INF 1515

Interfaces Gráficas

PyQt5, com comentários sobre Flet e Tkinter

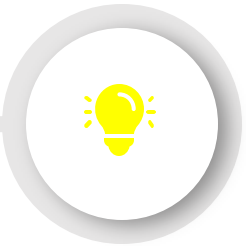
PUC
RIO

Ana Luiza Guimarães, Lucas Mendes

Interfaces gráficas de usuário (GUI) são essenciais para tornar aplicativos mais acessíveis e interativos, permitindo que o usuário interaja com o programa de forma intuitiva

Aplicativos

Conceito de classes: estruturas fundamentais na programação orientada a objetos

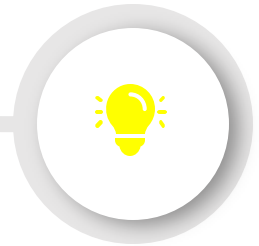


```python
class NomeDaClasse:
    def __init__(self, parametro1, parametro2):
        self.atributo1 = parametro1
        self.atributo2 = parametro2

    def metodo1(self):
        # Código do método
        pass

    def metodo2(self, parametro):
        # Código do método
        pass
```

- **Init:** método inicializador, construtor da classe, chamado automaticamente quando um novo objeto é criado
- **Self:** é uma referência à própria instância da classe, usada para acessar atributos e métodos dentro da classe.

Por que usar Classes para criar interfaces?

Aplicação PyQt5: são usadas para definir as janelas e os componentes da interface. Cada janela é criada como uma classe que herda de QWidget ou QMainWindow, isso permite um código mais organizado.



- **Flet:** usa classes para encapsular eventos e componentes, com uma lógica de organização similar ao PyQt5. Ele se destaca pela simplicidade e por facilitar a criação de aplicações responsivas.

- **Tkinter:** são usadas para gerenciar janelas e widgets, estruturando a interface de forma modular. Isso garante maior controle sobre os elementos.

Classe **QApplication**

- A classe **QApplication** é o ponto de partida de qualquer aplicação PyQt5. Ela é responsável por gerenciar os recursos da interface gráfica e supervisionar o ciclo de eventos do aplicativo.

```
app = QApplication(sys.argv)
```

- A linha acima cria uma instância de QApplication, passando **sys.argv** como parâmetro. **sys.argv** permite que o aplicativo receba argumentos da linha de comando, o que é útil em cenários avançados onde queremos personalizar a execução do programa, mas não é essencial para aplicações básicas.

Flet: a configuração inicial é feita através da classe Page. O equivalente ao QApplication seria configurar o evento principal chamando a função flet.app(target=main), onde main é a função que define a interface.

Tkinter: não há uma classe equivalente à QApplication. O ciclo de eventos é iniciado automaticamente com o método mainloop() da janela principal.s.

Eventos e Sinais

Eventos e sinais são conceitos fundamentais para permitir interatividade na interface. No PyQt5, cada ação do usuário gera um "sinal", como o clique de um botão ou a digitação de texto, e esse sinal pode ser conectado a uma função (slot) que define a resposta a essa ação.

Como Funciona o Sistema de Sinais e Slots

Sinais são eventos específicos, e slots são as funções que os recebem e processam. Quando você conecta um sinal a um slot, o PyQt5 executa o slot sempre que o sinal é emitido.

Por exemplo, o método **on_button_click** será chamado sempre que o botão for clicado, exibindo "Botão clicado!" no console.

Eventos e Sinais

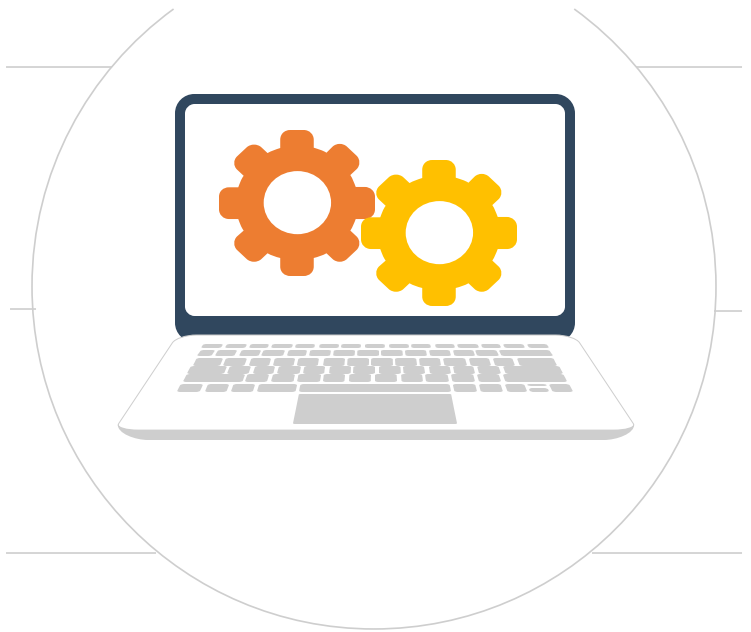
```python
import sys
from PyQt5.QtWidgets import QApplication, QWidget, QPushButton

# Função que será executada quando o botão for clicado
def on_button_click():
    print("Botão clicado!")

# Inicializa a aplicação
app = QApplication(sys.argv)

# Cria a janela principal
window = QWidget()
window.setWindowTitle("Minha Primeira Janela")  # Define o título da janela
window.setGeometry(100, 100, 400, 200)          # Define posição e tamanho (x, y, largura, altura)

# Cria o botão e conecta ao evento de clique
button = QPushButton("Clique aqui", window)  # Cria o botão e o associa à janela
button.setGeometry(150, 80, 100, 30)         # Define a posição e tamanho do botão
button.clicked.connect(on_button_click)      # Conecta o clique à função

# Exibe a janela
window.show()

# Inicia o loop de eventos
sys.exit(app.exec_())
```

Flet: Eventos são gerenciados por meio de propriedades específicas, como **on_click** ou **on_change**. Basta atribuir uma função ao evento, e ela será executada sempre que o evento for acionado.

Tkinter: Eventos são gerenciados com o método bind, que associa um evento, como **Button-1** para cliques, a uma função que será executada quando o evento ocorrer.

Componentes principais do PyQt5

1. **QLabel**
2. **QPushButton**
3. **QLineEdit**



4. **QCheckBox**
5. **QRadioButton**
6. **QComboBox**

Entre Outros...

Qlabel: exibir textos e imagens

- É ideal para rótulos, títulos, instruções ou ícones.

```python
import sys
from PyQt5.QtWidgets import QApplication, QLabel

# Inicializa a aplicação
app = QApplication(sys.argv)

# Cria o rótulo diretamente como o elemento principal
label = QLabel("Bem-vindo ao PyQt5!")
label.setGeometry(100, 100, 200, 40)  # Define posição (x, y) e (largura, altura)
label.show()

# Inicia o loop de eventos
sys.exit(app.exec_())
```

Flet: O equivalente seria o widget Text para exibir texto. Para imagens, você usaria o widget Image.

Tkinter: Use o widget Label, que também suporta texto e imagens.

QPushButton: Botão de ação

Representa um botão clicável. Pode disparar ações quando clicado, como abrir uma nova janela.

```python
import sys
from PyQt5.QtWidgets import QApplication, QPushButton

# Inicializa a aplicação
app = QApplication(sys.argv)

# Cria o botão diretamente como o elemento principal
button = QPushButton("Clique Aqui")
button.setGeometry(100, 100, 100, 30) # Define posição (x, y) e (largura, altura)
button.show()

# Inicia o loop de eventos
sys.exit(app.exec_())
```

Flet: O widget ElevatedButton ou TextButton é usado para botões clicáveis, e a ação é definida com o parâmetro on_click.
Tkinter: Use o widget Button e defina a função de callback com o parâmetro command.

QLineEdit: Campos de entrada de texto

Permite o usuário inserir textos. Ideal para formulários e tabelas.

```python
import sys
from PyQt5.QtWidgets import QApplication, QLineEdit

# Inicializa a aplicação
app = QApplication(sys.argv)

# Cria o campo de entrada diretamente como o elemento principal
input_field = QLineEdit()
input_field.setPlaceholderText("Digite seu nome")# Texto de exemplo no campo
input_field.setGeometry(100, 100, 200, 30)
input_field.show()

# Inicia o loop de eventos
sys.exit(app.exec_())
```

Flet: Use o widget TextField, que permite entrada de texto com suporte a placeholders por meio do parâmetro hint_text.
Tkinter: Use o widget Entry e configure o texto de exemplo adicionando um StringVar ou um Label adicional como guia.

QCheckBox: Caixas de seleção

Permite ao usuário marcar ou desmarcar uma opção. Ele é útil para configurações que podem ser ativadas ou desativadas.

```python
import sys
from PyQt5.QtWidgets import QApplication, QCheckBox

# Inicializa a aplicação
app = QApplication(sys.argv)

# Cria a checkbox diretamente como o elemento principal
checkbox = QCheckBox("Aceitar Termos e Condições")
checkbox.setGeometry(100, 100, 200, 30)
checkbox.show()

# Inicia o loop de eventos
sys.exit(app.exec_())
```

Flet: O widget equivalente é Checkbox, onde o estado marcado ou desmarcado é gerenciado por meio do parâmetro value e eventos.
Tkinter: Use o widget Checkbutton e associe um BooleanVar para controlar o estado da seleção.

QRadioButton: Seleção exclusiva

```python
import sys
from PyQt5.QtWidgets import QApplication, QRadioButton

# Inicializa a aplicação
app = QApplication(sys.argv)

# Cria o primeiro botão de rádio
radio_button1 = QRadioButton("Opção 1")
radio_button1.setGeometry(100, 100, 100, 30)
radio_button1.show()

# Cria o segundo botão de rádio
radio_button2 = QRadioButton("Opção 2")
radio_button2.setGeometry(100, 140, 100, 30)
radio_button2.show()

# Inicia o loop de eventos
sys.exit(app.exec_())
```

Permite que o usuário selecione uma opção entre várias alternativas, onde apenas um botão pode ser selecionado de cada vez.

Flet: Use o widget Radio dentro de um RadioGroup para criar seleções exclusivas.
Tkinter: Use o widget Radiobutton e associe todos os botões a uma mesma variável, como StringVar ou IntVar, para garantir a exclusividade.

QComboBox: Dropdown

```python
import sys
from PyQt5.QtWidgets import QApplication, QComboBox

# Inicializa a aplicação
app = QApplication(sys.argv)

# Cria o combo box diretamente como o elemento principal
combo_box = QComboBox()
combo_box.addItem("Opção 1")  # Adiciona a primeira opção
combo_box.addItem("Opção 2")  # Adiciona a segunda opção
combo_box.setGeometry(100, 100, 150, 30)
combo_box.show()

# Inicia o loop de eventos
sys.exit(app.exec_())
```

Cria um menu suspenso com várias opções, permitindo que o usuário selecione uma delas, útil quando se tem várias opções e você quer economizar espaço na interface.

Flet: O equivalente é o widget Dropdown, que permite criar menus suspensos configurando as opções com DropdownItem.
Tkinter:Use o widget OptionMenu ou Combobox (do módulo ttk) para criar menus suspensos.

Qslider: Controle Deslizante

Permite que o usuário selecione um valor dentro de um intervalo específico deslizando o controle entre os extremos.

```python
import sys
from PyQt5.QtWidgets import QApplication, QSlider
from PyQt5.QtCore import Qt

# Inicializa a aplicação
app = QApplication(sys.argv)

# Cria o slider diretamente como o elemento principal
slider = QSlider(Qt.Horizontal)  # Cria o slider na orientação horizontal
slider.setGeometry(100, 100, 150, 30)
slider.setMinimum(0)                    # Valor mínimo do slider
slider.setMaximum(100)                  # Valor máximo do slider
slider.setValue(50)                     # Define o valor inicial
slider.show()

# Inicia o loop de eventos
sys.exit(app.exec_())
```

Flet: Use o widget Slider, configurando os valores mínimo, máximo e inicial com min, max e value.
Tkinter:Use o widget Scale para criar sliders horizontais ou verticais.

QListWidget: Lista de itens

Permite exibir uma lista de itens para o usuário, onde ele pode selecionar uma ou mais opções

```python
import sys
from PyQt5.QtWidgets import QApplication, QListWidget

# Inicializa a aplicação
app = QApplication(sys.argv)

# Cria o list widget diretamente como o elemento principal
list_widget = QListWidget()
list_widget.setGeometry(100, 100, 200, 80)
list_widget.addItem("Item 1")              # Adiciona o primeiro item
list_widget.addItem("Item 2")              # Adiciona o segundo item
list_widget.addItem("Item 3")              # Adiciona o terceiro item
list_widget.show()

# Inicia o loop de eventos
sys.exit(app.exec_())
```

Flet: Use o widget ListView ou Dropdown (dependendo do contexto) para exibir listas de itens.
Tkinter: Use o widget Listbox para criar uma lista de itens com suporte a seleção.

QTableWidget: Tabela de dados

```python
import sys
from PyQt5.QtWidgets import QApplication, QTableWidget, QTableWidgetItem

# Inicializa a aplicação
app = QApplication(sys.argv)

# Cria o table widget diretamente como o elemento principal
table_widget = QTableWidget()
table_widget.setGeometry(100, 100, 300, 200)
table_widget.setRowCount(3)                    # Define o número de linhas
table_widget.setColumnCount(2)                 # Define o número de colunas

# Adiciona dados às células
table_widget.setItem(0, 0, QTableWidgetItem("Nome"))
table_widget.setItem(0, 1, QTableWidgetItem("Idade"))
table_widget.setItem(1, 0, QTableWidgetItem("Ana"))
table_widget.setItem(1, 1, QTableWidgetItem("25"))
table_widget.setItem(2, 0, QTableWidgetItem("João"))
table_widget.setItem(2, 1, QTableWidgetItem("30"))

table_widget.show()

# Inicia o loop de eventos
sys.exit(app.exec_())
```

Exibe dados em uma tabela com linhas e colunas personalizáveis.

Flet: Use o widget DataTable para exibir tabelas estruturadas com cabeçalhos e linhas de dados.
Tkinter: Use o widget Treeview (do módulo ttk) para criar tabelas e exibir dados tabulares

Mais exemplos de widgets PyQt5

QDateEdit

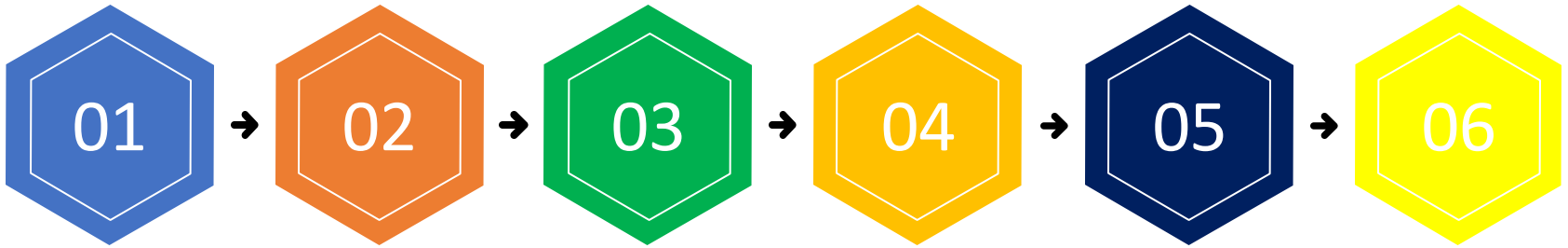
Editar datas

QProgressBar

Uma barra de progresso.

QSpinBox

Um seletor de números decimais



QDoubleSpinBox

Um seletor de números decimais.

QFontComboBox

Uma lista de fontes de texto interativa

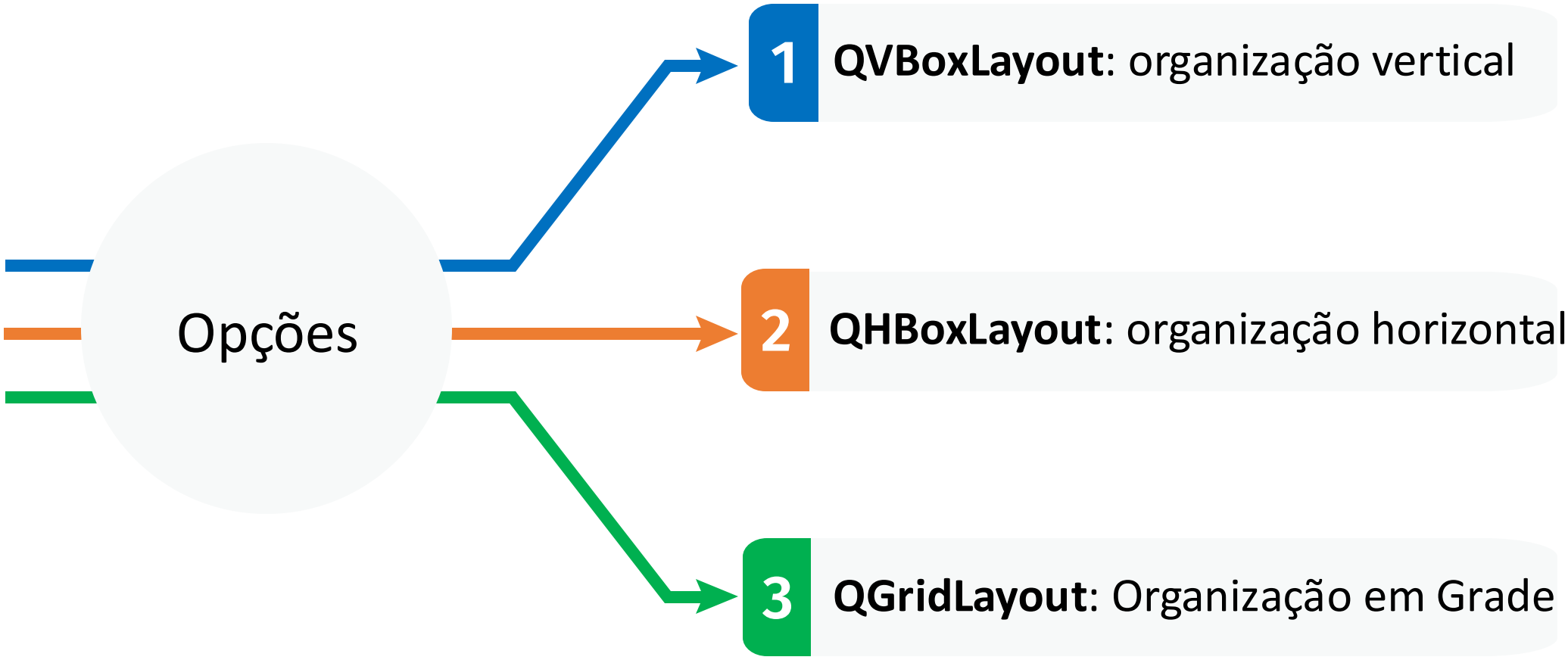
QLCDNumber

Exibe números em um display de sete segmentos, semelhante a um visor de relógios digitais.

Layouts e Organização de Componentes

Após criarmos uma janela e adicionarmos um (ou vários) widget a ela, para melhorar a organização usaremos ferramentas de layout. Assim, podemos redimensionar e mudar a posição dos widgets dentro da janela.

Organização



QVBoxLayout

```python
import sys
from PyQt5.QtWidgets import QApplication, QWidget, QPushButton, QVBoxLayout
# Inicializa a aplicação
app = QApplication(sys.argv)

# Cria a janela principal
window = QWidget()
window.setWindowTitle("Exemplo com QVBoxLayout")

# Cria os botões
button1 = QPushButton("Botão 1")
button2 = QPushButton("Botão 2")
button3 = QPushButton("Botão 3")

# Configura o layout vertical
layout = QVBoxLayout()
layout.addWidget(button1)
layout.addWidget(button2)
layout.addWidget(button3)

# Aplica o layout na janela
window.setLayout(layout)

# Exibe a janela
window.show()

# Inicia o loop de eventos
sys.exit(app.exec_())
```

Flet: O equivalente seria o widget Column, que organiza os widgets verticalmente. Elementos são adicionados à coluna usando o método controls.append(widget) dentro do Column.

Tkinter: Não há um gerenciador de layout específico para organização vertical, mas é possível usar o método pack() com o argumento side="top" para empilhar widgets verticalmente.

QHBoxLayout

```python
import sys
from PyQt5.QtWidgets import QApplication, QWidget, QCheckBox, QHBoxLayout
# Inicializa a aplicação
app = QApplication(sys.argv)

# Cria a janela principal
window = QWidget()
window.setWindowTitle("Exemplo com QHBoxLayout")

# Cria as caixas de seleção
checkbox1 = QCheckBox("Opção 1")
checkbox2 = QCheckBox("Opção 2")
checkbox3 = QCheckBox("Opção 3")

# Configura o layout horizontal
layout = QHBoxLayout()
layout.addWidget(checkbox1)
layout.addWidget(checkbox2)
layout.addWidget(checkbox3)

# Aplica o layout na janela
window.setLayout(layout)

# Exibe a janela
window.show()

# Inicia o loop de eventos
sys.exit(app.exec_())
```

Flet: Use o widget Row para organizar os componentes horizontalmente. Elementos são adicionados à linha com o método controls.append(widget) dentro do Row.
Tkinter: É possível usar o método pack() com o argumento side="left" ou o método grid() especificando a mesma linha (row) e diferentes colunas (column).

QGridLayout

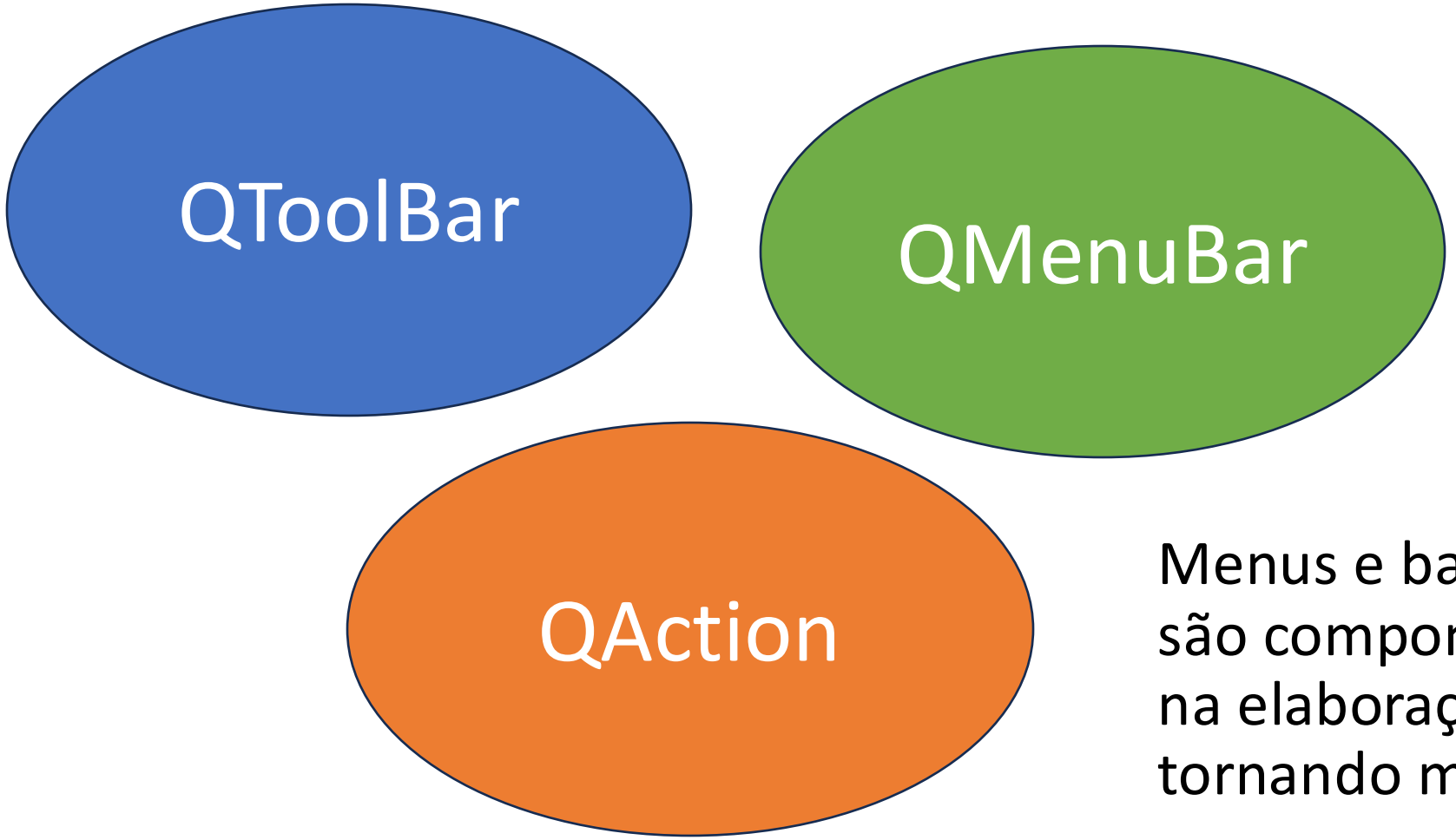
```python
import sys
from PyQt5.QtWidgets import QApplication, QWidget, QLabel, QLineEdit, QGridLayout

# Inicializa a aplicação
app = QApplication(sys.argv)

# Cria a janela principal
window = QWidget()
window.setWindowTitle("Exemplo com QGridLayout")

# Cria os widgets
label1 = QLabel("Nome:")
input1 = QLineEdit()
label2 = QLabel("Idade:")
input2 = QLineEdit()

# Configura o layout em grade
layout = QGridLayout()
layout.addWidget(label1, 0, 0)  # Adiciona na linha 0, coluna 0
layout.addWidget(input1, 0, 1)  # Adiciona na linha 0, coluna 1
layout.addWidget(label2, 1, 0)  # Adiciona na linha 1, coluna 0
layout.addWidget(input2, 1, 1)  # Adiciona na linha 1, coluna 1
```

```python
# Aplica o layout na janela
window.setLayout(layout)

# Exibe a janela
window.show()

# Inicia o loop de eventos
sys.exit(app.exec_())
```

Flet: O Flet não possui diretamente um layout de grade como o QGridLayout. Você pode usar múltiplos Row ou Column juntos para simular uma grade.

Tkinter: Use o método grid() para organizar widgets em linhas e colunas. Cada widget é posicionado com os argumentos row e column, como no PyQt5. Por exemplo, widget.grid(row=0, column=0).

Menus e Barras de Ferramentas



Menus e barras de ferramentas são componentes fundamentais na elaboração de interfaces, as tornando mais organizadas.

QToolBar: Barra de Ferramentas

É utilizada para criar barras que contêm botões, ícones e outros widgets que realizam tarefas específicas. Essas barras podem ser adicionadas a uma janela principal com o método addTollBar().

```python
import sys
from PyQt5.QtWidgets import QMainWindow, QApplication, QToolBar, QLabel,QCheckBox
from PyQt5.QtCore import Qt

class MainWindow(QMainWindow):
    def __init__(self):
        super().__init__()
        self.setWindowTitle("Toolbar com Checkbox")

        # Rótulo central
        self.label = QLabel("Checkbox não marcado")
        self.label.setAlignment(Qt.AlignCenter)
        self.setCentralWidget(self.label)

        # Criando a barra de ferramentas
        toolbar = QToolBar("Barra de Ferramentas")
        self.addToolBar(toolbar)

        # Adicionando um checkbox
        checkbox = QCheckBox("Marcar")
        checkbox.stateChanged.connect(self.alterar_texto)
        toolbar.addWidget(checkbox)
```

Flet: Barras de ferramenta sao frequentemente implementadas usando AppBar, que permite adicionar botões, ícones e menus na parte superior da janela.
Tkinter: Barras de ferramenta são criadas adicionando widgets como botões a um container, geralmente um Frame.

```python
    def alterar_texto(self, estado):
        if estado == Qt.Checked:
            self.label.setText("Checkbox marcado")
        else:
            self.label.setText("Checkbox não marcado")

app = QApplication(sys.argv)
window = MainWindow()
window.show()
sys.exit(app.exec_())
```

QMenuBar: Menu Principal

É utilizada para criar menus no topo da janela, com funções organizadas em itens hierárquicos.

```python
import sys
from PyQt5.QtWidgets import QMainWindow, QApplication, QMenuBar, QLabel

class MainWindow(QMainWindow):
    def __init__(self):
        super().__init__()
        self.setWindowTitle("Menu com Submenus")

        # Rótulo central
        self.label = QLabel("Texto Central")
        self.setCentralWidget(self.label)

        # Criando o menu
        menu_bar = self.menuBar()
        menu_arquivo = menu_bar.addMenu("Arquivo")

        # Adicionando submenu
        submenu_opcoes = menu_arquivo.addMenu("Opções")
        submenu_opcoes.addAction("Opção 1").triggered.connect(lambda: self.atualizar_label("Opção 1
        submenu_opcoes.addAction("Opção 2").triggered.connect(lambda: self.atualizar_label("Opção 2

        menu_arquivo.addAction("Sair").triggered.connect(self.close)
```

```python
    def atualizar_label(self, texto):
        self.label.setText(texto)

app = QApplication(sys.argv)
window = MainWindow()
window.show()
sys.exit(app.exec_())
```

Flet: Menus interativos e hierárquicos são implementados usando PopupMenuButton ou Dropdown. Submenus podem ser adicionados com múltiplos itens dentro de PopupMenuButton.
Tkinter: A classe Menu é oferecida para criar menus no topo da janela

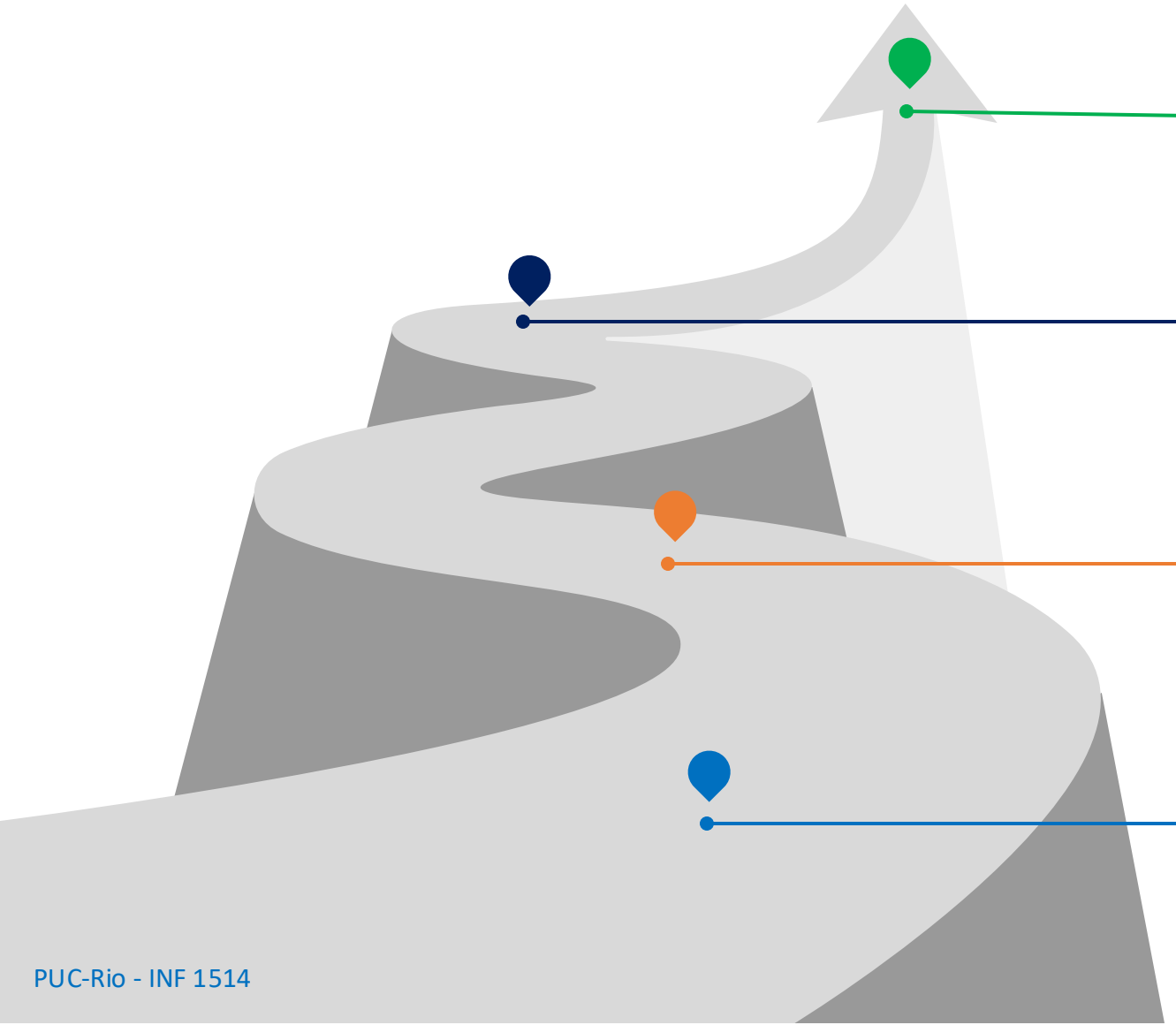
QAction: Interação em barras e menus

A classe permite definir ações reutilizáveis que podem ser compartilhadas entre menus e barras de ferramentas.

```python
import sys
from PyQt5.QtWidgets import QMainWindow, QApplication, QMenuBar, QLabel, QAction

class MainWindow(QMainWindow):
    def __init__(self):
        super().__init__()
        self.setWindowTitle("Exemplo com QAction")

        # Rótulo central
        self.label = QLabel("Texto Central")
        self.setCentralWidget(self.label)

        # Criando o menu
        menu_bar = self.menuBar()
        menu_arquivo = menu_bar.addMenu("Arquivo")

        # Criando ações
        acao_abrir = QAction("Abrir", self)
        acao_abrir.triggered.connect(lambda: self.atualizar_label("Abrir Selecionado"))
        menu_arquivo.addAction(acao_abrir)

        acao_salvar = QAction("Salvar", self)
        acao_salvar.triggered.connect(lambda: self.atualizar_label("Salvar Selecionado"))
        menu_arquivo.addAction(acao_salvar)
        acao_sair = QAction("Sair", self)
        acao_sair.triggered.connect(self.close)
        menu_arquivo.addAction(acao_sair)

    def atualizar_label(self, texto):
        self.label.setText(texto)

app = QApplication(sys.argv)
window = MainWindow()
window.show()
sys.exit(app.exec_())
```

Flet: Ações em flet são implementadas por meio de funções associadas aos eventos de cliques dos widgets, como on_click em PopupMenuButton ou IconButton.
Tkinter: As ações podem ser centralizadas definindo funções e vinculando-as a itens de menu ou botões na barra de ferramentas.

Exemplo: Lista de tarefas



04 Menu básico

O menu foi adicionado com a classe **QMenuBar**, que cria o menu **Arquivo**. Esse menu contém uma opção "sair" que está conectada ao método **close**, que encerra a aplicação..

03 Gerenciar tarefas

Marcar como concluídas e Remover tarefas

02 Adicionar tarefas

O usuário vai digitar a tarefa no campo de texto construído com **QLineEdit** e clicar no botão **Adicionar Tarefa**. O método irá criar um novo item na lista **QListWidget** e o exibe. Se o campo estiver vazio, uma mensagem de erro será exibida com **QMessageBox.warning**.

01 Estrutura da janela com QMainWindow

A classe **ToDoApp** vai herdar de QMainWindow, isso vai permitir que os menus sejam integrados com facilidade.

Uso de Multithreading em Interfaces Gráficas

Em interfaces gráficas, operações que consomem muito tempo (como leitura de arquivos, download ou cálculos pesados) podem travar a interface se executadas na thread principal. Para evitar isso, utilizamos threads secundárias para executar essas tarefas, garantindo que a interface continue responsiva.

PyQt fornece a classe **QThread** para facilitar o uso de threads secundárias em aplicações.

```python
from PyQt5.QtWidgets import QApplication, QLabel, QVBoxLayout, QPushButton, QWidget
from PyQt5.QtCore import QThread, pyqtSignal
import time

class WorkerThread(QThread):
    progress = pyqtSignal(int)  # Sinal para notificar o progresso

    def run(self):
        for i in range(1, 6):  # Simula uma tarefa longa
            time.sleep(1)
            self.progress.emit(i)  # Envia o progresso para a interface


class MainWindow(QWidget):
    def __init__(self):
        super().__init__()

        self.setWindowTitle("PyQt Multithreading")
        self.layout = QVBoxLayout()
        self.label = QLabel("Clique para iniciar uma tarefa longa.")
        self.button = QPushButton("Iniciar Tarefa")

        self.layout.addWidget(self.label)
        self.layout.addWidget(self.button)
        self.setLayout(self.layout)

        self.button.clicked.connect(self.start_task)
```

```python
    def start_task(self):
        self.thread = WorkerThread()
        self.thread.progress.connect(self.update_label)
        self.thread.start()

    def update_label(self, progress):
        self.label.setText(f"Progresso: {progress}/5")


app = QApplication([])
window = MainWindow()
window.show()
app.exec_()
```

Flet: O multithreading em Flet é implementado utilizando o pacote threading, garantindo que a interface gráfica permaneça responsiva.
Tkinter: Para manter a interface responsiva, o pacote threading também pode ser usado, executando tarefas longas em threads secundárias enquanto a interface principal continua operante.

Exercícios Interfaces Gráficas

1) Crie uma aplicação que converta temperaturas entre Celsius, Fahrenheit e Kelvin.

A janela deve possuir um espaço para a entrada da temperatura e opções para a escolha da unidade de entrada e saída.

a) Dicas: Implemente fórmulas para a conversão entre as escalas e use menus suspensos (dropdown) para selecionar as unidades.
b) Exigências: A entrada deve ser validada para receber apenas números reais e o resultado deve ser exibido com duas casas decimais.

2) Desenvolva uma aplicação que calcule a média aritmética de uma lista de números fornecida pelo usuário.

A janela deve conter um campo de entrada para os números, separados por vírgulas e um botão para calcular.

a) Dicas: Divida as entradas por vírgulas, converta os valores para números e utilize sum() e len() para calcular a média.
b) Exigências: A entrada deve ser validada para que todas os valores sejam números e caso não sejam fornecidos valores, exiba uma mensagem de erro.