Interface gráfica com python

Prof. Sérgio Assunção Monteiro

Prof. Kleber de Aguiar

Descrição

Apresentação dos principais frameworks e das bibliotecas necessárias para o desenvolvimento de aplicações de interface gráfica (GUI) no python, como adicionar widgets para montar a interface gráfica, como implementar uma interface para incluir dados em tabela e localizar, alterar e excluir dados em tabela.

Propósito

Compreender como criar uma aplicação GUI em python que interaja com um banco de dados, de modo a realizar as operações de inserção, consulta, alteração e exclusão de dados em uma tabela.

Preparação

Antes de iniciar este conteúdo, você precisar ter instalado uma versão do python, a IDE Spyder e o PostgreSQL.

Objetivos

Módulo 1

Frameworks e Bibliotecas para interface gráfica

Identificar alguns dos principais frameworks e as bibliotecas necessárias para a GUI

Módulo 2

Criação de aplicação GUI

Descrever a adição de widgets e montagem da interface gráfica

Módulo 3

Interação com banco de dados

Definir a interface para inclusão de dados em uma tabela no banco de dados

Módulo 4

Interface gráfica e banco de dados

Identificar a interface para localização, alteração e exclusão de dados em tabela

Introdução

Cada vez mais, nós geramos e consumimos dados para realizarmos atividades que vão desde a abertura de uma conta bancária até as informações que disponibilizamos nas redes sociais.

Portanto, é necessário entender como esses dados são gerenciados, pois são importantes para identificar pessoas, preferências, produtos, transações financeiras, entre tantas outras aplicações.

Para fazer esse gerenciamento, as aplicações usam sistemas gerenciadores de banco de dados a fim de realizar operações de inserção, consulta, alteração e exclusão de dados.

Além disso, para que o usuário possa interagir de modo eficiente com o sistema, é importante que ele tenha à disposição uma interface gráfica que facilite o seu acesso às funcionalidades implementadas.

A linguagem python aparece como uma opção muito eficaz para atingir esses objetivos, uma vez que oferece recursos para desenvolver aplicações que integrem interface gráfica com operações no banco de dados.

Ao longo deste conteúdo, apresentaremos alguns dos principais frameworks e as bibliotecas para desenvolver aplicações de interface gráfica, além de explorarmos como realizar aplicações no banco de dados.

1 - Frameworks e bibliotecas para interface gráfica

Ao final deste módulo, você será capaz de identificar alguns dos principais frameworks e as bibliotecas necessárias para a GUI.

Frameworks e bibliotecas para GUI

Conceitos

A linguagem python possui muitos frameworks para desenvolvimento de aplicações de interface gráfica para interação com o usuário, chamadas, comumente, de GUI (Graphical User Interface).

O framework mais comum é o Tkinter (python interface to Tcl/Tk-2020) que já faz parte da instalação python, mas existem outros frameworks com características específicas que podem torná-los a escolha adequada para um projeto.

Entre os frameworks para aplicações GUI mais comuns estão:

Tkinter

É o framework GUI padrão do python. Sua sintaxe é simples, possui muitos componentes para interação com o usuário. Além disso, seu código é aberto e é disponível sob a licença python. Caso ela não esteja instalada na sua versão do python, basta digitar o comando:

Terminal

```
pip install tkinter
```

Para quem usa a IDE Spyder (SPYDER, 2020), é necessário colocar uma exclamação antes do comando “pip”, ou seja:

Terminal

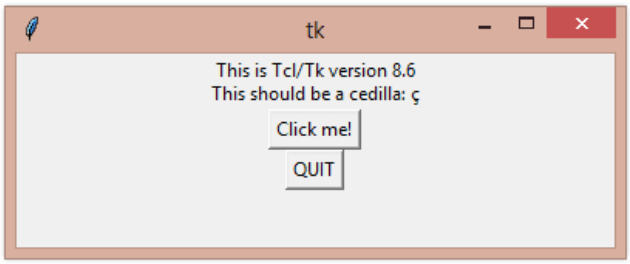
```
!pip install tkinter
```

Para testar a instalação, basta escrever o seguinte código na linha de comando:

Python

```
import tkinter
tkinter._test()
```

Se a instalação ocorreu normalmente, aparecerá uma tela com alguns componentes para que você possa interagir e ter uma impressão inicial do potencial desse framework:



Exemplo de aplicação do Tkinter.

Atenção!

Devido à importância do Tkinter, vamos explorá-lo com bastantes detalhes mais à frente.

Flexx

É um kit de ferramentas para o desenvolvimento de interfaces gráficas com o usuário implementado em python que faz uso de tecnologia web para sua renderização. O Flexx pode ser usado para criar tanto aplicações de desktop como para web e até mesmo exportar uma aplicação para um documento HTML independente. Para instalar o Flexx, basta digitar o comando:

Terminal

```
pip install flexx
```

Para quem usa a IDE Spyder, é necessário colocar uma exclamação antes do comando “pip”, isso vale para qualquer instalação de biblioteca/framework/pacote.

Comentário

Para evitar repetições sobre isso, apresentaremos aqui como instalar o pacote Flexx na IDE Spyder, mas, para os próximos casos, mostraremos apenas a instalação tradicional, ou seja, sem o símbolo de “exclamação”.

No caso do Spyder, o comando deve ser:

Terminal

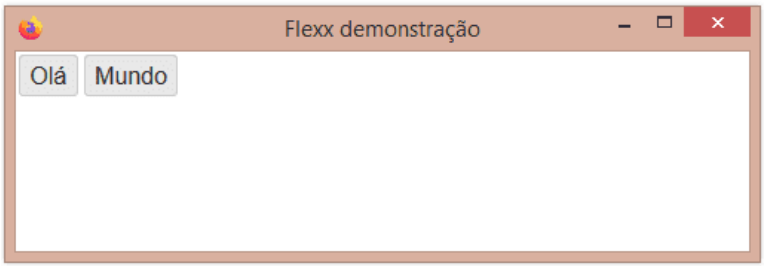
```
!pip install flexx
```

Uma forma prática de testar a instalação e aprender um pouco mais sobre esse framework é escrever o código abaixo na linha de comando, ou em um arquivo, e executar:

Python

```
from flexx import flx
class Exemplo(flx.Widget):

    def init(self):
        flx.Button(text='Olá')
        flx.Button(text='Mundo')
```

Se a instalação estiver correta, a seguinte janela vai se abrir:



Exemplo de aplicação com Flexx.

CEF python

É um projeto de código aberto voltado para o desenvolvimento de aplicações com integração ao Google Chrome. Existem muitos casos de uso para CEF. Por exemplo, ele pode ser usado para criar uma GUI baseada em HTML 5, pode usá-lo para testes automatizados, como também pode ser usado para web scraping, entre outras aplicações.

Para instalá-lo, basta digitar na linha de comando:

Terminal

```
pip install cefpython3
```

Para testar a instalação do CEF python, basta escrever o código abaixo na linha de comando, ou em um arquivo, e executar:

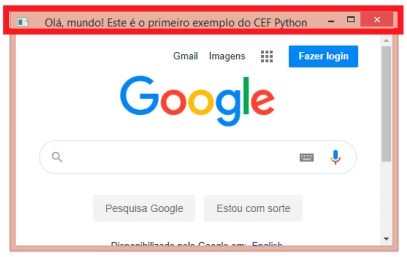
Python

```python
from cefpython3 import cefpython as cef
import platform
import sys
def main():
    check_versions()
    sys.excepthook = cef.ExceptHook # To shutdown all CEF processes on error
```

Atenção!

Como esse exemplo, apesar de simples, é um pouco maior, cabe lembrar que a indentação faz parte da sintaxe do python (python 3.8.5 documentation, 2020), por isso, é necessário ter bastante cuidado, caso contrário o programa apresentará erros.

Se tudo funcionou corretamente, quando o programa executar, abrirá a seguinte janela:



Exemplo de aplicação com CEF python.

> Observe o título da janela: "Olá, mundo! Este é o primeiro exemplo do CEF python".

Kivy

É um framework python de código aberto para o desenvolvimento de aplicações com interfaces de usuário e multitoque. Ele é escrito em python e Cython, baseado em OpenGL ES 2, suporta vários dispositivos de entrada e possui uma extensa biblioteca de componentes (widgets).

Com o mesmo código, a aplicação funciona para Windows, macOS, Linux, Android e iOS. Todos os widgets Kivy são construídos com suporte multitoque.

Para instalá-lo, é necessário escrever na linha de comando:

Terminal

```
pip install Kivy
```

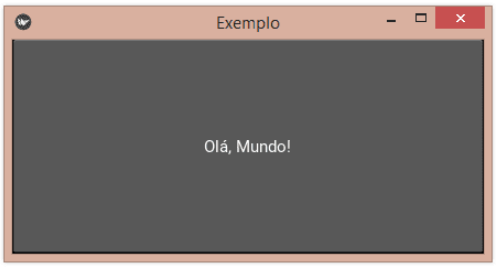
Uma forma de testar a instalação é escrever e executar o programa:

Python

```python
from kivy.app import App
```

```
from kivy.uix.button import Button

class ExemploApp(App):
    def build(self):
        return Button(text='Olá, Mundo!')
```

Se tudo funcionar corretamente, aparecerá a aplicação, conforme a próxima imagem.



Exemplo de aplicação Kivy.

Na imagem anterior, é possível identificar o título da janela “Exemplo” e um componente botão no centro da tela com a mensagem “Olá, Mundo!”. Um ponto interessante pode ser observado no código, que é a utilização da programação orientada a objetos.

Pyforms

É um framework python 3 para desenvolver aplicações que podem operar nos ambientes Desktop GUI, Terminal e Web. A biblioteca é composta por três sub-bibliotecas, cada uma implementando a camada responsável por interpretar a aplicação Pyforms em cada ambiente diferente:

1. Pyforms-gui.
2. Pyforms-web.
3. Pyforms-terminal.

Essas camadas podem ser usadas individualmente ou em conjunto, dependendo da instalação do Pyforms. Para fazer a instalação básica, é necessário escrever na linha de comando:

Python

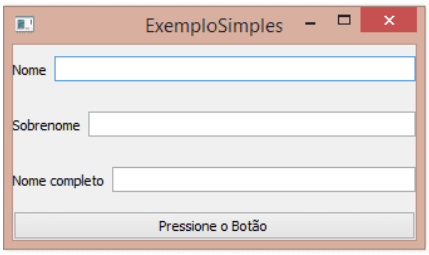
```
pip install pyforms
```

Uma forma de testar a instalação é escrever e executar o programa:

Python

```
import pyforms
from pyforms.basewidget import BaseWidget
from pyforms.controls import ControlText
from pyforms.controls import ControlButton

class ExemploSimples(BaseWidget):
```

Se tudo funcionar corretamente, aparecerá a aplicação conforme a imagem seguinte.



Exemplo de aplicação Pyform.

Comentário

No exemplo da imagem anterior, é possível ver três caixas de texto, chamadas de “controle de texto”, e um componente botão, chamado de “controle de botão” pelo Pyforms. Ele foi projetado a fim de desenvolver aplicações para executar no modo Windows GUI.

PyQt

Uma aplicação desenvolvida no framework PyQt e executada nas plataformas Windows, macOS, Linux, iOS e Android.

Trata-se de um framework que aborda, além de desenvolvimento GUI, abstrações de sockets de rede, threads, Unicode, expressões regulares, bancos de dados SQL, OpenGL, XML, entre outras aplicações.

Suas classes empregam um mecanismo de comunicação segura entre objetos que é fracamente acoplada, tornando mais fácil criar componentes de software reutilizáveis.

Para fazer a instalação básica, é necessário escrever na linha de comando:

Terminal

```
pip install PyQt5
```

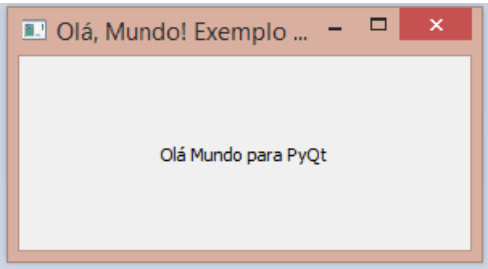
Uma forma de testar a instalação é escrever e executar o programa:

Python

```
import sys
from PyQt5 import QtCore, QtWidgets
from PyQt5.QtWidgets import QMainWindow, QLabel, QGridLayout, QWidget
from PyQt5.QtCore import QSize

class HelloWindow(QMainWindow):
    def __init__(self):
        QMainWindow.__init__(self)

        self.setMinimumSize(QSize(280, 120))
        self.setWindowTitle("Olá, Mundo! Exemplo PyQT5")

        centralWidget = QWidget(self)
        self.setCentralWidget(centralWidget)

        gridLayout = QGridLayout(self)
        centralWidget.setLayout(gridLayout)
```

```
        title = QLabel("Olá Mundo para PyQt", self)
        title.setAlignment(QtCore.Qt.AlignCenter)
```

Se tudo funcionar corretamente, aparecerá a aplicação conforme a imagem a seguir.



Exemplo de aplicação PyQt.

No exemplo da imagem acima, é possível identificar uma janela, o título da janela e uma mensagem ao centro.

wxPython

É um kit de ferramentas GUI baseadas em uma biblioteca C++ chamada wxWidgets que foi lançada em 1998. O wxpython usa os componentes (widgets) reais na plataforma nativa sempre que possível. Essa, inclusive, é a principal diferença entre o wxpython e outros kits de ferramentas, como PyQt ou Tkinter.

Atenção!

As aplicações desenvolvidas em wxpython se assemelham a aplicações nativas do sistema operacional em que estão sendo executadas.

As bibliotecas PyQt e Tkinter têm componentes personalizados. Por isso é bastante comum que as aplicações fiquem com um aspecto diferente das nativas do sistema operacional. Apesar disso, o wxpython também oferece suporte a componentes personalizados.

Para fazer a instalação básica, é necessário escrever na linha de comando:

Terminal

```
pip install wxpython
```

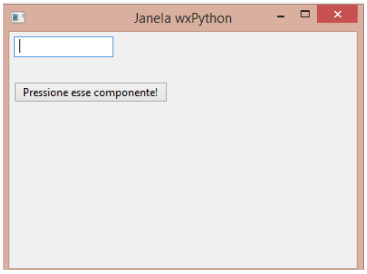
Uma forma de testar a instalação é escrever e executar o programa:

Python

```
import wx
class Janela(wx.Frame):
    def __init__(self, parent, title):
        super(Janela, self).__init__(parent, title=title, size = (400,300))
        self.panel = ExemploPainel(self)
        self.text_ctrl = wx.TextCtrl(self.panel, pos=(5, 5))
        self.btn_test = wx.Button(self.panel, label='Pressione esse componente!', pos=(5, 55))
```

```python
class ExemploPainel(wx.Panel):
    def __init__(self, parent):
        super(ExemploPainel, self).__init__(parent)


class ExemploApp(wx.App):
    def OnInit(self):
        self.frame = Janela(parent=None, title="Janela wxPython")
        self.frame.Show()
        return True

```

Se tudo funcionar corretamente, aparecerá a aplicação, conforme a imagem seguinte.



Exemplo de aplicação wxpython.

No exemplo da imagem anterior, é possível identificar uma janela, o seu respectivo título, uma caixa de texto e um botão com a mensagem “Pressione esse componente!”.

PyAutoGUI

Permite desenvolver aplicações python que controlem o mouse e o teclado para automatizar as interações com outros aplicativos.

Comentário

Uma das situações em que essa característica pode ser muito interessante é na implementação de testes que simulem a interação do usuário com o sistema.

O PyAutoGUI funciona no Windows, macOS e Linux e é executado no python.

Para fazer a instalação básica, é necessário escrever na linha de comando:

Terminal

```
!pip install PyAutoGUI
```

Uma forma de testar a instalação é escrever e executar o programa:

Python

```python
import pyautogui
screenWidth, screenHeight = pyautogui.size()
```

```
currentMouseX, currentMouseY = pyautogui.position()
pyautogui.moveTo(100, 150)
pyautogui.click()
```

Nesse código, basicamente tem-se esta sequência de instruções:

Obter o tamanho do monitor principal.

Obter a posição XY do mouse.

Mover o mouse para as coordenadas XY.

Clicar com o mouse.

Mover o mouse para as coordenadas XY e clicar nelas.

Mover o mouse 10 pixels para baixo de sua posição atual.

Clicar duas vezes com o mouse.

Usar a função de interpolação/atenuação para mover o mouse por 2 segundos com pausa de um quarto de segundo entre cada tecla.

Pressionar a tecla Esc.

Pressionar a tecla Shift e segurá-la.

Pressionar a tecla de seta para a esquerda 4 vezes.

Soltar a tecla Shift.

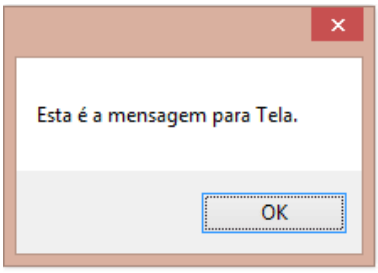
Pressionar a combinação de teclas de atalho Ctrl-C.

Mostrar uma caixa de alerta aparecer na tela e pausar o programa até clicar em OK.

Atenção!

Antes de executar o código, cabe um alerta: Essa aplicação, apesar de ser muito simples, vai interagir com o seu sistema.

Se tudo funcionar corretamente, aparecerá a aplicação conforme a próxima imagem.



Exemplo de aplicação PyAutoGUI.

A imagem vista anteriormente aparecerá depois do “cursor do mouse” se movimentar na tela e o “teclado” escrever a mensagem “Esta é a mensagem para tela”.

As possibilidades de aplicações são muitas para o PyAutoGUI.

PySimpleGUI

Esse pacote foi lançado em 2018 e possui portabilidade com os pacotes: Tkinter, PyQt, wxpython e Remi, portanto aumenta as possibilidades de uso de componentes na programação.

Para fazer a instalação básica, é necessário escrever na linha de comando:

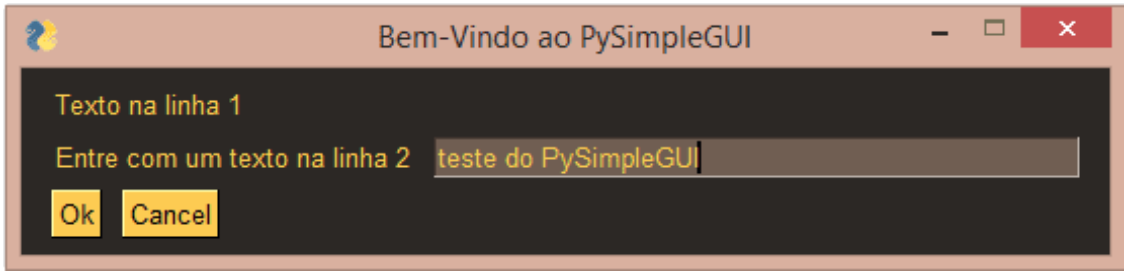
Terminal

```
pip install pysimplegui
```

Uma forma de testar a instalação é escrever e executar o programa:

```python
import PySimpleGUI as sg

sg.theme('DarkAmber')

layout = [ [sg.Text('Texto na linha 1')],
  [sg.Text('Entre com um texto na linha 2'), sg.InputText()],
```

Se tudo funcionar corretamente, aparecerá a aplicação conforme a imagem a seguir.



Exemplo PySimpleGUI.

No exemplo da imagem anterior, é possível identificar uma janela, o seu respectivo título, dois componentes “label”, uma caixa de texto e dois botões.

Atenção!

A lista de frameworks e bibliotecas disponíveis para python ainda tem muitos outros exemplos, como o PySide e o PyObject. A escolha deve levar em consideração a maturidade da biblioteca/framework e a necessidade de o projeto incorporar ou não aspectos mais elaborados de uma GUI.

Interface GUI

Vantagens e desvantagens de uma Interface GUI

Agora, discutiremos sobre alguns dos aspectos do desenvolvimento de um projeto de interface gráfica.

Vantagens

Entre as principais vantagens relacionadas às escolhas por uma biblioteca/framework para um projeto da interface GUI, estão:

- Facilidade de interação do usuário com o sistema por meio da utilização de componentes intuitivos.
- Simplicidade de utilizar os componentes em um programa.
- Compatibilidade do componente com múltiplas plataformas.
- Criação de uma camada de abstração para o programador sobre detalhes da programação dos componentes.
- Incorporação de aspectos que facilitem a experiência do usuário com o sistema.
- Facilidade para o usuário alternar rapidamente entre as funcionalidades do sistema.

- Bastante documentada e com uma comunidade engajada, o que ajuda a perceber novas aplicações e evoluções dos componentes que vão melhorar a aplicação dos componentes GUI no sistema.

Desvantagens

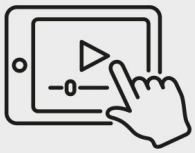
Também devem ser levadas em consideração as desvantagens relacionadas às escolhas de bibliotecas/frameworks da interface GUI:

- Alguns componentes de interface gráfica usam muitos recursos computacionais, pois têm como objetivo ser amigáveis para o usuário e não necessariamente fazer uso otimizado dos recursos. Como consequência, podem tornar as aplicações muitos lentas para plataformas de versões mais antigas.
- O uso dos componentes na programação pode ser excessivamente complexo, fazendo com que o desenvolvedor tenha que desviar esforço do objetivo principal para se preocupar com detalhes dos componentes GUI.
- As bibliotecas disponibilizam componentes, mas a aplicação deles no sistema é responsabilidade do desenvolvedor. Além de se preocupar com aspectos de programação, é necessário se preocupar como esses componentes vão ser disponibilizados no sistema, pois, caso contrário, algumas tarefas podem ser mais demoradas devido a muitos níveis de interação para selecionar a escolha desejada, como, por exemplo, o uso excessivo de menus e submenus.
- Os componentes precisam ter comportamentos previsíveis. Comportamentos inesperados são potencialmente perigosos, pois podem dar acesso não autorizado a partes do sistema, revelando, assim, vulnerabilidades de segurança.
- Sistemas baseados em componentes GUI requerem mais memória RAM para serem executados.
- Ainda sobre uso de recursos computacionais, sistemas baseados em componentes GUI demandam por processamento.
- Podem ter um comportamento distinto quando operam em diferentes plataformas, tanto na parte visual como pela demanda de recursos computacionais.
- O tempo necessário para utilizar os componentes GUI no sistema pode ser longo, em especial quando há necessidade de atender a muitos requisitos de infraestrutura.
- A curva de aprendizado para profissionais não experientes pode ser proibitiva para o desenvolvimento de projetos de curto de prazo.

Como criar uma aplicação com interface gráfica

Aprenda a criar uma aplicação com interface gráfica. Veja a apresentação do protótipo de sistema que será desenvolvido, os frameworks a serem utilizados e a preparação do ambiente para utilização.

Para assistir a um vídeo sobre o assunto, acesse a versão online deste conteúdo.



Falta pouco para atingir seus objetivos.

Vamos praticar alguns conceitos?

Questão 1

A linguagem Python é muito usada em diversas aplicações devido à simplicidade da sintaxe e por possuir muitos pacotes disponíveis que podem ser instalados por meio do comando “pip install ”. Sobre o comando “pip”, selecione a opção correta:

A Funciona como um gerenciador de dependências.

B Independentemente do ambiente de desenvolvimento, o comando “pip” deve ser usado da mesma forma.

C O comando “pip” é o único meio de fazer a instalação de pacotes no Python.

D Por se tratar de um comando nativo do Python, não é necessário se preocupar com a versão do “pip”.

E O “pip” é o interpretador utilizado pelo Python para gerar o executável.

Parabéns! A alternativa A está correta.

%0A%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%3Cp%20class%3D'c-paragraph'%3EO%20comando%20%E2%80%9Cpip%E2%80%9D%20%C3%A9%20um%20dos%20gerenciadores%20de%20depend%C3%AAncias%20do

Questão 2

A biblioteca Tkinter é considerada a biblioteca padrão do Python para o desenvolvimento de interface gráfica. Em relação à biblioteca Tkinter, selecione a opção correta:

A A biblioteca Tkinter sempre será a melhor escolha para um projeto de interface gráfica, pois é compatível com múltiplas plataformas.

B Seus componentes não se integram diretamente na linguagem Python, sendo necessário usar outros programas para ter acesso aos seus recursos.

C Seus componentes GUIs são considerados avançados e, portanto, apropriados para aplicações de jogos, por exemplo.

D A biblioteca Tkinter é a base para que todos os outros frameworks GUI possam ser utilizados.

E Para testar se ela está instalada corretamente, pode-se escrever duas linhas de código que são import tkinter e tkinter._test().

Parabéns! A alternativa E está correta.

%0A%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%3Cp%20class%3D'c-paragraph'%3EDe%20fato%2C%20a%20biblioteca%20Tkinter%20%C3%A9%20a%20mais%20importante%20do%20Python%20por%20estar%20dispon

2 - Criação de aplicação GUI

Ao final deste módulo, você será capaz de descrever a adição de widgets e montagem da interface gráfica.

Biblioteca Tkinter

Conceitos

Neste módulo, o nosso foco será na criação de uma aplicação GUI. Para isso, a biblioteca que vamos usar como referência é a Tkinter.

Comentário

Essa biblioteca GUI é considerada a padrão para desenvolvimento de interface gráfica no python. Ela fornece uma interface orientada a objetos que facilita a implementação de programas interativos.

Para usar o Tkinter – supondo que você já instalou o Pyhton e o pacote Tkinter –, é necessário executar as seguintes etapas:

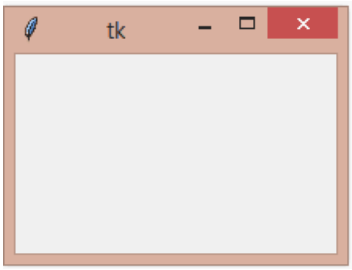
- Importar a biblioteca Tkinter.
- Criar a janela principal do programa GUI.
- Adicionar um ou mais dos widgets (componentes).

Entrar no loop de evento principal para tratar os eventos disparados pelo usuário.

O código fica assim:

```python
import tkinter
janela = tkinter.Tk()
janela.mainloop()
```

Esse programa produz a seguinte saída:



Exemplo de aplicação Tkinter.

Widgets Tkinter

O Tkinter possui diversos componentes (widgets), tais como botões, rótulos e caixas de texto usados para criar aplicações interativas com o usuário.

Os principais widgets do Tkinter, de acordo com Meier (2015), são:

Botão (Button)	É usado para exibir os botões na aplicação. São usados, por exemplo, para confirmar uma ação de salvar os dados.
Telas (Canvas)	É usado para desenhar formas, como linhas, ovais, polígonos e retângulos.
Botão de verificação (Checkbutton)	É usado para exibir várias opções como caixas de seleção. O usuário pode selecionar várias opções ao mesmo tempo.
Entrada de texto (Entry)	É usado para exibir uma caixa de texto de linha única para que o usuário digite valores de entrada.
Quadros (Frame)	É usado como um widget de contêiner, isso significa que outros componentes são adicionados a ele com o objetivo de organizar outros widgets.
Rótulo (Label)	É usado para fornecer uma legenda de linha única para outros widgets. Também pode conter imagens.
Caixa de listagem (Listbox)	É usado para fornecer uma lista de opções para um usuário.
Menubutton	É usado para exibir opções no menu.

Menu	É usado para fornecer várias possibilidades de comandos a um usuário. Esses comandos estão contidos no Menubutton.
Mensagem (Message)	É usado para exibir uma mensagem de texto e um botão para o usuário confirmar uma ação/td>
Botão de rádio (Radiobutton)	É usado para exibir várias opções, como botões de rádio. O usuário pode selecionar apenas uma opção por vez.
Escala (Scale)	É usado para fornecer um widget de controle deslizante.
Barra de rolagem (Scrollbar)	É usado para adicionar capacidade de rolagem a vários widgets.
Texto (Text)	É usado para exibir texto em várias linhas.
Toplevel	É usado para fornecer um contêiner de janela separado.
Spinbox	É uma variante do widget Entry padrão. Ele é usado para selecionar um número fixo de valores.
PanedWindow	É um widget de contêiner que pode conter qualquer número de painéis, organizados horizontalmente ou verticalmente.
LabelFrame	É um widget de contêiner simples. Seu objetivo principal é atuar como um espaçador, ou contêiner para layouts de janela.
tkMessageBox	Este módulo é usado para exibir caixas de mensagens.

Cada um desses widgets possuem propriedades que permitem personalizar tamanhos, cores e fontes que serão exibidos para o usuário.

Na próxima seção, apresentaremos alguns exemplos que vão ajudar a entender como desenvolver uma aplicação GUI.

Exemplos de aplicações GUI

Desenvolvendo exemplos de apliaçãoes GUI

Recomendação

Antes de iniciar esta seção, é fortemente recomendado que você tenha feito a instalação da versão mais atual do python e tente executar os exemplos.

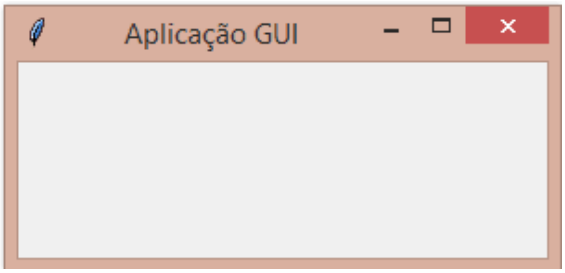
Vamos apresentar exemplos dos componentes:

1. Window
2. Label
3. Button
4. Entry
5. Radiobutton
6. Checkbox

7. Text
8. Message
9. Sliders
10. Dialog
11. Combobox

Widget Window (tk.Tk())

Para iniciar, a primeira aplicação vai exibir uma janela redimensionável, conforme a imagem seguinte.



Exemplo de uma janela redimensionável.

O código dessa aplicação é:

```python
import tkinter as tk
janela = tk.Tk()
janela.title(" Aplicação GUI")
janela.mainloop()
```

Agora, cada uma das linhas será analisada.

- Linha 1 - É feita a importação da biblioteca Tkinter.
- Linha 2 - É criada uma instância da classe Tk no objeto “janela”.
- Linha 3 - O método “title” é usado para definir um título que aparece no topo da janela, no caso, “Aplicação GUI”.
- Linha 4 - A aplicação inicia o loop de evento principal da janela.

A janela da aplicação anterior é redimensionável, ou seja, o usuário pode alterar as dimensões da janela se ele clicar com o cursor do mouse na janela e arrastá-la.

Para fixar o tamanho da janela, é necessário determinar essa propriedade conforme o código seguinte:

```python
import tkinter as tk
janela = tk.Tk()
janela.title(" Aplicação GUI NÃO Dimensionável")
janela.resizable(False, False)
janela.mainloop()
```

Atenção!

A principal diferença desse exemplo em relação ao anterior está na linha 4, onde a propriedade de redimensionar a janela é colocada como “Falso”.

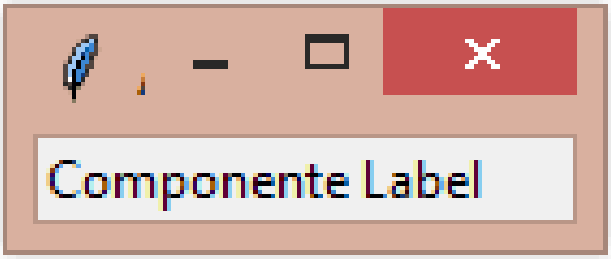
Widget Label

O próximo exemplo apresenta como usar o widget “Label”. O código para gerar uma aplicação com o “Label” é dado por:

Python

```python
import tkinter as tk
from tkinter import ttk
janela = tk.Tk()
janela.title(" Aplicação GUI com o Widget Label")
ttk.Label(janela, text="Componente Label" ).grid(column=0, row=0)
janela.mainloop()
```

Na próxima imagem, é exibido o resultado da execução:



Exemplo do componente Label.

Na linha 5, é feito o posicionamento do componente “label” na “janela” com o gerenciador de layout “grid”.

Widget Button

O próximo exemplo apresenta como usar o widget “Button”.

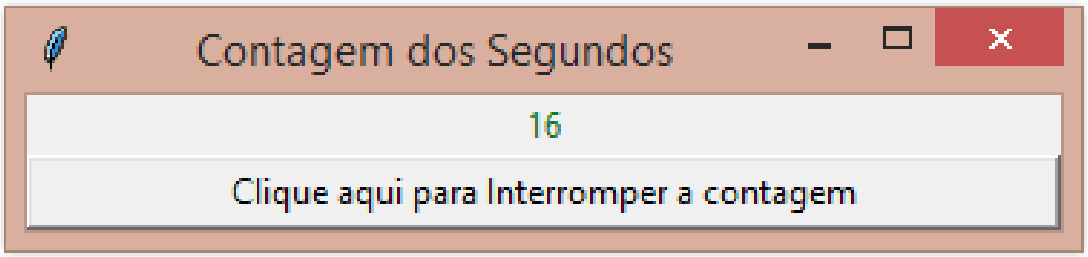
O código para gerar uma aplicação com o componente “Button” é dado por:

Python

```python
import tkinter as tk
contador = 0
def contador_label(lblRotulo):
    def funcao_contar():
        global contador
        contador = contador + 1
        lblRotulo.config(text=str(contador))
        lblRotulo.after(1000, funcao_contar)
        funcao_contar()
janela = tk.Tk()
janela.title("Contagem dos Segundos")
lblRotulo = tk.Label(janela, fg="green")
lblRotulo.pack()
contador_label(lblRotulo)
btnAcao = tk.Button(janela, text='Clique aqui para Interromper a contagem', width=50, command=janela.destroy)
btnAcao.pack()
janela.mainloop()

```

Este programa vai gerar uma janela com um contador de segundos – que utiliza um componente “label” – e um componente botão com a mensagem “Clique aqui para interromper a contagem”.

Na próxima imagem, é exibido o resultado da execução do programa.



Exemplo do componente botão.

As linhas mais importantes deste código são:

- Linha 14 - Chamada para a função “contador_label”, função que faz a contagem dos segundos e a atualização dos dados do componente “label”.
- Linha 15 - Criação de uma instância do componente “botão” com uma mensagem, largura do componente e o estabelecimento de um comportamento, no caso, fechar a janela, quando o usuário pressionar o botão.

Widget Entry

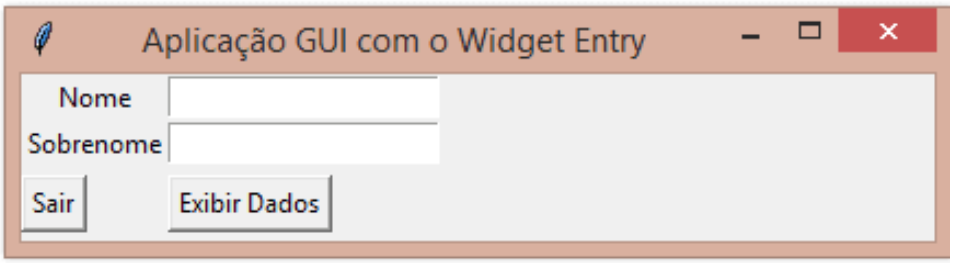
Agora, vamos analisar o componente “entry”. Ele é uma das principais formas de o usuário entrar com dados no sistema.

A seguir, é apresentado um exemplo de como usar esse componente:

Python

```python
import tkinter as tk
def mostrar_nomes():
    print("Nome: %s\nSobrenome: %s" % (e1.get(), e2.get()))
janela = tk.Tk()
janela.title("Aplicação GUI com o Widget Entry")
```

O código produzirá uma janela com duas entradas de dados (widgets entry), dois botões e dois componentes rótulos, conforme podemos ver na imagem a seguir.



Exemplo do componente entry.

Os códigos que estamos apresentando estão evoluindo em termos de aplicações práticas.

No caso do exemplo da imagem anterior, com poucas modificações, pode ser aplicado para muitas situações reais.

Agora, analisaremos os principais aspectos do código.

- Linha 2 - É implementada a função “mostrar_nomes”, que vai exibir na linha de comando os nomes que estão escritos nas instâncias “e1” e “e2” do componente “entry”.
- Linha 8 e 9 - São feitas as instâncias “e1” e “e2” do componente entry.
- Linha 10 e 11 - “e1” e “e2” são posicionados na janela.
- Linha 12 e 13 - São instanciados objetos do componente “botão”. Em especial, na linha 13, a função “mostrar_nomes” é associada ao comportamento do botão.

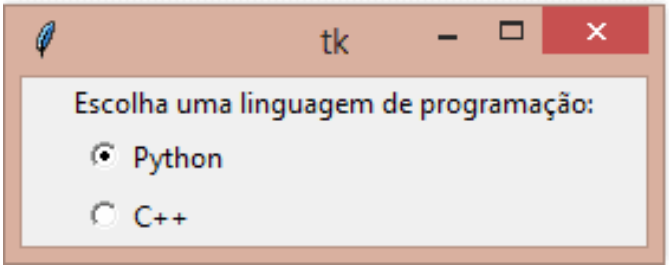
Widget Radiobutton

O próximo exemplo apresenta como usar o widget “Radiobutton”. O código para gerar uma aplicação com esse componente é dado por:

Python

```
1 import tkinter as tk
2 janela = tk.Tk()
3 v = tk.IntVar()
4 tk.Label(janela,text="""Escolha uma linguagem de programação:""",justify = tk.LEFT, padx = 20).pack()
5 tk.Radiobutton(janela,text="python",padx = 25,variable=v,value=1).pack(anchor=tk.W)
6 tk.Radiobutton(janela,text="C++",padx = 25,variable=v,value=2).pack(anchor=tk.W)
```

O código produzirá uma janela com dois “botões de rádio” (widgets radiobutton), conforme a imagem a seguir.



Exemplo do componente radiobutton.

- Linha 4 - É instanciado um componente “label” com a parametrização das propriedades “text”, “justify” e “padx” que correspondem, respectivamente, ao texto, a como ele será justificado e à largura em pixels do componente.

- Linha 5 e 6 - São instanciados os componentes “radiobuttons”. A propriedade “variable” recebe o valor “v”, indicando que o componente será colocado na vertical.

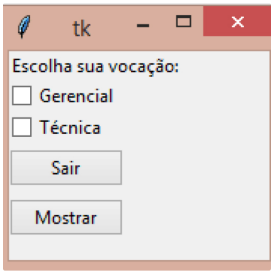
Widget Checkbox

O próximo exemplo apresenta como usar o widget “checkbox”. O código para gerar uma aplicação com esse componente é dado por:

Python

```
1  import tkinter as tk
2  from tkinter import ttk
3  janela = tk.Tk()
4  def escolha_carreira():
5     print("Gerencial: %d,\nTécnica : %d" % (var1.get(), var2.get()))
6  ttk.Label(janela, text="Escolha sua vocação:").grid(row=0, sticky=tk.W)
7  var1 = tk.IntVar()
8  ttk.Checkbutton(janela, text="Gerencial", variable=var1).grid(row=1, sticky=tk.W)
9  var2 = tk.IntVar()
10 ttk.Checkbutton(janela, text="Técnica", variable=var2).grid(row=2, sticky=tk.W)
11 ttk.Button(janela, text='Sair', command=janela.quit).grid(row=3, sticky=tk.W, pady=4)
12 ttk.Button(janela, text='Mostrar', command=escolha_carreira).grid(row=4, sticky=tk.W, pady=4)
13 janela.mainloop()
```

O código produzirá uma janela com um “label”, dois “checkboxes” e dois botões, conforme a Figura 16.



Exemplo do componente checkbox.

Agora, vamos analisar os principais aspectos do código.

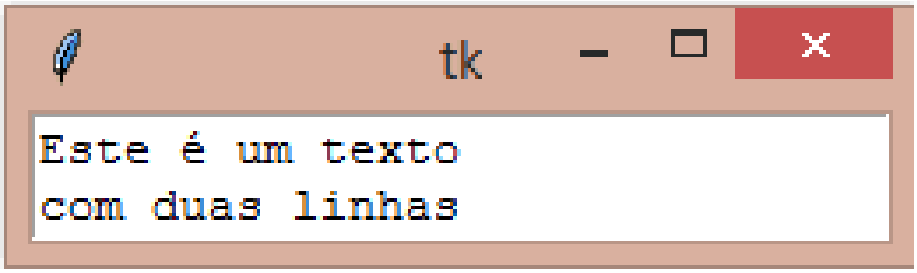
- Linha 4 - É implementada a função “escolha_carreira”, que exibirá os valores dos objetos “var1” e “var2”, que estão relacionados aos “checkboxes”.
- Linhas 7 e 9 - São instanciados os objetos “var1” e “var2”, que serão associados aos “checkboxes”. Quando um “checkbox” for selecionado, o seu respectivo objetivo vai retornar valor 1; caso o componente não seja selecionado, o valor do objeto será 0.
- Linhas 8 e 10 - São instanciados objetos dos componentes “checkboxes”, que são associados, respectivamente, às opções Gerencial e Técnica.

Widget Text

O próximo exemplo apresenta como usar o widget “Text”. O código para gerar uma aplicação com o componente “Text” é dado por:

```python
import tkinter as tk
janela = tk.Tk()
T = tk.Text(janela, height=2, width=30)
T.pack()
T.insert(tk.END, "Este é um texto\ncom duas linhas\n")
tk.mainloop()
```

O código produzirá uma janela com um texto, conforme a imagem a seguir.



Exemplo do componente Text.

Agora, vamos analisar os principais aspectos do código.

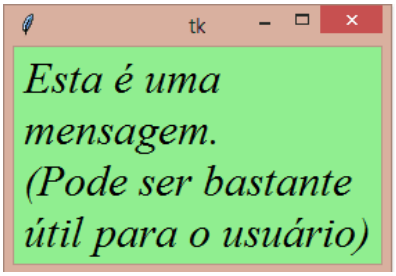
- Linha 3 - É feita uma instância do componente “Text”.
- Linha 5 - É inserido um texto na instância do componente “text”, que será exibido na tela. Observe que o texto é separado em duas linhas com o uso do “\n”.

Widget Message

O próximo exemplo apresenta como usar o widget “Message”. O código para gerar uma aplicação com o componente “Message” é dado por:

```python
import tkinter as tk
janela = tk.Tk()
mensagem_para_usuario = "Esta é uma mensagem.\n(Pode ser bastante útil para o usuário)"
msg = tk.Message(janela, text = mensagem_para_usuario)
msg.config(bg='lightgreen', font=('times', 24, 'italic'))
msg.pack()
```

O código produzirá uma janela com uma mensagem, conforme pode ser visto na imagem seguinte.



Exemplo do componente Message.

Agora, vamos analisar os principais aspectos do código.

- Linha 4 - É instanciado um componente “Message” com uma mensagem para o usuário.

- Linha 5 - O componente é configurado, determinando a cor do “background” e detalhes sobre a fonte da mensagem.

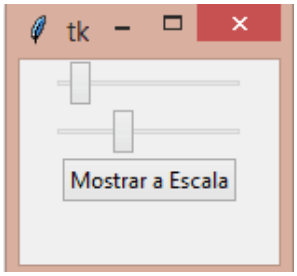
Widget Slider

O próximo exemplo apresenta como usar o widget “Slider”. O código para gerar uma aplicação com esse componente é dado por:

Python

```
1 import tkinter as tk
2 from tkinter import ttk
3 def mostrar_valores():
4     print (w1.get(), w2.get())
5 janela = tk.Tk()
6 w1 = ttk.Scale(janela, from_=0, to=50)
```

O código produzirá uma janela com duas linhas deslizantes, conforme a imagem a seguir.



Exemplo do componente Slider.

Agora, vamos analisar os principais aspectos do código.

- Linha 6 e 8 - São instanciados componentes “sliders”.

Além disso, são determinadas as propriedades “from”, “to” e “orient”, que são responsáveis, respectivamente, pelo espectro de escala componente e pela orientação do componente na tela.

Widget Dialog

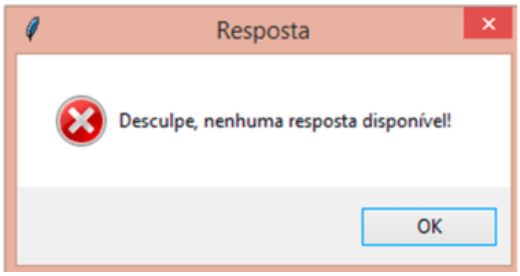
O próximo exemplo apresenta como usar o widget “Dialog”. O código para gerar uma aplicação com o componente “Dialog” é dado por:

Python

```
1 import tkinter as tk
2 from tkinter import messagebox as mb
3 def resposta():
4     mb.showerror("Resposta", "Desculpe, nenhuma resposta disponível!")
5 def verificacao():
6     if mb.askyesno('Verificar', 'Realmente quer sair?'):
```

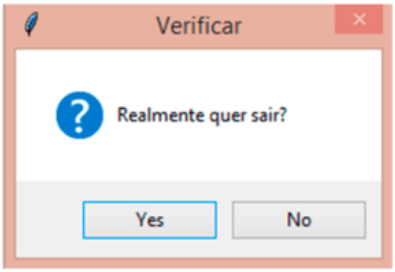
O código produzirá uma janela com dois botões, conforme a imagem a seguir.

Exemplo com dois componentes botões.



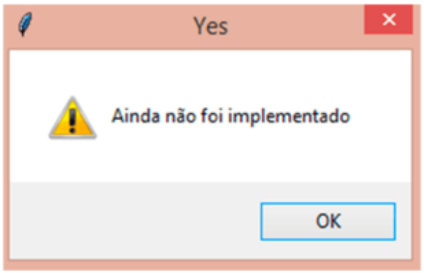
Resposta

Caso o usuário pressione o botão “Resposta”, aparecerá uma janela com a mensagem “Desculpe, nenhuma resposta disponível!”, conforme a figura.



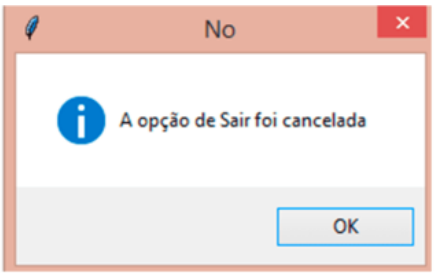
Sair

Caso o usuário tenha pressionado o botão “Sair”, aparecerá uma janela com a mensagem “Realmente quer sair?”, conforme mostrado na figura.



Resposta

Caso o usuário pressione o botão “Yes”, aparecerá uma janela com a mensagem “Ainda não foi implementado”, conforme mostra a imagem.



No

Por outro lado, se o usuário pressionar o botão “No” da janela da imagem, será exibida uma janela com a mensagem “A opção de Sair foi cancelada”, conforme mostrado na imagem.

Agora, vamos analisar os principais aspectos do código.

- Linha 4, 6, 7 e 9 - São instanciados componentes “messageDialog”.

Esse exemplo é bastante interessante, pois, apesar de pouca implementação, é possível que o usuário tenha bastante interação com o sistema.

Widget Combobox

O próximo exemplo apresenta como usar o widget “Combobox”. O código para gerar uma aplicação com esse componente é dado por:

Python

```
1  import tkinter as tk
2  from tkinter import ttk
3  # Criação de uma janela tkinter
4  janela = tk.Tk()
5  janela.title('Combobox')
6  janela.geometry('500x250')
7  # Componente Label
8  ttk.Label(janela, text = "Combobox Widget",background = 'green', foreground ="white",font = ("Times New Roman", 15
9  # Componente Label
10 ttk.Label(janela, text = "Selecione um mês :",font = ("Times New Roman", 10)).grid(column = 0,row = 5, padx = 10,
11 # Componente Combobox
12 n = tk.StringVar()
13 escolha = ttk.Combobox(janela, width = 27, textvariable = n)
14 # Adição de itens no Combobox
15 escolha['values'] = (' Janeiro',' Fevereiro',' Março',' Abril',' Maio',' Junho',' Julho',' Agosto',' Setembro',' O
16 escolha.grid(column = 1, row = 5)
17 escolha.current()
18 janela.mainloop()
```

O código produzirá uma janela com um “combobox”:

Exemplo do componente combobox.

Agora, vamos analisar os principais aspectos do código.

- Linha 17 - É instanciado um componente “combobox”.
- Linha 19 a 30 - Os meses do ano são atribuídos ao componente.

Projetando a interface gráfica

Agora, daremos continuidade e projetaremos a interface gráfica. Veja como são carregados os widgets necessários para o funcionamento do protótipo.

Para assistir a um vídeo sobre o assunto, acesse a versão online deste conteúdo.

Falta pouco para atingir seus objetivos.

Vamos praticar alguns conceitos?

Questão 1

Considere o fragmento de código Python abaixo.

```python
import tkinter
lacuna_I = lacuna_II.Tk()
lacuna_III.mainloop()
```

Para que o código seja compilado e executado corretamente, as palavras lacuna_I, lacuna_II e lacuna_III devem ser substituídas, respectivamente, por:

A tk, janela, tkinter

B raiz, tkinter, janela

C tkinter, raiz, janela

D raiz, tkinter e raiz

E tkinter, janela e raiz

Parabéns! A alternativa D está correta.

%0A%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%3Cp%20class%3D'c-paragraph'%3EA%20Tkinter%20%C3%A9%20considerada%20a%20biblioteca%20padr%C3%A3o%20de%20interface%20gr%C3%A1fica%20do%20Pytho

Questão 2

O Tkinter possui diversos componentes (widgets) que são úteis para criar aplicações que vão facilitar a interação do usuário com o sistema.

Nesse sentido, selecione a opção que relaciona corretamente o componente e sua respectiva funcionalidade.

A Button: fornece para o usuário a possibilidade de selecionar várias opções ao mesmo tempo.

B Checkbutton: é usado para a confirmação de uma ação.

C Entry: exibe uma mensagem de texto para o usuário.

D Label: é usado para fornecer uma legenda para outros componentes do projeto.

E Dialog: produz uma janela com um “combobox”.

Parabéns! A alternativa D está correta.

%0A%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%3Cp%20class%3D'c-paragraph'%3EO%20componente%20%E2%80%9Clabel%E2%80%9D%20%C3%A9%20muito%20%C3%BAtil%20para%20fornecer%20uma%20explica%C

3 - Interação com banco de dados

Ao final deste módulo, você será capaz de definir a interface para inclusão de dados em uma tabela no banco de dados.

Visão geral

Conceitos

O python possui muitas bibliotecas para interagir com diversos bancos de dados. Aqui, o foco será a integração com o PostgreSQL. Isso porque pode ser usado gratuitamente, possui bastante documentação disponível on-line e pode ser aplicado para resolver problemas reais.

Para explicar como o python interage com o PostgreSQL, trabalharemos na seguinte aplicação:

> **"Desenvolver um programa em que seja possível realizar as operações CRUD para gerenciar uma agenda telefônica que vai tratar de nomes e números de telefones".**

RUD

Acrônimo da expressão do idioma Inglês, Create (Criação), Read (Consulta), Update (Atualização) e Delete (Destruição).

A ideia é desenvolvermos uma aplicação CRUD para que os usuários possam interagir com o sistema para inserir, modificar, consultar e excluir dados de uma agenda telefônica, e que essas operações fiquem sob a responsabilidade do sistema gerenciador de banco de dados.

Dica

Inicialmente, é necessário ter o PostgreSQL instalado. Os exemplos apresentados aqui foram implementados na versão 4.24 do PostgreSQL.

Sql

```sql
CREATE TABLE public."AGENDA"
(
    id integer NOT NULL,
    nome text COLLATE pg_catalog."default" NOT NULL,
    telefone char(12) COLLATE pg_catalog."default" NOT NULL
)
```

```
TABLESPACE pg_default;
ALTER TABLE public."AGENDA"
    OWNER to postgres;
```

Para testar se a criação da tabela está correta, pode-se inserir um registro da seguinte maneira:

Sql

```sql
INSERT INTO public."AGENDA"( id, nome, telefone)
VALUES (1, 'teste 1', '02199999999');
```

Em seguida, fazemos uma consulta na tabela:

Sql

```sql
SELECT * FROM public." AGENDA";
```

Se tudo funcionar corretamente, aparecerá a seguinte saída:

	id integer	nome text	telefone character (12)
1	1	teste 1	02199999999

Saída da consulta na tabela AGENDA

Agora, vamos estudar o pacote que será utilizado para que o python possa interagir com o PostgreSQL: psycopg2.

Atenção!

O PostgreSQL pode ser integrado ao python usando o módulo psycopg2. Trata-se de um adaptador de banco de dados PostgreSQL. O pacote psycopg2 é simples, rápido e estável.

Para instalá-lo, basta digitar na linha de comando do python:

Terminal

```
pip install psycopg2
```

Para interagir com o banco de dados PostgreSQL com o uso da biblioteca psycopg2, primeiro é necessário criar um objeto Connection, que representa o banco de dados, e, em seguida, pode-se criar um objeto cursor que será bastante útil para executar todas as instruções SQL. Isso será detalhado um pouco mais à frente.

Antes disso, vamos apresentar as principais APIs (rotinas de interface de programação) da psycopg2:

PostgreSQL

psycopg2.connect (database = "NomeDoBancoDeDados", user = "LoginDoUsuário", senha = "SenhaDoBancoDeDados", host = "EndereçoDaAplicação", porta = "NúmeroDaPorta")

A conexão com o banco de dados PostgreSQL é feita com essa API. O banco de dados retorna um objeto de conexão, se o banco de dados for aberto com sucesso. Para aplicações que vão rodar localmente, utiliza-se o endereço de localhost dado por 127.0.0.1". A porta de comunicação padrão do PostgreSQL é a "5432", mas esse valor pode ser mudado.

Criar um cursor

connection.cursor ()

Esta API cria um cursor que será usado ao longo da programação para interagir com o banco de dados com python.

Executar uma instrução SQL

cursor.execute (sql [, parâmetros opcionais])

Esta rotina é aplicada para executar uma instrução SQL. A instrução SQL pode ser parametrizada.

Por exemplo, seja o trecho de código:

Sql

```
nomeDaTabela = 'tabelaExemplo'
cursor.execute(" insert into %s values (%%s, %%s)" % nomeDaTabela,[10, 20])
```

O comando executará a instrução “insert” para inserir os valores 10 e 20 na tabela ‘tabelaExemplo’.

Executa um comando SQL

cursor.executemany (sql, sequência_de_parâmetros) Esta rotina executa um comando SQL em todas as sequências de parâmetros.

Por exemplo, seja o trecho de código:

```sql
carros = (
(1, 'Celta', 35000),
(2, 'Fusca', 30000),
(3, 'Fiat Uno', 32000)
)

con = psycopg2.connect(database='BancoExemplo', user='postgres',
      password='postgres')

cursor = con.cursor()

query = "INSERT INTO cars (id, nome, preco) VALUES (%s, %s, %s)"
cursor.executemany(query, carros)

```

O trecho de código começa com uma lista de três carros, na qual cada carro possui um código de identificação, um nome e um preço.

Em seguida, é feita uma conexão com o banco de dados “BancoExemplo”.

Logo depois, é criado o cursor que será usado para realizar as operações sobre o banco de dados.

Por fim, é executada a rotina “executemany”, sendo que ela recebe uma query e uma lista de carros que serão inseridos no banco de dados.

- **cursor.callproc ('NomeDaFunção_Ou_NomeDoProcedimentoArmazenado', [parâmetros IN e OUT,])**
  Esta rotina faz chamada para uma função ou um procedimento armazenado do banco de dados. Os parâmetros IN e OUT correspondem, respectivamente, aos parâmetros de entrada e saída da função ou do procedimento armazenado e devem ser separados por vírgulas.
- **cursor.rowcount**
  Este atributo retorna o número total de linhas do banco de dados que foram modificadas, inseridas ou excluídas pela última instrução de “execute“.
- **connection.commit()**
  Este método confirma a transação atual. É necessário que ele seja chamado ao final de uma sequência de operações sql, pois, caso contrário, tudo o que foi feito desde a última chamada até o “commit” não será visível em outras conexões de banco de dados.
- **connection.rollback()**
  Este método reverte quaisquer mudanças no banco de dados desde a última chamada até o “commit”.
- **connection.close()**
  Este método fecha a conexão com o banco de dados. Ele não chama o “commit” automaticamente. Se a conexão com o banco de dados for fechada sem chamar o “commit” primeiro, as alterações serão perdidas.
- **cursor.fetchone()**
  Este método busca a próxima linha de um conjunto de resultados de consulta, retornando uma única sequência, ou nenhuma, quando não há mais dados disponíveis.
- **cursor.fetchmany([size = cursor.arraysize])**
  Esta rotina busca o próximo conjunto de linhas de um resultado de consulta, retornando uma lista. Uma lista vazia é retornada quando não há mais linhas disponíveis. O método tenta buscar quantas linhas forem indicadas pelo parâmetro “size”.
- **cursor.fetchall()** Esta rotina busca todas as linhas (restantes) de um resultado de consulta, retornando uma lista. Uma lista vazia é retornada quando nenhuma linha está disponível.

Exemplo prático com psycopg2

Um exemplo completo

Agora, apresentaremos um exemplo completo de como usar o psycopg2 que abordará:

Criação de uma tabela

Inserção de dados

Seleção de dados

Atualização de dados

Exclusão de dados

Agora, vamos aos exemplos que implementam as operações CRUD.

Criação de tabela

Este primeiro código mostra como criar uma tabela a partir do python. É uma alternativa em relação a criar a tabela usando o PostgreSQL.

Sql

```
import psycopg2
conn = psycopg2.connect(database = "postgres", user = "postgres", password = "senha123", host = "127.0.0.1", port =
print("Conexão com o Banco de Dados aberta com sucesso!")
cur = conn.cursor()
cur.execute('''CREATE TABLE Agenda(ID INT PRIMARY KEY NOT NULL,Nome TEXT NOT NULL,Telefone CHAR(12));''')
print("Tabela criada com sucesso!")
conn.commit()
conn.close()
```

Depois de colocar o programa para executar, se tudo funcionar corretamente, aparecerão as mensagens na tela:

Conexão com o Banco de Dados feita com Sucesso! Tabela criada com sucesso!

Agora, vamos analisar o código.

- Linha 1 - É feita a importação da biblioteca psycopg2.
- Linha 2 - É feita a conexão com o banco de dados. Observe os parâmetros da função “connect”, pois é necessário que você crie um banco no PostgreSQL com usuário e senha, conforme escrito na função.
- Linha 3 - É exibida uma mensagem de sucesso para o usuário.
- Linha 4 - É criado o cursor que vai permitir realizar operações no banco de dados.
- Linha 5 - Executa o comando SQL para criar a tabela “Agenda” com os campos “ID”, “Nome” e “Telefone”.
- Linha 9 - É exibida uma mensagem de criação da tabela com sucesso.
- Linha 10 - É executada a função “commit” para confirmar a execução das operações SQL.
- Linha 11 - Por fim, é fechada a conexão com o banco de dados.

Inserção de dados na tabela

O exemplo desse código mostra como inserir um registro em uma tabela a partir do python usando a biblioteca psycopg2.

O exemplo desse código mostra como inserir um registro em uma tabela a partir do python usando a biblioteca psycopg2.

Sql

```
import psycopg2
conn = psycopg2.connect(database = " postgres", user="postgres" , password=" senha123" , host="127.0.0.1", port="543
print ("Conexão com o Banco de Dados aberta com sucesso!")
cur=conn.cursor()
cur.execute("""INSERT INTO public."AGENDA" ("id", "nome" , "telefone" ) VALUES (1, 'Pessoa 1' , '02199999999' )""")
conn.commit()
print("Inserção realizada com sucesso!"); 8conn.close()
```

As mensagens que vão aparecer depois da execução do programa são:

Conexão com o Banco de Dados feita com Sucesso! Inserção realizada com sucesso!

Vamos analisar o código.

- Linha 1 a 4 - São realizadas as mesmas operações do exemplo anterior: Importação da biblioteca “psycopg2”, abrir a conexão com o banco de dados “postgres” e impressão da mensagem “Conexão aberta com sucesso!”.
- Linha 5 - É executado o comando SQL para inserir dados na tabela AGENDA.No caso, o registro é formado pelos seguintes dados: O campo “id” recebe o valor 1, o campo “nome” recebe o valor ‘Pessoa 1’ e, por fim, o campo “telefone” recebe o valor ‘02199999999’.
  Essa linha tem mais algumas questões que merecem destaque: O uso de aspas simples e duplas.
  No caso do banco de dados PostgreSQL, o nome da tabela e dos campos deve estar entre aspas duplas, por causa disso é que o comando insert possui três aspas duplas logo no início e no final. Sendo assim, muita atenção com isso, pois existem algumas variações conforme o sistema gerenciador de banco de dados escolhido.
- Linha 6 a 8 - Do mesmo modo como foi realizado no exemplo de criação da tabela “Agenda”, são realizadas as seguintes operações: “commit” das operações do banco de dados, fechamento da conexão com o banco de dados, impressão na linha de comando da mensagem “Inserção realizada com sucesso!”.

Seleção de dados na tabela

Antes de descrever o exemplo de seleção de dados, já podemos perceber algo em comum em todas as operações dos códigos para trabalhar com banco de dados no início do código:

Importação da biblioteca “psycopg2”.

Abertura da conexão com o banco de dados “postgres”.

ostgres

Perceba que esse nome não é do sistema gerenciador de banco de dados, e sim um nome que escolhemos. Poderia ser, por exemplo, "banco_dados_teste”.

E no final do código:

“Commit” das operações realizadas no banco de dados para confirmar a execução delas. Esse comando é obrigatório.

Fechamento da conexão com o banco de dados.

Agora, vamos analisar o código que mostra como selecionar um registro em uma tabela a partir da biblioteca psycopg2 do python.

Python

```
1 import psycopg2
2 conn = psycopg2.connect(database = " postgres", user="postgres" , password=" senha123" , host="127.0.0.1", port="54
3 print ("Conexão com o Banco de Dados aberta com sucesso!")
4 cur=conn.cursor()
5 cur.execute("""select * from public."AGENDA" where "id"=1""")
6 registro=cur.fetchone()
7 print(registro) 8 conn.commit() 9 print("Seleção realizada com sucesso!");
8 conn.close()
```

Depois de colocar o programa para executar, se tudo funcionar corretamente, aparecerão as mensagens na tela:

Conexão com o Banco de Dados feita com Sucesso!
(1, 'Pessoa 1', '02199999999 ')
Tabela criada com sucesso!

Vamos analisar os trechos mais importantes do código.

- Linha 5 - É feita a consulta na tabela “Agenda” pelo registro com “id” igual a 1, por meio do comando Select do SQL.
- Linha 6 - É executado o método “fetchone” que recupera exatamente um registro do “cursor” e atribui para a variável “registro”.
- Linha 7 - É impresso na linha de comando o resultado da consulta armazenado na variável “registro”.

Atualização de dados na tabela

Este exemplo mostra como atualizar os registros de uma tabela a partir do python.

Python

```
1  import psycopg2
2  conn = psycopg2.connect(database = " postgres", user="postgres" , password="senha123" , host="127.0.0.1" , port="5
3  print ("Conexão com o Banco de Dados aberta com sucesso!")
4  cur=conn.cursor()
5  print("Consulta antes da atualização")
6  cur.execute("""select * from public."AGENDA" where "id"=1""")
7  registro=cur.fetchone()
8  print(registro)
9  #Atualização de um único registro
10 cur.execute("""Update public."AGENDA" set "telefone"='02188888888' where "id"=1""")
11 conn.commit()
12 print("Registro Atualizado com sucesso! ")
13 cur = conn.cursor()
14 print(" Consulta depois da atualização")
15 cur.execute("""select * from public."AGENDA" where "id"=1""")
16 registro=cur.fetchone()
17 print(registro)
18 conn.commit()
19 print("Seleção realizada com sucesso!");
20 conn.close()
```

Este código possui três partes distintas, que são:

Parte 1

Uma consulta antes da atualização que mostra os dados do registro antes de serem modificados.

Parte 2

A atualização do registro que vai modificar os dados.

Parte 3

Uma consulta depois da atualização do registro que mostra como ficaram os dados do registro depois de serem atualizados.

Atenção!

A linha 10 é a mais importante deste código. É nela que é executado o comando “update” do sql, que atualizará o dado do campo “telefone” do registro, cujo campo “id” contenha o valor “1”.

Perceba, ainda, que é associado o comando “commit” para o comando “update” do sql na linha 11.

Exclusão de dados na tabela

Por fim, vamos ver o exemplo para excluir um registro de uma tabela.

Python

```
1  import psycopg2
2  conn = psycopg2.connect(database = " postgres", user="postgres" , password="senha123" , host="127.0.0.1" , port="5
3  print ("Conexão com o Banco de Dados aberta com sucesso!")
4  cur=conn.cursor()
5  cur.execute("""Delete from public."AGENDA" where "id"=1""")
6  conn.commit()
7  cont=cur.rowcount
8  print(cont, "Registro excluído com sucesso!")
9  print("Exclusão realizada com sucesso!");
10 conn.close()
```

Depois de colocar o programa para executar, se tudo funcionar corretamente, aparecerão as mensagens na tela:

Conexão com o Banco de Dados aberta com Sucesso!
1 Registro excluído com sucesso!Exclusão realizada
com sucesso!

Vamos analisar as partes mais importantes do código.

- Linha 5 - É feita a consulta na tabela “Agenda” pelo registro com “id” igual a 1, por meio do comando Select do SQL.É executado o comando “delete” do sql que excluirá o registro cujo campo “id” seja igual a “1”.
- Linha 7 - A propriedade “rowcount” do “cursor” retorna a quantidade de registros que foram excluídos da tabela “Agenda”.
- Linha 8 - É impresso na linha de comando o total de registros que foram excluídos.

Saiba mais

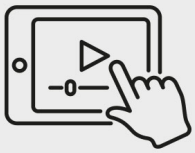
Além da biblioteca psycopg2, existem outras bibliotecas para trabalhar com vários sistemas gerenciadores de banco de dados. Por exemplo:

- pyMySQL: Biblioteca que faz interface com o MySQL.
- cx_Oracle: Biblioteca que faz interface com o Oracle.
- PySqlite: Biblioteca que faz interface com o SQLite.
- PyMongo: Biblioteca que faz interface com o mongodb, que é um banco de dados NoSQL.

Protótipo do sistema

Dando continuidade ao desenvolvimento do protótipo do sistema, veja a modelagem do banco de dados a ser utilizado, criando o banco e fazendo as inserções/atualizações/remoções necessárias.

Para assistir a um vídeo sobre o assunto, acesse a versão online deste conteúdo.



Falta pouco para atingir seus objetivos.

Vamos praticar alguns conceitos?

Questão 1

Considere o fragmento de código Python abaixo que utiliza a biblioteca “psycopg2” para fazer operações no sistema gerenciador de banco de dados Postgre:

Python

```
import psycopg2
conn = psycopg2.connect(database = "postgres", user = "postgres", password = " senha123", host = "127.0.0.1", p
cursor = conn.cursor()
cursor.execute("""lacuna_I INTO public."AGENDA" ("id", "nome", "telefone") VALUES (1, 'Pessoa 1', '02199999999'
conn. lacuna_II()
print("Inserção realizada com sucesso!");
```

Para que o código seja compilado e executado corretamente, as palavras lacuna_I, lacuna_II e lacuna_III devem ser substituídas, respectivamente, por:

A UPDATE, connect, close

B INSERT, fetchone, finally

C INSERT, commit, close

D INSERT, cursor e close

E UPDATE, commit, close

Parabéns! A alternativa C está correta.

%0A%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%3Cp%20class%3D'c-paragraph'%3EA%20palavra-chave%20%E2%80%9CINTO%E2%80%9D%20est%C3%A1%20relacionada%20ao%20comando%20%E2%80%9CINSERT%E2%80%9D%20do%20sql.%20A lo%20mediante%20o%20comando%20%E2%80%9Ccommit%E2%80%9D.%20No%20final%20da%20execu%C3%A7%C3%A3o%20de%20um%20program se%20fechar%20a%20conex%C3%A3o%20com%20o%20banco%20de%20dados%20com%20o%20comando%20%E2%80%9Cclose%E2%80%9D.%3C%

Questão 2

Existem muitas bibliotecas disponíveis para Python para fazer interface com sistemas gerenciadores de bancos de dados. Entre elas, está a “psycopg2”. Essa biblioteca faz interface com o PostgreSQL, que é um sistema importante para desenvolver aplicações reais. Nesse sentido, selecione a opção que relaciona B às APIs do psycopg2 e suas funcionalidades.

A cursor.rowcount: Indica se o comando sql – inserção, ou exclusão - foi executado com sucesso.

B connection.commit(): Abre a conexão com o banco de dados.

C cursor.execute: É aplicada para executar uma instrução SQL.

D connection.close(): Limpa as variáveis que estão associadas a operações com o banco de dados.

E cursor.execute: Fecha a conexão com o banco de dados.

Parabéns! A alternativa C está correta.

%0A%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%3Cp%20class%3D'c-paragraph'%3EO%20comando%20%E2%80%9Cexecute%E2%80%9D%20%C3%A9%20fundamental%20para%20executar%20as%20instru%C3%A7%C3%

4 - Interface gráfica e banco de dados

Ao final deste módulo, você será capaz de identificar a interface para localização, alteração e exclusão de dados em tabela.

Visão geral

Conceitos

Até o dado momento, vimos como criar uma aplicação com componentes de interface gráfica e como interagir com um banco de dados.

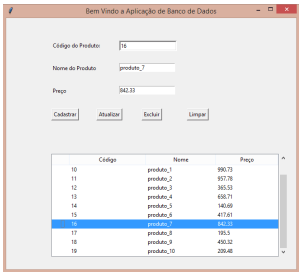
Neste módulo, criaremos uma aplicação que integra tanto elementos de interface gráfica quanto operações com banco de dados.

> **A nossa aplicação implementa as operações CRUD, que são: inserção, seleção, atualização e exclusão de dados.**

O usuário fará a entrada de dados mediante componentes de caixas de texto (widget entry) e confirmará a ação que deseja quando pressionar o botão correspondente.

Além disso, os dados que estão armazenados no banco são exibidos em um componente do tipo grade (widget treeview). O usuário tem a possibilidade de selecionar um registro na grade, o qual será exibido nas caixas de texto, onde poderá ser modificado ou excluído.

Observe a interface gráfica da nossa aplicação:



Interface da aplicação com Tkinter.

Atenção!

Perceba que alguns dados já estão armazenados no banco e são exibidos na grade. Veja que o usuário selecionou o “produto_7” na grade e seus dados estão exibidos nas caixas de texto.

Criação de tabelas e geração de dados

Criação de tabelas no Postgresql

A primeira ação a ser feita é a criação da tabela.

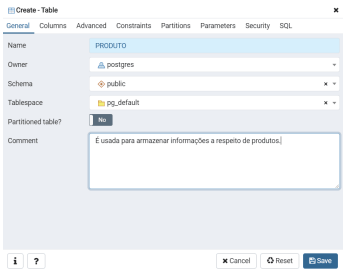
Comentário

No caso do nosso sistema, vamos criar a tabela Produto, que tem três campos: “CODIGO”, “NOME” e “PRECO”.

Estamos usando o PostgreSQL para gerenciar nossos dados. O PostgreSQL é um sistema gerenciador de banco de dados de licença gratuita e é considerado bastante robusto para aplicações de um modo geral.

Com a ferramenta pgAdmin, o desenvolvedor pode criar a tabela produto.

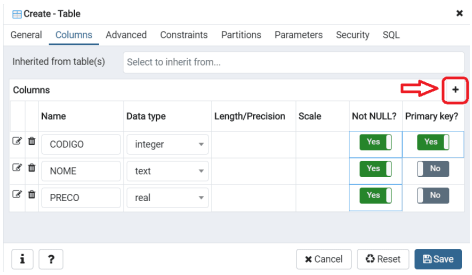
Veja como criar uma tabela no pgAdmin:



Criação de tabela no pgAdmin.

> **O campo “owner” é o usuário proprietário do banco de dados que terá acesso à tabela “PRODUTO”. No caso, usamos o usuário padrão do PostgreSQL.**

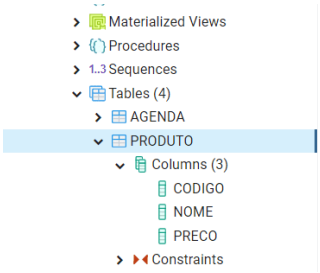
Em seguida, precisamos criar os campos tabelas. Para isso, selecionamos a opção “Columns” da imagem anterior. Será exibida a tela da imagem seguinte, na qual podemos adicionar as colunas por meio da opção “+” no canto superior direito da tela.



Criação dos campos da tabela no pgAdmin.

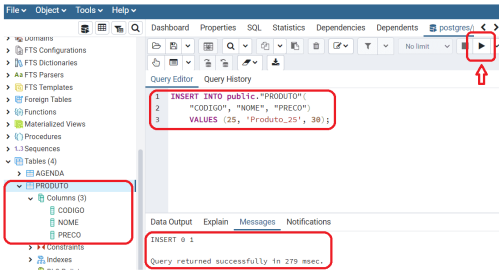
> **Perceba que o campo “CODIGO” é a chave primária da tabela e, além disso, todos os campos não podem ser vazios.**

Agora, salvamos a tabela e, finalmente, ela é criada no banco de dados. Na imagem a seguir, podemos ver a tabela no pgAdmin.



Visualização da tabela no pgAdmin.

O pgAdmin é bastante útil para fazermos operações sobre as tabelas. Na imagem seguinte, mostramos como inserir dados na tabela “PRODUTO”.



Inserção de dados no pgAdmin.

Para executar o comando de inserção, basta pressionar a seta no canto direito superior da tela.

Outra forma de criar uma tabela é mediante o próprio python, com o uso da biblioteca “psycopg2”.

Na próxima imagem, mostramos o código para criação da tabela “PRODUTO”. O código para criação da tabela pode ser baixado clicando aqui.

```
8  import psycopg2
9  conn = psycopg2.connect(database = "postgres", user = "postgres",
10                          password = "senha123",
11                          host = "127.0.0.1", port = "5432")
12 print("Conexão com o Banco de Dados feita com Sucesso!")
13 cur = conn.cursor()
14 cur.execute('''CREATE TABLE PRODUTO(CODIGO INT PRIMARY KEY NOT NULL,
15               NOME TEXT NOT NULL,
16               PRECO REAL NOT NULL);''')
17 print("Tabela criada com sucesso!")
18 conn.commit()
19 conn.close()
```

Criação de tabela no python.

Agora, as principais linhas serão analisadas.

- Linha 8 - Importamos a biblioteca psycopg2.
- Linha 9 - Fazemos a conexão ao banco de dados “postgres” com o usuário “postgres”, senha “senha123”, host local (127.0.0.1) e porta “5432”.
- Linha 13 - Abrimos o cursor. Lembre-se de que é com o cursor que fazemos as operações no banco de dados.
- Linha 14 a 16 - Executamos o comando sql “Create Table” para criar a tabela “PRODUTO” com os campos “CODIGO”, “NOME” e “PRECO”.
- Linha 18 - Executamos o comando “commit” para confirmar o comando sql.
- Linha 19 - Fechamos a conexão com o banco de dados.

Geração de dados aleatórios

Uma boa forma de iniciar o projeto é inserindo dados aleatórios na tabela. Para isso, vamos usar o pacote “faker”, que é bastante útil para gerar dados aleatórios.

Na imagem a seguir, mostramos o código para gerar os dados aleatórios.

```python
#----------------------------------------------------------------------
from faker import Faker
import psycopg2
#----------------------------------------------------------------------
conn = psycopg2.connect(database = "postgres", user = "postgres",
                        password = "senha123", host = "127.0.0.1", port = "5432")
print ("Conexão aberta com sucesso!")
cursor = conn.cursor()
fake = Faker('pt_BR')
#----------------------------------------------------------------------
n=10
for i in range(n):
    codigo = i+10
    nome ='produto_'+str(i+1)
    preco = fake.pyfloat(left_digits=3, right_digits=2, positive=True,
                         min_value=5, max_value=1000)
    print(preco)
    print(nome)


    comandoSQL = """ INSERT INTO public."PRODUTO" ("CODIGO", "NOME", "PRECO")
    VALUES (%s,%s,%s)"""
    registro = (codigo, nome, preco)
    cursor.execute(comandoSQL, registro)
#----------------------------------------------------------------------
conn.commit()
print("Inserção realizada com sucesso!");
conn.close()
#----------------------------------------------------------------------
```

Geração de dados aleatórios.

- Linha 15 - Importamos a biblioteca “faker”.
- Linha 22 - Instanciamos um objeto para gerar dados aleatórios no “português do Brasil”.
- Linha 25 a 31 - Geramos os dados aleatórios que serão inseridos na tabela.
- Linha 34 e 35 - Montamos o comando sql. Mas nesse momento, trata-se apenas de um texto (string) que deve seguir uma sintaxe cuidadosa: três aspas duplas no início e no final do texto. Além disso, perceba os “%s” no trecho do “VALUES” que serão usados para entrar com os dados das variáveis.
- Linha 36 - Criamos um registro com as variáveis que serão armazenadas na tabela.
- Linha 37 - Aplicamos o comando “execute”, que executa o comando sql com a entrada de dados que definimos na variável registro.

Comentário

Esse exemplo é simples, mas ilustra testes que podem ser feitos logo no começo do projeto que são úteis para validar a entrada de dados, além de ser útil para os desenvolvedores aprenderem mais sobre o próprio sistema.

Interação entre o sistema e o banco de dados

Interação com o banco de dados

Nesta seção, apresentaremos a parte do sistema responsável pelas operações CRUD.

O sistema foi desenvolvido usando programação orientada a objetos. O arquivo do programa foi salvo com o nome “crud.py”.

A classe com os métodos para realizar as operações para interagir com o banco de dados é a “AppBD”, conforme podemos ver na próxima imagem.

```python
#----------------------------------------------------------------------
#Essa classe possui métodos CRUD
#----------------------------------------------------------------------
import psycopg2

class AppBD:
    def __init__(self):
        print('Método Construtor')

    def abrirConexao(self):
        try:
          self.connection = psycopg2.connect(user="postgres",
                                    password="senha123",
                                    host="127.0.0.1",
                                    port="5432",
                                    database="postgres")
        except (Exception, psycopg2.Error) as error :
            if(self.connection):
                print("Falha ao se conectar ao Banco de Dados", error)
```

Classe para operações CRUD.

> **Antes de continuar, lembre-se de que a indentação (espaçamento) faz parte da sintaxe do python.**

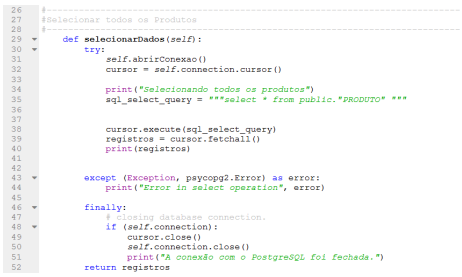
- Linha 12 - Declaramos a classe “AppBD”.
- Linha 13 - Implementamos o construtor da classe, que é o método que é chamado logo que um objeto da classe AppBD for instanciado.
- Linha 16 a 25 - Implementamos o método para abrir a conexão. Perceba as cláusulas “try” e “except”. Isso é fundamental para criar um programa confiável, ou seja, com tolerância a falhas, pois o programa tenta seguir o fluxo normal de execução, ou seja, abrir a conexão com o banco de

dados.

Comentário

Caso ocorra algum problema, ao invés de o programa interromper a execução e exibir uma mensagem de erro, revelando vulnerabilidades do sistema que podem ser exploradas por um atacante, ele vai exibir uma mensagem amigável para o usuário, no caso, será “Falha ao se conectar ao Banco de Dados”.

O próximo método é o que faz consulta no banco de dados:



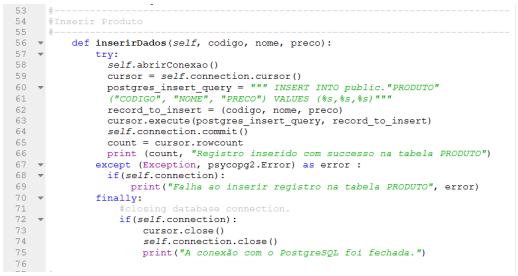
```
26  #------------------------------------------------------------------------------
27  #Selecionar todos os Produtos
28  #------------------------------------------------------------------------------
29      def selecionarDados(self):
30          try:
31              self.abrirConexao()
32              cursor = self.connection.cursor()
33
34              print("Selecionando todos os produtos")
35              sql_select_query = """select * from public."PRODUTO" """
36
37
38              cursor.execute(sql_select_query)
39              registros = cursor.fetchall()
40              print(registros)
41
42
43          except (Exception, psycopg2.Error) as error:
44              print("Error in select operation", error)
45
46          finally:
47              # closing database connection.
48              if (self.connection):
49                  cursor.close()
50                  self.connection.close()
51                  print("A conexão com o PostgreSQL foi fechada.")
52          return registros
```

Seleção de dados.

Vamos destacar os principais pontos do método “selecionarDados”.

- Linha 35 - Montamos a instrução de consulta do sql.
- Linha 38 - Executamos a instrução sql.
- Linha 39 - Recuperamos as linhas que retornaram da consulta sql.
- Linha 52 - Retornamos os registros para quem faz a chamada para o método “selecionarDados”.

Agora, vamos explicar o método para fazer a inserção de dados. O código do método está na imagem a seguir.



```
53  #------------------------------------------------------------------------------
54  #Inserir Produto
55  #------------------------------------------------------------------------------
56      def inserirDados(self, codigo, nome, preco):
57          try:
58              self.abrirConexao()
59              cursor = self.connection.cursor()
60              postgres_insert_query = """ INSERT INTO public."PRODUTO"
61              ("CODIGO", "NOME", "PRECO") VALUES (%s,%s,%s)"""
62              record_to_insert = (codigo, nome, preco)
63              cursor.execute(postgres_insert_query, record_to_insert)
64              self.connection.commit()
65              count = cursor.rowcount
66              print (count, "Registro inserido com sucesso na tabela PRODUTO")
67          except (Exception, psycopg2.Error) as error :
68              if(self.connection):
69                  print("Falha ao inserir registro na tabela PRODUTO", error)
70          finally:
71              #closing database connection.
72              if(self.connection):
73                  cursor.close()
74                  self.connection.close()
75                  print("A conexão com o PostgreSQL foi fechada.")
76
```

Inserção de dados.

- Linha 56 - Implementamos a função “inserirDados” e passamos como parâmetros os “codigo”, “nome” e “preco” que serão inseridos na tabela. Além disso, passamos o parâmetro “self”, que é usado para fazer referência aos atributos e métodos da própria classe.
- Linha 60 e 61 - Montamos a instrução sql para fazer a inserção dos dados.
- Linha 62 - Montamos o registro que será inserido na tabela.
- Linha 63 - Executamos a instrução sql para fazer a inserção do registro da variável “record_to_insert”.

Agora, vamos analisar o método responsável pela atualização de dados. O código é apresentado na imagem seguinte.

```
77  #------------------------------------------------------------------------------
78  #Atualizar Produto
79  #------------------------------------------------------------------------------
80      def atualizarDados(self, codigo, nome, preco):
81          try:
82              self.abrirConexao()
83              cursor = self.connection.cursor()
84
85              print("Registro Antes da Atualização ")
86              sql_select_query = """select * from public."PRODUTO"
87              where "CODIGO" = %s"""
88              cursor.execute(sql_select_query, (codigo,))
89              record = cursor.fetchone()
90              print(record)
91              # Atualizar registro
92              sql_update_query = """Update public."PRODUTO" set "NOME" = %s,
93              "PRECO" = %s where "CODIGO" = %s"""
94              cursor.execute(sql_update_query, (nome, preco, codigo))
95              self.connection.commit()
96              count = cursor.rowcount
97              print(count, "Registro atualizado com sucesso! ")
98              print("Registro Depois da Atualização ")
99              sql_select_query = """select * from public."PRODUTO"
100             where "CODIGO" = %s"""
101             cursor.execute(sql_select_query, (codigo,))
102             record = cursor.fetchone()
103             print(record)
104         except (Exception, psycopg2.Error) as error:
105             print("Erro na Atualização", error)
106         finally:
107             # closing database connection.
108             if (self.connection):
109                 cursor.close()
110                 self.connection.close()
111                 print("A conexão com o PostgreSQL foi fechada.")
112
```

Atualização dos dados.

- Linha 80 - Implementamos a função “atualizarDados” e passamos como parâmetros os “codigo”, “nome” e “preco” que serão modificados na tabela.
- Linha 92 e 93 - Montamos a instrução sql para fazer a modificação dos dados da tabela.
- Linha 94 - Executamos a instrução sql para fazer a modificação do registro de acordo com a tupla “(nome, preco, codigo)”.

Agora, vamos analisar o método que trata da exclusão de dados. O código é apresentado na imagem a seguir.

```
113 #---------------------------------------------------------------------------
114 #Excluir Produto
115 #---------------------------------------------------------------------------
116     def excluirDados(self, codigo):
117         try:
118             self.abrirConexao()
119             cursor = self.connection.cursor()
120             # Atualizar registro
121             sql_delete_query = """Delete from public."PRODUTO"
122             where "CODIGO" = %s"""
123             cursor.execute(sql_delete_query, (codigo, ))
124 
125             self.connection.commit()
126             count = cursor.rowcount
127             print(count, "Registro excluído com sucesso! ")
128         except (Exception, psycopg2.Error) as error:
129             print("Erro na Exclusão", error)
130         finally:
131             # closing database connection.
132             if (self.connection):
133                 cursor.close()
134                 self.connection.close()
135                 print("A conexão com o PostgreSQL foi fechada.")
136 
```

Exclusão de dados.

- Linha 116 - Implementamos a função “excluirDados” e passamos o “codigo” do produto que será excluído da tabela.
- Linha 121 e 122 - Montamos a instrução sql para fazer a exclusão dos dados da tabela.
- Linha 123 - Executamos a instrução sql para fazer a exclusão do registro de acordo com o parâmetro “codigo” passado para o método.

GUI: Interação com o usuário

Interface gráfica

Nesta seção, vamos apresentar a parte do sistema responsável pela interação com o usuário por meio de uma interface gráfica.

> **Do mesmo modo que a classe “AppBD”, esse programa também foi desenvolvido em programação orientada a objetos.**

O arquivo do programa foi salvo com o nome **“aplicacaoCRUD.py”**. A classe com os atributos e métodos para trabalhar com a interface gráfica e interagir com a classe responsável pelas operações com o banco de dados é a “PrincipalBD”, conforme podemos ver na imagem a seguir.

```
 9 import tkinter as tk
10 from tkinter import ttk
11 import crud as crud
12 
13 class PrincipalBD:
14     def __init__(self, win):
15         self.objBD = crud.AppBD()
16         #componentes
17         self.lbCodigo=tk.Label(win, text='Código do Produto:')
18         self.lblNome=tk.Label(win, text='Nome do Produto')
19         self.lblPreco=tk.Label(win, text='Preço')
20 
21         self.txtCodigo=tk.Entry(bd=3)
22         self.txtNome=tk.Entry()
23         self.txtPreco=tk.Entry()
24         self.btnCadastrar=tk.Button(win, text='Cadastrar', command=self.fCadastrarProduto)
25         self.btnAtualizar=tk.Button(win, text='Atualizar', command=self.fAtualizarProduto)
26         self.btnExcluir=tk.Button(win, text='Excluir', command=self.fExcluirProduto)
27         self.btnLimpar=tk.Button(win, text='Limpar', command=self.fLimparTela)
```

Classe principal.

- Linha 9 - Importamos a biblioteca Tkinter para interagir com os componentes gráficos.
- Linha 10 - Importamos o módulo ttk para podermos trabalhar com o componente “TreeView”, que foi usado como uma grade para exibir os dados armazenados na tabela “PRODUTO”.
- Linha 14 - Implementamos o construtor (__init__) da classe PrincipalBD, que será chamado logo que um objeto do tipo PrincipalBD for instanciado.
- Linha 17 a 23 - Instanciamos os componentes rótulos (“label”) e caixas de texto (“entry”).

- Linha 24 a 27 - Instanciamos os componentes botões (“button”), que vão acionar as operações CRUD.

Atenção!

Observe que os métodos “fCadastrarProduto”, “fAtualizarProduto”, “fExcluirProduto” e “fLimparProduto” estão vinculados aos botões, ou seja, quando o usuário pressionar um botão, o respectivo método será chamado.

O construtor ainda possui mais duas partes. Uma delas é responsável por instanciar e configurar o componente “TreeView”, conforme podemos ver na próxima imagem.

```
28      #----- Componente TreeView ------------------------------------------------
29      self.dadosColunas = ("Código", "Nome", "Preço")
30
31      self.treeProdutos = ttk.Treeview(win,
32                                       columns=self.dadosColunas,
33                                       selectmode='browse')
34
35      self.verscrlbar = ttk.Scrollbar(win,
36                                       orient="vertical",
37                                       command=self.treeProdutos.yview)
38      self.verscrlbar.pack(side ='right', fill ='x')
39
40      self.treeProdutos.configure(yscrollcommand=self.verscrlbar.set)
41
42      self.treeProdutos.heading("Código", text="Código")
43      self.treeProdutos.heading("Nome", text="Nome")
44      self.treeProdutos.heading("Preço", text="Preço")
45
46      self.treeProdutos.column("Código",minwidth=0,width=100)
47      self.treeProdutos.column("Nome",minwidth=0,width=100)
48      self.treeProdutos.column("Preço",minwidth=0,width=100)
49
50      self.treeProdutos.pack(padx=10, pady=10)
51
52      self.treeProdutos.bind("<<TreeviewSelect>>",
53                             self.apresentarRegistrosSelecionados)
```

Configuração do componente Treeview.

> **Observe nas linhas 52 e 53 que o método “apresentarRegistrosSelecionados” é vinculado à instância do componente “TreeView”. Esse método será explicado mais à frente.**

E a outra posiciona os componentes na tela, conforme podemos ver na imagem seguinte.

```
54      #------------------------------------------------------------------
55      #posicionamento dos componentes na janela
56      #------------------------------------------------------------------
57      self.lbCodigo.place(x=100, y=50)
58      self.txtCodigo.place(x=250, y=50)
59
60      self.lblNome.place(x=100, y=100)
61      self.txtNome.place(x=250, y=100)
62
63      self.lblPreco.place(x=100, y=150)
64      self.txtPreco.place(x=250, y=150)
65
66      self.btnCadastrar.place(x=100, y=200)
67      self.btnAtualizar.place(x=200, y=200)
68      self.btnExcluir.place(x=300, y=200)
69      self.btnLimpar.place(x=400, y=200)
70
71      self.treeProdutos.place(x=100, y=300)
72      self.verscrlbar.place(x=805, y=300, height=225)
73      self.carregarDadosIniciais()
```

Configuração de componentes.

> **Observe que, na linha 73, fazemos chamada para o método “carregarDadosIniciais”. Mais à frente, vamos explicá-lo com mais detalhes.**

Agora, vamos analisar o método “apresentarRegistrosSelecionados”, conforme podemos ver na imagem seguinte.

```
74  #------------------------------------------------------------------------------
75      def apresentarRegistrosSelecionados(self, event):
76          self.fLimparTela()
77          for selection in self.treeProdutos.selection():
78              item = self.treeProdutos.item(selection)
79              codigo,nome,preco = item["values"][0:3]
80              self.txtCodigo.insert(0, codigo)
81              self.txtNome.insert(0, nome)
82              self.txtPreco.insert(0, preco)
```

Exibir dados selecionados no Treeview.

Este método exibe os dados selecionados na grade (componente “TreeView”) nas caixas de texto, de modo que o usuário possa fazer alterações, ou exclusões sobre eles.

- Linha 76 - Fazemos a chamada para a função “fLimparTela”, que limpa o conteúdo das caixas de texto.
- Linha 77 - Obtemos os registros que foram selecionados na grade de registros.
- Linha 79 - Os dados do item selecionados são, agora, associados às variáveis “codigo”, “nome” e “preco”.

- Linha 80 a 82 - Os valores das variáveis são associados às caixas de texto.

Agora, vamos analisar o método “carregarDadosIniciais”, que é apresentado na imagem a seguir.

```
83  #------------------------------------------------------------------------
84  def carregarDadosIniciais(self):
85      try:
86        self.id = 0
87        self.iid = 0
88        registros=self.objBD.selecionarDados()
89        print("********** dados dsponíveis no BD **********")
90        for item in registros:
91            codigo=item[0]
92            nome=item[1]
93            preco=item[2]
94            print("Código = ", codigo)
95            print("Nome = ", nome)
96            print("Preço  = ", preco, "\n")
97
98            self.treeProdutos.insert('', 'end',
99                                     iid=self.iid,
100                                    values=(codigo,
101                                            nome,
102                                            preco))
103           self.iid = self.iid + 1
104           self.id = self.id + 1
105       print('Dados da Base')
106   except:
107     print('Ainda não existem dados para carregar')
```

Carregar dados da tabela no Treeview.

> **Este método carrega os dados que já estão armazenados na tabela para serem exibidos na grade de dados (componente “TreeView”).**

- Linha 86 e 87 - Os atributos “id” e “iid” são iniciados com valor 0. Eles são necessários para gerenciar o componente “TreeView”.
- Linha 88 - É feita a chamada para o método “selecionarDados” que está na classe “AppBD”. Ele recupera todos os registros armazenados na tabela.
- Linha 91 e 93 - Obtemos os valores dos registros e associamos às respectivas variáveis.
- Linha 98 a 102 - Os dados são adicionados ao componente “TreeView”.

Agora, vamos apresentar o método “fLerCampos”, conforme podemos ver na imagem abaixo.

```
108 #------------------------------------------------------------------------
109 #LerDados da Tela
110 #------------------------------------------------------------------------
111     def fLerCampos(self):
112         try:
113           print("********** dados dsponíveis **********")
114           codigo = int(self.txtCodigo.get())
115           print('codigo', codigo)
116           nome=self.txtNome.get()
117           print('nome', nome)
118           preco=float(self.txtPreco.get())
119           print('preco', preco)
120           print('Leitura dos Dados com Sucesso!')
121         except:
122           print('Não foi possível ler os dados.')
123         return codigo, nome, preco
```

Entrada de dados.

Este método lê os dados que estão nas caixas de texto e os retorna para quem faz a chamada.

Por exemplo, na linha 114, a variável “codigo” recebe o valor da caixa de texto “txtCodigo” depois que ele é convertido para um valor do tipo “inteiro”.

Na linha 123, as variáveis “codigo”, “nome” e “preco” retornam para quem faz a chamada do método.

Agora, vamos apresentar o método “fCadastrarProduto”, conforme podemos ver na imagem a seguir.

```
124 #------------------------------------------------------------------------
125 #Cadastrar Produto
126 #------------------------------------------------------------------------
127     def fCadastrarProduto(self):
128         try:
129           print("********** dados dsponíveis **********")
130           codigo, nome, preco= self.fLerCampos()
131           self.objBD.inserirDados(codigo, nome, preco)
132           self.treeProdutos.insert('', 'end',
133                                    iid=self.iid,
134                                    values=(codigo,
135                                            nome,
136                                            preco))
137           self.iid = self.iid + 1
138           self.id = self.id + 1
139           self.fLimparTela()
140           print('Produto Cadastrado com Sucesso!')
141         except:
142           print('Não foi possível fazer o cadastro.')
```

Inserção de dados no banco.

Este método tem como objetivo fazer a inserção dos dados na tabela “PRODUTOS”.

- Linha 130 - Os dados digitados nas caixas de texto são recuperados nas variáveis “codigo”, “nome” e “preco”.
- Linha 131 - Fazemos a chamada ao método “inserirDados”, que fará a inserção dos dados na tabela “PRODUTO”.

- Linha 132 a 136 - Os dados são inseridos no componente grade (“TreeView”).

Agora, vamos analisar o método “fAtualizarProduto”, conforme podemos ver na próxima imagem.

```
143 #-----------------------------------------------------------------------------
144 #Atualizar Produto
145 #-----------------------------------------------------------------------------
146     def fAtualizarProduto(self):
147         try:
148           print("************ dados dsponíveis ***********")
149           codigo, nome, preco= self.fLerCampos()
150           self.objBD.atualizarDados(codigo, nome, preco)
151           #recarregar dados na tela
152           self.treeProdutos.delete(*self.treeProdutos.get_children())
153           self.carregarDadosIniciais()
154           self.fLimparTela()
155           print('Produto Atualizado com Sucesso!')
156         except:
157           print('Não foi possível fazer a atualização.')
```

Atualização de dados no banco.

> O objetivo deste método é atualizar os dados que o usuário selecionou na grade de dados (o componente “TreeView”).

- Linha 149 - Os dados selecionados da grade de dados são recuperados nas variáveis “codigo”, “nome” e “preco”.
- Linha 150 - Chamamos a função “atualizarDados”, que fará as modificações dos dados na tabela “PRODUTO”.
- Linha 152 - Os dados selecionados são removidos da grade de dados.
- Linha 153 - Fazemos a chamada ao método “carregarDadosIniciais” para recarregar a grade de dados com os dados da tabela.

Agora, vamos analisar o método “fExcluirProduto”, conforme podemos ver na imagem seguinte.

```
158 #-----------------------------------------------------------------------------
159 #Excluir Produto
160 #-----------------------------------------------------------------------------
161     def fExcluirProduto(self):
162         try:
163           print("************ dados dsponíveis ***********")
164           codigo, nome, preco= self.fLerCampos()
165           self.objBD.excluirDados(codigo)
166           #recarregar dados na tela
167           self.treeProdutos.delete(*self.treeProdutos.get_children())
168           self.carregarDadosIniciais()
169           self.fLimparTela()
170           print('Produto Excluído com Sucesso!')
171         except:
172           print('Não foi possível fazer a exclusão do produto.')
```

Exclusão de dados no banco.

O objetivo deste método é excluir os dados que o usuário selecionou na grade de dados (o componente “TreeView”).

- Linha 164 - Os dados selecionados da grade de dados são recuperados nas variáveis “codigo”, “nome” e “preco”.
- Linha 165 - Chamamos a função “excluirDados”, que excluirá os dados da tabela “PRODUTO”.
- Linha 167 - Os dados selecionados são removidos da grade de dados.
- Linha 168 - Fazemos a chamada ao método “carregarDadosIniciais” para recarregar a grade de dados com os dados da tabela.

Agora, vamos analisar o método “fLimparTela”, conforme podemos ver na imagem a seguir.

```
173 #-----------------------------------------------------------------------------
174 #Limpar Tela
175 #-----------------------------------------------------------------------------
176     def fLimparTela(self):
177         try:
178           print("************ dados dsponíveis ***********")
179           self.txtCodigo.delete(0, tk.END)
180           self.txtNome.delete(0, tk.END)
181           self.txtPreco.delete(0, tk.END)
182           print('Campos Limpos!')
183         except:
184           print('Não foi possível limpar os campos.')
```

Limpar os componentes da tela.

> Este método limpa o conteúdo das caixas de texto, conforme podemos ver nas linhas 179 a 181.

Por fim, apresentamos o programa principal que vai iniciar a execução do sistema, conforme podemos ver na próxima imagem.

```
185  #-------------------------------------------------------------------------
186  #Programa Principal
187  #-------------------------------------------------------------------------
188  janela=tk.Tk()
189  principal=PrincipalBD(janela)
190  janela.title('Bem Vindo a Aplicação de Banco de Dados')
191  janela.geometry("820x600+10+10")
192  janela.mainloop()
```

Programa principal.

- Linha 188 - Instanciamos o objeto raiz da aplicação gráfica que chamamos de “janela”.
- Linha 189 - Instanciamos o objeto principal que vai gerenciar a execução da aplicação.
- Linha 190 - Escrevemos uma mensagem para o título da janela.
- Linha 192 - Configuramos as dimensões da janela.
- Linha 192 - Chamamos o método “mainloop”, que coloca a aplicação para executar até que o usuário interrompa a execução.

Finalização da construção do protótipo

Acompanhe a finalização da construção do protótipo do sistema utilizando a linguagem python, uma interface GUI e a conexão ao banco de dados.

Para assistir a um vídeo sobre o assunto, acesse a versão online deste conteúdo.

Falta pouco para atingir seus objetivos.

Vamos praticar alguns conceitos?

Questão 1

Considere o fragmento de código Python abaixo que utiliza a biblioteca “psycopg2” para fazer CRIAR a tabela PRODUTOv2 no PostgreSQL:

Python

```
1  import psycopg2
2  lacuna_I
3  conn = psycopg2.connect(database = "postgres", user = "postgres", password = "senha123", host = "127.0.0.1", pc
4  print("Conexão com o Banco de Dados aberta com sucesso!")
5  cur = conn.cursor()
6  cur.execute('''CREATE TABLE PRODUTOv2
```

```
(CODIGO INT PRIMARY KEY NOT NULL,
NOME TEXT NOT NULL,
PRECO CHAR(12));''')
print("Tabela criada com sucesso!")
lacuna_II
conn.commit()
conn.close()
```

Selecione a opção correta, para que o programa execute corretamente:

A As palavras lacuna_I e lacuna_II devem ser substituídas por “#” (símbolo de comentário), pois não afetam a execução do programa.

B O tipo campo “PRECO” deve ser modificado para “REAL NOT NULL”, caso contrário vai ocorrer um erro na criação da tabela.

C As palavras lacuna_I e lacuna_II devem ser substituídas por “try:” e “except (Exception, psycopg2.Error) as error:” para prevenir a ocorrência de exceções.

D A tabela deve ser renomeada para “PRODUTO”, ao invés de “PRODUTOv2”.

E As palavras lacuna_I e lacuna_II devem ser substituídas por “Begin”, para marcar o início do programa, e “End”, para indicar o fim do trecho sql.

Parabéns! A alternativa C está correta.

%0A%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%3Cp%20class%3D'c-paragraph'%3EApesar%20de%20n%C3%A3o%20serem%20obrigat%C3%B3rias%2C%20as%20cl%C3%A1usulas%20%E2%80%9Ctry-except%E2%80%9D%20previnem%20a%20ocorr%C3%AAncia%20de%20exce%C3%A7%C3%B5es%2C%20evitando%20que%20o%20programa%20seja

Questão 2

Durante a execução de um programa em Python podem ocorrer problemas que, se não forem tratados adequadamente, vão interromper o programa e exibir uma mensagem de erro para o usuário sem nenhum tipo de tratamento. Nesse sentido, selecione a opção que apresenta a forma adequada de tratar exceções no Python:

A if-else

B try-except

C if-elif

D try-catch

E if-else

Parabéns! A alternativa B está correta.

%0A%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%3Cp%20class%3D'c-paragraph'%3EA%20cl%C3%A1usula%20%E2%80%9Ctry%E2%80%9D%20tenta%20executar%20o%20fluxo%20normal%20do%20programa.%20Caso%2
except%E2%80%9D%2C%20mas%20trata-
se%20de%20uma%20boa%20pr%C3%A1tica%20de%20programa%C3%A7%C3%A3o.%3C%2Fp%3E%0A%20%20%20%20%20%20%20%20%20%20%20%2

Considerações finais

No decorrer do texto, apresentamos os principais frameworks e as bibliotecas para desenvolver aplicações de interface gráfica, exploramos a biblioteca “tkinter” com exemplos para usar os seus componentes gráficos e também a biblioteca “psycopg2”, para realizar operações no PostgreSQL, além de apresentarmos uma aplicação que relaciona interface gráfica com operações no banco de dados.

A linguagem python facilita bastante o desenvolvimento de aplicações de interface gráfica e que façam operações no banco de dados. É natural que o desenvolvedor iniciante tenha a impressão de que é um projeto complexo, mas, com um pouco de prática, as vantagens de programar nesse ambiente vão ficar evidentes.

O python é uma linguagem muito bem documentada e possui uma comunidade engajada em resolver problemas e disponibilizar as soluções em ambientes públicos. Exploramos algumas bibliotecas, mas existem muitas outras. A escolha de uma delas deve levar em consideração questões como tempo para desenvolvimento, maturidade do programador e quais os recursos computacionais que se têm à disposição.

Ouça agora sobre o processo de desenvolvimento de interfaces gráficas com a linguagem Python.

Para ouvir o *áudio*, acesse a versão online deste conteúdo.

Referências

MEIER, B. A. **python GUI Programming Cookbook**. Birmingham, United Kingdom: Packt Publishing Ltd., 2015.

python. **python 3.8.5 documentation**. python.org. Publicado em 20 jul. 2020.

python. **Tkinter – python interface to Tcl/Tk**. docs.python.org. Consultado em 23 out. 2020.

SPYDER. **The Scientific python Development Environment**. [s.d.] spyder-ide.org. Consultado em 23 out. 2020.

Explore +

Para saber mais sobre os assuntos tratados neste tema, pesquise na internet:

- Os sites oficiais do tkinter e do psycopg2 e aprenda mais detalhes técnicos sobre aplicações GUI e de banco de dados, além de poder fazer downloads.
- Os sites oficiais das bibliotecas/frameworks/pacotes Kivy, Pyforms, PyQt, wxpython, PyAutoGUI e PySimpleGUI para obter mais informações sobre aplicações GUI no python.
- Os sites oficiais das bibliotecas/frameworks/pacotes pyMySQL, cx_Oracle, PySqlite e PyMongo para obter mais informações sobre aplicações de banco de dados no python.