PyQt5 — O fantástico mundo das GUIs

https://medium.com/@wilfilho/pyqt5-o-fantastico-mundo-das-guis-62914b1b39c1

Na realidade, eu escrevi esse post quando a versão 5.2 do PyQt foi lançado no qual continha lindas e relevantes modificações no mesmo. Porém, apenas estou postando agora. Atualmente o PyQt5 está versão 5.4, e possui algumas modificações interessantes.

Percebo que não há muitos tutoriais que abordam o PyQt, principalmente o PyQt5, sendo assim, resolvi criar esta série de tutoriais. Poderíamos falar, por que não utilizar a documentação? Para quem está iniciando eu creio que seja mais difícil encarar a documentação. Apesar dela possuir todas as informações sobre o que queremos, nós não sabemos quais dessas centenas de informações olhar primeiro, para assim termos um avanço gradual. Este conjunto de posts é justamente para dar um caminho bem iluminado para quem tem interesse em conhecer o PyQt.

Faço uma ressalva da documentação do PyGtk, que é muito boa.

Este é o primeiro artigo de uma série de diversos outros onde abordarei o funcionamento e execução da nova versão do Qt utilizando o Python. Inicialmente será um conjunto de artigos, como descrito anteriormente, que depois poderão se transformar em vídeo aulas, havendo uma chance um tanto remota da realização do mesmo. Espero que todos possuam uma ótima leitura e conhecimento, para tal, podermos discutirmos em um momento posterior.

Nota: *Este conjunto de artigos são baseados na documentação do Qt5.*

Para não nos perdemos ao longo das postagens, resolvi criar um roteiro para seguirmos e aprendermos de forma gradual sem muitas dificuldades. Seguiremos o seguinte roteiro:

> ***#1 — Introduzindo conceitos e estudos sobre o PyQt5***
>
> ***#2 — Conhecendo o coração e as artérias do PyQt5***
>
> ***#3 — Entendendo e utilizando os componentes visuais(widgets)***
>
> ***#4— Gerenciando diversos tipos de armazenamentos de dados***
>
> ***#5 — Realizando deploy de uma aplicação desktop***

Antes de abordamos de fato as características dos filhos do Qt, vamos entender o que é o Qt em si. Creio que alguns nunca tenham ouvido sequer falado do Qt, então vamos iniciar dos seus primórdios.

Conhecendo algumas características do Qt

O Qt é um framework multiplataforma, criado originalmente no C++ para desenvolver aplicações desktop. Inicialmente mantido e criado pela empresa TrollTech e agora mantido pelo Qt Project. É e foi utilizado por diversas empresas de grande renome no mundo tecnológico, tais como a Google, NASA, Disney, Samsumg, Philips, Volvo, etc. Apesar do Qt ser um framework inicialmente criado para desenvolver aplicações desktop, acabou sendo portado para outras plataformas, onde podemos citar o mundo mobile. Hoje ele suporta grande parte dos sistemas mobile, como o Android, IOS, Windows Mobile, BlackBerry, (o extinto) Symbian e o Maemo. Porém, não paramos por aí, o Qt ainda influenciou, influencia e serviu como base para a criação de ambientes desktop, podendo citar o conhecido KDE e o recém chegado LXQt.

Quando o Qt foi desenvolvido, por volta de 1991–1992, ainda não existiam parte das linguagens famosas que conhecemos hoje, como por exemplo o Ruby ou o Java. Com o surgimento dessas linguagens e o aprimoramento das mesmas, o Qt acabou sendo portado para elas, originando assim, os bindings do Qt. Hoje existe uma série de bindings para diversas linguagens. No Ruby o binding do Qt é chamado de QtRuby, no Java de QtJambi, no C# de Qyoto, na linguagem Lua é chamado de QtLua e assim por diante.

No Python existem dois bindings, sendo eles, o PyQt e o PySide. Como podemos observar, dentre as linguagens citadas anteriormente somente o Python possui dois bindings para o Qt.

O Qt é licenciado basicamente de três formas. Vejamos logo abaixo:

Licença comercial: A licença comercial do Qt não é obrigatória, mas sim recomendada para criar aplicações comerciais, onde desejamos vender aplicações com que utilizam o Qt, desejamos possuir um maior suporte e ferramentas prontas e não disponibilizar o código fonte para o usuário. Além disso, a licença comercial do Qt fornece um conjunto de funcionalidades e ferramentas que não estão presentes nas outras formas de licença adquiridas e aceitas pelo Qt. A empresa que mantém a versão comercial do Qt é a Digia, que é uma empresa Finlandesa de software. Esta licença também está atrelada ao Qt Enteprise.

Licença LGPL: Creio que todos conheçam este tipo de licença, fazendo com que o Qt possa ser utilizado dentro dos limites da licença LGPL. O Qt utiliza a versão 2.1 da licença LGPL e possui também uma exceção para o desenvolvimento de aplicações comerciais utilizando a mesma.

Licença GPL: O Qt também pode ser adquirido através da licença GPL v3, contanto que você respeite(novamente) os limites da mesma. Inclusive o PyQt obtém o Qt através da linceça GPL.

Por fim, o Qt possui duas linguagens próprias, sendo elas o QML e o QtScript. Onde o QML também é utilizado na criação de aplicativos para a versão mobile do Ubuntu e o QtScript foi baseado no Ecmascript. A principal ferramenta utilizada para a criação e desenvolvimento utilizando o Qt é o Qt Creator, porém possuem diversas outras opções específicas para determinadas linguagens.

Conhecendo o PyQt

Como já mencionado, o PyQt é um dos bindings do Python para o Qt. Foi criado e é mantido pela Riverbank Computing. A IDE utilizada no PyQt é o QtDesigner, que é substancialmente diferente do QtCreator, pois não possui suporte ao QML nem ao QtScript. Basicamente o PyQt possui um dual license, onde há a presença da licença GPL(na terceira versão)para a criação de aplicações open source(e que respeitem os termos dessa licença) e a licença comercial, destinada obviamente, para a criação e distribuição de aplicações comerciais utilizando o PyQt.

O PyQt utiliza-se de uma das características principais do Python. Para que você entenda de uma forma mais simplificada, o PyQt nada mais é do que a junção do útil ao agradável. No qual detém todo o poder e riqueza do Qt, utilizando toda a praticidade e produtividade que o Python oferece ao desenvolvedor. Caso deseje obter uma base de aplicações feitas com o PyQt, acesse o seguinte link. Em meio a esta lista, podemos destacar aplicações como a Eric IDE e a Spyder IDE. Outra aplicação de muito destaque no Python e que utiliza o PyQt é a Ninja IDE. Esses são poucos exemplos de uma série de outras aplicações que o utilizam.

Introdução ao PyQt5

O PyQt5 traz grandes mudanças em relação ao PyQt4(como já descrito). Uma grande e fundamental dessas mudanças é a presença de novos módulos e consequentemente novas classes, deixando biblioteca mais flexível a diversos tipos de situações. Antes de iniciarmos a parte prática, vamos deter um pouco de conhecimento sobre esses módulos. Vejamos a seguir:

QtCore: Como o seu próprio nome já nos descreve, este módulo é responsável por cuidar e oferecer todas as funcionalidades relacionadas ao core do PyQt. Sendo assim, temos como exemplo: gerenciamento de sinais, eventos, slots, arquivos, threads, expressões regulares, alocação de memória, temporizadores e algumas outras funcionalidades.

QtGui: No PyQt4 ele era responsável por armazenar os mais diversos tipos de widgets, desde aqueles de mais baixo nível até os de mais alto nível. Agora, no PyQt5, ele foi dividido em dois, sendo o outro módulo chamado de QtWidgets. O QtGui ficou com a função de ser um módulo de GUI base, fornecendo suporte a tratamento de imagens, textos, renderização de gráficos 2D e integração com o OpenGL.

QtWidgets: Um dos pontos interessante dessa nova versão do Qt, é a nomenclatura dos módulos que as deixa mais amigáveis aos desenvolvedores. O módulo QtWidgets ficou responsável por armazenar todos os widgets de alto nível presentes no PyQt. Exemplo: nele podemos encontrar os mais diversos tipos de layout(QGridLayout, QFormLayout, QHBoxLayout, etc), à até mesmo botões, áreas de texto, locais para inserir datas, janelas, etc.

QtSql: Este módulo trata diretamente com o acesso a dados, mais especificamente, o acesso ao banco de dados. É possível conectar, criar, remover e clonar conexões, além é claro, de realizar consultas no banco de dados através de 11 classes que suprem totalmente as nossas necessidades. Dentro dessas 11 classes, ainda estão incluídas 2 classes(QSqlTableModel e QSqlQueryModel) que implementam o paradigma Model/View, onde os dados fornecidos por essas classes podem ser utilizados diretamente em componentes visuais, como QListView e QTableView.

QtTest: Hoje em dia, a criação de testes de unidade nos nossos projetos é indispensável, uma vez que torna o nosso projeto mais limpo e parcialmente livre de bugs posteriores que venhamos a ter. Este módulo trata exatamente disso, possuindo uma diferença em relação ao seu pai(o Qt), onde este não implementa a realização dos testes igualmente à forma do Qt, em vez disso, utiliza-se dos componentes padrões do Python utilizados para realizar testes de unidade e simula ações realizadas pelos usuários. Nela podemos testar diretamente diversos tipos de eventos, como por exemplo: podemos testar se um botão está realmente sendo ativado por determinado sinal, através de métodos disponibilizados pela classe QTest, tais como *mouseClick()*, *mousePress() ou mouseReleased()*. Sendo os métodos anteriores relacionados diretamente aos sinais suportados por um botão no PyQt.

QtNetwork: Apesar de ser um módulo extra, digamos assim, é indispensável dentro do PyQt devido ao fato de que uma quantidade muito grande de aplicações utilizam-se da realização de conexões para se conectar com o que quer que seja. Neste módulo é possível construir clientes e servidores UDP, TCP, HTTP, SSL e FTP. No PyQt4 ele somente suporta IPv4, porém nesta nova versão, o IPv6 também é suportado. Outra funcionalidade interessante é o suporte a conexões baseadas no protocolo TCP e a utilização de certificados SSL.

QtWebKit: Seguindo a mesma linha do QtGui, o QtWebKit na nova versão do Qt foi divido em dois, onde uma parte(QtWebKit) ficou responsável por tratar o core deste módulo e o ***QtWebKitWidgets***, ficou responsável disponibilizar a parte visual da mesma. Ou seja, dentro do QtWebKit estão classes como: QWebHistory(responsável por tratar o histórico de navegação), QWebSettings, QWebElement, etc. Enquanto dentro do módulo QtWebKitWidgets ficam classes como: QWebView(utilizada para exibir páginas web), QWebInspector, QWebFrame, etc.

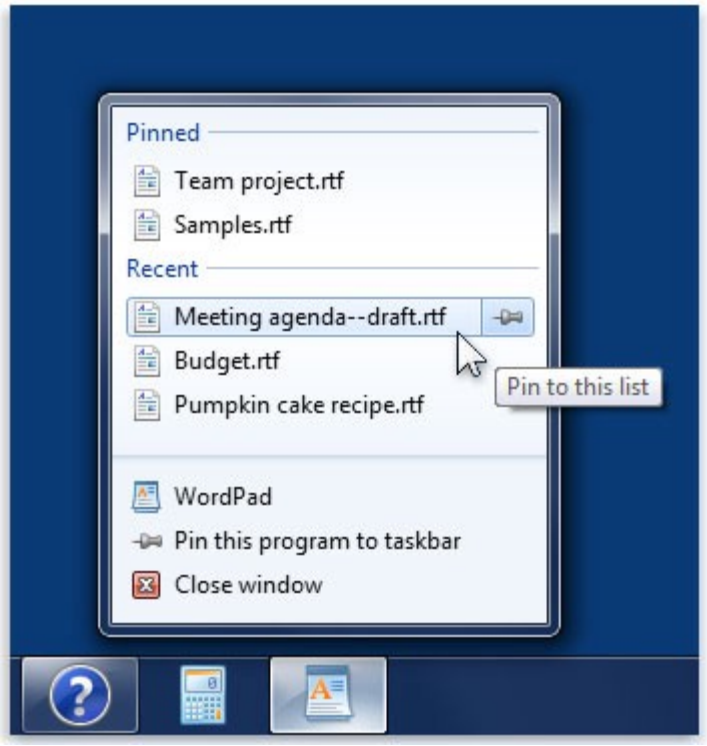
QtSvg: Módulo simples que permite realizar a leitura e renderização de arquivos svg. Este módulo ainda implementa uma classe chamada QSvgWidget, que permite de forma prática a visualização do conteúdo adquirido com a renderização de um arquivo sgv.

QtXml: Os arquivos XML são utilizados nos mais diversos tipos de aplicações e plataformas, seja ela mobile, desktop ou na web. É claro que o PyQt não poderia deixar de ter módulos exclusivamente para tratar deste tipo de arquivo. Este módulo contém classes que implementam as interfaces SAX e DOM do XML. Um outro módulo que incrementa a utilização do XML no PyQt é o ***QtXmlPatterns***, que implementa o suporte para o XQuery e o XPath e a customização de data models no XML.

QtMultimedia: Nesta nova versão do Qt, este módulo foi bem ampliado, uma vez que em versões anteriores oferecia somente suporte a manipulação e utilização de áudios e vídeos, agora possui suporte a câmera e rádio. Em relação às versões passadas, o QtMultimedia foi divido em dois nesta nova versão, onde a outra parte é chamada de ***QtMultimediaWidgets***, que fornece widgets para controle das funcionalidades presentes no ***QtMultimedia***.

QtPrintSupport: Este módulo contém classes que anteriormente estavam presente no módulo QtGui. Ele é responsável por fornecer diálogos e classes para realizar a impressão de documentos.

QtWinExtras, QtMacExtras e QtX11Extras: Outra novidade bastante interessante é a integração total com a parte gráfica do sistema operacional que o usuário está utilizando. O QtWinExtras, permite que nós por exemplo, criemos JumpLists no Windows, coisa que não era possível em versões anteriores. O QtMacExtras e o QtX11Extras seguem a mesma linha e só poderão ser importados no sistema operacional que corresponde a cada um deles.



Exemplo de uma JumpList no Windows 7.

QtScript: Permite criar scripts automatizados para realizar ações desejadas utilizando o EMACScript com um interpretador do Qt. Pode ser basntante útil em algumas ocasiões. Outro módulo que está na mesma linha que este é o ***QtScriptTools,*** provendo de uma classse adicional.

QtPortSerial: Este módulo permite com que nós tratemos comunicações com a porta serial, não tendo que recorrer a outras bibliotecas para desempenhar tal função. Apesar de ser um pouco limitado, atende bem a grande parte dos requisitos, sendo um bom avanço para o PyQt. Conta com duas classes: QPortSerial e QPortSerialInfo.

QtQuick: Provém de todos os tipos básicos para criar interfaces gráficas utilizando o QML, ou seja, possibilita criar interfaces gráficas, animações, data models, etc. Vale ressaltar que este módulo contém todas essas funcionalidades apenas dentro do QML. As classes que estão dentro do módulo QtQuick no PyQt, apenas interpretam os códigos criados utilizando o QML.

QtQml: Este módulo é a ponte de ligação entre o PyQt e o QML. Permite tratar, interpretar e implementar funcionalidades do QML no PyQt, como por exemplo: permite implementar a engine do QML para poder interpretá-lo.

QtSensors: Responsável pela manipulação de sensores, este módulo é apenas suportado na versão mobile do Qt, onde podemos citar como exemplo algumas plataformas, tais como: Android, BlackBerry 10, iOS, SailFishand e WinRT.

QtWebSockets: Como já podíamos esperar, o Qt5 não deixaria de fora uma tecnologia tão recentemente quanto a WebSockets.

QtDesigner: Módulo que permite criar plugins personalizados para o QtDesigner. Este módulo é essencial, pois você pode adaptar a própria IDE às suas necessidades.

Resumindo o PyQt5

Pudemos ver uma grande quantidade de módulos anteriormente, e de fato, estes não são todos os módulos do PyQt5, por exemplo: possuímos ainda o módulo Enginio e o QtPositioning, que não foram abordados. Acima, estão apenas os módulos mais utilizados e notórios.

O PyQt5 traz um grande conjunto de modificações em relação ao PyQt4. Iniciando desde a organização dos módulos até novos conceitos que agora possuímos. No PyQt4 tínhamos pouco mais de 20 módulos, no PyQt5 temos quase 30 módulos que deixam a biblioteca muito abrangente e flexível a diversos cenários.

Na organização dos módulos percebemos que o QtGui foi dividido em três, onde possuímos o QtWidgets, QtPrintSupport e o próprio QtGui. A mesma coisa aconteceu com os módulos QtMultimedia e QtWebKit. Aliás, o QtWebKit, que na versão 4 do Qt utilizava o WebKit 1, agora passa a utilizar o WebKit 2, dando suporte às tecnologias web mais recente, como o HTML5 e CSS3, além de dar uma maior performance na renderização das páginas. Com essa nova organização, o nome dos módulos se tornam mais semânticos e amigáveis para todos os níveis de desenvolvedor, onde cada um foi nomeado de acordo com a sua função.

Além da organização dos módulos, houve um acréscimo na quantidade dos mesmos, onde percebemos o surgimento do QtQml, QtQuick, QtPortSerial, QtSensors, QtPositioning, QtQuick, etc. Na versão anterior, as classes responsáveis por criar um elo entre o PyQt e o QML ficavam em um módulo chamado QtDeclarative, porém agora, este módulo foi excluído e possuímos o QtQuick e o QtQml, que nos fornece uma melhor tratamento destes tipos de interfaces gráfica. O QtPortSerial nos possibilita ter uma interação com a comunicação serial, sem necessitarmos utilizar uma outra biblioteca, além de nos poupar um bom tempo.

O surgimento do *QtWinExtras, QtMacExtras e QtX11Extras*, também são de extrema importância, uma vez que possibilitam uma maior portabilidade e integração com o ambiente gráfico no qual o usuário está utilizando. É crível que as demais bibliotecas de interfaces gráficas presentes no Python não possibilite este tipo de funcionalidade, adicionando um ponto de vantagem para o PyQt.

A ampliação dos módulos também é uma característica marcante em relação a versão anterior, uma vez que módulos como o QtNetwork e o QtCore recebem um upgrade para suportar as tecnologias mais atuais. No QtNetwork temos o suporte ao IPv6, utilização de certificados SSL, dentre outros. No módulo QtCore podemos perceber a presença de classes que permitem tratar arquivos JSON, também, fazendo com que não necessitemos recorrer à bibliotecas de terceiros. Podemos ver ainda no QtCore, a presença de classes que possibilitam tratar de uma forma ainda melhor as threads.

Não podemos deixar de destacar alguns projetos que surgem paralelos ao PyQt5. O *pyqtdeploy*, como o próprio nome já nos diz, permite realizar o deploy de aplicações criadas com o PyQt4 e PyQt5, não necessitando recorrer à bibliotecas como o py2exe ou cx_Freeze. O *dip*, também é outro projeto de extrema importância para projetos que prezam pelas boas práticas, possuindo diversas abstrações que agilizam o nosso desenvolvimento quando utilizados tais quesitos(sempre :P).

Por fim, podemos destacar uma forte presença e incentivo do Qt em utilizarmos o QML na GUI da nossa aplicação. O QML já estava presente no Qt desde versões passadas, porém agora, pelo que podemos ver, o mesmo recebeu um conjunto de upgrades que permite ao desenvolvedor possuir não só uma renderização mais rápida, mas também um riqueza visual muito grande e um melhor desempenho.

Instalando o PyQt5 no Windows

Atualmente os binários do PyQt5 para o Windows somente estão disponíveis para a versão 3.4 do Python, sendo assim, caso não tenha esta versão do Python, trate de baixá-la e instalá-la.

Para realizar a instalação é muito simples. Antes de mais nada, vamos fazer o download do PyQt5. Para tal, deveremos acessar o seguinte link: http://www.riverbankcomputing.com/software/pyqt/download5.

Você deverá selecionar a versão mais condizente com o seu sistema operacional. Caso você realize o download da versão errada, terá problemas ao importar o PyQt.

Após realizarmos o download, deveremos executar o arquivo e clicar em Next(forevermente). Feito isso, o PyQt5 já estará instalado no seu computador. Para termos certeza se ele foi instalado com sucesso, deveremos abrir o interpretador do Python e executar a seguinte linha:

```
>>> from PyQt5 import QtWidgets
```

Poderá ocorrer um erro na importação se você não baixou a versão correta(como já descrito). Para resolver isto, basta baixar a outra versão.

Instalando o PyQt5 no Mac OS X

Para facilitar a instalação do PyQt5 no Mac OS X, utilizaremos o gerenciador de pacotes *Homebrew*. Caso não o tenha instalado(but, what?), execute a seguinte linha de comando no seu terminal:

```
$ ruby -e "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/master/install)"
```

Após isto, teremos o *Homebrew* instalado. Agora deveremos executar mais uma linha de comando.

```
$ brew install pyqt5
```

Para verificar se o PyQt5 foi realmente instalado no seu Mac, vamos realizar os mesmos procedimentos da instalação no Windows, porém com uma diferença. Deveremos fechar o terminal e abri-lo novamente. Após termos aberto o terminal, vamos abrir o terminal do Python e executar a seguinte linha:

```
>>> from PyQt5 import QtWidgets
```

Caso possua algum erro, é recomendável que instale o PyQt5 manualmente. Para tal, você precisa baixar o source do PyQt5 e compilá-lo(se vira rapá!).

***Dica**: Baixe e compile o Qt5 antes.*

Instalando o PyQt5 no Ubuntu

Pelo fato do PyQt5 não possuir ainda nenhum pacote nos repositórios do Ubuntu, diferentemente do Arch Linux que já possui, temos que fazer tudo na mão. Ou seja, vamos ter que baixar o Qt5, baixar o SIP, o PyQt5 colocar tudo na panela e mexer até ficar pronto.

A primeira coisa que devemos fazer é instalar o Qt5, pois necessitamos do ***qmake*** para poder compilá-lo. Sendo assim, devemos acessar: http://www.qt.io/download-open-source/. Você deverá baixar a versão mais condizente com o seu sistema operacional.

***Dica**: você pode utilizar o* **wget** *para baixar.*

Após baixarmos o Qt5, devemos instalá-lo obviamente. É interessante darmos permissão ao arquivo, sendo assim, execute a seguinte linha:

```
$ chmod +x qt-opensource-linux-x64-5.4.0.run
```

***Atenção**: se o nome do seu arquivo não for este, você deverá modificar a linha acima indicando-o, caso contrário teremos um erro de* "no surch file".

Depois bastamos executar o arquivo, seguir as pages wizards até o final. Antes de baixarmos ou insalarmos o PyQt5, devemos instalar dois pacotes. O *build-essential* e o *python-dev*. O último é importante pelo fato do SIP utilizar durante a sua compilação. Devemos executar as seguintes linhas:

```
$ sudo apt-get install build-essential$ sudo apt-get install python3.4-dev
```

Agora que temos estes dois pacotes essenciais instalados, iremos baixar primeiramente o SIP. O SIP na verdade, de uma forma bem resumida, é o senhor que faz toda a mágica do Qt ser possível no PyQt. No PySide temos o Shiboken. Devemos acessar o seguinte link: http://www.riverbankcomputing.com/software/sip/download. Após o download, devemos entrar na pasta e executar o seguinte comando:

```
$ sudo python3.4 configure.py
```

Isto irá preparar os arquivos para nós realizarmos a compilação. Após isso, devemos executar o comando: ***sudo make***. E após este último comando: ***sudo make install***. Feito isso, temos o SIP instalado. Agora, vamos realizar o download do PyQt5. Bastamos acessar o seguinte link: http://www.riverbankcomputing.com/software/pyqt/download5 e realizar o download. Depois de descompactado, deveremos entrar na pasta e executar o mesmo comando que executamos inicialmente no SIP, ou seja:

```
$ sudo python3.4 configure.py
```

Caso possua o seguinte erro: "*Error: Use the —qmake argument to explicitly specify a working Qt qmake.".* Isso indica que nós devemos especificar o local do qmake. Este local por acaso está dentro da pasta onde você instalou o Qt5. A estrutura de diretórios até o qmake é a seguinte: qt5_path/5.4/gcc_64/bin/qmake. Caso o não esteja neste diretório, você deverá procurar sempre a pasta bin dentro do local onde foi instalado o seu Qt. Devemos assim, executar o seguinte comando(após termos encontrado o qmake):

```
$ sudo python3.4 configure.py --qmake=/home/will/Qt5.4.0/5.4/gcc_64/bin/qmake
```

Após isto, deveremos executar o comando: ***sudo make*** e por fim: ***sudo make install***. Feito isso, teremos o PyQt5 instalado e pronto para uso.

Extra: *Caso você tenha um erro relacionado a instalação da tool* ***pyuic5*** *você deverá fazer o seguinte, executar o configure.py com a opção de* ***no-tools****. Assim:* ***python3.4 configure.py —no-tools****.*

Nascendo no fantástico mundo das GUIs

Bem, após termos tido um conhecimento inicial sobre o PyQt5 e também possuí-lo instalado na nossa máquina, chegou a hora de pormos a mão na massa e partir para algo prático(finalmente, uhu!).

Sabemos que, antes de mais nada, quando desejamos utilizar uma biblioteca ou uma funcionalidade da mesma, devemos importar o que desejamos. No PyQt5, não é diferente. Sendo assim, inicialmente deveremos realizar as seguintes importações:

```
from PyQt5.QtWidgets import *
import sys
```

Primeiramente importamos tudo que estiver dentro do módulo QtWidgets. Realizar esta importação é essencial pelo fato de quê, dentro dele possui as classes QApplication e QWidget(classes que utilizaremos nesse exemplo inicial).

Importamos o sys para não fugir de um padrão nos códigos que utilizam o PyQt. Ele pegará os argumentos da linha de comando e jogará como parâmetro na instanciação da classe QApplication. Entenderemos melhor isto um pouco mais a frente.

Após realizarmos as importações que desejamos, devemos criar uma classe que conterá as configurações e os widgets presentes na nossa janela. Mas eis que surge uma questão. É possível utilizar o PyQt sem OO? Sim, é possível, porém não é recomendável. A cada instante que o seu código for ficando maior, o mesmo se tornará mais difícil de dar manutenção. Sendo assim, se ainda não sabe OO, sugiro que primeiramente tenha um bom conhecimento para seguir neste tutorial. O tipo de janela que utilizaremos para este exemplo inicial será o QWidget. Deixaremos dessa forma:

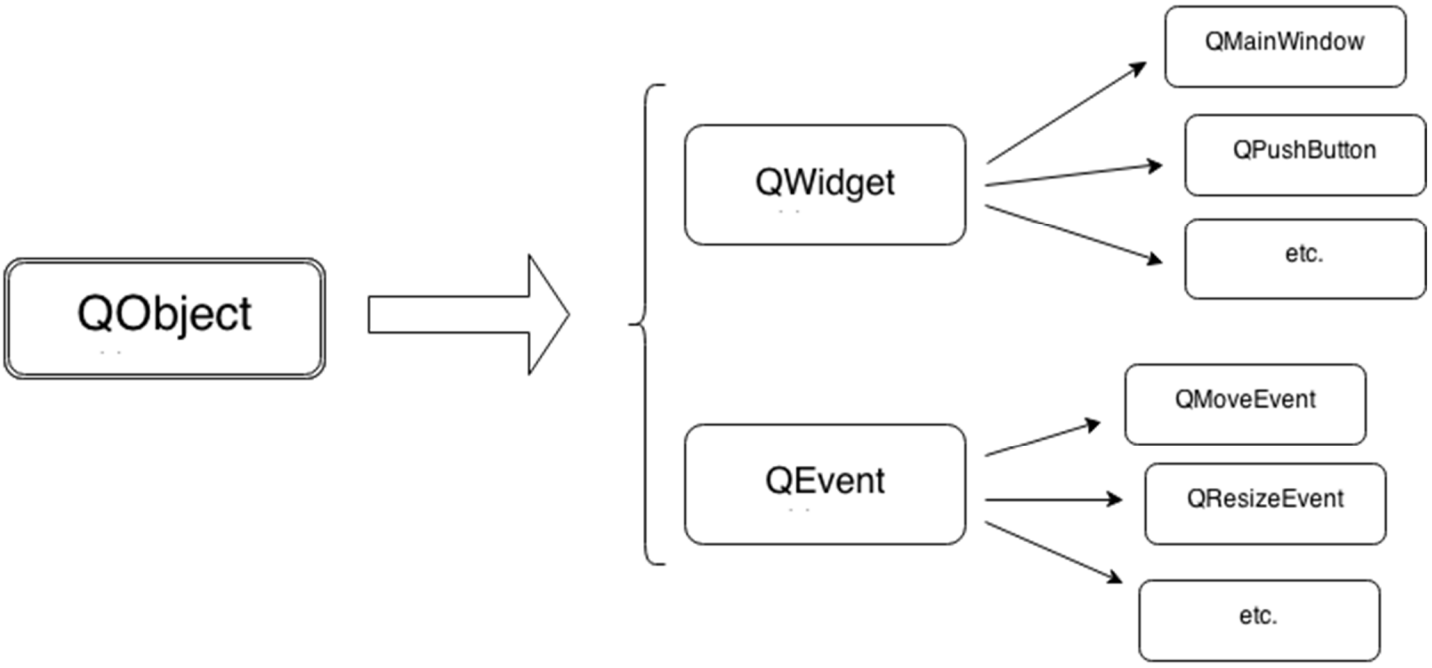
```
class Window(QWidget):
```

No trecho de código acima, podemos ver claramente a presença de uma herança simples, onde estamos herdando da classe QWidget todos os seus objetos e métodos. Além do QWidget, existem mais duas classes que são frequentemente utilizadas como janela top-level das nossas aplicações, sendo elas: QMainWindow e QDialog. A classe QMainWindow, como o próprio nome nos sugere, é uma classe que nos possibilita ter uma janela principal de onde podemos chamar modals dialogs(diálogos filhos, de grosso modo). Nela também há a presença de métodos que facilitam a criação e introdução de alguns widgets, por exemplo: existe métodos para criar barra de status, DockWidgets, toolbars, etc., onde declaramos o método e ele nos retorna uma instância pré-configurada do que queremos. Já a classe QDialog, possui um formato de janela diferente e métodos diferentes, afinal, é uma classe para se criar diálogos. Podemos perceber nela, a presença de apenas um botão para fechar, além dele responder por padrão a algumas teclas, como por exemplo, à tecla Esc. A classe QWidget é a classe mãe de todos os widgets. Isso significa dizer que, botões(QPushButton), áreas de texto(QTextEdit, QLineEdit, QPlainText, etc.), a classe QMainWindow, a classe QDialog e todas as demais que fazem parte do âmbito visual das nossas aplicações derivam dela.

Pais e filhos

No PyQt, os widgets podem ser definidos de duas formas: aqueles que não possuem pai(conhecidos como widgets pai) ou aqueles que necessitam de um pai para serem exibidos(conhecido como widgets filho). Aqueles que não possuem pai nada mais são do que janelas top-level, como as classes citadas logo acima: QMainWindow, QDialog e QWidget. Já o segundo tipo de widgets, necessitam de um pai para serem exibidos, ou seja, necessitam de classes como QMainWindow, QWidget ou QDialog para serem mostrados. Exemplos: botões(QPushButton), áreas de texto(QLineEdit, QPlainText, QTextEdit, etc.), barras de rolagem(QScrollbar), abas(QTabWidget), barras de status(QStatusBar), menus(QMenu), etc. Vale fazer uma observação de que a classe QWidget pode assumir os dois papéis, ou seja, pode ser utilizada com um pai ou sem um pai. O PyQt utiliza desse artifício de widgets pai e widgets filhos para ter certeza de que está excluindo tudo que deveria. Por exemplo, quando uma janela utilizando a classe QMainWindow é fechada, ele verifica se a classe QMainWindow é pai ou filho, caso ele seja pai, o mesmo excluirá a QMainWindow e todos os seus filhos, ou seja, todos os widgets que estão dentro dela.

***OBS**: O termo "pai" nesta ocasião está relacionado a palavra "frame", ou seja, local onde deve ser colocado algo ou deve ser armazenado algo.*



Exemplo de onde são derivadas classes importantes no PyQt.

Todas as classes, e consequentemente widgets, presentes no PyQt são derivados da classe QObject, seja diretamente ou indiretamente. Dessa forma, por exemplo, a classe QWidget é derivada da QObject, que a classe QMainWindow é derivada da classe QWidget. Para um melhor entendimento devemos observar a imagem acima, onde podemos ver claramente essas heranças. Na mesma imagem podemos ver ainda que a a classe QEvent também é derivada da classe QObject e que possui classes que derivam dela, como por exemplo a QMoveEvent. É claro, existem mais classes que não são derivadas nem da classe QEvent, nem da classe QWidget, que não estão neste exemplo, como por exemplo a classe QAbstractAnimation, que possui 6 classes que são derivadas dela(QVariantAnimation, QPropertyAnimation, QPauseAnimation, QAnimationGroup, QSequentialAnimationGroup e QParallelAnimationGroup).

Após declararmos a nossa classe, deveremos declarar o inicializador da mesma. O mesmo poderá receber um parâmetro adicional, este chamado de *parent* e deverá ser atribuído um valor nulo. Vejamos a seguir:

```
def __init__(self, parent=None):
```

O parâmetro *parent* é utilizado para indicar que nesta janela existe um frame pai, ou seja, um local onde todos os widgets estarão. Neste caso, temos uma janela que agrupará grande parte dos widgets. Ele pode ser simplesmente omitido caso desejares, porém eu sugiro que não, pois é algo padrão no PyQt.

Agora devemos inicializar a classe da qual estamos derivando, ou seja, a classe QWidget. Utilizaremos o **super()** ao invés das formas convencionais. Ficará da seguinte forma:

```
super(Window, self).__init__(parent)
```

Estamos chegando às partes finais do nosso script. Deveremos sair do bloco da nossa classe e criar uma instância da classe QApplication, da seguinte forma:

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

Nesta linha de código estão envolvidas muitas informações, porém vamos por partes. Sempre quando criamos uma instância da classe QApplication, devemos passar como parâmetro uma lista. Por padrão passamos como parâmetro uma lista de informações obtidas na linha de comando, ou seja, passamos como parâmetro o *sys.argv*. Poderíamos passar uma lista vazia se desejássemos, por exemplo:

```
root = QApplication([])
```

Ainda sobre o trecho de código acima, é de extrema importância criar uma instância da classe QApplication, pelo seguintes motivos:

1 — A classe QApplication é responsável habilitar o tratamento de eventos na nossa aplicação. Sem tratamento de eventos, ou seja, sem reconhecimento de ações, sem reações.

2 — É responsável por adquirir características gráficas do sistema operacional, tais como cores e fontes. Isso é importante pois deixa a aplicação nativa e padronizada.

3 — Caso não criemos, a nossa janela não será apresentada pelo simples fato de quê não conseguiremos criar o loop para exibir a nossa aplicação, ela também não será alocada na memória, além dos dois itens citados anteriormente.

Não olhe apenas em uma única direção

Vimos logo acima que se não criarmos uma instância da classe QApplication a nossa interface não poderá ser mostrada, porém isso não é tudo. Nós podemos dispensar a utilização da classe QApplication, pela utilização da classe QCoreApplication ou QGuiApplication dependendo da ocasião. Para definirmos a utilização de cada uma dessas classes é simples, vejamos: Quando a nossa aplicação for algo normal, ou seja, possuir interface gráfica(maioria dos casos) e possuir classes baseadas na classe QWidget, então nós utilizamos a classe QApplication; Quando a nossa aplicação não possuir interface e apenas for utilizada na linha de comando, utilizamos a classe QCoreApplication; Quando a nossa aplicação não utilizar classes que se baseiam na classe QWidget e possuir interface, então

utilizamos a QGuiApplication(poucos casos). Talvez agora, quando você for criar algo utilizando o PyQt ficará se perguntando sobre qual utilizar, QGuiApplication ou QApplication. Bem, a dica que posso dar é a seguinte: sabemos que todos os widgets presentes no PyQt são derivados da classe QWidget, então sabemos que vamos utilizar a classe QApplication. Outro ponto interessante de se analisar neste tipo de ocasião é se a minha aplicação possui uma janela ou não, se possuir então você utiliza a classe QApplication. A classe QGuiApplication é e deve ser somente utilizada em casos específicos. Por via das dúvidas, utilize QApplication.

Extra: *Um local onde o podemos ver bastante a presença do QGuiApplication é na integração do PyQt com o QML, pois nesta ocasião não há a presença de classes baseadas na QWidget.*

Temos também que criar uma instância da nossa janela principal, ou seja, da classe Window e imprimi-la utilizando o método *show()*. Podemos ter diversas janelas dentro de um único arquivo, sendo assim é importante definir qual será a principal e quais serão as janelas filhas. Ficará da seguinte forma:

```
app = Window()
app.show()
```

O método *show()* está presente em grande parte dos widgets e serve para exibir algo na tela.

Por fim, temos que criar um loop infinito, onde a nossa interface será mostrada infinitamente(:P) até que o usuário a feche. Para isso, deveremos escrever a seguinte linha de código:

```
sys.exit(root.exec_())
```

O *sys.exit()* garantirá que a nossa janela será fechada corretamente, além de que os pixels serão “limpos” quando a mesma for fechada. O código final, ficará da seguinte maneira:

```
from PyQt5.QtWidgets import *
import sys

class Window(QWidget):
    def __init__(self, parent=None):
        super(Window, self).__init__()

root = QApplication(sys.argv)
app = Window()
app.show()
sys.exit(root.exec_())
```

Importando da forma correta, definindo título e algo mais

Agora que detemos conhecimento sobre como criar uma janela utilizando o PyQt e quais são os trechos de códigos básicos, iremos brincar um pouco com a nossa janela que foi criada anteriormente.

A forma com a qual importamos o que queríamos é a menos apropriada possível, pois de acordo com as convenções de código do Python, nós estamos diminuindo a legibilidade do nosso código, uma vez que não estamos especificando o que usamos. Sabemos que as classes que utilizamos para criar o nosso hello world foram: QWidget e QApplication. Sendo assim, iremos modificar a primeira linha do nosso script para que a mesma fique da seguinte forma:

```
from PyQt5.QtWidgets import QApplication, QWidget
```

Dica*: No início do post foram apresentados diversos módulos descrevendo cada um deles, justamente para sabermos qual devemos utilizar e quais classes utilizar. Sendo assim, quando não souber qual módulo utilizar basta ir até o mesmo para acender uma luz na sua mente. E jamais importe tudo, apenas o necessário.*

Modificar o título da janela é uma tarefa bem simples, precisamos apenas utilizar o método *setWindowTitle()*, que receberá como parâmetro uma string. Dentro do bloco do inicializador da nossa classe, deveremos escrever a seguinte linha:

```
self.setWindowTitle("Hello World")
```

Modificar o tamanho da nossa janela é tão simples quanto modificar o título. Podemos utilizar dois métodos, sendo eles: *resize()* ou *setGeometry()*. Como estamos iniciando, utilizaremos o *resize()*. Ele receberá dois parâmetros, sendo que o primeiro definirá a largura e o segundo a altura.

***Nota**: Ambos devem ser inteiros.*

```
self.resize(250, 300)
```

Modificar o ícone da nossa janela, exige um pouco mais de trabalho. Primeiro deveremos importar a classe QIcon, que está localizada dentro do módulo QtGui e não no módulo QtWidgets. Sendo assim, as nossas importações ficarão da seguinte maneira:

```
from PyQt5.QtWidgets import QApplication, QWidget
from PyQt5.QtGui import QIcon
import sys
```

Após isso, utilizaremos o método *setWindowIcon()*, onde o mesmo receberá como parâmetro uma instância da classe QIcon. A classe QIcon, por sua vez, receberá como parâmetro uma string indicando o diretório da imagem. Após a declaração do método *resize()*, deveremos incluir a seguinte linha:

```
self.setWindowIcon(QIcon("/static/logo.ico"))
```

Bem, estamos chegando ao final deste post introdutório ao PyQt5. Para realizar o download do código apresentado neste post, basta acessar esse link.

Onde posso encontrar exemplos?

Existem muitos locais na internet onde poderá encontrar exemplos de código utilizando o PyQt, porém se desejas encontrar códigos especificamente utilizando o PyQt5, você terá basicamente dois locais confiáveis. O primeiro deles é o diretório onde está instalado o próprio PyQt5. Para encontrarmos, bastamos ir até o local onde o mesmo está instalado, entrar na pasta *examples* e ver asdiversas rosas.

Outro local onde contém muito mais exemplos do que o descrito anteriormente é a documentação do Qt. Dependendo do seu conhecimento, este local poderá ser um caminho de pedras ou de rosas. Se conhece o C++ ou tem o mínimo de conhecimento possível, conseguirá se virar bem com os exemplos. Caso não tenha, ficará meio difícil. O link para a documentação do Qt5 está no início do post.

Finalizando

Enfim, chegamos a final desta introdução ao PyQt5, onde detivemos conhecimento do Qt, do próprio PyQt, dos módulos contidos dentro do PyQt5, das suas características e do seu funcionamento. Aprendemos a realizar a instalação do mesmo no Mac OS X, no Windows e no Ubuntu. Também criamos a nossa primeira aplicação, onde foi abordado o funcionamento de cada linha de código fundamental na criação de uma janela utilizando o PyQt. Após isso, brincamos um pouco quando aprendemos a modificar o título, ícone e tamanho da nossa janela.

Claro, este foi um post inicial para entendermos um pouco sobre a biblioteca. No próximo post começaremos a desenvolver um software real que se estenderá até o final do conjunto de posts.

Espero que tenha usufruído desta leitura, até mais.