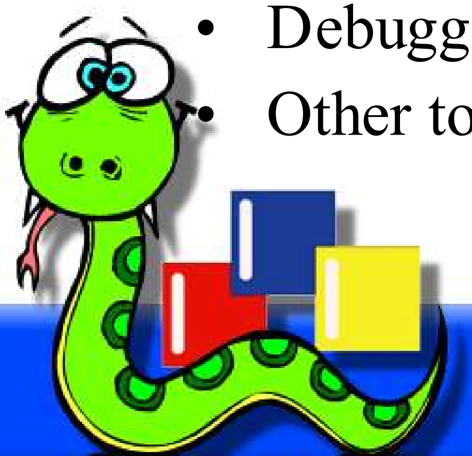
Up & Running with wxPython

Robin Dunn

O'Reilly Open Source Convention
July 26–30, 2004

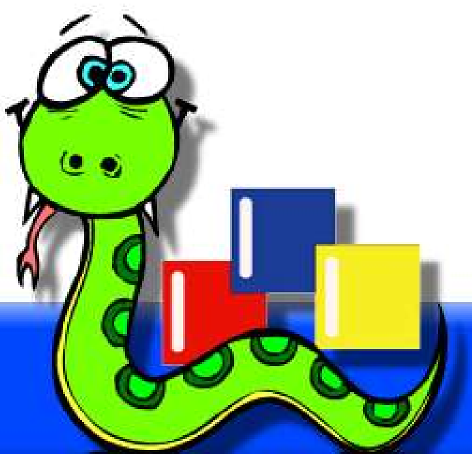
Presentation overview

- Introduction to wxPython
- Getting started
- Application fundamentals
- Widgets galore
- Event handling
- Organizing your layout
- Drawing
- Drag and drop
- Debugging with PyCrust
- Other tools

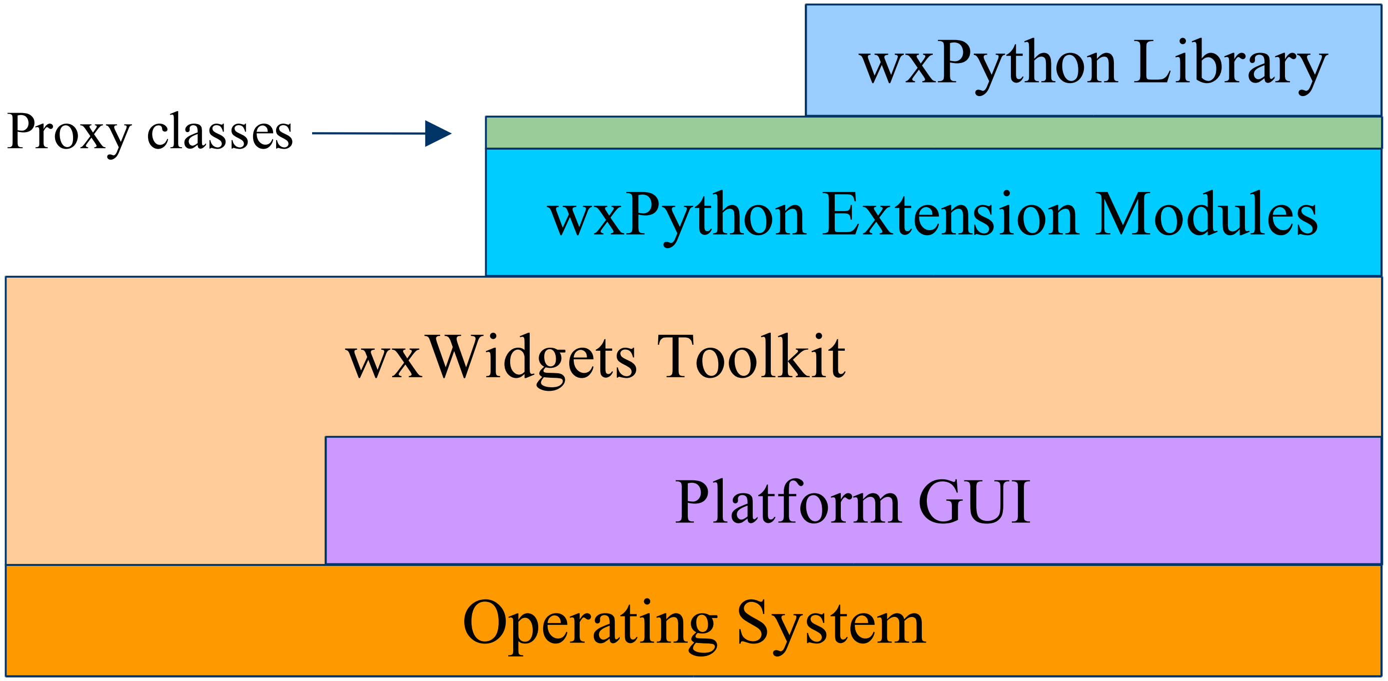


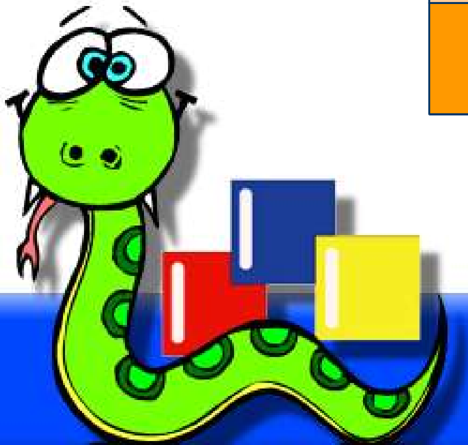
Introduction to wxPython

- wxPython is a GUI toolkit for Python, built upon the wxWidgets C++ toolkit.
  - Cross platform: Windows, Linux, Unix, OS X.
  - Uses native widgets/controls, plus many platform independent widgets.
- Mature, well established projects.
  - wxWidgets: 1992
  - wxPython: 1996

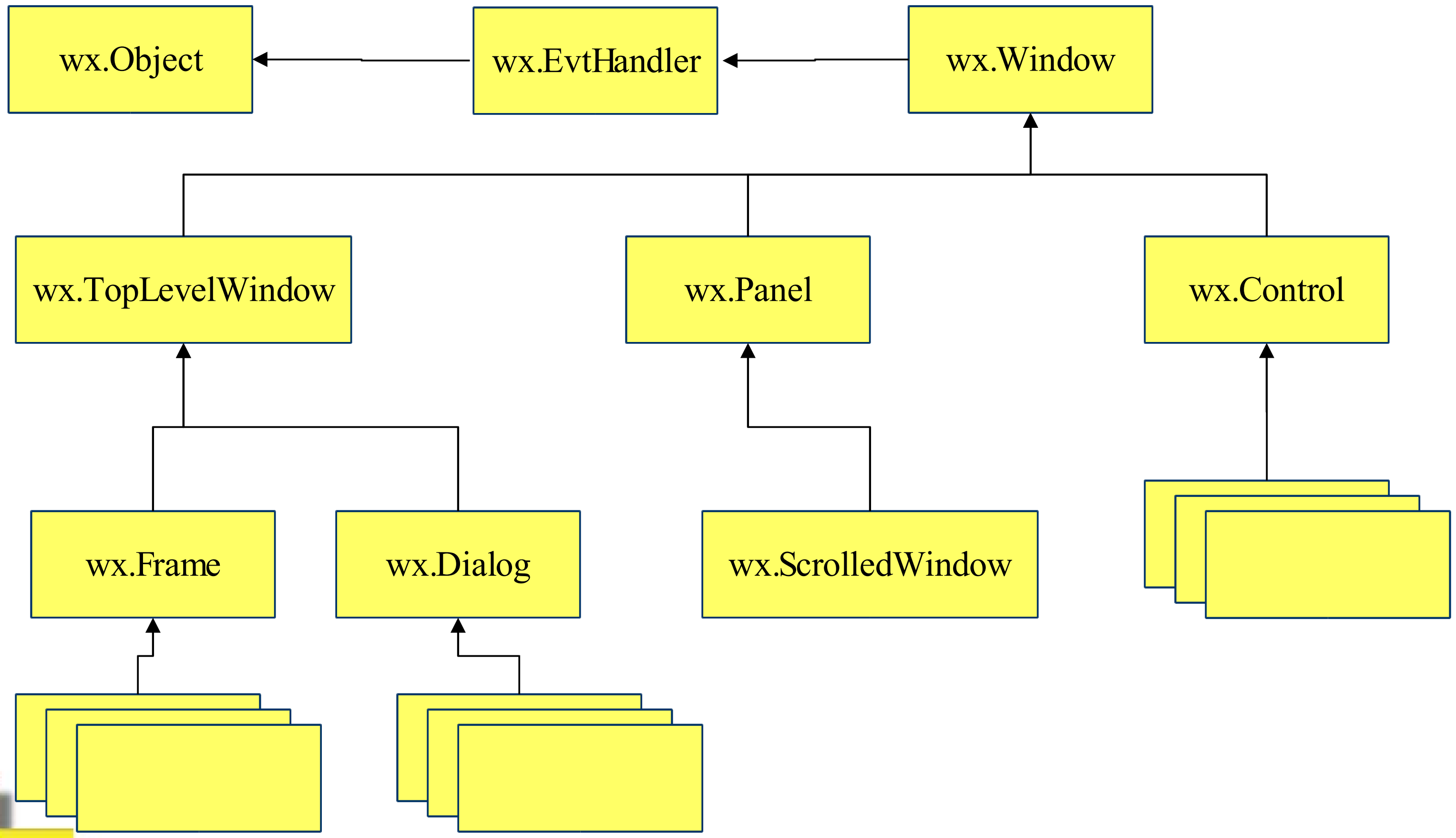


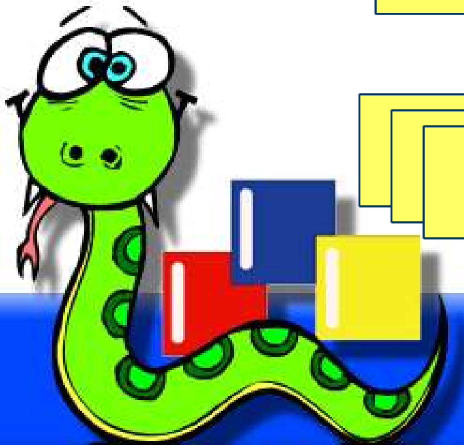
Introduction: architecture





Introduction: partial class hierarchy



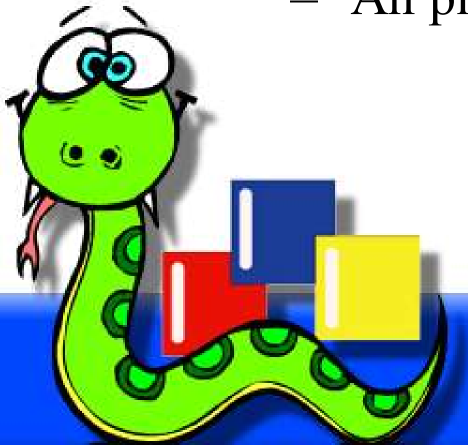


Getting started with wxPython

- Installation is simple -- binary installers are available at SourceForge and via http://wxPython.org/download.php for:
  - Windows: *.exe
  - Linux: *.rpm (and *.deb's are available separately.)
  - OS X: *.dmg, a disk image that contains an Installer package.
- Can be built from source for other Unix-like systems.

Getting started with wxPython

- Choose an installer.
- Which version of Python do you use?
  - 2.2, or 2.3
- Unicode?
  - Windows, but be careful with Win9x/ME
  - Linux/Unix, with the GTK2 build
  - OS X, soon
- or ANSI?
  - All platforms

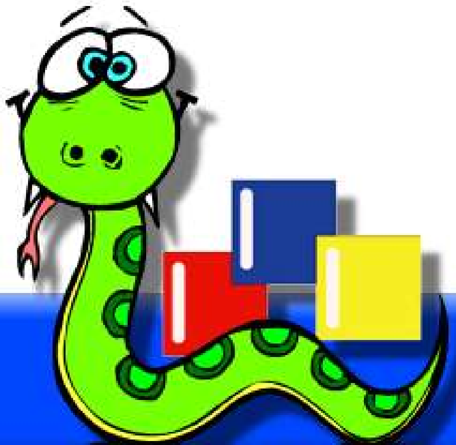


Getting started with wxPython

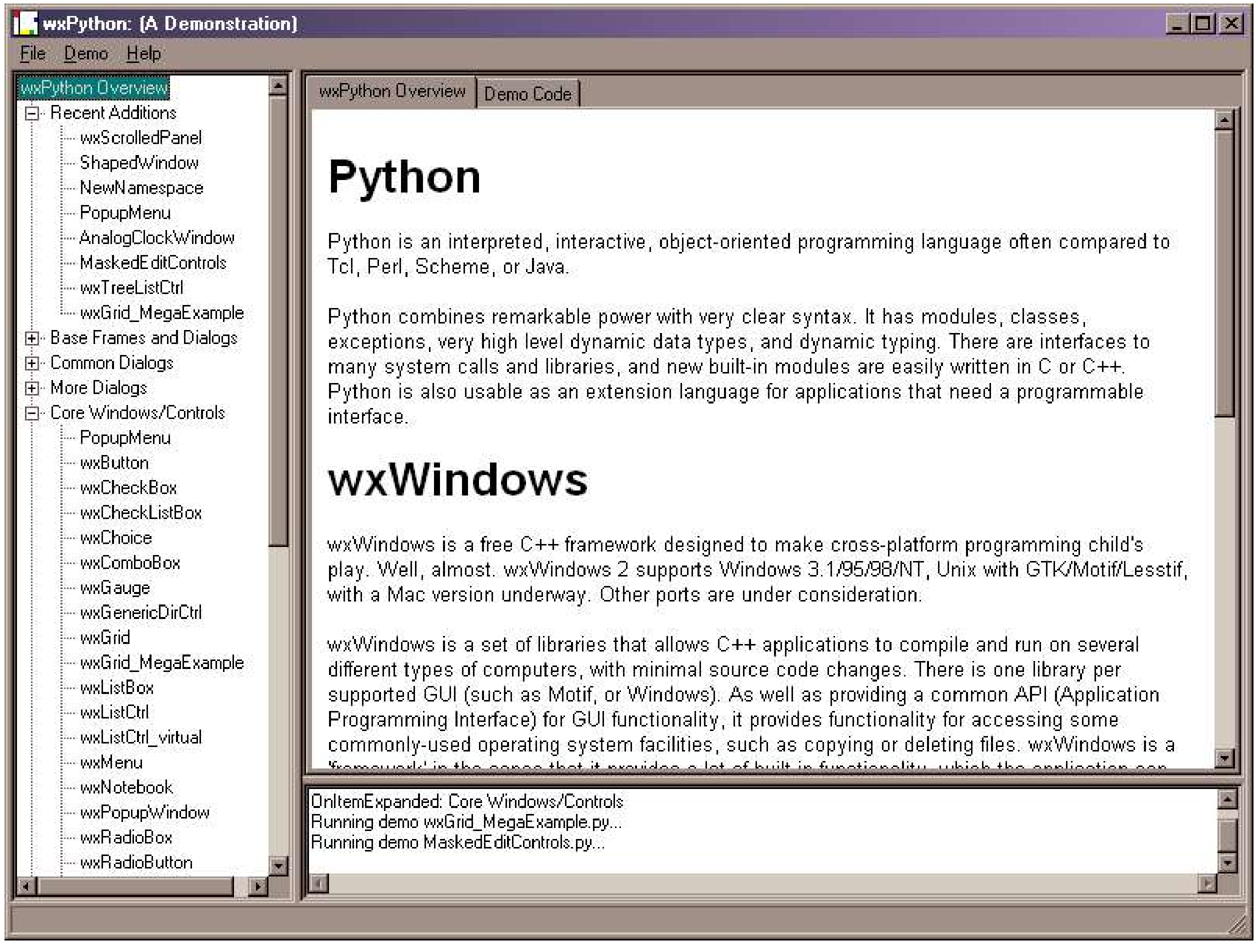
- Choose an editor or development environment:
  - Boa Constructor
  - WingIDE
  - PyAlaMode
  - SCiTE
  - Emacs, vi, etc.
- It’s just plain text, so an ordinary editor and command line will do.

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



Python

Python is an interpreted, interactive, object-oriented programming language often compared to Tcl, Perl, Scheme, or Java.

Python combines remarkable power with very clear syntax. It has modules, classes, exceptions, very high level dynamic data types, and dynamic typing. There are interfaces to many system calls and libraries, and new built-in modules are easily written in C or C++. Python is also usable as an extension language for applications that need a programmable interface.

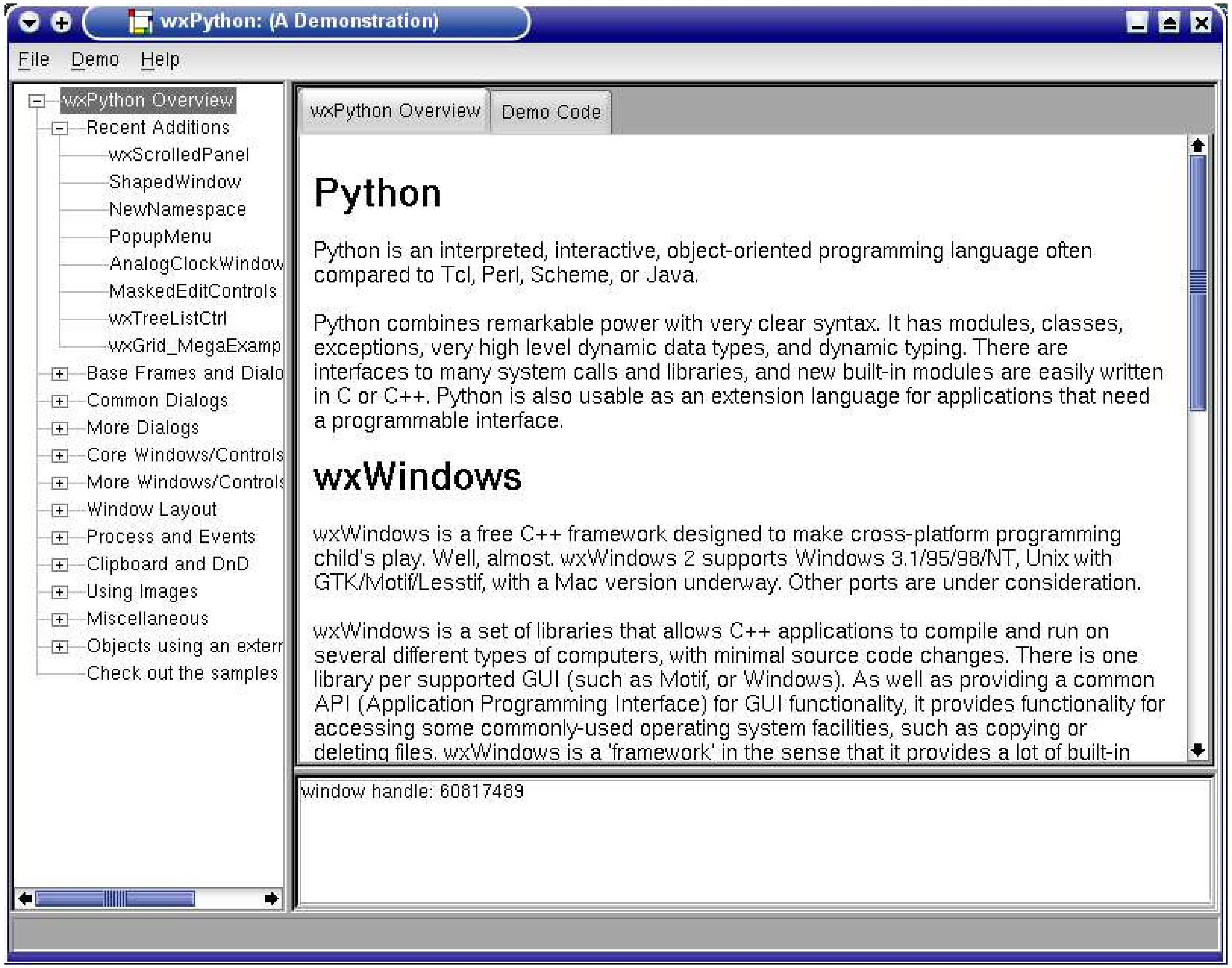
wxWindows

wxWindows is a free C++ framework designed to make cross-platform programming child's play. Well, almost. wxWindows 2 supports Windows 3.1/95/98/NT, Unix with GTK/Motif/Lesstif, with a Mac version underway. Other ports are under consideration.

wxWindows is a set of libraries that allows C++ applications to compile and run on several different types of computers, with minimal source code changes. There is one library per supported GUI (such as Motif, or Windows). As well as providing a common API (Application Programming Interface) for GUI functionality, it provides functionality for accessing some commonly-used operating system facilities, such as copying or deleting files. wxWindows is a

```
OnItemExpanded: Core Windows/Controls
Running demo wxGrid_MegaExample.py...
Running demo MaskedEditControls.py...
```

Getting started with wxPython



Python

Python is an interpreted, interactive, object-oriented programming language often compared to Tcl, Perl, Scheme, or Java.

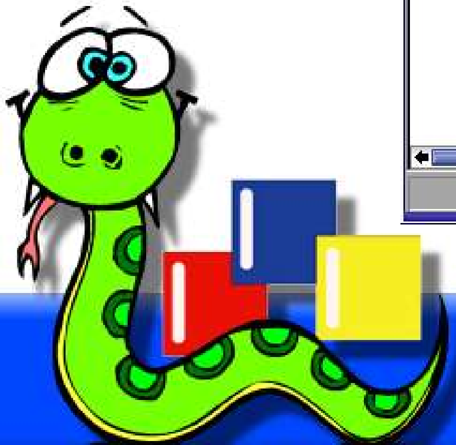
Python combines remarkable power with very clear syntax. It has modules, classes, exceptions, very high level dynamic data types, and dynamic typing. There are interfaces to many system calls and libraries, and new built-in modules are easily written in C or C++. Python is also usable as an extension language for applications that need a programmable interface.

wxWindows

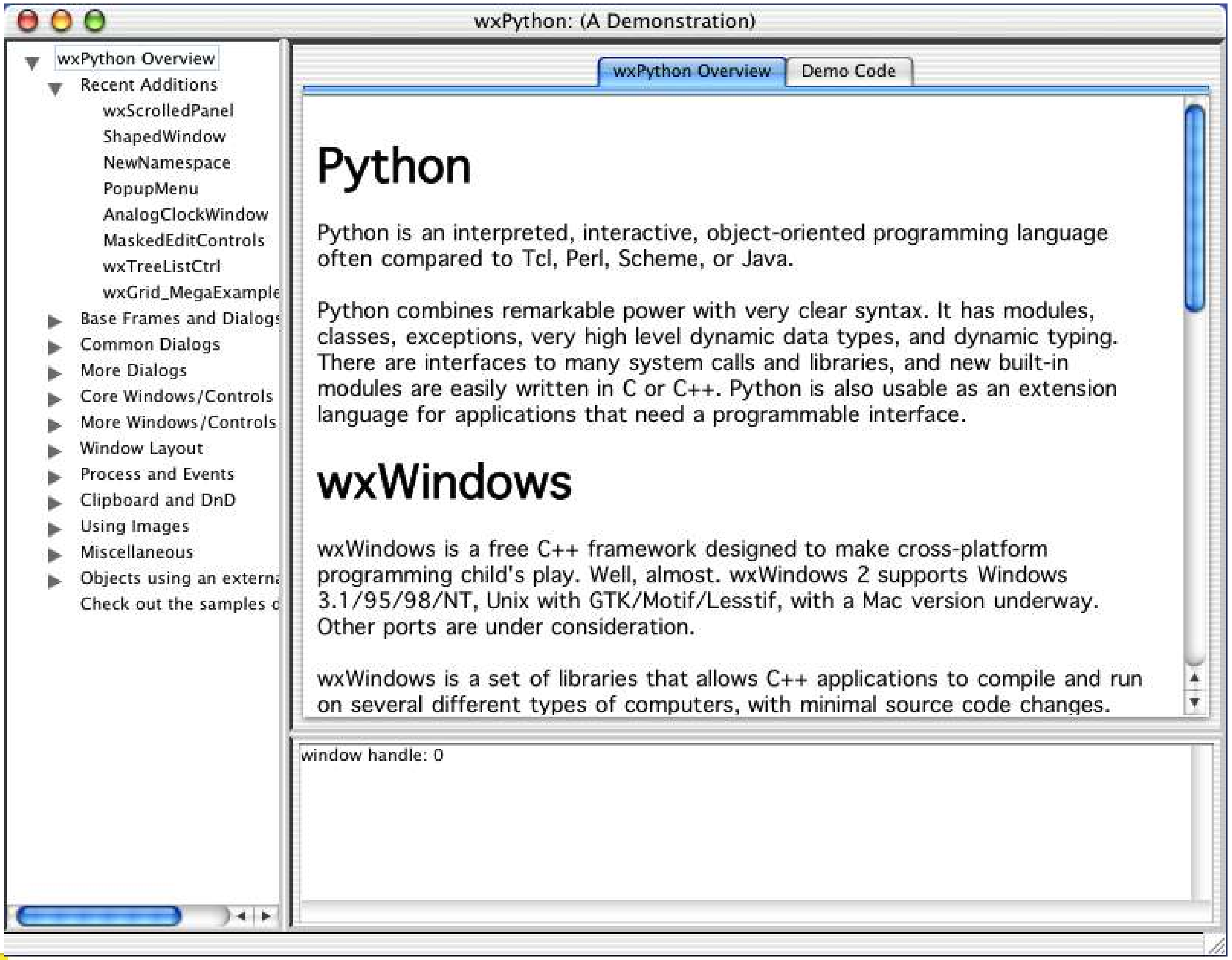
wxWindows is a free C++ framework designed to make cross-platform programming child's play. Well, almost. wxWindows 2 supports Windows 3.1/95/98/NT, Unix with GTK/Motif/Lesstif, with a Mac version underway. Other ports are under consideration.

wxWindows is a set of libraries that allows C++ applications to compile and run on several different types of computers, with minimal source code changes. There is one library per supported GUI (such as Motif, or Windows). As well as providing a common API (Application Programming Interface) for GUI functionality, it provides functionality for accessing some commonly-used operating system facilities, such as copying or deleting files. wxWindows is a 'framework' in the sense that it provides a lot of built-in

```
window handle: 60817489
```



Getting started with wxPython



Python

Python is an interpreted, interactive, object-oriented programming language often compared to Tcl, Perl, Scheme, or Java.

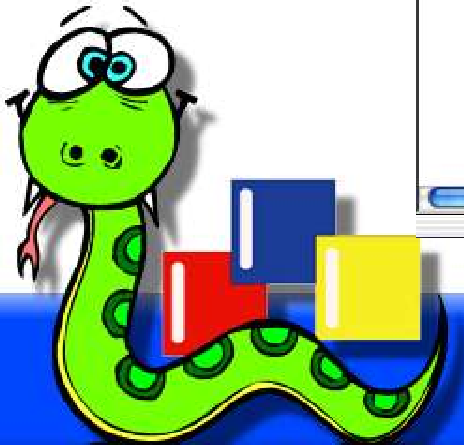
Python combines remarkable power with very clear syntax. It has modules, classes, exceptions, very high level dynamic data types, and dynamic typing. There are interfaces to many system calls and libraries, and new built-in modules are easily written in C or C++. Python is also usable as an extension language for applications that need a programmable interface.

wxWindows

wxWindows is a free C++ framework designed to make cross-platform programming child's play. Well, almost. wxWindows 2 supports Windows 3.1/95/98/NT, Unix with GTK/Motif/Lesstif, with a Mac version underway. Other ports are under consideration.

wxWindows is a set of libraries that allows C++ applications to compile and run on several different types of computers, with minimal source code changes.

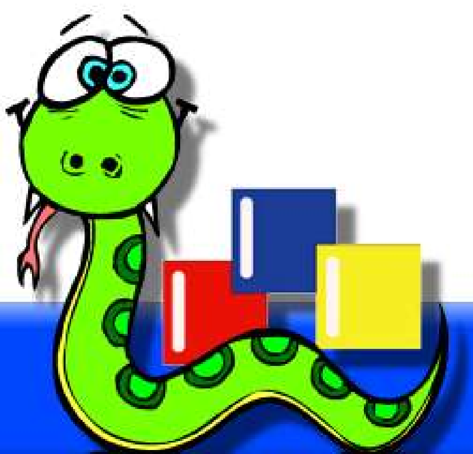
```
window handle: 0
```



Demo time...

Application fundamentals

```
import wx

class App(wx.App):

    def OnInit(self):
        title = 'Bare Frame'
        frame = wx.Frame(parent=None, id=-1, title=title)
        frame.Show()
        return True

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

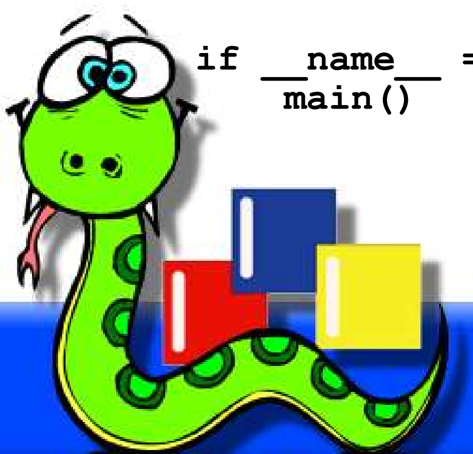


Application fundamentals

```
import wx

class Frame(wx.Frame):
    pass

class App(wx.App):
    def OnInit(self):
        title = 'Spare'
        self.frame = Frame(parent=None, id=-1, title=title)
        self.frame.Show()
        self.SetTopWindow(self.frame)
        return True

if __name__ == '__main__':
    app = App()
    app.MainLoop()
```

Application fundamentals

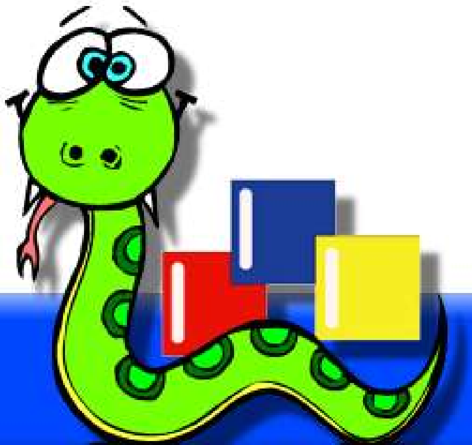
```
import wx

class Frame(wx.Frame):
    def __init__(self, parent=None, id=-1, title='Title',
                 pos=wx.DefaultPosition, size=(400, 200)):
        wx.Frame.__init__(self, parent, id, title, pos, size)


class App(wx.App):
    def OnInit(self):
        self.frame = Frame()
        self.frame.Show()
        self.SetTopWindow(self.frame)
        return True

def main():
    app = App()
    app.MainLoop()

if __name__ == '__main__':
    main()
```



Application fundamentals

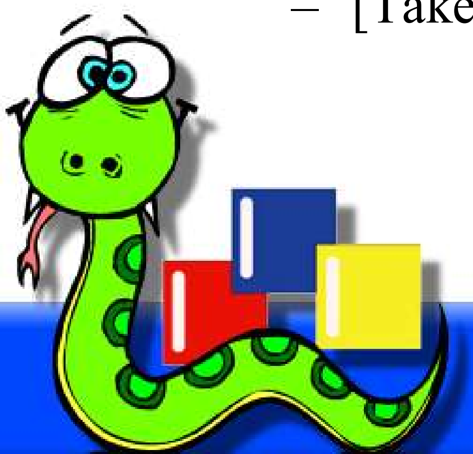
```
"""app.py has a basic application class."""

import wx

from frame import Frame

class App(wx.App):
    def OnInit(self):
        self.frame = Frame(title='This is my App')
        self.frame.Show()
        self.SetTopWindow(self.frame)
        return True

def main():
    app = App()
    app.MainLoop()

if __name__ == '__main__':
    main()
```

Code break...



Widgets galore: top level windows

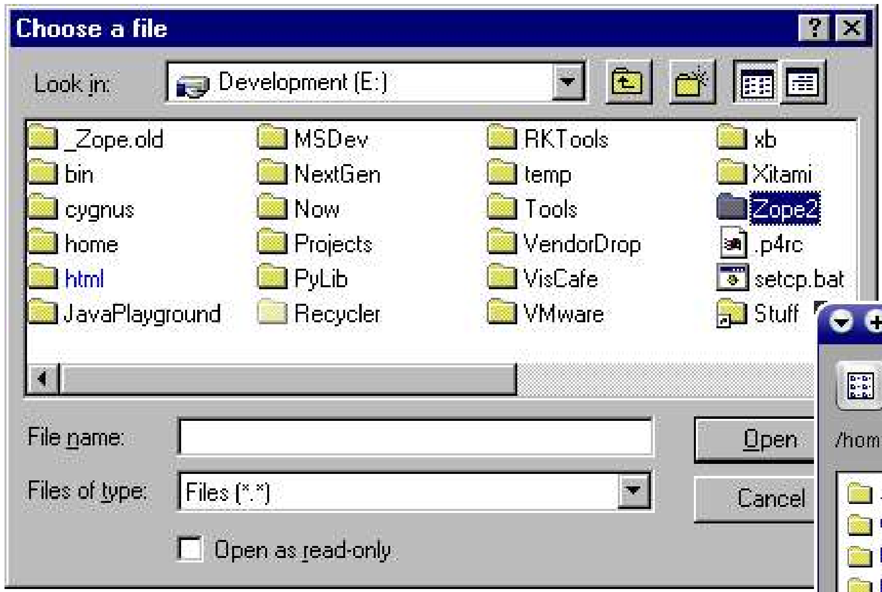
- wx.Frame
  - A container for other windows.
  - Can automatically manage a MenuBar, ToolBar, and a StatusBar.
- wx.Dialog
  - For Modal or Modeless dialog boxes.
- wx.MiniFrame
  - Good for floating tool pallets, etc.
- wx.MDIParentFrame, wx.MDIChildFrame
  - [Take a wild guess :-]

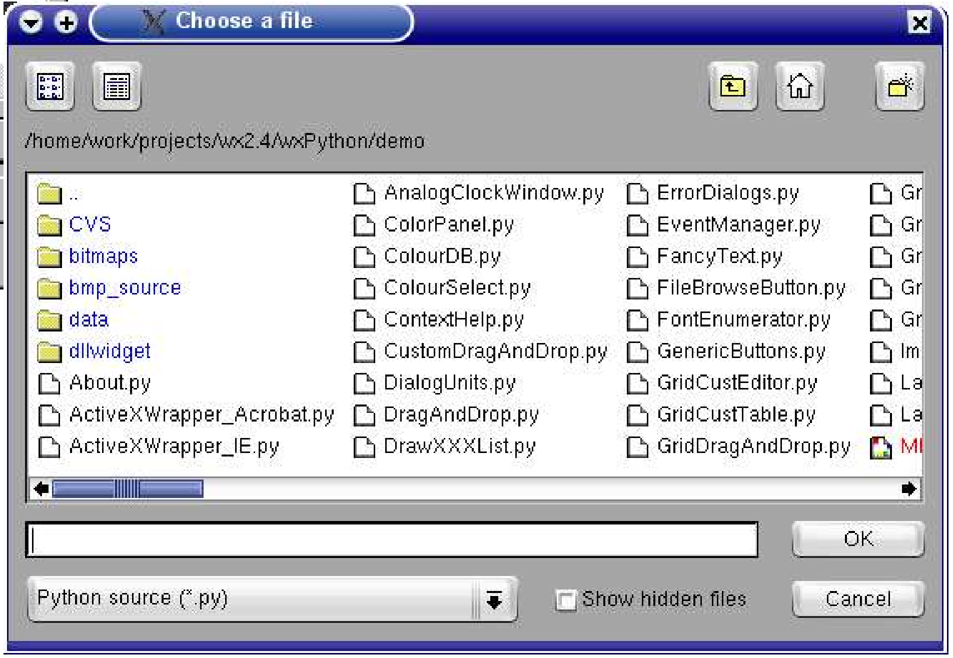


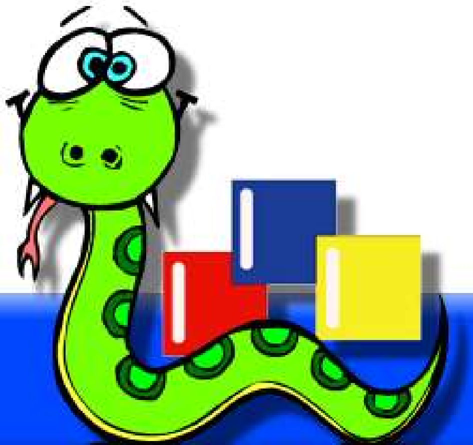
Widgets galore: common dialogs

- 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.

Widgets galore: common dialogs

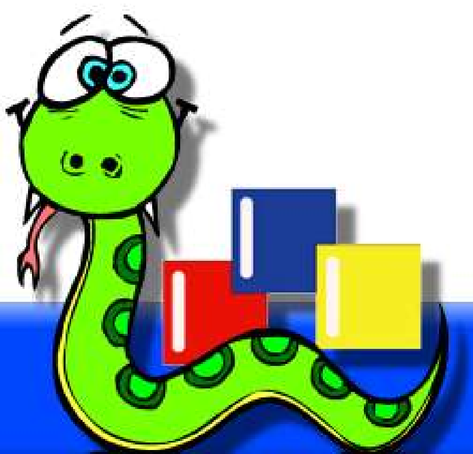






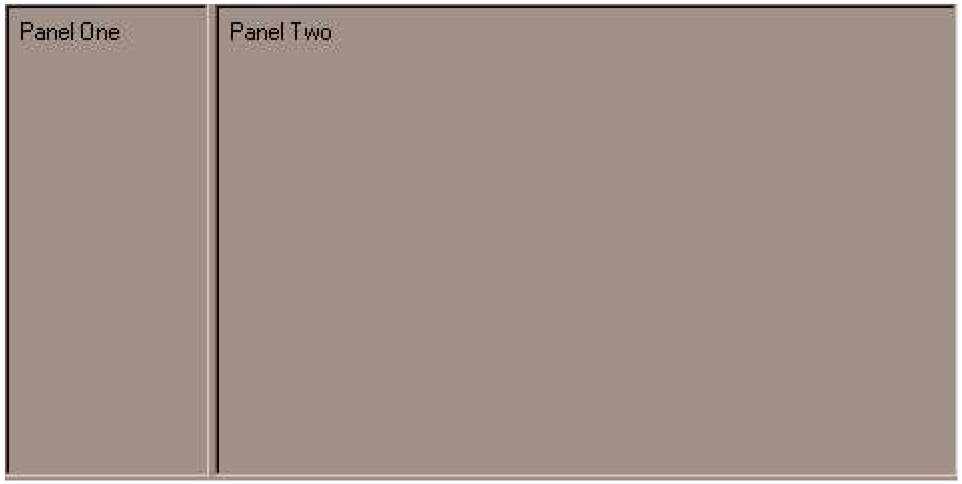
Widgets galore: basic windows

- 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.



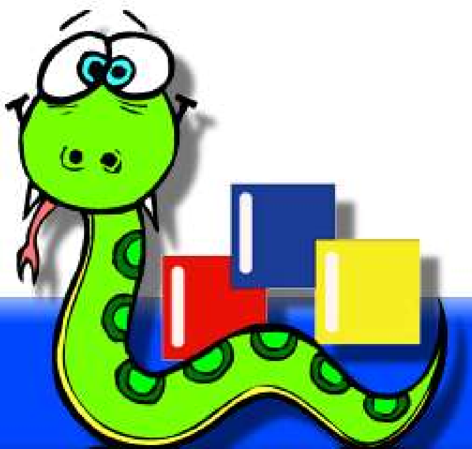
Widgets galore

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

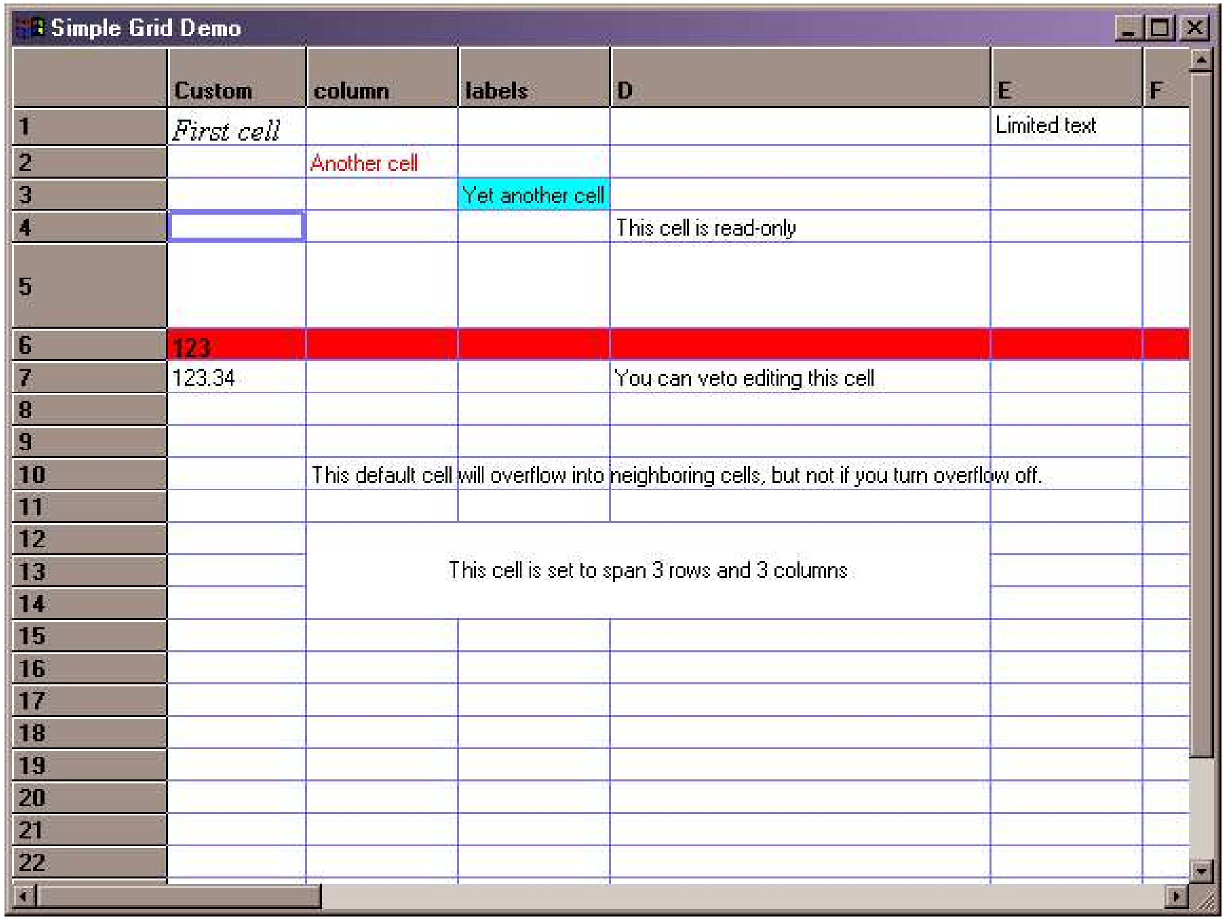


Widgets galore

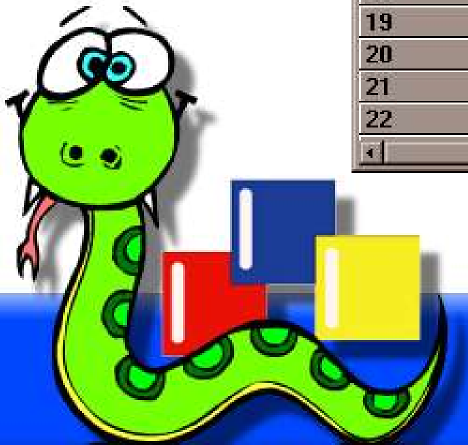
- wx.grid.Grid
  - Table or spreadsheet-like capabilities.
  - Editors, Renderers, Tables (the data provider) can all be customized and “plugged in”.



Widgets galore

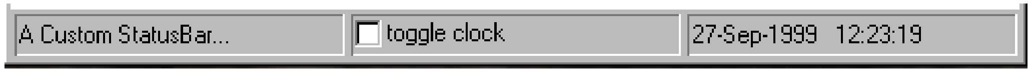

Simple Grid Demo

	Custom	column	labels	D	E	F
1	*First cell*				Limited text	
2		Another cell				
3			Yet another cell			
4				This cell is read-only		
5						
6	123					
7	123.34			You can veto editing this cell		
8						
9						
10		This default cell will overflow into neighboring cells, but not if you turn overflow off.				
11						
12		This cell is set to span 3 rows and 3 columns				
13						
14						
15						
16						
17						
18						
19						
20						
21						
22						



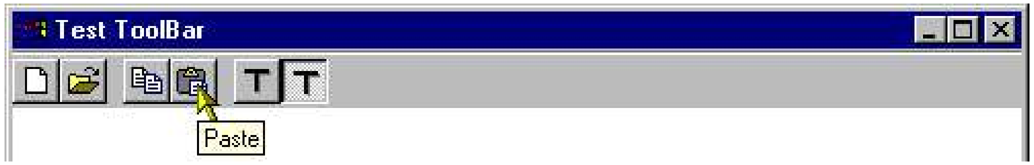
Widgets galore

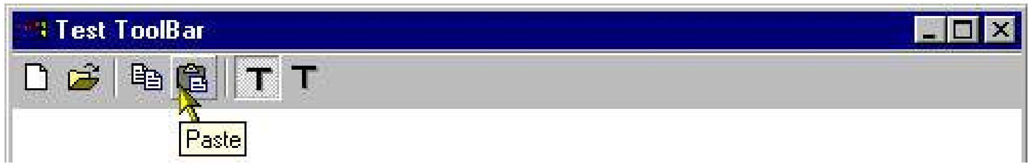
- wx.StatusBar

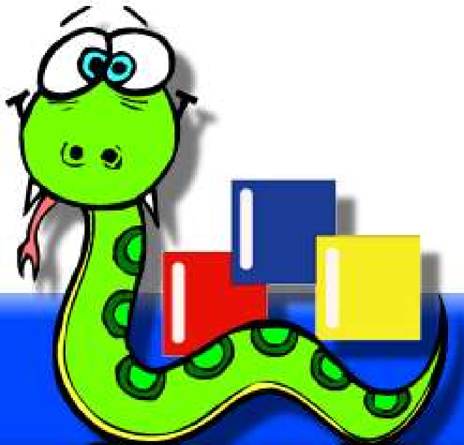


A Custom StatusBar... | toggle clock | 27-Sep-1999 12:23:19

- wx.ToolBar

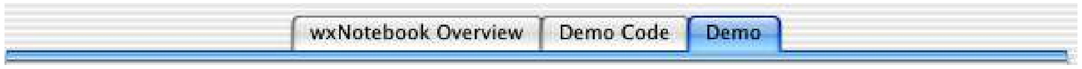


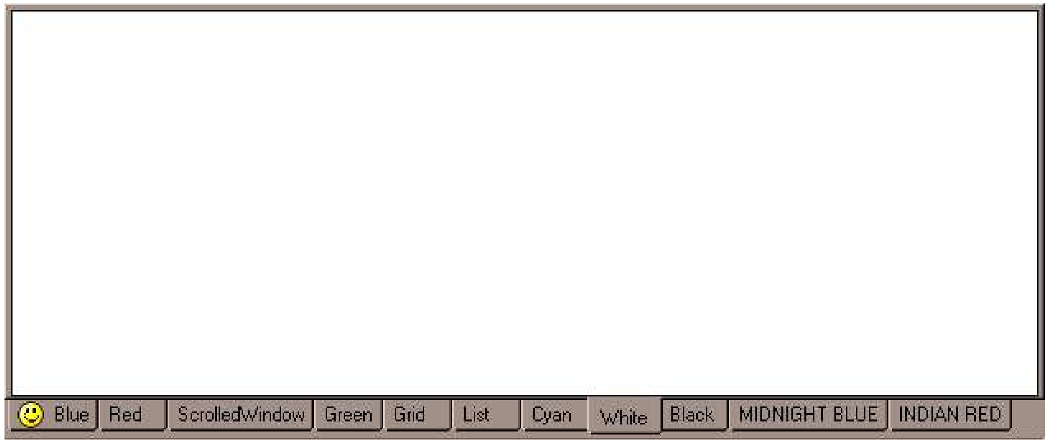




Widgets galore

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





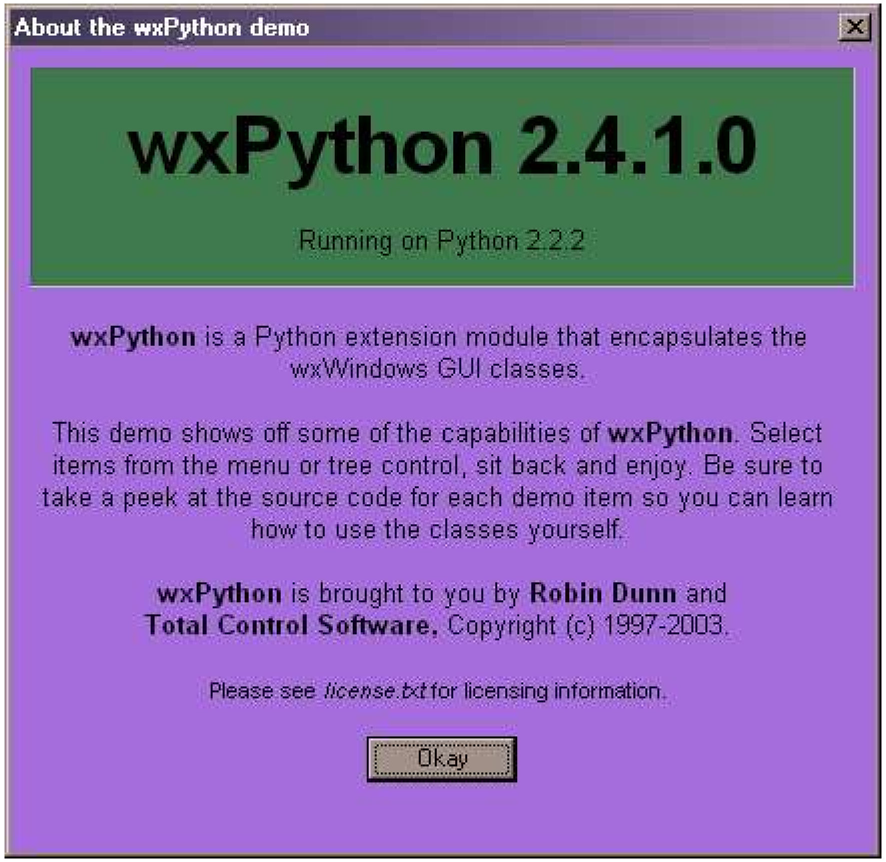
Widgets galore

- 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="wxButton">
    <param name="label" value="Okay">
    <param name="id"    value="wxID_OK">
</wxp>
```

Widgets galore

- wx.html.HtmlWindow



wxPython 2.4.1.0

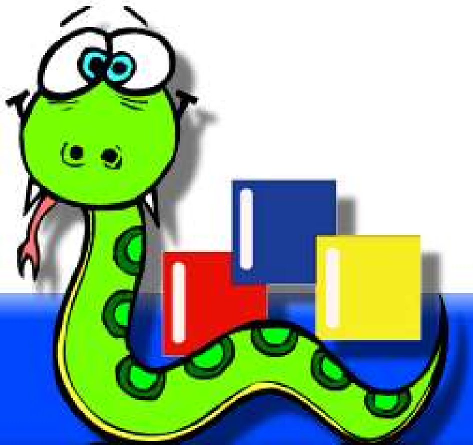
Running on Python 2.2.2

wxPython is a Python extension module that encapsulates the wxWindows GUI classes.

This demo shows off some of the capabilities of **wxPython**. Select items from the menu or tree control, sit back and enjoy. Be sure to take a peek at the source code for each demo item so you can learn how to use the classes yourself.

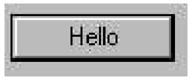
wxPython is brought to you by **Robin Dunn** and **Total Control Software,** Copyright (c) 1997-2003.

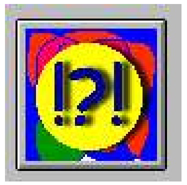
Please see *license.txt* for licensing information.

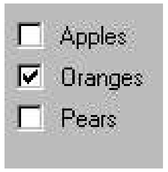


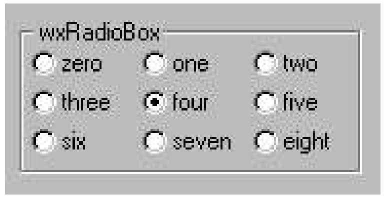
Widgets galore: controls

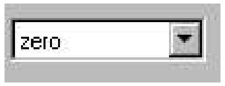
- wx.Button, wx.BitmapButton
- wx.RadioBox, wx.RadioButton
- wx.CheckBox
- wx.Choice
- wx.ComboBox
- wx.SpinButton

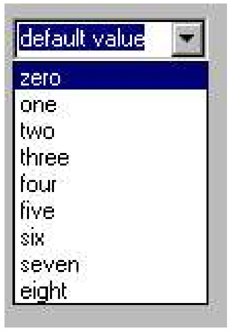


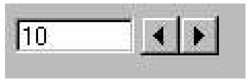


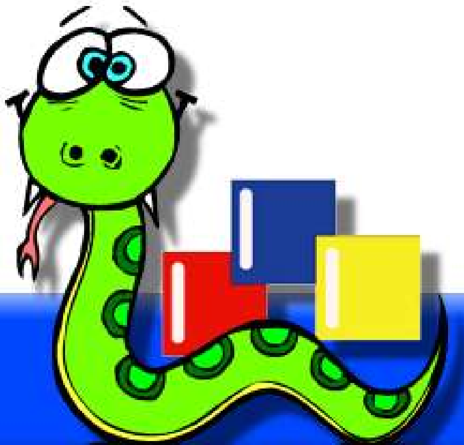






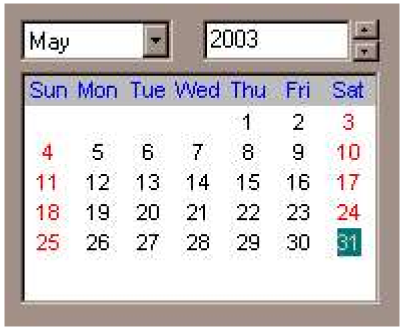




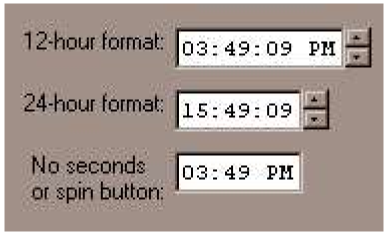


Widgets galore: controls

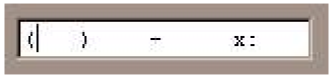
- wx.ToggleButton
- wx.gizmos.EditableListBox
- wx.lib.masked.TextCtrl
- wx.calendar.CalendarCtrl
- wx.lib.masked.TimeCtrl



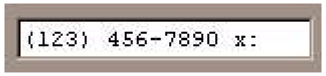
Sun	Mon	Tue	Wed	Thu	Fri	Sat
				1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29	30	31



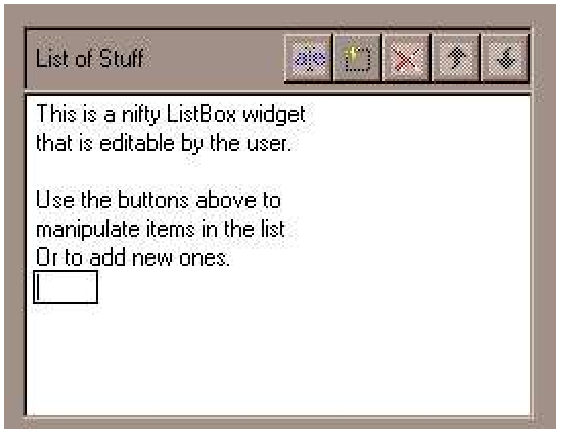
12-hour format: 03:49:09 PM
24-hour format: 15:49:09
No seconds or spin button: 03:49 PM



```
(|   )    -       x:
```



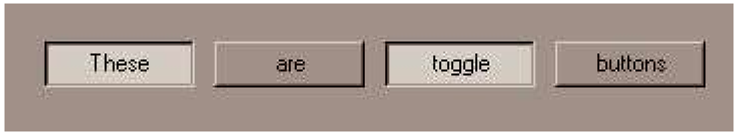
```
(123) 456-7890 x:
```



List of Stuff

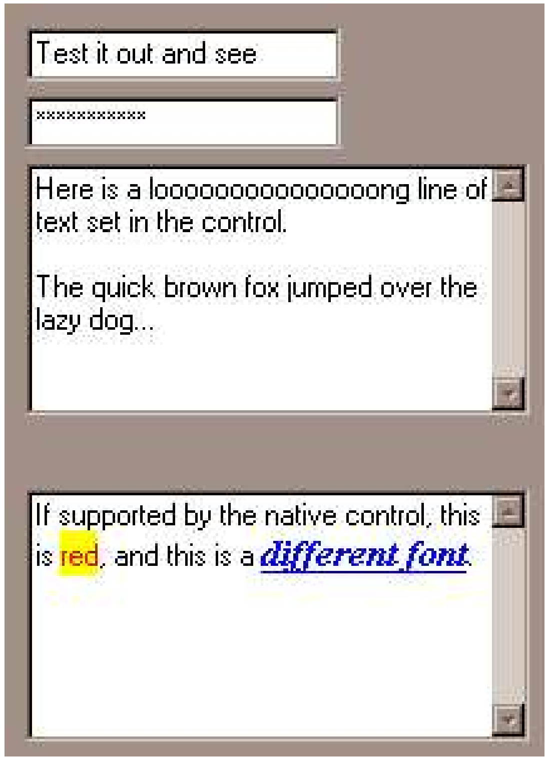
This is a nifty ListBox widget
that is editable by the user.

Use the buttons above to
manipulate items in the list
Or to add new ones.



Widgets galore: controls

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

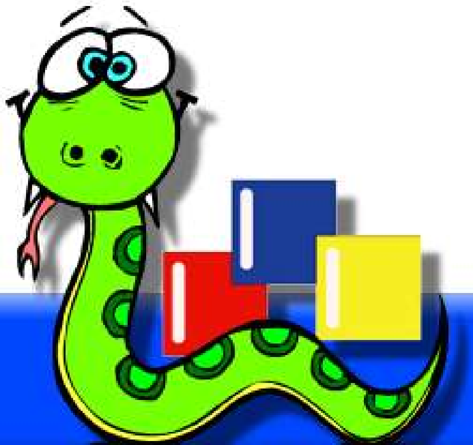

Test it out and see

****

Here is a loooooooooooooooong line of text set in the control.

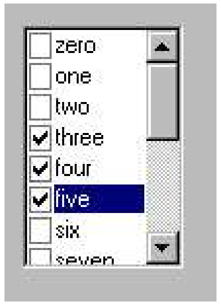
The quick brown fox jumped over the lazy dog...

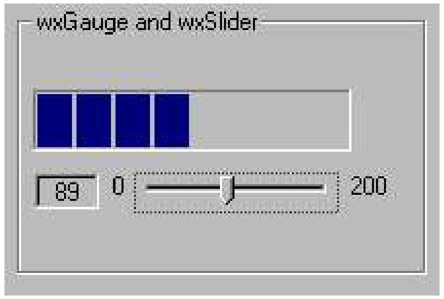
If supported by the native control, this is red, and this is a *different font*.

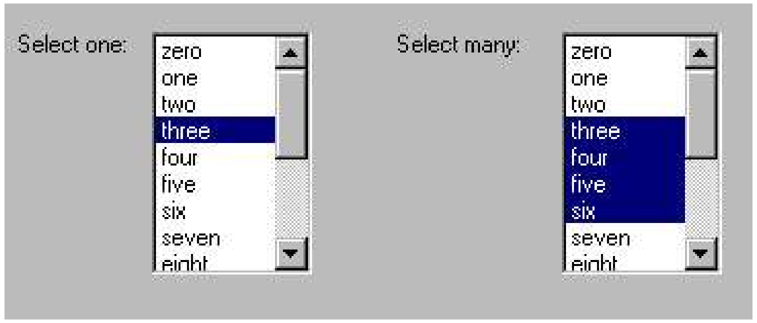


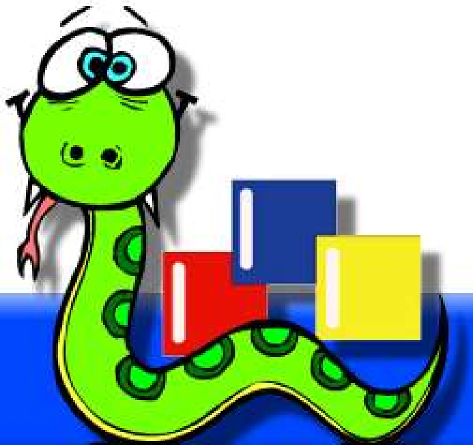
Widgets galore: controls

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



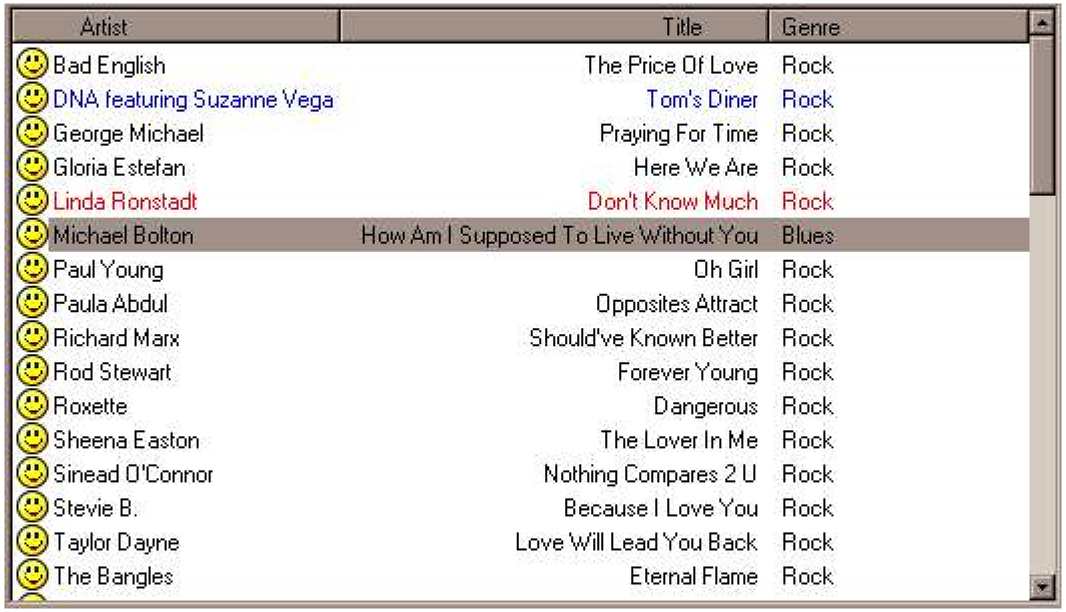






Widgets galore: controls

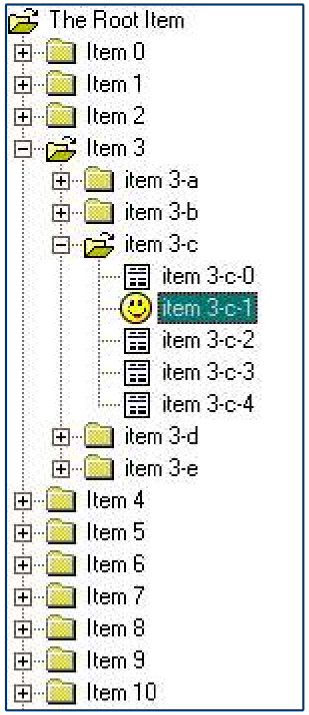
- wx.ListCtrl
  - Supports list, icon, small icon, report views.
  - Virtual mode, where data items are provided by overloaded methods.

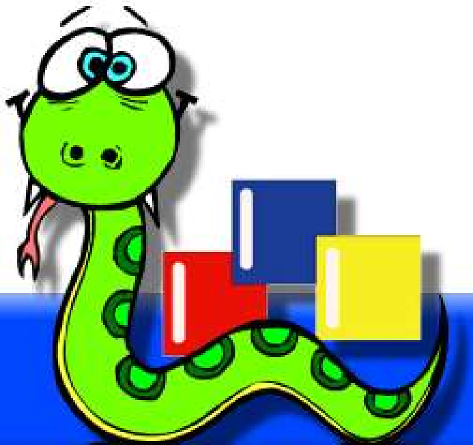


Artist	Title	Genre
Bad English	The Price Of Love	Rock
DNA featuring Suzanne Vega	Tom's Diner	Rock
George Michael	Praying For Time	Rock
Gloria Estefan	Here We Are	Rock
Linda Ronstadt	Don't Know Much	Rock
Michael Bolton	How Am I Supposed To Live Without You	Blues
Paul Young	Oh Girl	Rock
Paula Abdul	Opposites 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

Widgets galore: controls

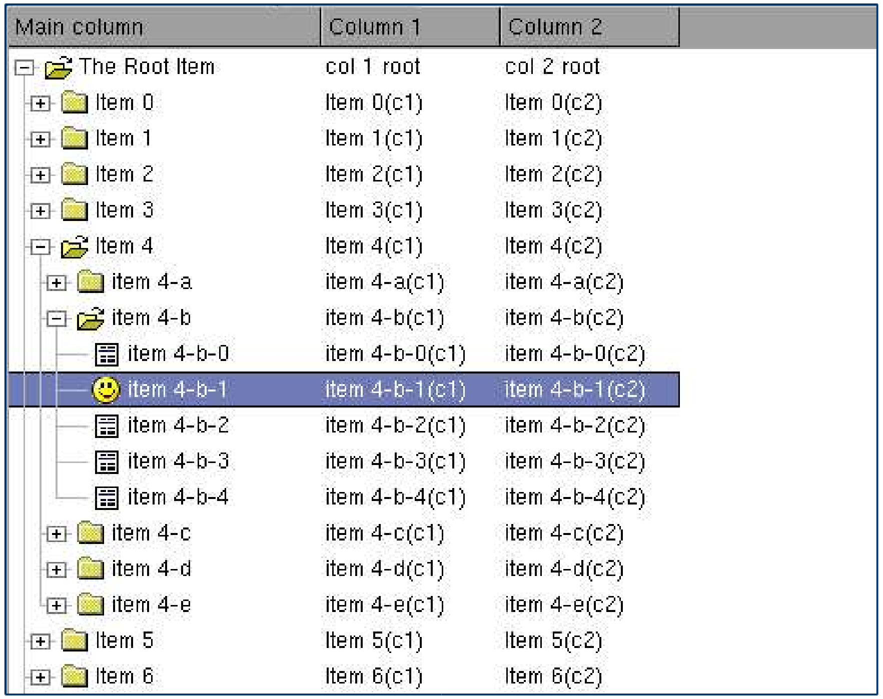
- wx.TreeCtrl
  - Supports images for various node states.
  - Can be virtualized by delaying the adding of child items until the parent is expanded.





Widgets galore: controls

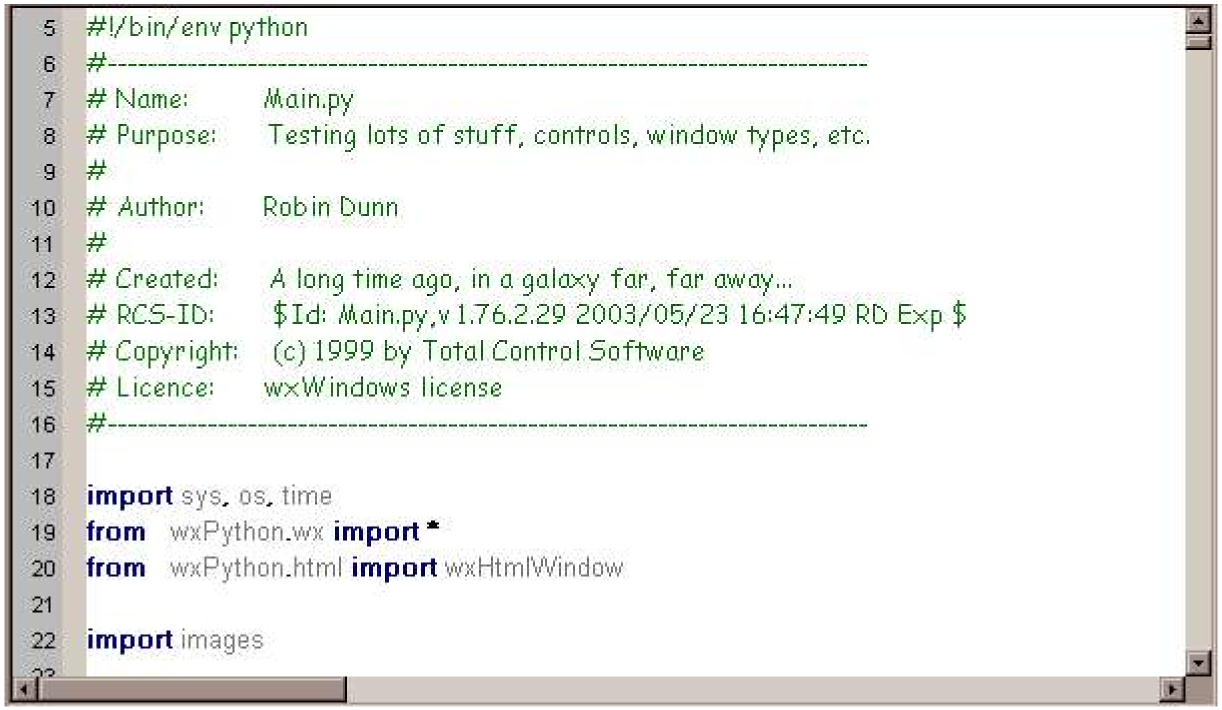
- wx.gizmos.TreeListCtrl



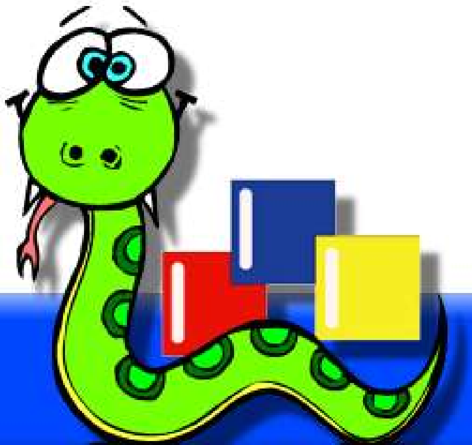
Main column	Column 1	Column 2
The Root Item	col 1 root	col 2 root
Item 0	Item 0(c1)	Item 0(c2)
Item 1	Item 1(c1)	Item 1(c2)
Item 2	Item 2(c1)	Item 2(c2)
Item 3	Item 3(c1)	Item 3(c2)
Item 4	Item 4(c1)	Item 4(c2)
item 4-a	item 4-a(c1)	item 4-a(c2)
item 4-b	item 4-b(c1)	item 4-b(c2)
item 4-b-0	item 4-b-0(c1)	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)

Widgets galore

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



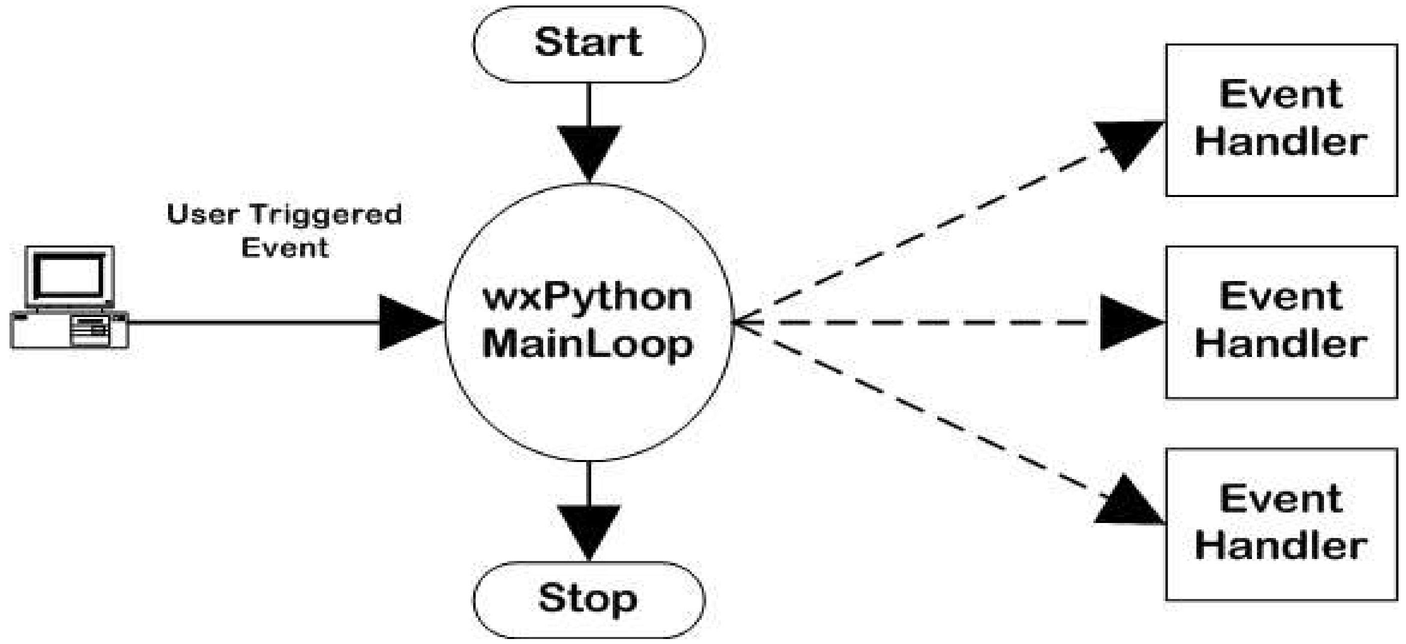
```
#!/bin/env python
#----------------------------------------------------------------------------
# Name:         Main.py
# Purpose:      Testing lots of stuff, controls, window types, etc.
#
# Author:       Robin Dunn
#
# Created:      A long time ago, in a galaxy far, far away...
# RCS-ID:       $Id: Main.py,v 1.76.2.29 2003/05/23 16:47:49 RD Exp $
# Copyright:    (c) 1999 by Total Control Software
# Licence:      wxWindows license
#----------------------------------------------------------------------------

import sys, os, time
from   wxPython.wx import *
from   wxPython.html import wxHtmlWindow

import images
```



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.

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 behaviour, 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.

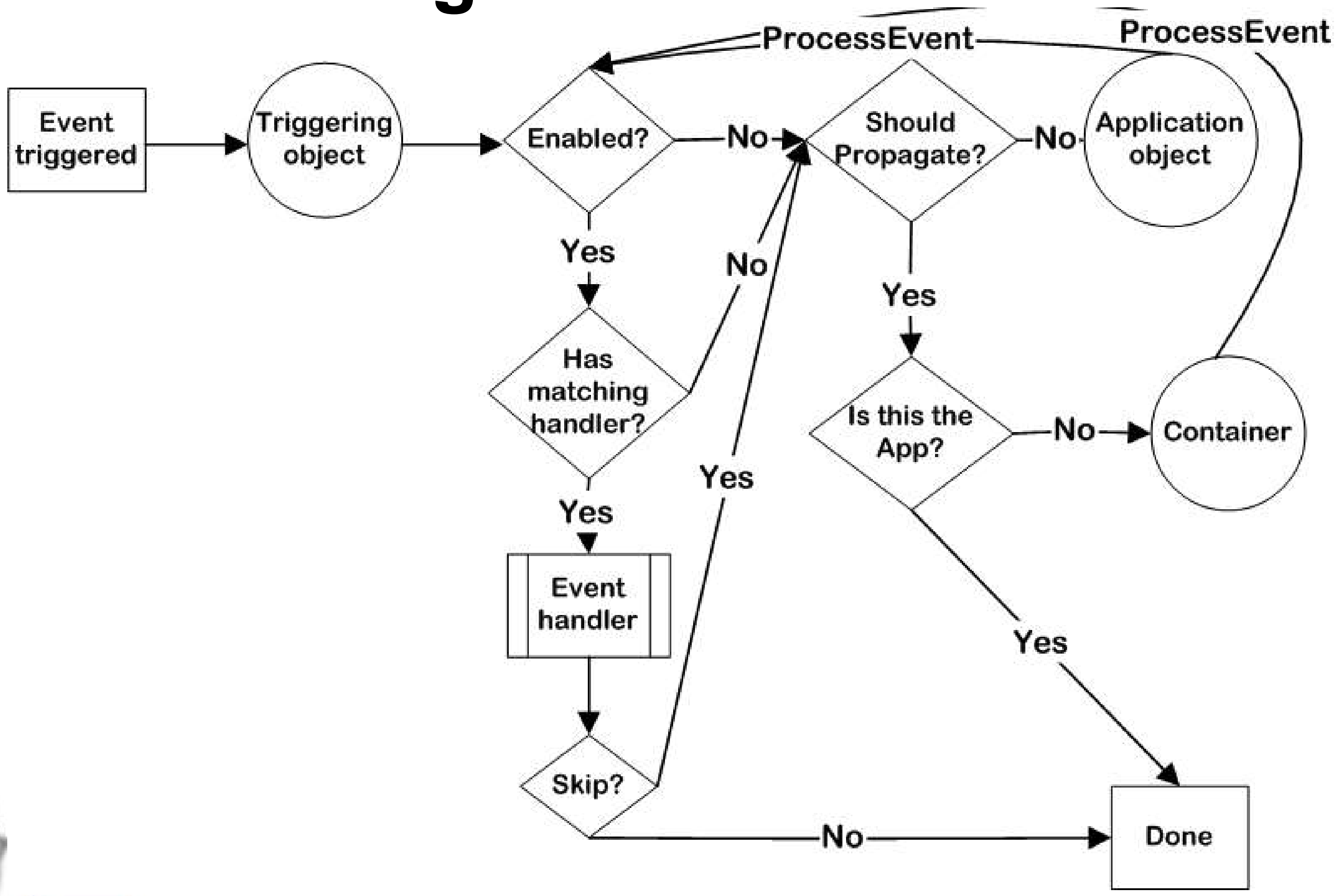
Event Handling

- 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

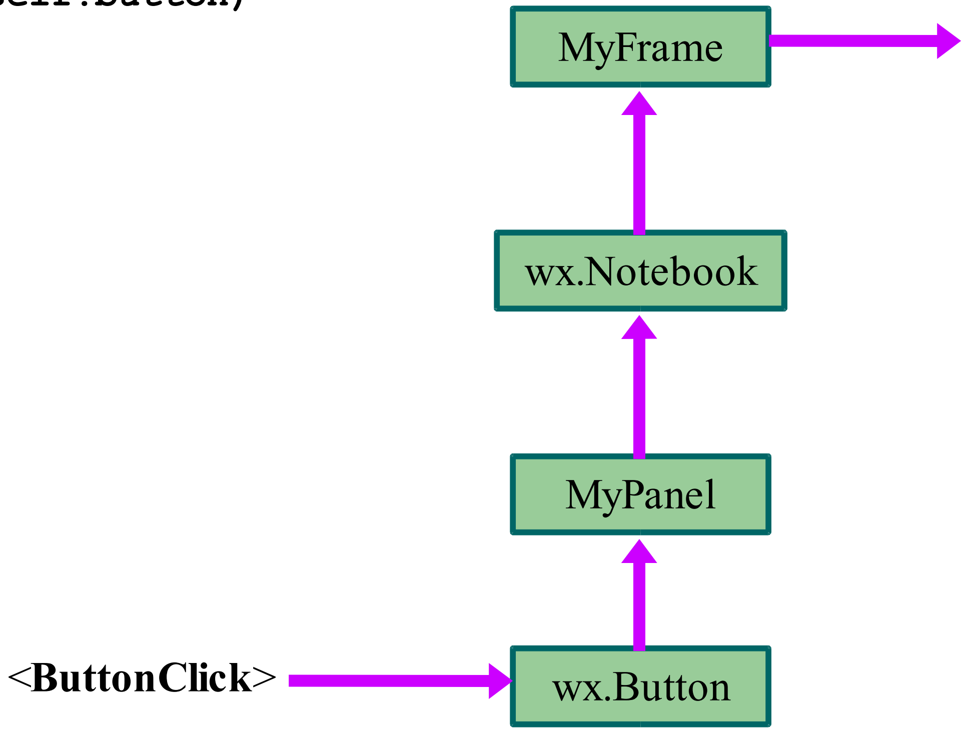
- Each handler is passed an event object when called.
- Two classifications of event objects:
  - Classes derived from **wx.Event**
    - Events that only make sense for the window where the event took place, such as wx.PaintEvent, wx.KeyEvent, wx.SizeEvent, etc.
  - Classes derived from **wx.CommandEvent**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

Event handling

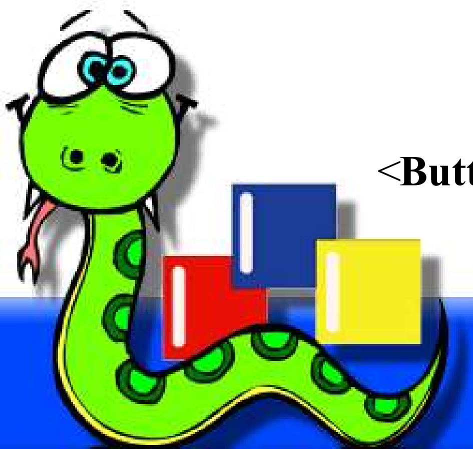


In search of Event Handlers…

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



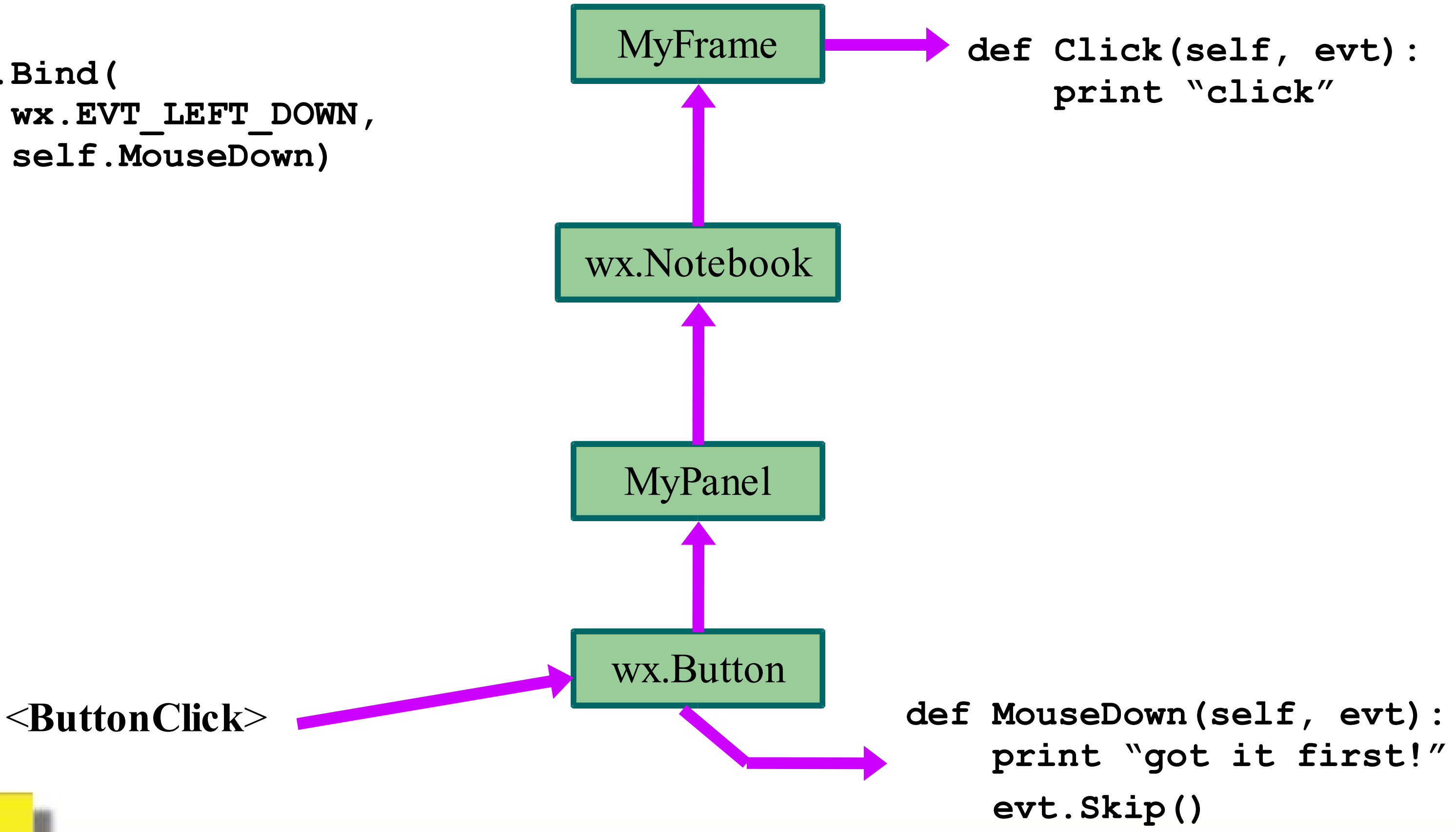
```
def Click(self, evt):
    print “click”
```



In search of Event Handlers…

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

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



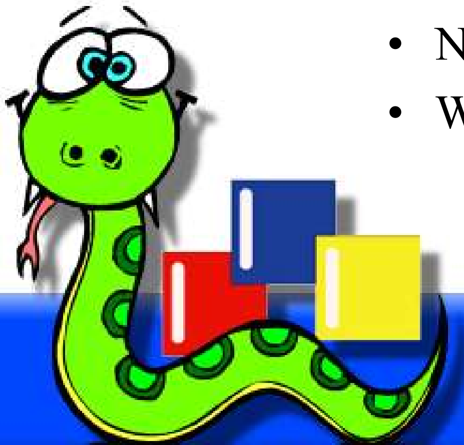
```
def Click(self, evt):
    print "click"
```

```
def MouseDown(self, evt):
    print "got it first!"
    evt.Skip()
```

Code break...

Organizing your layout

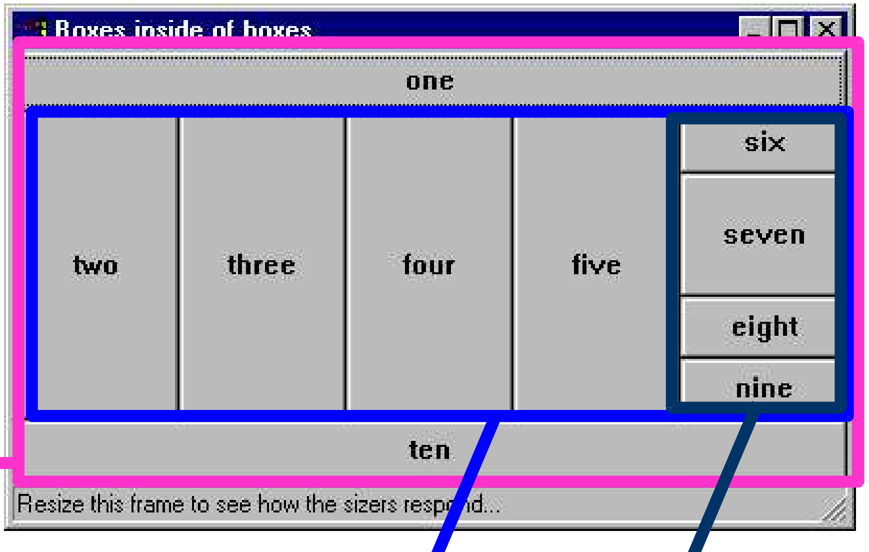
- There are various ways to do layout:
  - Brute force
    - All widgets are positioned and sized pixel by pixel.
    - Has to be redone in every EVT_SIZE event.
    - Painful, cross-platform issues.
  - Layout Constraints
    - Powerful, but complex and verbose.
    - Deals with the relationships between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



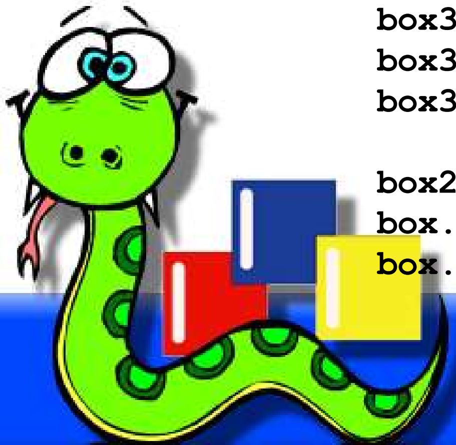
Organizing your layout

- Sizers
  - Similar to LayoutManagers in Java.
  - Not as flexible as LayoutContraints, but much simpler, once you get over the hump.
  - Relationships defined by containment within sizers or nested sizers.
  - All items (windows 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.

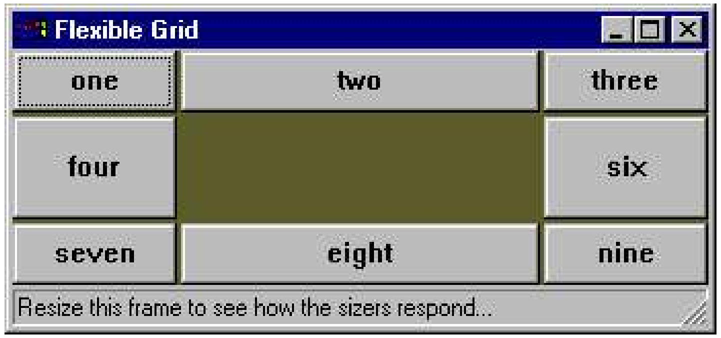
wx.BoxSizer

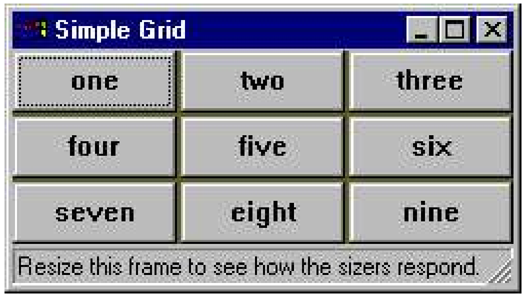


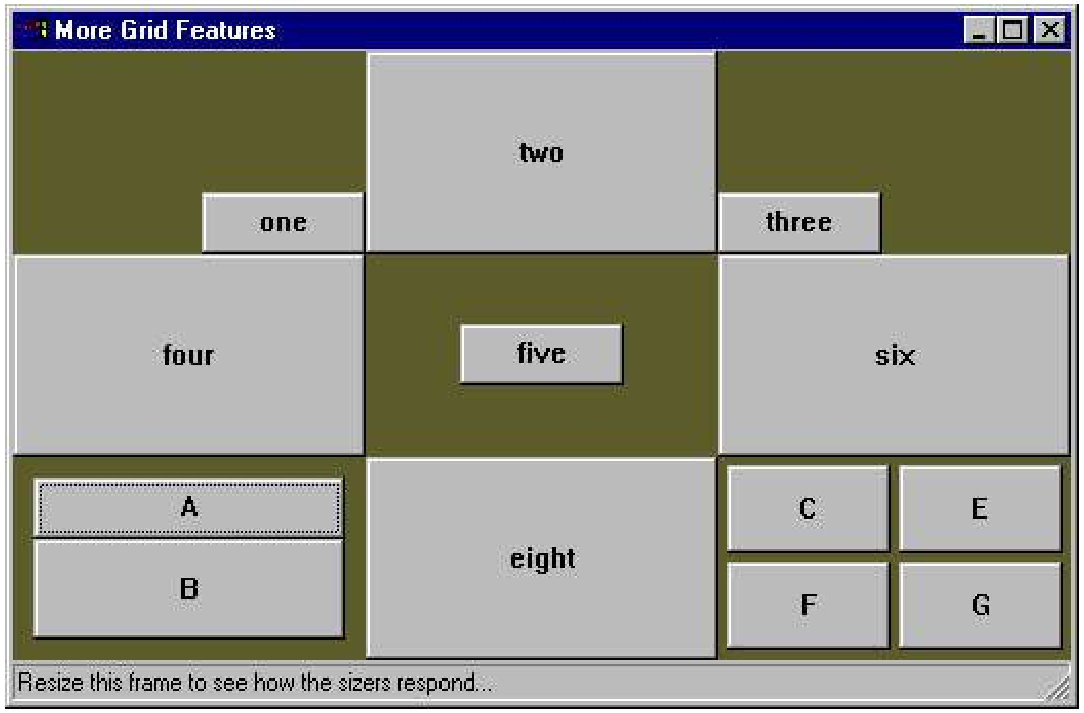
```
box = wx.BoxSizer(wx.VERTICAL)
box.Add(wx.Button(win, 1010, "one"), 0, wx.EXPAND)
box2 = wx.BoxSizer(wx.HORIZONTAL)
box2.Add(wx.Button(win, 1010, "two"),   0, wx.EXPAND)
box2.Add(wx.Button(win, 1010, "three"), 0, wx.EXPAND)
box2.Add(wx.Button(win, 1010, "four"),  0, wx.EXPAND)
box2.Add(wx.Button(win, 1010, "five"),  0, wx.EXPAND)

box3 = wx.BoxSizer(wx.VERTICAL)
box3.Add(wx.Button(win, 1010, "six"),   0, wx.EXPAND)
box3.Add(wx.Button(win, 1010, "seven"), 2, wx.EXPAND)
box3.Add(wx.Button(win, 1010, "eight"), 1, wx.EXPAND)
box3.Add(wx.Button(win, 1010, "nine"),  1, wx.EXPAND)

box2.Add(box3, 1, wx.EXPAND)
box.Add(box2, 1, wx.EXPAND)
box.Add(wx.Button(win, 1010, "ten"), 0, wx.EXPAND)
```

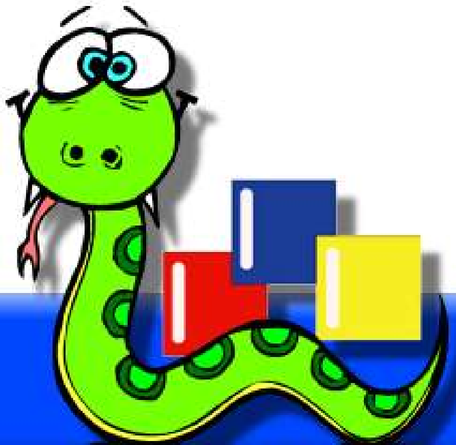


wx.GridSizer

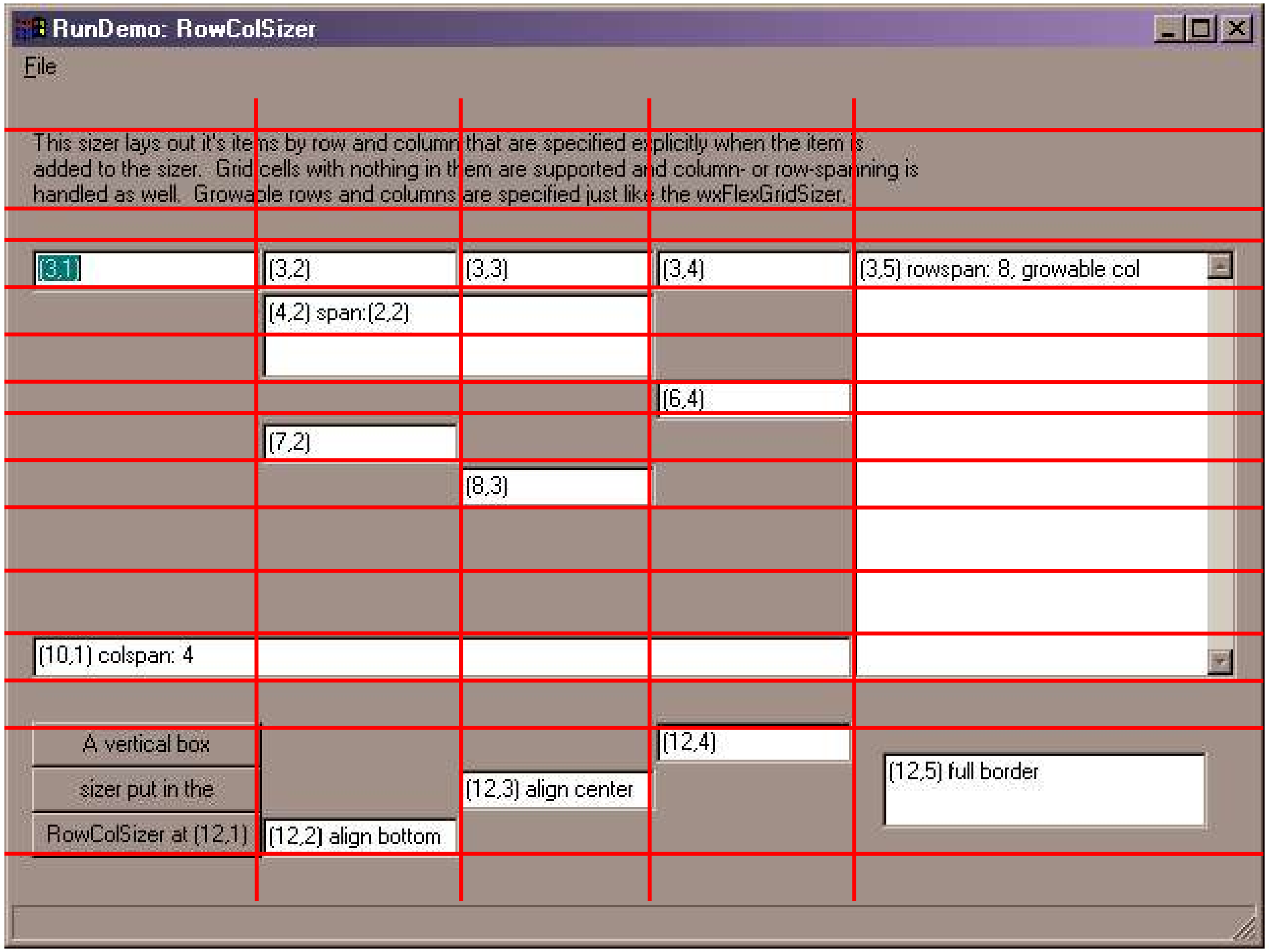




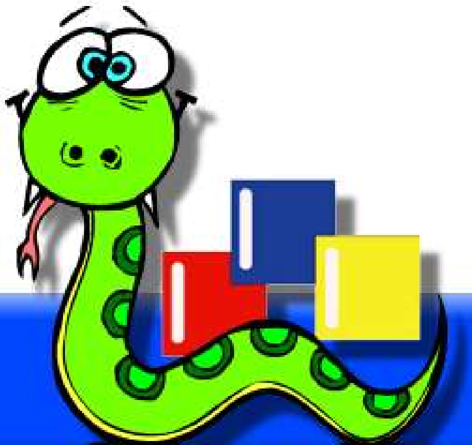




RowColSizer



This sizer lays out it's items by row and column that are specified explicitly when the item is added to the sizer. Grid cells with nothing in them are supported and column- or row-spanning is handled as well. Growable rows and columns are specified just like the wxFlexGridSizer.

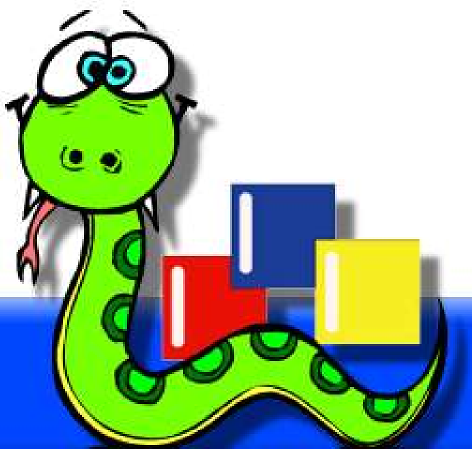


Drawing

- 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

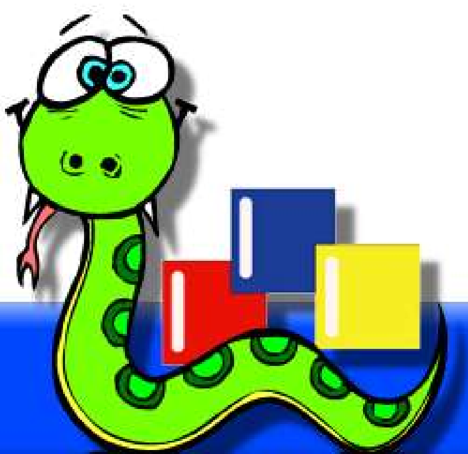
- DC’s have many drawing primitives:
  - DrawArc, DrawBitmap, DrawElipse, DrawLine, DrawLines, DrawPoint, DrawPolygon, DrawRectangle, DrawRoundedRectangle, DrawSpline, DrawText
- And work with GDI objects:
  - wx.Font, wx.Bitmap, wx.Brush, wx.Pen, wx.Mask, wx.Icon, etc.

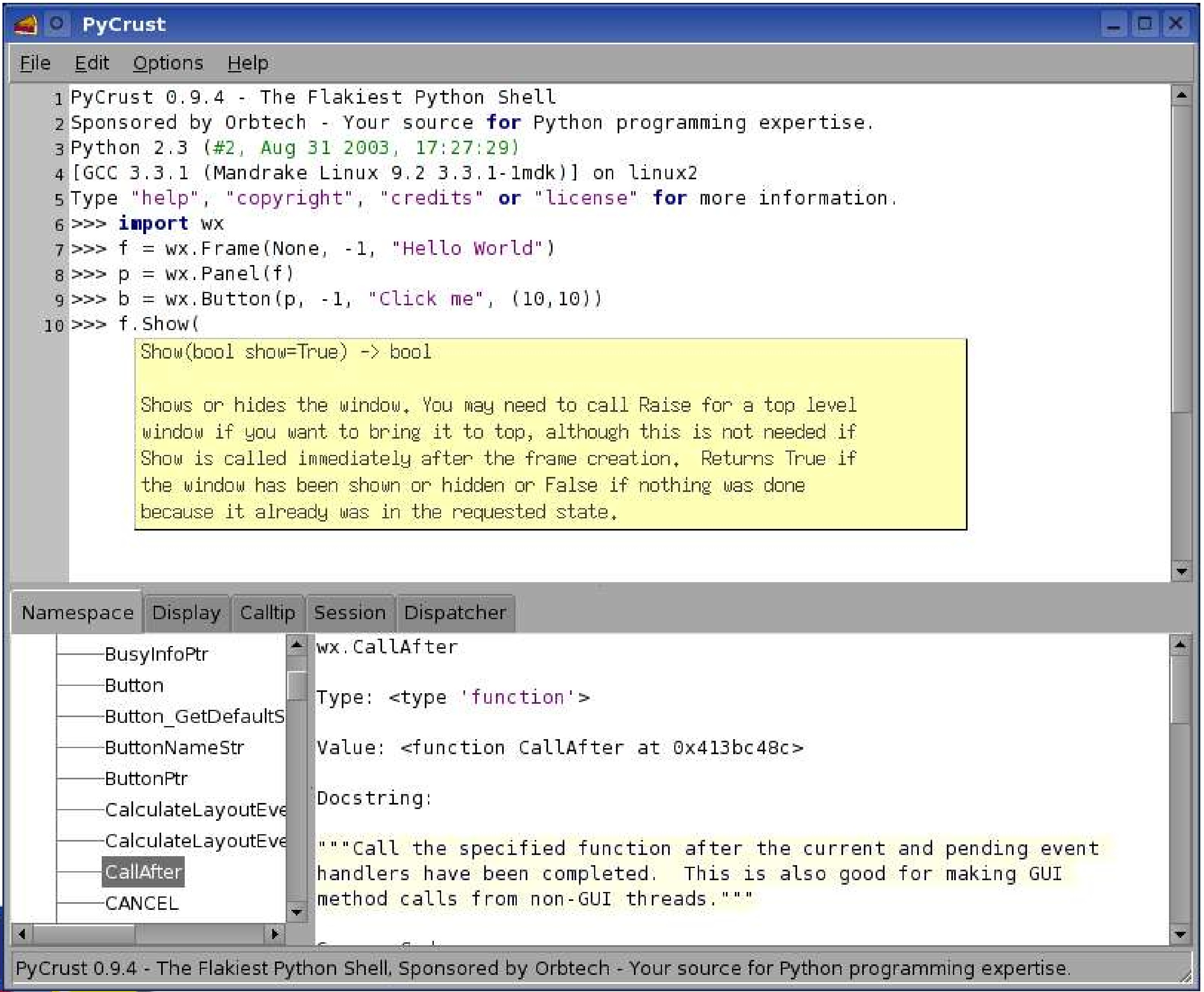


Code break...

Debugging with PyCrust

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



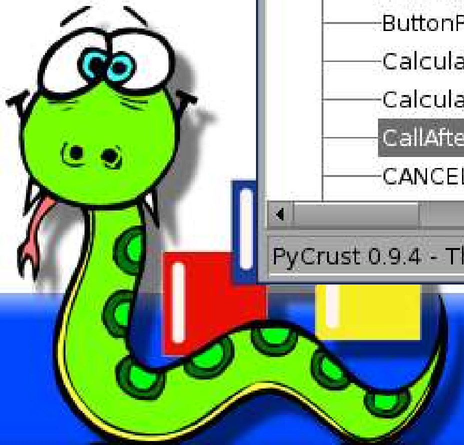


```
PyCrust 0.9.4 - The Flakiest Python Shell
Sponsored by Orbtech - Your source for Python programming expertise.
Python 2.3 (#2, Aug 31 2003, 17:27:29)
[GCC 3.3.1 (Mandrake Linux 9.2 3.3.1-1mdk)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> import wx
>>> f =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

```
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.

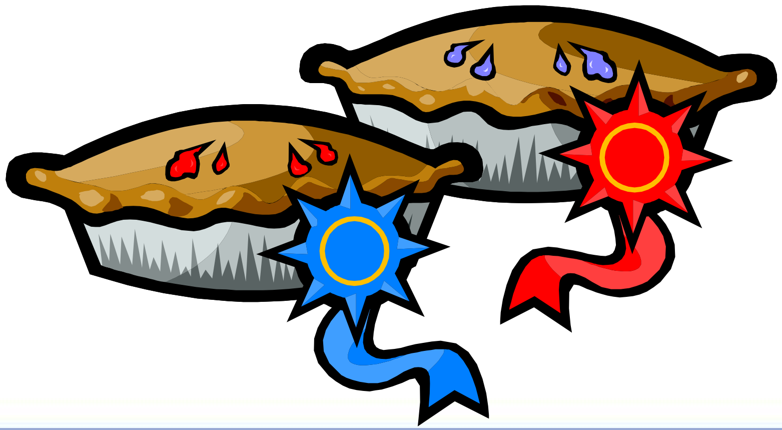


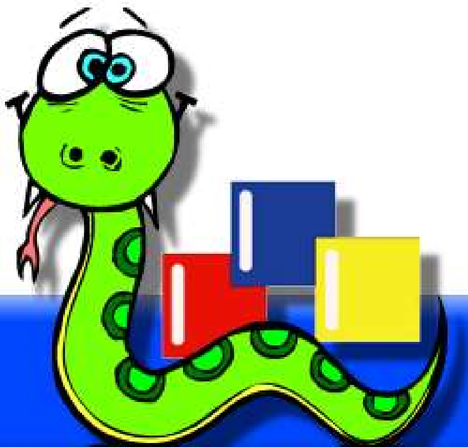
PyCrust Embeddable Components

- Interactive Shell:
  py.shell
- Namespace Viewer:
  py.filling
- Integrated Combo:
  py.crust

PyCrust Features

- Colorized Python Code
- Attribute/Method Auto-Completion
- Function/Method Calltips
- Multiline Command Editing
- Command History/Recall





PyCrust demo...

Other tools

- wxDesigner
- Boa Constructor
- wxGlade
- WingIDE
- PythonCard
- Chandler

Questions?

Last minute additions

- Slides of this presentation are available at: http://wxPython.org/OSCON2004/basic/