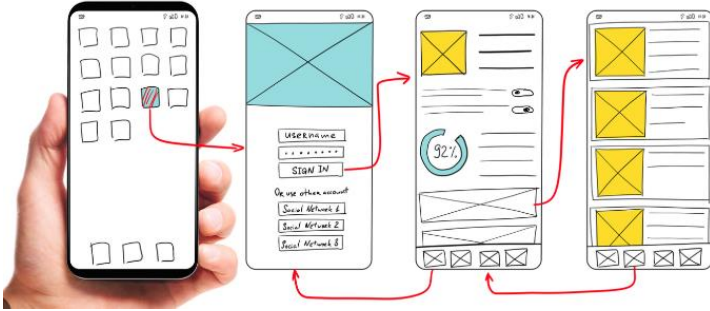


PROGRAMAÇÃO EM PYTHON

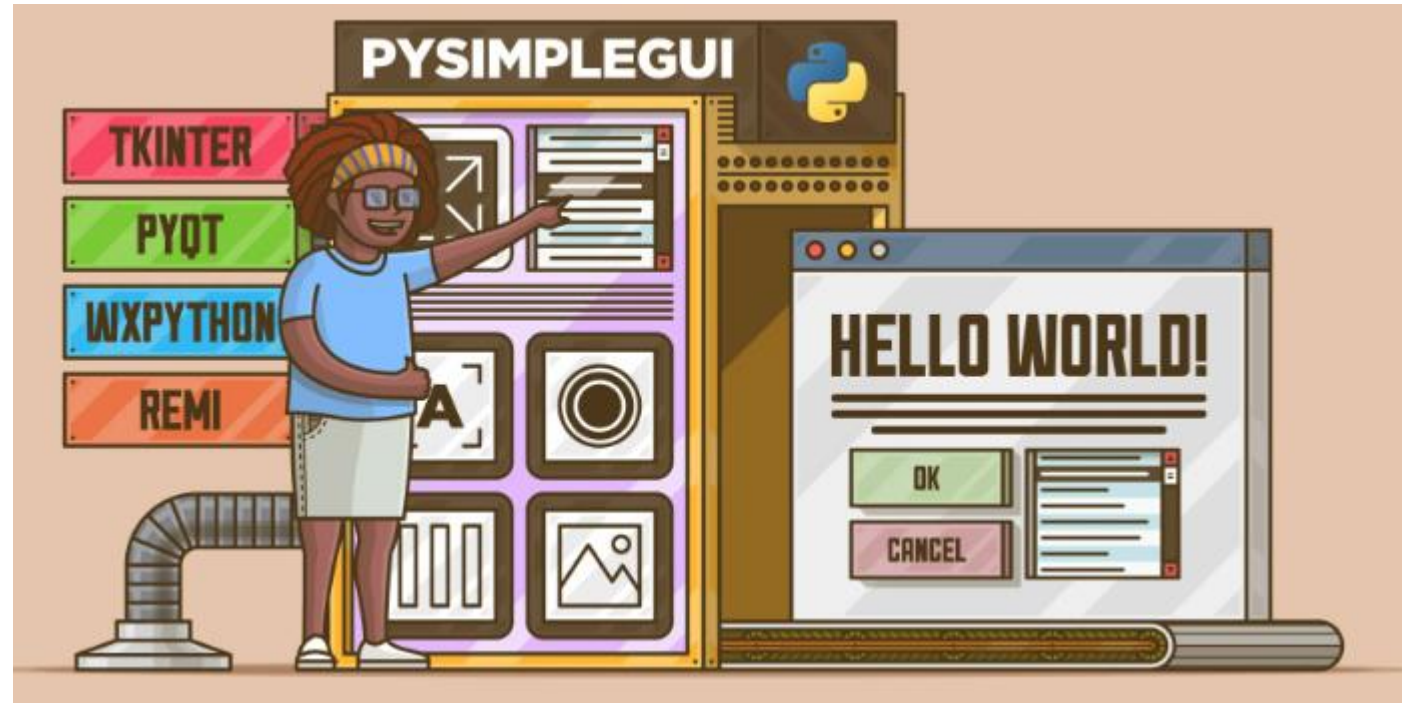
(Telas -Widgets do PyQt6)



Interface Gráfica do Usuário



A GUI, também conhecida como UI (User Interface), é um conjunto de elementos visuais e interativos que permitem aos usuários interagir com um software ou sistema operacional de forma mais amigável e intuitiva.



GUI para Python

Algumas das principais interfaces gráficas para Python, junto com os links para suas respectivas documentações:

Tkinter

1. Tkinter é a biblioteca padrão de interfaces gráficas para o Python.
2. [Documentação do Tkinter](#)

PyQt

1. PyQt é um conjunto de bindings Python para as bibliotecas Qt, que são usadas para criar aplicações GUI.
2. [Documentação do PyQt](#)

Kivy

1. Kivy é uma biblioteca de código aberto para o desenvolvimento de aplicações multi-touch e outras aplicações de interface de usuário.
2. [Documentação do Kivy](#)

wxPython

1. wxPython é um wrapper para a biblioteca GUI wxWidgets, que permite criar programas GUI nativos para várias plataformas.
2. [Documentação do wxPython](#)

PySide

1. PySide, também conhecido como Qt for Python, é um projeto oficial da Qt que fornece bindings para o framework Qt.
2. [Documentação do PySide](#)

PyGTK

1. PyGTK é uma biblioteca que permite que você escreva programas GUI em Python usando a biblioteca GTK.
2. [Documentação do PyGTK](#)

PySimpleGUI

1. PySimpleGUI visa simplificar o processo de criar GUIs com menos linhas de código do que outras bibliotecas.
2. [Documentação do PySimpleGUI](#)

Essas bibliotecas oferecem diferentes funcionalidades e níveis de complexidade, permitindo que os desenvolvedores escolham a que melhor se adapta às suas necessidades específicas de projeto.

GUI para Python



PyQt6

PyQt6 é um conjunto de ligações Python para o kit de ferramentas Qt6 da **Qt Company**, que permite a criação de aplicações GUI (Graphical User Interface) sofisticadas e multiplataforma. Essencialmente, o PyQt6 facilita o desenvolvimento de aplicações gráficas em Python, utilizando as funcionalidades e componentes poderosos fornecidos pelo Qt6.



Características Principais

1. Multiplataforma:

PyQt6 pode ser executado em diferentes sistemas operacionais, incluindo Windows, macOS e Linux, permitindo a criação de aplicações multiplataforma.

2. Componentes Ricos:

Inclui uma ampla variedade de widgets e componentes para construção de interfaces de usuário, como botões, rótulos, caixas de texto, menus, barras de ferramentas, etc.

3. Desenvolvimento Orientado a Objetos:

Aproveita a natureza orientada a objetos de Python e Qt para organizar o código de maneira modular e reutilizável.

4. Integração com Python:

Permite usar bibliotecas Python juntamente com PyQt6, proporcionando um ambiente robusto e flexível para o desenvolvimento de aplicações complexas.

5. Compatibilidade com Qt Designer:

Permite o uso do Qt Designer, uma ferramenta de design visual, para criar interfaces gráficas e gerar código Python correspondente.

6. Documentação Extensa:

Fornecer uma documentação detalhada e exemplos que ajudam os desenvolvedores a começar rapidamente e resolver problemas específicos.

PyQt6

Principais Módulos do PyQt6

- **QtWidgets:** Contém classes para a criação de widgets e layouts.
- **QtCore:** Fornece funcionalidades centrais como gerenciamento de sinal e slot, temporizadores e outras classes de utilidade.
- **QtGui:** Contém classes para manipulação de gráficos, fontes, imagens, etc.
- **QtNetwork:** Proporciona classes para programação de rede.
- **QtMultimedia:** Inclui classes para áudio e vídeo.





Usando PyQt6

Para executar o código Python que importa o módulo `sys` e as classes de widgets do PyQt6, você precisa garantir que o PyQt6 esteja instalado no seu ambiente Python. Aqui estão os passos detalhados para realizar essa tarefa:

1. Instalar o PyQt6: Primeiro, certifique-se de que você tem o PyQt6 instalado. Você pode instalar o PyQt6 usando o comando `pip` no terminal:

```
pip install PyQt6
```

2. No seu script, importe a bibliotecas `sys` e `PyQt6.QtWidgets` especificando os elementos que precisará:

```
1 import sys
2 from PyQt6.QtWidgets import *
```

```
import sys
from PyQt6.QtWidgets import QApplication, QWidget, QLabel, QPushButton, QVBoxLayout
```


Principais Widgets do PyQt6

- **QWidget:** É a classe base de todos os widgets. Todos os componentes visuais de uma aplicação PyQt6 herdam de QWidget.

- **QLabel:** Exibe texto ou imagem. É usado para mostrar informações ao usuário.

```
label = QLabel('Texto de exemplo')
```

- **QPushButton:** Um botão que o usuário pode clicar para executar uma ação.

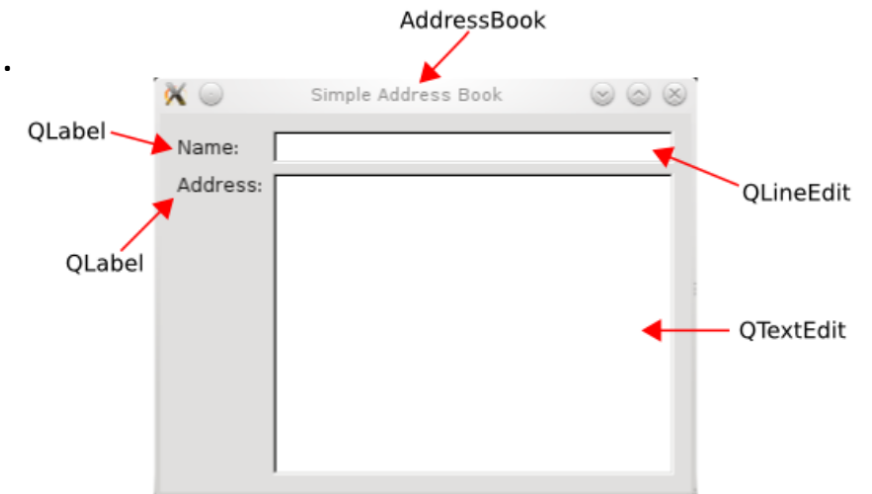
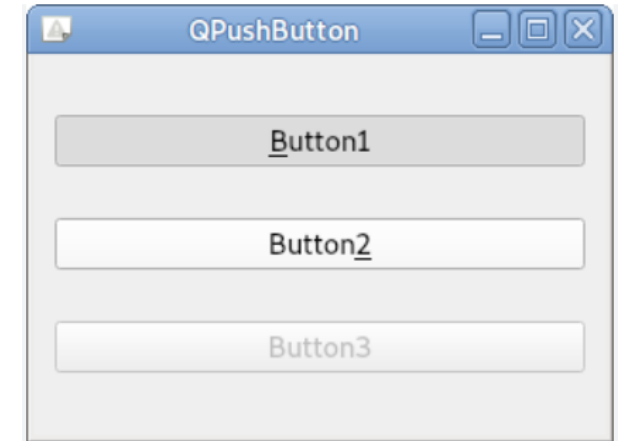
```
button = QPushButton('Clique-me')
```

- **QLineEdit:** Um campo de entrada de texto de uma linha.

```
line_edit = QLineEdit()
```

- **QTextEdit:** Um campo de entrada de texto multilinhas.

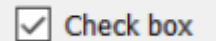
```
text_edit = QTextEdit()
```



Principais Widgets do PyQt6

- **QCheckBox:** Uma caixa de seleção que pode ser marcada ou desmarcada.
- **QRadioButton:** Um botão de rádio que pode ser selecionado ou desmarcado, geralmente usado em grupos para selecionar uma única opção entre várias.
- **QComboBox:** Uma caixa de combinação que permite ao usuário selecionar uma opção a partir de um menu suspenso.

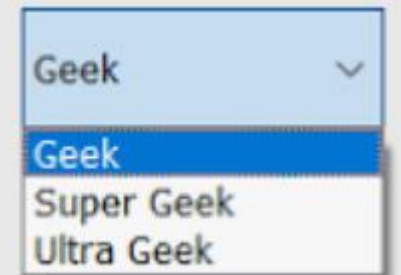
```
checkbox = QCheckBox('Opção')
```



```
radio_button = QRadioButton('Opção 1')
```



```
combo_box = QComboBox()  
combo_box.addItem('Opção 1')  
combo_box.addItem('Opção 2')
```



Principais Widgets do PyQt6

- **QListWidget:** Um widget que exibe uma lista de itens.

```
list_widget = QListWidget()
list_widget.addItem('Item 1')
list_widget.addItem('Item 2')
```

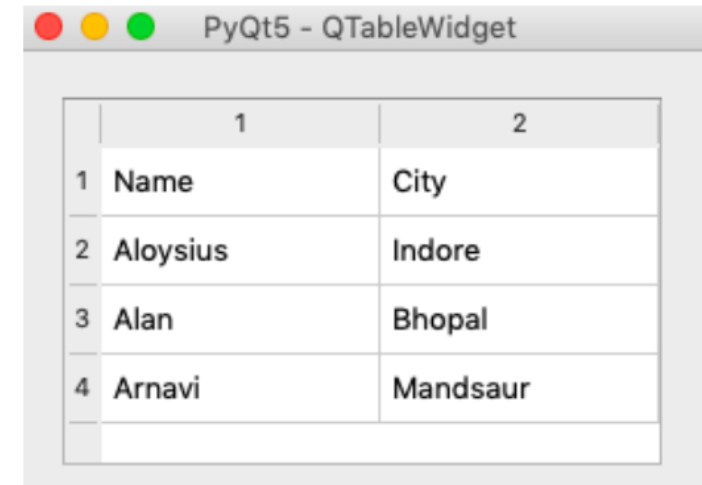


- **QTableWidget:** Um widget para exibir dados em uma tabela.

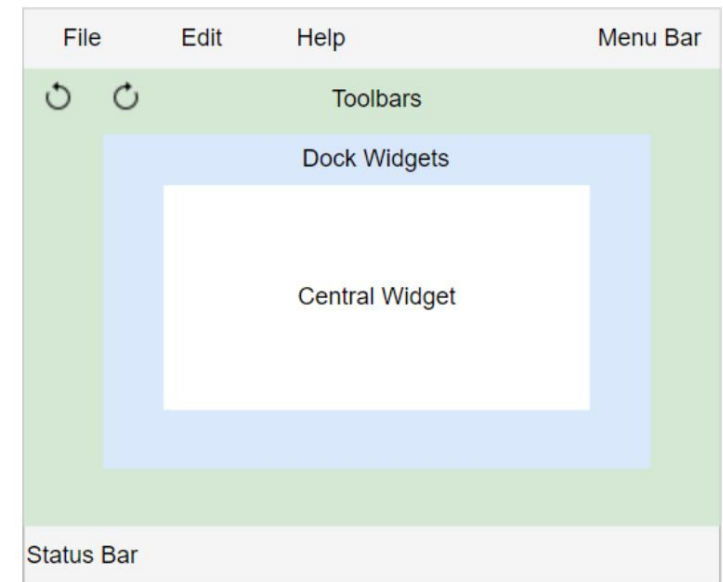
```
table_widget = QTableWidget(3, 2) # 3 linhas e 2 colunas
```

- **QMainWindow:** Uma janela principal padrão, que pode conter menus, barras de ferramentas, barras de status, etc.

<https://www.pythontutorial.net/pyqt/pyqt-qmainwindow/>



	1	2
1	Name	City
2	Aloysius	Indore
3	Alan	Bhopal
4	Arnavi	Mandsaur



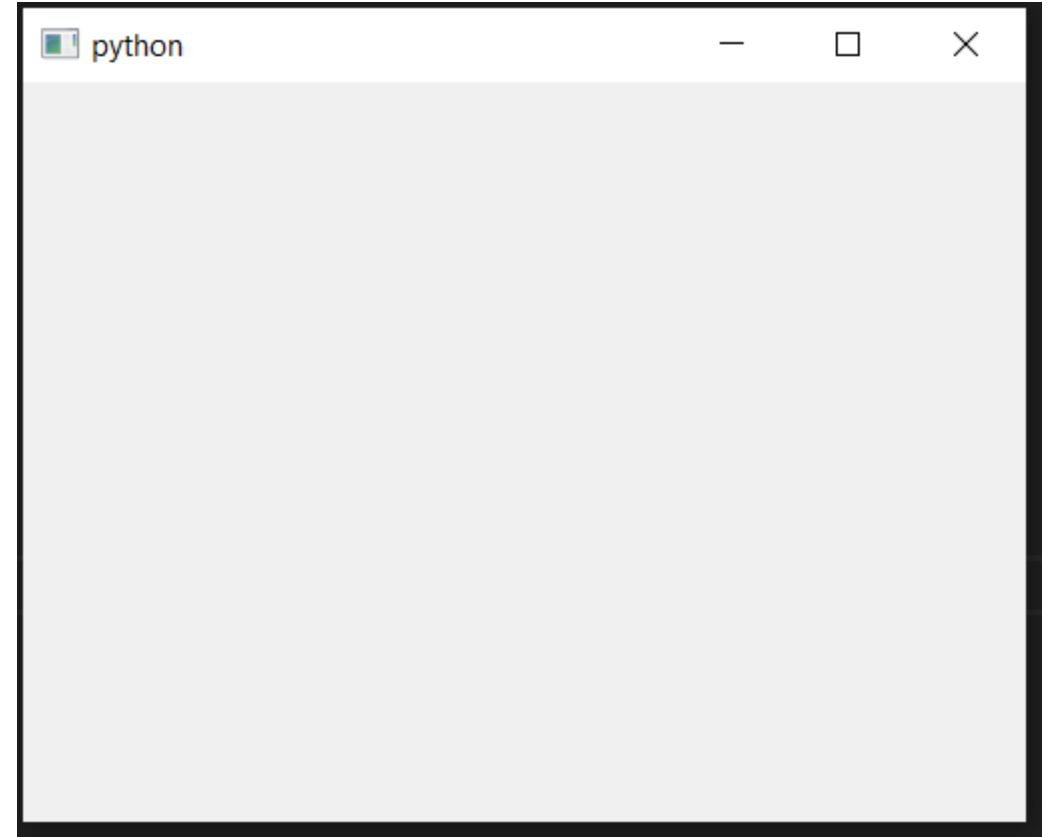
Criar janela

```
import sys
from PyQt6.QtWidgets import *

app = QApplication(sys.argv)

janela = QWidget()
janela.resize(400,400)

janela.show()
app.exec()
```



Inserir Widgets na janela

```
import sys
from PyQt6.QtWidgets import *

app = QApplication(sys.argv)

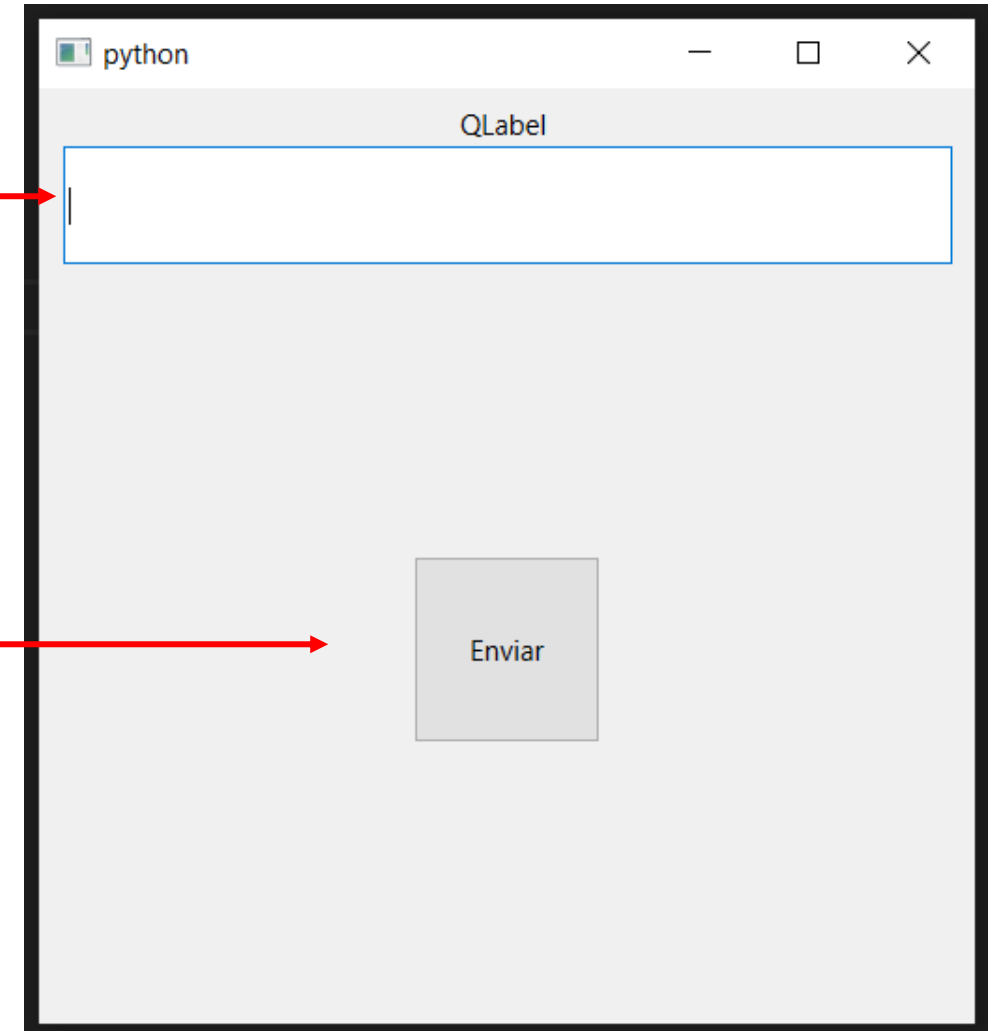
janela = QWidget()
janela.resize(400,400) # (x, y)

#Inserindo elementos na tela
label= QLabel("QLabel",janela) #(O que vai escrito, onde é inserido)
label.setGeometry(10,5,380,20) # (x, y, largura,altura)

linha_de_texto = QLineEdit('',janela)
linha_de_texto.setGeometry(10,35,380,50)

botao = QPushButton('Enviar',janela)
botao.setGeometry(160,200,80,80)

janela.show()
app.exec()
```



Qt Style Sheets

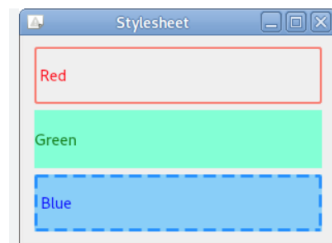
Os **Qt Style Sheets** permitem que você aplique estilos aos widgets de maneira muito semelhante ao CSS usado para estilizar páginas web.

Pode aplicar direto no código:

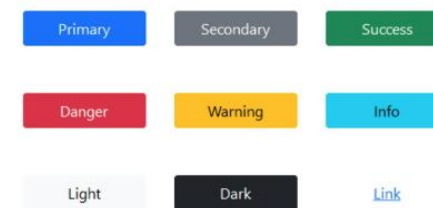
```
self.button = QPushButton('Clique-me')
self.button.setStyleSheet("""
    QPushButton {
        background-color: lightgreen;
        border: 2px solid darkgreen;
        font-size: 16px;
    }
    QPushButton:hover {
        background-color: green;
        color: white;
    }
    """)
```

Pode separar em arquivos:

```
# style.css
# style.css > QWidget
1 QWidget{
2     background-color: white;
3 }
4 QLineEdit{
5     background-color: tomato;
6     color: white;
7     font-size: 18px;
8     border: none;
9 }
10 }
```



QPushButton with StyleSheet



```
# style.css
calculadora.py
calculadora.py > potencia
84 with open('style.css', 'r') as file:
85     app.setStyleSheet(file.read())
86
```

Qt Style Sheets Reference
<https://doc.qt.io/qt-6/stylesheet-reference.html>

Estilizando Widgets

`elemento.setStyleSheet()`

```
label= QLabel("QLabel",janela) #(O que vai escrito, onde é inserido)
label.setGeometry(10,5,380,20) # (x, y, largura,altura)
label.setStyleSheet('background-color: blue;color:white; font-size:
20px; font: 14pt "MS Shell Dlg 2"; qproperty-alignment: AlignCenter;')

linha_de_texto = QLineEdit('',janela)
linha_de_texto.setGeometry(10,35,380,50)

botao = QPushButton('Enviar',janela)
botao.setGeometry(160,200,80,80)
botao.clicked.connect(funcao_apertou)
botao.setStyleSheet('background-color: black;color:white;
border-style: solid;border-color:red;border-width: 1px;border-radius:
5px;font-size: 20px;')
```



Estilizando Widgets em arquivos separados

```
with open('estilo_teste.css','r') as file:  
    app.setStyleSheet(file.read())
```



The screenshot shows a code editor with two tabs: 'teste.py' and '# estilo_teste.css'. The 'teste.py' tab is active, showing a Python script. The script has line numbers 14 through 19. Lines 16 and 17 contain the code to open 'estilo_teste.css' and apply its styles to the application.

```
14  
15  
16 with open('estilo_teste.css','r') as file:  
17     app.setStyleSheet(file.read())  
18  
19
```




The screenshot shows a code editor with two tabs: 'teste.py' and '# estilo_teste.css'. The '# estilo_teste.css' tab is active, showing CSS styles. The styles are for 'QLabel' and 'QPushButton'. The 'QLabel' style is on lines 1-6, and the 'QPushButton' style is on lines 8-16. The styles include background-color, color, font-size, font, text-align, border-style, border-color, border-width, border-radius, and font-size.

```
# estilo_teste.css > QPushButton  
1 QLabel{  
2     background-color: blue;  
3     color: white; font-size: 20px;  
4     font: 14pt "MS Shell Dlg 2";  
5     text-align: center;  
6 }  
7  
8 QPushButton{  
9     background-color: black;  
10    color: white;  
11    border-style: solid;  
12    border-color: red;  
13    border-width: 1px;  
14    border-radius: 5px;  
15    font-size: 20px;  
16 }
```


Chamar função no QPushButton

```
4
5 def funcao_apertou():
6     print("Você apertou o botão!")
7
8
9 app = QApplication(sys.argv)
10
11
12 janela = QWidget()
13 janela.resize(400,400) # (x, y)
14
15
16 with open('estilo_teste.css',
17         ... app.setStyleSheet(file.read())
18
19
```



The screenshot shows a Python IDE with a Qt window titled 'python'. The window contains a 'QLabel' widget and a button labeled 'Enviar'. The button is a dark rectangle with the text 'Enviar' in white. The IDE interface includes a terminal at the bottom showing the output of the program.

```
PS C:\Users\Professor\Desktop\telas e Calculadora> python ra/teste.py
Traceback (most recent call last):
  File "c:\Users\Professor\Desktop\telas e Calculadora\teste.py", line 17, in <module>
    app.setStyleSheet(file.read())
  File "c:\Users\Professor\Desktop\telas e Calculadora\teste.py", line 17, in <module>
    app.setStyleSheet(file.read())
ModuleNotFoundError: No module named 'PyQt6.QtWidgets'
PS C:\Users\Professor\Desktop\telas e Calculadora> python ra/teste.py
Você apertou o botão!
```

```
24
25 linha_de_texto = QLineEdit('',janela)
26 linha_de_texto.setGeometry(10,35,380,50)
27
28 botao = QPushButton('Enviar',janela)
29 botao.setGeometry(160,200,80,80)
30 botao.clicked.connect(funcao_apertou)
31
```

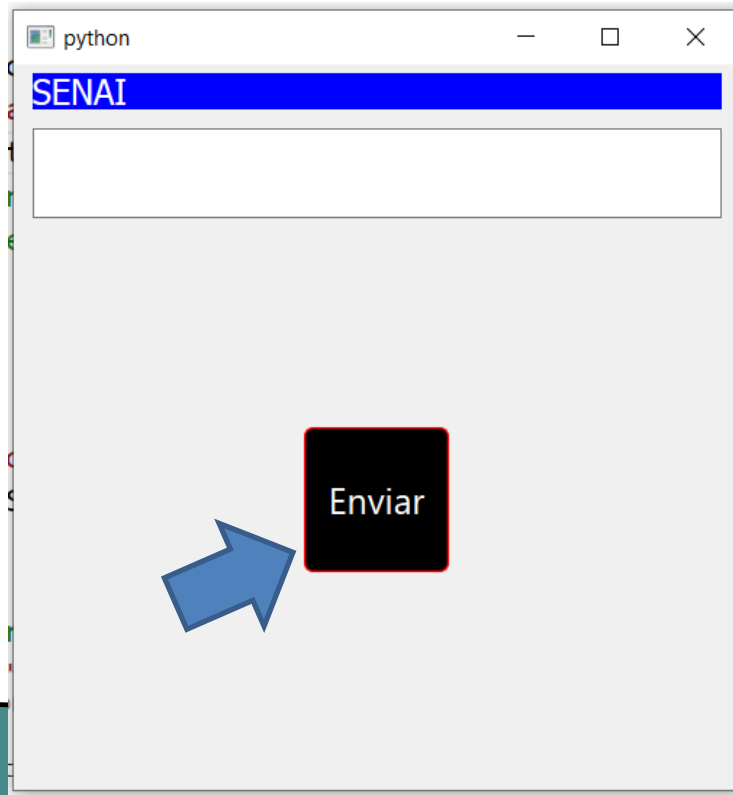
Ao apertar o botão a função “funcao_apertou” é executada imprimindo no terminal a frase do “print”

Chamar função no QPushButton

#Funções

```
def funcao_apertou():  
    print("Você apertou o botão!")  
    label.setText('SENAI')
```

Ao apertar o botão, substitui o texto do QLabel por "SENAI".



#Funções

```
def funcao_apertou():  
    print("Você apertou o botão!")  
    texto = linha_de_texto.text()  
    label.setText(texto)
```

Pegando o que foi digitado no QLineEdit e exibindo ele no QLabel ao clicar no botão

