Qt para Python

1. **Visão Geral**

   Qt é um “Widget Tool Kit” multiplataforma de código aberto que é amplamente usado em Linux, Windows e MacOs e mantido pela “The Qt Company”. É usado para criar a parte gráfica/visual de uma aplicação.
   A biblioteca Qt é famosa por ser extremamente bem documentada. Apesar da documentação ser para C++, é fácil "traduzir" os exemplos para Python.
   A biblioteca Qt é subdividida em vários módulos, entre os principais estão:

   - **Qt Core**: Fornece funcionalidade não-GUI principal, como sinal e slots, propriedades, classes básicas de modelos de itens e serialização.
   - **Qt GUI**: Estende QtCore com funcionalidade GUI: Eventos, janelas e telas, OpenGL e pintura 2D baseada em raster, bem como imagens.
   - **Qt Widgets**: Fornece widgets prontos para usar para seu aplicativo, incluindo elementos gráficos para sua IU

   Utilizaremos neste documento em grande parte o módulo Qt Widgets, que contém as classes dos componentes básicos que iremos precisar.
   Utilizaremos o Qt a partir do PySide2, que é um wrapper multiplataforma da linguagem Python para a biblioteca Qt.

   O material deste estudo está disponível em: https://github.com/leuribeiru/QtforPhyton

2. **Instalação e configuração do ambiente**

   a. **Python e pip**

      Faça o download do Python em “https://www.python.org/downloads/”
      As versões acima da 2.7.9 ou 3.4 o pip já está incluso, caso sua versão não possua o pip instalado você pode baixar e instalá-lo manualmente de acordo com as instruções em https://pip.pypa.io/en/stable/installing/
      Caso tenha algum problema com o pip, você pode atualizá-lo com os comandos abaixo:
      Linux: `pip install -U pip`
      Windows: `python -m pip install -U pip`

   b. **Ambiente virtual (virtualenv)**

      É importante criar um ambiente virtual, para que dessa forma, projetos diferentes possam usar dependências em versões diferentes, sem que isso possa afetar o funcionamento de cada um. Para isso, neste projeto iremos utilizar o virtualenv, mas você pode usar qualquer ambiente de desenvolvimento, virtual ou não. Caso queira ignorar esta etapa, pule para o item **f** .
      Para instalar o virtualenv utilize os comandos abaixo:
      Linux: `pip install virtualenv`
      Windows: `python -m pip install virtualenv`

c. **Criação de uma virtualenv**
Para criar uma virtualenv é simples, basta utilizar o comando abaixo dentro da pasta onde a virtualenv será criada:
Linux/Windows: `virtualenv nome_da_virtualenv`

d. **Ativando a virtualenv**
Após a criação da virtualenv, precisamos ativá-la, para que todos os pacotes instalados sejam referentes a esta virtualenv, para isso executamos os comandos abaixo:
Linus: `source nome_da_virtualenv/bin/activate`
Windows: `nome_da_virtualenv\Scripts\activate`

e. **Desativando a virtualenv**
Este passo não deve ser executado, é somente para conhecimento, caso queira deixar de trabalhar na virtualenv que está ativa basta executar o comando abaixo:
Linux/Windows: `deactivate`

f. **Instalando o Qt**
Para instalar o Qt para python basta executar o comando abaixo, lembre-se, se estiver utilizando uma virtualenv, faça a instalação com a virtualenv ativada:
Linux: `pip install PySide2`
Windows: `python -m pip install PySide2`

g. **Verifique se tudo deu certo**
Crie um arquivo nome_do_arquivo.py inclua o conteúdo abaixo:

```python
import PySide2.QtCore

print("PySide2 version: " + PySide2.__version__)

print("PySide2 Core version: " + PySide2.QtCore.__version__)

```

GitHub: *teste.py*

Salve o arquivo e execute com o comando:
`python nome_do_arquivo.py`
A saída deve exibir as informações de versão do PySide2 e versão do Core do PySide2:

```
PySide2 version: 5.15.1
PySide2 Core version: 5.15.1
```

3. **Componentes básicos**

Para qualquer aplicativo GUI usando Qt, há precisamente um objeto QApplication, não importa se o aplicativo tem 0, 1 ou mais janelas. Este objeto recebe os parâmetros do sistema (sys.argv)

Primeiro, vamos criar uma janela, para exibir os componentes básicos, abaixo segue um exemplo de como criar uma janela básica:

```python
import sys
from PySide2.QtWidgets import QApplication, QMainWindow

aplicacao = QApplication(sys.argv)

janela = QMainWindow()
# setGeometry(esquerda, topo, largura, altura)
janela.setGeometry( 100, 50, 300, 200 )
janela.setWindowTitle("Primeira Janela")
janela.show()

aplicacao.exec_()
sys.exit()
```

GitHub: *componentes_basicos/janela.py*

Criamos uma instância da classe QMainWindow, que é a classe que cria uma janela, e configuramos a posição e tamanho utilizando o método setGeometry, e alteramos o título da janela usando o método setWindowTitle.
O método show() é chamado depois que toda configuração de visualização já estiver configurada

É necessário chamar o método exec_() do objeto QApplication para que a aplicação seja executada.
Por último executamos sys.exit() para finalizar a execução da aplicação.

a. Botão

A classe QPushButton é responsável por renderizar um botão dentro da janela.

Criamos um botão, configuramos sua posição dentro da janela, o tamanho, e aplicamos o estilo para alterar a cor de fundo e a cor da letra no botão (linhas 9 a 17):

```
1  import sys
2  from PySide2.QtWidgets import QApplication, QMainWindow, QPushButton
3
4  aplicacao = QApplication(sys.argv)
5  janela = QMainWindow()
6  # setGeometry(esquerda, topo, largura, altura)
7  janela.setGeometry( 100, 50, 300, 200 )
8  janela.setWindowTitle("Primeira Janela")
9  # instância de um botão dentro da janela
10 botao = QPushButton("Meu Botão", janela)
11 # posição dentro da janela (esquerda, topo)
12 botao.move(50,50)
13 # tamanho (largura, altura)
14 botao.resize(200,100)
15 # estilo do botão
16 botao.setStyleSheet("QPushButton \
17 {background-color: blue; color: white; font-size: 32px}")
18 janela.show()
19 aplicacao.exec_()
20 sys.exit()
```

GitHub: *componentes_basicos/botao.py*

Note que o método show() de QMainWindow deve ser executado somente depois que todos os componentes são configurados dentro da janela, caso contrário não serão renderizados na janela.

b. Caixa de entrada de texto

Para criarmos uma caixa de entrada de texto utilizamos a classe QLineEdit para instanciamos um elemento do tipo caixa de texto dentro da janela,, configuramos sua posição, tamanho, e aplicamos o estilo próprio (linhas 9 a 17):

```
1  import sys
2  from PySide2.QtWidgets import QApplication, QMainWindow, QLineEdit
3
4  aplicacao = QApplication(sys.argv)
5  janela = QMainWindow()
6  # setGeometry(esquerda, topo, largura, altura)
7  janela.setGeometry( 100, 50, 300, 200 )
8  janela.setWindowTitle("Primeira Janela")
9  # cria um campo de entrada de texto na janela
10 texto_entrada = QLineEdit(janela)
11 # posição (esquerda, topo)
12 texto_entrada.move(50, 75)
13 # tamanho (largura, altura)
14 texto_entrada.resize(200, 50)
15 # estilo da caixa de texto
16 texto_entrada.setStyleSheet('QLineEdit \
17 {font-size: 22px; font-style: italic; padding: 15px}')
18 janela.show()
19 aplicacao.exec_()
20 sys.exit()
```

GitHub: *componentes_basicos/texto_entrada.py*

c. Label de texto

A classe QLabel é responsável por criar uma instância de um label de texto. No exemplo, criamos um objeto do tipo label dentro da janela, em seguida, assim como nos outros elementos que criamos, configuramos sua posição, tamanho, e aplicamos o estilo próprio (linhas 10 a 18):

```
1   import sys
2   from PySide2.QtWidgets import QApplication, QMainWindow, QLabel
3
4   aplicacao = QApplication(sys.argv)
5
6   janela = QMainWindow()
7   # setGeometry(esquerda, topo, largura, altura)
8   janela.setGeometry( 100, 50, 300, 200 )
9   janela.setWindowTitle("Primeira Janela")
10  # cria um label dentro da janela
11  label = QLabel(janela)
12  # configura o texto do label
13  label.setText("Label 1")
14  # posiciona o label na janela (esquerda, topo)
15  label.move(100, 50)
16  # aplica um estilo ao label
17  label.setStyleSheet('QLabel \
18  {font-size: 20px; color: green; background-color: white; }')
19  janela.show()
20  aplicacao.exec_()
21  sys.exit()
```

GitHub: *componentes_basicos/label.py*

d. Radio Button

Os radio buttons são objetos de seleção exclusiva, ou seja, somente uma opção pode ser escolhida por vez, para isso é importante separá-los em um grupo, desta forma podemos ter 1 ou mais grupos de radio buttons independentes. Para isso, no exemplo, utilizaremos a classe QGroupBox junto com a classe QRadioButton, o primeiro para criarmos a área onde ficarão os radio buttons e o segundo para os próprios radio buttons (linhas 10 a 23):

```
1  import sys
2  from PySide2.QtWidgets import QApplication, QMainWindow, QGroupBox, QRadioButton
3
4  aplicacao = QApplication(sys.argv)
5
6  janela = QMainWindow()
7  # setGeometry(esquerda, topo, largura, altura)
8  janela.setGeometry( 100, 50, 300, 200 )
9  janela.setWindowTitle("Primeira Janela")
10 # cria uma instancia de um grupo de seleção dentro da janela
11 group_box = QGroupBox("Selecione uma opção", janela)
12 group_box.move(50,50)
13 group_box.resize(200,100)
14 group_box.setStyleSheet('QGroupBox \
15 {background-color: yellow}')
16 # cria os radio buttons dentro do grupo de seleção
17 radio_btn_1 = QRadioButton("Opção 1", group_box)
18 radio_btn_1.move(10,20)
19 radio_btn_2 = QRadioButton("Opção 2", group_box)
20 radio_btn_2.move(10,40)
21 radio_btn_3 = QRadioButton("Opção 3", group_box)
22 radio_btn_3.move(10,60)
23 radio_btn_3.setChecked(True)
24 janela.show()
25
26 aplicacao.exec_()
27 sys.exit()
```

GitHub: *componentes_basicos/radio.py*

Note que quando criamos o QGroupBox, colocamos ele dentro da janela, e quando criamos os QRadioButton colocamos eles dentro do QGroupBox.

e. Checkbox

Diferente do radio button, um checkbox não é de seleção exclusiva, ou seja, várias opções podem ser selecionadas. Para criarmos os checkbox, usamos a classe QCheckbox e instanciamos um objeto dentro da janela, como mostra o código a seguir (linhas 14 a 20):

```
1  import sys
2  from PySide2.QtWidgets import QApplication, QMainWindow, QLabel, QCheckBox
3
4  aplicacao = QApplication(sys.argv)
5
6  janela = QMainWindow()
7  # setGeometry(esquerda, topo, largura, altura)
8  janela.setGeometry( 100, 50, 300, 200 )
9  janela.setWindowTitle("Primeira Janela")
10 # cria uma instancia de um grupo de seleção dentro da janela
11 label = QLabel("Selecione uma ou mais opções", janela)
12 label.move(30,30)
13 label.resize(200,20)
14 # cria os checkbox dentro do grupo de seleção
15 checkbox_1 = QCheckBox("Opção 1", janela)
16 checkbox_1.move(30,60)
17 checkbox_2 = QCheckBox("Opção 2", janela)
18 checkbox_2.move(30,80)
19 checkbox_3 = QCheckBox("Opção 3", janela)
20 checkbox_3.move(30,100)
21 janela.show()
22
23 aplicacao.exec_()
24 sys.exit()
```

GitHub: *componentes_basicos/checkbox.py*

f. Lista suspensa

Para utilizar uma lista suspensa utilizamos uma instância de QComboBox, e adicionamos os itens através do método addItem ou addItems, o primeiro adiciona somente um item por vez enquanto o segundo adiciona todos os itens de um array (linhas 10 a 17):

```
1  import sys
2  from PySide2.QtWidgets import QApplication, QMainWindow, QComboBox
3
4  aplicacao = QApplication(sys.argv)
5
6  janela = QMainWindow()
7  # setGeometry(esquerda, topo, largura, altura)
8  janela.setGeometry( 100, 50, 300, 200 )
9  janela.setWindowTitle("Primeira Janela")
10 # cria uma lista suspensa
11 lista_suspensa = QComboBox(janela)
12 lista_suspensa.move(100,90)
13 lista_suspensa.resize(100, 20)
14 # adiciona um item à lista suspensa
15 lista_suspensa.addItem("Opção 1")
16 # adiciona vários itens de uma vez na lista suspensa
17 lista_suspensa.addItems(["Opção 2", "Opção 3", "Opção 4"])
18
19 janela.show()
20
21 aplicacao.exec_()
22 sys.exit()
```

GitHub: *componentes_basicos/lista_suspensa.py*

g. Slider

Para criar um slider dentro da janela é preciso utilizar a classe QSlide. E para mudarmos a orientação do slider é preciso importar o Qt, de PySide2.QtCore, para usarmos a constante Qt.Horizontal no método setOrientation do slider (linhas 12 a 17):

```
1   import sys
2   from PySide2.QtWidgets import QApplication, QMainWindow, QSlider
3   from PySide2.QtCore import Qt
4
5   aplicacao = QApplication(sys.argv)
6
7   janela = QMainWindow()
8   # setGeometry(esquerda, topo, largura, altura)
9   janela.setGeometry( 100, 50, 300, 200 )
10  janela.setWindowTitle("Primeira Janela")
11
12  # cria um objeto slider dentro da janela
13  slider = QSlider(janela)
14  slider.resize(100,30)
15  slider.move(100, 85)
16  # modifica a orientação do slider
17  slider.setOrientation(Qt.Horizontal)
18
19  janela.show()
20
21  aplicacao.exec_()
22  sys.exit()
```

GitHub: *componentes_basicos/slider.py*

4. **Obtenção de atributos**

   Nesta seção será demonstrado como obter os atributos dos componentes, para demonstração, os atributos serão obtidos no evento de clique no botão dos exemplos, não é preciso entender agora o evento de clique no botão, pois este será utilizado como padrão somente para entendimento de como obter os atributos e abordado posteriormente.

   a. **Conteúdo de uma caixa de texto**

      O valor contido em uma caixa de texto é obtido através do método text() do componente QLineEdit (linha 23):

```python
import sys
from PySide2.QtWidgets import QApplication, QMainWindow, QLineEdit, QLabel, QPushButton

aplicacao = QApplication(sys.argv)
janela = QMainWindow()
janela.setGeometry( 100, 50, 300, 200 )
janela.setWindowTitle("Primeira Janela")

texto_entrada = QLineEdit(janela)
texto_entrada.move(50, 10)
texto_entrada.resize(200, 30)

label = QLabel("Texto Obtido:", janela)
label.move(50, 90)
label.resize(200,30)

resultado = QLabel("", janela)
resultado.move(50, 120)
resultado.resize(200,30)

def onButtonClick():
    # atribui o valor da caixa de texto para o label resultado
    resultado.setText(texto_entrada.text())
    # texto_entrada.text() obtém o valor que está na caixa de texto

button = QPushButton("Obter texto", janela)
button.move(50, 50)
button.resize(205,30)
# determina a função que sera executada ao clicar no botão
button.clicked.connect(onButtonClick)

janela.show()
aplicacao.exec_()
sys.exit()
```

GitHub: *obtencao_de_atributos/texto_entrada.py*

b. Opção selecionada radio/checkbox/lista suspensa

O método isChecked() da classe QRadoiButton retorna True caso esteja marcado e False caso não esteja marcado. Desta forma basta verificar qual deles está marcado (linhas 32 a 37).

```
1  import sys
2  from PySide2.QtWidgets import QApplication, QMainWindow, QGroupBox, QRadioButton, QPushButton, QLabel
3
4  aplicacao = QApplication(sys.argv)
5
6  janela = QMainWindow()
7  janela.setGeometry( 100, 50, 300, 300 )
8  janela.setWindowTitle("Primeira Janela")
9  group_box = QGroupBox("Selecione uma opção", janela)
10 group_box.move(50,20)
11 group_box.resize(200,100)
12 radio_btn_1 = QRadioButton("Opção 1", group_box)
13 radio_btn_1.move(10,20)
14 radio_btn_2 = QRadioButton("Opção 2", group_box)
15 radio_btn_2.move(10,40)
16 radio_btn_3 = QRadioButton("Opção 3", group_box)
17 radio_btn_3.move(10,60)
18 radio_btn_3.setChecked(True)
19
20 label = QLabel("Opção selecionada: ", janela)
21 label.move(50, 190)
22 label.resize(200,30)
23
24 resultado = QLabel("", janela)
25 resultado.move(50, 210)
26 resultado.resize(200,30)
27
28 def onButtonClicked():
29     checked = ""
30     # verifica qual radiobutton esta selecionado e
31     # atribui a variavel checked o texto correspondente
32     if(radio_btn_1.isChecked()):
33         checked = radio_btn_1.text()
34     elif(radio_btn_2.isChecked()):
35         checked = radio_btn_2.text()
36     elif(radio_btn_3.isChecked()):
37         checked = radio_btn_3.text()
38     # atribui a label resultado o valor do radio button selecionado
39     resultado.setText(checked)
40
41 botao = QPushButton("Obter opção selecionada", janela)
42 botao.move(50, 140)
43 botao.resize(200, 30)
44 botao.clicked.connect(onButtonClicked)
45
46 janela.show()
47
48 aplicacao.exec_()
49 sys.exit()
```

GitHub: *obtencao_de_atributos/radio.py*

O checkbox também tem o método isChecked(), que retorna True se o campo estiver marcado e False se não estiver marcado, porém é preciso verificar todos os checkboxes mesmo que algum já esteja marcado, pois neste caso mais de uma opção pode ser selecionada.(linhas 32 a 37)

```python
import sys
from PySide2.QtWidgets import QApplication, QMainWindow, QLabel, QCheckBox, QPushButton, QLabel

aplicacao = QApplication(sys.argv)

janela = QMainWindow()
janela.setGeometry( 100, 50, 300, 300 )
janela.setWindowTitle("Primeira Janela")
label = QLabel("Selecione uma ou mais opções", janela)
label.move(50,20)
label.resize(200,20)
checkbox_1 = QCheckBox("Opção 1", janela)
checkbox_1.move(50,40)
checkbox_2 = QCheckBox("Opção 2", janela)
checkbox_2.move(50,60)
checkbox_3 = QCheckBox("Opção 3", janela)
checkbox_3.move(50,80)
checkbox_3.setChecked(True)

label_resultado = QLabel("Opções selecionadas: ", janela)
label_resultado.move(50, 190)
label_resultado.resize(200,30)

resultado = QLabel("", janela)
resultado.move(50, 210)
resultado.resize(200,60)

def onButtonClicked():
    checked = ""
    # verifica quais checkboxes estão selecionados e
    # atribui a variavel checked o texto correspondente
    if(checkbox_1.isChecked()):
        checked += "\n" + checkbox_1.text()
    if(checkbox_2.isChecked()):
        checked += "\n" + checkbox_2.text()
    if(checkbox_3.isChecked()):
        checked += "\n" + checkbox_3.text()
    # atribui a label resultado o valor dos checkboxes button selecionado
    resultado.setText(checked)

botao = QPushButton("Obter opções selecionadas", janela)
botao.move(50, 140)
botao.resize(200, 30)
botao.clicked.connect(onButtonClicked)

janela.show()

aplicacao.exec_()
sys.exit()
```

GitHub: [obtencao_de_atributos/checkbox.py]

Para obter o atributo selecionado em uma lista suspensa basta verificar a opção corrente na lista suspensa, o método currentText() retorna o texto da opção que está selecionada no momento. (linha 24)

```python
import sys
from PySide2.QtWidgets import QApplication, QMainWindow, QComboBox, QLabel, QPushButton

aplicacao = QApplication(sys.argv)

janela = QMainWindow()
janela.setGeometry( 100, 50, 300, 200 )
janela.setWindowTitle("Primeira Janela")
lista_suspensa = QComboBox(janela)
lista_suspensa.move(100,20)
lista_suspensa.resize(100, 20)
lista_suspensa.addItem("Opção 1")
lista_suspensa.addItems(["Opção 2", "Opção 3", "Opção 4"])

label = QLabel("Opção selecionada: ", janela)
label.move(100,80)
label.resize(100,20)

resultado = QLabel("", janela)
resultado.move(100,110)
resultado.resize(100,20)

def onButtonClicked():
    resultado.setText(lista_suspensa.currentText())

botao = QPushButton("Obter opção selecionada", janela)
botao.move(80, 50)
botao.resize(140,20)
botao.clicked.connect(onButtonClicked)

janela.show()

aplicacao.exec_()
sys.exit()
```

GitHub: *obtencao_de_atributos/lista_suspensa.py*

c. Valor marcado no slider

Um slider tem o método value(), que retorna o valor correspondente a posição do slider (linha 27), note também que foi configurado um limite de valor mínimo e máximo para o slider.

```
1  import sys
2  from PySide2.QtWidgets import QApplication, QMainWindow, QSlider, QPushButton, QLabel
3  from PySide2.QtCore import Qt
4
5  aplicacao = QApplication(sys.argv)
6
7  janela = QMainWindow()
8  janela.setGeometry( 100, 50, 300, 200 )
9  janela.setWindowTitle("Primeira Janela")
10
11 slider = QSlider(janela)
12 slider.resize(100,30)
13 slider.move(100, 20)
14 slider.setOrientation(Qt.Horizontal)
15 slider.setMinimum(0)
16 slider.setMaximum(100)
17
18 label = QLabel("Valor do slider", janela)
19 label.move(100,90)
20 label.resize(100,20)
21
22 resultado = QLabel("", janela)
23 resultado.move(100,120)
24 label.resize(100,20)
25
26 def onButtonClicked():
27     resultado.setText(str(slider.value()))
28
29 botao = QPushButton("Obter valor", janela)
30 botao.move(100, 60)
31 botao.resize(100,20)
32 botao.clicked.connect(onButtonClicked)
33
34 janela.show()
35
36 aplicacao.exec_()
37 sys.exit()
```

GitHub: *obtencao_de_atributos/slider.py*

5. Tratamento básico de eventos

No Qt para Python, os componentes tem atributos que representam os possíveis eventos, e a função que será executada é passada para o método connect() desses atributos, verificando os exemplos abaixo esta explicação fará mais sentido.

a. Clique no botão

No código abaixo foram criados 2 botões na janela, que quando clicados, exibe a informação de qual botão foi clicado. O atributo clicked é que captura o evento de clique no botão, e a função que será executada é passada no parâmetro do método connect() do atributo clicked (linhas 24 a 27).

```
1  import sys
2  from PySide2.QtWidgets import QApplication, QMainWindow, QPushButton, QLabel
3
4  aplicacao = QApplication(sys.argv)
5  janela = QMainWindow()
6  janela.setGeometry( 100, 50, 350, 200 )
7
8  label = QLabel("", janela)
9  label.move(100,100)
10 label.resize(200,50)
11 botao1 = QPushButton("Botão 1", janela)
12 botao1.move(50,30)
13 botao1.resize(100,50)
14 botao2 = QPushButton("Botão 2", janela)
15 botao2.move(200,30)
16 botao2.resize(100,50)
17
18 def onButton1Clicked():
19     label.setText("Clicou no botão 1")
20
21 def onButton2Clicked():
22     label.setText("Clicou no botão 2")
23
24 # define qual função será executada quando o botão 1 for clicado
25 botao1.clicked.connect(onButton1Clicked)
26 # define qual função será executada quando o botão 2 for clicado
27 botao2.clicked.connect(onButton2Clicked)
28
29 janela.show()
30 aplicacao.exec_()
31 sys.exit()
```

GitHub: *tratamento_basico_de_evento/botao.py*

b. <enter> no input

Para capturar que o usuário teclou <enter> no input de texto, o QEditLine oferece o atributo returnPressed, e para definir a ação que será realizada basta passar a função que será executada como parâmetro para o método connect() (linha 25). Neste exemplo apenas exibimos o mesmo texto que está no QEditLine no momento que o usuário pressiona a tecla <enter>

```
1  import sys
2  from PySide2.QtWidgets import QApplication, QMainWindow, QLineEdit, QLabel
3
4  aplicacao = QApplication(sys.argv)
5  janela = QMainWindow()
6  janela.setGeometry( 100, 50, 300, 200 )
7  janela.setWindowTitle("Primeira Janela")
8
9  texto_entrada = QLineEdit(janela)
10 texto_entrada.move(50, 30)
11 texto_entrada.resize(200, 30)
12
13 label = QLabel("Escreva algo e aperte <enter>", janela)
14 label.move(50, 60)
15 label.resize(200,30)
16
17 result = QLabel("", janela)
18 result.move(50,90)
19 result.resize(200,30)
20
21 def onReturnPressed():
22     result.setText(texto_entrada.text())
23
24 # quando o usuário pressionar <enter> na caixa de texto
25 texto_entrada.returnPressed.connect(onReturnPressed)
26
27 janela.show()
28 aplicacao.exec_()
29 sys.exit()
```

GitHub: *tratamento_basico_de_evento/texto_entrada.py*

c. Marcar/desmarcar radio/checkbox

Para capturar o evento de mudança de status no radio button usamos o atributo clicked, já que o valor muda de acordo com a interação do usuário, e passamos a função que será executada no método connect() (linhas 37 a 39)

```
1  import sys
2  from PySide2.QtWidgets import QApplication, QMainWindow, QGroupBox, QRadioButton, QLabel
3
4  aplicacao = QApplication(sys.argv)
5
6  janela = QMainWindow()
7  # setGeometry(esquerda, topo, largura, altura)
8  janela.setGeometry( 100, 50, 300, 220 )
9  janela.setWindowTitle("Primeira Janela")
10
11 label = QLabel("Radio button clicada", janela)
12 label.move(50,120)
13 label.resize(100, 30)
14
15 result = QLabel("", janela)
16 result.move(50,150)
17 result.resize(100,60)
18
19 group_box = QGroupBox("Selecione uma opção", janela)
20 group_box.move(50,20)
21 group_box.resize(200,100)
22 radio_btn_1 = QRadioButton("Opção 1", group_box)
23 radio_btn_1.move(10,20)
24 radio_btn_2 = QRadioButton("Opção 2", group_box)
25 radio_btn_2.move(10,40)
26 radio_btn_3 = QRadioButton("Opção 3", group_box)
27 radio_btn_3.move(10,60)
28 radio_btn_3.setChecked(True)
29
30 def onClickRadio():
31     text = ""
32     text += "radio button 1: " + str(radio_btn_1.isChecked())
33     text += "\nradio button 2: " + str(radio_btn_2.isChecked())
34     text += "\nradio button 3: " + str(radio_btn_3.isChecked())
35     result.setText(text)
36
37 radio_btn_1.clicked.connect(onClickRadio)
38 radio_btn_2.clicked.connect(onClickRadio)
39 radio_btn_3.clicked.connect(onClickRadio)
40
41 onClickRadio()
42 janela.show()
43
44 aplicacao.exec_()
45 sys.exit()
```

GitHub: *tratamento_basico_de_evento/radio.py*

Para checkbox também usamos o atributo clicked e passamos a função a ser executada quando o usuário interagir com o checkbox para o método connect() (linhas 36 a 38).

```
1  import sys
2  from PySide2.QtWidgets import QApplication, QMainWindow, QGroupBox, QCheckBox, QLabel
3
4  aplicacao = QApplication(sys.argv)
5
6  janela = QMainWindow()
7  janela.setGeometry( 100, 50, 300, 200 )
8  janela.setWindowTitle("Primeira Janela")
9
10 label = QLabel("Checkboxes selecionados", janela)
11 label.move(50,120)
12 label.resize(130, 30)
13
14 result = QLabel("", janela)
15 result.move(50,140)
16 result.resize(100,60)
17
18 group_box = QGroupBox("Selecione uma ou mais opções", janela)
19 group_box.move(50,20)
20 group_box.resize(200,100)
21
22 checkbox_1 = QCheckBox("Opção 1", group_box)
23 checkbox_1.move(10,20)
24 checkbox_2 = QCheckBox("Opção 2", group_box)
25 checkbox_2.move(10,40)
26 checkbox_3 = QCheckBox("Opção 3", group_box)
27 checkbox_3.move(10,60)
28
29 def onClickCheckBox():
30     text = ""
31     text += "checkbox 1: " + str(checkbox_1.isChecked())
32     text += "\ncheckbox 2: " + str(checkbox_2.isChecked())
33     text += "\ncheckbox 3: " + str(checkbox_3.isChecked())
34     result.setText(text)
35
36 checkbox_1.clicked.connect(onClickCheckBox)
37 checkbox_2.clicked.connect(onClickCheckBox)
38 checkbox_3.clicked.connect(onClickCheckBox)
39
40 onClickCheckBox()
41 janela.show()
42
43 aplicacao.exec_()
44 sys.exit()
```

GitHub: *tratamento_basico_de_evento/checkbox.py*

d. Alteração do valor da lista suspensa

Para detectar o evento de mudança no valor selecionado de uma lista suspensa a classe QComboBox oferece o atributo currentTextChanged, e para definir qual ação será realizada quando o valor muda passamos a função que será executada como parâmetro do método connect() (linha 22)

```
1  import sys
2  from PySide2.QtWidgets import QApplication, QMainWindow, QComboBox, QLabel
3
4  aplicacao = QApplication(sys.argv)
5
6  janela = QMainWindow()
7  janela.setGeometry( 100, 50, 300, 200 )
8  janela.setWindowTitle("Primeira Janela")
9  label = QLabel("Opção selecionada", janela)
10 label.move(100,60)
11 label.resize(100,30)
12 resultado = QLabel("", janela)
13 resultado.move(100,90)
14 resultado.resize(100,30)
15 lista_suspensa = QComboBox(janela)
16 lista_suspensa.move(100,30)
17 lista_suspensa.resize(100, 20)
18 lista_suspensa.addItem("Opção 1")
19 lista_suspensa.addItems(["Opção 2", "Opção 3", "Opção 4"])
20 def onCurrentTextChanged():
21     resultado.setText(lista_suspensa.currentText())
22 lista_suspensa.currentTextChanged.connect(onCurrentTextChanged)
23
24 onClickComboBox()
25 janela.show()
26
27 aplicacao.exec_()
28 sys.exit()
```

GitHub: *tratamento_basico_de_evento/lista_suspensa.py*

e. Alteração do valor do slider

Para detectar o evento de mudança de valor no slider, enquanto desliza a barra, a classe QSlider oferece o atributo onValueChanged, para definir a ação que será realizada basta passar no parâmetro do método connect() a função que será executada. (linha 26)

```
1  import sys
2  from PySide2.QtWidgets import QApplication, QMainWindow, QSlider, QPushButton, QLabel
3  from PySide2.QtCore import Qt
4
5  aplicacao = QApplication(sys.argv)
6
7  janela = QMainWindow()
8  janela.setGeometry( 100, 50, 300, 200 )
9  janela.setWindowTitle("Primeira Janela")
10
11 label = QLabel("Valor do slider", janela)
12 label.move(100,90)
13 label.resize(100,20)
14 resultado = QLabel("", janela)
15 resultado.move(100,120)
16 label.resize(100,20)
17
18 slider = QSlider(janela)
19 slider.resize(100,30)
20 slider.move(100, 20)
21 slider.setOrientation(Qt.Horizontal)
22 slider.setMinimum(0)
23 slider.setMaximum(100)
24 def onValueChanged():
25     resultado.setText(str(slider.value()))
26 slider.valueChanged.connect(onValueChanged)
27
28 onValueChanged()
29 janela.show()
30
31 aplicacao.exec_()
32 sys.exit()
```

GitHub: *tratamento_basico_de_evento/slider.py*

6. **Exemplo Prático**
    a. **Janela Gráfica com slider de 0 a 100**
    b. **O usuário marca no slider a nota que ele acredita que um teste de usabilidade System Usability Scale irá dar (hipótese)**
    c. **Depois, clica no botão e seleciona o CSV gerado por um formulário de entrevista do SUS**
    d. **O score SUS é calculado pelo programa, e é exibido na interface o resultado, e o quanto a hipótese (marcada no slider) se aproximou do mesmo.**

Imports necessários:

```
import sys
from PySide2.QtWidgets import QApplication, QMainWindow, QLabel, QSlider, QPushButton, QFileDialog
from PySide2.QtCore import Qt
import csv
```

Criando a janela gráfica

```
aplicacao = QApplication(sys.argv)

janela = QMainWindow()
# setGeometry(esquerda, topo, largura, altura)
janela.setGeometry( 100, 50, 300, 250 )
janela.setWindowTitle("Calculo de usabilidade")

janela.show()

aplicacao.exec_()
sys.exit()
```

Criando o slider e os labels

```
label = QLabel("Informe a nota que você acredita que \no teste de usabilidade irá resultar: ", janela)
label.move(20,20)
label.resize(200,40)

label_slider_0 = QLabel("0", janela)
label_slider_0.move(20, 60)
label_slider_0.resize(20, 20)

label_slider_100 = QLabel("100", janela)
label_slider_100.move(260, 60)
label_slider_100.resize(20, 20)

slider = QSlider(janela)
slider.move(20, 80)
slider.resize(260, 20)
slider.setOrientation(Qt.Horizontal)
slider.setMinimum(0)
slider.setMaximum(100)
```

Criando o label para exibir o valor que estará marcado no slider:

```
text_hipotese = QLabel("Hipótese: ", janela)
text_hipotese.move(20, 110)
text_hipotese.resize(60, 20)

result_hipotese = QLabel("", janela)
result_hipotese.move(70, 110)
result_hipotese.resize(60, 20)
```

Capturando o evento de mudança de valor no slider e atribuindo ao label result_hipotese o valor no slider:

```
def onSliderChanged():
    result_hipotese.setText(str(slider.value()) + " pontos")

slider.valueChanged.connect(onSliderChanged)
```

Criando o botão “Importar CSV”:

```
botao = QPushButton("Importar CSV", janela)
botao.move(20, 140)
```

Definindo a função que será executada ao clicar no botão:

```
def onClickedBotao():
    fileDialog = QFileDialog()
    nome_arquivo, tipo_arquivo = fileDialog.getOpenFileName(
        filter="Comma Separated Values (*.csv)")

botao.clicked.connect(onClickedBotao)
```

Criando uma função para ler o arquivo CSV e retornar o cálculo de SUS:

```
def calcular_sus(nome_arquivo):
    arquivo_csv = open(nome_arquivo, 'r')
    conteudo_csv = csv.reader(arquivo_csv, delimiter=',')
    somatorio = 0
    num_entrevistados = 0
    for index_linha, linha in enumerate(conteudo_csv):
        if index_linha != 0:
            num_entrevistados += 1
            for index_coluna, coluna in enumerate(linha):
                if index_coluna != 0:
                    if coluna == "": coluna = "3"
                    if index_coluna % 2 == 0:
                        somatorio += 5-int(coluna)
                    else:
                        somatorio += int(coluna)-1
    res = (somatorio/num_entrevistados)*2.5
    return int(round(res, 0))
```

Criando componentes para exibir os resultados:

```
resultado = QLabel("", janela)
resultado.move(20, 180)
resultado.resize(120,20)

diferenca = QLabel("", janela)
diferenca.move(20, 210)
diferenca.resize(260,20)
```

Utilizando a função na ação do clique no botão e alterando os resultados exibidos:

```
def onClickedBotao():
    fileDialog = QFileDialog()
    nome_arquivo, tipo_arquivo = fileDialog.getOpenFileName(
        filter="Comma Separated Values (*.csv)")
    if nome_arquivo != "":
        res_sus = calcular_sus(nome_arquivo)
        resultado.setText("Resultado: " + str(res_sus) + " pontos")
        diferenca.setText(
            "Diferença entre o Resultado e a Hipótese: "+ str(
                round(res_sus - slider.value(),2) ) + " pontos")
```

Ajustando a função que detecta a mudança de valor do slider para limpar os campos resultado e direfença, já que o valor da hipótese foi alterado:

```
def onSliderChanged():
    result_hipotese.setText(str(slider.value()) + " pontos")
    resultado.setText("")
    diferenca.setText("")
```

GitHub: *exemplo_pratico_basico/calculo_usabilidade.py*

7. Referências

- Qt Documentation (Website)
  https://doc.qt.io/
- O que é GTK e Qt? (Linux) - DTL #3 (vídeo - YouTube)
  https://www.youtube.com/watch?v=O-VGFH3eMhY&ab_channel=Diolinux
- Curso de PyQt5 (playlist - YouTube)
  https://www.youtube.com/watch?v=zFM0xd5Zbcg&list=PLwsAoT89dh3qJ8JcprQ8AuHY8AGasvx4G&ab_channel=Eletr%C3%B4nicaePrograma%C3%A7%C3%A3o
- PySide2 GUI com python3 e QT5 (Website)
  https://medium.com/@AlexandreESouza/pyside2-gui-com-python3-e-qt5-2fe2467f8422