Getting Started with wxPython

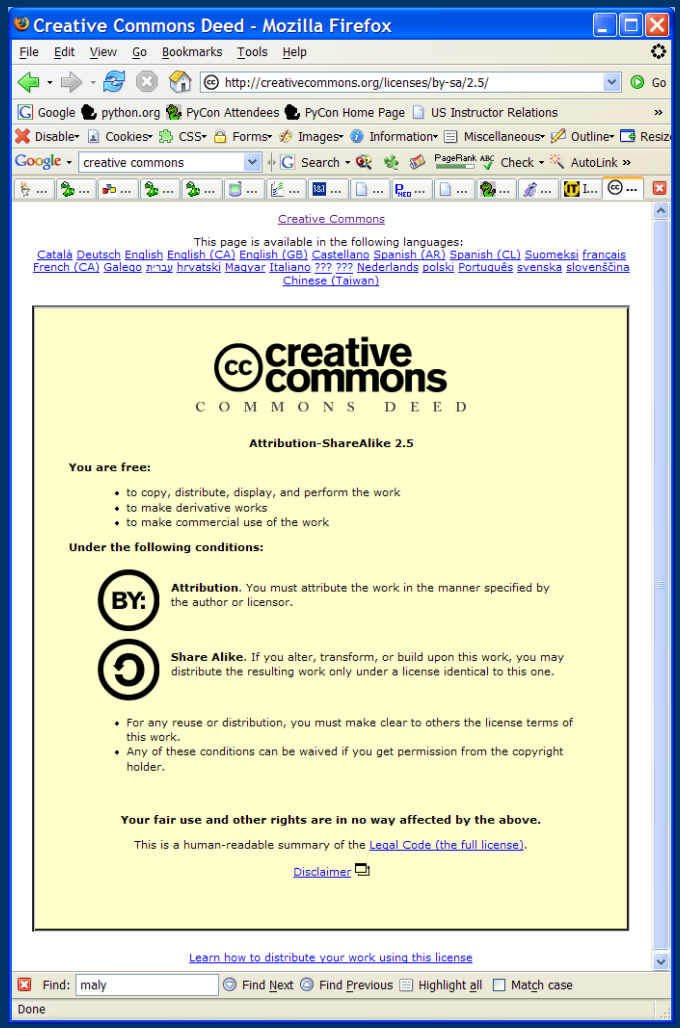
Steve Holden

Holden Web LLC

PyCon TX 2006

Creative Commons License

- Attribution Share-Alike 2.5
  SEE: http://creativecommons.org/licenses/by-sa/2.5/
- This course is copyright ©2006, Holden Web Ltd
  SEE: http://www.holdenweb.com/
- You may use these materials for courses and develop them as long as everything you develop remains available in the same way



Your Instructor: Steve Holden

- Long-term interest in object-oriented programming
  - Since 1973 (SmallTalk)
- Python user since 1998
- Author of *Python Web Programming*
- *Lots* of computing experience
  - Five years as lecturer at Manchester University
  - 15 years as a professional instructor and consultant
  - Still learning ...

General Approach to Exercises

There are relatively few exercises
(we only have 3 hours!)

Load the **`exNN.py`** program into an editor
Run it
Tweak it and store it as **`exNNa.py`**
Run it
Tweak it and store it as **`exNNb.py`**
...
You probably get the idea:
"Rinse, lather and repeat for a few minutes"

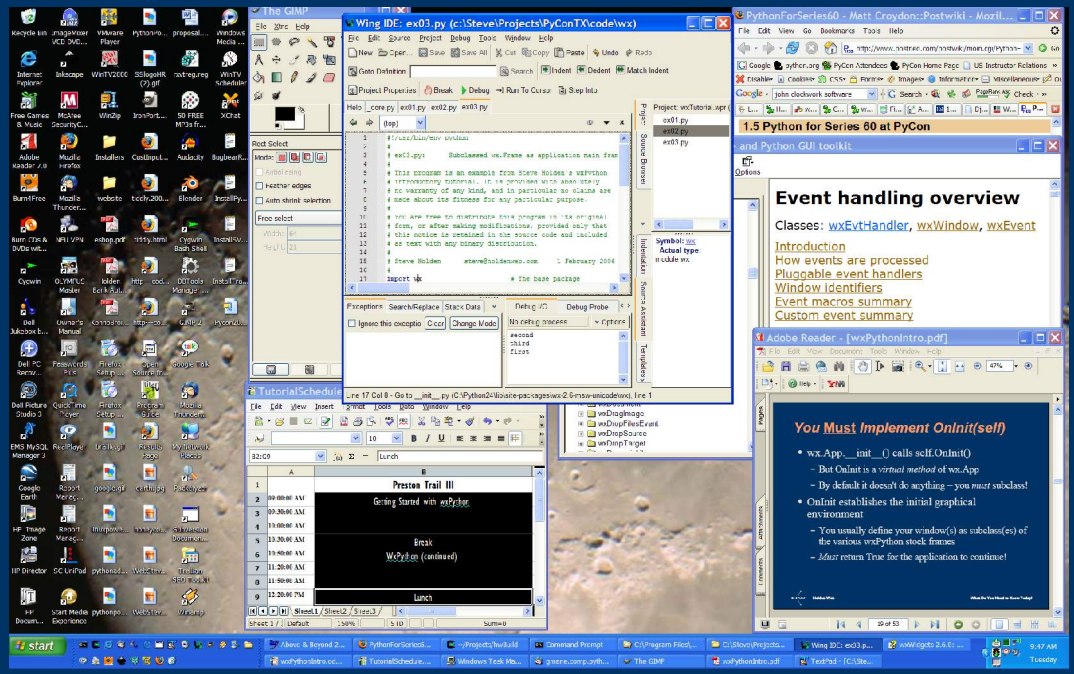
Why wxPython?

wxPython is an open source GUI toolkit based on the wxWidgets (formerly wxWindows) library

It is designed to be cross-platform and supports most Unix/Linux platforms, MS Windows and MacOS X

Several open-source and proprietary GUI-builders are based on wxPython
(Glade, PythonCard, Boa Constructor, wxDesigner ...)

Extensive sample programs, capable community



The computer desktop is a 2-1/2-dimensional space

z-ordering determines what obscures what (and hence which program "sees" particular events)

icons

desktop

http://www.onemindonevoice.org/hope/hopeupdate.html

Window manager

GUI	GUI	GUI	GUI
AcroRead	Firefox	WingIDE	Impress

Each GUI Is a Data Structure

- Each *widget* is either
  - Atomic, or
  - Made up of 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

The Basic wxPython Objects

- wx.App – represents the whole application
  - If the wxApp is not already created many wxPython functions just crash
- wx.Window – almost all widgets are wx.Windows
- wx.Frame – a free-standing window
- wx.Dialog – used to create interaction features
  - Many common dialogs are available “canned”
- wx.Panel – used to hold collections of widgets

Some Common Controls (Widgets)

- wx.Button – click to trigger a program action
- wx.Listbox – holds a list of selectable items
- wx.Choice – a pulldown list
- wx.TextCtrl – lets you enter single-/multi-line text
- wx.CheckBox – for simple yes/no choices
- wx.RadioButton – for mutually exclusive choices
- etc., etc., ... , and last but not least
  - wx.Sizer classes control layout and handle resizing

You Subclass wx.Widgets

- Your application is a wx.App subclass, your frames are wx.Frame subclasses ... and so on
- Your subclasses specialize the abstract behavior of the widgets provided by the toolkit
- Instantiating your subclass creates a widget
  - Which is a data structure registered with your (wx.)App, which therefore receives events
- Properly-written components are easy to re-use
  - Isn't that why we use Python? 

Events are Asynchronous

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

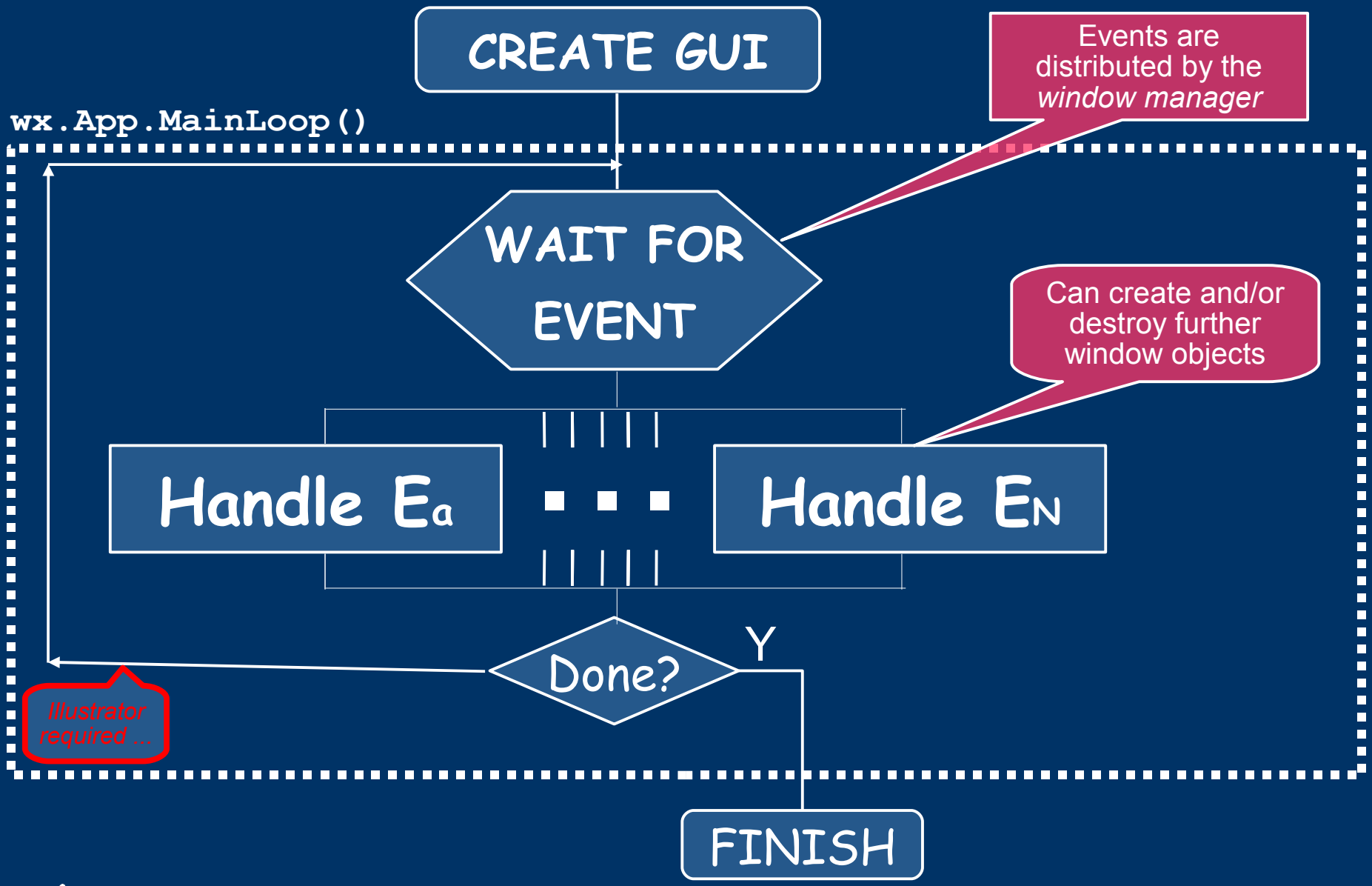
Events are Processed by Handlers

- You bind a handler to the event(s) it should process
  - Handlers are methods of the wx.Widget subclass
- When executed, an event is passed to the handler
  - Though the information is often ignored
- The handler can do almost anything:
  - Modify the user interface: create windows, enable and/or disable controls
  - Change data: write to a database, record input states

Structure of a wxPython Program

- Create a subclass of wx.App (e.g. MyApp)
  - Override OnInit() method to
    - Create initial window(s)
    - Show the main window
- Call myApp.MainLoop()
  - Adds the window to the window manager's list of event consumers
- Erm ... that's it!
  - Except that pesky “create initial window(s)” ☹

Flow of a wxPython Application



Structure of a wxPython Application

1. Create application (wx.App subclass)
2. Create (at least) main frame for application
3. Show main frame
4. Call application.MainLoop()
   - Yields control to window manager
   - Event handlers are run as events occur
     - ***Asynchronously !!!! (scary, isn't it?)***

- This is very different from a traditional program

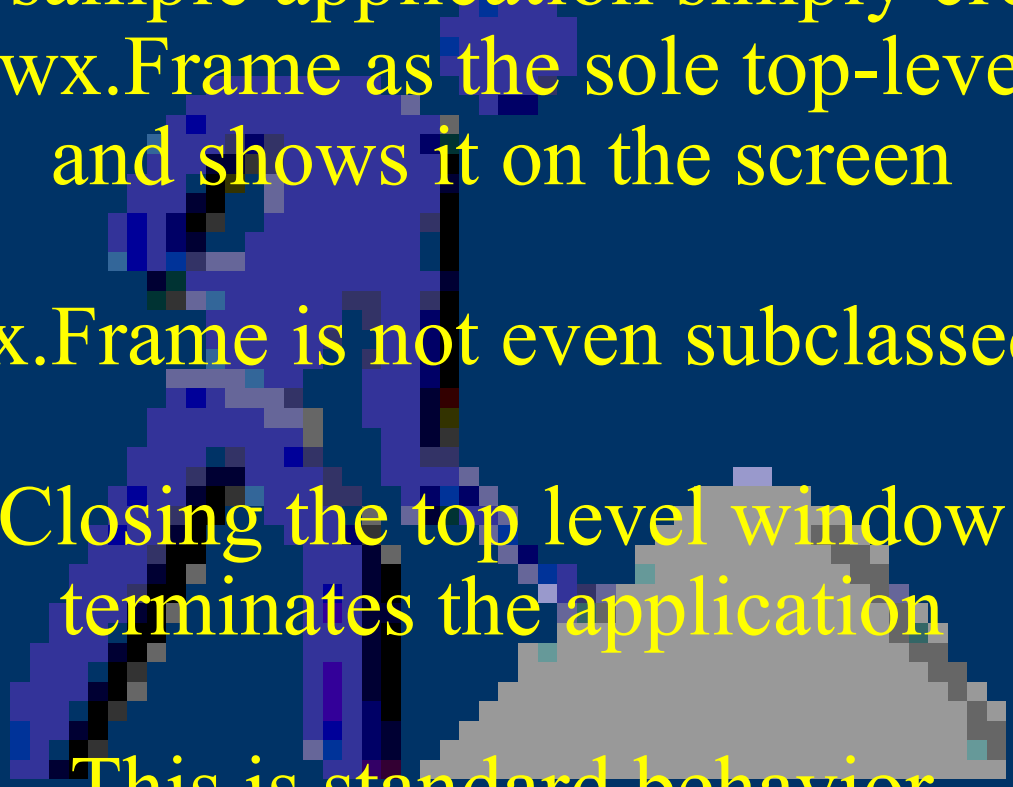
ex01.py: The Simplest Application

This sample application simply creates
an empty wx.Frame as the sole top-level window
and shows it on the screen

wx.Frame is not even subclassed!

Closing the top level window
terminates the application

This is standard behavior



You Must Implement OnInit(self)

- wx.App.__init__() calls self.OnInit()
  - But OnInit is a *virtual method* of wx.App
  - By default it doesn't do anything – you *must* subclass!
- .OnInit() establishes the initial graphical environment
  - Your window(s) are subclass(es) of the various wxPython stock frames
- *Must* return `True` for the application to continue!

wx.App Argument

- wx.App(True)
  - Uncaught exceptions are reported in a pop-up window
- wx.App(False)
  - Uncaught exceptions printed to standard error
- Uncaught exceptions *need not end the program*!
  - Callback exceptions don't cause MainLoop() to return
  - This can help or hinder debugging
  - You get several tries to provoke the error
    - Sometimes the error provokes you

Your App Window Is a wx.Frame

- Or an "acceptable" subclass thereof
- First argument identifies *parent window* or **`None`**
- Other arguments are optional, best as keywords
  - *title* – A string that appears in the title bar
  - *pos* – An (x, y) tuple specifying screen position
  - *size* – A (w, h) tuple specifying window dimensions
  - *style* – How the window is decorated on screen
  - Others *way* beyond the scope of this class

Your Frame is Parent to Controls and/or Other Windows

- All controls *must* have at least two arguments
  - *parent* – specifies the parent frame (or **`None`**)
  - *id* – allows the control to be located in code
    - -1 (wx.ID_ANY) tells wxPython to pick a unique value
- The remaining arguments depend on the control
- Python attributes can be bound to widgets
- They can also be located by their *id* value
  - Using wx.Frame.FindWindowById(*id*)

Names Can Be Bound to Widgets

- A local variable disappears at the end of the method that creates it

```
Btn1 = wx.Button(...)
btns.append(wx.Button(...))
```

- An instance variable is good for the lifetime of the instance containing the widget

```
self.btn1 = wx.Button(...)
self.gui.c = wx.Choice(...)
```

- Globals are *almost* always a bad idea

```
app = MyApp(...)
#every rule has an exception
```

Widgets Can Be Identified

- Give symbolic identity to significant widgets
- Quote identities when creating the widgets

```
Btn1 = 10001; Btn2 = 10002
    ...
Chc1 = 10073
ADD=wx.NewId()
```

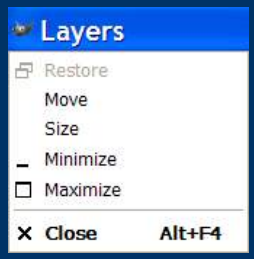
```
wx.Button(f, id=Btn1, ...)
wx.Button(f, id=Btn2, ...)
    ...
wx.Choice(f, id=Chc1, ...)
```

- Widgets can be looked up as needed
- Controls have a GetId() method

```
c = f.GetWindowById(chc)
x = c.GetSelection()
```

Initial Window Created in .OnInit()

- Parent frame is always None
- Default style (**wx.DEFAULT_FRAME_STYLE**) has
  - Title bar buttons
    - minimise, normalise/maximise and close
  - Title bar menu



- A frame can contain any window/widget ...
  - *Except* another frame or a dialog

Most Applications Subclass wx.Frame

- Other frames are similar with added complexities
  - Logic looks something like this:

```
class MyFrame(wx.Frame):
    def __init__(self, parent, id, title, # Can add other args here
                 pos=wx.DefaultPosition,  # Provide sensible defaults
                 size=wx.DefaultSize,
                 style=wx.DEFAULT_FRAME_STYLE):
        wx.Frame.__init__(self, parent, id, title, pos, size, style)
        # Add controls to self here

class MyApp(wx.App):
      def OnInit(self):
            f = MyFrame(None, -1, title="My Window")
            # creating the frame establishes its contents
            f.Show()
            return True

app = MyApp(False)      # Errors become invisible from .pyw
app.MainLoop()
```

Your Frame Can Create Child Frames

- Typically the child frame is created in response to a button click in the parent frame
- Frames remain invisible until you .Show() them
  - You can also .ShowModal() a wx.Dialog subclass
    - Stops *user* events reaching the parent frame until the child wx.Dialog is destroyed
- Destroying a wx.Frame destroys all its children
- Closing the main wx.Frame ends the application
    - wx.Dialog is less cooperative as a main window

Showing a Child Window

- Other frames are similar with added complexities
  - This is one way to do it (see **`ex02.py`**):

```
class MyFrame(wx.Frame):
    def __init__(self, parent, id, title,
                 pos=wx.DefaultPosition,
                 size=wx.DefaultSize,
                 style=wx.DEFAULT_FRAME_STYLE):
        wx.Frame.__init__(self, parent, id, title, pos, size, style)
        # Create controls and subwindows before returning ...
        t1 = wx.StaticText(self, -1, "Main Frame", pos=(50,50))
        f2 = wx.Frame(self, -1, "Second Frame")
        f2.Show()
```

- Normally frames are created in event handlers
  - Observe that second frame closes when first is closed

Component First Argument is Parent

- This establishes the component hierarchy
- Some components are containers
  - wx.Panel, wx.Dialog, wc.MenuBar, wx.Notebook, wx.SashWindow, ...
- Others are for internal purposes
  - wx.Timer, wx.Event, wx.Font, ...
- The rest are for display or direct end-user interaction: what we think of as “widgets”
  - wx.Button, wx.Choice, wx.Menu, wx.RadioBox, wx.TextCtrl, ...

Widgets Are Positioned

- You can give an absolute position
  - `(..., pos=(50, 50), ...)`
  - Technically a wx.Position argument is required
    - In Python you can use `(x, y)` or `[x, y]`
  - Same for size: should be a wx.Size but () and [] OK
- Some tools insist on absolute size and position
- But *wx.Sizer* handles positioning more flexibly
  - Adapts much better to dynamic layouts
  - Sizers are covered later: let's walk before we run ☺

Widget Conventional Wisdom

- Use keyword arguments in constructors:

```
MainFrame = wx.Frame(None,
                     title="A Title",
                     size=(500, 400))
```

- This avoids a bunch of unneeded defaults, like:
  - wx.DefaultSize
  - wx.DefaultPosition
  - wx.ID_ANY, etc
- GUI builders may not do it that way ...

“Ugly” Reality

```
# Code by wxDesigner: see http://www.roebling.de/

def SiteParams( parent, call_fit = True, set_sizer = True ):
    item0 = wx.BoxSizer( wx.VERTICAL )
    item1 = wx.BoxSizer( wx.HORIZONTAL )
    item2 = wx.StaticText( parent, ID_TEXT, "Site:",
                                        wx.DefaultPosition, [20,-1],
wx.ALIGN_RIGHT )
    item1.Add( item2, 1, wx.ALIGN_CENTER|wx.ALL, 5 )
    item1.Add( [ 20, 20 ] , 0, wx.GROW|wx.ALIGN_CENTER_HORIZONTAL|wx.ALL, 5 )
    item3 = wx.Choice( parent, chcSiteName, wx.DefaultPosition, [120,-1],
        ["ChoiceItem"] , 0 )
    item1.Add( item3, 2, wx.ALIGN_CENTER|wx.ALL, 5 )
    item0.Add( item1, 1, wx.GROW|wx.ALIGN_CENTER_VERTICAL|wx.ALL, 5 )
    item5 = wx.StaticBox( parent, -1, "Database Parameters" )
              ...
    item0.Add( item46, 0, wx.GROW|wx.ALIGN_CENTER_VERTICAL|wx.ALL, 0 )

    if set_sizer == True:
        parent.SetSizer( item0 )
        if call_fit == True:
            item0.SetSizeHints( parent )

    return item0
```

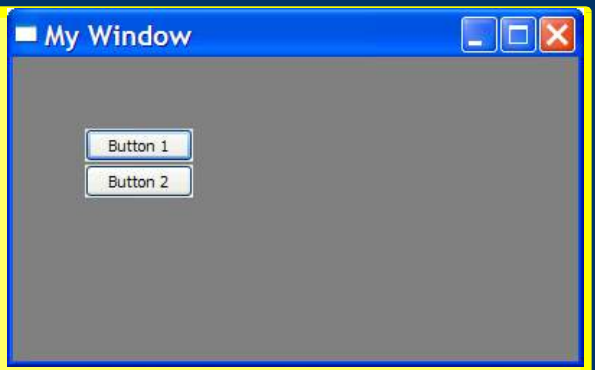
ex03.py: *Two Buttons*

This application creates a wx.Frame with two buttons positioned inside it

Try ...

- changing button labels
- changing button positions
- using a wx.Dialog instead of a wx.Frame
- omitting some wx.Button arguments

ex03.py *Essentials*




```
class MyFrame(wx.Frame):
    def __init__(self, parent, id, title,
                 pos=wx.DefaultPosition,
                 size=wx.DefaultSize,
                 style=wx.DEFAULT_FRAME_STYLE):
        wx.Frame.__init__(self, parent, id, title, pos, size, style)
        b1 = wx.Button(self, -1, "Button 1", pos=(50, 50))
        b2 = wx.Button(self, -1, "Button 2", pos=(50, 75))

class MyApp(wx.App):
      def OnInit(self):
          f = MyFrame(None, -1, "My Window")
          # The frame establishes its own contents
          f.Show()
          return True

app = MyApp(False)
app.MainLoop()                  # ... and wait for events to process
```

Common Widget Arguments

- **id** – numeric identifier used for widget lookup
- **pos** – absolute (x, y) position in window
- **size** – mostly defaults to “just large enough”
- **style** – sets widgets' appearance and behaviour
  - Options vary between the widgets
  - Flags can be added or OR'ed together:
    - e.g: `wx.TE_READONLY | wx.TE_PROCESS_ENTER`
- **validator** – used to limit entry to valid data
- **name** – optional “window name”

Less commonly used

Some Real Program Statements

```
txt1 = wx.TextCtrl(parent, txtDbName, "",
  wx.DefaultPosition, wx.DefaultSize, 0)
```

- TxtDbName is a symbolic id
- Default size and position not really needed

```
item2 = wx.ListBox(parent, chcPageNames,
  wx.DefaultPosition, [80,100], [],
  wx.LB_SINGLE)
```

- [80, 100] is ListBox size
- wx.LB_SINGLE allows just one item to be selected

wx.Widget.Destroy()

- Fortunately, frames are generally well-behaved
- `wx.Frame.Destroy()` also normally destroys all child widgets automatically
- Some frames do have to be explicitly destroyed
  - We conveniently ignore this ☺
- It's usually acceptable to
  - Create windows on the fly each time they are needed
    - Removes any initialisation hassle
  - Destroy them after each use

wx.Button


```
wx.Button(parent,
                  id, label="", pos=(-1, -1),
                  size=(-1, -1), style=0,
                  validator=wx.DefaultValidator,
                  name="button")
```

- Typically used to trigger program actions
- Left-click generates an interface event
  - We talk about event handling in the next section

wx.Button Methods

- `.SetDefault()`
  - causes RETURN to act like a click on this button
- `.SetLabel(newlabel)`
  - changes the caption on the button

Generally speaking, you can expect GetXXX()/SetXXX() pairs to exist wherever both make sense

wx.TextCtrl


```
wx.TextCtrl(parent,
                  id, value="", pos=(-1, -1),
                  size=(-1, -1), style=0,
                  validator=wx.DefaultValidator,
                  name="text")
```

- Single- or multi-line text entry
- Many style options, including
  - wx.TE_MULTILINE allows multiple-line data
  - wx.TE_READONLY renders data non-editable
  - wx.TE_LEFT, wx.TE_RIGHT, wx.TE.CENTRE

wx.TextCtrl Methods

- `.GetValue()` : reads current text from control
- `.SetValue(s)` : sets control entry to *s*
- `.GetInsertionPoint()` : locates text cursor
- `.GetSelection()` : returns (from, to) position
- `.GetStringSelection()` : returns text of selected item
- `.SetInsertionPoint(pos)` : sets insertion point
- `.SetInsertionPointEnd()` : sets insertion point at end of control's text

wx.ListBox


```
wx.ListBox(parent,
                  id, choices=[], pos=(-1, -1),
                  size=(-1, -1), style=0,
                  validator=wx.DefaultValidator,
                  name="button")
```

- Can create with choices loaded, or add later
- Can report index or string selection(s)
- Again a number of style choices can be made
  - wx.LB_SINGLE, wx.LB_MULTIPLE
  - Styles wx.LB_NEEDED_SB, wx.LB_ALWAYS_SB control vertical scrollbar presence

wx.ListBox Methods

- `.Clear()` : removes all entries
- `.Append(s)` : adds an *s* entry at the end of the list
- `.GetSelection()` : returns index of current selection
- `.GetStringSelection()` : returns current selection
- `.GetSelections()` : returns list of selection indexes
  - When wx.TB_MULTILINE style is asserted
- Many other methods
  - Many of them inherited from wx.ControlWithItems

wx.Choice


```
wx.Choice(parent,
                  id, choices=[], pos=(-1, -1),
                  size=(-1, -1), style=0,
                  validator=wx.DefaultValidator,
                  name="text")
```

- Pulldown choice from a list
- For once, no special style options at all!
- By default control appears with empty selection
  - Helpful to user to select something initially
    - Otherwise user just sees a blank pulldown

wx.Choice Methods

- `.GetStringSelection()` : returns current selection
- `.GetSelection()` : returns index of current selection
- This also is a wx.ControlWithItems subclass
  - Much behaviour is common with wx.ListBox
    - One or two minor inconsistencies
- This control can be multi-column on some platforms
  - But better not to rely on non-cross-platform features

wx.StaticText

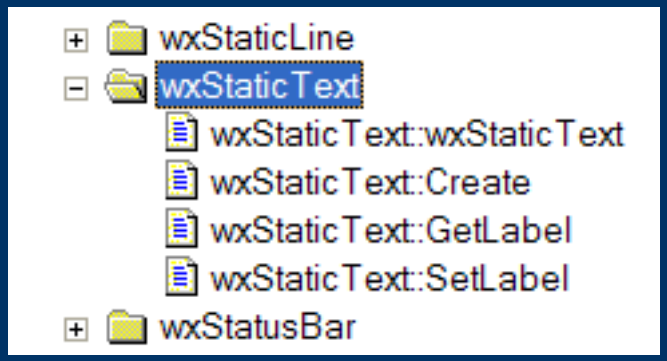

```
wx.ListBox(parent,
                  id, label="", pos=(-1, -1),
                  size=(-1, -1), style=0,
                  validator=wx.DefaultValidator,
                  name="button")
```

- Simple display of text
- `.GetLabel()` : returns current string value
- `.SetLabel(s)` : sets string value to given argument
- Styles: wx.ALIGN_{LEFT, RIGHT, CENTRE}
  - If changing will also need wx.ST_NO_AUTORESIZE

There is an Inheritance Graph

- Only locally-implemented methods documented
- There *will* be *many* inherited methods too!

Documentation Index:



A simple static text object has 252 methods and attributes!

What the programmer sees:

```
>>> dir(someStaticText)
['AcceptsFocus', 'AcceptsFocusFromKeyboard',
'AddChild', 'AddPendingEvent', 'AssociateHandle',
'Bind', 'CacheBestSize', 'CaptureMouse', 'Center',
'CenterOnParent', 'CenterOnScreen', 'Centre',
'CentreOnParent', 'CentreOnScreen',
'ClearBackground',
...
'GetLabel',
...
'SetLabel',
...
'TransferDataToWindow', 'Unbind',
'UnregisterHotKey', 'Update', 'UpdateWindowUI',
'UseBgCol', 'Validate', 'WarpPointer', '__class__',
'__delattr__', '__dict__', '__doc__',
'__getattribute__', '__hash__', '__init__',
'__module__',
'__new__', '__reduce__', '__reduce_ex__',
'__repr__', '__setattr__', '__str__', '__weakref__',
'_setOORInfo', 'this', 'thisown']
>>> len(dir(someStaticText))
252
>>>
```

Learn the Documentation

- There are dozens of different wxPython controls
  - Some of them offer features historically available from other Python libraries
  - Most Python users keep using what they already use
    - There is no need to completely rewrite your app
- Windows conveniently formats as a .chm
- Now a separate download from the code
  - Sign that wxPython is maturing?
  - Production run-time install does not need docs!

Events are of Different Types

- **wxEvent** The event base class
- **wxActivateEvent** A window or application activation event
- **wxCloseEvent** A close window or end session event
- **wxEraseEvent** An erase background event
- **wxFocusEvent** A window focus event
- **wxKeyEvent** A keypress event
- **wxIdleEvent** An idle event
- **wxInitDialogEvent** A dialog initialisation event
- **wxJoystickEvent** A joystick event
- **wxMenuEvent** A menu event
- **wxMouseEvent** A mouse event
- **wxMoveEvent** A move event
- **wxPaintEvent** A paint event
- **wxQueryLayoutInfoEvent** Used to query layout information
- **wxSetCursorEvent** Used for special cursor processing based on current mouse position
- **wxSizeEvent** A size event
- **wxScrollWinEvent** A scroll event sent by a scrolled window (not a scroll bar)
- **wxSysColourChangedEvent** A system colour change event
- ***Try not to worry: button clicks may be the only events you ever need to use!***

Event Model

- Bind handlers to events with procedure calls
  - Formerly EVT_XXX(control, handler)
  - Nowadays widget.Bind(event, handler)
- `EVT_XXX(window, func)`
- `EVT_XXX(window, ID, func)`
- `EVT_XXX(window, ID1, ID2, func)`
  - Choice would depend on type of event

```
btn.Bind(EVT_BUTTON, self.clickhandler)
```

  - This is the future, so we might as well use it!

Bindings are Dynamic

- MainLoop() examines bindings when window manager passes events to an App window
  - Event table determines which widget(s) receive the event
  - Analysis uses spatial position and z-ordering
- Event handlers get run "automagically"
  - Called from "inside" MainLoop()
  - This is the process of *event distribution*
  - Event handlers interact with application state
- Bindings can be dynamically modified

Handler Called with Event Argument

- If handler is a bound method it will also be bound to the particular instance:

```
class MyDialog(wx.Dialog):
                    ...
    def OnClick(self, event):
        # can use instance state to process
        # each instance in a specific way
                    ...
    btn = wx.Button(...)
                    ...
    btn.Bind(EVT_BUTTON, self.OnClick)
```

- Here there can be many instances of MyDialog
  - Each click will be handled by the instance whose button was clicked

Event Responses Should be Short

- They are called from MainLoop()
  - Delay here holds up responses to other events!
  - wx.Dialog.ShowModal() is an exception to this rule
    - It allows repainting &c of “suspended” windows
- There is typically not much to do anyway
  - Except in modal dialogs

Getting a Click From a User

```
# Use simple message dialog to display a message
d = wx.MessageDialog(frame, "Is it OK to proceed",
                           caption = "Message box",
                           style = wx.OK | wx.CANCEL)
res = d.ShowModal() # Holds everything up for response
if res=wx.ID_OK:
     # User wants to continue with action ...
```

- Style values affect user choices and defaults
  - Can show different choices of button(s)
  - Can vary the default for the dialog
  - Can change the displayed icon (for MS Windows)

MainLoop Searches for a Handler

- The search is potentially complex
  - But usually simple in practice
- A handler can call event.Skip() to pass on event
  - If so, MainLoop will continue the search for a handler
- The handler should avoid “stalling” the GUI
  - e.g. time.sleep(5) ...
  - No further events are processed until handler returns

Sizers Take the Strain of Layout

- Drop components into a sizer
  - They will be laid out in a predictable way
- For complex layouts, drop-in components can be pre-built sizers as well as atomic controls
  - Group and regroup components before committing to a design
  - You need to be able to think visually both *top-down* and *bottom-up* to capture your design

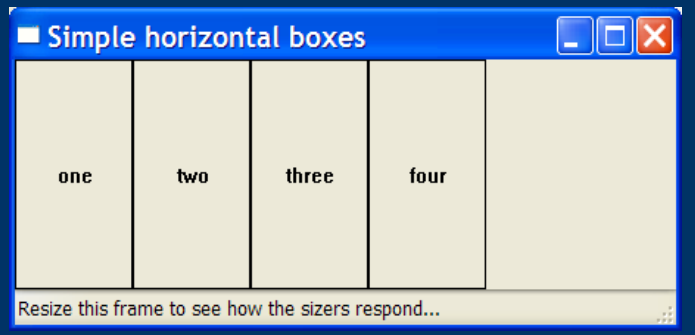
There are Three Types of Sizer

- Box Sizers
  - wx.BoxSizer(x)
    for x in (wx.HORIZONTAL, wx.VERTICAL)
- GridSizers
  - wx.GridSizer
  - wx.FlexGridSizer
- *[No, NEVER User the]* wx.GridBagSizer?
  - Has anyone, ever?

wx.BoxSizer(d)

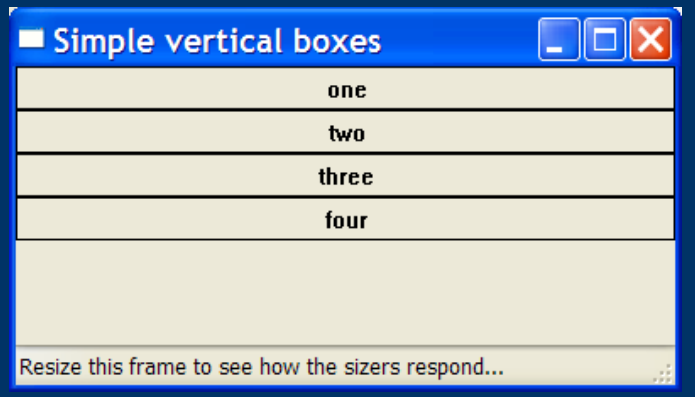
d = wx.HORIZONTAL

- **establishes *widths* of controls**
- **adds controls left-to-right**
- **resizes *vertically***



d = wx.VERTICAL

- **establishes *heights* of controls**
- **adds controls top-to-bottom**
- **resizes horizontally**



BoxSizer Creation and Population

```
class MyFrame(wx.Frame):
    def __init__(self, parent, id, title,
                 pos=wx.DefaultPosition,
                 size=wx.DefaultSize,
                 style=wx.DEFAULT_FRAME_STYLE):
        wx.Frame.__init__(self, parent, id, title, pos, size, style)
        b1 = wx.Button(self, -1, "Button 1") # No positions on
        b2 = wx.Button(self, -1, "Button 2") # these widgets
        s = wx.BoxSizer(wx.VERTICAL)
        s.Add(b1)
        s.Add(b2)
        self.SetSizer(s)
        self.SetSizeHints(parent)
```

- This example is not sophisticated
  - Handles window resize rather poorly
  - But sizers let you specify how it *should* be handled

Adding Items to Sizers

- Sizer.Add() has different signatures – summary
  - s.Add(window_or_sizer, flags)
  - s.Add(window_or_sizer, proportion, flags, border)
- *Window_or_sizer* can be
  - A frame
  - A control
  - Another sizer
  - An [x, y] s*pacer*, used only to adjust layout
- Sizer.Insert() and Sizer.Prepend() are similar

Sizer.Add() Flags

- wx.TOP, wx.BOTTOM, wx.LEFT, wx.RIGHT, wx.ALL
  - specify where to add border (if non-zero)
- wx.EXPAND
  - Says whether item expands in the direction size is not fixed by the sizer
- wx.SHAPED
  - maintains an object's aspect ratio when expanding

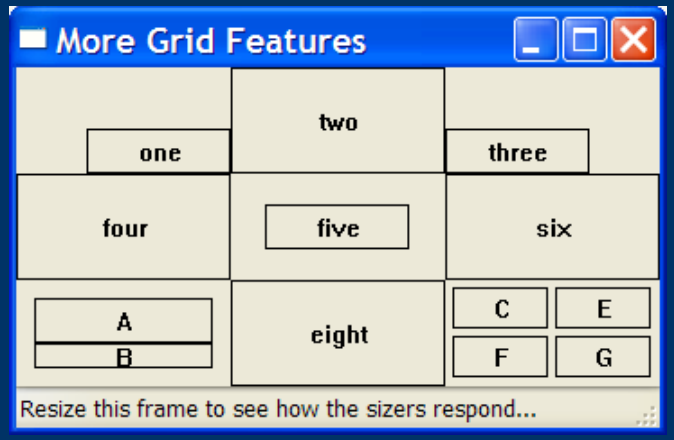
Sizer.Add() Flags (2)

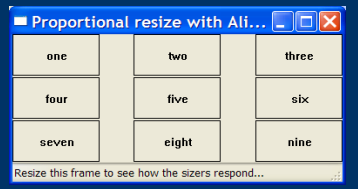
- wx.FIXED_MINSIZE
  - inhibits resizing
- wx.ALIGN_CENTER, wx.ALIGN_LEFT, wx.ALIGN_RIGHT, wx.ALIGN_TOP, wx.ALIGN_BOTTOM, wx.ALIGN_CENTER_VERTICAL, wx.ALIGN_CENTER_HORIZONTAL
  - specify alignment within the space allotted to items

wx.GridSizer(), wx.FlexGridSizer()

GridSizer

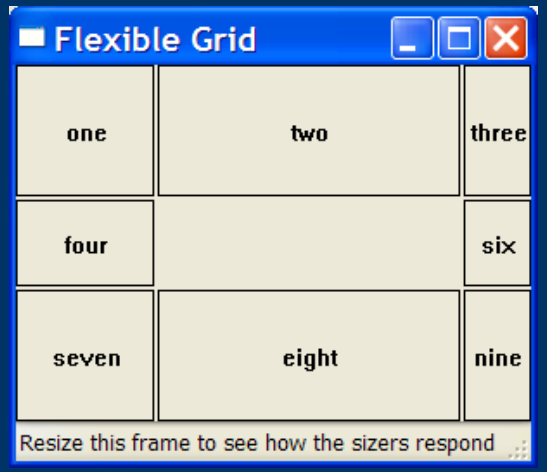
- **ALL cells are the same size**
  - **widest control sets column width**
  - **highest control sets column height**





FlexGridSizer

- **Each row and each column independently sized**



Use the wxPython Demo

- Many good sizer examples
- You can run it as often as you like
- It's even worth reading (and, if you want, stealing!) the code
- Lots of other useful snippets in there!
- Well-programmed application
- Good idea of current wxPython practices
- Reasonably well maintained
- Sometimes the *only* documentation on a recent feature

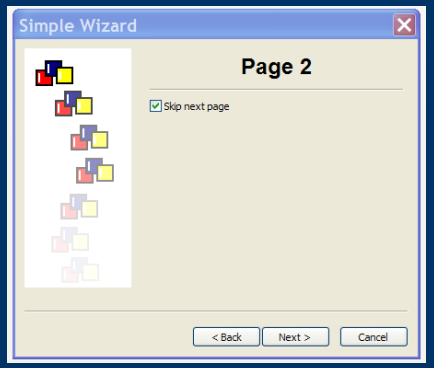
Boxes vs Grids vs Notebooks (1)

- Some people just “think in tables”
- Others find visual analysis easier
- This course *isn't* about aesthetics

Conclusion: DO WHAT WORKS

Boxes vs Grids vs Notebooks (2)

- Notebooks help keep window sizes down
  - Effectively, each tab adds “interface real estate”
  - Group together “similar” pieces of functionality
  - Buttons can apply over all notebook pages
- ***... and don't forget the wizards!!!!***



Composing Window Designs

- “Column of buttons”
  s=BoxSizer(v); for button in ...: s.Add(button)
- “Row of buttons”
  same but use BoxSizer(h)
- Rows and columns of other things
- Splitter windows are useful
  - wxPython recently added multi-split support

Putting It All Together

This class deliberately has relatively few exercises based on sample code

While hands-on experience is important we have focused on advancing your understanding

Your instructor will now discuss either or both of

a) some of the instructor's own code

b) some of the wxPython samples

Final Notes

- Thanks for coming to class!
- You don't know it all
  - Your instructor doesn't know it all either ...
- It takes time and patience to learn how to build GUIs
  - The knowledge from this class should get you started
  - Nothing to fear from wxPython code!
- Good luck, and keep in touch!!

steve@holdenweb.com

Bonus: Canned Dialogs List

ColourDialog
DirDialog
FileDialog
FindReplaceDialog
FontDialog
MessageDialog
MultiChoiceDialog
PageSetupDialog
PrintDialog
ProgressDialog
SingleChoiceDialog
TextEntryDialog

Bonus: Other wxPython Resources

`http://yergler.net/talks/desktopapps_uk/`
Good application with three-phase create

`http://wiki.wxpython.org/index.cgi/DoubleBufferedDrawing`
Tutorial with nice code