

Linguagem de Programação Python

Prof. Victor Sotero

Aula 01 – Introdução ao PyQT6

Contextualização

- Python é uma linguagem de programação muito robusta. Isso implica em dizer que ela consegue se adequar as mais diversas particularidades;
- Sendo assim, embora não seja a melhor solução para a criação de Front-End, nós podemos fazer uso de bibliotecas que permitem tal funcionalidade;
- Por isso, através da linguagem Python é possível criar aplicações web, mobile, IoT, etc.

Contextualizando

- Embora existam várias bibliotecas específicas para a criação de interfaces gráficas em Python, vamos utilizar a biblioteca PyQt6;
- Essa biblioteca possui um grande acervo de ferramentas, assim como uma facilidade para ser implementada.

Interface Gráfica em Python

- A interface gráfica corresponde a uma camada de abstração contendo funcionalidades para o usuário poder “surfar” no programa;
- Sua ideia, além de oferecer um visual intuitivo, é proteger a estrutura interna do programa (funcionam atrás das cortinas);
- Antes de prover um visual bonito é preciso que a interface gráfica seja bastante intuitiva e funcional.

Biblioteca GUI

- Conforme dito anteriormente, a linguagem Python possui diversas ferramentas (bibliotecas) que auxiliam a criação de interfaces gráficas;
- Nosso foco será a biblioteca PyQt;
- Ao término da Unidade 2 do curso vamos buscar o entendimento das diversas ferramentas oferecidas pela biblioteca.

Conceitos Básicos

Estrutura de um programa

- Todo programa possui uma estrutura de código onde é preciso abstrair os scripts internos e entregar funcionalidades para o usuário;
- Usando como exemplo básico uma calculadora, ela deve possuir uma interface para o usuário enviar dados, escolher a opção de cálculo (funções matemáticas internas) e um retorno que é basicamente o que o usuário espera.

Elementos

- Correspondem aos elementos que compõem uma interface gráfica;
- É preciso prover interfaces que ofereçam uma boa experiência de uso de aplicação;
- Dentre os principais elementos podemos citar: botões, caixa de textos, textos, imagens entre outros elementos.

Eventos

- Devemos dedicar grande parte dos nossos esforços para garantir que os elementos sejam gatilhos para acionar funcionalidades;
- Uma vez aberto o programa, o usuário terá acesso as ferramentas. Essas ferramentas são chamadas de eventos.



PyQT6


- Vamos precisar instalar essa biblioteca dentro do seu projeto;
- Para nossos estudos vamos estar utilizando o VS Code como ambiente de desenvolvimento. Na medida que avançarmos, estaremos inserindo novas ferramentas dentro do contexto;

PyQT6

- A biblioteca PyQt não vem por padrão, por isso precisamos instalar ela através do terminal;
- O comando para instalação é o `pip install pyqt6`;
- Para ter certeza que a biblioteca foi instalada execute o comando `pip list` . O PyQt deve aparecer na lista de bibliotecas.

PyQT6

- Crie um arquivo dentro da sua pasta e import duas bibliotecas `sys` e `PyQT6`. Se tudo estiver correto as bibliotecas devem ficar na cor acinzentada (significa que a biblioteca foi importada mas ainda não utilizada.)

```
 main.py  
1  import sys  
2  import PyQt6
```

Criando uma Janela Principal

- A biblioteca PyQt utiliza o padrão POO onde dentro da classe principal estaremos inserindo os elementos com os respectivos eventos;
- Sendo assim, crie uma estrutura inicial igual a estrutura a seguir:

Janela Principal

```
main.py > ...
1  import sys
2  from PyQt6.QtWidgets import *
3
4  class JanelaPrincipal(QWidget): #classe principal que vai conter elementos
5      def __init__(self):
6          super().__init__()
7          self.setWindowTitle("Minha Janela PyQt6")
8          # self.setGeometry(100, 100, 800, 600)# (borda esquerda, superior,largura, altura )
9          self.show()# faz a tela ser exibida
10
11  qt = QApplication(sys.argv) #variavel qt instanciando a classe QApplication: permite usar recursos do SO
12  app = JanelaPrincipal() #instaciando a classe
13  sys.exit(qt.exec()) #encerra totalmente a aplicação assim que fechada
```

Fixando os limites da janela

- No exemplo anterior, caso o usuário tente ajustar o tamanho da tela, ao clicar na borda a mesma vai mudar.
- Para “travar” esse ajuste devemos inserir o método `setFixedSize(self.size())`

```
def __init__(self):  
    super().__init__()  
    self.setWindowTitle("Minha Janela PyQt6")  
    self.setFixedSize(self.size())  
    self.setGeometry(100, 100, 800, 600)# (bor  
    self.show()# faz a tela ser exibida
```


Utilizando ícones na janela

- Precisamos importar QIcon;
- Dentro do método construtor inserimos mais uma linha e apontamos o lugar onde salvou a imagem.

```
1  import sys
2  from PyQt6.QtWidgets import *
3  from PyQt6.QtGui import QIcon #Importando recurso específico de ícones
4
5  class JanelaPrincipal(QWidget): #classe principal que vai conter elementos
6      def __init__(self):
7          super().__init__()
8          self.setWindowTitle("Minha Janela PyQt6")
9          self.setFixedSize(self.size())
10         self.setGeometry(100, 100, 800, 600) # (borda esquerda, superior, largura, altura )
11         self.setWindowIcon(QIcon('senac.png')) #apontando para imagem que está no mesmo diretório
12         self.show() # faz a tela ser exibida
```

Inserindo bloco de elementos

- Vamos criar uma estrutura onde iremos inserir os blocos de códigos referentes aos elementos visuais.
- Essa `interface()` vai conter botões que com funções específicas ou textos que terão outras funções. Em resumo, cada elemento possui as suas particularidades.

QLabel

- Seguindo essa lógica, uma das estruturas que vamos estar utilizando com frequência são as Labels.
- Essas estruturas acondicionam textos que serão elementos visuais das janelas;
- Vamos criar dentro de interface() uma variável de nome texto1. Esse objeto vai instanciar e utilizar o método QLabel() que recebe uma string. Seu propósito é inserir elementos textuais.

QLabel – resize()

- Na sequência podemos aplicar sobre o texto1 o método `resize()` que vai receber dois parâmetros que representam o tamanho do bloco onde o elemento está inserido;
- Como não é o propósito fazer com que um elemento sobreponha outro, é necessário utilizar esse método para evitar isso.

QLabel – move()

- Vamos definir a posição do elemento label através do método `move()` ;
- Esse método recebe dois parâmetros: o primeiro corresponde ao número de pixels contados desde a borda esquerda, o segundo corresponde a posição do elemento em relação ao topo da janela.

QLabel

```
def __init__(self):  
    super().__init__()  
    self.setWindowTitle("Minha Janela PyQt6")  
    self.setFixedSize(self.size())  
    self.setGeometry(100, 100, 800, 600) # (border  
    self.setWindowIcon(QIcon('senac.png')) #apc  
    self.Interface()  
    self.show() # faz a tela ser exibida
```

```
14     def interface(self):  
15         texto1 = QLabel(' Olá, Victor Sotero!', self)  
16         texto1.resize(250, 250)  
17         texto1.move(100, 50)  
18         texto2 = QLabel(' Tricolor de coração!', self)  
19         texto2.resize(250, 250)  
20         texto2.move(200, 50)
```

Inserindo botões e atribuindo funções

- Para o início de nossos estudos, vamos utilizar o elemento QPushButton e utilizar, também, o método move() para posicioná-lo;
- Vamos adicionar a opção SAIR em um botão;
- Primeiro precisamos criar o evento que vai chamar a função, depois criamos a própria função;
- Utilizamos o método `clicked.connect()` para chamar a função sair.

Inserindo Botões e atribuindo funções

```
21         botao1 = QPushButton('SAIR',self)# criando o botão
22         botao1.move(100,200)
23         botao1.clicked.connect(self.sair) #ação ao ser clicado
24     def sair(self):# função sair que ao ser chamada vai encerrar a aplicação
25         sys.exit(qt.exec())
```


Alterando componentes ao pressionar o botão

- Vamos aplicar uma formatação nos componentes de textos utilizados anteriormente;
- Desse modo, ao clicar em um segundo botão NOME ele vai pegar um label e colocar todas as letras em maiúsculo;
- Para isso precisamos utilizar a função `setText()` para selecionar a label específica e mudar.

Alterando componentes ao pressionar o botão

É preciso colocar o self para permitir que o objeto seja acessado por outros métodos.

```
23 self.texto2 = QLabel(' Tricolor de coração!', self)
24 self.texto2.resize(250, 250)
25 self.texto2.move(200,50)
26
27 botao2 = QPushButton('NOME',self)# criando o botão
28 botao2.move(200,200)
29 botao2.clicked.connect(self.muda_maiusculo) #ação ao ser clicado
```

```
31 def muda_maiusculo(self):
32     self.texto2.setText('TRICOLOR DE CORAÇÃO!')
```

- Continuamos...