o nome esta corrigido e salvo.

Neste exemplo vamos gerar um relatório de um cliente especifico, o cliente escolhido foi Dell Rey Raiz, para isso primeiro vou na treeview e executo o clique duplo no cliente que desejo gerar o relatório, em seguida é ir em opções na barra de menu e selecionar ficha simples do usuário.



Após fazer esses passos abrira uma pagina no navegador padrão, exibindo o pdf com as informações do cliente escolhido.  
