Interface gráfica com python

Neste conteúdo, você aprenderá a desenvolver aplicações com interface gráfica (GUI) utilizando Python e Tkinter, integrando operações CRUD (create, read, update, delete) com um banco de dados PostgreSQL por meio da biblioteca psycopg2. Serão abordados tópicos como criação de tabelas, inserção de dados aleatórios e a construção de uma GUI interativa e eficiente. Esse conhecimento é importante para criar aplicações fáceis de usar e que interagem de forma eficaz com o banco de dados.

Prof. Sérgio Assunção Monteiro e Prof. Kleber de Aguiar

Objetivos

- Identificar alguns dos principais frameworks e as bibliotecas necessárias para a GUI.
- Descrever a adição de widgets e montagem da interface gráfica.
- Definir a interface para inclusão de dados em uma tabela no 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.

Neste vídeo, vamos abordar os principais tópicos sobre interfaces gráficas com Python. Vamos falar sobre o desenvolvimento de componentes de GUI (Interface Gráfica de Usuário), a criação de aplicações que interagem com bancos de dados e como integrar essas duas partes, garantindo que as ações dos usuários nos componentes resultem em comandos para o banco de dados. Não deixe de conferir!

Conteúdo interativo

Acesse a versão digital para assistir ao vídeo.

Frameworks e bibliotecas para GUI (Graphical User Interface)

Conhecer os principais frameworks e bibliotecas para GUI (Graphical User Interface) em Python é essencial para criar aplicações interativas e atraentes. Bibliotecas como Tkinter, PyQt e Kivy permitem a criação de interfaces ricas e funcionais. Dominar esses recursos amplia as possibilidades de desenvolvimento e oferece uma vantagem competitiva, especialmente com a crescente demanda por aplicações com interfaces intuitivas e responsivas.

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:

Framework e biblioteca para GUI

Neste vídeo, vamos explorar os principais frameworks para GUI em Python, como Tkinter, PyQt e Kivy. Vamos discutir suas características, vantagens e como cada um pode ser usado para criar interfaces gráficas ricas e interativas. Acompanhe para descobrir qual framework é mais adequado para o seu projeto e como começar a utilizá-los. Assista!

Conteúdo interativo

Acesse a versão digital para assistir ao vídeo.

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:

```
plain-text
pip install tkinter
```

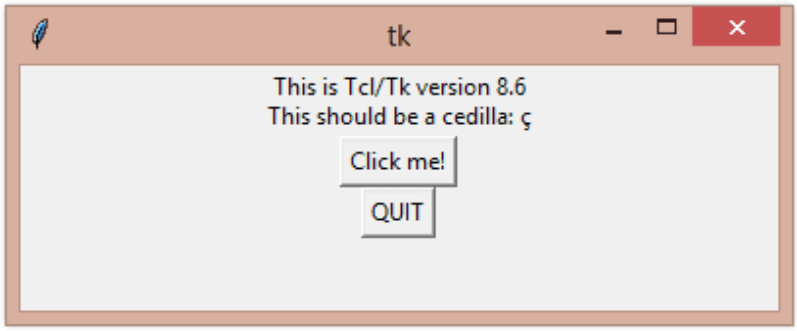
Para quem usa a IDE Spyder (SPYDER, 2020), é necessário colocar uma exclamação antes do comando “pip”, ou seja:

```
plain-text
!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:

```
plain-text

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”.

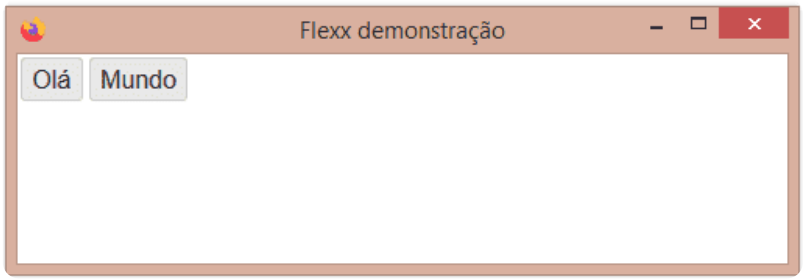
No caso do Spyder, o comando deve ser:

```
plain-text

!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')

if __name__ == '__main__':
    a = flx.App(Exemplo, title='Flexx demonstração')
    m = a.launch()
    flx.run()
```

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:

```
python

pip install cefpython3
```

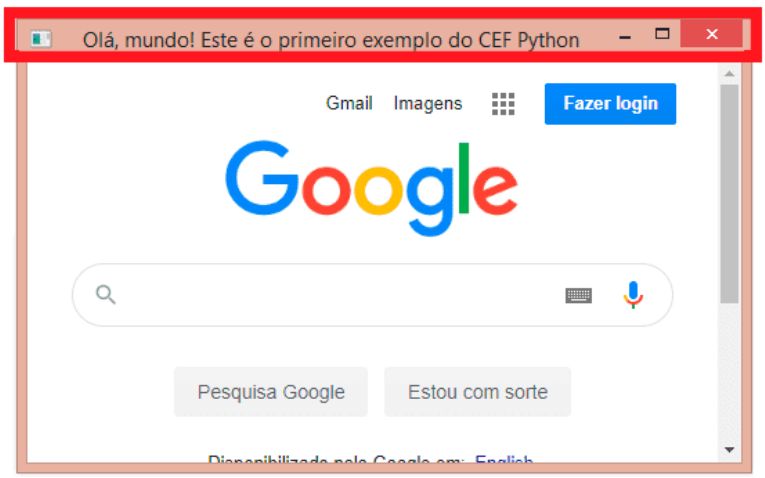
Para testar a instalação do CEF python, basta escrever o código abaixo na linha de comando, ou em um arquivo, e executar:

```
python

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
    cef.Initialize()
    cef.CreateBrowserSync(url=" https://www.google.com/",window_title="Olá, mundo! Este é
o primeiro exemplo do CEF python")
    cef.MessageLoop()
    cef.Shutdown()

def check_versions():
    ver=cef.GetVersion()
    print("[hello_world.py] CEF python {ver}".format(ver=ver["version"]))
    print("[hello_world.py] Chromium {ver}".format(ver=ver["chrome_version"]))
    print("[hello_world.py] CEF {ver}".format(ver=ver["cef_version"]))
    print("[hello_world.py] python {ver}
    {arch}".format(ver=platform.python_version(),arch=platform.architecture()[0])) assert
    cef.__version__>= "57.0", "CEF python v57.0+ required to run this"


if __name__ == '__main__':
    main()
```

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.

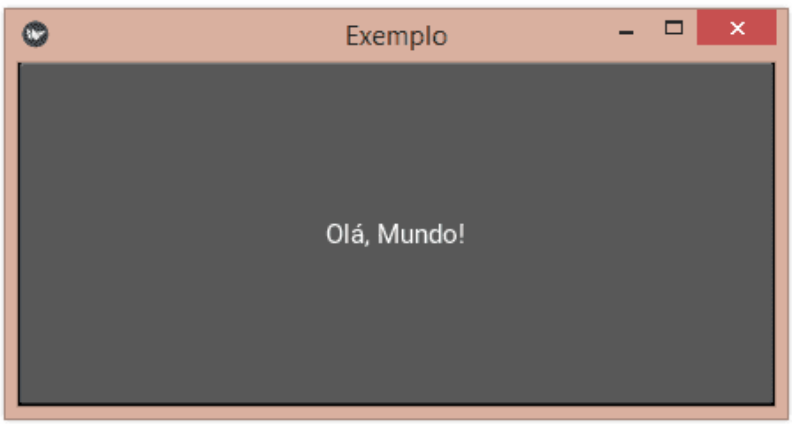
Para instalá-lo, é necessário escrever na linha de comando:

```
plain-text

pip install Kivy
```

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

```
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.

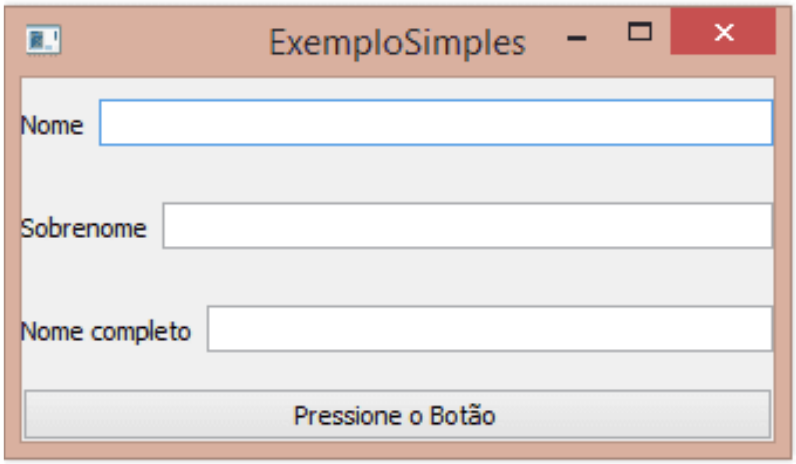
Essas camadas podem ser usadas individualmente ou em conjunto, dependendo da instalação do Pyforms. Para fazer a instalação básica, é necessário escrever na linha de comando:

```
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):

    def __init__(self):
        super(ExemploSimples,self).__init__('ExemploSimples')
        #Definition of the forms fields
        self._nome = ControlText('Nome', 'Default value')
        self._sobrename = ControlText('Sobrenome')
        self._nomeCompleto = ControlText('Nome completo')
        self._button = ControlButton('Pressione o Botão')


#Execute the application
if __name__ == " __main__":
    from pyforms import start_app
    start_app(ExemploSimples)
```

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.

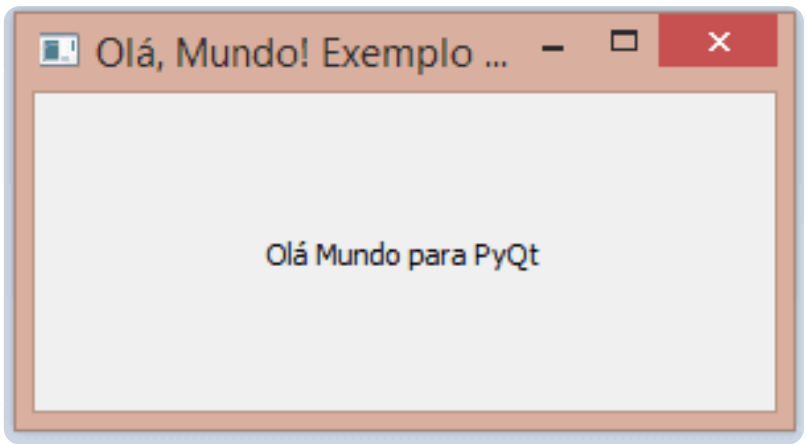
Para fazer a instalação básica, é necessário escrever na linha de comando:

```plain-text
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)
      gridLayout.addWidget(title, 0, 0)

if __name__ == "__main__":
   app = QtWidgets.QApplication(sys.argv)
   mainWin = HelloWindow()
   mainWin.show()
   sys.exit( app.exec_() )
```

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.

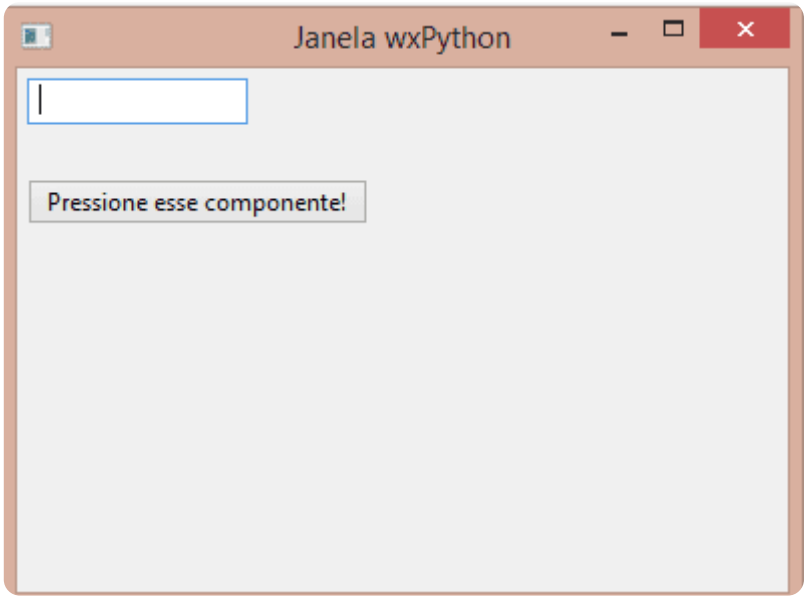
Para fazer a instalação básica, é necessário escrever na linha de comando:

```
plain-text

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))


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


app = ExemploApp()
app.MainLoop()
```

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:

```plain-text
!pip install PyAutoGUI
```

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

```
python

import pyautogui
screenWidth, screenHeight = pyautogui.size()
currentMouseX, currentMouseY = pyautogui.position()
pyautogui.moveTo(100, 150)
pyautogui.click()
pyautogui.click(100, 200)
pyautogui.move(0, 10)
pyautogui.doubleClick()
pyautogui.moveTo(500, 500, duration=2, tween=pyautogui.easeInOutQuad)
pyautogui.write('Olá, Mundo!', interval=0.25)
pyautogui.press('esc')
pyautogui.keyDown('shift')
pyautogui.press(['left', 'left', 'left', 'left'])
pyautogui.keyUp('shift')
pyautogui.hotkey('ctrl', 'c')
pyautogui.alert('Esta é a mensagem para Tela.')
```

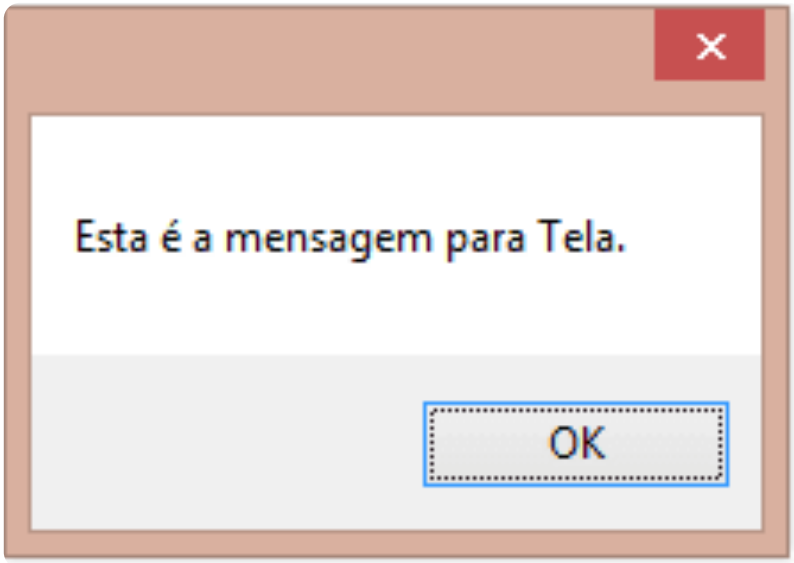
Nesse código, basicamente tem-se esta sequência de instruções:

- 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.

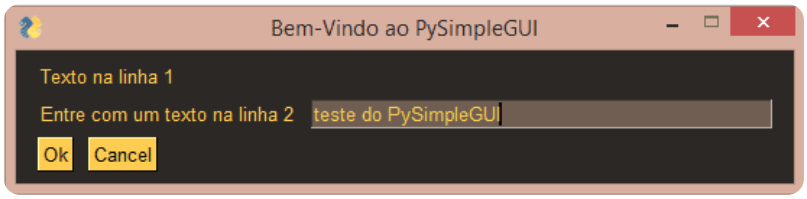
Para fazer a instalação básica, é necessário escrever na linha de comando:

```
plain-text

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()],
  [sg.Button('Ok'), sg.Button('Cancel')] ]
  window = sg.Window('Bem-Vindo ao PySimpleGUI', layout)

while True:
  event, values = window.read()
  if event == sg.WIN_CLOSED or event == 'Cancel':
    break
  print('Você entrou com: ', values[0])

window.close()
```

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.

Atividade 1

Ao desenvolver uma aplicação de interface gráfica em Python, você deseja escolher um framework que ofereça suporte abrangente para desenvolvimento tanto para desktop quanto para aplicações web. Qual dos seguintes frameworks seria a escolha mais adequada para atender a essa necessidade?

A

Tkinter

B

PyQt

C

Kivy

D

Flexx

E

PySimpleGUI

A alternativa D está correta.

Flexx é especialmente projetado para desenvolver interfaces gráficas que funcionam tanto em desktop quanto em aplicações web, utilizando tecnologia web para renderização. Isso o torna ideal para projetos que precisam de portabilidade entre diferentes plataformas.

Interfaces GUI

Para desenvolver software de forma eficiente, é necessário entender as vantagens e desvantagens das interfaces gráficas (GUI). As GUIs facilitam interações intuitivas, permitindo que os usuários acessem funcionalidades complexas por meio de componentes visuais simples. Porém, elas também podem exigir muitos recursos e aumentar a complexidade do desenvolvimento. Avaliar esses prós e contras ajuda os desenvolvedores a escolher a melhor abordagem, garantindo aplicativos que sejam tanto usáveis quanto otimizados em performance e segurança.

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.

Interfaces GUI

Neste vídeo, vamos falar sobre as vantagens e desvantagens das interfaces gráficas (GUI) no desenvolvimento de software. Vamos mostrar como as GUIs podem melhorar a interação e a experiência do usuário, mas também discutir os desafios, como o uso de recursos e a complexidade. Assista!

> **Conteúdo interativo**
>
> Acesse a versão digital para assistir ao vídeo.

Atividade 2

Considerando a implementação de interfaces gráficas em projetos de software, qual das seguintes afirmações melhor descreve uma desvantagem comum das aplicações GUI?

A

Redução do tempo de desenvolvimento devido à simplificação da interface.

B

Maior consumo de recursos computacionais.

C

Menor necessidade de documentação.

D

Aumento da portabilidade entre diferentes sistemas operacionais.

E

Facilidade na implementação de funcionalidades de rede.

A alternativa B está correta.

Aplicações GUI tendem a consumir mais recursos computacionais, como memória e processamento, especialmente quando comparadas a aplicações baseadas em texto, devido à complexidade dos elementos visuais e interativos que necessitam de renderização e gerenciamento.

Verificando o aprendizado

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.

A alternativa A está correta.

O comando “pip” é um dos gerenciadores de dependências do Python. Ele facilita a instalação de pacotes, mas possui algumas pequenas variações de acordo com o ambiente de desenvolvimento. Por exemplo, no Spyder, o comando “pip” deve ser precedido do símbolo de exclamação “!”.

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().

A alternativa E está correta.

De fato, a biblioteca Tkinter é a mais importante do Python por estar disponível na instalação padrão do Python, no entanto, apesar da facilidade de aplicação, ela não é adequada para aplicações específicas, como jogos.

Biblioteca Tkinter

Integrante padrão do Python, esse biblioteca é fundamental para quem quer começar a desenvolver interfaces gráficas de usuário (GUI). Com sua sintaxe simples e muitos widgets disponíveis, ela facilita a criação de aplicações intuitivas e eficientes. Além disso, sua compatibilidade multiplataforma e a extensa documentação disponível tornam Tkinter uma escolha acessível para muitos, especialmente para aqueles que estão começando a explorar o desenvolvimento de GUIs em Python. Conhecê-la é um passo importante para criar aplicações visuais interativas.

Neste vídeo, mostraremos como você pode usar o Tkinter para deixar suas aplicações Python mais atraentes e interativas, começando pela instalação e configuração até a criação de widgets, tanto básicos quanto avançados. Confira!

Conteúdo interativo

Acesse a versão digital para assistir ao vídeo.

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:

1. Importar a biblioteca Tkinter.

2. Criar a janela principal do programa GUI.

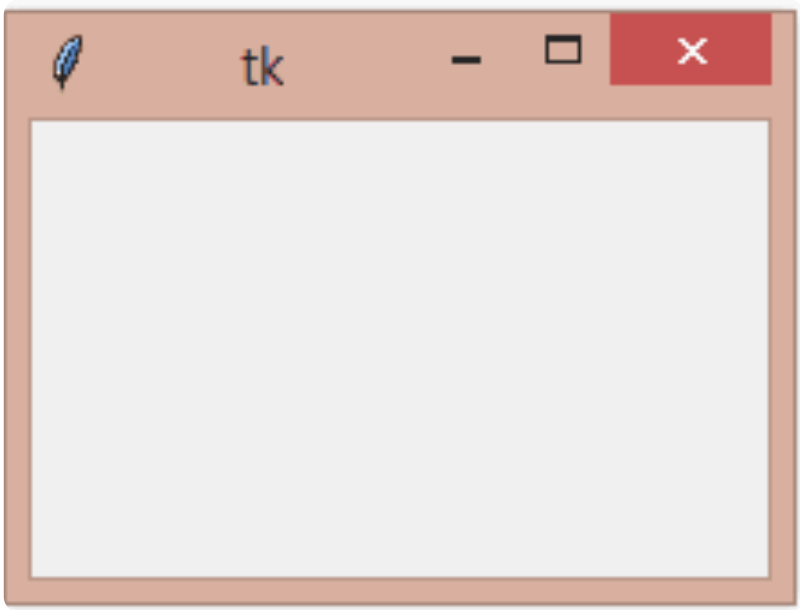
3. Adicionar um ou mais dos widgets (componentes).

4. 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.

Atividade 1

Ao desenvolver uma aplicação GUI em Python para gerenciamento de tarefas, você decide usar a biblioteca Tkinter devido à sua integração direta com Python. Qual das seguintes características do Tkinter facilita a criação de uma interface intuitiva para os usuários?

A

Suporte nativo para multithreading.

B

Integração com bases de dados SQL.

C

Ampla gama de widgets interativos.

D

Capacidades avançadas de renderização 3D.

E

Suporte integrado para criptografia de dados.

A alternativa C está correta.

Tkinter oferece uma variedade de widgets, como botões, menus e caixas de texto, que são essenciais para criar interfaces gráficas interativas e fáceis de usar, tornando-o ideal para aplicações como gerenciadores de tarefas.

Exemplos de aplicações GUI

Exemplos práticos de aplicações GUI são importantes para mostrar como os conceitos de design de interface funcionam na prática. Com exemplos como formulários, menus de navegação e diálogos interativos, os programadores podem entender melhor a integração dos componentes GUI e usar esses exemplos como base para seus próprios projetos. Esses exemplos também podem ser usados como base para experimentação e adaptação em projetos próprios, ajudando a entender melhor as capacidades e limitações das ferramentas de desenvolvimento de interfaces gráficas.

Confira este vídeo tutorial sobre como criar interfaces gráficas de usuário (GUI) em Python. Nele, você aprenderá a implementar widgets como janelas, botões e caixas de texto, além de explorar diferentes formas de adicionar interatividade às suas aplicações.

Conteúdo interativo

Acesse a versão digital para assistir ao vídeo.

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.

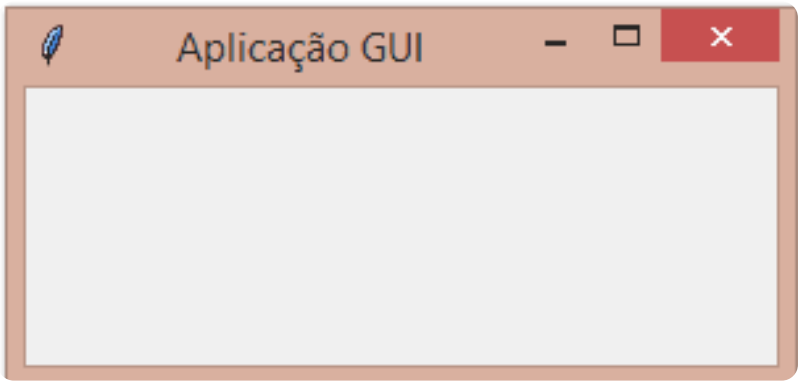
Vamos apresentar exemplos dos componentes:

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()
```

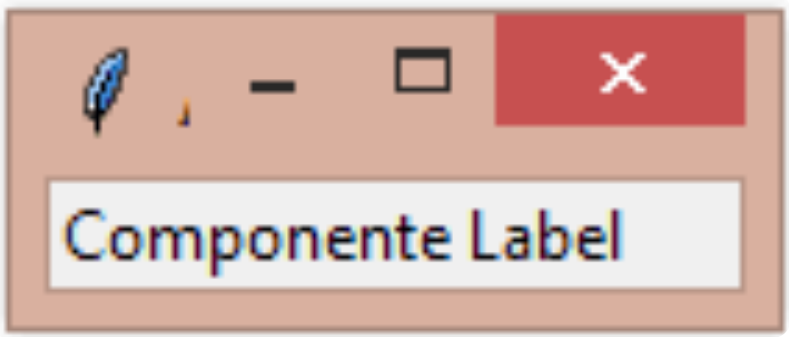
> ⚠
>
> Atenção
>
> A principal diferença desse exemplo em relação ao anterior está na linha 4, onde a propriedade de redimensionar a janela é colocada como “Falso”.

Widget Label

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

```python
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

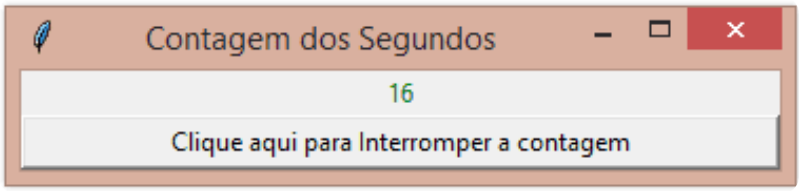
O próximo exemplo apresenta como usar o widget “Button”.

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

```
python

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.

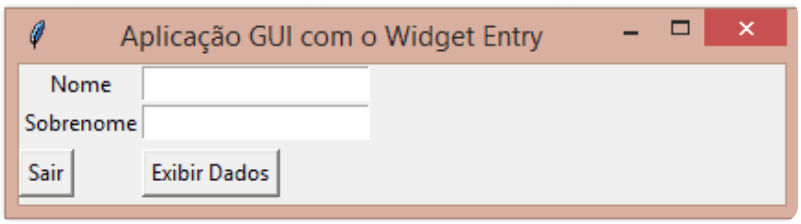
Widget Entry

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

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")
tk.Label(janela,text="Nome").grid(row=0)

tk.Label(janela,text="Sobrenome").grid(row=1)
e1 = tk.Entry(janela)
e2 = tk.Entry(janela)
e1.grid(row=0, column=1)
e2.grid(row=1, column=1)
tk.Button(janela, text='Sair',command=janela.quit).grid(row=3,column=0,sticky=tk.W,pady=4)
tk.Button(janela, text='Exibir Dados',
command=mostrar_nomes).grid(row=3,column=1,sticky=tk.W,pady=4)
tk.mainloop()
```

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.

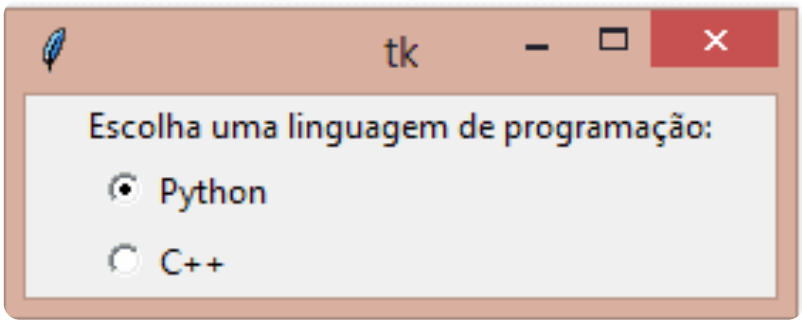
- Linha 2 - É implementada a função “mostrar_nomes”, que vai exibir na linha de comando os nomes que estão escritos nas instâncias “e1” e “e2” do componente “entry”.
- Linha 8 e 9 - São feitas as instâncias “e1” e “e2” do componente entry.
- Linha 10 e 11 - “e1” e “e2” são posicionados na janela.
- Linha 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

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

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.

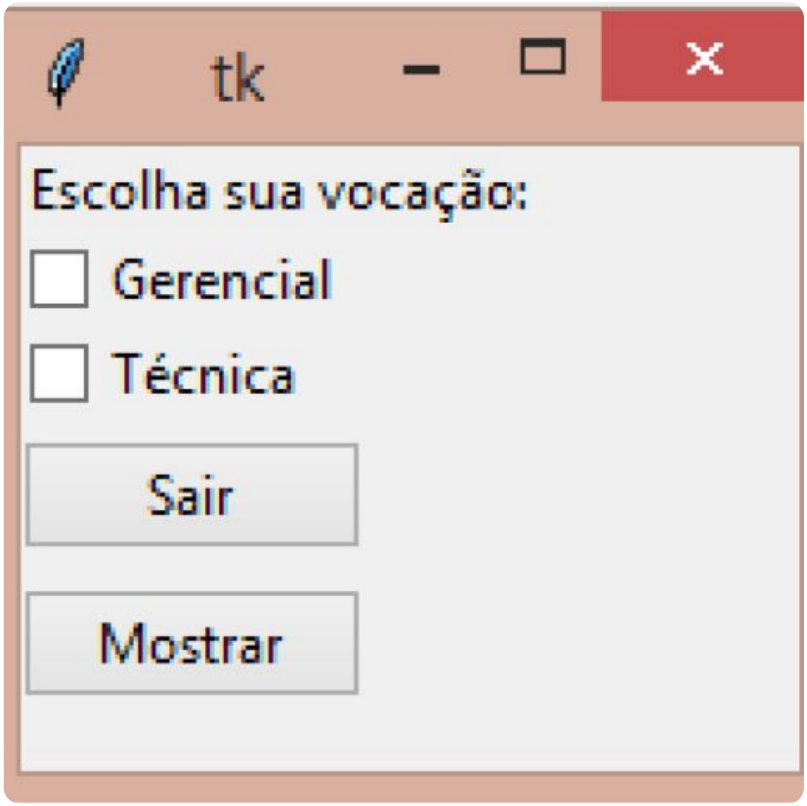
- Linha 5 e 6 - São instanciados os componentes “radiobuttons”. A propriedade “variable” recebe o valor “v”, indicando que o componente será colocado na vertical.

Widget Checkbox

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

```
python

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

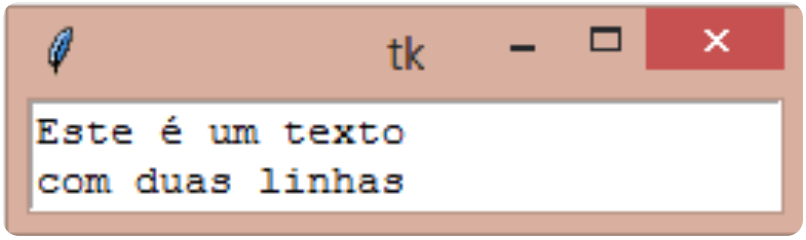
- Linha 4 - É implementada a função “escolha_carreira”, que exibirá os valores dos objetos “var1” e “var2”, que estão relacionados aos “checkboxes”.
- Linhas 7 e 9 - São instanciados os objetos “var1” e “var2”, que serão associados aos “checkboxes”. Quando um “checkbox” for selecionado, o seu respectivo 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.

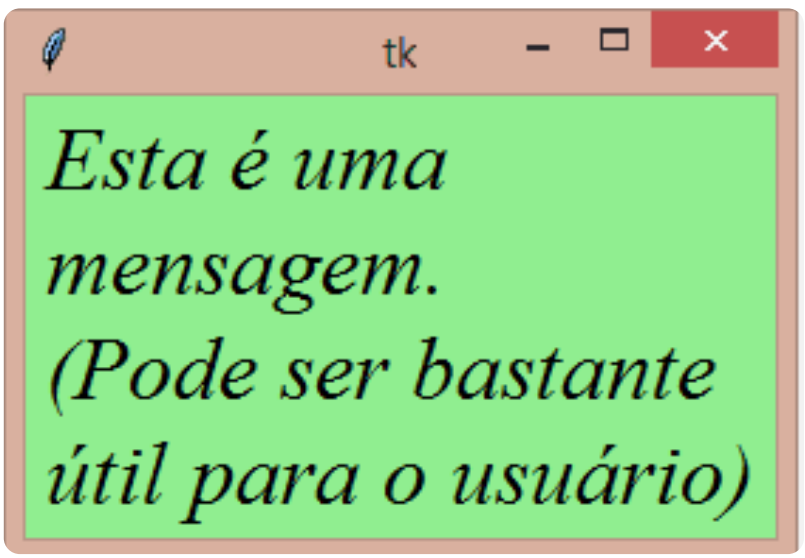
- Linha 3 - É feita uma instância do componente “Text”.
- Linha 5 - É inserido um texto na instância do componente “text”, que será exibido na tela. Observe que o texto é separado em duas linhas com o uso do “\n”.

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()
janela.mainloop()
```

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.

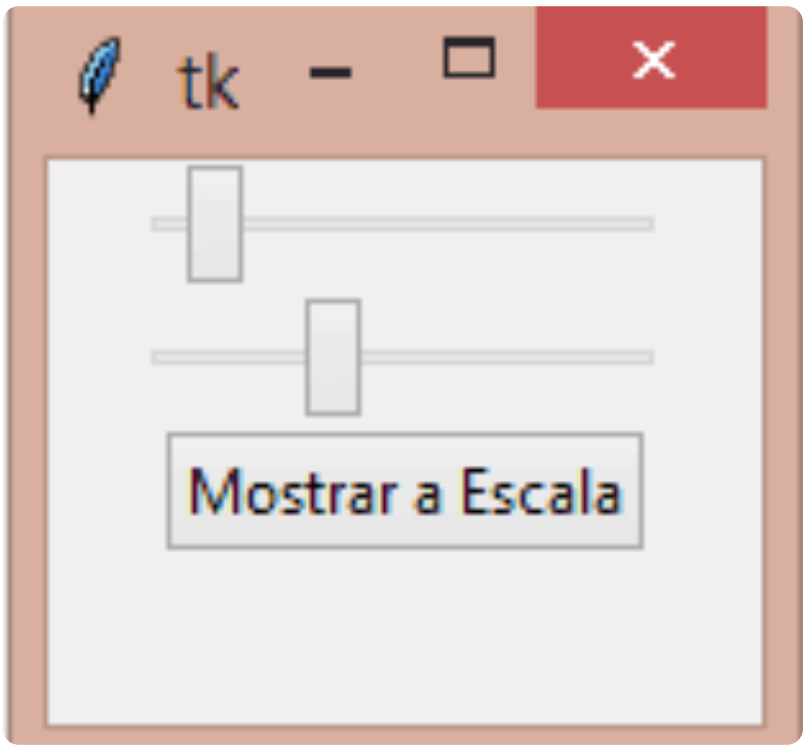
- Linha 5 - O componente é configurado, determinando a cor do “background” e detalhes sobre a fonte da mensagem.

Widget Slider

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

```
python

import tkinter as tk
from tkinter import ttk
def mostrar_valores():
   print (w1.get(), w2.get())
janela = tk.Tk()
w1 = ttk.Scale(janela, from_=0, to=50)
w1.pack()
w2 = ttk.Scale(janela, from_=0, to=100, orient=tk.HORIZONTAL)
w2.pack()
ttk.Button(janela, text='Mostrar a Escala', command=mostrar_valores).pack()
janela.mainloop()
```

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.

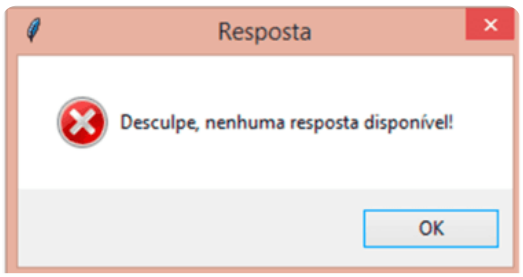
Widget Dialog

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

```
python

import tkinter as tk
from tkinter import messagebox as mb
def resposta():
   mb.showerror("Resposta", "Desculpe, nenhuma resposta disponível!")
def verificacao():
   if mb.askyesno('Verificar', 'Realmente quer sair?'):
      mb.showwarning('Yes', 'Ainda não foi implementado')
   else:
      mb.showinfo('No', 'A opção de Sair foi cancelada')
tk.Button(text='Sair', command=verificacao).pack(fill=tk.X)
tk.Button(text='Resposta', command=resposta).pack(fill=tk.X)
tk.mainloop()
```

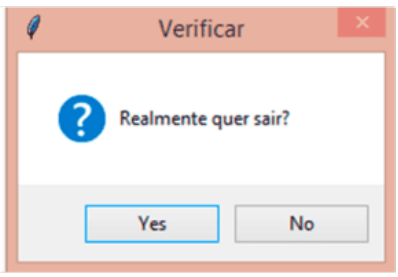
O código produzirá uma janela com dois botões, conforme a imagem a seguir.

Exemplo com dois componentes botões.



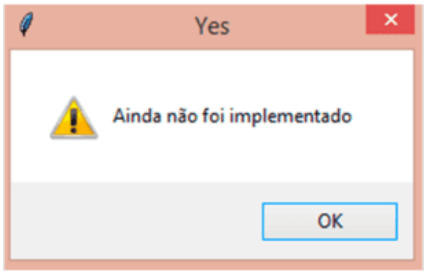
Resposta

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



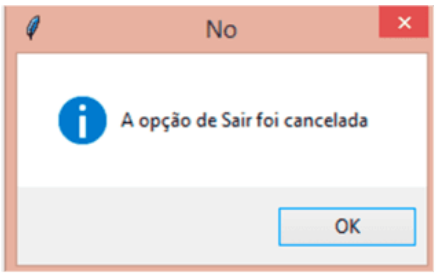
Sair

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

import tkinter as tk
from tkinter import ttk
# Criação de uma janela tkinter
janela = tk.Tk()
janela.title('Combobox')
janela.geometry('500x250')
# Componente Label
ttk.Label(janela, text = "Combobox Widget",background = 'green', foreground ="white",font
= ("Times New Roman", 15)).grid(row = 0, column = 1)
# Componente Label
ttk.Label(janela, text = "Selecione um mês :",font = ("Times New Roman", 10)).grid(column
= 0,row = 5, padx = 10, pady = 25)
# Componente Combobox
n = tk.StringVar()
escolha = ttk.Combobox(janela, width = 27, textvariable = n)
# Adição de itens no Combobox
escolha['values'] = (' Janeiro',' Fevereiro',' Março',' Abril',' Maio',' Junho','
Julho',' Agosto',' Setembro',' Outubro',' Novembro',' Dezembro')
escolha.grid(column = 1, row = 5)
escolha.current()
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.

Atividade 2

Diferentes widgets podem ser implementados para melhorar a funcionalidade de uma aplicação e a experiência de seus usuários. Qual dos seguintes widgets é mais adequado para permitir que os usuários escolham entre várias opções predefinidas?

A

Text

B

Button

C

Slider

D

Radiobutton

E

Entry

A alternativa D está correta.

Radiobuttons permitem que os usuários escolham uma única opção de um conjunto predefinido, sendo ideais para seleções exclusivas entre várias opções, facilitando uma interação clara e direta.

Verificando o aprendizado

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

A alternativa D está correta.

A Tkinter é considerada a biblioteca padrão de interface gráfica do Python. Para que um programa que faz uso de seus componentes possa rodar, é necessário importar a biblioteca, instanciar um componente “window” que é a “raiz” da interface gráfica onde os demais componentes serão adicionados e, por fim, fazer a chamada (invocação) do método “mainloop”.

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”.

A alternativa D está correta.

O componente "label" é muito útil para fornecer uma explicação a respeito de uma funcionalidade do sistema, por exemplo, ele pode acompanhar uma caixa de texto (widget entry) com a legenda "nome", que dá a entender para o usuário que ele precisa preencher seu próprio nome.

Conceitos de integração com banco de dados

A capacidade de executar operações CRUD (create, read, update, delete) diretamente do Python para gerenciar dados em um banco otimiza processos e facilita a gestão de grandes volumes de informações. Com uma boa integração, é possível realizar análises complexas, garantir a integridade dos dados e melhorar a experiência do usuário, tornando as aplicações mais dinâmicas e interativas.

Neste vídeo, vamos explorar como as aplicações interagem com bancos de dados usando Python e PostgreSQL. Vamos aprender a configurar conexões, executar operações CRUD e manipular dados de forma eficiente. Assista!

Conteúdo interativo

Acesse a versão digital para assistir ao vídeo.

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”.

Saiba mais

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

Em seguida, fazemos uma consulta na tabela:

```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:

```
plain-text

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.

Atividade 1

Em uma aplicação Python que interage com um banco de dados PostgreSQL, o desenvolvedor precisa garantir que as alterações feitas em uma transação sejam permanentes e visíveis para outros usuários. Qual comando deve ser utilizado para confirmar as operações realizadas durante a transação?

A

cursor.execute()

B

connection.rollback()

C

connection.close()

D

cursor.fetchall()

E

connection.commit()

A alternativa E está correta.

O comando connection.commit() é usado para confirmar a transação atual no banco de dados. Ele garante que todas as alterações feitas sejam salvas permanentemente, tornando-as visíveis para outros usuários e aplicações.

Integração com banco de dados com psycopg2

Utilizando a biblioteca psycopg2, é possível conectar Python ao PostgreSQL e realizar operações CRUD de forma eficiente e segura. Essa integração permite manipular dados diretamente a partir de scripts Python, automatizar tarefas repetitivas e desenvolver soluções complexas com armazenamento persistente. Dominar o psycopg2 é essencial para quem deseja trabalhar com grandes volumes de dados e otimizar o desempenho das aplicações.

Neste vídeo, você aprenderá a usar o psycopg2 para integrar Python ao PostgreSQL. Vamos mostrar, com exemplos práticos, como realizar operações CRUD de maneira eficiente e segura, aprimorando suas habilidades em programação de bancos de dados. Confira!

> Conteúdo interativo
>
> Acesse a versão digital para assistir ao vídeo.

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 = "5432")
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.

```
sql

import psycopg2
conn = psycopg2.connect(database = " postgres", user="postgres" , password=" senha123" ,
host="127.0.0.1", port="5432" )
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”.

Saiba mais

postgres 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

import psycopg2
conn = psycopg2.connect(database = " postgres", user="postgres" , password=" senha123" ,
host="127.0.0.1", port="5432" )
print ("Conexão com o Banco de Dados aberta com sucesso!")
cur=conn.cursor()
cur.execute("""select * from public."AGENDA" where "id"=1""")
registro=cur.fetchone()
print(registro) 8 conn.commit() 9 print("Seleção realizada com sucesso!");
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
import psycopg2
conn = psycopg2.connect(database = " postgres", user="postgres" , password="senha123" ,
host="127.0.0.1" , port="5432" )
print ("Conexão com o Banco de Dados aberta com sucesso!")
cur=conn.cursor()
print("Consulta antes da atualização")
cur.execute("""select * from public."AGENDA" where "id"=1""")
registro=cur.fetchone()
print(registro)
#Atualização de um único registro
cur.execute("""Update public."AGENDA" set "telefone"='02188888888' where "id"=1""")
conn.commit()
print("Registro Atualizado com sucesso! ")
cur = conn.cursor()
print(" Consulta depois da atualização")
cur.execute("""select * from public."AGENDA" where "id"=1""")
registro=cur.fetchone()
print(registro)
conn.commit()
print("Seleção realizada com sucesso!");
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

import psycopg2
conn = psycopg2.connect(database = " postgres", user="postgres" , password="senha123" ,
host="127.0.0.1" , port="5432" )
print ("Conexão com o Banco de Dados aberta com sucesso!")
cur=conn.cursor()
cur.execute("""Delete from public."AGENDA" where "id"=1""")
conn.commit()
cont=cur.rowcount
print(cont, "Registro excluído com sucesso!")
print("Exclusão realizada com sucesso!");
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.

Atividade 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 as APIs do psycopg2 com 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.

A alternativa C está correta.

O comando “execute” é fundamental para executar as instruções do sql para fazer inserção, consulta, atualização e exclusão de registros no banco de dados.

Verificando o aprendizado

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", port = "5432")
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!");
conn.lacuna_III()
```

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

A alternativa C está correta.

A palavra-chave “INTO” está relacionada ao comando “INSERT” do sql. Após a execução de um comando sql, é necessário confirmá-lo mediante o comando “commit”. No final da execução de um programa, deve-se fechar a conexão com o banco de dados com o comando “close”.

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.

A alternativa C está correta.

O comando "execute" é fundamental para executar as instruções do sql para fazer inserção, consulta, atualização e exclusão de registros no banco de dados.

Visão geral da aplicação com GUI e banco de dados

A integração de uma interface gráfica com um banco de dados é essencial para o desenvolvimento de aplicações interativas e eficientes. Uma visão geral dessa aplicação, que combina GUI e operações de banco de dados, destaca a importância prática de oferecer uma experiência intuitiva ao usuário, permitindo a manipulação de dados diretamente pela interface. As operações CRUD (create, read, update, delete) são facilitadas por componentes gráficos, como caixas de texto e grades, que tornam a interação com o banco de dados mais acessível e visualmente compreensível. Essa abordagem não apenas melhora a usabilidade, mas também aumenta a eficiência na gestão de dados.

Neste vídeo, mostraremos como integrar uma interface gráfica (GUI) com operações de banco de dados. Você aprenderá a criar uma interface intuitiva com Tkinter e a realizar operações CRUD de forma eficiente. Veja como melhorar a experiência do usuário e a gestão de dados em suas aplicações. Não deixe de conferir!

> Conteúdo interativo
>
> Acesse a versão digital para assistir ao vídeo.

Conceitos

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

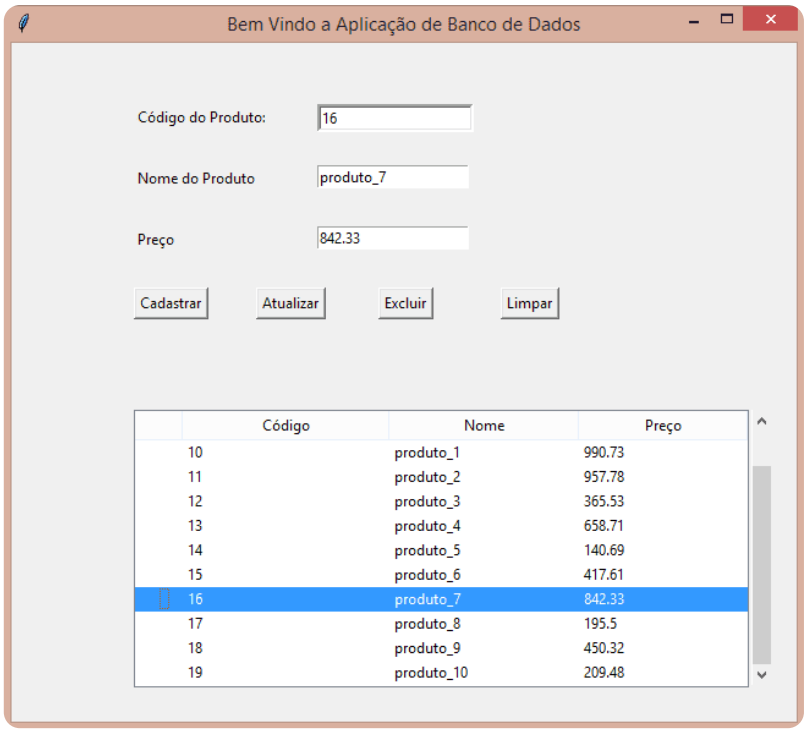
Neste módulo, criaremos uma aplicação que integra tanto elementos de interface gráfica quanto operações com banco de dados.

> 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.

Atividade 1

Você está desenvolvendo uma aplicação que integra uma interface gráfica com um banco de dados, utilizando Tkinter para a GUI e PostgreSQL para o banco de dados. Qual a principal vantagem de integrar uma interface gráfica (GUI) com operações de banco de dados em uma aplicação?

A

Melhoria na velocidade de execução dos comandos SQL.

B

Redução no uso de memória pela aplicação.

C

Facilidade para o usuário final interagir com os dados.

D

Aumento da segurança dos dados armazenados.

E

Eliminação da necessidade de validação de dados.

A alternativa C está correta.

A principal vantagem de integrar uma GUI com operações de banco de dados é a melhoria na experiência do usuário, tornando a interação com os dados mais intuitiva e acessível, sem a necessidade de conhecimentos técnicos avançados.

Criação de tabelas e geração de dados

A criação de tabelas e a inserção de dados aleatórios no PostgreSQL são etapas fundamentais no desenvolvimento de aplicações robustas e eficientes. As tabelas constituem a estrutura básica de armazenamento de dados em um banco de dados relacional, permitindo a organização e a gestão de informações de forma estruturada e acessível.

Definir corretamente as tabelas, com os tipos de dados apropriados e as restrições necessárias, é essencial para garantir a integridade e a consistência dos dados. Além disso, a inserção de dados aleatórios é uma prática valiosa durante a fase de desenvolvimento e teste de uma aplicação.

Comentário

Utilizando bibliotecas como Faker, desenvolvedores podem gerar grandes volumes de dados fictícios que simulam cenários reais de uso. Isso permite testar a aplicação sob condições próximas às do ambiente de produção, identificando e corrigindo problemas de desempenho e lógica antes que afetem os usuários finais. A prática de inserir dados aleatórios também é útil para validar a funcionalidade de consultas SQL, relatórios e outras operações de banco de dados.

Em resumo, criar tabelas bem estruturadas e utilizar dados aleatórios para testes são passos necessários para garantir que a aplicação final seja confiável, eficiente e pronta para lidar com os desafios do mundo real.

Criação de tabelas e geração de dados

Neste vídeo, você aprenderá a se conectar a um banco de dados PostgreSQL usando psycopg2. Vamos explorar a criação de tabelas, a inserção de dados aleatórios com Faker e a exibição de dados, oferecendo uma experiência prática e ampla. Confira!

Conteúdo interativo

Acesse a versão digital para assistir ao vídeo.

Roteiro de prática

Você foi contratado para desenvolver uma aplicação para uma pequena loja que precisa gerenciar seu estoque de produtos. A aplicação deve permitir que o usuário insira, visualize, atualize e exclua informações sobre os produtos. A interface gráfica será criada posteriormente usando Tkinter e o banco de dados será gerenciado com PostgreSQL, utilizando a biblioteca psycopg2 para realizar as operações de CRUD.

Objetivo

Desenvolver uma aplicação que se conecta a um banco de dados PostgreSQL. A aplicação deve permitir a criação de uma tabela de produtos e inserção de dados.

1. Instalação das bibliotecas necessárias

````
plain-text

```bash
   pip install psycopg2
   pip install faker
```
````

2. Criação do banco de dados e tabela

Passo 1: crie um banco de dados chamado lojadb no PostgreSQL usando pgAdmin ou linha de comando.

Passo 2: crie uma tabela Produto com os campos Codigo, Nome e Preco.

Código SQL para criação da tabela:

````
plain-text

```sql
   CREATE TABLE PRODUTO (
       CODIGO SERIAL PRIMARY KEY,
       NOME VARCHAR(100) NOT NULL,
       PRECO NUMERIC(10, 2) NOT NULL
   );
```
````

3. Criação da tabela via Python

plain-text

```python
import psycopg2

conn = psycopg2.connect(
    database="lojadb",
    user="postgres",
    password="senha123",
    host="127.0.0.1",
    port="5432"
)
print("Conexão com o banco de dados aberta com sucesso!")
cur = conn.cursor()
cur.execute('''
    CREATE TABLE PRODUTO (
        CODIGO SERIAL PRIMARY KEY,
        NOME VARCHAR(100) NOT NULL,
        PRECO NUMERIC(10, 2) NOT NULL
    );
''')
conn.commit()
print("Tabela criada com sucesso!")
conn.close()
```

4. Geração de dados aleatórios

plain-text

```python
from faker import Faker
import psycopg2

fake = Faker('pt_BR')

conn = psycopg2.connect(
    database="lojadb",
    user="postgres",
    password="senha123",
    host="127.0.0.1",
    port="5432"
)
cur = conn.cursor()

for _ in range(10):
    nome = fake.word()
    preco = round(fake.random_number(digits=5) / 100, 2)
    cur.execute('''
        INSERT INTO PRODUTO (NOME, PRECO) VALUES (%s, %s)
    ''', (nome, preco))

conn.commit()
print("Dados inseridos com sucesso!")
conn.close()
```

5. Testando a aplicação

Execute os scripts para criar a tabela e inserir dados.

Atividade 2

Você está desenvolvendo uma aplicação que insere dados em uma tabela PostgreSQL usando a biblioteca psycopg2 em Python. O que aconteceria se o desenvolvedor não utilizasse o comando conn.commit() após executar uma operação de inserção de dados?

A

Os dados seriam inseridos corretamente na tabela, mas com um atraso.

B

Os dados seriam inseridos corretamente na tabela sem problemas.

C

Os dados seriam inseridos, mas não estariam disponíveis para outras transações até que a conexão fosse fechada.

D

Os dados não seriam inseridos na tabela e a operação seria revertida quando a conexão fosse fechada.

E

Os dados seriam inseridos apenas se a tabela estivesse vazia.

A alternativa D está correta.

O comando conn.commit() é necessário para confirmar a transação no PostgreSQL. Sem esse comando, qualquer operação de inserção, atualização ou exclusão não será salva permanentemente no banco de dados e será revertida quando a conexão for fechada.

Interação entre o sistema e o banco de dados

A interface gráfica do usuário e o banco de dados são essenciais para o desenvolvimento de sistemas interativos e eficientes. Essa integração permite que os usuários interajam com os dados de maneira intuitiva e visualmente agradável, facilitando operações como inserção, atualização, consulta e exclusão de informações sem a necessidade de conhecimentos técnicos avançados. Utilizar uma GUI torna o sistema mais acessível e user-friendly, enquanto o banco de dados garante o armazenamento seguro, estruturado e eficiente das informações.

Comentário

Essa combinação é fundamental em diversos setores, como comércio, saúde, educação e finanças, em que a manipulação de grandes volumes de dados precisa ser realizada de maneira rápida e precisa. Além disso, a integração facilita a validação e a consistência dos dados, reduzindo erros e melhorando a confiabilidade do sistema.

Ferramentas como Tkinter para a criação de interfaces gráficas e psycopg2 para a comunicação com bancos de dados PostgreSQL são exemplos de tecnologias que permitem essa integração de maneira robusta e flexível. Em resumo, a integração GUI-banco de dados não só melhora a experiência do usuário, mas também aumenta a eficiência operacional e a segurança das informações, tornando-se um componente indispensável no desenvolvimento de aplicações modernas.

Interação entre o sistema e o banco de dados

Neste vídeo, você aprenderá a desenvolver uma aplicação completa com Python, PostgreSQL e Tkinter. Vamos criar uma interface gráfica interativa, integrar operações CRUD com um banco de dados e gerar dados aleatórios para testes. Aprimore suas habilidades com este tutorial prático e completo. Assista!

Conteúdo interativo

Acesse a versão digital para assistir ao vídeo.

Roteiro de prática

Você foi contratado por uma pequena loja de eletrônicos que precisa de um sistema para gerenciar seu inventário de produtos. O sistema deve permitir a criação, leitura, atualização e exclusão (CRUD) de informações sobre os produtos. Para isso, você utilizará PostgreSQL como banco de dados e Python com a biblioteca psycopg2 para executar as operações de banco de dados. Além disso, você utilizará a biblioteca Faker para gerar dados aleatórios durante a fase de desenvolvimento e testes.

Objetivo

Desenvolver uma aplicação Python que se conecte a um banco de dados PostgreSQL e implemente operações CRUD.

1. Instalação das bibliotecas necessárias

````
plain-text

```bash
   pip install psycopg2
   pip install faker
   pip install tk
   ```
````

2. Criação da tabela no PostgreSQL

Passo 1: crie um banco de dados chamado lojadb no PostgreSQL usando pgAdmin ou linha de comando.

Passo 2: crie uma tabela Produto com os campos Codigo, Nome e Preco.

Código SQL para criação da tabela:

````
plain-text

CREATE TABLE PRODUTO (
       CODIGO SERIAL PRIMARY KEY,
       NOME VARCHAR(100) NOT NULL,
       PRECO NUMERIC(10, 2) NOT NULL
   );
   ```
````

3. Implementação da classe Python para operações CRUD
````

plain-text

```python
import psycopg2
from psycopg2 import Error
from faker import Faker

class AppBD:
    def __init__(self):
        self.conn = None
        self.cur = None
        self.connect_to_db()

    def connect_to_db(self):
        try:
            self.conn = psycopg2.connect(
                database="lojadb",
                user="postgres",
                password="senha123",
                host="127.0.0.1",
                port="5432"
            )
            self.cur = self.conn.cursor()
            print("Conexão com o Banco de Dados aberta com sucesso!")
        except (Exception, Error) as error:
            print("Falha ao se conectar ao Banco de Dados", error)

    def selecionar_dados(self):
        try:
            self.cur.execute("SELECT * FROM PRODUTO")
            registros = self.cur.fetchall()
            return registros
        except (Exception, Error) as error:
            print("Erro ao selecionar dados", error)
            return []

    def inserir_dados(self, nome, preco):
        try:
            self.cur.execute(
                '''INSERT INTO PRODUTO (NOME, PRECO) VALUES (%s, %s)''',
                (nome, preco)
            )
            self.conn.commit()
            print("Inserção realizada com sucesso!")
        except (Exception, Error) as error:
            print("Erro ao inserir dados", error)

    def atualizar_dados(self, codigo, nome, preco):
        try:
            self.cur.execute(
                '''UPDATE PRODUTO SET NOME = %s, PRECO = %s WHERE CODIGO = %s''',
                (nome, preco, codigo)
            )
            self.conn.commit()
            print("Atualização realizada com sucesso!")
        except (Exception, Error) as error:
            print("Erro ao atualizar dados", error)

    def excluir_dados(self, codigo):
        try:
            self.cur.execute(
                '''DELETE FROM PRODUTO WHERE CODIGO = %s''',
                (codigo,)
            )
            self.conn.commit()
            print("Exclusão realizada com sucesso!")
        except (Exception, Error) as error:
            print("Erro ao excluir dados", error)
```

4. Geração de dados aleatórios

````
plain-text
 ```python
   # Criando uma instância da classe e inserindo dados aleatórios
   app_bd = AppBD()
   fake = Faker('pt_BR')

   for _ in range(10):
       nome = fake.word()
       preco = round(fake.random_number(digits=5) / 100, 2)
       app_bd.inserir_dados(nome, preco)
   ```
````

Atividade 3

Você está desenvolvendo uma aplicação em Python com interface gráfica usando Tkinter para gerenciar um banco de dados de produtos. O que aconteceria se o desenvolvedor não utilizar o comando root.mainloop() no final do script?

A

A aplicação não será executada e apresentará um erro de sintaxe.

B

A interface gráfica será exibida, mas não responderá a eventos do usuário.

C

A interface gráfica será exibida e funcionará corretamente.

D

A aplicação funcionará, mas sem conexão ao banco de dados.

E

A aplicação não conseguirá executar operações CRUD no banco de dados.

A alternativa B está correta.

O comando root.mainloop() é necessário para iniciar o loop principal do Tkinter, que mantém a janela aberta e permite que a aplicação responda a eventos do usuário. Sem esse comando, a GUI não processará os eventos, tornando-se inoperante.

GUI: Interação com o usuário

A interface gráfica é fundamental para a interação do usuário com aplicações, tornando o uso de sistemas complexos mais intuitivo e acessível. Ao oferecer uma representação visual dos dados e funcionalidades, a GUI (Interface Gráfica do Usuário) facilita a navegação, execução de tarefas e a tomada de decisões. Componentes visuais como botões, caixas de texto, e menus permitem que usuários, mesmo sem conhecimento técnico avançado, interajam de forma eficiente com o sistema.


Funcionário gerenciando livros de biblioteca.

No contexto de gerenciamento de bibliotecas, por exemplo, uma GUI possibilita a visualização, inserção, atualização e exclusão de registros de livros por meio de ações simples, como clicar em um botão ou preencher um formulário.

Isso não apenas melhora a experiência do usuário, mas também reduz a probabilidade de erros, uma vez que operações complexas podem ser abstraídas por trás de uma interface amigável.

Ferramentas como Tkinter em Python permitem a criação de interfaces gráficas robustas e personalizáveis, promovendo a acessibilidade e usabilidade dos sistemas. Em resumo, a interface gráfica é muito importante para tornar aplicações mais práticas e eficientes, facilitando a interação do usuário e melhorando a eficiência operacional.

GUI: interação com o usuário

Neste vídeo, você aprenderá a desenvolver uma aplicação completa com Python, PostgreSQL e Tkinter. Vamos mostrar como criar uma interface gráfica intuitiva para gerenciar operações CRUD em um banco de dados. Siga o passo a passo para integrar esses componentes e construir um sistema eficiente e interativo.

> Conteúdo interativo
>
> Acesse a versão digital para assistir ao vídeo.

Roteiro de prática

1. Configuração do ambiente

- Instalar o PostgreSQL.
- Configurar um banco de dados e uma tabela para os registros dos livros.
- Instalar as bibliotecas necessárias: psycopg2, Faker e tkinter.

2. Criação da tabela no banco de dados

````
plain-text

```sql
   CREATE TABLE livros (
       id SERIAL PRIMARY KEY,
       titulo VARCHAR(255),
       autor VARCHAR(255),
       ano_publicacao INTEGER,
       genero VARCHAR(100)
   );
   ```
````

3. Conexão ao banco de dados com Python

````
plain-text
```python
   import psycopg2
   from faker import Faker

   class BancoDados:
       def __init__(self):
           self.conexao = psycopg2.connect(
               dbname="sua_base_de_dados",
               user="seu_usuario",
               password="sua_senha",
               host="localhost"
           )
           self.cursor = self.conexao.cursor()

       def criar_tabela(self):
           self.cursor.execute("""
               CREATE TABLE IF NOT EXISTS livros (
                   id SERIAL PRIMARY KEY,
                   titulo VARCHAR(255),
                   autor VARCHAR(255),
                   ano_publicacao INTEGER,
                   genero VARCHAR(100)
               )
           """)
           self.conexao.commit()

       def inserir_dados(self, titulo, autor, ano_publicacao, genero):
           self.cursor.execute("""
               INSERT INTO livros (titulo, autor, ano_publicacao, genero)
               VALUES (%s, %s, %s, %s)
           """, (titulo, autor, ano_publicacao, genero))
           self.conexao.commit()
   ```
````

4. Geração de dados aleatórios

````
plain-text
```python
   fake = Faker()
   bd = BancoDados()
   bd.criar_tabela()
   for _ in range(100):
       bd.inserir_dados(fake.text(max_nb_chars=20), fake.name(), fake.year(),
fake.word())
   ```
````

5. Implementação da interface gráfica

plain-text

```python
  import tkinter as tk
  from tkinter import ttk

  class BibliotecaGUI:
      def __init__(self, root, bd):
          self.bd = bd
          self.root = root
          self.root.title("Gerenciador de Biblioteca")

          self.tree = ttk.Treeview(root, columns=("ID", "Título", "Autor", "Ano",
"Gênero"), show="headings")
          self.tree.heading("ID", text="ID")
          self.tree.heading("Título", text="Título")
          self.tree.heading("Autor", text="Autor")
          self.tree.heading("Ano", text="Ano de Publicação")
          self.tree.heading("Gênero", text="Gênero")
          self.tree.pack()

          self.carregar_dados()

      def carregar_dados(self):
          registros = self.bd.selecionar_dados()
          for registro in registros:
              self.tree.insert("", "end", values=registro)

  root = tk.Tk()
  app_bd = BancoDados()
  app_gui = BibliotecaGUI(root, app_bd)
  root.mainloop()
  ```
```

6. Teste das operações CRUD através da interface gráfica

- Execute o script e verifique as funcionalidades da interface gráfica.
- Utilize os botões para inserir, atualizar e excluir dados, e veja os resultados no Treeview.

# Atividade 4

Você está desenvolvendo uma aplicação em Python com interface gráfica usando Tkinter para gerenciar um banco de dados de livros. Durante a fase de desenvolvimento, você decide fazer uma alteração importante no código. O que aconteceria se o desenvolvedor alterar o método carregar_dados para não limpar mais os itens da Treeview antes de exibir os dados carregados?

A

A Treeview sempre exibirá os dados corretos dos registros carregados.

B

A Treeview pode mostrar dados antigos juntamente com os novos dados.

C
```

A aplicação apresentará um erro e não exibirá nenhum dado.

D

Os registros carregados não serão exibidos na interface gráfica.

E

A funcionalidade de carregamento de registros deixará de funcionar completamente.

A alternativa B está correta.

Sem limpar os itens da Treeview antes de exibir novos dados, os novos valores podem ser concatenados aos antigos, causando confusão e inconsistência na interface gráfica.

Verificando o aprendizado

Questão 1

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

```
python

import psycopg2
lacuna_I
conn = psycopg2.connect(database = "postgres", user = "postgres", password = "senha123",
host = "127.0.0.1", port = "5432")
print("Conexão com o Banco de Dados aberta com sucesso!")
cur = conn.cursor()
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.

A alternativa C está correta.

Apesar de não serem obrigatórias, as cláusulas "try-except" previnem a ocorrência de exceções, evitando que o programa seja interrompido abruptamente caso ocorra algum problema.

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

A alternativa B está correta.

A cláusula "try" tenta executar o fluxo normal do programa. Caso ocorra algum problema, a cláusula "except" permite tratar a exceção e exibir uma mensagem amigável para o usuário sem a interrupção do programa. O Python não obriga o uso das cláusulas "try-except", mas trata-se de uma boa prática de programação.

Considerações finais

O que você aprendeu neste conteúdo?

- Estabelecer uma conexão com um banco de dados PostgreSQL utilizando a linguagem Python.
- Realizar operações CRUD sobre os dados: criação (create), leitura (read), atualização (update) e exclusão (delete).
- Desenvolver aplicações com interface gráfica (GUI) amigáveis usando bibliotecas da linguagem Python.
- Integrar interfaces gráficas com banco de dados para manipulação de dados através da interface.

Podcast

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

Conteúdo interativo

Acesse a versão digital para ouvir o áudio.

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.

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.