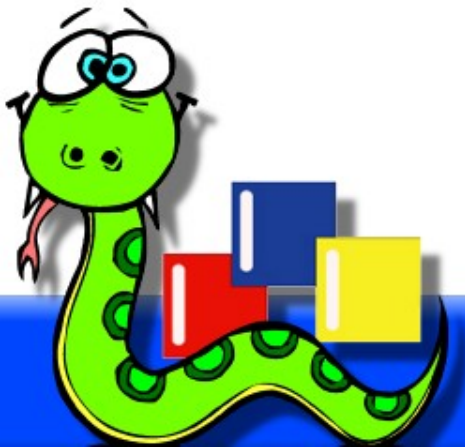


# An Introduction to wxPython

Robin Dunn

O'Reilly Open Source Convention  
July 24–28, 2006



# Presentation Overview

- Your Instructor
- Introduction
- GUI Basics
- wxPython Fundamentals
- wxPython Widgets
- Event Handling
- Window Layout
- Device Contexts
- Tools



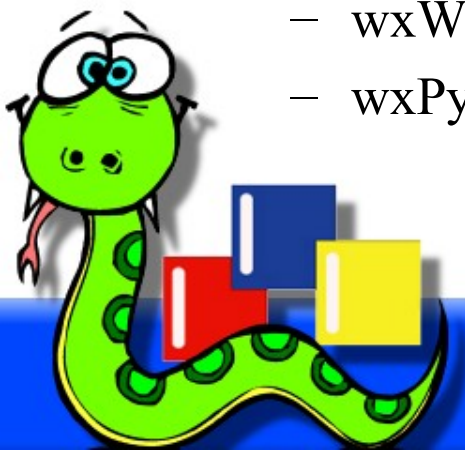
# Robin Dunn

- Creator and maintainer of wxPython
- Programming for 25+ years, started in Junior High
- Working on wxPython for about 9 years
  - More or less full-time for 3+ years
- Member of the wxWidgets core team
- Currently on contract with OSAF
- Coauthor of *wxPython in Action*

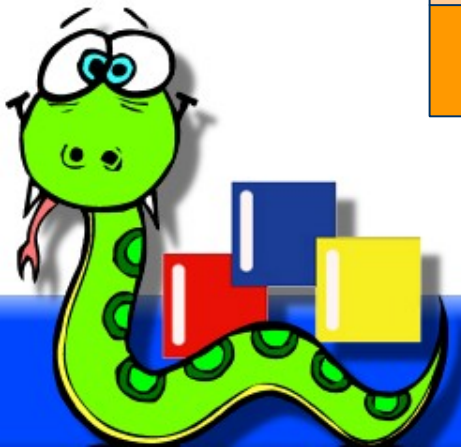
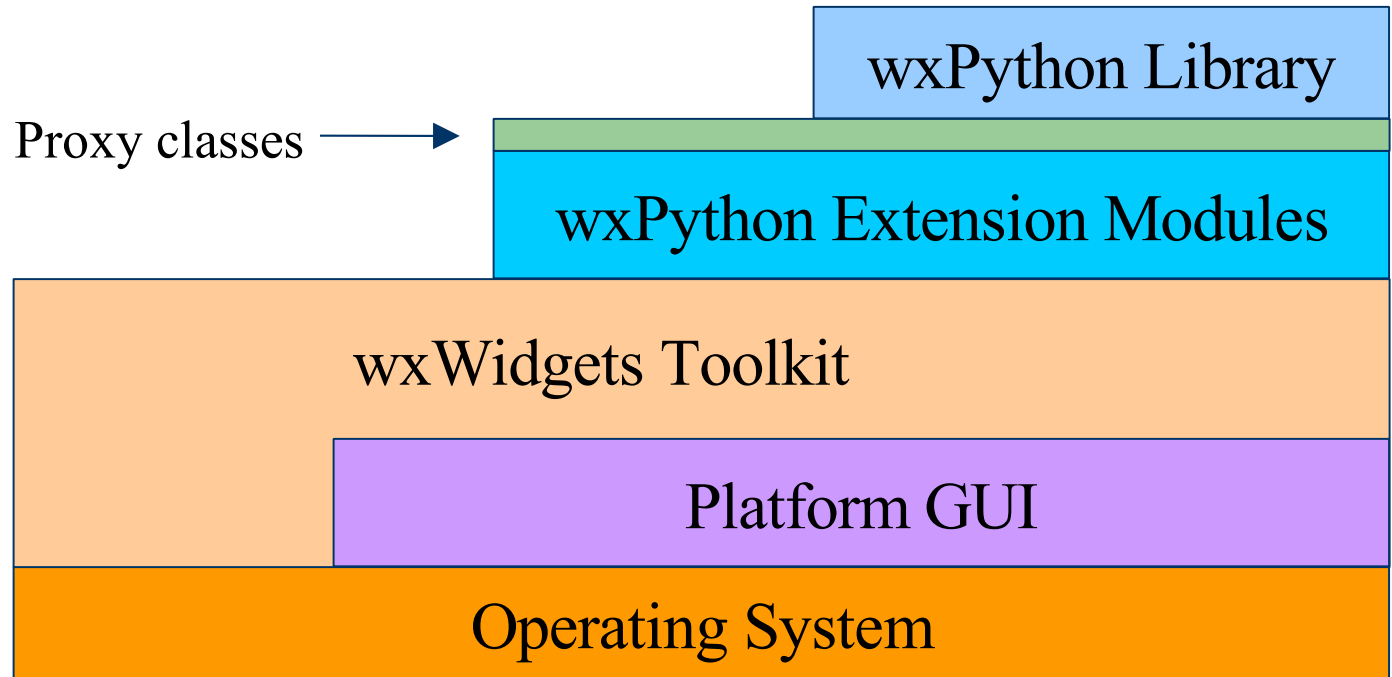


# Why wxPython?

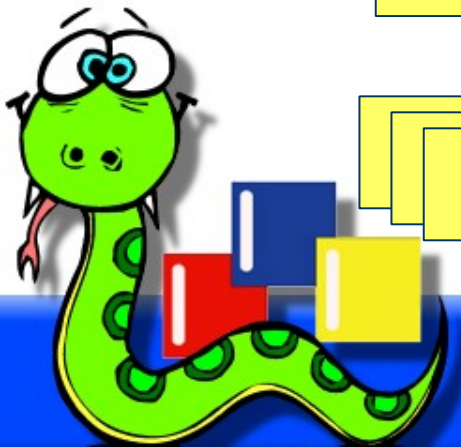
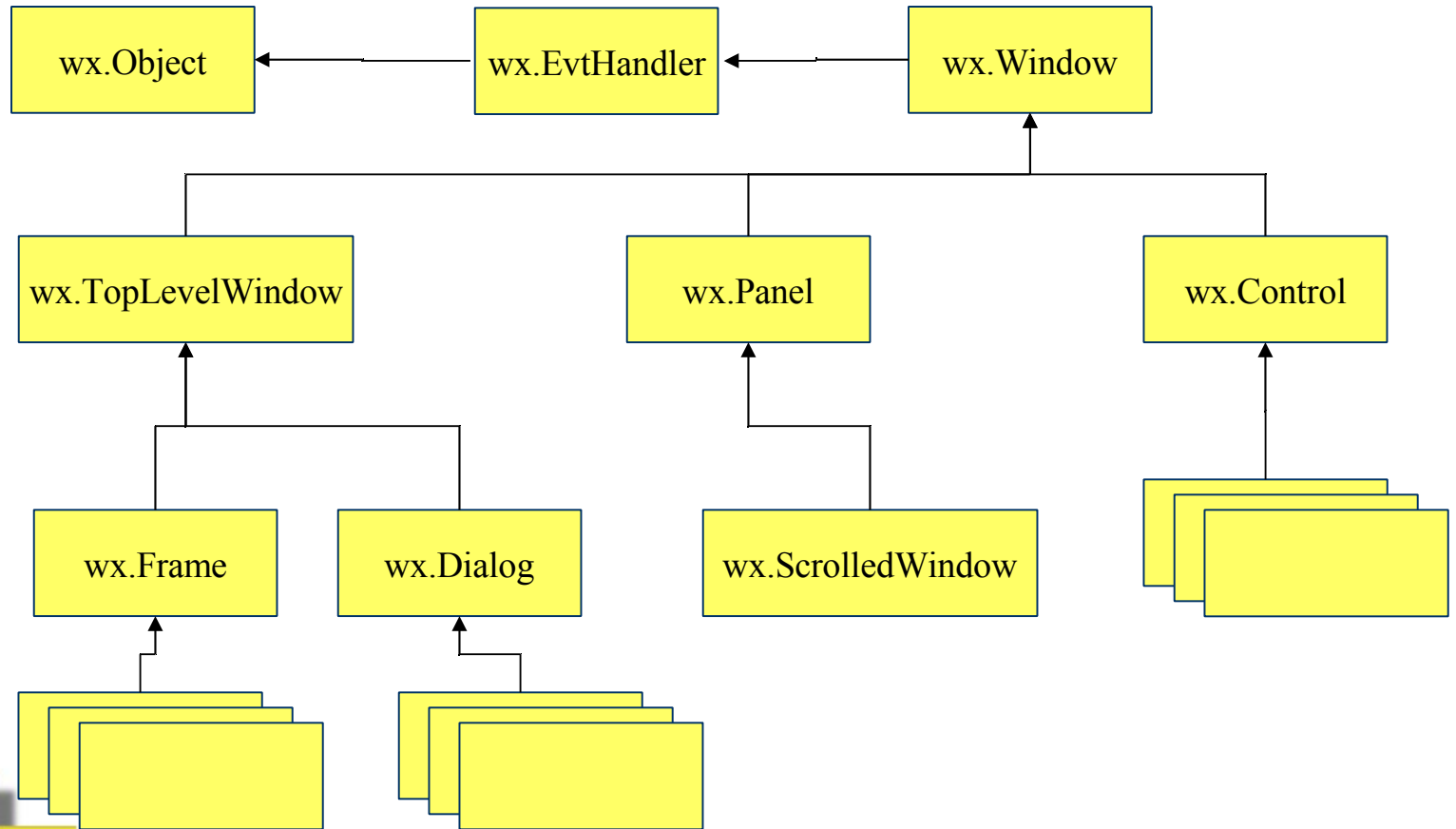
- wxPython is an **open source** GUI toolkit based on the wxWidgets (formerly wxWindows) library
- Designed to be cross-platform and supports most Unix/Linux platforms, MS Windows and Mac OS X
- Uses **native widgets** wherever possible to preserve native Look and Feel.
- Extensive sample programs, helpful and capable community
- Mature, well established projects.
  - wxWidgets: 1992
  - wxPython: 1996



# Basic architecture



# Partial class hierarchy



# Getting started with wxPython

- Choose an installer
  - <http://wxPython.org/downloads.php>
  - Windows \*.exe installers, Linux RPMs or OSX \*.dmg
  - Can be built from source with a few prerequisites
- Which version of Python do you use?
  - 2.3, 2.4, 2.5
- Unicode or ANSI?
  - Unicode builds available on all platforms, but be careful with Win9x/ME
  - ANSI available for platforms, but may be phased out soon.



# Getting started with wxPython

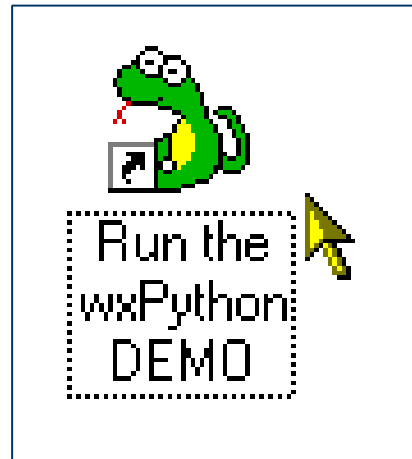
- Choose an editor or development environment:
  - Boa Constructor
  - WingIDE
  - SPE
  - SCiTE
  - Emacs, vi, etc.
- It's just plain text, so any ordinary editor and command line will do.



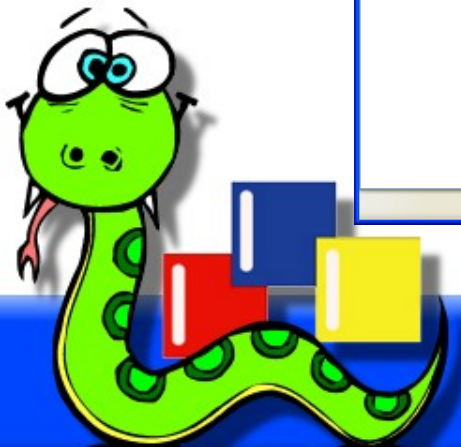
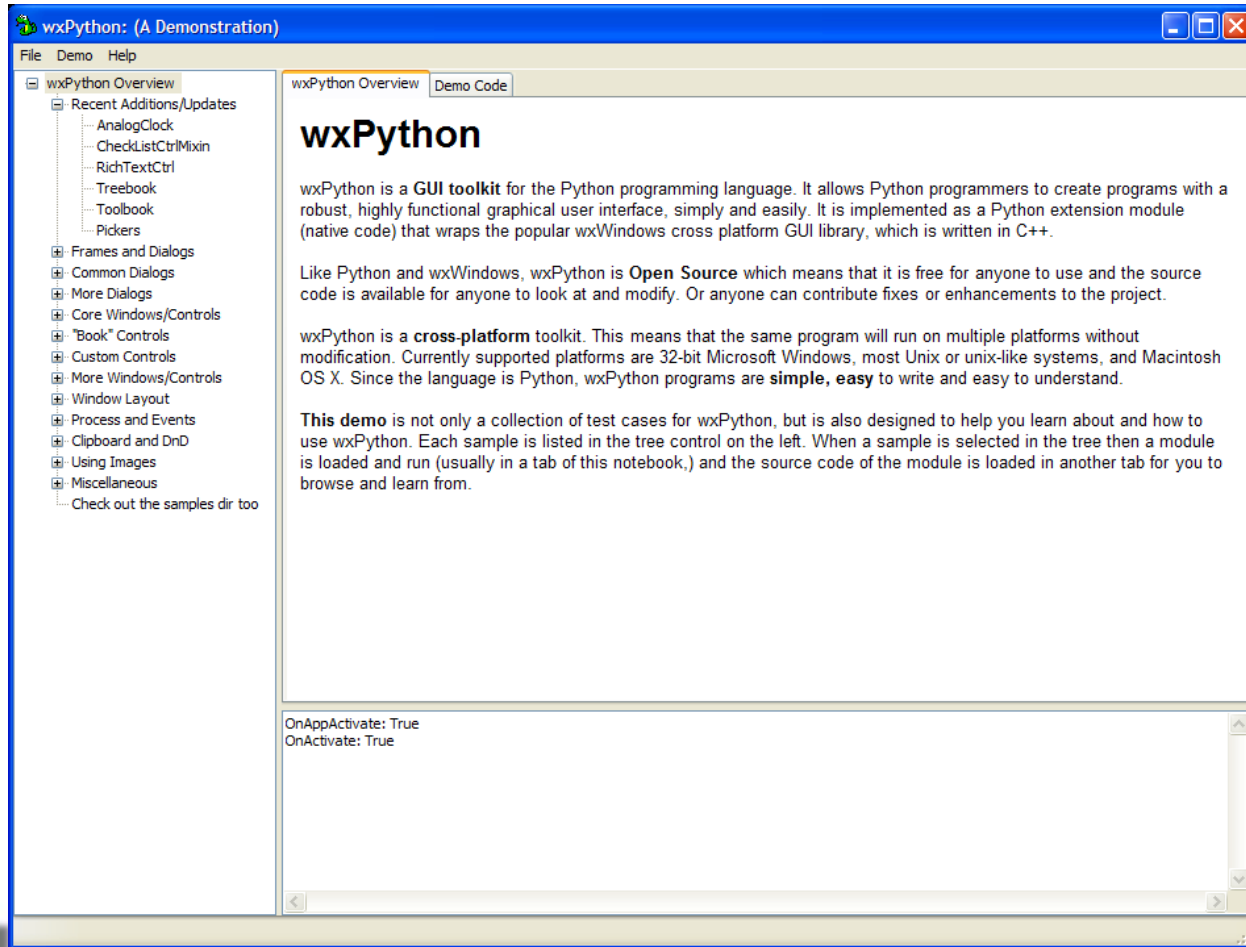


# Getting started with wxPython

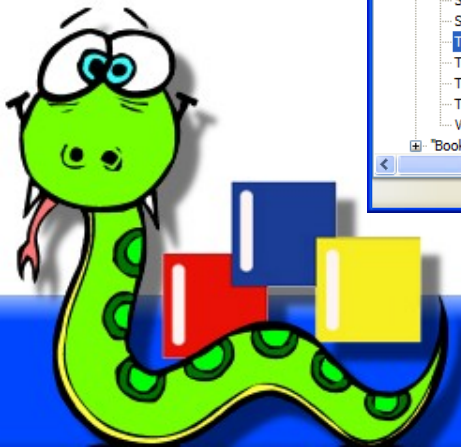
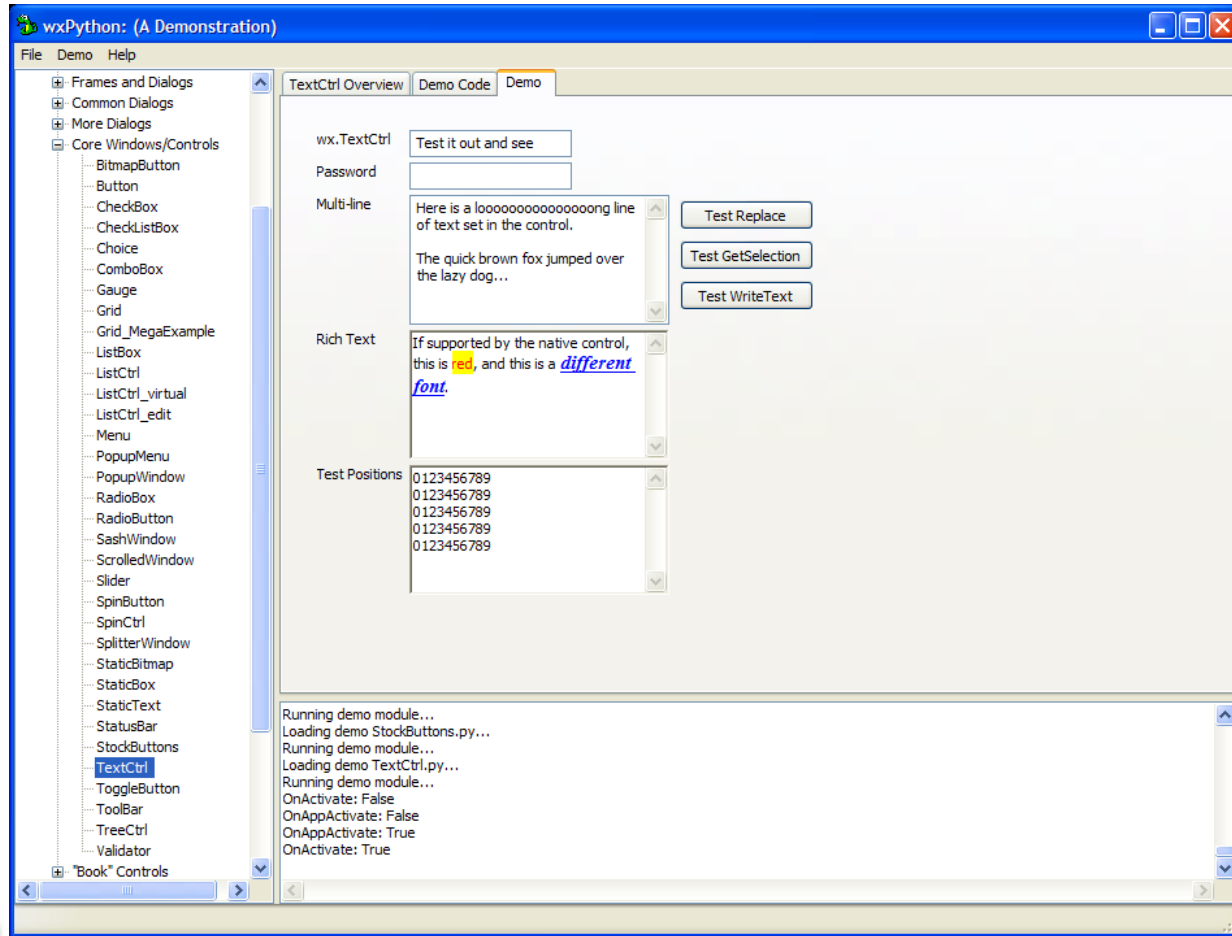
- Ready, set, go!
- The wxPython Demo is a great way to learn about the capabilities of the toolkit.



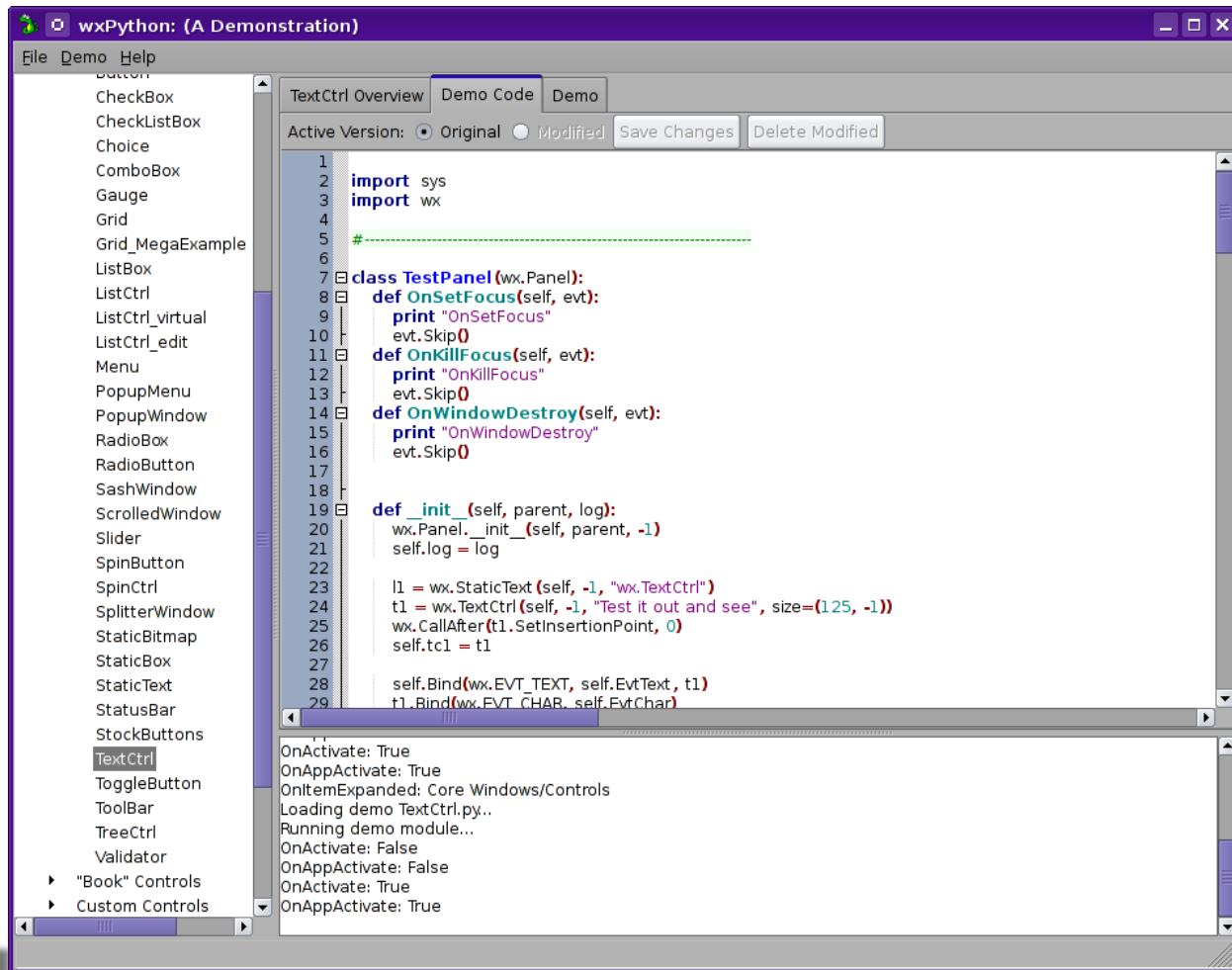
# Getting started with wxPython



# Getting started with wxPython

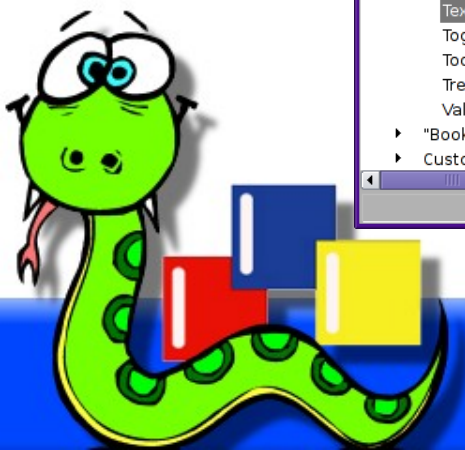


# Getting started with wxPython

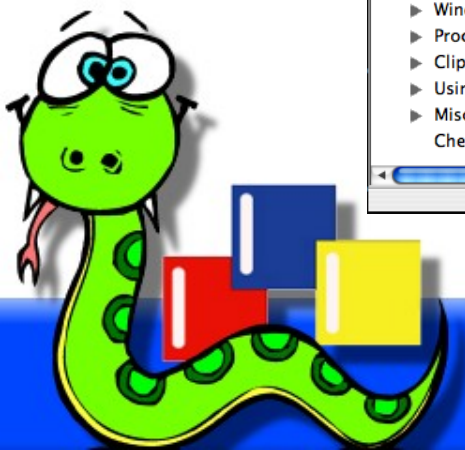
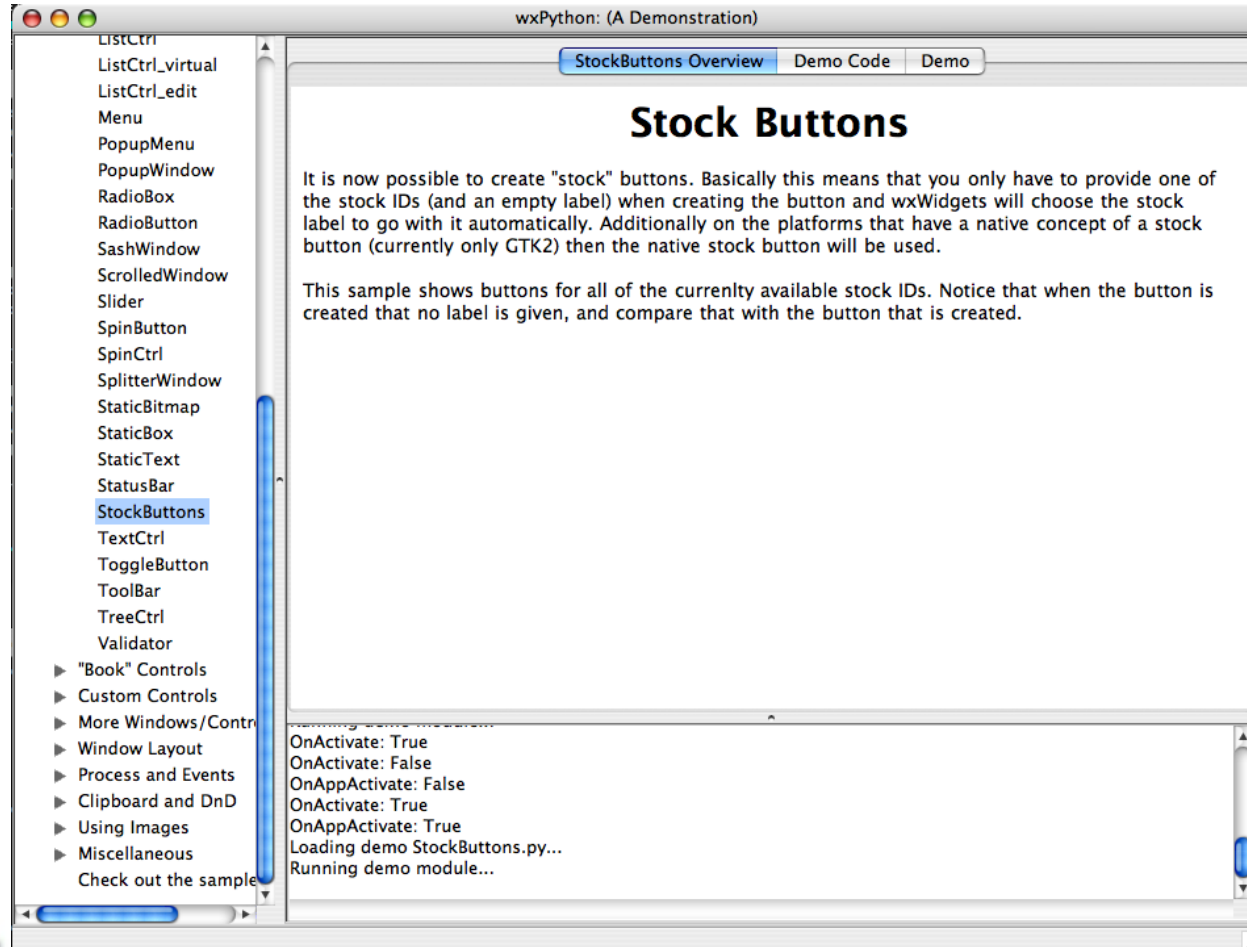


```
1 import sys
2 import wx
3
4 #-----
5
6
7 class TestPanel(wx.Panel):
8     def OnSetFocus(self, evt):
9         print "OnSetFocus"
10        evt.Skip()
11    def OnKillFocus(self, evt):
12        print "OnKillFocus"
13        evt.Skip()
14    def OnWindowDestroy(self, evt):
15        print "OnWindowDestroy"
16        evt.Skip()
17
18
19 def __init__(self, parent, log):
20     wx.Panel.__init__(self, parent, -1)
21     self.log = log
22
23     t1 = wx.StaticText(self, -1, "wx.TextCtrl")
24     t1 = wx.TextCtrl(self, -1, "Test it out and see", size=(125, -1))
25     wx.CallAfter(t1.SetInsertionPoint, 0)
26     self.tc1 = t1
27
28     self.Bind(wx.EVT_TEXT, self.EvtText, t1)
29     t1.Bind(wx.EVT_CHAR, self.EvtChar)
```

OnActivate: True  
OnAppActivate: True  
OnItemExpanded: Core Windows/Controls  
Loading demo TextCtrl.py..  
Running demo module...  
OnActivate: False  
OnAppActivate: False  
OnActivate: True  
OnAppActivate: True



# Getting started with wxPython



# Demo time...



# GUI Basics

- GUI's are composed of a collection of windows, or *widgets*
  - Some widgets are top-level windows that are managed by the OS
  - Some are contained in other widgets
- You can think of a window as a *tree of graphical components*
- Before you can display a window you must
  - Create the component tree, and (optionally) ...
  - Associate events with particular objects and actions



# GUI Basics

- GUI applications are *event driven*
  - The application spends most of its time waiting for something to happen, such as a keystroke, or mouse movement.
  - When that something (*the event*) happens, information about it is collected and sent to a handler.
  - Events are dispatched asynchronously
    - meaning they can happen in any order



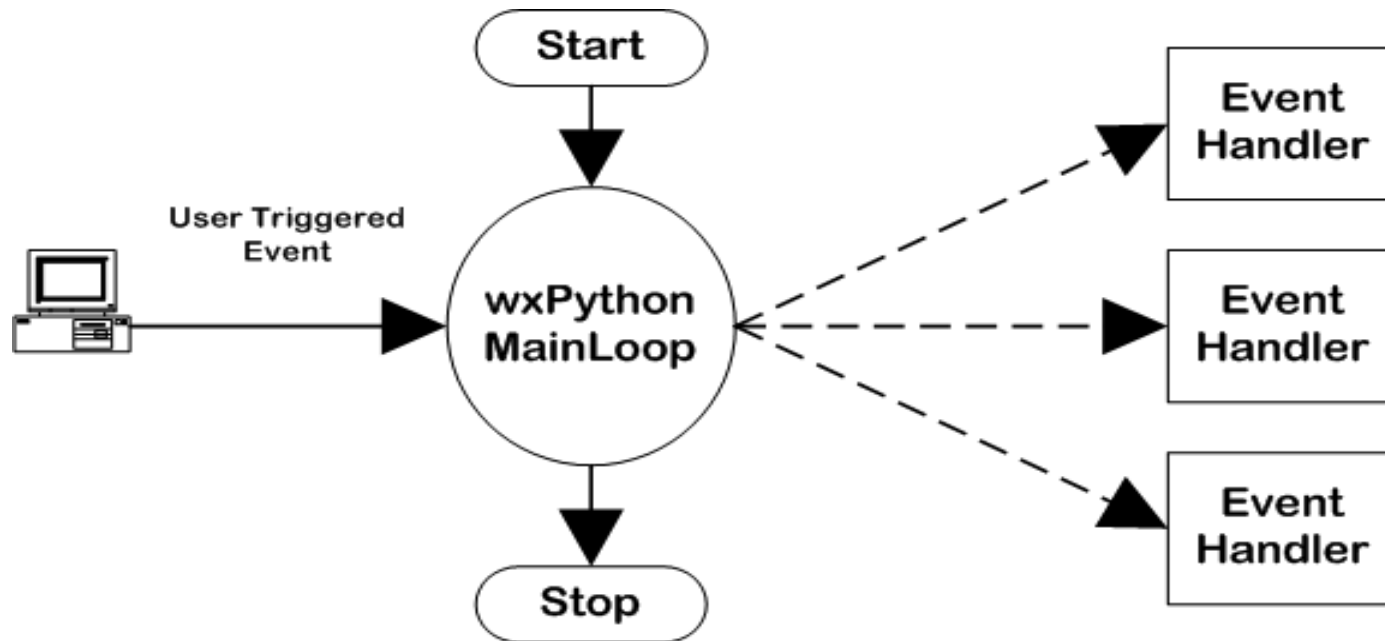


# GUI Basics

- Many events are a direct result of user actions
  - Left-click on a button
  - Select a menu item
  - Drag an item from one panel to another
    - That would actually be a sequence of events
- Other events are raised by the system
  - Timer countdown expires
  - An obscured part of a window is exposed



# GUI Basics



# Hello World!

```
# ex01.py
import wx

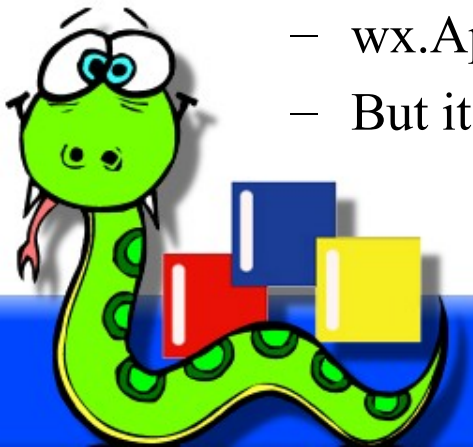
class App(wx.App):
    def OnInit(self):
        frame = wx.Frame(parent=None, title="Hello World! 1")
        frame.Show()
        return True

app = App()
app.MainLoop()
```



# wxPython Fundamentals

- Every application needs an instance of the `wx.App` class
  - Some parts of the C++ library are not initialized until the app is created, so it must be done before most other things.
  - APIs for starting and stopping the application
  - Provides the central *event loop* and dispatches events to handlers
  - Other per-application functionality
- Traditionally, you subclass `wx.App` and override `OnInit` for creating the initial application widgets
  - Not strictly needed any longer
  - `wx.App` can be used without subclassing
  - But it often still makes sense for design purposes



# Hello World!

```
# ex02.py
import wx

app = wx.App()
frame = wx.Frame(parent=None, title="Hello World! 2")
frame.Show()
app.MainLoop()
```



# wxPython Fundamentals

- `wx.App` can redirect standard output
  - Sends print statements and writes to `sys.stdout` or `sys.stderr` to a window or a file
  - An easy way to view status messages or tracebacks
  - Controlled by parameters to `wx.App.__init__`



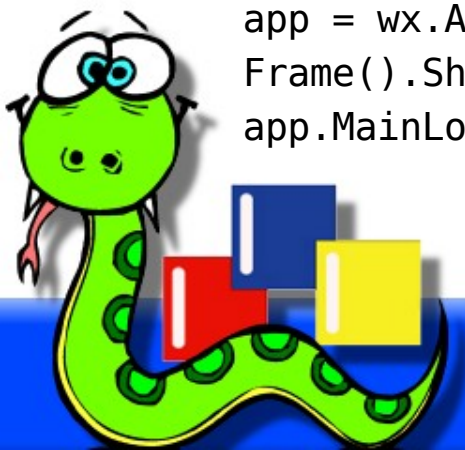
# Hello World!

```
# ex03.py
import wx

class Frame(wx.Frame):
    def __init__(self):
        wx.Frame.__init__(self, parent=None, title="Hello World! 3")
        b1 = wx.Button(self, label="Hello", pos=(20,20))
        b2 = wx.Button(self, label="World", pos=(20,60))
        self.Bind(wx.EVT_BUTTON, self.OnHelloWorld)

    def OnHelloWorld(self, evt):
        print "Hello World!"

app = wx.App(redirect=True)
Frame().Show()
app.MainLoop()
```



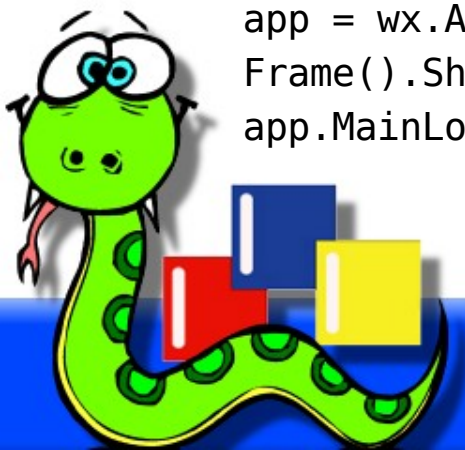
# Hello World!

```
# ex03.py
import wx

class Frame(wx.Frame):
    def __init__(self):
        wx.Frame.__init__(self, parent=None, title="Hello World! 3")
        b1 = wx.Button(self, label="Hello", pos=(20,20))
        b2 = wx.Button(self, label="World", pos=(20,60))
        self.Bind(wx.EVT_BUTTON, self.OnHelloWorld)

    def OnHelloWorld(self, evt):
        print "Hello World!"

app = wx.App(redirect=True)
Frame().Show()
app.MainLoop()
```





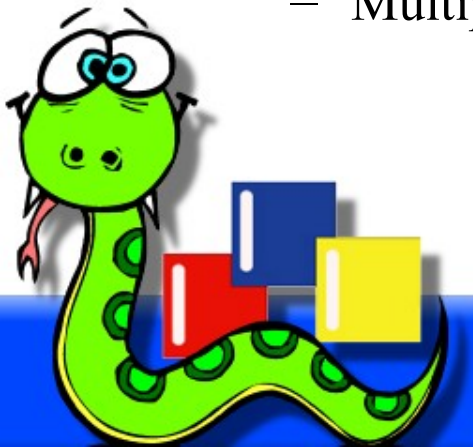
# wxPython Fundamentals

- Every application needs a `wx.App` and one or more top level windows
- Window/Widget classes can be used directly, but you will often subclass them to add-in your application's functionality
- Events are signals from the user or the system that your application is interested in.
- Events are delivered to event handler functions (usually members of the derived widget classes)
- Events can happen in any order



# wxPython widgets: top level windows

- wx.Frame
  - A container for other windows.
  - Can automatically manage a MenuBar, ToolBar, and a StatusBar.
- wx.Dialog
  - For Modal or Modeless dialog boxes.
- wx.Miniframe
  - Good for floating tool pallets, etc.
- wx.MDIParentFrame, wx.MDIChildFrame
  - Multiple Document Interface

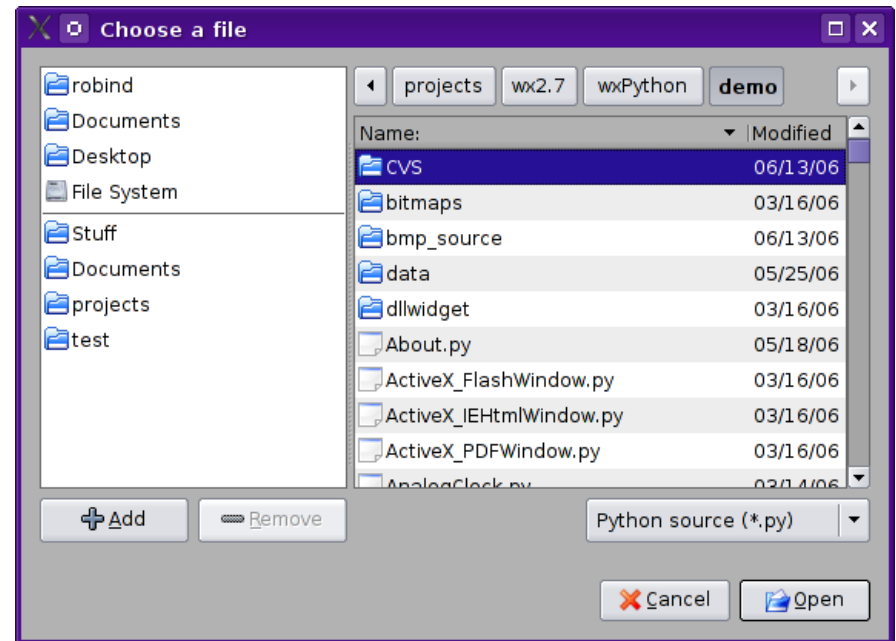
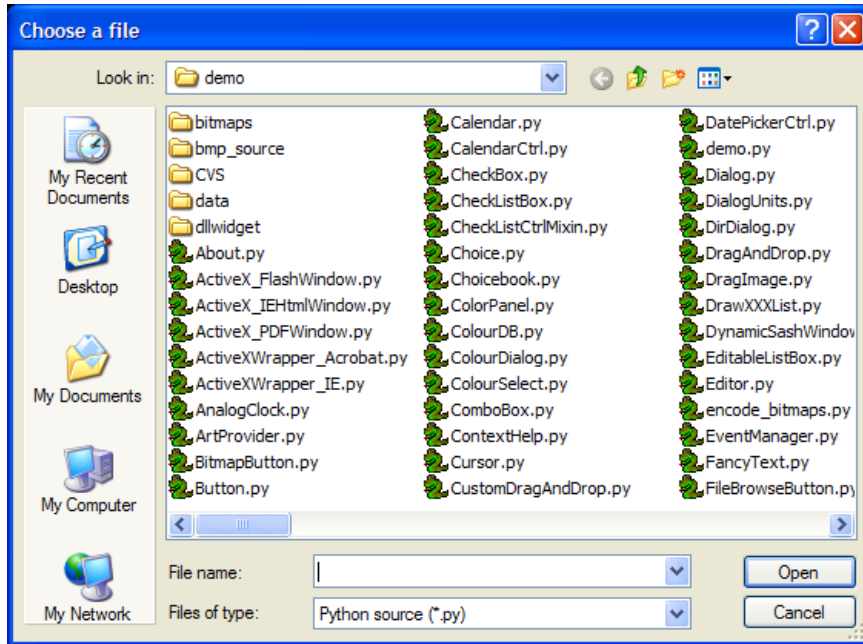


# wxPython widgets : common dialogs

- All standard Windows common dialogs:
  - Color, Directory, File,
  - Font, PageSetup, Print,
  - Message, Progress,
  - FindReplace, etc.
- For other platforms either native dialogs are used, or suitable recreations in wxWidgets are provided.



# wxPython widgets : common dialogs



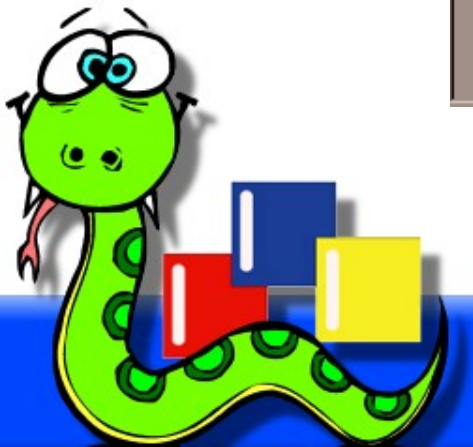
# wxPython widgets : basic windows

- wx.Window
  - General purpose window.
- wx.Panel
  - Can do tab-traversal of controls.
  - Uses standard system color for the background.
- wx.ScrolledWindow
  - Manages its own scrollbars and scrolling of client area.
  - Transforms coordinates based on scrollbar positions.



# wxPython widgets

- wx.SplitterWindow
  - Can be split vertically or horizontally.
  - Draggable sash for redistributing the space between sub-windows.

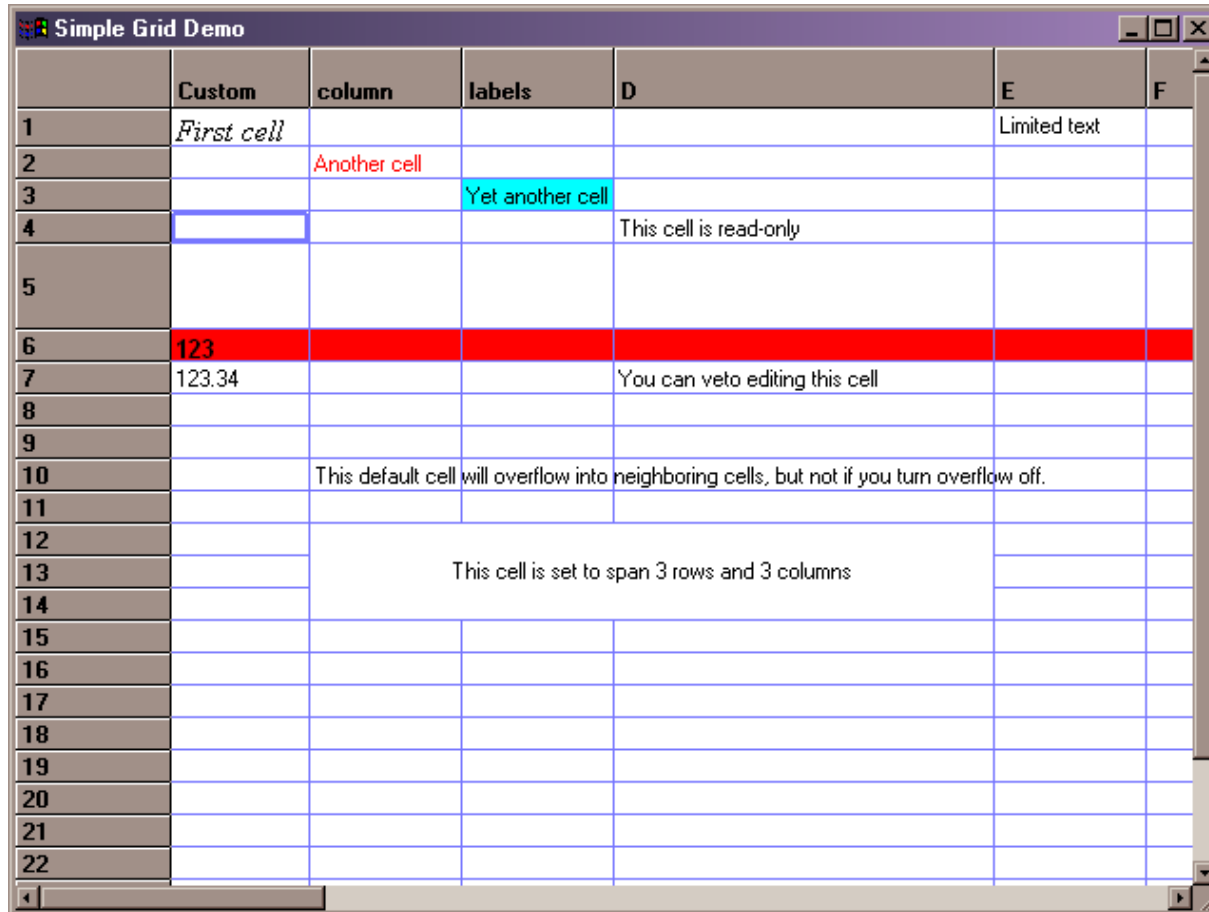


# wxPython widgets

- `wx.grid.Grid`
  - Table or spreadsheet-like capabilities.
  - Editors, Renderers, Tables (the data provider) can all be customized and “plugged in”.

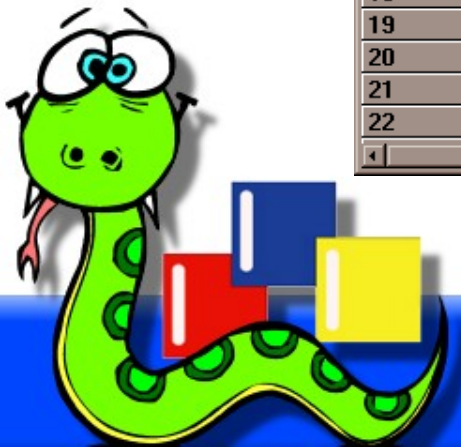


# wxPython widgets



The screenshot shows a window titled "Simple Grid Demo" containing a grid with 22 rows and 6 columns. The columns are labeled "Custom", "column", "labels", "D", "E", and "F". The rows are numbered 1 through 22. The grid demonstrates various wxPython grid features:

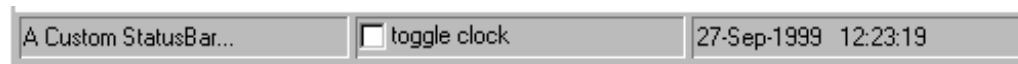
- Row 1: Cell (1, Custom) contains "First cell" in italics. Cell (1, E) contains "Limited text".
- Row 2: Cell (2, column) contains "Another cell" in red text.
- Row 3: Cell (3, labels) contains "Yet another cell" in cyan text.
- Row 4: Cell (4, Custom) is empty and has a blue border. Cell (4, D) contains "This cell is read-only".
- Row 6: The entire row (6, Custom) through (6, F) is highlighted in red.
- Row 7: Cell (7, Custom) contains "123". Cell (7, D) contains "You can veto editing this cell".
- Row 10: Cell (10, labels) contains "This default cell will overflow into neighboring cells, but not if you turn overflow off." and spans across columns "column", "labels", and "D".
- Row 12: Cell (12, labels) contains "This cell is set to span 3 rows and 3 columns" and spans across columns "column", "labels", and "D" for rows 12, 13, and 14.





# wxPython widgets

- wx.StatusBar

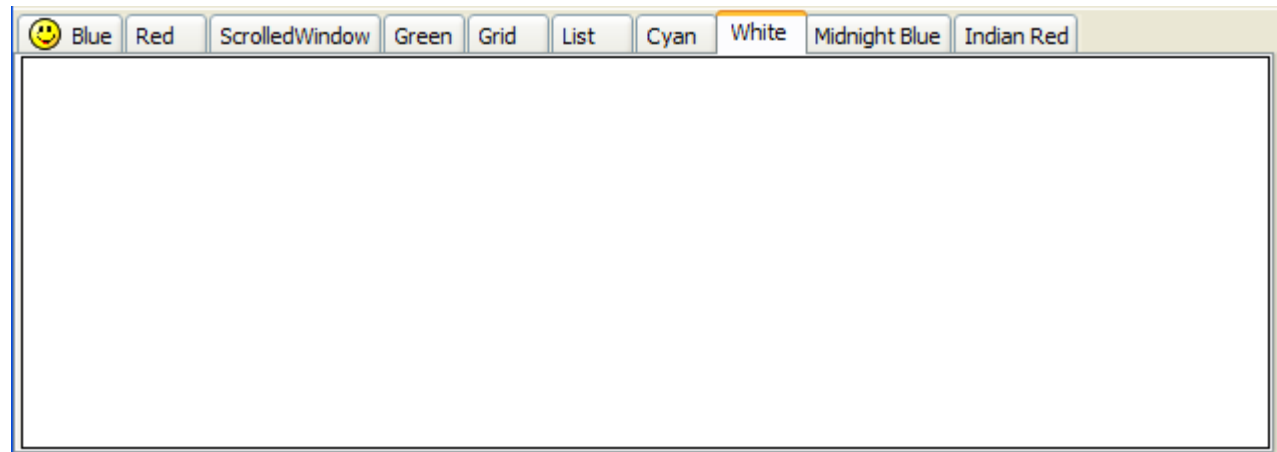
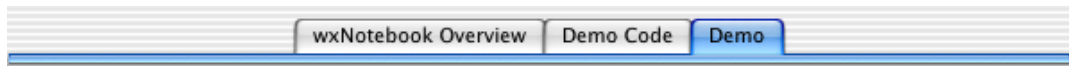


- wx.ToolBar



# wxPython widgets

- wx.Notebook
  - Manages multiple windows with tabs.
  - Tabs can be on any side of the notebook that the platform supports.



# wxPython widgets

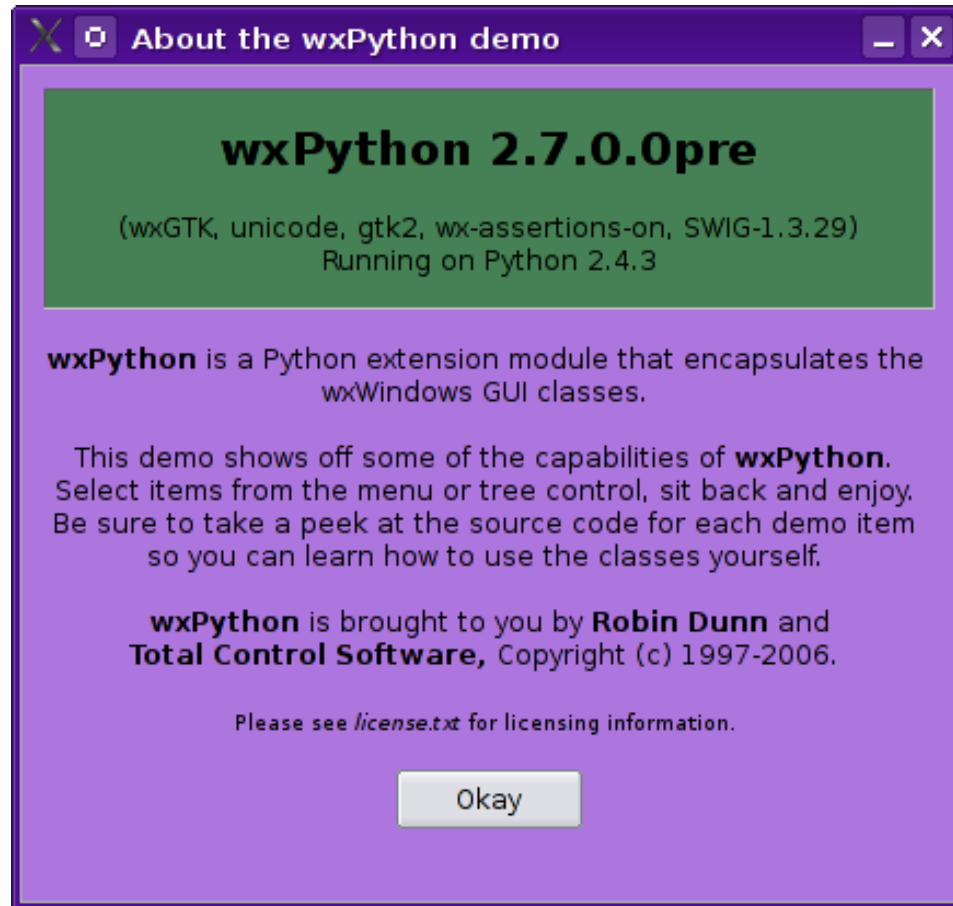
- wx.html.HtmlWindow
  - Capable of parsing and rendering most simple HTML tags.
  - Custom Tag Handlers can change or add to how HTML is rendered.

```
<wxp class="wx.Button">  
    <param name="label" value="Okay">  
    <param name="id" value="wxID_OK">  
</wxp>
```



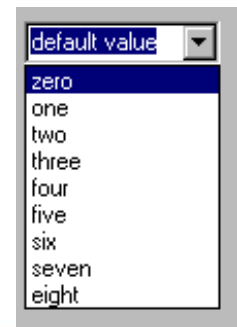
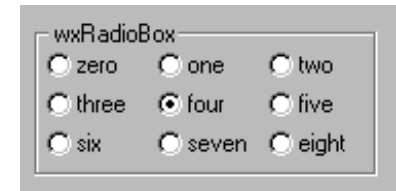
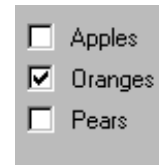
# wxPython widgets

- wx.html.HtmlWindow



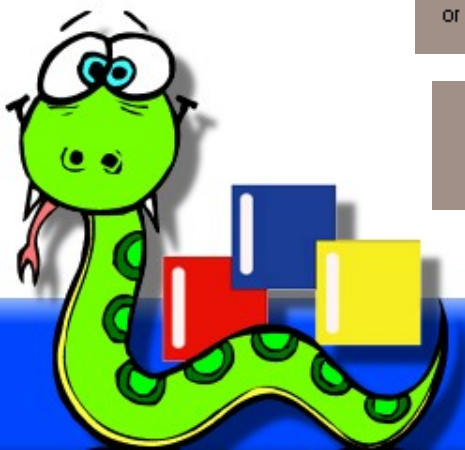
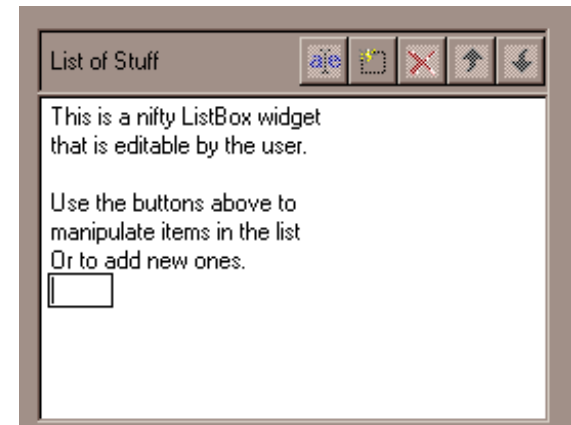
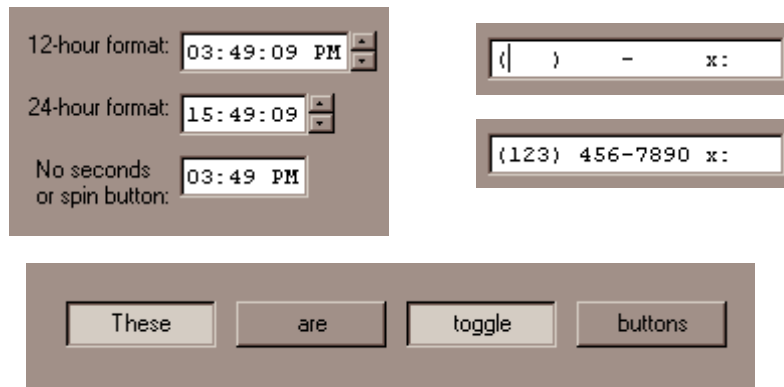
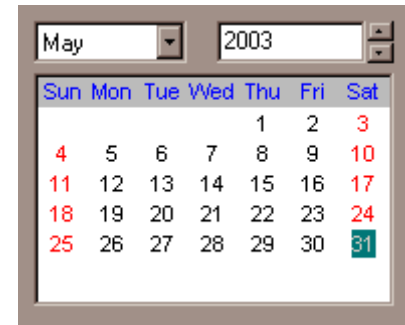
# wxPython widgets: controls

- wx.Button, wx.BitmapButton
- wx.RadioButton, wx.RadioButton
- wx.CheckBox
- wx.Choice
- wx.ComboBox
- wx.SpinButton



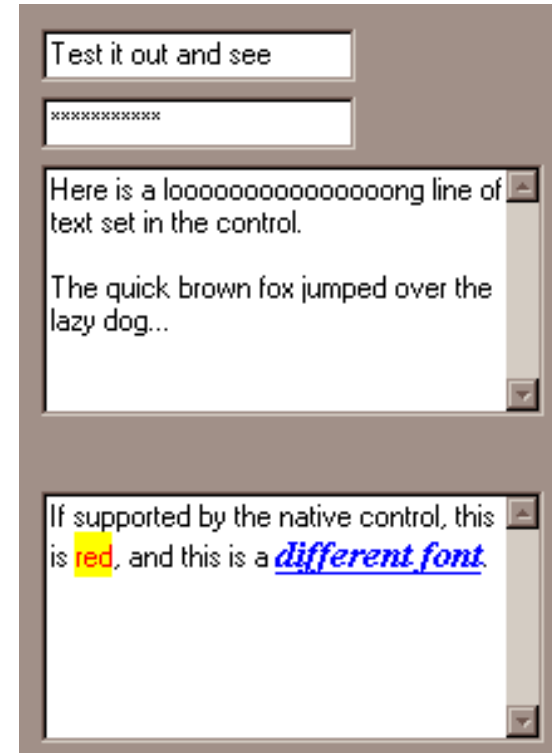
# wxPython widgets: controls

- wx.ToggleButton
- wx.gizmos.EditableListBox
- wx.lib.masked.TextCtrl
- wx.calendar.CalendarCtrl
- wx.lib.masked.TimeCtrl



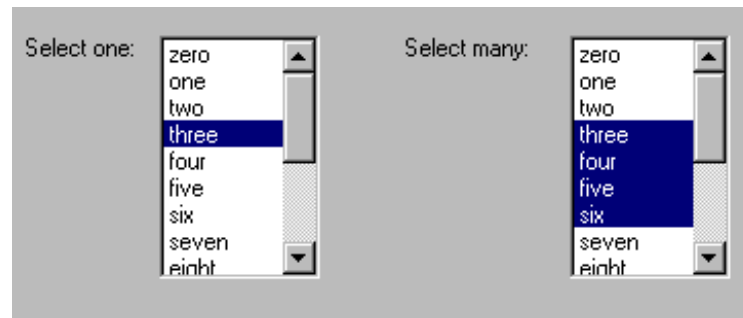
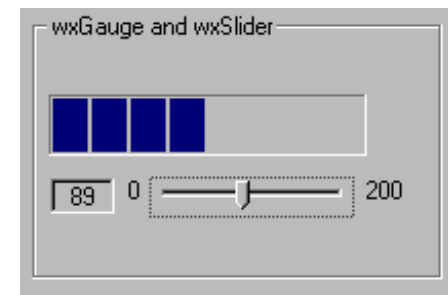
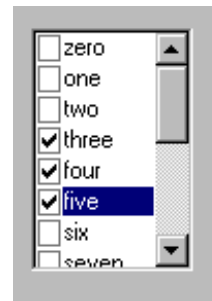
# wxPython widgets: controls

- wx.TextCtrl
  - Password masking, multi-line with or without word-wrap, simple attributes, etc.



# wxPython widgets: controls

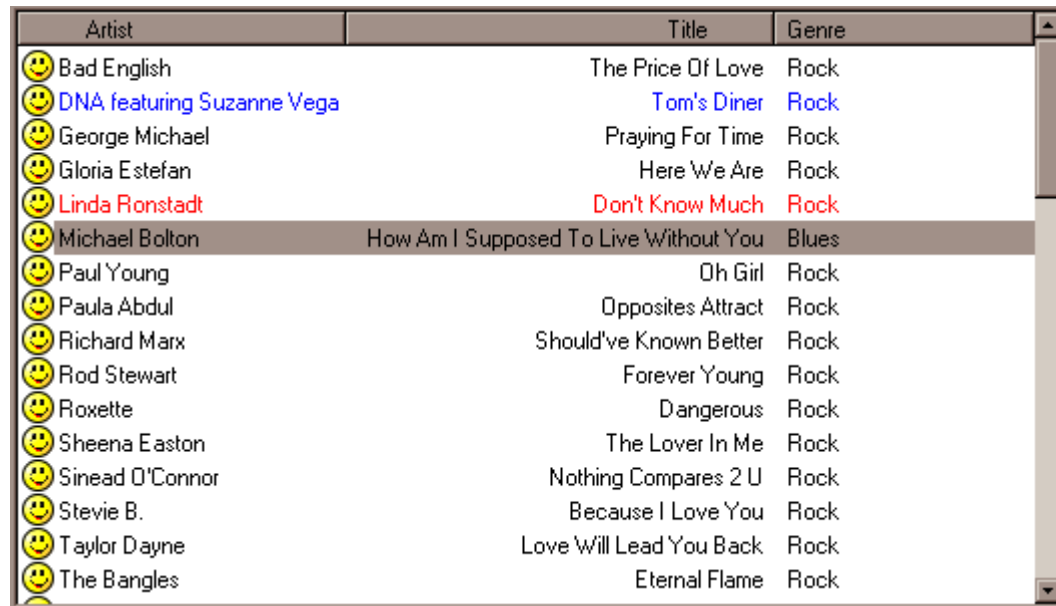
- wx.ListBox
- wx.CheckListBox
- wx.Gauge
- wx.Slider
- wx.StaticBox





# wxPython widgets : controls

- wx.ListCtrl
  - Supports list, icon, small icon, report views.
  - Virtual mode, where data items are provided by overloaded methods.

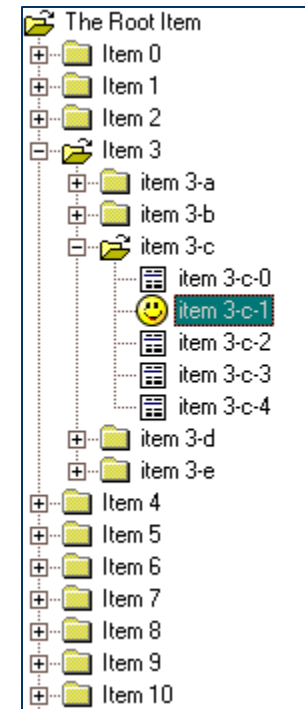


Artist	Title	Genre
Bad English	The Price Of Love	Rock
DNA featuring Suzanne Vega	Tom's Diner	Rock
George Michael	Praying For Time	Rock
Gloria Estefan	Here We Are	Rock
Linda Ronstadt	Don't Know Much	Rock
Michael Bolton	How Am I Supposed To Live Without You	Blues
Paul Young	Oh Girl	Rock
Paula Abdul	Opposites Attract	Rock
Richard Marx	Should've Known Better	Rock
Rod Stewart	Forever Young	Rock
Roxette	Dangerous	Rock
Sheena Easton	The Lover In Me	Rock
Sinead O'Connor	Nothing Compares 2 U	Rock
Stevie B.	Because I Love You	Rock
Taylor Dayne	Love Will Lead You Back	Rock
The Bangles	Eternal Flame	Rock



# wxPython widgets : controls

- wx.TreeCtrl
  - Supports images for various node states.
  - Can be virtualized by delaying the adding of child items until the parent is expanded.



# wxPython widgets : controls

- `wx.gizmos.TreeListCtrl`

Main column	Column 1	Column 2
[-] 📁 The Root Item	col 1 root	col 2 root
[+] 📁 Item 0	Item 0(c1)	Item 0(c2)
[+] 📁 Item 1	Item 1(c1)	Item 1(c2)
[+] 📁 Item 2	Item 2(c1)	Item 2(c2)
[+] 📁 Item 3	Item 3(c1)	Item 3(c2)
[-] 📁 Item 4	Item 4(c1)	Item 4(c2)
[+] 📁 Item 4-a	item 4-a(c1)	item 4-a(c2)
[-] 📁 item 4-b	item 4-b(c1)	item 4-b(c2)
📄 item 4-b-0	item 4-b-0(c1)	item 4-b-0(c2)
😊 item 4-b-1	item 4-b-1(c1)	item 4-b-1(c2)
📄 item 4-b-2	item 4-b-2(c1)	item 4-b-2(c2)
📄 item 4-b-3	item 4-b-3(c1)	item 4-b-3(c2)
📄 item 4-b-4	item 4-b-4(c1)	item 4-b-4(c2)
[+] 📁 item 4-c	item 4-c(c1)	item 4-c(c2)
[+] 📁 item 4-d	item 4-d(c1)	item 4-d(c2)
[+] 📁 item 4-e	item 4-e(c1)	item 4-e(c2)
[+] 📁 Item 5	Item 5(c1)	Item 5(c2)
[+] 📁 Item 6	Item 6(c1)	Item 6(c2)



# wxPython widgets

- wx.stc.StyledTextCtrl
  - (wx port of Scintilla)

```
5  #!/bin/env python
6  #-----
7  # Name:      Main.py
8  # Purpose:   Testing lots of stuff, controls, window types, etc.
9  #
10 # Author:    Robin Dunn
11 #
12 # Created:   A long time ago, in a galaxy far, far away...
13 # RCS-ID:    $Id: Main.py,v 1.76.2.29 2003/05/23 16:47:49 RD Exp $
14 # Copyright: (c) 1999 by Total Control Software
15 # Licence:   wxWindows license
16 #-----
17
18 import sys, os, time
19 from wxPython.wx import *
20 from wxPython.html import wxHtmlWindow
21
22 import images
23
```



# wxPython widgets

- And many others...

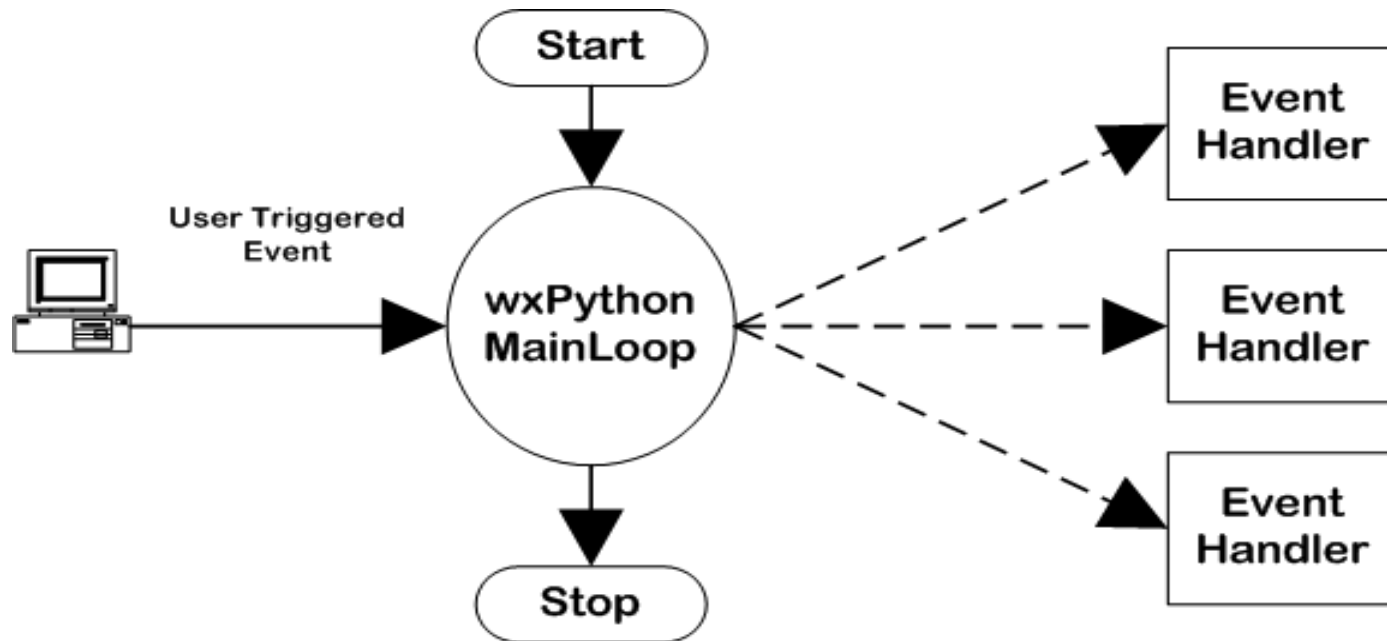


# Event Handling

- Most, if not all, GUI systems and toolkits are designed to be event driven, meaning that the main flow of your program is not sequential from beginning to end.
- When something happens that is of interest to you (an event), the system or toolkit calls a bit of your code that deals with that event (event handler).
- When your event handler finishes, control returns to the “main loop” and your program waits for the next event.
- While one event handler is running all others are blocked, so don't do things that take a “long time” to complete.

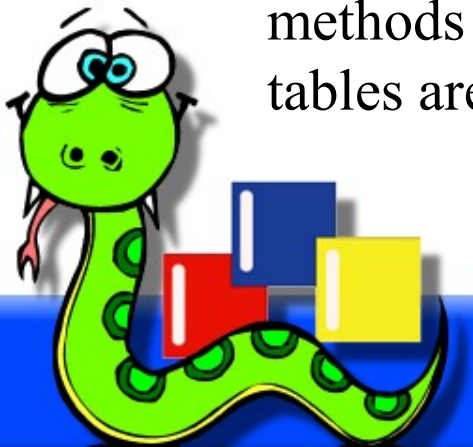


# Event handling



# Event Handling

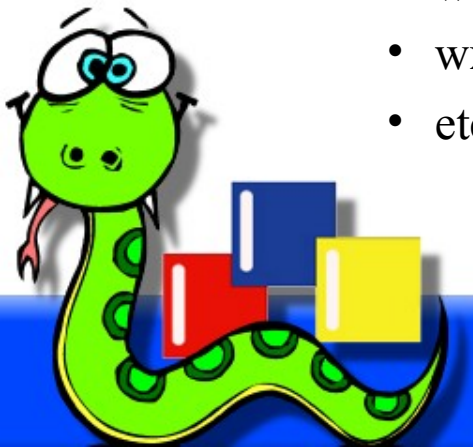
- Various event-handling models:
- **Callbacks:** Standalone functions associated with an event by calling a toolkit function. There are encapsulation problems.
- **Message based:** Messages sent to windows for controlling behavior, or for events.
- **Virtual methods:** One for each type of event. Solves encapsulation, but leads to clutter, inflexible classes, and many derived classes just to handle an event differently.
- **Static event tables:** Events are associated with classes and methods at compile time via a table. When the event occurs the tables are searched for a match and the method is invoked.





# Event Handling

- wxPython uses **Dynamic Event Tables**
  - Built at run-time.
  - Events can be “bound” to any callable object that will serve as the Event Handler:
    - any method of the class receiving the event, or other classes
    - standalone functions
    - any object with a `__call__` method
  - Handlers are connected to events with a set of binder objects:
    - `wx.EVT_MENU`
    - `wx.EVT_PAINT`
    - `wx.EVT_SIZE`
    - etc.



# Event Handling

- The connection, or *binding*, between event and handler is made with the Bind method

```
def Bind(self, event, handler, source=None,  
        id=wx.ID_ANY, id2=wx.ID_ANY)
```

```
self.Bind(wx.EVT_BUTTON, self.OnButton, theBtn)
```

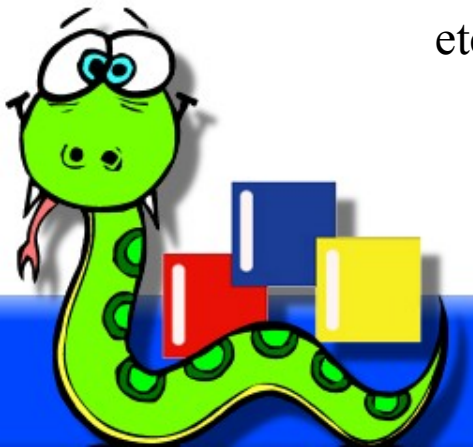
or

```
theBtn.Bind(wx.EVT_BUTTON, self.OnButton)
```

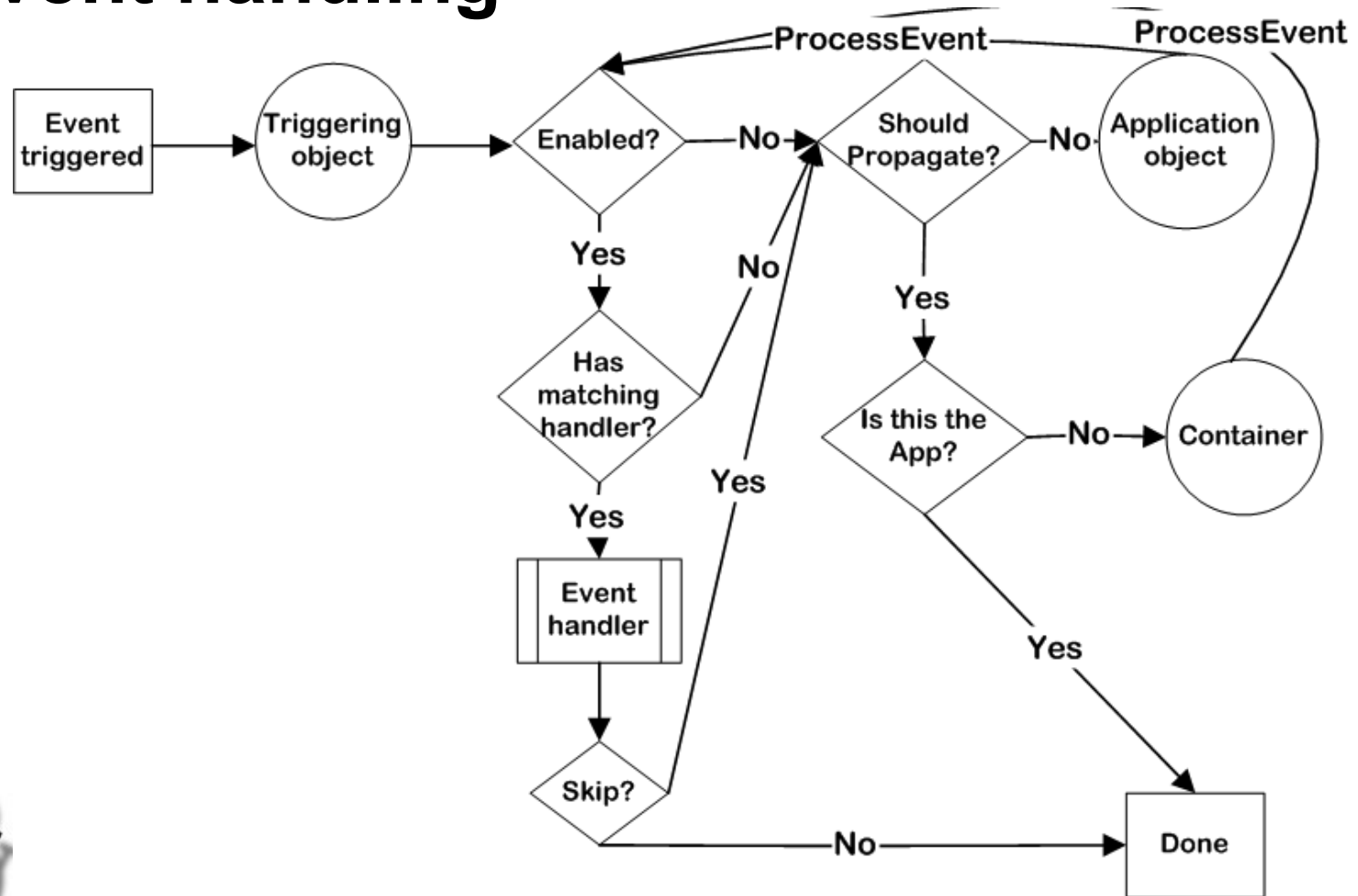


# Event Handling

- Each handler is passed an event object when called.
  - Contains information about the event
- Two classifications of event objects:
  - Classes derived from **wx.Event**
    - Events that only make sense for the window where the event took place, such as `wx.PaintEvent`, `wx.KeyEvent`, `wx.SizeEvent`, etc.
  - Classes derived from **wx.CommandEvent**
    - Events that may be of interest for any object up the “containment hierarchy,” such as `wx.MenuEvent`, `wx.NotebookEvent`, `wx.ListEvent`, etc.

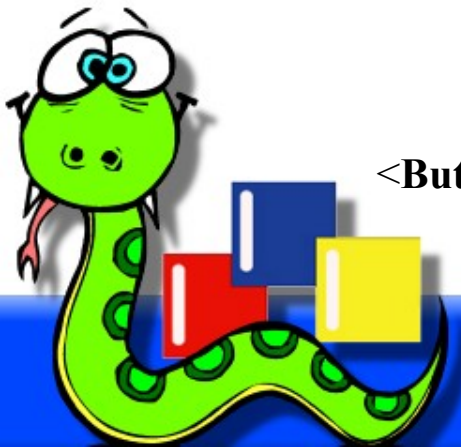
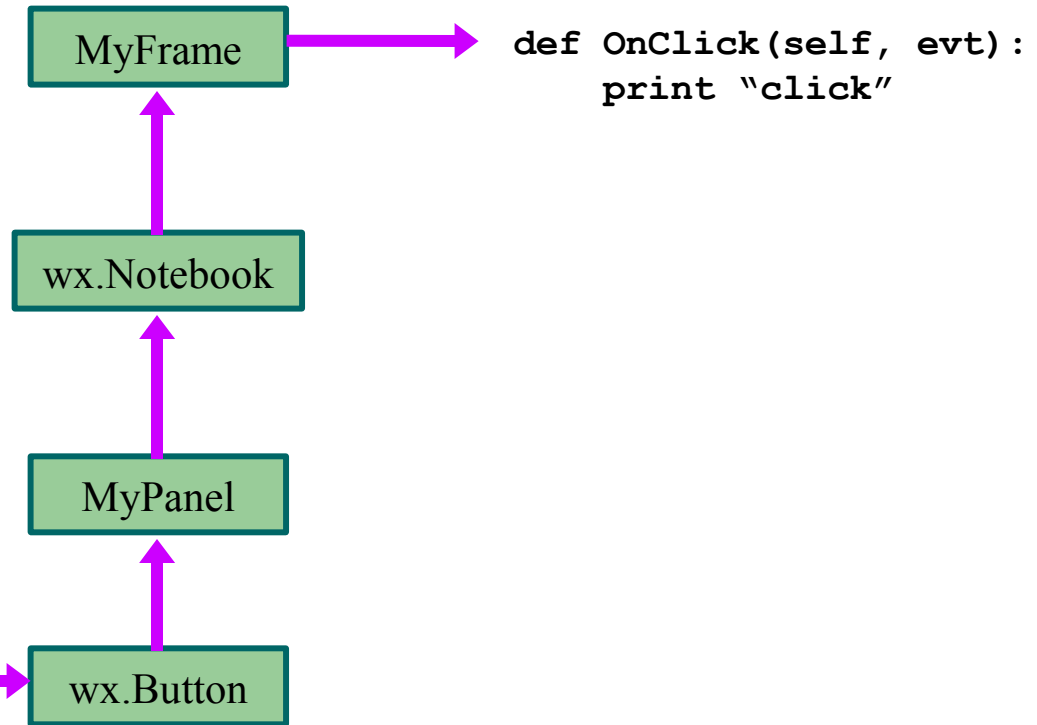


# Event handling



# In search of Event Handlers...

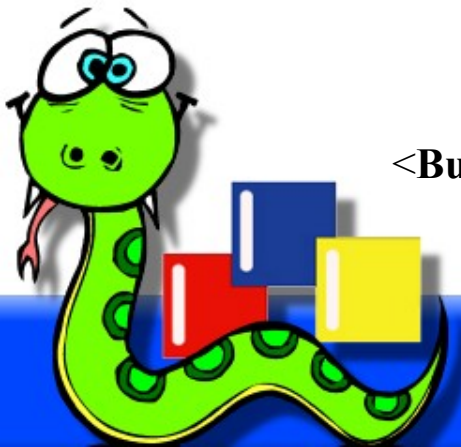
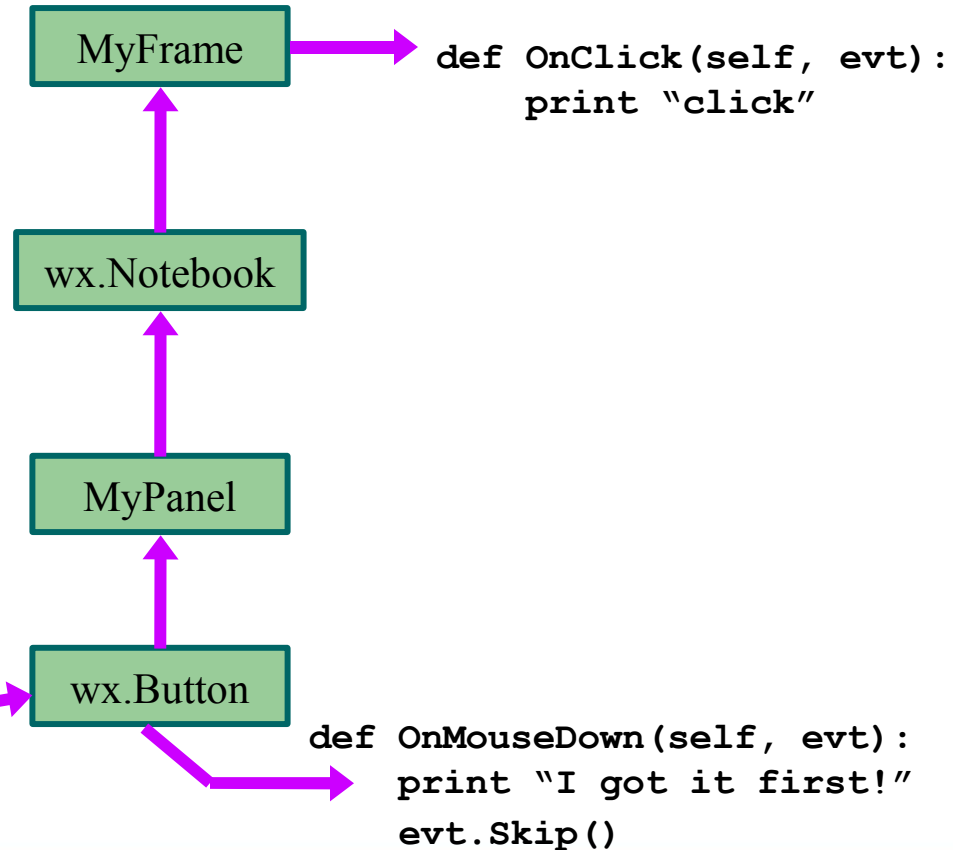
```
self.Bind(wx.EVT_BUTTON,  
          self.OnClick,  
          self.button)
```



# In search of Event Handlers...

```
self.Bind(wx.EVT_BUTTON,  
          self.OnClick,  
          self.button)
```

```
self.button.Bind(  
    wx.EVT_LEFT_DOWN,  
    self.OnMouseDown)
```

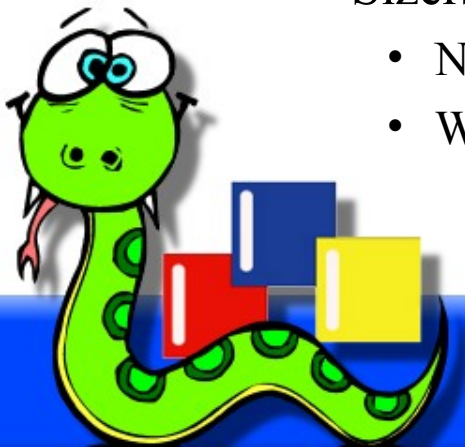


# Code break...



# Organizing your layout

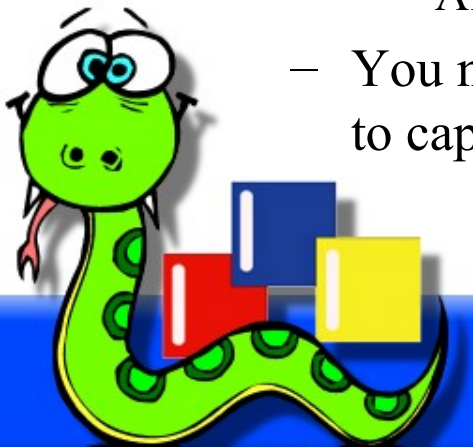
- There are various ways to do layout:
  - Brute force
    - All widgets are positioned and sized pixel by pixel.
    - Has to be redone in every EVT\_SIZE event.
    - Painful, cross-platform issues.
  - Layout Constraints
    - Powerful, but complex and verbose.
    - Deals with the relationships between widgets.
    - See the docs and demo for more details.
  - Sizers
    - Not as flexible or complex, but powerful enough.
    - Worth the pain.



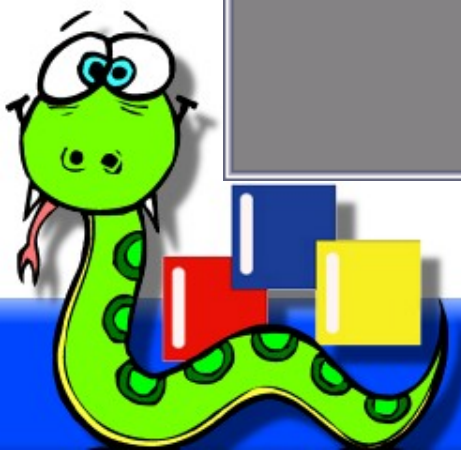
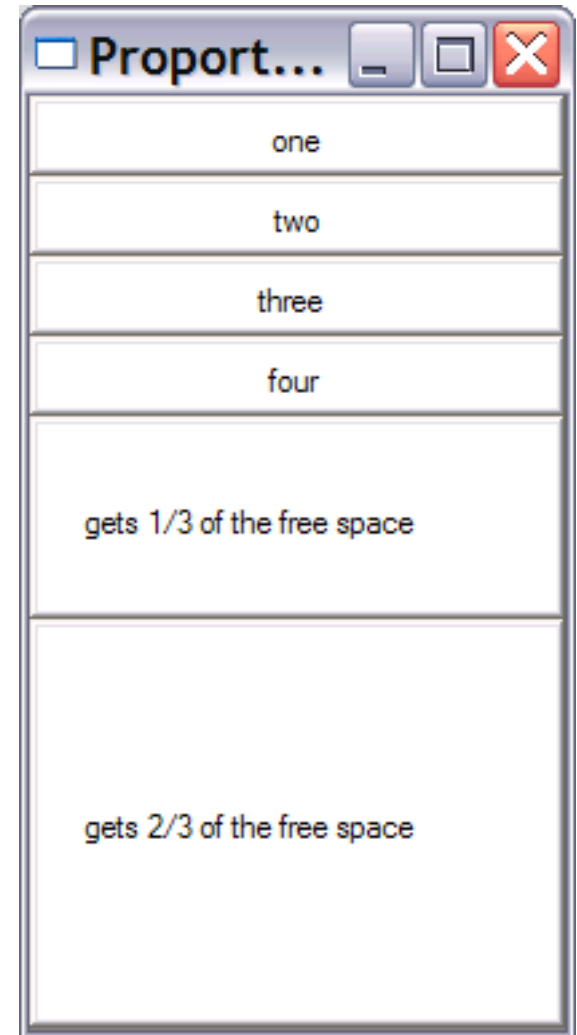
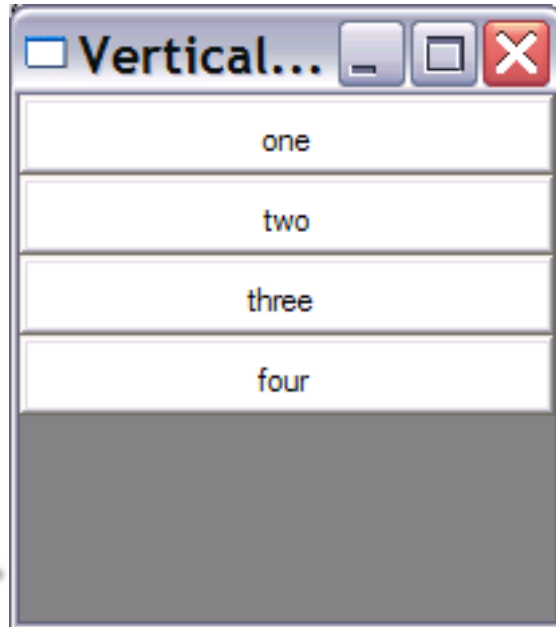


# Organizing your layout

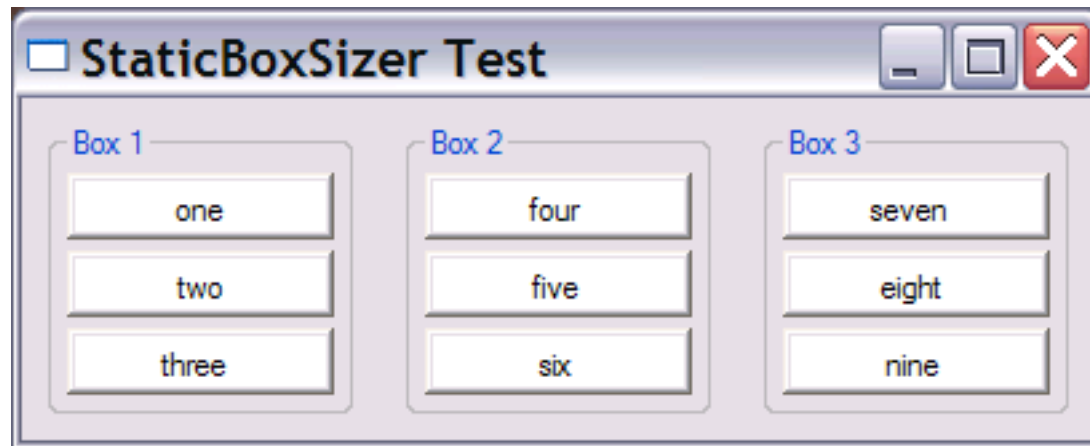
- Sizers
  - Similar to LayoutManagers in Java.
  - Not as flexible as LayoutConstraints, but much simpler, once you get over the hump.
  - Relationships defined by containment within sizers or nested sizers.
  - All items (widgets or nested sizers) added to a Sizer are laid out by a specific algorithm determined by the class of sizer.
  - An item's position within its allotted space is also controllable.
    - Empty space on borders
    - Alignment
  - You need to be able to think visually both *top-down* and *bottom-up* to capture your design



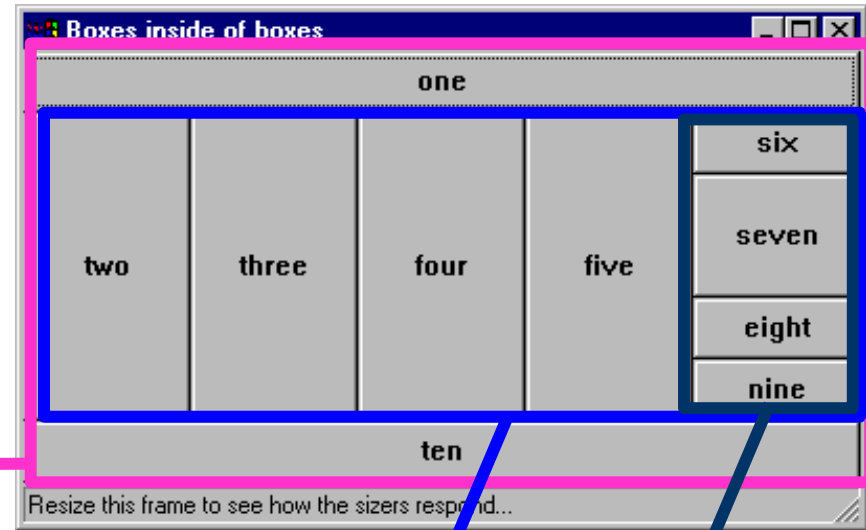
# wx.BoxSizer



# wx.StaticBoxSizer



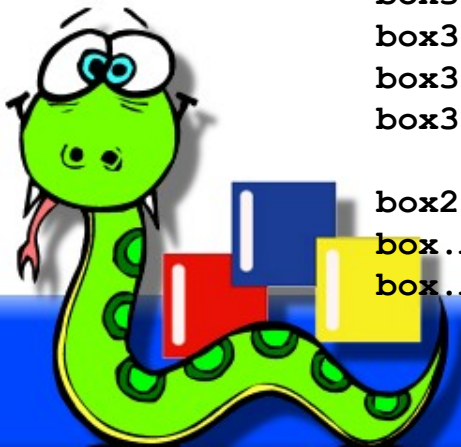
# wx.BoxSizer



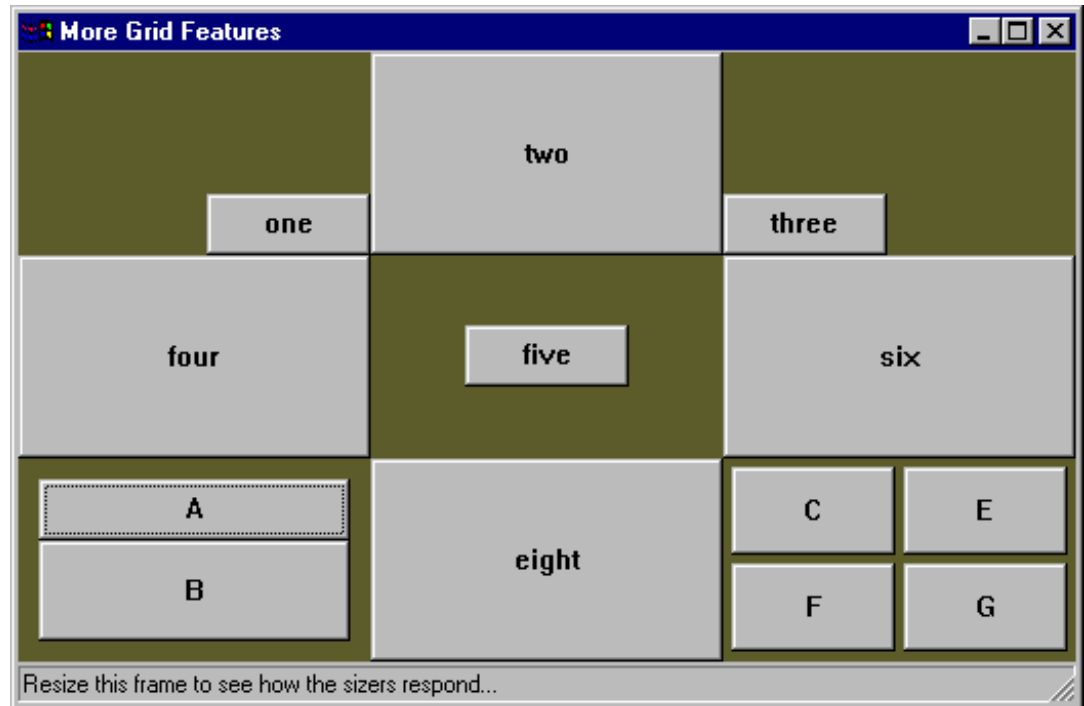
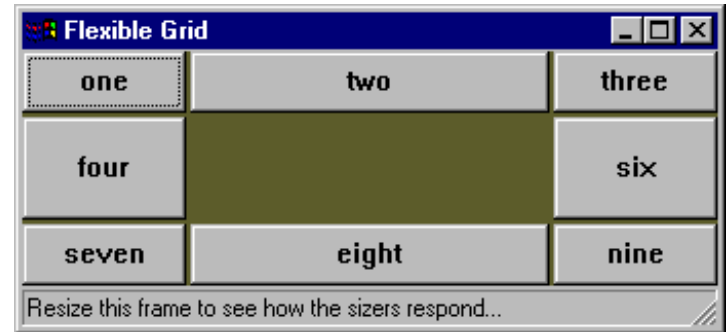
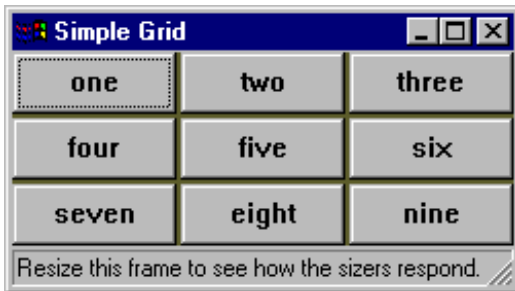
```
box = wx.BoxSizer(wx.VERTICAL)
box.Add(wx.Button(win, -1, "one"), 0, wx.EXPAND)
box2 = wx.BoxSizer(wx.HORIZONTAL)
box2.Add(wx.Button(win, -1, "two"), 0, wx.EXPAND)
box2.Add(wx.Button(win, -1, "three"), 0, wx.EXPAND)
box2.Add(wx.Button(win, -1, "four"), 0, wx.EXPAND)
box2.Add(wx.Button(win, -1, "five"), 0, wx.EXPAND)

box3 = wx.BoxSizer(wx.VERTICAL)
box3.Add(wx.Button(win, -1, "six"), 0, wx.EXPAND)
box3.Add(wx.Button(win, -1, "seven"), 2, wx.EXPAND)
box3.Add(wx.Button(win, -1, "eight"), 1, wx.EXPAND)
box3.Add(wx.Button(win, -1, "nine"), 1, wx.EXPAND)

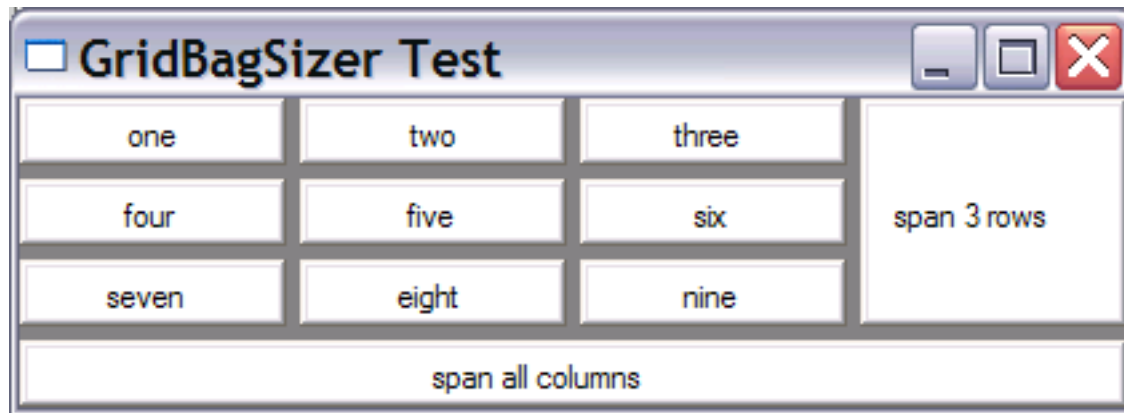
box2.Add(box3, 1, wx.EXPAND)
box.Add(box2, 1, wx.EXPAND)
box.Add(wx.Button(win, -1, "ten"), 0, wx.EXPAND)
```



# wx.GridSizer

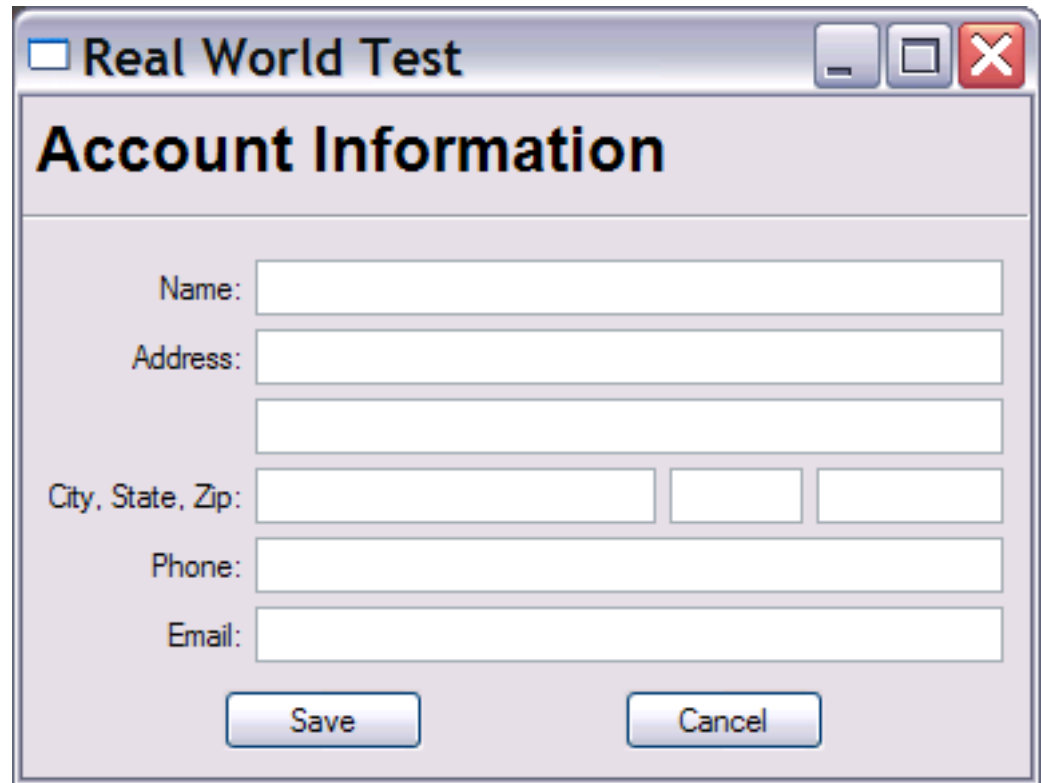


# wx.GridBagSizer



# Sizers in the Real World

- Can you see how to get here from there?



Real World Test

## Account Information

Name:

Address:

City, State, Zip:

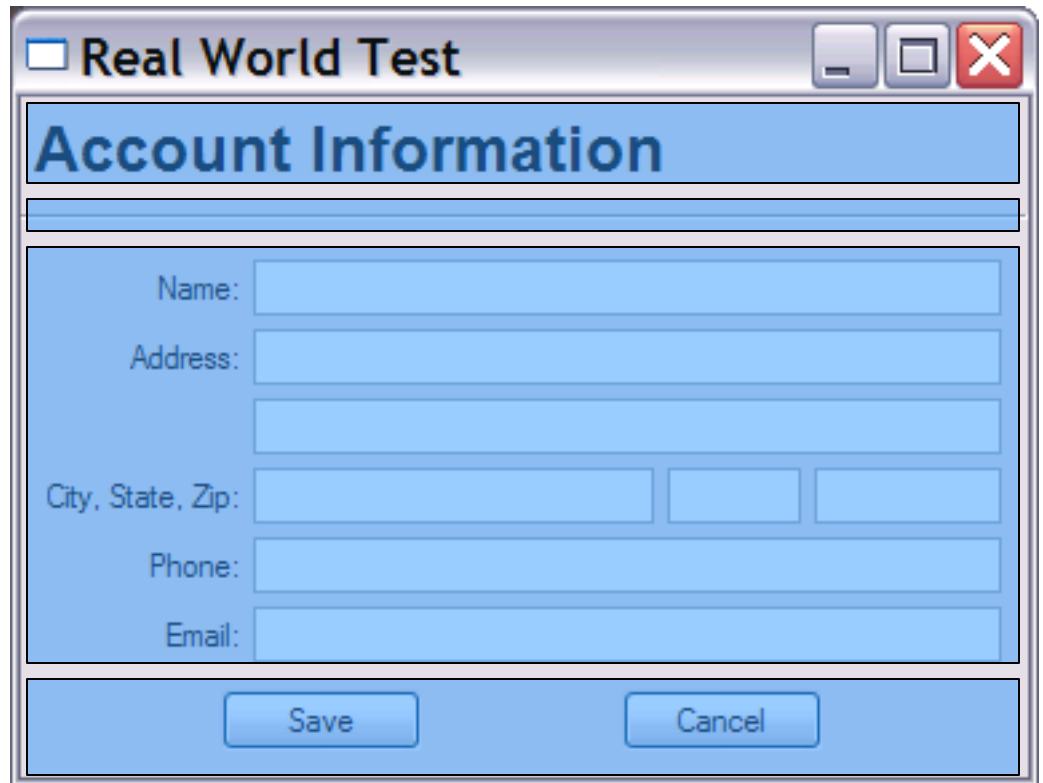
Phone:

Email:



# Sizers in the Real World

Vertical Box Sizer



The screenshot shows a window titled "Real World Test" with a standard Mac OS-style title bar (minimize, maximize, close buttons). The window content is titled "Account Information" in a blue header bar. Below the header is a form with the following fields:

- Name: [text input]
- Address: [text input]
- [text input]
- City, State, Zip: [text input] [text input] [text input]
- Phone: [text input]
- Email: [text input]

At the bottom of the form are two buttons: "Save" and "Cancel".

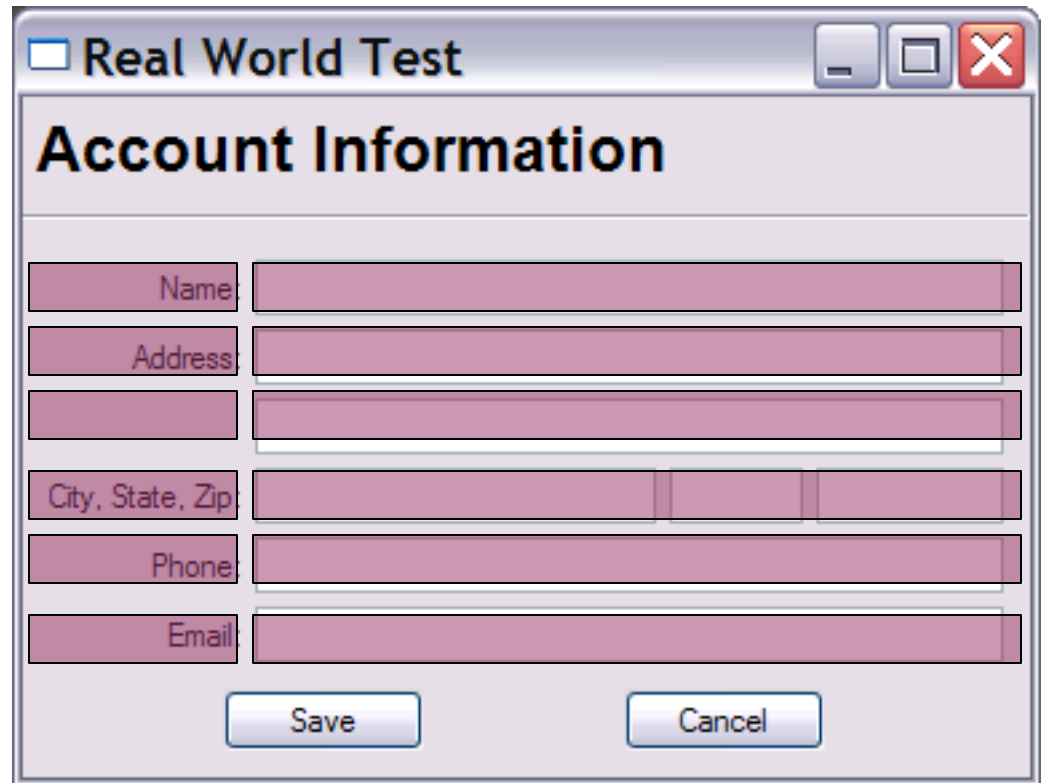




# Sizers in the Real World

Vertical Box Sizer

Flex Grid Sizer



The screenshot shows a window titled "Real World Test" with a close button. Inside the window is a form titled "Account Information". The form contains several input fields: "Name", "Address", "City, State, Zip" (with three sub-fields), "Phone", and "Email". At the bottom of the form are "Save" and "Cancel" buttons.

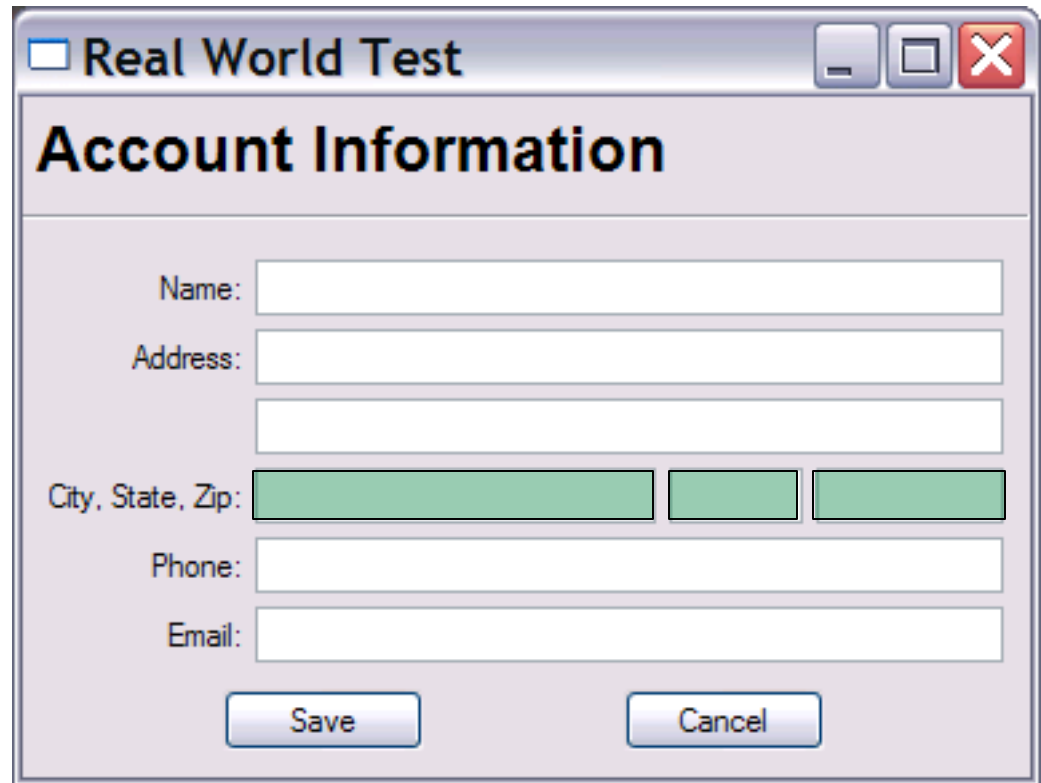


# Sizers in the Real World

Vertical BoxSizer

Flex GridSizer

Horizontal BoxSizer



The screenshot shows a window titled "Real World Test" with a close button. The window contains a form titled "Account Information". The form has the following fields:

- Name:
- Address:
- City, State, Zip:
- Phone:
- Email:

At the bottom of the form are two buttons: "Save" and "Cancel".



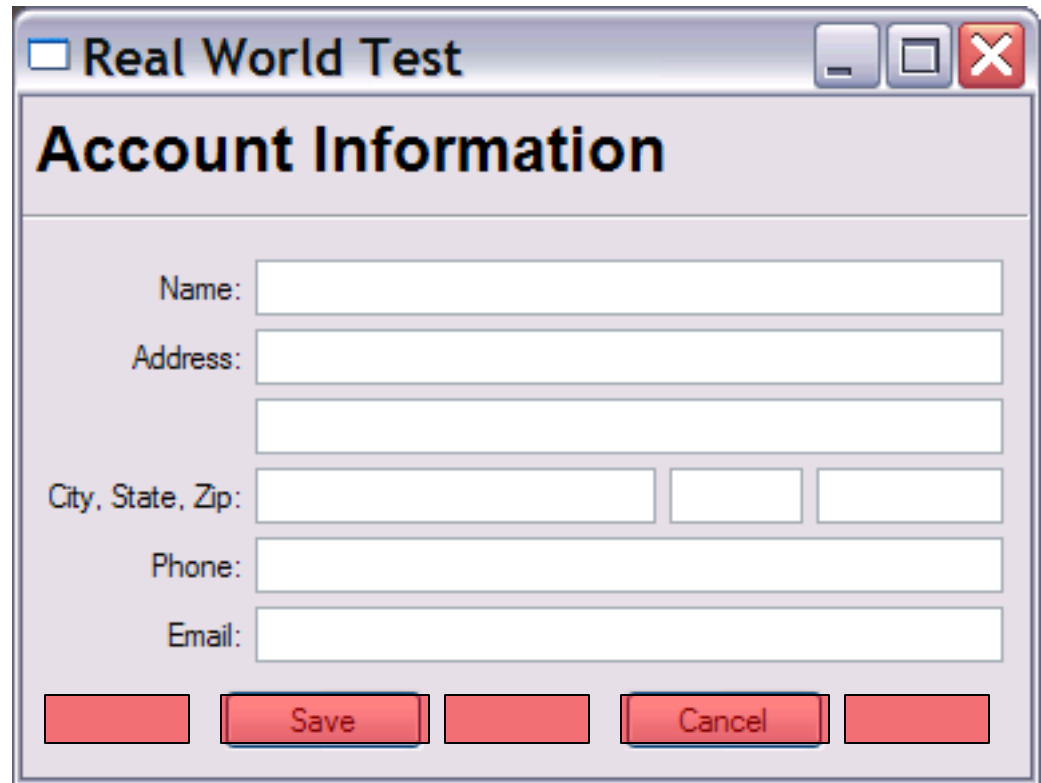
# Sizers in the Real World

Vertical BoxSizer

Flex GridSizer

Horizontal BoxSizer

Horizontal BoxSizer



The screenshot shows a window titled "Real World Test" with standard window controls (minimize, maximize, close). The window contains a form titled "Account Information". The form has the following fields:

- Name:
- Address:
- 
- City, State, Zip:
- Phone:
- Email:

At the bottom of the form are four buttons: a red button, a "Save" button, another red button, and a "Cancel" button.



# Code break...



# Drawing

- A `wx.DC` is a *device context* onto which graphics and text can be drawn.
- Represents a number of output devices in a generic way:
  - windows
  - printers
  - bitmaps
  - the whole screen
- The same code may be used to draw on different devices.



# Drawing

- DC's have many drawing primitives:
  - DrawArc, DrawBitmap, DrawEllipse, DrawLine, DrawLines, DrawPoint, DrawPolygon, DrawRectangle, DrawRoundedRectangle, DrawSpline, DrawText
- And work with GDI objects:
  - wx.Font, wx.Bitmap, wx.Brush, wx.Pen, wx.Mask, wx.Icon, etc.

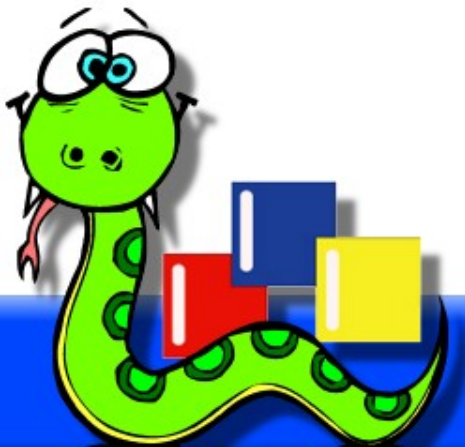


# Code break...



# Tools: PyCrust

- Interactive Python Shell
- 100% Python
- Part of wxPython
- Standalone App
- Embeddable Components

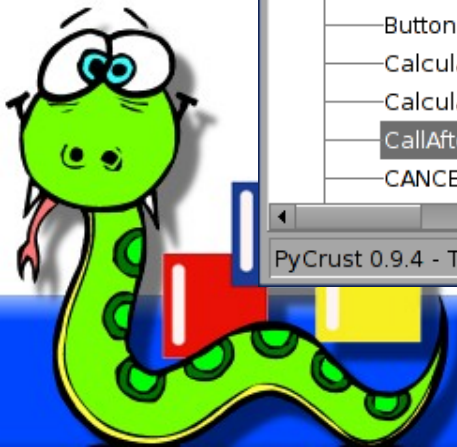




```
PyCrust
File Edit Options Help
1 PyCrust 0.9.4 - The Flakiest Python Shell
2 Sponsored by Orbtech - Your source for Python programming expertise.
3 Python 2.3 (#2, Aug 31 2003, 17:27:29)
4 [GCC 3.3.1 (Mandrake Linux 9.2 3.3.1-1mdk)] on linux2
5 Type "help", "copyright", "credits" or "license" for more information.
6 >>> import wx
7 >>> f = wx.Frame(None, -1, "Hello World")
8 >>> p = wx.Panel(f)
9 >>> b = wx.Button(p, -1, "Click me", (10,10))
10 >>> f.Show(
    Show(bool show=True) -> bool

    Shows or hides the window. You may need to call Raise for a top level
    window if you want to bring it to top, although this is not needed if
    Show is called immediately after the frame creation. Returns True if
    the window has been shown or hidden or False if nothing was done
    because it already was in the requested state.
)

Namespace Display Calltip Session Dispatcher
- BusyInfoPtr
- Button
- Button_GetDefaultS
- ButtonNameStr
- ButtonPtr
- CalculateLayoutEve
- CalculateLayoutEve
- CallAfter
- CANCEL
wx.CallAfter
Type: <type 'function'>
Value: <function CallAfter at 0x413bc48c>
Docstring:
"""Call the specified function after the current and pending event
handlers have been completed. This is also good for making GUI
method calls from non-GUI threads."""
PyCrust 0.9.4 - The Flakiest Python Shell, Sponsored by Orbtech - Your source for Python programming expertise.
```

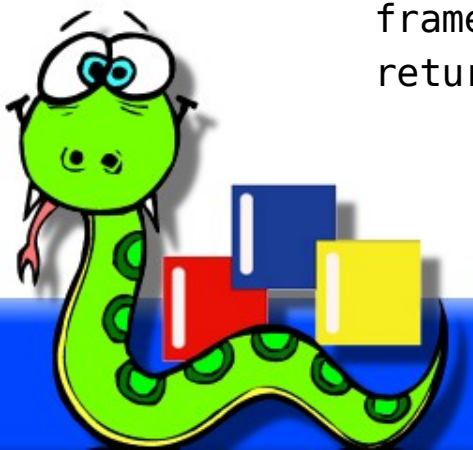


# Tools: Embedding PyCrust components

```
# ex04.py
import wx
import wx.py
...
class MyApp(wx.App):
    def OnInit(self):
        frame = MyFrame()
        frame.Show()

        shell = wx.py.shell.ShellFrame(
            frame, locals={ 'wx': wx, 'frame': frame})
        shell.Show()

        frame.Raise()
        return True
```



# PyCrust demo...



# Tools: XRCed

- XRC is an XML based resource format
  - Used for specifying the content and layout of
    - Panels
    - Frames
    - Dialogs
    - Menus
    - Toolbars
  - Can be dynamically loaded at runtime, creating all the specified widgets





# Other tools

- wxDesigner
- wxGlade
- Boa Constructor
- DialogBlocks
- SPE
- WingIDE
- Dabo
- And many others...



# Questions?



# Other resources

- wxPython website:
- wxPyWiki:
- Mailing lists:
- wxWidgets website:
- *wxPython in Action*

<http://wxPython.org>

<http://wiki.wxPython.org>

wxPython-users, wx-users

<http://wxWidgets.org>

