LECTURE 17

GUI Programming

GUI PROGRAMMING

Today, we're going to begin looking at how we can create GUIs (Graphical User Interfaces) in Python.

So far, every application we've built has been either console-based or a web application. If we want to create a user-friendly standalone application, we really must create a nice interface for it. So, let's take our console-based Blackjack application and give it a GUI.

First, let's see what packages are available to help us do this.

GUI PROGRAMMING IN PYTHON

As it turns out, there are a huge number of modules available to help you create an interface. Some of these include:

- Tkinter: wrapper around Tcl/Tk. Python's standard GUI.
- PyQt: bindings for the Qt application framework.
- wxPython: wrapper around wxWidgets C++ library.
- pyGTK: wrapper around GTK+.
- PyJamas/PyJamas Desktop
- etc.

CHOOSING A TOOLKIT

Toolkit	Important Points
Tkinter	- Limited theme support: “look” was relatively the same until recently. - Relatively limited widget options. - Bundled with Python since the beginning. - Most commonly used.
PyQt	- Limited licensing choices. - Very good docs - Very beautiful applications. - Huge! Both good and bad...
wxPython	- Very large user base. - Great widget selection. - Bad docs. - Unlikely to disappoint halfway through the project.
pyGTK	- Originally developed for GIMP. - Stable with a full selection of widgets. - Only offers themeable native widgets on Windows + Linux. Mac lags behind somewhat. - Some quirks to work around.

PYQT

In this class, we will be exploring basic GUI development with PyQt since it is a widely-used toolkit. However, the other mentioned options are all very good and it may be necessary to branch out depending on the complexity of the application.

First, let's discuss some of the mechanics of GUI development. Afterwards, we could spend tons of class time learning about PyQt's ~1000 classes but we won't. What we'll do is build an application together and get familiar with the common parts of a PyQt application.

As a note, we'll be using PyQt4 in this lecture. PyQt5 is the most up-to-date version.

GUI PROGRAMMING IN PYQT

Firstly, PyQt is a multi-platform GUI toolkit. It has approximately ~1000 classes divided into a set of modules. Among these are QtCore and QtGui – the most commonly used PyQt modules.

QtCore contains non-GUI core functionality (e.g. data types, files, urls, etc).

QtGui contains GUI core functionality (e.g. buttons, windows, toolbars, etc).

Besides these, there are other modules (like, QtNetwork and QtOpenGL) which focus on specific functionality.

GUI PROGRAMMING IN PYQT

Programming a GUI is not that different than programming an object-oriented console application.

What is different is that GUI programming involves the use of a *Toolkit* and GUI developers must follow the pattern of program design specified by the toolkit.

As a developer, you should be familiar with the API and design rules of at least one toolkit – preferably one that is multiplatform with bindings in many languages!

GUI PROGRAMMING IN PYQT

GUI programming necessarily means object-oriented programming with an event-driven framework. This should make sense: your job as a GUI developer is to create an application that responds to events.

For instance, when a user clicks their mouse on a certain button, the program should do X. When the user presses enter in a text field, the program should do Y. You're defining the behavior of the program as a response to external events.

BASIC PYQT

Let's dive right in by looking at an embarrassingly basic PyQt program.

```
import sys
from PyQt4 import QtGui # Import basic GUI components

app = QtGui.QApplication(sys.argv)
w = QtGui.QWidget()
w.resize(250, 150)
w.move(300, 300)
w.setWindowTitle('Our first gooey!')
w.show()
app.exec_()
```

The QApplication class manages the application's control flow and main settings. It controls the main event loop through which all events are handled and scheduled. No matter how many windows there are, there is only one QApplication instance.

BASIC PYQT

Let's dive right in by looking at an embarrassingly basic PyQt program.

```
import sys
from PyQt4 import QtGui # Import basic GUI components

app = QtGui.QApplication(sys.argv)
w = QtGui.QWidget()
w.resize(250, 150)
w.move(300, 300)
w.setWindowTitle('Our first gooey!')
w.show()
app.exec_()
```

A widget is a control element which is visible and can be manipulated by the user. These include elements such as buttons, text fields, radio selections, etc.

If we create a basic widget Qwidget instance without a parent widget, it automatically becomes a window.

BASIC PYQT

Let's dive right in by looking at an embarrassingly basic PyQt program.

```
import sys
from PyQt4 import QtGui # Import basic GUI components

app = QtGui.QApplication(sys.argv)
w = QtGui.QWidget()
w.resize(250, 150)
w.move(300, 300)
w.setWindowTitle('Our first gooey!')
w.show()
app.exec_()
```

QWidget implements a variety of methods for it and its derived classes. These include resize, move, setWindowTitle (for widgets with no parents), among many others.

Widgets and their children are created in memory and made visible with the show() method.

BASIC PYQT

Let's dive right in by looking at an embarrassingly basic PyQt program.

```
import sys
from PyQt4 import QtGui # Import basic GUI components

app = QtGui.QApplication(sys.argv)
w = QtGui.QWidget()
w.resize(250, 150)
w.move(300, 300)
w.setWindowTitle('Our first gooey!')
w.show()
app.exec_()
```

Calling exec_() on our QApplication instance will start our main event loop.

BASIC PYQT

Let's dive right in by looking at an embarrassingly basic PyQt program.

```
import sys
from PyQt4 import QtGui # Import basic GUI components

app = QtGui.QApplication(sys.argv)
w = QtGui.QWidget()
w.resize(250, 150)
w.move(300, 300)
w.setWindowTitle('Our first gooey!')
w.show()
app.exec_()
```

BASIC PYQT

Here we introduce a new GUI component: tooltips! (Those things that pop up to give you info about whatever you're hovering over.)

Also, we have buttons. We pass the button text as the first argument and the parent widget as the next argument.

```
import sys
from PyQt4 import QtGui, QtCore

class Example(QtGui.QWidget):
    def __init__(self):
        super(Example, self).__init__()
        self.initUI()

    def initUI(self):
        QtGui.QToolTip.setFont(QtGui.QFont('SansSerif', 10))
        self.setToolTip('This is window for our gooey.')

        btn = QtGui.QPushButton('Click me if you dare.', self)
        btn.setToolTip('This is a button for our gooey.')
        btn.resize(btn.sizeHint())
        btn.move(50, 50)

        qbtn = QtGui.QPushButton('or just quit', self)
        qbtn.clicked.connect(QtCore.QCoreApplication.instance().quit)
        qbtn.resize(qbtn.sizeHint())
        qbtn.move(100, 100)

        self.setGeometry(300, 300, 250, 150)
        self.setWindowTitle('Our first gooey!')
        self.show()

app = QtGui.QApplication(sys.argv)
ex = Example()
app.exec_()
```

BASIC PYQT

sizeHint() returns the recommended size for a widget.

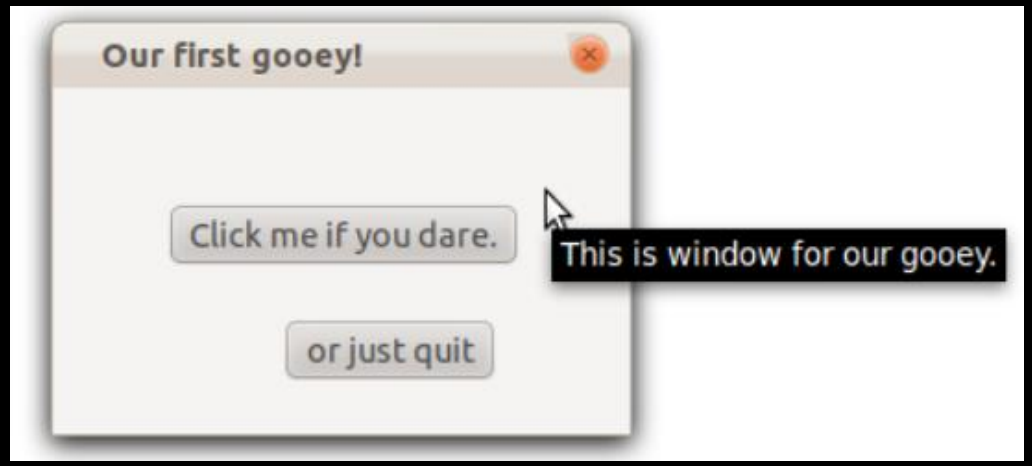
move() moves widget to x,y in parent.

btn has no associated action but qbtn will quit the application when clicked.

setGeometry() represents a call to move and resize all at once.

```
import sys
from PyQt4 import QtGui, QtCore

class Example(QtGui.QWidget):
    def __init__(self):
        super(Example, self).__init__()
        self.initUI()

    def initUI(self):
        QtGui.QToolTip.setFont(QtGui.QFont('SansSerif', 10))
        self.setToolTip('This is window for our gooey.')

        btn = QtGui.QPushButton('Click me if you dare.', self)
        btn.setToolTip('This is a button for our gooey.')
        btn.resize(btn.sizeHint())
        btn.move(50, 50)

        qbtn = QtGui.QPushButton('or just quit', self)
        qbtn.clicked.connect(QtCore.QCoreApplication.instance().quit)
        qbtn.resize(qbtn.sizeHint())
        qbtn.move(100, 100)

        self.setGeometry(300, 300, 250, 150)
        self.setWindowTitle('Our first gooey!')
        self.show()

app = QtGui.QApplication(sys.argv)
ex = Example()
app.exec_()
```

BASIC PYQT




```
import sys
from PyQt4 import QtGui, QtCore

class Example(QtGui.QWidget):
    def __init__(self):
        super(Example, self).__init__()
        self.initUI()

    def initUI(self):
        QtGui.QToolTip.setFont(QtGui.QFont('SansSerif', 10))
        self.setToolTip('This is window for our gooey.')

        btn = QtGui.QPushButton('Click me if you dare.', self)
        btn.setToolTip('This is a button for our gooey.')
        btn.resize(btn.sizeHint())
        btn.move(50, 50)

        qbtn = QtGui.QPushButton('or just quit', self)
        qbtn.clicked.connect(QtCore.QCoreApplication.instance().quit)
        qbtn.resize(qbtn.sizeHint())
        qbtn.move(100, 100)

        self.setGeometry(300, 300, 250, 150)
        self.setWindowTitle('Our first gooey!')
        self.show()

app = QtGui.QApplication(sys.argv)
ex = Example()
app.exec_()
```

BASIC PYQT


```
import sys
from PyQt4 import QtGui, QtCore

class Example(QtGui.QWidget):
    def __init__(self):
        super(Example, self).__init__()
        self.initUI()

    def initUI(self):
        QtGui.QToolTip.setFont(QtGui.QFont('SansSerif', 10))
        self.setToolTip('This is window for our gooey.')

        btn = QtGui.QPushButton('Click me if you dare.', self)
        btn.setToolTip('This is a button for our gooey.')
        btn.resize(btn.sizeHint())
        btn.move(50, 50)

        qbtn = QtGui.QPushButton('or just quit', self)
        qbtn.clicked.connect(QtCore.QCoreApplication.instance().quit)
        qbtn.resize(qbtn.sizeHint())
        qbtn.move(100, 100)

        self.setGeometry(300, 300, 250, 150)
        self.setWindowTitle('Our first gooey!')
        self.show()

app = QtGui.QApplication(sys.argv)
ex = Example()
app.exec_()
```

BASIC PYQT

We are also free to define the behavior of our application by overriding built-in methods.

For example, QtGui.QWidget has a method called closeEvent() which receives an instance of a window close request.
By default, we just accept the request and close the window.
Here, we'll override the function to warn the user that they're making a terrible mistake.

```
import sys
from PyQt4 import QtGui, QtCore

class Example(QtGui.QWidget):
    def __init__(self):
        super(Example, self).__init__()
        self.initUI()
    def initUI(self):
        ...
    def closeEvent(self, event):
        reply = QtGui.QMessageBox.question(self, 'Message',"Are you sure?",
                    QtGui.QMessageBox.Yes|QtGui.QMessageBox.No,
                    QtGui.QMessageBox.No)
        if reply == QtGui.QMessageBox.Yes:
            event.accept()
        else:
            event.ignore()

app = QtGui.QApplication(sys.argv)
ex = Example()
app.exec_()
```

BASIC PYQT

We introduce a new GUI component, the Message Box. This is just a little pop-up message for the user with some buttons.

The Message Box has a static method called question into which we're passing in the parent widget, title, text, yes button and no buttons (already defined), and the default button.

Depending on the selection made by the user, we either grant the request or discard it.

```
import sys
from PyQt4 import QtGui, QtCore

class Example(QtGui.QWidget):
    def __init__(self):
        super(Example, self).__init__()
        self.initUI()
    def initUI(self):
        ...
    def closeEvent(self, event):
        reply = QtGui.QMessageBox.question(self, 'Message',"Are you sure?",
                    QtGui.QMessageBox.Yes|QtGui.QMessageBox.No,
                    QtGui.QMessageBox.No)
        if reply == QtGui.QMessageBox.Yes:
            event.accept()
        else:
            event.ignore()

app = QtGui.QApplication(sys.argv)
ex = Example()
app.exec_()
```

BASIC PYQT

```
import sys
from PyQt4 import QtGui, QtCore

class Example(QtGui.QWidget):
    def __init__(self):
        super(Example, self).__init__()
        self.initUI()
    def initUI(self):
        ...
    def closeEvent(self, event):
        reply = QtGui.QMessageBox.question(self, 'Message',"Are you sure?",
                    QtGui.QMessageBox.Yes|QtGui.QMessageBox.No,
                    QtGui.QMessageBox.No)
       if reply == QtGui.QMessageBox.Yes:
            event.accept()
       else:
            event.ignore()

app = QtGui.QApplication(sys.argv)
ex = Example()
app.exec_()
```

BASIC PYQT

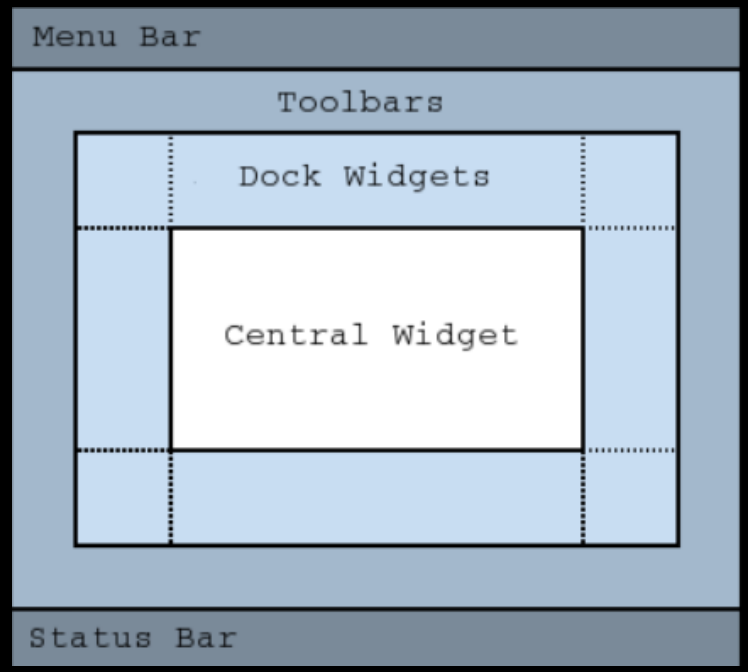
```
import sys
from PyQt4 import QtGui, QtCore

class Example(QtGui.QWidget):
    def __init__(self):
        super(Example, self).__init__()
        self.initUI()
    def initUI(self):
        ...
    def closeEvent(self, event):
        reply = QtGui.QMessageBox.question(self, 'Message',"Are you sure?",
                    QtGui.QMessageBox.Yes|QtGui.QMessageBox.No,
                    QtGui.QMessageBox.No)
       if reply == QtGui.QMessageBox.Yes:
            event.accept()
       else:
            event.ignore()

app = QtGui.QApplication(sys.argv)
ex = Example()
app.exec_()
```

QT MAIN WINDOW

The QMainWindow class provides a main application window. QMainWindow has its own layout as opposed to QWidget (but it inherits from QWidget).

QMainWindow allows for:

- QToolBar
- QDockWidget
- QMenuBar
- QStatusBar
- Any widget can occupy Central Widget.



BASIC PYQT

Look! We're inheriting from QMainWindow now!

We're going to add a menuBar with File > Exit. First, we create a QAction instance for exiting the application, for which the text is "Exit". We add a Ctrl-Q shortcut for it and a status tip which will display in the status bar of our main window.

When the action is taken, we quit the application.

```
import sys
from PyQt4 import QtGui, QtCore

class Example(QtGui.QMainWindow):
    def __init__(self):
        super(Example, self).__init__()
        self.initUI()
    def initUI(self):
        ...
        exitAction = QtGui.QAction('Exit', self)
        exitAction.setShortcut('Ctrl+Q')
        exitAction.setStatusTip('Exit application')
        exitAction.triggered.connect(QtGui.qApp.quit)

        menubar = self.menuBar()
        fileMenu = menubar.addMenu('File')
        fileMenu.addAction(exitAction)
        ...
    def closeEvent(self, event):
        ...

app = QtGui.QApplication(sys.argv)
ex = Example()
app.exec_()
```

BASIC PYQT

Look! We're inheriting from QMainWindow now!

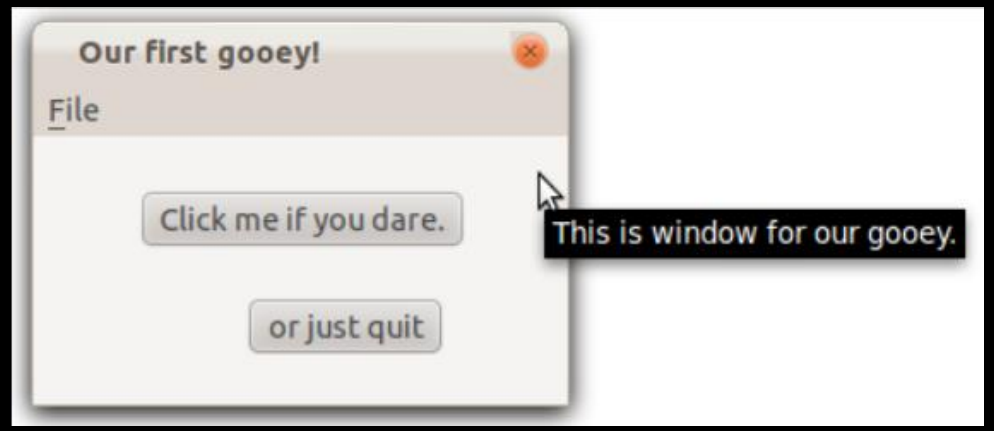
We reference the menu bar for our main window by calling menuBar(). Will also create and return an empty menu bar if there is none.

Then we add a new menu with the text "File".
Then we add the Exit action to the File menu.

```
import sys
from PyQt4 import QtGui, QtCore

class Example(QtGui.QMainWindow):
    def __init__(self):
        super(Example, self).__init__()
        self.initUI()
    def initUI(self):
        ...
        exitAction = QtGui.QAction('Exit', self)
        exitAction.setShortcut('Ctrl+Q')
        exitAction.setStatusTip('Exit application')
        exitAction.triggered.connect(QtGui.qApp.quit)

        menubar = self.menuBar()
        fileMenu = menubar.addMenu('File')
        fileMenu.addAction(exitAction)
        ...
    def closeEvent(self, event):
        ...

app = QtGui.QApplication(sys.argv)
ex = Example()
app.exec_()
```

BASIC PYQT

Look! We're inheriting from QMainWindow now!




```
import sys
from PyQt4 import QtGui, QtCore

class Example(QtGui.QMainWindow):
    def __init__(self):
        super(Example, self).__init__()
        self.initUI()
    def initUI(self):
        ...
        exitAction = QtGui.QAction('Exit', self)
        exitAction.setShortcut('Ctrl+Q')
        exitAction.setStatusTip('Exit application')
        exitAction.triggered.connect(QtGui.qApp.quit)

        menubar = self.menuBar()
        fileMenu = menubar.addMenu('File')
        fileMenu.addAction(exitAction)
        ...
    def closeEvent(self, event):
        ...

app = QtGui.QApplication(sys.argv)
ex = Example()
app.exec_()
```

BASIC PYQT

Look! We're inheriting from QMainWindow now!


```
import sys
from PyQt4 import QtGui, QtCore

class Example(QtGui.QMainWindow):
    def __init__(self):
        super(Example, self).__init__()
        self.initUI()
    def initUI(self):
        ...
        exitAction = QtGui.QAction('Exit', self)
        exitAction.setShortcut('Ctrl+Q')
        exitAction.setStatusTip('Exit application')
        exitAction.triggered.connect(QtGui.qApp.quit)

        menubar = self.menuBar()
        fileMenu = menubar.addMenu('File')
        fileMenu.addAction(exitAction)
        ...
    def closeEvent(self, event):
        ...

app = QtGui.QApplication(sys.argv)
ex = Example()
app.exec_()
```