



# WXPYTHON

*GUI TOOLKIT*

**tutorialspoint**  
SIMPLY EASY LEARNING

[www.tutorialspoint.com](http://www.tutorialspoint.com)



<https://www.facebook.com/tutorialspointindia>



<https://twitter.com/tutorialspoint>

## About the Tutorial

---

wxPython is a blend of wxWidgets and Python programming library. This introductory tutorial provides the basics of GUI programming and helps you create desktop GUI applications.

## Audience

---

This tutorial is designed for software programmers who are keen on learning how to develop GUI applications for the desktop.

## Prerequisites

---

You should have a basic understanding of computer programming terminologies. A basic understanding of Python and any of the programming languages is a plus.

## Disclaimer & Copyright

---

© Copyright 2015 by Tutorials Point (I) Pvt. Ltd.

All the content and graphics published in this e-book are the property of Tutorials Point (I) Pvt. Ltd. The user of this e-book is prohibited to reuse, retain, copy, distribute or republish any contents or a part of contents of this e-book in any manner without written consent of the publisher.

We strive to update the contents of our website and tutorials as timely and as precisely as possible, however, the contents may contain inaccuracies or errors. Tutorials Point (I) Pvt. Ltd. provides no guarantee regarding the accuracy, timeliness or completeness of our website or its contents including this tutorial. If you discover any errors on our website or in this tutorial, please notify us at [contact@tutorialspoint.com](mailto:contact@tutorialspoint.com).

## Table of Contents

---

About the Tutorial.....	i
Audience .....	i
Prerequisites .....	i
Disclaimer & Copyright.....	i
Table of Contents .....	ii
1. WXPYTHON – INTRODUCTION.....	1
2. ENVIRONMENT.....	2
Windows .....	2
Linux .....	2
MacOS.....	2
3. HELLO WORLD.....	3
4. WX.FRAME CLASS.....	5
Window Style Constants .....	5
wx.Frame Class Member Functions .....	6
wx.Frame event binders.....	6
5. WX.PANEL CLASS.....	7
6. GUI BUILDER TOOLS .....	8
7. MAJOR CLASSES .....	12
8. EVENT HANDLING.....	15
9. LAYOUT MANAGEMENT .....	20
10. WX.BOXSIZER .....	21

11. WX.GRIDSIZER .....	26
12. WX.FLEXGRIDSIZER .....	29
13. WX.GRIDBAGSIZER .....	32
14. WX.STATICBOXSIZER.....	35
15. BUTTONS.....	38
16. WX.STATICTEXT CLASS.....	43
17. WX.TEXTCTRL CLASS.....	47
18. WX.RADIOBUTTON & WX.RADIOBOX .....	51
19. WX.CHECKBOX CLASS .....	55
20. WX.COMBOBOX & WX.CHOICE CLASS .....	57
21. WX.GAUGE CLASS.....	61
22. WX.SLIDER CLASS .....	64
23. MENU ITEM, MENU & MENUBAR.....	67
24. WX.TOOLBAR CLASS .....	72
25. WX.DIALOG CLASS .....	76
<b>MessageDialog</b> .....	<b>77</b>
<b>wx.TextEntryDialog</b> .....	<b>79</b>
<b>wx.FileDialog Class</b> .....	<b>82</b>
<b>wx.FontDialog Class</b> .....	<b>86</b>
26. WX.NOTEBOOK CLASS .....	89

27. DOCKABLE WINDOWS .....	93
28. MULTIPLE DOCUMENT INTERFACE.....	96
29. WX.SPLITTERWINDOW CLASS.....	98
30. DRAWING API.....	101
wx.Colour Class .....	101
wx.Pen Class .....	102
wx.Brush Class .....	102
31. WX.HTMLWINDOW CLASS.....	105
32. LISTBOX & LISTCTRL CLASS .....	107
33. DRAG AND DROP.....	113

# 1. wxPython – Introduction

**wxPython** is a Python wrapper for **wxWidgets** (which is written in C++), a popular cross-platform GUI toolkit. Developed by Robin Dunn along with Harri Pasanen, wxPython is implemented as a Python extension module.

Just like wxWidgets, wxPython is also a free software. It can be downloaded from the official website <http://wxpython.org>. Binaries and source code for many operating system platforms are available for download on this site.

Principal modules in wxPython API include a core module. It consists of **wxObject** class, which is the base for all classes in the API. Control module contains all the widgets used in GUI application development. For example, wx.Button, wx.StaticText (analogous to a label), wx.TextCtrl (editable text control), etc.

wxPython API has GDI (Graphics Device Interface) module. It is a set of classes used for drawing on widgets. Classes like font, color, brush, etc. are a part of it. All the container window classes are defined in Windows module.

Official website of wxPython also hosts Project Phoenix – a new implementation of wxPython for Python 3.\*. It focuses on improving speed, maintainability, and extensibility. The project began in 2012 and is still in beta stage.

## 2. Environment

### Windows

---

Prebuilt binaries for Windows OS (both 32 bit and 64 bit) are available on <http://www.wxpython.org/download.php> page. Latest versions of installers available are:

[wxPython3.0-win32-3.0.2.0-py27.exe](#) for 32-bit Python 2.7

[wxPython3.0-win64-3.0.2.0-py27.exe](#) for 64-bit Python 2.7

wxPython demo, samples and wxWidgets documentation is also available for download on the same page.

[wxPython3.0-win32-docs-demos.exe](#)

### Linux

---

wxPython binaries for many Linux distros can be found in their respective repositories. Corresponding package managers will have to be used to download and install. For instance on Debian Linux, following command should be able to install wxPython.

```
sudo apt-get install python-wxgtk3.0
```

### MacOS

---

Prebuilt binaries for MacOS in the form of disk images are available on the download page of the official website.

# 3. Hello World

A simple GUI application displaying Hello World message is built using the following steps:

- Import wx module.
- Define an object of Application class.
- Create a top level window as object of wx.Frame class. Caption and size parameters are given in constructor.
- Although other controls can be added in Frame object, their layout cannot be managed. Hence, put a Panel object into the Frame.
- Add a StaticText object to display 'Hello World' at a desired position inside the window.
- Activate the frame window by show() method.
- Enter the main event loop of Application object.

```
import wx

app = wx.App()
window = wx.Frame(None, title="wxPython Frame", size=(300,200))
panel=wx.Panel(window)
label=wx.StaticText(panel, label="Hello World", pos=(100,50))
window.Show(True)
app.MainLoop()
```

The above code produces the following output:





**wxFrame object** is the most commonly employed top level window. It is derived from **wxWindow class**. A frame is a window whose size and position can be changed by the user. It has a title bar and control buttons. If required, other components like menu bar, toolbar and status bar can be enabled. A wxFrame window can contain any frame that is not a dialog or another frame.

## 4. wx.Frame Class

**wx.Frame Class** has a default constructor with no arguments. It also has an overloaded constructor with the following parameters:

```
Wx.Frame (parent, id, title, pos, size, style, name)
```

Parameter	Description
Parent	Window parent. If 'None' is selected the object is at the top level window. If 'None' is not selected, the frame appears on top of the parent window
id	Window identifier. Usually -1 to let the identifier be generated automatically
Title	Caption to appear in the title bar
Pos	The starting position of the frame. If not given, wxDefaultPosition is as decided by OS
Size	Dimensions of the window. wxDefaultSize is decided by OS
style	Appearance of the window controlled by style constants
name	The internal name of object

### Window Style Constants

wx.DEFAULT_FRAME_STYLE
wx.CAPTION
wx.MINIMIZE_BOX
wx.MAXIMIZE_BOX
wx.CLOSE_BOX
wx.SYSTEM_MENU
wx.RESIZE_BORDER
wx.STAY_ON_TOP
wx.FRAME_FLOAT_ON_PARENT

**wx.DEFAULT\_FRAME\_STYLE** is defined as:

wx.MINIMIZE\_BOX | wx.MAXIMIZE\_BOX | wx.RESIZE\_BORDER | wx.SYSTEM\_MENU | wx.CAPTION | wx.CLOSE\_BOX | wx.CLIP\_CHILDREN

## Example

```
window=wx.Frame(None, -1, "Hello", pos=(10,10), size=(300,200), style=
wxDEFAULT_FRAME_STYLE, name="frame")
```

## wx.Frame Class Member Functions

CreateStatusBar()	Creates the status bar at bottom of the window
CreateToolBar()	Creates the toolbar at the top or left of the window
GetMenuBar()	Gets reference to menu bar
GetStatusBar()	Gets reference to statusbar
SetMenuBar()	Displays the menu bar object in the frame
setStatusBar()	Associates the status bar object to the frame
SetToolBar()	Associates a toolbar object to the frame
SetStatusText()	Displays text on the status bar
Create()	Creates a frame with provided parameters
Centre()	Places the frame at the center of display
SetPosition()	Places the frame at given screen coordinates
SetSize()	Resizes the frame to given dimensions
SetTitle()	Inserts the given text in the title bar

## wx.Frame event binders

EVT_CLOSE	When the frame is being closed by the user clicking the close button or programmatically
EVT_MENU_OPEN	When a menu is about to be opened
EVT_MENU_CLOSE	When a menu has just been closed
EVT_MENU_HIGHLIGHT	When the menu item with the specified id has been highlighted

## 5. wx.Panel Class

Widgets such as button, text box, etc. are placed on a panel window. **wx.Panel class** is usually put inside a wxFrame object. This class is also inherited from wxWindow class.

Although controls can be manually placed on panel by specifying the position in screen coordinates, it is recommended to use a suitable layout scheme, called **sizer** in wxPython, to have better control over the placement and address the resizing issue.

In **wxPanel constructor**, the parent parameter is the wx.Frame object in which the panel is to be placed. Default value of id parameter is wx.ID\_ANY, whereas the default style parameter is wxTAB\_TRAVERSAL.

wxPython API has the following sizers, using which controls are added into a panel object:

wx.BoxSizer	Widgets are arranged in a vertical or horizontal box
wx.StaticBoxSizer	Adds a staticbox around the sizer
wx.GridSizer	One control each added in equal sized cells of a grid
wx.FlexGridSizer	Control added in cell grid can occupy more than one cell
wx.GridBagSizer	Controls explicitly positioned in a grid and spanning over more than one row and/or column

Sizer object is applied as the layout manager of the panel using SetSizer() method of wxPanel class.

```
wx.Panel.SetSizer(wx.???Sizer())
```

Panel object in turn is added to the top level frame.

## 6. GUI Builder Tools

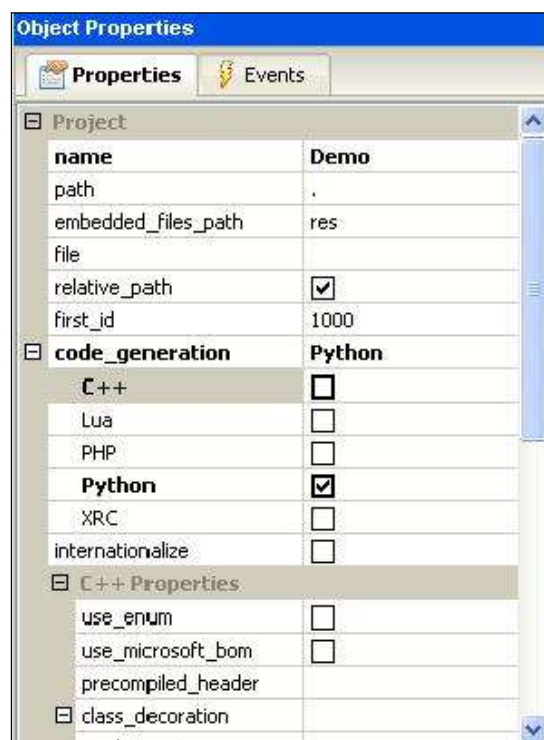
Creating a good looking GUI by manual coding can be tedious. A visual GUI designer tool is always handy. Many GUI development IDEs targeted at wxPython are available. Following are some of them:

- wxFormBuilder
- wxDesigner
- wxGlade
- BoaConstructor
- gui2py

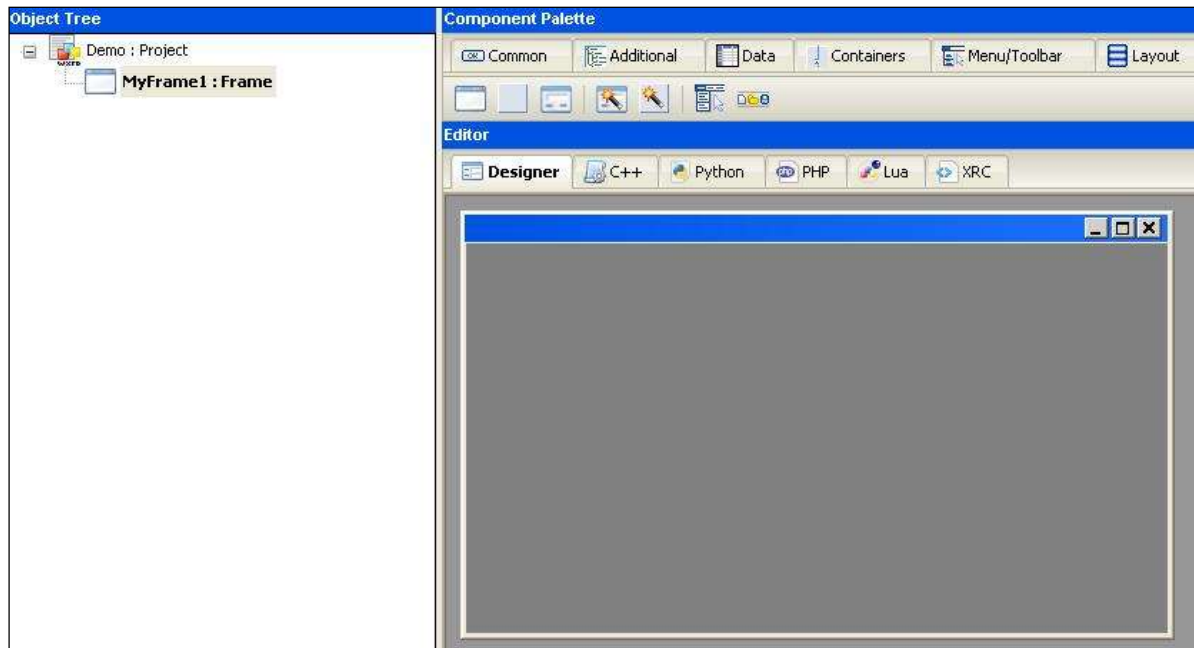
**wxFormBuilder** is an open source, cross-platform WYSIWYG GUI builder that can translate the wxWidget GUI design into C++, Python, PHP or XML format. A brief introduction to usage of wxFormBuilder is given here.

First of all the latest version of wxFormBuilder needs to be downloaded and installed from <http://sourceforge.net/projects/wxformbuilder/>. On opening the application, a new project with blank grey area at the center appears.

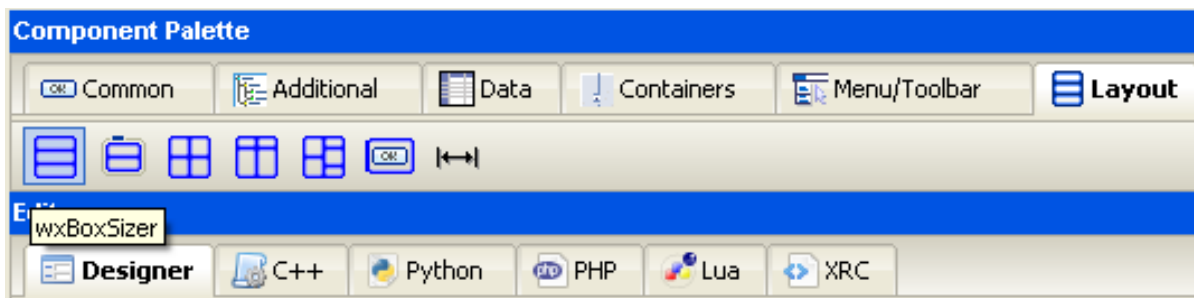
Give a suitable name to the project and choose Python as code generation language. This is done in the Object properties window as shown in the following image:



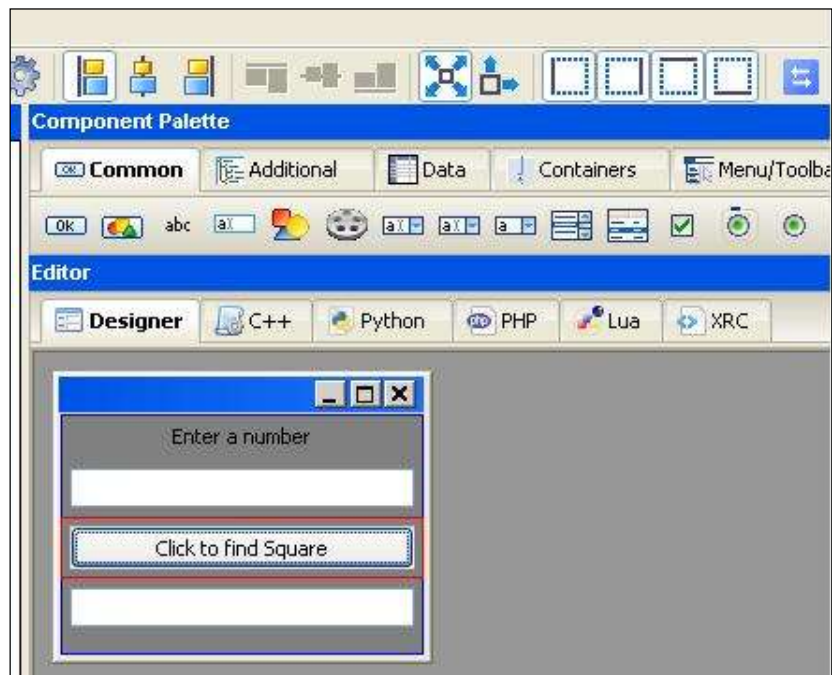
Then from 'Forms' tab of components palette, choose Frame.



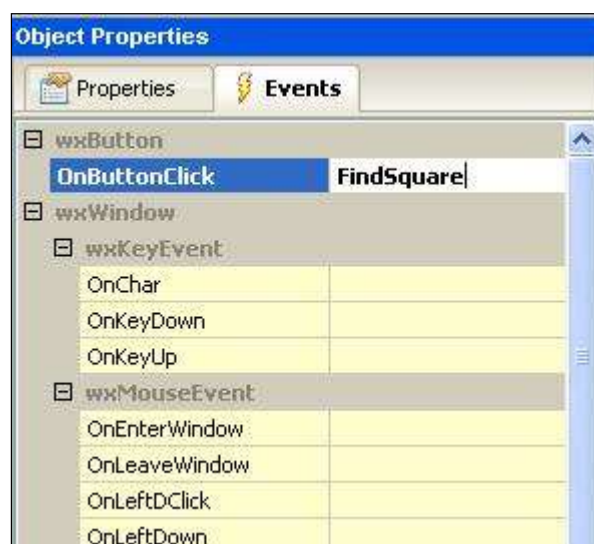
Add a vertical wxBoxSizer from 'Layouts' tab.



Add necessary controls in the Box with suitable captions. Here, a StaticText (label), two TextCtrl objects (text boxes) and a wxButton object are added. The frame looks like the following image:



Enable Expand and Stretch on these three controls. In the object properties for wxButton object, assign a function `findsquare()` to OnButtonClick event.



Save the project and press F8 to generate Python code for developed GUI. Let the generated file be named as Demo.py

In the executable Python script, import demo.py and define FindSquare() function. Declare Application object and start a main event loop. Following is the executable code:

```
import wx

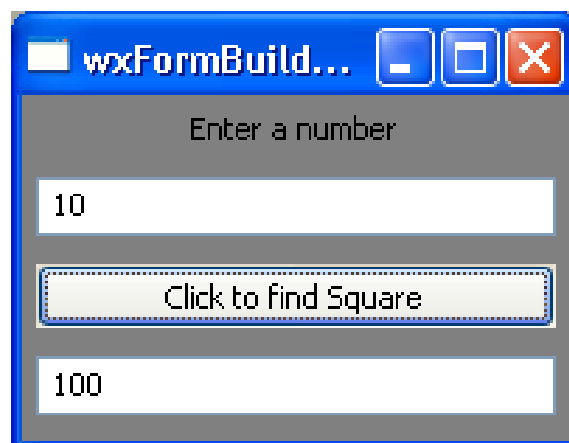
#import the newly created GUI file
import demo

class CalcFrame(demo.MyFrame1):
    def __init__(self,parent):
        demo.MyFrame1.__init__(self,parent)

    def FindSquare(self,event):
        num = int(self.m_textCtrl1.GetValue())
        self.m_textCtrl2.SetValue (str(num*num))

app = wx.App(False)
frame = CalcFrame(None)
frame.Show(True)
#start the applications
app.MainLoop()
```

The above code produces the following output:

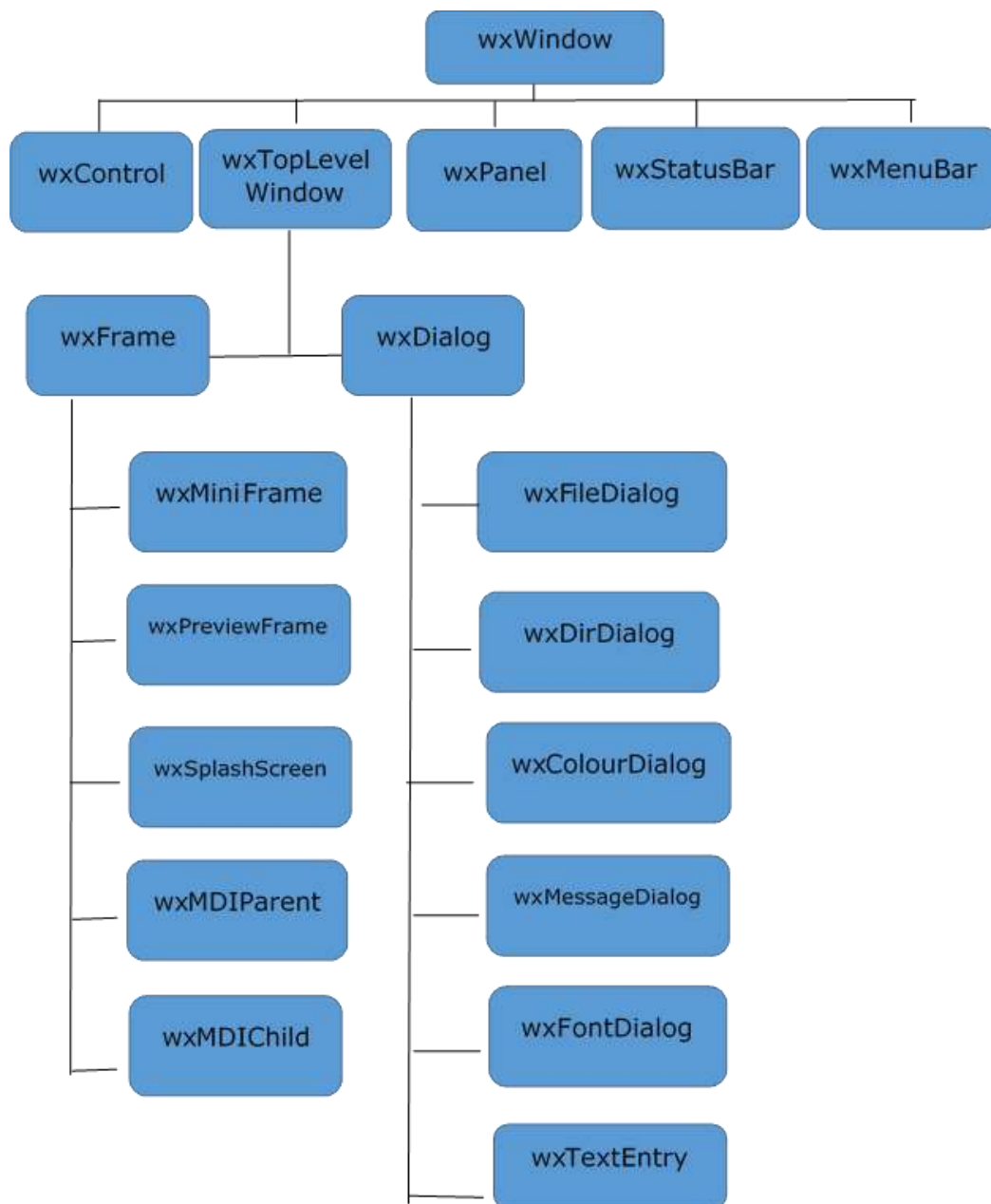


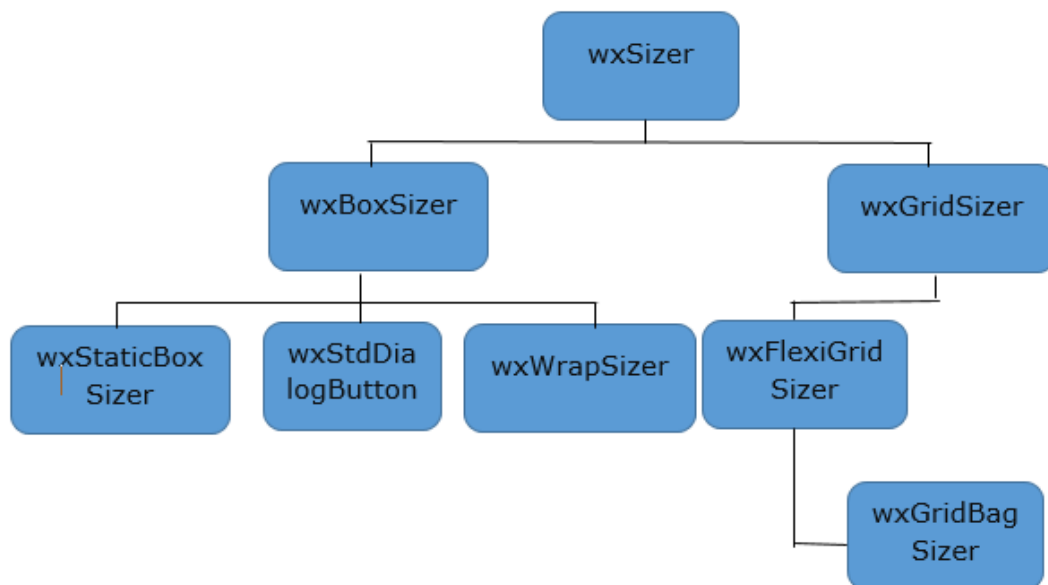
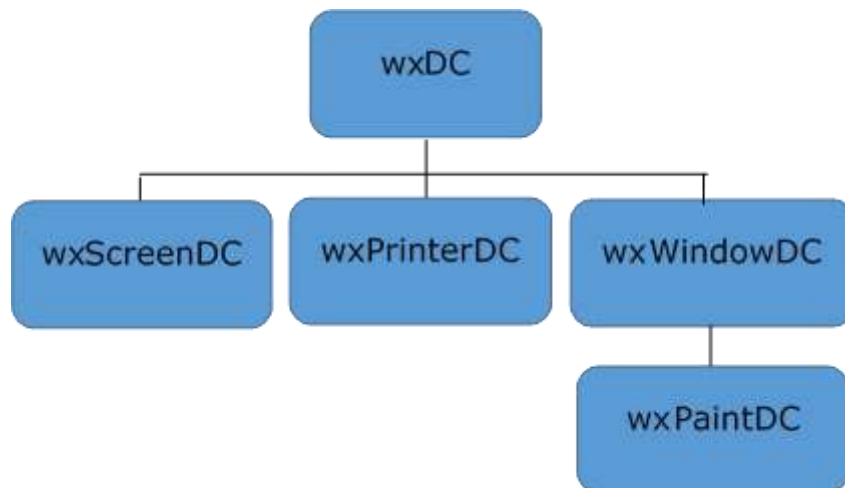
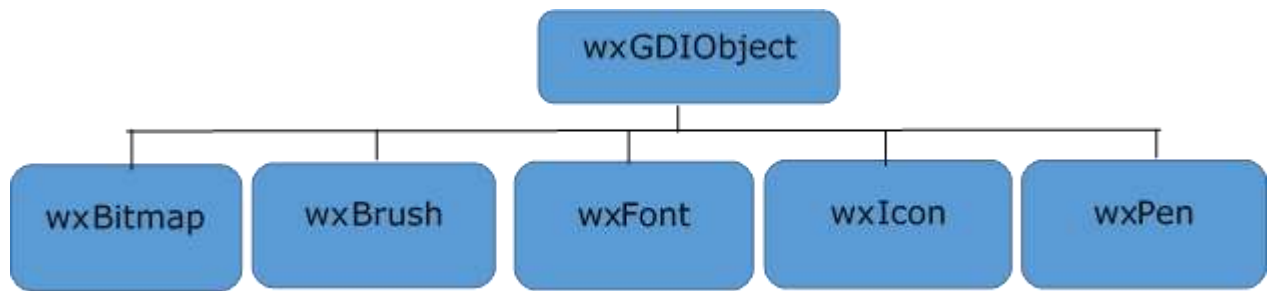


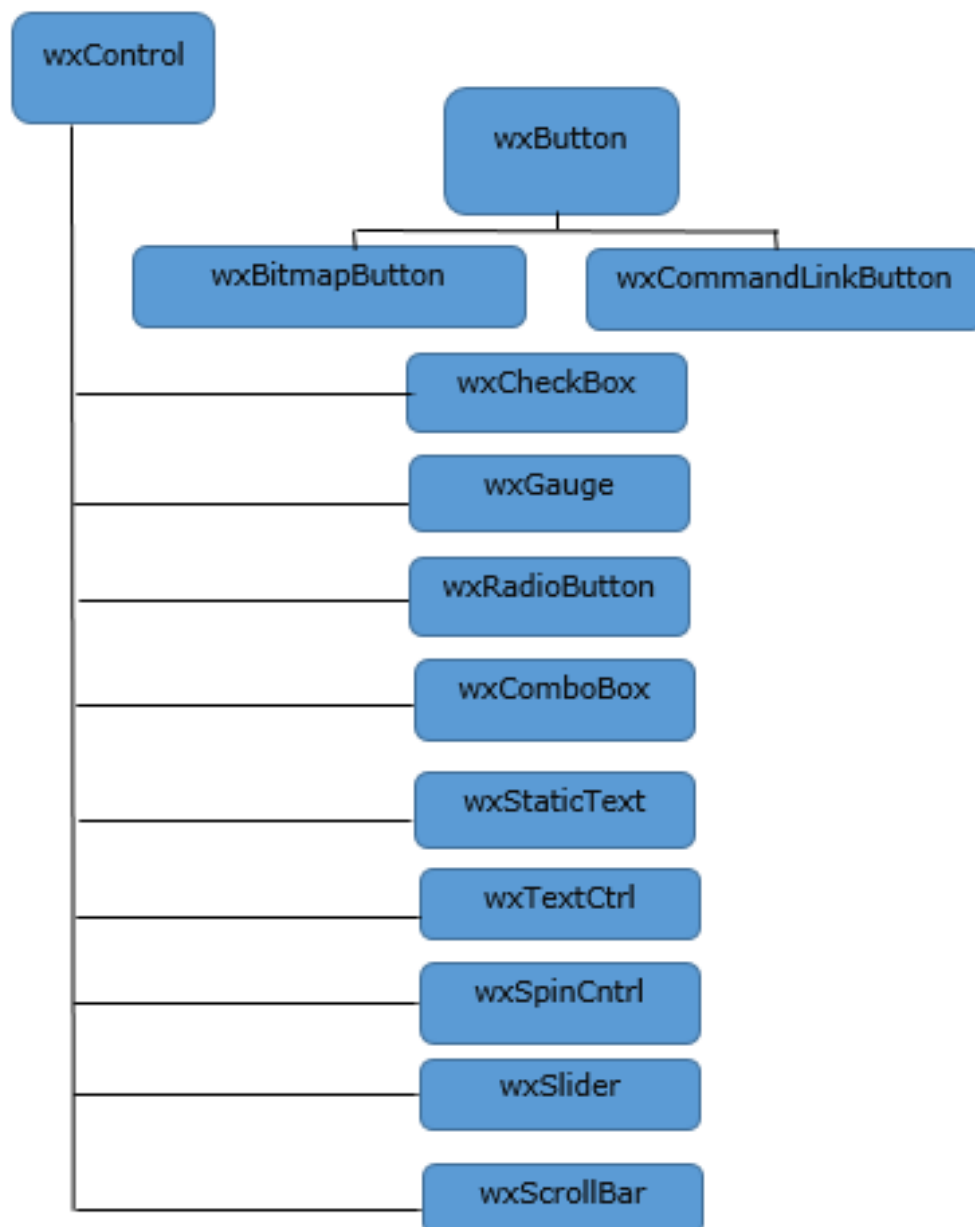
# 7. Major Classes

Original wxWidgets (written in C++) is a huge class library. GUI classes from this library are ported to Python with wxPython module, which tries to mirror the original wxWidgets library as close as possible. So, wx.Frame class in wxPython acts much in the same way as wxFrame class in its C++ version.

wxObject is the base for most of the classes. An object of wxApp (wx.App in wxPython) represents the application itself. After generating the GUI, application enters in an event loop by MainLoop() method. Following diagrams depict the class hierarchy of most commonly used GUI classes included in wxPython.







## 8. Event Handling

Unlike a console mode application, which is executed in a sequential manner, a GUI based application is event driven. Functions or methods are executed in response to user's actions like clicking a button, selecting an item from collection or mouse click, etc., called events.

Data pertaining to an event which takes place during the application's runtime is stored as object of a subclass derived from **wx.Event**. A display control (such as Button) is the source of event of a particular type and produces an object of Event class associated to it. For instance, click of a button emits a wx.CommandEvent. This event data is dispatched to event handler method in the program. wxPython has many predefined event binders. An **Event binder** encapsulates relationship between a specific widget (control), its associated event type and the event handler method.

For example, to call **OnClick() method** of the program on a button's click event, the following statement is required:

```
self.b1.Bind(EVT_BUTTON, OnClick)
```

**Bind() method** is inherited by all display objects from wx.EvtHandler class. EVT\_.BUTTON here is the binder, which associates button click event to OnClick() method.

### Example

In the following example, the MoveEvent, caused by dragging the top level window – a wx.Frame object in this case – is connected to **OnMove() method** using wx.EVT\_MOVE binder. The code displays a window. If it is moved using mouse, its instantaneous coordinates are displayed on the console.

```
import wx

class Example(wx.Frame):

    def __init__(self, *args, **kw):
        super(Example, self).__init__(*args, **kw)
        self.InitUI()

    def InitUI(self):
        self.Bind(wx.EVT_MOVE, self.OnMove)
        self.SetSize((250, 180))
        self.SetTitle('Move event')
        self.Centre()
        self.Show(True)
```

```

def OnMove(self, e):
    x, y = e.GetPosition()
    print "current window position x=",x," y=",y

ex = wx.App()
Example(None)
ex.MainLoop()

```

The above code produces the following output:



```

current window position x= 562 y= 309
current window position x= 562 y= 309
current window position x= 326 y= 304
current window position x= 384 y= 240
current window position x= 173 y= 408
current window position x= 226 y= 30
current window position x= 481 y= 80

```

Some of the subclasses inherited from wx.Event are listed in the following table:

wxKeyEvent	Occurs when a key is presses or released
wxPaintEvent	Is generated whenever contents of the window needs to be redrawn
wxMouseEvent	Contains data about any event due to mouse activity like mouse button pressed or dragged

wxScrollEvent	Associated with scrollable controls like wxScrollbar and wxSlider
wxCommandEvent	Contains event data originating from many widgets such as button, dialogs, clipboard, etc.
wxMenuEvent	Different menu-related events excluding menu command button click
wxColourPickerEvent	wxColourPickerCtrl generated events
wxDirFileickerevent	Events generated by FileDialog and DirDialog

Events in wxPython are of two types. Basic events and Command events. A basic event stays local to the window in which it originates. Most of the wxWidgets generate command events. A **command event** can be propagated to window or windows, which are above the source window in class hierarchy.

## Example

Following is a simple example of event propagation. The complete code is:

```
import wx

class MyPanel(wx.Panel):

    def __init__(self, parent):
        super(MyPanel, self).__init__(parent)

        b=wx.Button(self, label='Btn', pos=(100,100))
        b.Bind(wx.EVT_BUTTON, self.btnclick)
        self.Bind(wx.EVT_BUTTON, self.OnButtonClicked)

    def OnButtonClicked(self, e):

        print 'Panel received click event. propagated to Frame class'
        e.Skip()

    def btnclick(self,e):
        print "Button received click event. propagated to Panel class"
        e.Skip()

class Example(wx.Frame):
```

```

def __init__(self,parent):
    super(Example, self).__init__(parent)

    self.InitUI()

def InitUI(self):

    mpnl = MyPanel(self)
    self.Bind(wx.EVT_BUTTON, self.OnButtonClicked)

    self.SetTitle('Event propagation demo')
    self.Centre()
    self.Show(True)

def OnButtonClicked(self, e):

    print 'click event received by frame class'
    e.Skip()

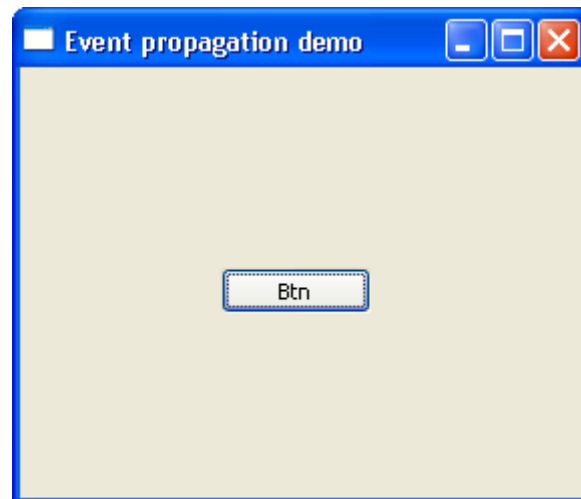
ex = wx.App()
Example(None)
ex.MainLoop()

```

In the above code, there are two classes. **MyPanel**, a wx.Panel subclass and Example, a wx.Frame subclass which is the top level window for the program. A button is placed in the panel.

This Button object is bound to an event handler btnclk() which propagates it to parent class (MyPanel in this case). Button click generates a **CommandEvent** which can be propagated to its parent by Skip() method.

MyPanel class object also binds the received event to another handler OnButtonClicked(). This function in turn transmits to its parent, the Example class. The above code produces the following output:



Button received click event. Propagated to Panel class.

Panel received click event. Propagated to Frame class.

Click event received by frame class.



## 9. Layout Management

A GUI widget can be placed inside the container window by specifying its absolute coordinates measured in pixels. The coordinates are relative to the dimensions of the window defined by size argument of its constructor. Position of the widget inside the window is defined by **pos** argument of its constructor.

```
import wx

app = wx.App()
window = wx.Frame(None, title="wxPython Frame", size=(300,200))
panel=wx.Panel(window)
label=wx.StaticText(panel, label="Hello World", pos=(100,50))
window.Show(True)
app.MainLoop()
```

This **Absolute Positioning** however is not suitable because of the following reasons:

- The position of the widget does not change even if the window is resized.
- The appearance may not be uniform on different display devices with different resolutions.
- Modification in the layout is difficult as it may need redesigning the entire form.

wxPython API provides Layout classes for more elegant management of positioning of widgets inside the container. The advantages of Layout managers over absolute positioning are:

- Widgets inside the window are automatically resized.
- Ensures uniform appearance on display devices with different resolutions.
- Adding or removing widgets dynamically is possible without having to redesign.

Layout manager is called Sizer in wxPython. Wx.Sizer is the base class for all sizer subclasses. Let us discuss some of the important sizers such as wx.BoxSizer, wx.StaticBoxSizer, wx.GridSizer, wx.FlexGridSizer, and wx.GridBagSizer.

# 10. wx.BoxSizer

This sizer allows the controls to be arranged in row-wise or column-wise manner. **BoxSizer's** layout is determined by its orientation argument (either wxVERTICAL or wxHORIZONTAL).

Box=wx.BoxSizer(wxHORIZONTAL)
-------------------------------

Box=wx.BoxSizer(wxVERTICAL)
-----------------------------

Add() method (inherited from wxSizer) appends it to the next row/column of the sizer.

Box.Add(control, proportion, flag, border)
--

The proportion parameter controls how the control changes its size in response to dimensions of the container. Combination of various flag parameters decides the appearance of control in the sizer. Following are some of the flags:

## Alignment Flags

wx.ALIGN_TOP
--------------

wx.ALIGN_BOTTOM
-----------------

wx.ALIGN_LEFT
---------------

wx.ALIGN_RIGHT
----------------

wx.ALIGN_CENTER_VERTICAL
--------------------------

wx.ALIGN_CENTER_HORIZONTAL
----------------------------

## Border Flags

wx.TOP
--------

wx.BOTTOM
-----------

wx.LEFT
---------

wx.RIGHT
----------

wx.ALL
--------

## Behavior Flags

wx.EXPAND	Item will expand to fill the space provided to it (wx.GROW is the same)
wx.SHAPED	Similar to EXPAND but maintains the item's aspect ratio
wx.FIXED_MINSIZE	Does not let the item become smaller than its initial minimum size
wx.RESERVE_SPACE_EVEN_IF_HIDDEN	Does not allow the sizer to reclaim an item's space when it is hidden

The border parameter is an integer, the space in pixels to be left between controls. For example,

```
b=wx.StaticText(self, -1, "Enter a number")
Box.Add(b,1,wx.ALL|wx.EXPAND,10)
```

Following are some more methods of wx.BoxSizer class:

SetOrientation()	Sets orientation wxHORIZONTAL or wxVERTICAL
AddSpacer()	Adds nonstretchable space
AddStretchSpacer()	Adds stretchable space so that resizing the window will affect control size proportionately
Clear()	Removes children from sizer
Detach()	Removes a control from sizer without destroying
Insert()	Inserts a child control at a specified position
Remove()	Removes a child from sizer and destroys it

## Example

In the following code, a vertical box sizer is applied to a panel object which is placed inside wxFrame window.

```
p=wx.Panel(self)
vbox=wx.BoxSizer(wx.VERTICAL)
```

The first row in the box displays a label (wx.StaticText object) in the center with a border of 20 pixels around it.

```
l1=wx.StaticText(p,label="Enter a number",style= wx.ALIGN_CENTRE )
vbox.Add(l1,0, wx.ALL|wx.EXPAND|wx.ALIGN_CENTER_HORIZONTAL, 20)
```

In the second row, a wx.Button object is displayed. Because of wx.EXPAND flag it occupies the entire width of the window.

```
b1=wx.Button(p, label="Btn1")
vbox.Add(b1,0,wx.EXPAND)
```

The next row also contains a button. It is not added with EXPAND flag. Instead, because of ALIGN\_CENTER\_HORIZONTAL, button with default size appears in the center horizontally.

```
b2=wx.Button(p, label="Btn2")
vbox.Add(b2,0,wx.ALIGN_CENTER_HORIZONTAL)
```

In the next row, a TextCtrl object with proportion parameter set to 1 and EXPAND flag set is added. As a result, it is taller in size.

```
t=wx.TextCtrl(p)
vbox.Add(t,1,wx.EXPAND,10)
```

The last row holds a horizontal sizer object, which in turn has a label and button separated by a stretchable space.

```
hbox=wx.BoxSizer(wx.HORIZONTAL)
l2=wx.StaticText(p,label="Label2",style= wx.ALIGN_CENTRE)
hbox.Add(l2,0,wx.EXPAND)
b3=wx.Button(p,label="Btn3")
hbox.AddStretchSpacer(1)
hbox.Add(b3,0,wx.ALIGN_LEFT,20)
vbox.Add(hbox,1,wx.ALL|wx.EXPAND)
```

Lastly, the vertical box sizer is applied to wx.Panel object.

Following is the complete code:

```
import wx

class Example(wx.Frame):

    def __init__(self, parent, title):
        super(Example, self).__init__(parent, title=title, size=(200,300))

        self.InitUI()
        self.Centre()
        self.Show()
```

```

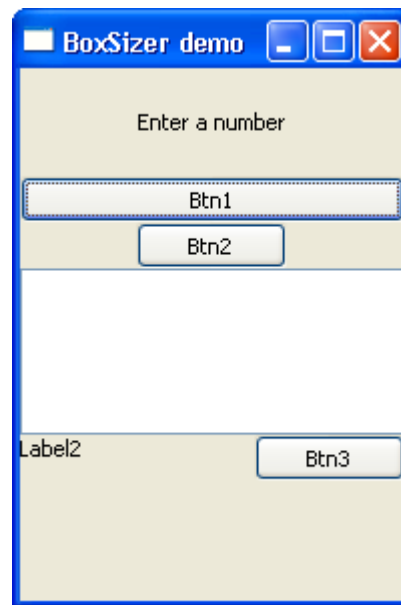
def InitUI(self):
    p=wx.Panel(self)
    vbox=wx.BoxSizer(wx.VERTICAL)
    l1=wx.StaticText(p,label="Enter a number",style= wx.ALIGN_CENTRE )
    vbox.Add(l1,0, wx.ALL|wx.EXPAND|wx.ALIGN_CENTER_HORIZONTAL, 20)
    b1=wx.Button(p, label="Btn1")
    vbox.Add(b1,0,wx.EXPAND)

    b2=wx.Button(p, label="Btn2")
    vbox.Add(b2,0,wx.ALIGN_CENTER_HORIZONTAL)
    t=wx.TextCtrl(p)
    vbox.Add(t,1,wx.EXPAND,10)
    hbox=wx.BoxSizer(wx.HORIZONTAL)
    l2=wx.StaticText(p,label="Label2",style= wx.ALIGN_CENTRE)
    hbox.Add(l2,0,wx.EXPAND)
    b3=wx.Button(p,label="Btn3")
    hbox.AddStretchSpacer(1)
    hbox.Add(b3,0,wx.ALIGN_LEFT,20)
    vbox.Add(hbox,1,wx.ALL|wx.EXPAND)
    p.SetSizer(vbox)

app = wx.App()
Example(None, title='BoxSizer demo')
app.MainLoop()

```

The above code produces the following output:



# 11. wx.GridSizer

As the name suggests, a **GridSizer** object presents a two dimensional grid. Controls are added in the grid slot in the left-to-right and top-to-bottom order. GridSizer object takes four parameters:

```
wx.GridSizer(rows, columns, vgap, hgap)
```

vgap and hgap parameters control vertical and horizontal spacing between the adjacent controls.

Following table shows some important methods of wxGridSizer class:

Add()	Adds a control in the next available grid slot
AddMany()	Adds each item in the list of controls
SetRows()	Sets the number of rows in the sizer
GetRows()	Retrieves the number of rows in the sizer
SetCols()	Sets the number of columns in the sizer
GetCols()	Retrieves the number of columns in size
SetVGap()	Sets vertical gap (in pixels) between the cells
GetVGap()	Returns the value of vgap between the cells
SetHGap()	Sets the horizontal gap (in pixels) between the cells
GetHGap()	Returns the value of hgap between the cells

The following code demonstrates a simple gridsizer of a 4 by 4 grid with vertical and horizontal gap of 5 pixels.

```
Gs = wx.GridSizer(4, 4, 5, 5)
```

Sixteen button objects are successively added using a 'for' loop.

```
for i in range(1,17):  
    btn="Btn"+str(i)  
    gs.Add(wx.Button(p,label=btn),0,wx.EXPAND)
```

The complete code is as follows:

```
import wx

class Example(wx.Frame):

    def __init__(self, parent, title):
        super(Example, self).__init__(parent, title=title, size=(300,200))

        self.InitUI()
        self.Centre()
        self.Show()

    def InitUI(self):

        p=wx.Panel(self)

        gs = wx.GridSizer(4, 4, 5, 5)
        for i in range(1,17):
            btn="Btn"+str(i)
            gs.Add(wx.Button(p,label=btn),0,wx.EXPAND)

        p.SetSizer(gs)

app = wx.App()
Example(None, title='Grid demo')
app.MainLoop()
```



The above code produces the following output:



## 12. wx.FlexiGridSizer

This sizer also has a two dimensional grid. However, it provides little more flexibility in laying out the controls in the cells. Although all the controls in the same row have the same height, and all the controls in the same column have the same width, the size of each cell is not uniform as in GridSizer.

Width and/or height of cells in a single column/row can be allowed to extend by AddGrowableRow() and AddGrowableCol() method.

**wx.FlexiGridSizer** class constructor takes four parameters:

```
wx.FlexiGridSizer(rows, cols, vgap, hgap)
```

A brief description of major methods of wx.FlexiGridSizer is given below:

AddGrowableCol()	Specifies a column of given index to grow if extra height is available
AddGrowRow()	Specifies a row of given index to grow if extra width is available
SetFlexibleDirection()	Specifies whether sizer's flexibility affects the row, the column or both

### Example

A two-column form is designed with the following code. The first column contains labels and the second contains text boxes. The second column is set to be growable. Similarly, the third row is set to be growable. (Note that the row index and the column index starts from 0). The second parameter in AddGrowableCol() and AddGrowableRow() function is the proportion of growth.

```
fgs.AddGrowableRow(2, 1)
fgs.AddGrowableCol(1, 1)
```

The entire code is as follows:

```
import wx

class Example(wx.Frame):

    def __init__(self, parent, title):
        super(Example, self).__init__(parent, title=title,
                                      size=(300, 250))

        self.InitUI()
```

```

        self.Centre()
        self.Show()

    def InitUI(self):
        panel = wx.Panel(self)

        hbox = wx.BoxSizer(wx.HORIZONTAL)

        fgs = wx.FlexGridSizer(3, 2, 10,10)

        title = wx.StaticText(panel, label="Title")
        author = wx.StaticText(panel, label="Name of the Author")
        review = wx.StaticText(panel, label="Review")

        tc1 = wx.TextCtrl(panel)
        tc2 = wx.TextCtrl(panel)
        tc3 = wx.TextCtrl(panel, style=wx.TE_MULTILINE)

        fgs.AddMany([(title), (tc1, 1, wx.EXPAND), (author),
                     (tc2, 1, wx.EXPAND), (review, 1, wx.EXPAND), (tc3, 1, wx.EXPAND)])

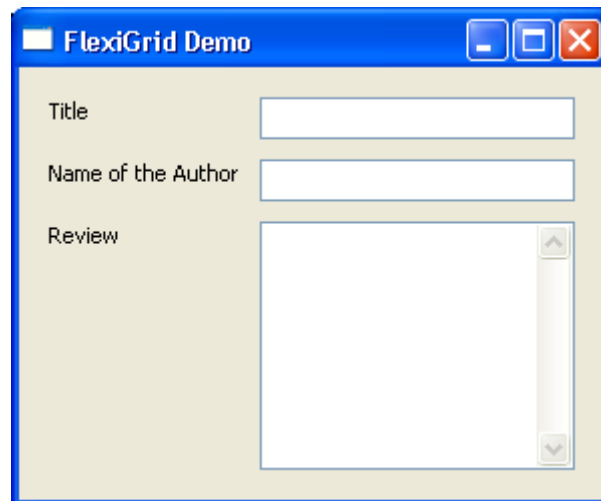
        fgs.AddGrowableRow(2, 1)
        fgs.AddGrowableCol(1, 1)

        hbox.Add(fgs, proportion=2, flag=wx.ALL|wx.EXPAND, border=15)
        panel.SetSizer(hbox)

app = wx.App()
Example(None, title='FlexiGrid Demo')
app.MainLoop()

```

The above code produces the following output:



A screenshot of a wxPython window titled "FlexiGrid Demo". The window has a blue title bar with standard minimize, maximize, and close buttons. The main content area has a light beige background. It contains three labels on the left: "Title", "Name of the Author", and "Review". To the right of "Title" and "Name of the Author" are single-line text input fields. To the right of "Review" is a multi-line text area with a vertical scrollbar on its right side.

# 13. wx.GridBagSizer

**GridBagSizer** is a versatile sizer. It offers more enhancements than FlexiGridSizer. **Child widget** can be added to a specific cell within the grid. Furthermore, a child widget can occupy more than one cell horizontally and/or vertically. Hence, a static text and multiline text control in the same row can have different width and height.

**Gridbag layout** should be meticulously planned by deciding the position, span and the gap. wx.GridBagSizer class has only one constructor taking two arguments.

```
Wx.GridBagSizer(vgap,hgap)
```

The most important method of GridBagsizer class is Add() which takes position as the mandatory argument. Span, alignment, border flags, and border size parameters are optional. If not explicitly used they assume default values.

```
Wx.GridbagSizer().Add(control, pos, span, flags, border)
```

The following table lists the methods of GridBagSizer class:

Add()	Adds given control at the specified position in the grid
GetItemPosition()	Returns the position of control in the grid
SetItemPosition()	Places a control at the specified position in the grid
GetItemSpan()	Returns row/column spanning of an item
SetItemSpan()	Spans the specified item over the number of rows/columns

The following code displays a form in which there are labels (StaticText) associated with textboxes (TexCtrl). TextCtrl objects are added with span parameter specified. Hence, the width of text boxes spans more than one column. Text box for name spans over two columns.

```
tc = wx.TextCtrl(panel)
sizer.Add(tc, pos=(0, 1), span=(1, 2), flag=wx.EXPAND|wx.ALL, border=5)
```

Textbox for address is a multiline text control spanning over three columns.

```
tc1 = wx.TextCtrl(panel,style=wx.TE_MULTILINE)
sizer.Add(tc1, pos=(1,1), span=(1, 3), flag=wx.EXPAND|wx.ALL, border=5)
```

The row containing multiline text control for description is set to be growable so that it expands vertically downwards, if the form is stretched.

```
tc4 = wx.TextCtrl(panel,style=wx.TE_MULTILINE)
sizer.Add(tc4, pos=(3,1), span=(1,3), flag=wx.EXPAND|wx.ALL, border=5)
```

```
sizer.AddGrowableRow(3)
```

Following is the complete code:

```
import wx

class Example(wx.Frame):

    def __init__(self, parent, title):
        super(Example, self).__init__(parent, title=title)

        self.InitUI()
        self.Centre()
        self.Show()

    def InitUI(self):

        panel = wx.Panel(self)
        sizer = wx.GridBagSizer(0,0)

        text = wx.StaticText(panel, label="Name:")
        sizer.Add(text, pos=(0, 0), flag=wx.ALL, border=5)

        tc = wx.TextCtrl(panel)
        sizer.Add(tc, pos=(0, 1), span=(1, 2), flag=wx.EXPAND|wx.ALL, border=5)

        text1 = wx.StaticText(panel, label="address")
        sizer.Add(text1, pos=(1, 0), flag=wx.ALL, border=5)
        tc1 = wx.TextCtrl(panel, style=wx.TE_MULTILINE)
        sizer.Add(tc1, pos=(1,1), span=(1, 3), flag=wx.EXPAND|wx.ALL, border=5)

        text2=wx.StaticText(panel, label="age")
        sizer.Add(text2, pos=(2, 0), flag=wx.ALL, border=5)
        tc2=wx.TextCtrl(panel)
        sizer.Add(tc2, pos=(2,1), flag=wx.ALL, border=5)
        text3=wx.StaticText(panel, label="Mob.No")
        sizer.Add(text3, pos=(2, 2), flag=wx.ALIGN_CENTER|wx.ALL, border=5)
```

```

tc3=wx.TextCtrl(panel)
sizer.Add(tc3, pos=(2,3),flag=wx.EXPAND|wx.ALL, border=5)

text4 = wx.StaticText(panel, label="Description")
sizer.Add(text4, pos=(3, 0), flag=wx.ALL, border=5)
tc4 = wx.TextCtrl(panel,style=wx.TE_MULTILINE)
sizer.Add(tc4, pos=(3,1), span=(1,3), flag=wx.EXPAND|wx.ALL, border=5)
sizer.AddGrowbleRow(3)

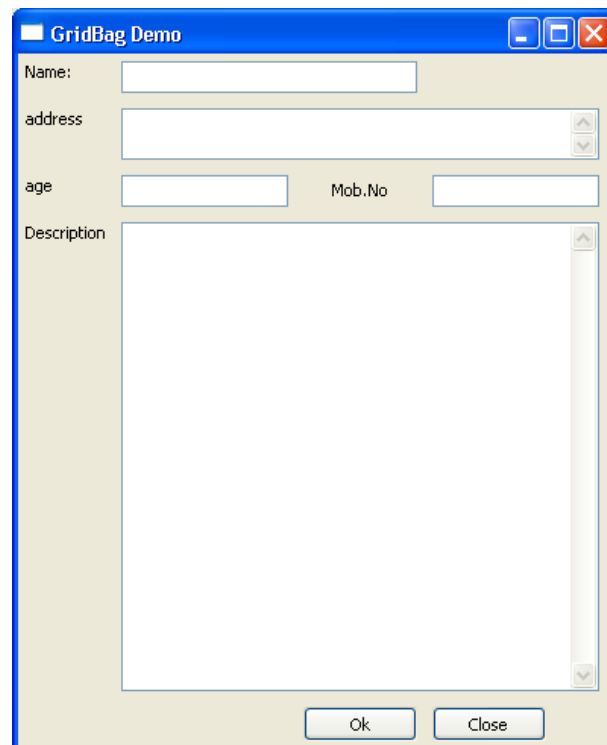
buttonOk = wx.Button(panel, label="Ok")
buttonClose = wx.Button(panel, label="Close" )
sizer.Add(buttonOk, pos=(4, 2),flag=wx.ALL, border=5)
sizer.Add(buttonClose, pos=(4, 3), flag=wx.ALL, border=5)

panel.SetSizerAndFit(sizer)

app = wx.App()
Example(None, title='GridBag Demo')
app.MainLoop()

```

The above code produces the following output:



# 14. wx.StaticBoxSizer

A StaticBoxSizer puts a box sizer into a static box. It provides a border around the box along with a label at the top. Following steps are involved in preparing a staticboxsizer:

- Create a wx.StaticBox object.
- Declare a wx.StaticBoxSizer with the above static box as its argument.
- Create the controls and add in staticbox sizer.
- Set it as the sizer for the frame.

## Example

In the following example, two staticbox sizers are created and added into a top vertical box sizer, which controls the layout of the panel inside a frame.

The first staticbox sizer is created around a static box named 'Name'.

```
nm = wx.StaticBox(panel, -1, 'Name:')
nmSizer = wx.StaticBoxSizer(nm, wx.VERTICAL)
```

A Horizontal box sizer, holding two labels and two text boxes, is added into nmSizer static box sizer.

```
nmbox = wx.BoxSizer(wx.HORIZONTAL)

fn = wx.StaticText(panel, -1, "First Name")
nmbox.Add(fn, 0, wx.ALL|wx.CENTER, 5)
nm1 = wx.TextCtrl(panel, -1, style=wx.ALIGN_LEFT)
nm2 = wx.TextCtrl(panel, -1, style=wx.ALIGN_LEFT)
ln = wx.StaticText(panel, -1, "Last Name")

nmbox.Add(nm1, 0, wx.ALL|wx.CENTER, 5)
nmbox.Add(ln, 0, wx.ALL|wx.CENTER, 5)
nmbox.Add(nm2, 0, wx.ALL|wx.CENTER, 5)

nmSizer.Add(nmbox, 0, wx.ALL|wx.CENTER, 10)
```

Similarly, another staticbox sizer holds a static box named 'Buttons'.

```
sbox = wx.StaticBox(panel, -1, 'buttons:')
sboxSizer = wx.StaticBoxSizer(sbox, wx.VERTICAL)
```



Two button objects, named 'ok' and 'cancel' are put in a horizontal box sizer, which in turn, is placed inside the second staticbox sizer.

```
hbox=wx.BoxSizer(wx.HORIZONTAL)
okButton = wx.Button(panel, -1, 'ok')
hbox.Add(okButton, 0, wx.ALL|wx.LEFT, 10)
cancelButton = wx.Button(panel, -1, 'cancel')
hbox.Add(cancelButton, 0, wx.ALL|wx.LEFT, 10)
sboxSizer.Add(hbox, 0, wx.ALL|wx.LEFT, 10)
```

Two static box sizers, 'name' and 'Buttons' are added into a vertical box sizer acting as the layout manager of the panel in the top level frame.

```
panel=wx.Panel(self)
vbox=wx.BoxSizer(wx.VERTICAL)

vbox.Add(nmSizer,0, wx.ALL|wx.CENTER, 5)
vbox.Add(sboxSizer,0, wx.ALL|wx.CENTER, 5)
panel.SetSizer(vbox)
```

Following is the complete code:

```
import wx

class Mywin(wx.Frame):
    def __init__(self, parent, title):
        super(Mywin, self).__init__(parent, title=title)

        panel=wx.Panel(self)
        vbox=wx.BoxSizer(wx.VERTICAL)
        nm = wx.StaticBox(panel, -1, 'Name:')
        nmSizer = wx.StaticBoxSizer(nm, wx.VERTICAL)

        nmbox = wx.BoxSizer(wx.HORIZONTAL)
        fn = wx.StaticText(panel, -1, "First Name")
        nmbox.Add(fn, 0, wx.ALL|wx.CENTER, 5)
        nm1 = wx.TextCtrl(panel, -1, style=wx.ALIGN_LEFT)
        nm2 = wx.TextCtrl(panel, -1, style=wx.ALIGN_LEFT)
        ln = wx.StaticText(panel, -1, "Last Name")

        nmbox.Add(nm1, 0, wx.ALL|wx.CENTER, 5)
```

```

nmbox.Add(ln, 0, wx.ALL|wx.CENTER, 5)
nmbox.Add(nm2, 0, wx.ALL|wx.CENTER, 5)
nmSizer.Add(nmbox, 0, wx.ALL|wx.CENTER, 10)

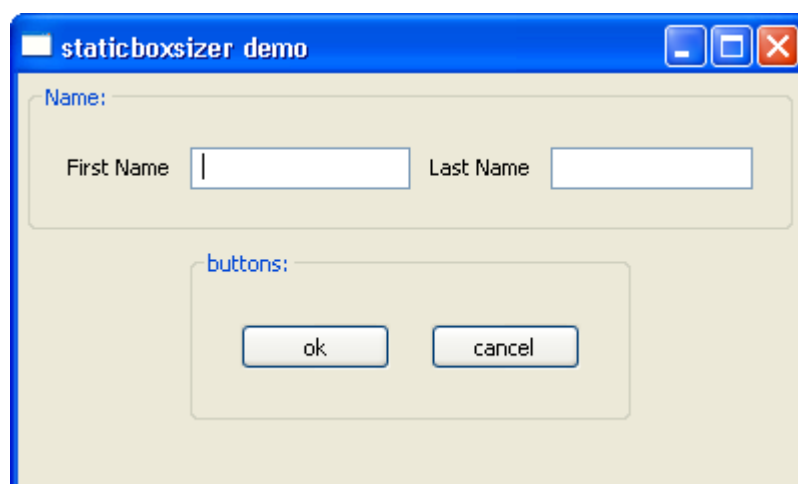
sbox = wx.StaticBox(panel, -1, 'buttons:')
sboxSizer = wx.StaticBoxSizer(sbox, wx.VERTICAL)
hbox=wx.BoxSizer(wx.HORIZONTAL)
okButton = wx.Button(panel, -1, 'ok')
hbox.Add(okButton, 0, wx.ALL|wx.LEFT, 10)
cancelButton = wx.Button(panel, -1, 'cancel')
hbox.Add(cancelButton, 0, wx.ALL|wx.LEFT, 10)
sboxSizer.Add(hbox, 0, wx.ALL|wx.LEFT, 10)
vbox.Add(nmSizer,0, wx.ALL|wx.CENTER, 5)
vbox.Add(sboxSizer,0, wx.ALL|wx.CENTER, 5)
panel.SetSizer(vbox)
self.Centre()

panel.Fit()
self.Show()

app = wx.App()
Mywin(None, 'staticboxsizer demo')
app.MainLoop()

```

The above code produces the following output:



# 15. Buttons

Button widget is most widely used in any GUI interface. It captures the click event generated by the user. Its most obvious use is to trigger a handler function bound to it.

wxPython class library provides different types of buttons. There is a simple, traditional button, **wx.Button** class object, which carries some text as its caption. A two-state button is also available, which is named as **wx.ToggleButton**. Its pressed or depressed state can be identified by eventhandler function.

Another type of button, **wx.BitmapButton** displays a bitmap (image) as icon on its face.

Constructor for wx.Button class and wx.ToggleButton class takes the following arguments:

```
Wx.Button(parent, id, label, pos, size, style)
```

These are some important methods of wx.Button class:

SetLabel()	Sets the button's caption programmatically
GetLabel()	Returns the button's caption
SetDefault()	Button is set to default for the top level window. Emulates the click event on pressing Enter key

Two important methods of wx.ToggleButton class are:

GetValue()	Returns the state of toggle button (on/off)
SetValue()	Sets the state of button programmatically

In order to create a bitmap button, firstly, a bitmap object needs to be constructed out of an image file.

The following variation of wx.Bitmap class constructor is most commonly used:

```
Wx.Bitmap(filename, wx.BITMAP_TYPE)
```

Some of the predefined bitmap type constants are:

- wx.BITMAP\_TYPE\_BMP
- wx.BITMAP\_TYPE\_ICO
- wx.BITMAP\_TYPE\_CUR
- wx.BITMAP\_TYPE\_TIFF
- wx.BITMAP\_TYPE\_TIF
- wx.BITMAP\_TYPE\_GIF

- wx.BITMAP\_TYPE\_PNG
- wx.BITMAP\_TYPE\_JPEG
- wx.BITMAP\_TYPE\_PCX
- wx.BITMAP\_TYPE\_ICON
- wx.BITMAP\_TYPE\_ANY

This bitmap object is used as one of the parameters for wx.BitmapButton class constructor.

```
Wx.BitmapButton(parent, id, bitmap, pos, size, style)
```

On some OS platforms, the bitmap button can display both bitmap and label. SetLabel() methods assign the caption. On other platforms, it serves as an internal label.

The normal button as well as bitmap button emits a wx.CommandEvent. EVT\_BUTTON binder associates a handler function to it.

The toggle button on the other hand uses wx.TOGGLEBUTTON binder for event handling.

In the following example, buttons of all three types are placed in a vertical box sized to a panel.

Simple button object is created using the statement:

```
self.btn=wx.Button(panel,-1,"click Me")
```

Toggle button is constructed by following statement:

```
self.tbtn=wx.ToggleButton(panel , -1, "click to on")
```

These buttons are added into vertical sizer using the following statements:

```
vbox.Add(self.btn,0,wx.ALIGN_CENTER)  
vbox.Add(self.tbtn,0,wx.EXPAND|wx.ALIGN_CENTER)
```

**Note:** Because of wx.EXPAND flag, the toggle button occupies the entire width of the frame.

Using EVT\_BUTTON and EVT\_TOGGLEBUTTON binders they are associated with the respective handlers.

```
self.btn.Bind(wx.EVT_BUTTON,self.OnClicked)  
self.tbtn.Bind(wx.EVT_TOGGLEBUTTON,self.OnToggle)
```

Three bitmap buttons are added into a horizontal box sizer. These buttons display an image as icon as their caption.

```

bmp = wx.Bitmap("NEW.BMP", wx.BITMAP_TYPE_BMP)
self.bmpbtn = wx.BitmapButton(panel, id=wx.ID_ANY, bitmap=bmp,
size=(bmp.GetWidth()+10, bmp.GetHeight()+10))

bmp1 = wx.Bitmap("OPEN.BMP", wx.BITMAP_TYPE_BMP)
self.bmpbtn1 = wx.BitmapButton(panel, id=wx.ID_ANY, bitmap=bmp1,
size=(bmp.GetWidth()+10, bmp.GetHeight()+10))

bmp2 = wx.Bitmap("SAVE.BMP", wx.BITMAP_TYPE_BMP)
self.bmpbtn2 = wx.BitmapButton(panel, id=wx.ID_ANY, bitmap=bmp2,
size=(bmp.GetWidth()+10, bmp.GetHeight()+10))

```

Click event of these three buttons is directed to OnClicked() method.

```

self.bmpbtn.Bind(wx.EVT_BUTTON,self.OnClicked)
self.bmpbtn1.Bind(wx.EVT_BUTTON,self.OnClicked)
self.bmpbtn2.Bind(wx.EVT_BUTTON,self.OnClicked)

```

Internal labels of these buttons are set to NEW, OPEN and SAVE respectively.

OnClicked() event handler function retrieves the label of source button, which caused the click event. That label is printed on the console.

```

def OnClicked(self, event):
    btn=event.GetEventObject().GetLabel()
    print "Label of pressed button=",btn

```

OnToggle() event handler is triggered when the toggle button is clicked. Its state is read by GetValue() method and accordingly, the button's caption is set.

```

def OnToggle(self,event):
    state=event.GetEventObject().GetValue()
    if state==True:
        print "off"
        event.GetEventObject().SetLabel("click to off")
    else:
        print "on"
        event.GetEventObject().SetLabel("click to on")

```

The complete code listing is as follows:

```
import wx

class Mywin(wx.Frame):
    def __init__(self, parent, title):
        super(Mywin, self).__init__(parent, title=title, size=(200,150))

        panel=wx.Panel(self)
        vbox=wx.BoxSizer(wx.VERTICAL)

        self.btn=wx.Button(panel, -1, "click Me")
        vbox.Add(self.btn, 0, wx.ALIGN_CENTER)
        self.btn.Bind(wx.EVT_BUTTON, self.OnClicked)

        self.tbtn=wx.ToggleButton(panel , -1, "click to on")
        vbox.Add(self.tbtn, 0, wx.EXPAND|wx.ALIGN_CENTER)
        self.tbtn.Bind(wx.EVT_TOGGLEBUTTON, self.OnToggle)

        hbox=wx.BoxSizer(wx.HORIZONTAL)

        bmp = wx.Bitmap("NEW.BMP", wx.BITMAP_TYPE_BMP)
        self.bmpbtn = wx.BitmapButton(panel, id=wx.ID_ANY, bitmap=bmp,
size=(bmp.GetWidth()+10, bmp.GetHeight()+10))
        hbox.Add(self.bmpbtn, 0, wx.ALIGN_CENTER)
        self.bmpbtn.Bind(wx.EVT_BUTTON, self.OnClicked)
        self.bmpbtn.SetLabel("NEW")

        bmp1 = wx.Bitmap("OPEN.BMP", wx.BITMAP_TYPE_BMP)
        self.bmpbtn1 = wx.BitmapButton(panel, id=wx.ID_ANY, bitmap=bmp1,
size=(bmp.GetWidth()+10, bmp.GetHeight()+10))
        hbox.Add(self.bmpbtn1, 0, wx.ALIGN_CENTER)
        self.bmpbtn1.Bind(wx.EVT_BUTTON, self.OnClicked)
        self.bmpbtn1.SetLabel("OPEN")

        bmp2 = wx.Bitmap("SAVE.BMP", wx.BITMAP_TYPE_BMP)
        self.bmpbtn2 = wx.BitmapButton(panel, id=wx.ID_ANY, bitmap=bmp2,
size=(bmp.GetWidth()+10, bmp.GetHeight()+10))
        hbox.Add(self.bmpbtn2, 0, wx.ALIGN_CENTER)
        self.bmpbtn2.Bind(wx.EVT_BUTTON, self.OnClicked)
```

```

        self.bmpbtn2.SetLabel("SAVE")

        vbox.Add(hbox,1,wx.ALIGN_CENTER)
        panel.SetSizer(vbox)

        self.Centre()
        self.Show()
        self.Fit()

    def OnClicked(self, event):
        btn=event.GetEventObject().GetLabel()
        print "Label of pressed button=",btn
    def OnToggle(self,event):
        state=event.GetEventObject().GetValue()
        if state==True:
            print "Toggle button state off"
            event.GetEventObject().SetLabel("click to off")
        else:
            print " Toggle button state on"
            event.GetEventObject().SetLabel("click to on")

app = wx.App()
Mywin(None, 'Button demo')
app.MainLoop()

```

The above code produces the following output:



Label of pressed button= click Me

Toggle button state off

Toggle button state on

Label of pressed button = NEW

Label of pressed button = OPEN

Label of pressed button = SAVE

# 16. wx.StaticText Class

Another important element in a GUI interface is a label, a read-only text of one or more lines. It is usually placed on the frame either as an identifier of another widget or as an informative string.

In wxPython, **wx.StaticText class** object presents a control holding such read-only text. It can be termed as a passive control since it doesn't produce any event. Wx.StaticText class constructor requires the following usual parameters:

```
wx.StaticText(parent, id, label, position, size, style)
```

Predefined style enumerators are:

wx.ALIGN_LEFT	Controls the alignment of label within the size
wx.ALIGN_RIGHT	
wx.ALIGN_CENTER	
wx.ST_NO_AUTORESIZE	Prevents auto resizing of the label
wx.ST_ELLIPSIZE_START	Ellipsis (...) appears at the start, in the middle or at the end, if the size of the text is bigger than the label size
wx.ST_ELLIPSIZE_MIDDLE	
wx.ST_ELLIPSIZE_END	

The following methods of wx.StaticText class are also useful:

SetLabel()	Sets label of object programmatically
GetLabel()	Returns object's label
SetForegroundColour()	Sets color of label's text
SetBackgroundColour()	Sets label's background
Wrap()	Wraps label's text if it can not be accommodated within size.

The above features of StaticText class are demonstrated in the following example. Three StaticText objects are placed in a vertical box sizer.

The first object has multi-line text which is center aligned. The second label's text is set to wrap around beyond 200 pixels. The third label shows ellipsis (...) in the middle of the text.



In order to set the label's font, a font object is first created.

```
Wx.Font(pointsize, fontfamily, fontstyle, fontweight)
```

Fontfamily parameter takes the following values:

wx.FONTFAMILY_DEFAULT	Chooses a default font
wx.FONTFAMILY_DECORATIVE	Chooses a decorative font
wx.FONTFAMILY_ROMAN	Chooses a formal, serif font
wx.FONTFAMILY_SCRIPT	Chooses a handwriting font
wx.FONTFAMILY_SWISS	Chooses a sans-serif font
wx.FONTFAMILY_MODERN	Chooses a fixed pitch font
wx.FONTFAMILY_TELETYPE	Chooses a teletype (monospaced) font

FontStyle parameter enumerations are:

Wx.FONTSTYLE_NORMAL	The font is drawn without slant
wx.FONTSTYLE_ITALIC	The font is slanted in italic style
wx.FONTSTYLE_SLANT	The font is slanted, but in roman style

FontWeight parameters are:

Wx.FONTWEIGHT_NORMAL	Normal font
wx.FONTWEIGHT_LIGHT	Light font
wx.FONTWEIGHT_BOLD	Bold font

The complete code listing is:

```
import wx

class Mywin(wx.Frame):
    def __init__(self, parent, title):
        super(Mywin, self).__init__(parent, title=title, size=(600,200))
```

```

        panel=wx.Panel(self)
        box=wx.BoxSizer(wx.VERTICAL)
        lbl=wx.StaticText(panel, -1, style=wx.ALIGN_CENTER)
        txt1 = "Python GUI development"
        txt2="using wxPython"
        txt3=" Python port of wxWidget "
        txt=txt1+"\n"+txt2+"\n"+txt3
        font = wx.Font(18, wx.ROMAN, wx.ITALIC, wx.NORMAL)
        lbl.SetFont(font)
        lbl.SetLabel(txt)
        box.Add(lbl,0,wx.ALIGN_CENTER)
        lblwrap=wx.StaticText(panel, -1, style=wx.ALIGN_RIGHT)
        txt=txt1+txt2+txt3
        lblwrap.SetLabel(txt)
        lblwrap.Wrap(200)
        box.Add(lblwrap,0,wx.ALIGN_LEFT)
        lbl1=wx.StaticText(panel, -1, style=wx.ALIGN_LEFT | wx.ST_ELLIPSIZE_MIDDLE)
        lbl1.SetLabel(txt)
        lbl1.SetForegroundColour((255,0,0))
        lbl1.SetBackgroundColour((0,0,0))
        font=self.GetFont()
        font.SetPointSize(20)
        lbl1.SetFont(font)
        box.Add(lbl1,0,wx.ALIGN_LEFT)
        panel.SetSizer(box)
        self.Centre()
        self.Show()

app = wx.App()
Mywin(None, 'StaticText demo')
app.MainLoop()

```

The above code produces the following output:



# 17. wx.TextCtrl Class

In a GUI interface, the input is most commonly collected in a text box where the user can type using the keyboard. In wxPython, an object of wx.TextCtrl class serves this purpose. It is a control in which the text can be displayed and edited. The **TextCtrl widget** can be a single line, multi-line or a password field. TextCtrl class constructor takes the following form:

```
wx.TextCtrl(parent, id, value, pos, size, style)
```

The style parameter takes one or more constants from the following list:

```
wx.TE_MULTILINE    The text control allows multiple lines. If this style is not
                    specified, the line break characters should not be used in the controls value.
wx.TE_PASSWORD      The text will be echoed as asterisks
wx.TE_READONLY      The text will not be user-editable
wx.TE_LEFT          The text in the control will be left-justified (default)
wx.TE_CENTRE        The text in the control will be centered
wx.TE_RIGHT         The text in the control will be right-justified
```

The important methods of wx.TextCtrl class are:

```
AppendText()        Adds text to the end of text control
Clear()             Clears the contents
GetValue()           Returns the contents of the text box
Replace()            Replaces the entire or portion of text in the box
SetEditable()        Makes the text box editable or read-only
SetMaxLength()       Sets maximum number of characters the control can hold
SetValue()           Sets the contents in the text box programmatically
IsMultiLine()        Returns true if set to TE_MULTILINE
```

The following event binders are responsible for event handling related to entering text in TextCtrl box:

EVT_TEXT	Responds to changes in the contents of text box, either by manually keying in, or programmatically
EVT_TEXT_ENTER	Invokes associated handler when Enter key is pressed in the text box

EVT_TEXT_MAXLEN	Triggers associated handler as soon as the length of text entered reaches the value of SetMaxLength() function
-----------------	--

## Example

In the following example, four objects of wx.TextCtrl class with different attributes are placed on the panel.

```
self.t1=wx.TextCtrl(panel)
self.t2=wx.TextCtrl(panel,style=wx.TE_PASSWORD)
self.t3=wx.TextCtrl(panel,size=(200,100),style=wx.TE_MULTILINE)
self.t4 = wx.TextCtrl ( panel, value="ReadOnly Text", style = wx.TE_READONLY |
wx.TE_CENTER )
```

While the first is a normal text box, the second is a password field. The third one is a multiline text box and the last text box is non-editable.

EVT\_TEXT binder on first box triggers OnKeyTyped() method for each key stroke in it. The second box is having its MaxLength set to 5. EVT\_TEXT\_MAXLEN binder sends OnMaxLen() function running as soon as the user tries to type more than 5 characters. The multiline text box responds to Enter key pressed because of EVT\_TEXT\_ENTER binder.

The complete code is as follows:

```
import wx

class Mywin(wx.Frame):
    def __init__(self, parent, title):
        super(Mywin, self).__init__(parent, title=title,size=(350,250))

        panel=wx.Panel(self)
        vbox=wx.BoxSizer(wx.VERTICAL)

        hbox1=wx.BoxSizer(wx.HORIZONTAL)
        l1=wx.StaticText(panel, -1, "Text Field")
        hbox1.Add(l1, 1, wx.EXPAND|wx.ALIGN_LEFT|wx.ALL,5)
        self.t1=wx.TextCtrl(panel)
        hbox1.Add(self.t1,1,wx.EXPAND|wx.ALIGN_LEFT|wx.ALL,5)
        self.t1.Bind(wx.EVT_TEXT,self.OnKeyTyped)
        vbox.Add(hbox1)

        hbox2=wx.BoxSizer(wx.HORIZONTAL)
```

```

l2=wx.StaticText(panel, -1, "password field")
hbox2.Add(l2, 1, wx.ALIGN_LEFT|wx.ALL,5)
self.t2=wx.TextCtrl(panel,style=wx.TE_PASSWORD)
self.t2.SetMaxLength(5)
hbox2.Add(self.t2,1,wx.EXPAND|wx.ALIGN_LEFT|wx.ALL,5)
vbox.Add(hbox2)
self.t2.Bind(wx.EVT_TEXT_MAXLEN,self.OnMaxLen)

hbox3=wx.BoxSizer(wx.HORIZONTAL)
l3=wx.StaticText(panel, -1, "Multiline Text")
hbox3.Add(l3,1, wx.EXPAND|wx.ALIGN_LEFT|wx.ALL,5)
self.t3=wx.TextCtrl(panel,size=(200,100),style=wx.TE_MULTILINE)
hbox3.Add(self.t3,1,wx.EXPAND|wx.ALIGN_LEFT|wx.ALL,5)
vbox.Add(hbox3)
self.t3.Bind(wx.EVT_TEXT_ENTER,self.OnEnterPressed)

hbox4=wx.BoxSizer(wx.HORIZONTAL)
l4=wx.StaticText(panel, -1, "Read only text")
hbox4.Add(l4, 1, wx.EXPAND|wx.ALIGN_LEFT|wx.ALL,5)
self.t4=wx.TextCtrl(panel, value="ReadOnly
Text",style=wx.TE_READONLY|wx.TE_CENTER)
hbox4.Add(self.t4,1,wx.EXPAND|wx.ALIGN_LEFT|wx.ALL,5)
vbox.Add(hbox4)
panel.SetSizer(vbox)

self.Centre()
self.Show()
self.Fit()

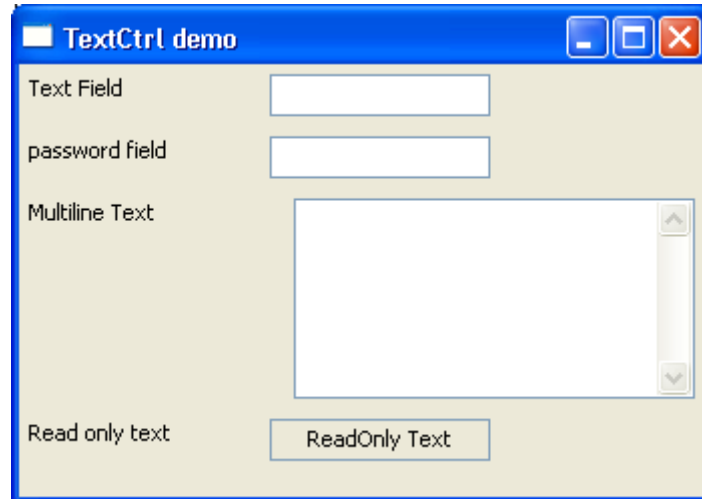
def OnKeyTyped(self, event):
    print event.GetString()
def OnEnterPressed(self,event):
    print "Enter pressed"
def OnMaxLen(self,event):
    print "Maximum length reached"

app = wx.App()
Mywin(None, 'TextCtrl demo')

```

```
app.MainLoop()
```

The above code produces the following output:



# 18. wx.RadioButton & wx.RadioButton

A Radio button usually represents one of many selectable buttons available for the user in a group. Each button, an object of wx.RadioButton class carries a text label next to a round button.

In order to create a group of mutually selectable buttons, the style parameter of first wx.RadioButton object is set to wx.RB\_GROUP. Subsequent button objects are added to a group.

wxPython API also consists of wx.RadioButton class. Its object offers a border and label to the group. Buttons in the group can be arranged horizontally or vertically.

wx.RadioButton constructor looks like:

```
wx.RadioButton(parent, id, label, pos, size, style)
```

The style parameter is present only for the first button in the group. Its value is wx.RB\_GROUP. For subsequent buttons in the group, wx.RB\_SINGLE style parameter may be optionally used.

wx.RadioButton event binder wx.EVT\_RADIOBUTTON triggers the associated handler every time any of the buttons in the group is clicked.

Two important methods of wx.RadioButton class are SetValue() – to select or deselect a button programmatically – and GetValue() – which returns true if a button is selected and is false otherwise.

A **wx.RadioButton** places a collection of mutually exclusive buttons in a static box. Each button in the group takes its label from a List object which acts as 'choices' parameter for wx.RadioButton constructor.

Buttons in RadioButton are laid out in row-wise or column-wise manner. For that 'style' parameter of constructor should be either wx.RA\_SPECIFY\_ROWS or wx.RA\_SPECIFY\_COLS. Number of rows/columns is decided by the value of 'majordimensions' parameter.

Prototype of wx.RadioButton constructor is:

```
wx.RadioButton(parent, id, label, pos, size, choices[], initialdimensions, style)
```

Following are the important methods in wx.RadioButton class:

GetSelection()	Returns the index of the selected item
SetSelection()	Selects an item programmatically
GetString()	Returns the label of the selected item
SetString()	Assigns the label to the selected item
Show()	Shows or hides the item of the given index



Event binder associated with wx.RadioButton object is wx.EVT\_RADIOBOX. Associated event handler identifies the button selection and processes it.

## Example

The following example demonstrates the use of RadioBox as well as RadioButton. Firstly, three RadioButtons, grouped by specifying wx.RB\_GROUP style are placed on the panel.

```
self.rb1 = wx.RadioButton(pnl,11, label='Value A', pos=(10,10), style =
wx.RB_GROUP)
self.rb2 = wx.RadioButton(pnl,22, label='Value B',pos=(10,40))
self.rb3 = wx.RadioButton(pnl,33, label='Value C',pos=(10,70))
```

The RadioBox, on the other hand, reads labels for its buttons from a lblList[] object.

```
lblList = ['Value X', 'Value Y', 'Value Z']
self.rbox=wx.RadioButton(pnl,label='RadioBox', pos=(80,10), choices=lblList ,
majorDimension=1, style=wx.RA_SPECIFY_ROWS)
```

Two event binders, one for radio group and other for RadioBox, are declared.

```
self.Bind(wx.EVT_RADIOBUTTON, self.OnRadiogroup)
self.rbox.Bind(wx.EVT_RADIOBOX,self.onRadioBox)
```

The corresponding event handlers identify the button selected and display the message on the console window.

```
def OnRadiogroup(self, e):
    rb = e.GetEventObject()
    print rb.GetLabel(), ' is clicked from Radio Group'
def onRadioBox(self,e):
    print self.rbox.GetStringSelection(), ' is clicked from Radio Box'
```

The complete code is as follows:

```
import wx

class Example(wx.Frame):

    def __init__(self, parent, title):
        super(Example, self).__init__(parent, title=title,size=(300,200))

        self.InitUI()
```

```

def InitUI(self):
    pnl = wx.Panel(self)

    self.rb1 = wx.RadioButton(pnl,11, label='Value A', pos=(10,10),
style=wx.RB_GROUP)
    self.rb2 = wx.RadioButton(pnl,22, label='Value B',pos=(10,40))
    self.rb3 = wx.RadioButton(pnl,33, label='Value C',pos=(10,70))
    self.Bind(wx.EVT_RADIOBUTTON, self.OnRadiogroup)

    lblList = ['Value X', 'Value Y', 'Value Z']

    self.rbox=wx.RadioButton(pnl, label='RadioBox', pos=(80,10), choices =
lblList, majorDimension=1, style=wx.RA_SPECIFY_ROWS)
    self.rbox.Bind(wx.EVT_RADIOBOX,self.onRadioBox)

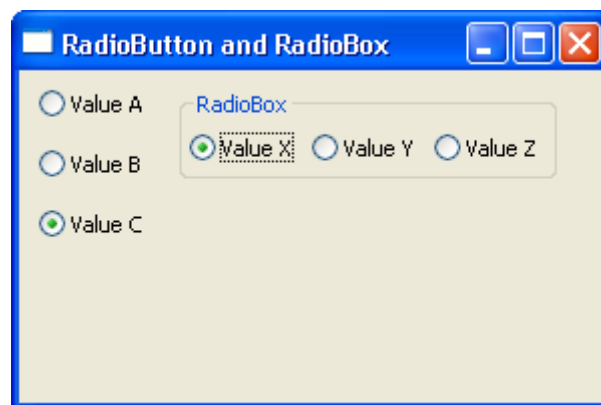
    self.Centre()
    self.Show(True)

def OnRadiogroup(self, e):
    rb = e.GetEventObject()
    print rb.GetLabel(),' is clicked from Radio Group'
def onRadioBox(self,e):
    print self.rbox.GetStringSelection(),' is clicked from Radio Box'

ex = wx.App()
Example(None,'RadioButton and RadioBox')
ex.MainLoop()

```

The above code produces the following output:



Value B is clicked from Radio Group

Value C is clicked from Radio Group

Value Y is clicked from Radio Box

Value X is clicked from Radio Box

# 19. wx.CheckBox Class

A **checkbox** displays a small labeled rectangular box. When clicked, a checkmark appears inside the rectangle to indicate that a choice is made. Checkboxes are preferred over radio buttons when the user is to be allowed to make more than one choice. In this case, the third state is called mixed or undetermined state, generally used in 'doesn't apply' scenario.

Normally, a checkbox object has two states (checked or unchecked). **Tristate checkbox** can also be constructed if the appropriate style parameter is given.

wx.CheckBox class constructor takes the following parameters:

```
wx.CheckBox(parent, id, label, pos, size, style)
```

The following style parameter values can be used:

wx.CHK_2STATE	Creates two state checkbox. Default
wx.CHK_3STATE	Creates three state checkbox
wx.ALIGN_RIGHT	Puts a box label to the left of the checkbox

This class has two important methods: GetState() returns true or false depending on if the checkbox is checked or not. SetValue() is used to select a checkbox programmatically.

wx.EVT\_CHECKBOX is the only event binder available. Associated event handler will be invoked every time any checkbox on the frame is checked or unchecked.

## Example

Following is a simple example demonstrating the use of three checkboxes. Handler function OnChecked() identifies the checkbox, which is responsible for the event and displays its state.

The complete code is:

```
import wx

class Example(wx.Frame):

    def __init__(self, parent, title):
        super(Example, self).__init__(parent, title=title, size=(200,200))

        self.InitUI()
```

```
def InitUI(self):

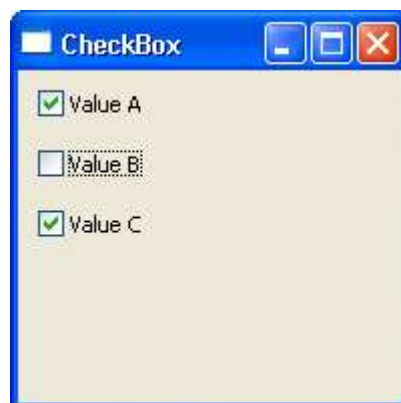
    pnl = wx.Panel(self)

    self.cb1 = wx.CheckBox(pnl, label='Value A',pos=(10,10))
    self.cb2 = wx.CheckBox(pnl, label='Value B',pos=(10,40))
    self.cb3 = wx.CheckBox(pnl, label='Value C',pos=(10,70))
    self.Bind(wx.EVT_CHECKBOX,self.onChecked)
    self.Centre()
    self.Show(True)

def onChecked(self, e):
    cb = e.GetEventObject()
    print cb.GetLabel(), ' is clicked',cb.GetValue()

ex = wx.App()
Example(None, 'CheckBox')
ex.MainLoop()
```

The above code produces the following output:



Value A is clicked True

Value B is clicked True

Value C is clicked True

Value B is clicked False

## 20. wx.ComboBox & wx.Choice Class

A **wx.ComboBox** object presents a list of items to select from. It can be configured to be a dropdown list or with permanent display.

The selected item from the list is displayed in a text field, which by default is editable, but can be set to be read-only in the presence of wx.CB\_READONLY style parameter.

wxPython API contains a **wx.Choice class**, whose object is also a dropdown list, which is permanently read-only.

The parameters used by wx.ComboBox class constructor are:

```
wx.ComboBox(parent, id, value, pos, size, choices[], style)
```

The value parameter is the text to be displayed in the text box of combobox. It is populated from the items in choices[] collection.

The following style parameters are defined for wx.ComboBox:

wx.CB_SIMPLE	Combobox with permanently displayed list
wx.CB_DROPDOWN	Combobox with dropdown list
wx.CB_READONLY	Chosen item is not editable
wx.CB_SORT	List is displayed in alphabetical order

The following table shows commonly used methods of wx.ComboBox class:

GetCurrentSelection ()	Returns the item being selected
SetSelection()	Sets the item at the given index as selected
GetString()	Returns the string associated with the item at the given index
SetString()	Changes the text of item at the given index
SetValue()	Sets a string as the text displayed in the edit field of combobox
GetValue()	Returns the contents of the text field of combobox
FindString()	Searches for the given string in the list
GetStringSelection()	Gets the text of the currently selected item

Event binders for events generated by this class are as follows:

<code>wx.COMBOBOX</code>	When item from the list is selected
<code>wx.EVT_TEXT</code>	When combobox text changes
<code>wx.EVT_COMBOBOX_DROPDOWN</code>	When list drops down
<code>wx.EVT_COMBOBOX_CLOSEUP</code>	When list folds up

`wx.Choice` class constructor prototype is as follows:

```
wx.Choice(parent, id, pos, size, n, choices[], style)
```

Parameter 'n' stands for number of strings with which the choice list is to be initialized. Like `comboBox`, the list is populated with items in `choices[]` collection.

For `Choice` class, only one style parameter is defined. It is `wx.CB_SORT`. Only one event binder processes the event emitted by this class. It is `wx.EVT_CHOICE`.

## Example

This example displays the features of `wx.ComboBox` and `wx.Choice`. Both objects are put in a vertical box sizer. The lists are populated with items in `languages[]` List object.

```
languages = ['C', 'C++', 'Python', 'Java', 'Perl']
self.combo=wx.ComboBox(panel,choices=languages)
self.choice=wx.Choice(panel,choices=languages)
```

Event binders `EVT_COMBOBOX` and `EVT_CHOICE` process corresponding events on them.

```
self.combo.Bind(wx.EVT_COMBOBOX, self.OnCombo)
self.choice.Bind(wx.EVT_CHOICE, self.OnChoice)
```

The following handler functions display the selected item from the list on the label.

```
def OnCombo(self, event):
    self.label.SetLabel("selected "+ self.combo.GetValue() +" from Combobox")
    def OnChoice(self,event):
        self.label.SetLabel("selected "+ self.choice.GetString(
self.choice.GetSelection() ) +" from Choice")
```

The complete code listing is as follows:

```
import wx

class Mywin(wx.Frame):
    def __init__(self, parent, title):
        super(Mywin, self).__init__(parent, title=title, size=(300,200))

        panel=wx.Panel(self)
        box=wx.BoxSizer(wx.VERTICAL)
        self.label=wx.StaticText(panel, label="Your choice:" , style=wx.ALIGN_CENTRE)
        box.Add(self.label, 0 , wx.EXPAND | wx.ALIGN_CENTER_HORIZONTAL | wx.ALL, 20)
        cblbl=wx.StaticText(panel, label="Combo box", style=wx.ALIGN_CENTRE)
        box.Add(cblbl, 0, wx.EXPAND | wx.ALIGN_CENTER_HORIZONTAL | wx.ALL, 5)
        languages = ['C', 'C++', 'Python', 'Java', 'Perl']
        self.combo=wx.ComboBox(panel, choices=languages)
        box.Add(self.combo, 1, wx.EXPAND | wx.ALIGN_CENTER_HORIZONTAL | wx.ALL, 5)
        chlbl=wx.StaticText(panel, label="Choice control", style=wx.ALIGN_CENTRE)
        box.Add(chlbl, 0, wx.EXPAND | wx.ALIGN_CENTER_HORIZONTAL | wx.ALL, 5)
        self.choice=wx.Choice(panel, choices=languages)
        box.Add(self.choice, 1, wx.EXPAND | wx.ALIGN_CENTER_HORIZONTAL | wx.ALL, 5)

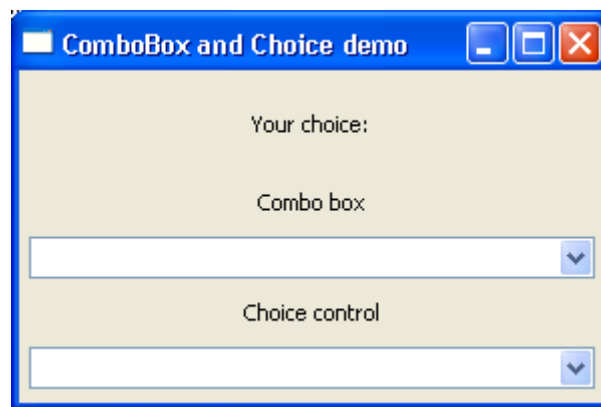
        box.AddStretchSpacer()
        self.combo.Bind(wx.EVT_COMBOBOX, self.OnCombo)
        self.choice.Bind(wx.EVT_CHOICE, self.OnChoice)
        panel.SetSizer(box)
        self.Centre()
        self.Show()

    def OnCombo(self, event):
        self.label.SetLabel("You selected "+self.combo.GetValue()+" from Combobox")
    def OnChoice(self, event):
        self.label.SetLabel("You selected "+ self.choice.GetString
(self.choice.GetSelection())+" from Choice")

app = wx.App()
Mywin(None, 'ComboBox and Choice demo')
app.MainLoop()
```



The above code produces the following output:



# 21. wx.Gauge Class

Progressbar control in wxPython is called **Gauge**. Wx.Gauge class object shows a vertical or horizontal bar, which graphically shows incrementing quantity. It is typically used to demonstrate progression of a process like copying files or installing a software.

**Wx.Gauge control** can be used in determinate as well as indeterminate mode. When the time required to complete any operation can be fairly accurately determined, the gauge progress bar shows the percentage of completed task. However, in indeterminate mode, it only indicates that the process is ongoing.

In determinate mode, the progress position is updated periodically. In indeterminate mode, calling Pulse() function will update the progress bar.

Parameters required by Wx.Gauge class constructor is:

```
wx.Gauge(parent, id, range, pos, size, style)
```

The range parameter sets the maximum value for the gauge. In indeterminate mode, this parameter is ignored.

The possible style parameters for Gauge class are:

wx.GA_HORIZONTAL	The horizontal layout of the progress bar
wx.GA_VERTICAL	The vertical layout of the progress bar
wx.GA_SMOOTH	Smooths progress bar with one pixel wide update step
wx.GA_TEXT	Displays the current value in percent

Some of the important methods of this class are listed in the following table:

GetRange()	Returns the maximum value of the gauge
SetRange()	Sets the maximum value for the gauge
GetValue()	Returns the current value of the gauge
SetValue()	Sets the current value programmatically
Pulse()	Switches the gauge to indeterminate mode

## Example

In the following example, a horizontal Gauge object is added in the vertical box sizer of panel.

```
self.gauge = wx.Gauge(pnl, range=20, size=(250, 25), style = wx.GA_HORIZONTAL)
```

There is also a button whose click event is associated with a handler function.

```
self.btn1 = wx.Button(pnl, label="Start")
self.Bind(wx.EVT_BUTTON, self.OnStart, self.btn1)
```

The handler function OnStart() updates the progress of gauge after every second.

```
def OnStart(self, e):
    while True:
        time.sleep(1);
        self.count = self.count + 1
        self.gauge.SetValue(self.count)
        if self.count>=20:
            print "end"
            return
```

The complete code for the example is as follows:

```
import wx
import time
class Mywin(wx.Frame):

    def __init__(self, parent, title):
        super(Mywin, self).__init__(parent, title=title,size=(300,200))
        self.InitUI()

    def InitUI(self):
        self.count = 0
        pnl = wx.Panel(self)
        vbox = wx.BoxSizer(wx.VERTICAL)
        hbox1 = wx.BoxSizer(wx.HORIZONTAL)
        hbox2 = wx.BoxSizer(wx.HORIZONTAL)
```

```

self.gauge = wx.Gauge(pnl, range=20, size=(250, 25), style= wx.GA_HORIZONTAL)
self.btn1 = wx.Button(pnl, label="Start")
self.Bind(wx.EVT_BUTTON, self.OnStart, self.btn1)
hbox1.Add(self.gauge, proportion=1, flag=wx.ALIGN_CENTRE)
hbox2.Add(self.btn1, proportion=1, flag=wx.RIGHT, border=10)

vbox.Add((0, 30))
vbox.Add(hbox1, flag=wx.ALIGN_CENTRE)
vbox.Add((0, 20))
vbox.Add(hbox2, proportion=1, flag=wx.ALIGN_CENTRE)
pnl.SetSizer(vbox)

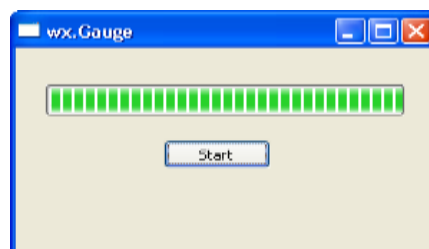
self.SetSize((300, 200))
self.Centre()
self.Show(True)

def OnStart(self, e):
    while True:
        time.sleep(1);
        self.count = self.count + 1
        self.gauge.SetValue(self.count)
        if self.count>=20:
            print "end"
            return

ex = wx.App()
Mywin(None, 'wx.Gauge')
ex.MainLoop()

```

The above code produces the following output:



## 22. wx.Slider Class

A **slider** presents the user with a groove over which a handle can be moved. It is a classic widget to control a bounded value. Position of the handle on the groove is equivalent to an integer between the lower and upper bounds of the control.

wxPython API contains wx.Slider class. It offers same functionality as that of Scrollbar. Slider offers a convenient way to handle dragging the handle by slider specific wx.EVT\_SLIDER event binder.

The definition of wx.Slider constructor takes the following eight parameters:

```
wx.Slider(parent, id, value, minValue, maxValue, pos, size, style)
```

Slider's lower and upper values are set by minValue and maxValue parameters. The starting value is defined by the value parameter.

Many style parameter values are defined. Following are some of them:

wxSL_HORIZONTAL	Horizontal slider
wxSL_VERTICAL	Vertical slider
wxSL_AUTOTICKS	Displays tickmarks on the slider
wxSL_LABELS	Displays the min, max, and current value
wxSL_MIN_MAX_LABELS	Displays the min and max value
wxSL_VALUE_LABEL	Displays the current value only

The useful methods of wx.Slider class are:

GetMin()	Returns the minimum value of the slider
GetMax()	Returns the maximum value of the slider
GetValue()	Returns the current value of the slider
SetMin()	Sets the minimum value of the slider
SetMax()	Sets the maximum value of the slider
SetRange()	Sets the minimum and maximum slider values
SetValue()	Sets the current value programmatically
SetTick()	Displays the tick mark at the given position
SetTickFreq()	Sets the tick interval between the min and max values

As the slider behaves similar to a scroll bar, the scroll bar event binders can also be used along with it.

wx.EVT_SCROLL	Processes the scroll event
wx.EVT_SLIDER	When the slider position changes, either by moving the handle or programmatically

## Example

In the example that follows, the slider is used to control the size of a label. First of all, a slider object is placed in a vertical box size below which is a StaticText.

```
self.sld = wx.Slider(pnl, value=10, minValue=1, maxValue=100, style=
wx.SL_HORIZONTAL|wx.SL_LABELS)

self.txt = wx.StaticText(pnl, label='Hello', style=wx.ALIGN_CENTER)
```

Wx.EVT\_SLIDER binder is associated with OnSliderScroll() handler.

```
self.sld.Bind(wx.EVT_SLIDER, self.OnSliderScroll)
```

The handler itself is fetching slider's current value and using it as font size for the label's text.

```
def OnSliderScroll(self, e):
    obj = e.GetEventObject()
    val = obj.GetValue()
    font=self.GetFont()
    font.SetPointSize(self.sld.GetValue())
    self.txt.SetFont(font)
```

The complete code is as follows:

```
import wx

class Mywin(wx.Frame):

    def __init__(self, parent, title):
        super(Mywin, self).__init__(parent, title=title, size=(250,150))
        self.InitUI()

    def InitUI(self):
        pnl = wx.Panel(self)
        vbox=wx.BoxSizer(wx.VERTICAL)
```

```

        self.sld = wx.Slider(pnl, value=10, minValue=1, maxValue=100,
style=wx.SL_HORIZONTAL|wx.SL_LABELS)
        vbox.Add(self.sld,1,flag = wx.EXPAND | wx.ALIGN_CENTER_HORIZONTAL
|wx.TOP, border=20)
        self.sld.Bind(wx.EVT_SLIDER, self.OnSliderScroll)
        self.txt = wx.StaticText(pnl, label='Hello',style=wx.ALIGN_CENTER)
        vbox.Add(self.txt,1,wx.ALIGN_CENTRE_HORIZONTAL)
        pnl.SetSizer(vbox)
        self.Centre()
        self.Show(True)

    def OnSliderScroll(self, e):
        obj = e.GetEventObject()
        val = obj.GetValue()
        font=self.GetFont()
        font.SetPointSize(self.sld.GetValue())
        self.txt.SetFont(font)
ex = wx.App()
Mywin(None,'Slider demo')
ex.MainLoop()

```

Run the code and try dragging the slider handle to see the label's font size changing. The above code produces the following output:



## 23. Menu Item, Menu & MenuBar

A horizontal bar just below the title bar of a top level window is reserved to display a series of menus. It is an object of **wx.MenuBar class** in wxPython API.

An object of wx.Menu class is added to the menu bar. It is also used to create context menu and popup menu. Each menu may contain one or more wx.MenuItem objects or cascaded Menu objects.

wx.MenuBar class has a parameterized constructor in addition to a default one.

```
wx.MenuBar()
```

```
wx.MenuBar(n, menus, titles, style)
```

Parameter 'n' represents the number of menus. **Menu** is an array of menus and titles, and an array of strings. If the style parameter is set to wx.MB\_DOCKABLE, the menubar can be docked.

Following is a list of methods of wx.MenuBar class:

Append()	Adds menu object to bar
Check()	Checks or unchecks a menu
Enable()	Enables or disables menu
Remove()	Remove a menu from bar

A wx.Menu class object is a pull-down list of one or more menu items, one of which may be selected by the user.

The following table shows frequently required methods of wx.Menu class:

Append()	Adds a menu item in the menu
AppendMenu()	Appends a sub menu
AppendRadioItem()	Appends a selectable radio item
AppendCheckItem()	Appends a checkable menu item
AppendSeparator()	Adds a separator line
Insert()	Inserts a new menu at the given position
InsertRadioItem()	Inserts a radio item at the given position
InsertCheckItem()	Inserts a new check item at the given position



InsertSeparator()	Inserts a separator line
Remove()	Removes an item from the menu
GetMenuItems()	Returns a list of menu items

A **Menu Item** can be directly added using Append() function, or an object of wx.MenuItem class is used to append.

```
wx.Menu.Append(id, text, kind)

Item=wx.MenuItem(parentmenu, id, text, kind)
wx.Menu.Append(Item)
```

In order to define a menu item, the menu in which it is to be added must be mentioned.

wxPython has a large number of standard IDs to be assigned to standard menu items. On some OS platforms, they are associated with standard icons too.

wx.ID_SEPARATOR
wx.ID_ANY
wx.ID_OPEN
wx.ID_CLOSE
wx.ID_NEW
wx.ID_SAVE
wx.ID_SAVEAS
wx.ID_EDIT
wx.ID_CUT
wx.ID_COPY
wx.ID_PASTE

However, any unique integer number can be assigned as ID. The text parameter is its caption. Kind parameter takes one of the following enumerators:

wx.ITEM_NORMAL	Normal menu item
wx.ITEM_CHECK	Check (or toggle) menu item
wx.ITEM_RADIO	Radio menu item

wx.Menu class also has AppendRadioItem() and AppendCheckItem() that don't require kind parameter.

A menu Item can be set to display an icon or shortcut. SetBitmap() function of wx.MenuItem class requires a bitmap object to be displayed.

```
wx.MenuItem.SetBitmap(wx.Bitmap(image file))
```

EVT\_MENU event binder helps in processing the menu selection.

```
self.Bind(wx.EVT_MENU, self.menuhandler)
```

## Example

The following example demonstrates most of the above mentioned features of menu system in wxPython. It shows a File menu displayed in the Menu bar. Normal menu item, a sub menu, radio items and check items are added into it. Menu items having an icon are also present.

Event handler, when invoked retrieves ID associated with the event and can be further processed. For instance, if 'New' menu item is selected, it is echoed in the text box on the frame.

The complete code is as follows:

```
import wx

class Mywin(wx.Frame):

    def __init__(self, parent, title):
        super(Mywin, self).__init__(parent, title=title, size=(250,150))
        self.InitUI()

    def InitUI(self):
        menubar = wx.MenuBar()

        fileMenu = wx.Menu()
        newitem=wx.MenuItem(fileMenu,wx.ID_NEW, text="New",kind=wx.ITEM_NORMAL)
        newitem.SetBitmap(wx.Bitmap("new.bmp"))
        fileMenu.AppendItem(newitem)

        fileMenu.AppendSeparator()

        editMenu=wx.Menu()
        copyItem=wx.MenuItem(editMenu, 100,text="copy",kind = wx.ITEM_NORMAL)
```

```

copyItem.SetBitmap(wx.Bitmap("copy.bmp"))
editMenu.AppendItem(copyItem)
cutItem=wx.MenuItem(editMenu, 101,text="cut",kind = wx.ITEM_NORMAL)
cutItem.SetBitmap(wx.Bitmap("cut.bmp"))
editMenu.AppendItem(cutItem)
pasteItem=wx.MenuItem(editMenu, 102,text="paste",kind = wx.ITEM_NORMAL)
pasteItem.SetBitmap(wx.Bitmap("paste.bmp"))
editMenu.AppendItem(pasteItem)
fileMenu.AppendMenu(wx.ID_ANY, "Edit",editMenu)
fileMenu.AppendSeparator()

radio1=wx.MenuItem(fileMenu, 200,text="Radio1",kind = wx.ITEM_RADIO)
radio2=wx.MenuItem(fileMenu, 300,text="radio2",kind = wx.ITEM_RADIO)
fileMenu.AppendItem(radio1)
fileMenu.AppendItem(radio2)
fileMenu.AppendSeparator()

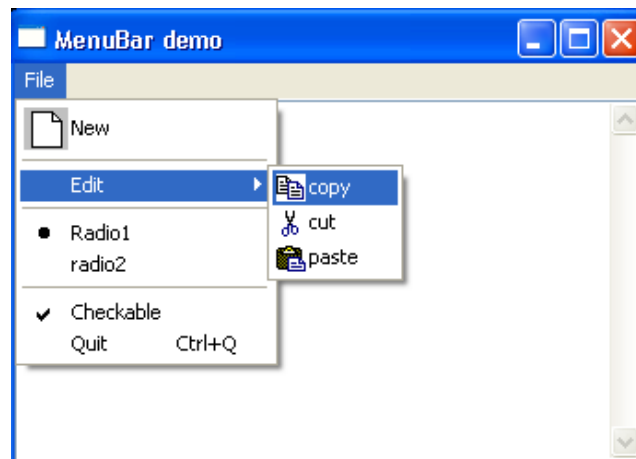
fileMenu.AppendCheckItem(103,"Checkable")
quit = wx.MenuItem(fileMenu, wx.ID_EXIT, '&Quit\tCtrl+Q')
fileMenu.AppendItem(quit)
menubar.Append(fileMenu, '&File')
self.SetMenuBar(menubar)
self.text=wx.TextCtrl(self,-1, style=wx.EXPAND|wx.TE_MULTILINE)
self.Bind(wx.EVT_MENU, self.menuhandler)
self.SetSize((350, 250))
self.Centre()
self.Show(True)

def menuhandler(self, event):
    id=event.GetId()
    if id==wx.ID_NEW:
        self.text.AppendText("new"+"\\n")

ex = wx.App()
Mywin(None,'MenuBar demo')
ex.MainLoop()

```

The above code produces the following output:



## 24. wx.ToolBar Class

One or more horizontal strips of toolbars comprising of buttons with text caption or icons are normally placed just below the MenuBar in a top level frame.

If the style parameter of **wx.ToolBar** object is set to `wx.TB_DOCKABLE`, it becomes dockable. A floating toolbar can also be constructed using wxPython's `AUIToolBar` class.

Constructor without any arguments places a toolbar with default parameters. Additional parameters can be passed to `wx.ToolBar` class constructor as follows:

```
wx.ToolBar(parent, id, pos, size, style)
```

Style parameters defined for `wx.ToolBar` include the following constants:

<code>wx.TB_FLAT</code>	Gives the toolbar a flat look
<code>wx.TB_HORIZONTAL</code>	Specifies the horizontal layout (default)
<code>wx.TB_VERTICAL</code>	Specifies the vertical layout
<code>wx.TB_DEFAULT_STYLE</code>	Combines <code>wx.TB_FLAT</code> and <code>wx.TB_HORIZONTAL</code>
<code>wx.TB_DOCKABLE:</code>	Makes the toolbar floatable and dockable
<code>wx.TB_NO_TOOLTIPS:</code>	Doesn't show the short help tooltips for the tools, when the mouse hovers over them
<code>wx.TB_NOICONS:</code>	Specifies no icons in the toolbar buttons; by default they are shown
<code>wx.TB_TEXT</code>	Shows the text in the toolbar buttons; by default only icons are shown

Tool buttons of different features can be added into the toolbar. `Wx.ToolBar` class has the following useful methods:

<code>AddTool()</code>	Adds a tool button to the toolbar. The type of tool is specified by <code>kind</code> parameter
<code>AddRadioTool()</code>	Adds a button belonging to a mutually exclusive group of buttons
<code>AddCheckTool()</code>	Adds a toggle button to the toolbar
<code>AddLabelTool()</code>	Adds a tool with icon and label
<code>AddSeparator()</code>	Adds a separator to denote groups of tool buttons
<code>AddControl()</code>	Adds any control to the toolbar. Eg. <code>wx.Button</code> , <code>wx.ComboBox</code> , etc.

ClearTools()	Removes all buttons from the toolbar
RemoveTool()	Removes give tool button from the toolbar
Realize()	Should be called after addition of tool buttons

AddTool() method takes atleast three parameters:

```
AddTool(parent, id, bitmap)
```

The parent parameter is the toolbar in which the button is added. Image icon is specified by bitmap parameter.

The general tool buttons emit EVT\_TOOL event. Other controls if added to the toolbar must be bound by respective CommandEvent binder to the event handler.

## Example

In the following example, the toolbar shows two normal tool buttons, three radio tool buttons and a combobox.

First of all, the toolbar object is activated.

```
tb = wx.ToolBar( self, -1 )
self.ToolBar = tb
```

Using AddTool() method, two tools with icons for 'New' and 'Save' are added.

```
tb.AddTool( 101, wx.Bitmap("new.png") )
tb.AddTool(102,wx.Bitmap("save.png"))
```

A group of RadioTools is then added to the toolbar, only one of which is selectable at a time.

```
right=tb.AddRadioTool(222,wx.Bitmap("right.png"))
center=tb.AddRadioTool(333,wx.Bitmap("center.png"))
justify=tb.AddRadioTool(444,wx.Bitmap("justify.png"))
```

A wx.ComboBox control is now added to the toolbar using AddControl() method. Combo box list contains the names of fonts.

```
self.combo=wx.ComboBox( tb, 555, value="Times", choices =
["Arial","Times","Courier"] )
```

Realize() method needs to be called in order to finalize the toolbar construction.

```
tb.Realize()
```

Finally, event binders for the toolbar and the combobox are registered.

```
tb.Bind(wx.EVT_TOOL, self.Onright)
tb.Bind(wx.EVT_COMBOBOX, self.OnCombo)
```

Respective event handlers' append methods process the event source. While EVT\_TOOL event's ID is displayed in the text box below the toolbar, the selected font name is added to it when EVT\_COMBOBOX event triggers.

```
def Onright(self, event):
    self.text.AppendText(str(event.GetId())+"\n")
def OnCombo(self, event):
    self.text.AppendText( self.combo.GetValue()+"\n")
```

The entire code is as follows:

```
import wx

class Mywin(wx.Frame):

    def __init__(self, parent, title):
        super(Mywin, self).__init__(parent, title=title)
        self.InitUI()

    def InitUI(self):
        menubar = wx.MenuBar()
        menu=wx.Menu()
        menubar.Append(menu,"File")
        self.SetMenuBar(menubar)

        tb = wx.ToolBar( self, -1 )
        self.ToolBar = tb

        tb.AddTool( 101, wx.Bitmap("new.png") )
        tb.AddTool(102,wx.Bitmap("save.png"))

        right=tb.AddRadioTool(222,wx.Bitmap("right.png"))
        center=tb.AddRadioTool(333,wx.Bitmap("center.png"))
        justify=tb.AddRadioTool(444,wx.Bitmap("justify.png"))

        tb.Bind(wx.EVT_TOOL, self.Onright)
```

```

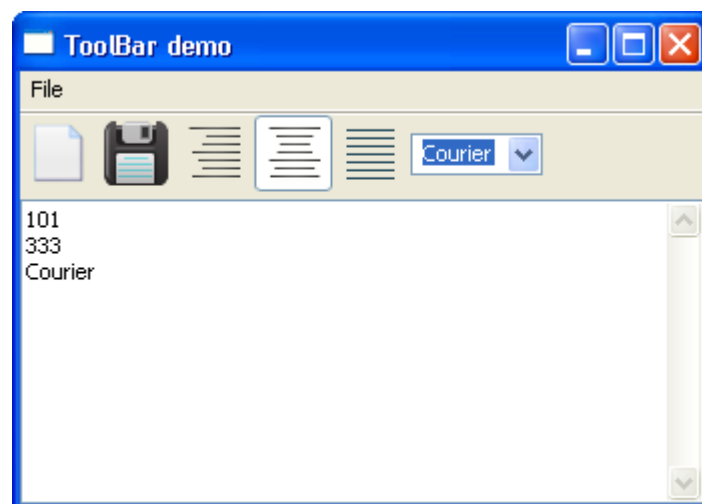
        tb.Bind(wx.EVT_COMBOBOX, self.OnCombo)
        self.combo=wx.ComboBox( tb, 555, value= "Times", choices=
["Arial","Times","Courier"])
        tb.AddControl(self.combo )
        tb.Realize()
        self.SetSize((350, 250))

        self.text=wx.TextCtrl(self, -1, style=wx.EXPAND|wx.TE_MULTILINE)
        self.Centre()
        self.Show(True)
    def Onright(self, event):
        self.text.AppendText(str(event.GetId())+"\n")
    def OnCombo(self,event):
        self.text.AppendText( self.combo.GetValue()+"\n")

ex = wx.App()
Mywin(None,'ToolBar demo')
ex.MainLoop()

```

The above code produces the following output:





## 25. wx.Dialog Class

Although a **Dialog class** object appears like a **Frame**, it is normally used as a pop-up window on top of a parent frame. The objective of a Dialog is to collect some data from the user and send it to the parent frame. Dialog frame can be modal (where it blocks the parent frame) or modeless (dialog frame can be bypassed). ShowModal() method displays dialog frame in the modal manner, while Show() makes it modeless.

wxPython has a number of preconfigured Dialog widgets such as MessageDialog, FileDialog, FontDialog, etc.

wx.Dialog supports the use of Sizers just as a wx.Frame object. Hence, a custom Dialog can be designed.

Wx.Dialog class constructor takes the following usual parameters:

```
wx.Dialog(parent, id, title, pos, size, style)
```

Default appearance of Dialog widget shows only Close box in the title bar. However, it can be customized using a combination of the following style parameters:

wx.CAPTION	Puts a caption on the dialog box
wx.DEFAULT_DIALOG_STYLE	Equivalent to a combination of wxCAPTION, wxCLOSE_BOX and wxSYSTEM_MENU
wx.RESIZE_BORDER	Displays a resizable frame around the window
wxSYSTEM_MENU	Displays a system menu
wx.CLOSE_BOX	Displays a close box on the frame
wx.MAXIMIZE_BOX	Displays a maximize box on the dialog
wx.MINIMIZE_BOX	Displays a minimize box on the dialog
wx.STAY_ON_TOP	Ensures the dialog stays on top of all other windows
wx.DIALOG_NO_PARENT	Prevents creating orphan dialog. Not recommended for modal dialogs

Two even binders are defined for this class:

EVT_CLOSE	When the dialog is being closed by the user or programmatically
EVT_INIT_DIALOG	When the dialog is being initialized

As mentioned above, the objective of Dialog is to collect data and return to the parent window. However, some useful methods are available for Dialog class.

DoOK()	Called when OK button on the dialog is pressed
ShowModal()	Shows the dialog in application modal fashion
ShowWindowModal()	Dialog is modal to top level parent window only
EndModal()	Ends a modal dialog passing the value from ShowModal invocation

One of the preconfigured dialogs is MessageDialog. It is used to display a message of one or more lines with buttons having standard IDs. Here is a select list of standard buttons on MessageDialog.

wx.OK	Shows OK button
wx.CANCEL	Shows Cancel button
wx.YES_NO	Shows Yes, No buttons
wx.YES_DEFAULT	Makes Yes button as default
wx.NO_DEFAULT	Makes No button as default
wx.ICON_EXCLAMATION	Shows an alert icon
wx.ICON_ERROR	Shows an error icon
wx.ICON_HAND	Same as wx.ICON_ERROR
wx.ICON_INFORMATION	Show an info icon
wx.ICON_QUESTION	Shows a question icon

## MessageDialog

This is declared with the following constructor:

```
wx.MessageDialog(parent, message, caption, style, pos)
```

One or more lines of the text to be displayed is the message parameter, while the caption is displayed on the title bar. Default style parameter is wx.OK|wx.ECNR. Other style parameters allow the message box to be customized.

wx.MessageBox is a convenience function to construct a message box instead of using MessageDialog.

## Example

Given below is a simple demonstration of modal and modeless behavior of Dialog. The parent window is a wx.Frame object with two buttons. Click event on the first button displays a dialog in modal fashion. Hence, any operation on the parent window is prevented till the dialog is closed. The second button displays a modeless dialog, which doesn't obstruct access to parent window. The third button displays a MessageBox.

The entire code is as follows:

```
import wx

class MyDialog(wx.Dialog):
    def __init__(self, parent, title):
        super(MyDialog, self).__init__(parent, title=title, size=(250,150))
        panel=wx.Panel(self)
        self.btn=wx.Button(panel, wx.ID_OK, label="ok", size=(50,20), pos=(75,50))

class Mywin(wx.Frame):

    def __init__(self, parent, title):
        super(Mywin, self).__init__(parent, title=title, size=(250,150))
        self.InitUI()

    def InitUI(self):
        panel=wx.Panel(self)
        btn=wx.Button(panel, label="Modal Dialog", pos=(75,10))
        btn1=wx.Button(panel, label="Modeless Dialog", pos=(75,40))
        btn2=wx.Button(panel, label="MessageBox", pos=(75,70))
        btn.Bind(wx.EVT_BUTTON, self.OnModal)
        a=btn1.Bind(wx.EVT_BUTTON, self.OnModeless)
        print a
        btn2.Bind(wx.EVT_BUTTON, self.Onmsgbox)
        self.Centre()
        self.Show(True)

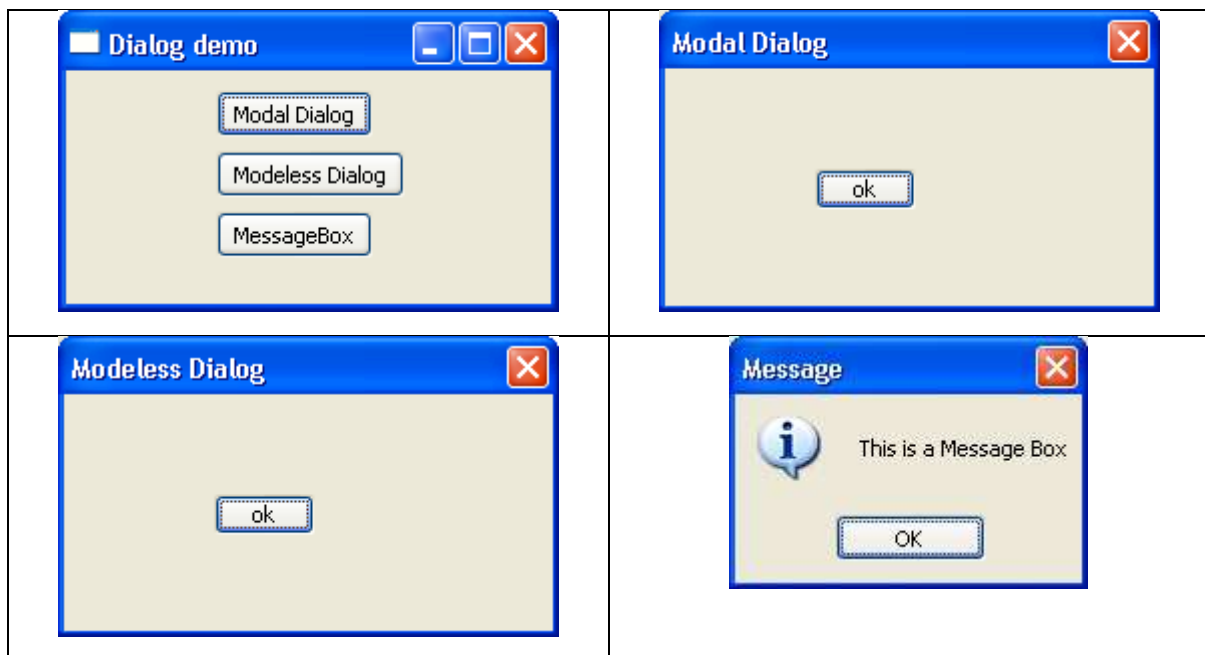
    def OnModal(self, event):
        a=MyDialog(self, "Dialog").ShowModal()
        print a

    def OnModeless(self, event):
        a=MyDialog(self, "Dialog").Show()
```

```
def Onmsgbox(self, event):
    wx.MessageBox("This is a Message Box", "Message" ,wx.OK |
wx.ICON_INFORMATION)

ex = wx.App()
Mywin(None,'MenuBar demo')
ex.MainLoop()
```

The above code produces the following output:



## wx.TextEntryDialog

Object of this class displays a dialog with one text field, a customizable label prompting the user to input and two buttons with predefined styles.

Although this dialog requests a one line input, the text box can be customized by using TextCtrl styles like password and multiline.

Contents of the text field are collected as return value when the user clicks OK button.

TextEntryDialog constructor is as follows:

```
wx.TextEntryDialog(parent, id, message, caption, value, style, pos)
```

The text to be displayed on the Dialog window is passed as the message parameter. The caption parameter is the string to be displayed in the title bar. Default string in the text box is the value parameter. TextCtrl in dialog can be configured to display password characters (wx.TE\_PASSWORD) and/or multiline (wx.TE\_MULTILINE).

Other methods of TextEntry class are as listed in the following table:

SetMaxLength()	Sets the maximum number of characters the user can enter into the text box
SetValue()	Sets the text box value programmatically
GetValue()	Returns the contents of the text box
ShowModal()	Shows dialog modally. Returns wx.ID_OK if the user confirms input, and wx.ID_CANCEL if the dialog is rejected

## Example

Top level frame in the following example shows a button and a read-only TextCtrl widget.

```
self.text = wx.TextCtrl(pnl, size=(250, 25), style=wx.TE_READONLY)
self.btn1 = wx.Button(pnl, label="Enter Text")
```

The button responds to click and invokes the OnClick() function.

```
self.Bind(wx.EVT_BUTTON, self.OnClick, self.btn1)
```

OnClick() function displays a TextEntryDialog.

```
dlg = wx.TextEntryDialog(self, 'Enter Your Name', 'Text Entry Dialog')
```

Return value of the dialog is fetched by GetValue() function and displayed in the TextCtrl object of top level frame.

```
if dlg.ShowModal() == wx.ID_OK:
    self.text.SetValue("Name entered:"+dlg.GetValue())
```

The complete code is as follows:

```
import wx

class Mywin(wx.Frame):

    def __init__(self, parent, title):
        super(Mywin, self).__init__(parent, title=title, size=(300,200))

        self.InitUI()

    def InitUI(self):
        self.count = 0
        pnl = wx.Panel(self)
```

```

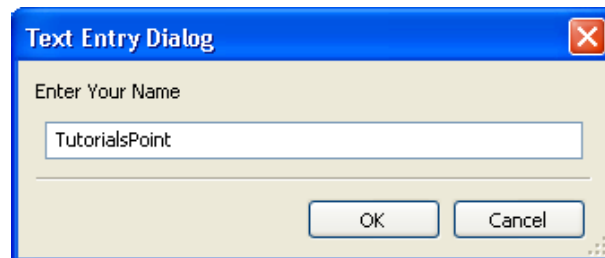
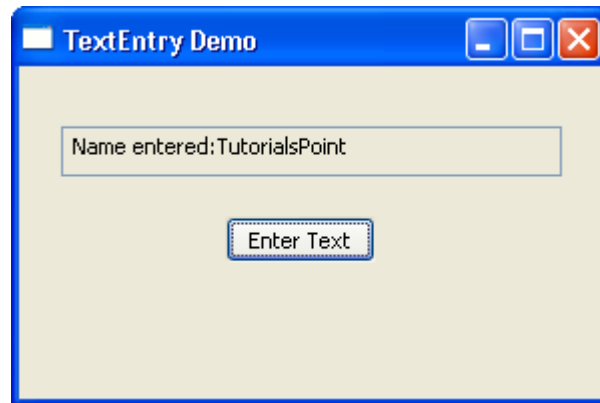
        vbox = wx.BoxSizer(wx.VERTICAL)
        hbox1 = wx.BoxSizer(wx.HORIZONTAL)
        hbox2 = wx.BoxSizer(wx.HORIZONTAL)
        self.text = wx.TextCtrl(pnl, size=(250, 25), style=wx.TE_READONLY)
        self.btn1 = wx.Button(pnl, label="Enter Text")
        self.Bind(wx.EVT_BUTTON, self.OnClick, self.btn1)
        hbox1.Add(self.text, proportion=1, flag=wx.ALIGN_CENTRE)
        hbox2.Add(self.btn1, proportion=1, flag=wx.RIGHT, border=10)
        vbox.Add((0, 30))
        vbox.Add(hbox1, flag=wx.ALIGN_CENTRE)
        vbox.Add((0, 20))
        vbox.Add(hbox2, proportion=1, flag=wx.ALIGN_CENTRE)
        pnl.SetSizer(vbox)
        self.Centre()
        self.Show(True)

    def OnClick(self, e):
        dlg = wx.TextEntryDialog(self, 'Enter Your Name', 'Text Entry Dialog')
        if dlg.ShowModal() == wx.ID_OK:
            self.text.SetValue("Name entered:"+dlg.GetValue())
        dlg.Destroy()

ex = wx.App()
Mywin(None, 'TextEntry Demo')
ex.MainLoop()

```

The above code produces the following output:



## wx.FileDialog Class

This class represents a file selector dialog. It enables the user to navigate through the file system and select a file to open or save. Appearance of the dialog is OS specific.

A file filter can also be applied to display the files of specified extensions only. Starting directory and default file name can also be set.

FileDialog constructor's prototype looks like this:

```
wx.FileDialog(parent, message, DefaultDir, DefaultFile, wildcard, style, pos, size)
```

The message represents text to be displayed. DefaultDir is the initial directory. One or more types of files can be set as file filter represented by wildcard parameter.

Style parameters defined for FileDialog are:

wx.FD_DEFAULT_STYLE	Equivalent to wxFD_OPEN
wx.FD_OPEN	This is an open dialog; default button's label of the dialog is "Open"
wx.FD_SAVE	This is a save dialog; default button's label of the dialog is "Save"

<code>wx.FD_OVERWRITE_PROMPT</code>	For save dialog only: prompts for a confirmation if a file will be overwritten
<code>wx.FD_MULTIPLE</code>	For open dialog only: allows selecting multiple files
<code>wx.FD_CHANGE_DIR:</code>	Changes the current working directory to the directory where the file(s) chosen by the user are

Member functions of `wx.FileDialog` class:

<code>GetDirectory()</code>	Returns default directory
<code>GetFileName()</code>	Returns default file name
<code>GetPath()</code>	Returns full path of selected file
<code>SetDirectory()</code>	Sets default directory
<code>SetFilename()</code>	Sets default file
<code>SetPath()</code>	Sets full path
<code>ShowModal()</code>	Displays dialog, returns <code>wx.ID_OK</code> if the user clicks OK button and <code>wx.ID_CANCEL</code> otherwise

## Example

In the following example, the top level frame shows a button and a multiline `TextCtrl`.

```
self.text = wx.TextCtrl(pnl, size=(-1,200),style=wx.TE_MULTILINE)
self.btn1 = wx.Button(pnl, label="Open a File")
```

`EVT_BUTTON` event binder registers `OnClick()` function with the button.

```
self.Bind(wx.EVT_BUTTON, self.OnClick, self.btn1)
```

`OnClick()` function displays a `FileDialog` in open mode. Its selection is returned as `dlg`. The selected file is obtained by `GetPath()` function and its contents are displayed in `TextCtrl` box on parent window.

```
def OnClick(self, e):
    wildcard="Text Files (*.txt)|*.txt"
    dlg = wx.FileDialog(self, "Choose a file", os.getcwd(), "", wildcard,
wx.OPEN)
    if dlg.ShowModal() == wx.ID_OK:
```



```

        f = open(dlg.GetPath(), 'r')
        with f:
            data = f.read()
            self.text.SetValue(data)

```

The complete code is as follows:

```

import wx
import os
class Mywin(wx.Frame):

    def __init__(self, parent, title):
        super(Mywin, self).__init__(parent, title=title)

        self.InitUI()

    def InitUI(self):
        self.count = 0
        pnl = wx.Panel(self)
        vbox = wx.BoxSizer(wx.VERTICAL)
        hbox1 = wx.BoxSizer(wx.HORIZONTAL)
        hbox2 = wx.BoxSizer(wx.HORIZONTAL)
        self.text = wx.TextCtrl(pnl, size=(-1,200),style=wx.TE_MULTILINE)
        self.btn1 = wx.Button(pnl, label="Open a File")
        self.Bind(wx.EVT_BUTTON, self.OnClick, self.btn1)
        hbox1.Add(self.text, proportion=1, flag=wx.ALIGN_CENTRE)
        hbox2.Add(self.btn1, proportion=1, flag=wx.RIGHT, border=10)
        vbox.Add(hbox2, proportion=1, flag=wx.ALIGN_CENTRE)

        vbox.Add(hbox1, proportion=1,flag=wx.EXPAND|wx.ALIGN_CENTRE)

        pnl.SetSizer(vbox)
        self.Centre()
        self.Show(True)

    def OnClick(self, e):
        wildcard="Text Files (*.txt)|*.txt"
        dlg = wx.FileDialog(self, "Choose a file", os.getcwd(), "", wildcard,
wx.OPEN)

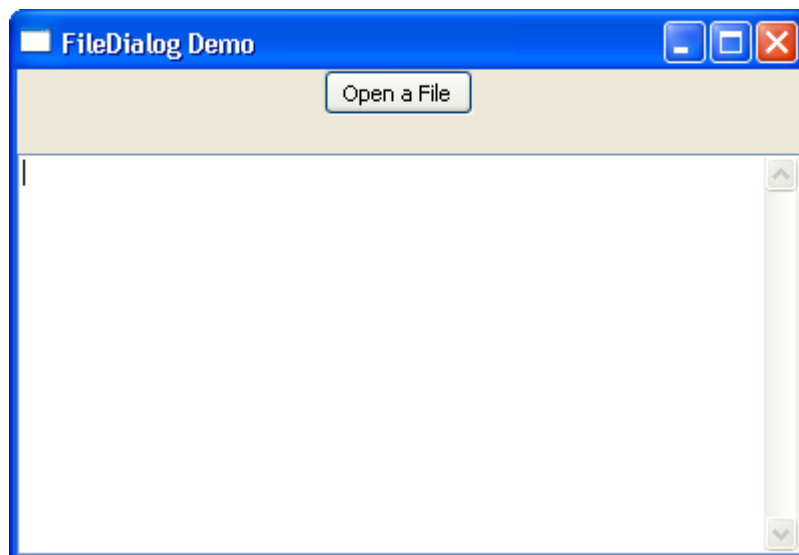
```

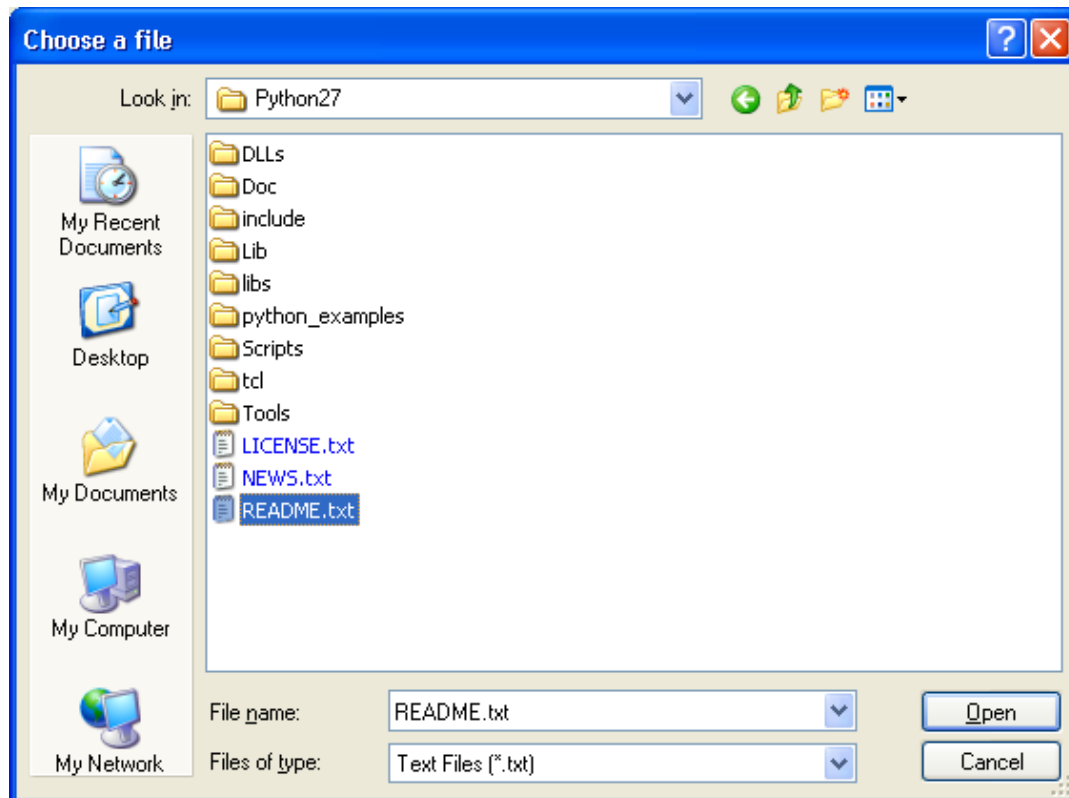
```
        if dlg.ShowModal() == wx.ID_OK:
            f = open(dlg.GetPath(), 'r')
            with f:
                data = f.read()
                self.text.SetValue(data)

        dlg.Destroy()

ex = wx.App()
Mywin(None, 'FileDialog Demo')
ex.MainLoop()
```

The above code produces the following output:





## wx.FontDialog Class

The object of this class is a font chooser dialog. Appearance of this dialog too is OS specific. Attributes, such as name, size, weight, etc. of the selected font are returned as the return value of this dialog.

Fontdata parameter required for this class constructor is used to initialize these attributes.

```
wx.FontDialog(parent, data)
```

GetFontData() method of this class contains the parameters of the selected font.

The following code demonstrating the use of FontDialog has a button and a label (StaticText object).

```
self.text = wx.StaticText(pnl, label="hello")
self.btn1 = wx.Button(pnl, label="Choose Font")
```

The button when clicked triggers OnClick() event handler function.

```
def OnClick(self, e):
    dlg = wx.FontDialog(self,wx.FontData())
    if dlg.ShowModal() == wx.ID_OK:
        data = dlg.GetFontData()
        font = data.GetChosenFont()
        self.text.SetFont(font)

    dlg.Destroy()
```

The chosen font is then applied to label's text.

The complete code is as follows:

```
import wx
import os
class Mywin(wx.Frame):

    def __init__(self, parent, title):
        super(Mywin, self).__init__(parent, title=title,size=(250,200))

        self.InitUI()

    def InitUI(self):
        self.count = 0
        pnl = wx.Panel(self)
        vbox = wx.BoxSizer(wx.VERTICAL)
        hbox1 = wx.BoxSizer(wx.HORIZONTAL)
        hbox2 = wx.BoxSizer(wx.HORIZONTAL)
        self.text = wx.StaticText(pnl, label="hello")
        self.btn1 = wx.Button(pnl, label="Choose Font")
```

```

self.Bind(wx.EVT_BUTTON, self.OnClick, self.btn1)
hbox1.Add(self.text, proportion=1, flag=wx.ALIGN_CENTRE)
hbox2.Add(self.btn1, proportion=1, flag=wx.ALIGN_CENTRE, border=10)
vbox.Add(hbox2, flag=wx.ALIGN_CENTRE)

vbox.Add(hbox1, proportion=1, flag=wx.ALIGN_CENTRE)

pnl.SetSizer(vbox)
self.Centre()
self.Show(True)

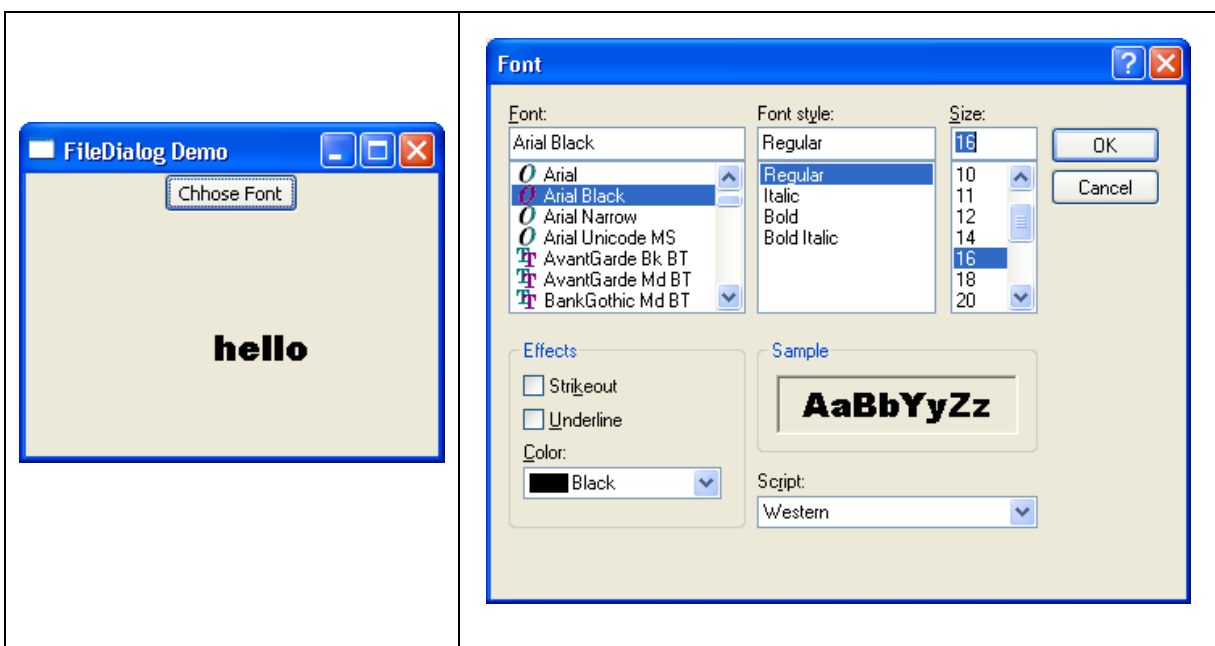
def OnClick(self, e):
    dlg = wx.FontDialog(self, wx.FontData())
    if dlg.ShowModal() == wx.ID_OK:
        data = dlg.GetFontData()
        font = data.GetChosenFont()
        self.text.SetFont(font)

    dlg.Destroy()

ex = wx.App()
Mywin(None, 'FileDialog Demo')
ex.MainLoop()

```

The above code produces the following output:



## 26. wx.Notebook Class

wxPython class library contains various 'book' control. A **book control** allows the user to switch between various panels in a frame. This is especially useful when a large amount of data is to be presented.

Book classes are inherited from **wx.BookCtrlBase** class. The following type of book controls are available:

- wx.Notebook
- wx.Choicebook
- wx.Listbook
- wx.Treebook

**wx.Notebook** widget presents a tabbed control. One Notebook object in a frame has one or more tabs (called Pages), each of them having a panel showing the layout of controls. The user can switch between pages by clicking on the respective tab title.

Notebook class constructor takes familiar parameters as the following:

```
wx.Notebook(parent, id, pos, size, style)
```

The following style parameters are available for customization of the widget:

wx.NB_TOP	Places tabs on the top side
wx.NB_LEFT	Places tabs on the left side
wx.NB_RIGHT	Places tabs on the right side
wx.NB_BOTTOM	Places tabs under the notebook pages instead of above the notebook pages
wx.NB_FIXEDWIDTH	All tabs will have the same width
wx.NB_MULTILINE	There can be several rows of tabs

Member functions of Notebook class:

OnSelChange()	Handler function called when the page selection is changed
SetPadding()	Sets the amount space around each page's icon and label, in pixels
GetSelection()	Returns the currently selected page

SetSelection()	Sets the selection to the given page, returning the previous selection
AddPage()	Adds a new page
DeletePage()	Deletes a page of given index
InsertPage()	Inserts a new tab at the given index
RemovePage()	Removes the page without deleting

Event binders defined for this class are:

EVT_NOTEBOOK_PAGE_CHANGED()	The page selection was changed
EVT_NOTEBOOK_PAGE_CHANGING()	The page selection is about to be changed

## Example

The following example deploys a Notebook control in a top level frame.

```
nb=wx.Notebook(self)
```

Two classes based on wx.Panel are designed. The first, puts a multi-line TextCtrl in it.

```
class MyPanel1(wx.Panel):
    def __init__(self, parent):
        super(MyPanel1, self).__init__(parent)
        text=wx.TextCtrl(self,style=wx.TE_MULTILINE,size=(250,150))
```

The second, shows a RadioBox having three radio buttons.

```
class MyPanel2(wx.Panel):
    def __init__(self, parent):
        super(MyPanel2, self).__init__(parent)
        lblList = ['Value X', 'Value Y', 'Value Z']
        rbox=wx.RadioBox(self, label='RadioBox', pos=(25,10), choices =
        lblList, majorDimension=1, style=wx.RA_SPECIFY_ROWS)
```

Objects of these two panel classes are added as pages in Notebook on the top level frame.

```
nb.AddPage(MyPanel1(nb),"Editor")
nb.AddPage(MyPanel2(nb),"RadioButtons")
```

The complete code is as follows:

```
import wx

class MyDialog(wx.Dialog):
    def __init__(self, parent, title):
        super(MyDialog, self).__init__(parent, title=title, size=(250,150))
        panel=wx.Panel(self)
        self.btn=wx.Button(panel,wx.ID_OK,label="ok",size=(50,20),pos=(75,50))

class Mywin(wx.Frame):

    def __init__(self, parent, title):
        super(Mywin, self).__init__(parent, title=title, size=(250,150))
        self.InitUI()

    def InitUI(self):
        nb=wx.Notebook(self)
        nb.AddPage(MyPanel1(nb),"Editor")
        nb.AddPage(MyPanel2(nb),"RadioButtons")
        self.Centre()
        self.Show(True)

class MyPanel1(wx.Panel):
    def __init__(self, parent):
        super(MyPanel1, self).__init__(parent)
        text=wx.TextCtrl(self,style=wx.TE_MULTILINE,size=(250,150))

class MyPanel2(wx.Panel):
    def __init__(self, parent):
        super(MyPanel2, self).__init__(parent)
        lblList = ['Value X', 'Value Y', 'Value Z']
        rbox=wx.RadioBox(self, label='RadioBox', pos=(25,10), choices =
lblList,
                                majorDimension=1, style=wx.RA_SPECIFY_ROWS)

ex = wx.App()
Mywin(None,'NoteBook demo')
ex.MainLoop()
```



The above code produces the following output:



## 27. Dockable Windows

**wxAui** is an Advanced User Interface library incorporated in wxWidgets API. Wx.aui.AuiManager the central class in AUI framework.

**AuiManager** manages the panes associated with a particular frame using each panel's information in wx.aui.AuiPanelInfo object. Let us learn about various properties of PanelInfo object control docking and floating behavior.

Putting dockable windows in the top level frame involves the following steps:

First, create an AuiManager object.

```
self.mgr = wx.aui.AuiManager(self)
```

Then, a panel with required controls is designed.

```
pnl=wx.Panel(self)
pbox=wx.BoxSizer(wx.HORIZONTAL)
text1 = wx.TextCtrl(pnl, -1, "Dockable", style=wx.NO_BORDER | wx.TE_MULTILINE)
pbox.Add(text1, 1, flag=wx.EXPAND)
pnl.SetSizer(pbox)
```

The following parameters of AuiPanelInfo are set.

- Direction: Top, Bottom, Left, Right, or Center
- Position: More than one pane can be placed inside a dockable region. Each is given a position number.
- Row: More than one pane appears in one row. Just like more than one toolbar appearing in the same row.
- Layer: Panes can be placed in layers.

Using this PanelInfo, the designed panel is added into the manager object.

```
info1=wx.aui.AuiPaneInfo().Bottom()
self.mgr.AddPane(pnl,info1)
```

Rest of the top level window may have other controls as usual.

The complete code is as follows:

```
import wx
import wx.aui

class Mywin(wx.Frame):

    def __init__(self, parent, title):
        super(Mywin, self).__init__(parent, title=title, size=(300,300))

        self.mgr = wx.aui.AuiManager(self)

        pnl=wx.Panel(self)
        pbox=wx.BoxSizer(wx.HORIZONTAL)
        text1 = wx.TextCtrl(pnl, -1, "Dockable", style=wx.NO_BORDER | wx.TE_MULTILINE)
        pbox.Add(text1, 1, flag=wx.EXPAND)
        pnl.SetSizer(pbox)

        info1=wx.aui.AuiPaneInfo().Bottom()
        self.mgr.AddPane(pnl, info1)
        panel=wx.Panel(self)
        text2 = wx.TextCtrl(panel, size=(300,200), style= wx.NO_BORDER |
wx.TE_MULTILINE)
        box=wx.BoxSizer(wx.HORIZONTAL)
        box.Add(text2, 1, flag=wx.EXPAND)

        panel.SetSizerAndFit(box)
        self.mgr.Update()

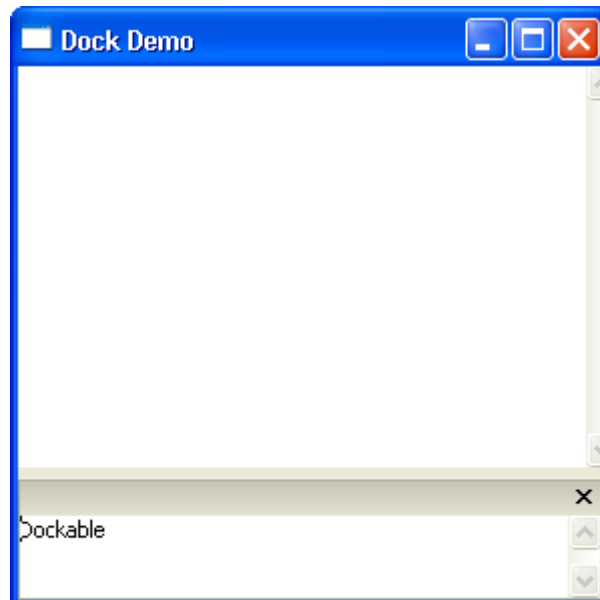
        self.Bind(wx.EVT_CLOSE, self.OnClose)
        self.Centre()
        self.Show(True)

    def OnClose(self, event):
        self.mgr.UnInit()
        self.Destroy()

app = wx.App()
```

```
Mywin(None, "Dock Demo")  
  
app.MainLoop()
```

The above code produces the following output:



## 28. Multiple Document Interface

A typical GUI application may have multiple windows. Tabbed and stacked widgets allow to activate one such window at a time. However, many a times this approach may not be useful as view of other windows is hidden.

One way to display multiple windows simultaneously is to create them as independent windows. This is called as SDI (**Single Document Interface**). This requires more memory resources as each window may have its own menu system, toolbar, etc.

MDI framework in wxPython provides a wx.MDIParentFrame class. Its object acts as a container for multiple child windows, each an object of wx.MDICHildFrame class.

Child windows reside in the MDIClientWindow area of the parent frame. As soon as a child frame is added, the menu bar of the parent frame shows a Window menu containing buttons to arrange the children in a cascaded or tiled manner.

### Example

The following example illustrates the uses of MDIParentFrame as top level window. A Menu button called NewWindow adds a child window in the client area. Multiple windows can be added and then arranged in a cascaded or tiled order.

The complete code is as follows:

```
import wx

class MDIFrame(wx.MDIParentFrame):
    def __init__(self):
        wx.MDIParentFrame.__init__(self, None, -1, "MDI Parent", size=(600,400))
        menu = wx.Menu()
        menu.Append(5000, "&New Window")
        menu.Append(5001, "E&xit")
        menubar = wx.MenuBar()
        menubar.Append(menu, "&File")
        self.SetMenuBar(menubar)
        self.Bind(wx.EVT_MENU, self.OnNewWindow, id=5000)
        self.Bind(wx.EVT_MENU, self.OnExit, id=5001)

    def OnExit(self, evt):
        self.Close(True)

    def OnNewWindow(self, evt):
        win = wx.MDICHildFrame(self, -1, "Child Window")
```

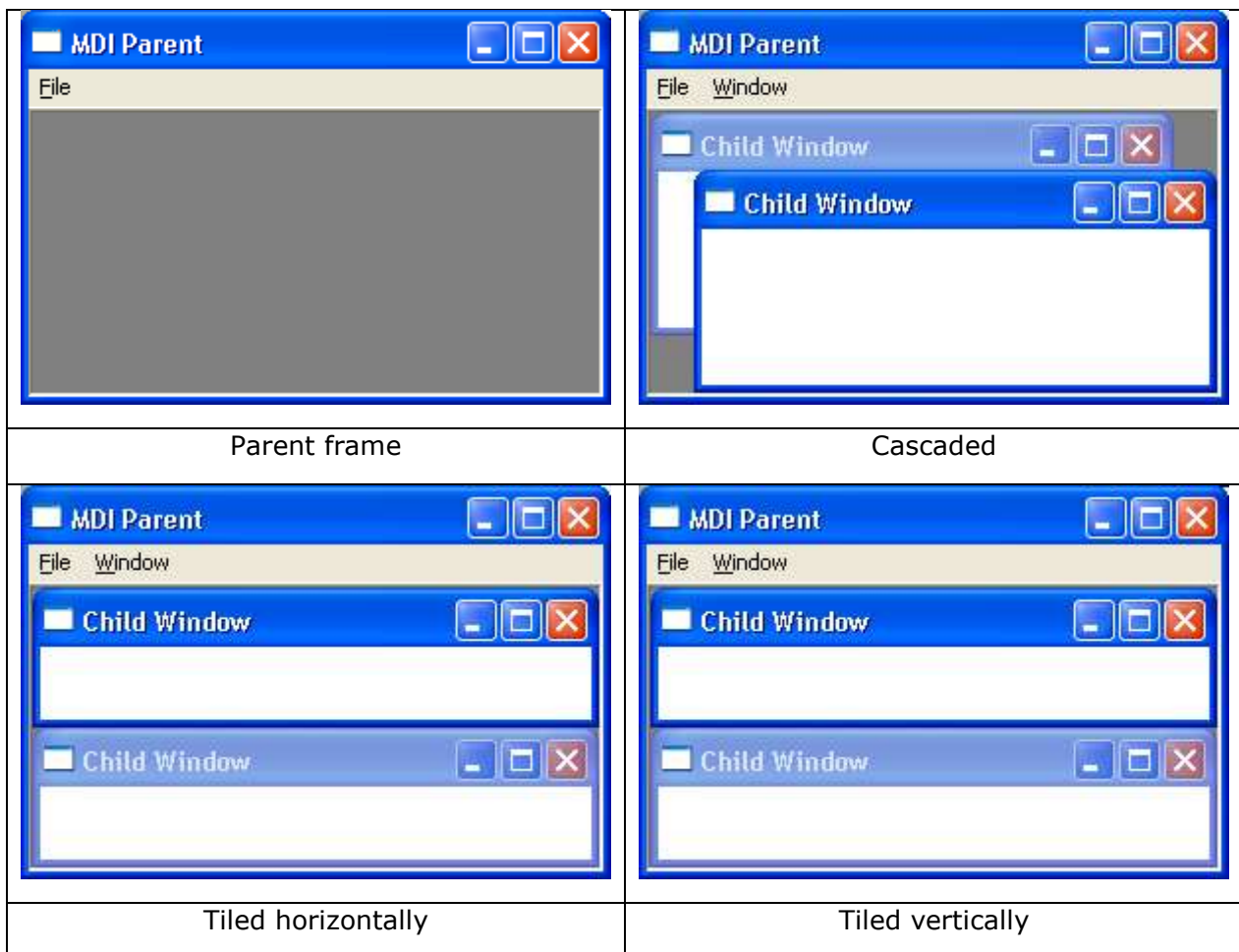
```

win.Show(True)

app = wx.App()
frame = MDIFrame()
frame.Show()
app.MainLoop()

```

The above code produces the following output:



## 29. wx.SplitterWindow Class

Object of this class is a layout manager, which holds two subwindows whose size can be changed dynamically by dragging the boundaries between them. The **Splitter control** gives a handle that can be dragged to resize the controls.

**wx.SplitterWindow** class has a very basic constructor with all parameters having usual default values.

```
wx.SplitterWindow(self, id, pos, size, style)
```

The list of predefined values for style parameter is as follows:

wxSP_3D	Draws a 3D effect border and sash
wxSP_THIN_SASH	Draws a thin sash
wxSP_3DSASH	Draws a 3D effect sash (part of default style)
wxSP_BORDER	Draws a standard border
wxSP_NOBORDER	No border (default)
wxSP_PERMIT_UNSPPLIT	Always allow to unsplit, even with the minimum pane size other than zero

Event binders for SplitterWindow class:

EVT_SPLITTER_SASH_POS_CHANGING()	The sash position is in the process of being changed
EVT_SPLITTER_SASH_POS_CHANGED()	The sash position was changed
EVT_SPLITTER_UNSPPLIT()	The splitter has been just unsplit
EVT_SPLITTER_DCLICK()	The sash was double clicked. The default behavior is to unsplit the window when this happens

The following code demonstrates the functioning of SplitterWindow. The splitter object is added to the top level frame.

```
splitter = wx.SplitterWindow(self, -1)
```

A Panel is designed to hold a multi-line TextCtrl object.

```
b=wx.BoxSizer(wx.HORIZONTAL)
self.text=wx.TextCtrl(panel1,style=wx.TE_MULTILINE)
b.Add(self.text, 1, wx.EXPAND)
panel1.SetSizerAndFit(b)
```

A ListBox object is placed in another panel.

```
panel2 = wx.Panel(splitter, -1)
languages = ['C', 'C++', 'Java', 'Python', 'Perl', 'JavaScript', 'PHP',
'VB.NET', 'C#']
lst = wx.ListBox(panel2, size=(100,300), choices= languages,
style=wx.LB_SINGLE)
hbox1 = wx.BoxSizer(wx.HORIZONTAL)
hbox1.Add(lst,1)
panel2.SetSizer(hbox1)
```

The splitter object is vertically split and two panels are added to two subwindows. The width of subwindows can be resized with the help of sash.

```
splitter.SplitVertically(panel2, panel1)
```

The complete listing of code is as follows:

```
import wx
class Mywin(wx.Frame):

    def __init__(self, parent, title):
        super(Mywin, self).__init__(parent, title=title,size=(350,300))

        splitter = wx.SplitterWindow(self, -1)
        panel1 = wx.Panel(splitter, -1)
        b=wx.BoxSizer(wx.HORIZONTAL)
        self.text=wx.TextCtrl(panel1,style=wx.TE_MULTILINE)
        b.Add(self.text, 1, wx.EXPAND)
        panel1.SetSizerAndFit(b)
```



```

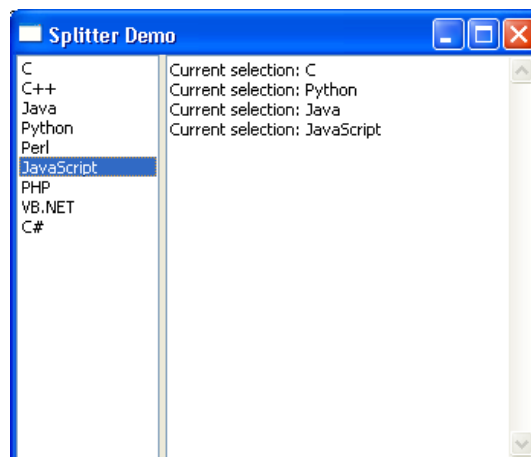
        panel2 = wx.Panel(splitter, -1)
        languages = ['C', 'C++', 'Java', 'Python', 'Perl', 'JavaScript', 'PHP',
                    'VB.NET', 'C#']
        lst = wx.ListBox(panel2, size=(100,300), choices= languages,
                        style=wx.LB_SINGLE)
        hbox1 = wx.BoxSizer(wx.HORIZONTAL)
        hbox1.Add(lst,1)
        panel2.SetSizer(hbox1)
        splitter.SplitVertically(panel2, panel1)
        self.Centre()
        self.Bind(wx.EVT_LISTBOX, self.onListBox, lst)
        self.Show(True)

    def onListBox(self, event):
        self.text.AppendText( "Current selection: " +
            event.GetEventObject().GetStringSelection() + "\n")

ex = wx.App()
Mywin(None, 'Splitter Demo')
ex.MainLoop()

```

The above code produces the following output:



# 30. Drawing API

**GDI+** (Graphics Drawing Interface), **CoreGraphics** and **Cairo libraries** form the framework of drawing API in wxPython. `wx.GraphicsContext` is the primary drawable object, using which various Device Context objects are created.

`wx.DC` is an abstract class. Its derived classes are used to render graphics and text on different devices. The Device Context classes are:

- **wx.ScreenDC** - Use this to paint on the screen, as opposed to an individual window.
- **wx.ClientDC** - Use this to paint on the client area of the window (the part without borders and other decorations), but do not use it from within an `wx.PaintEvent`.
- **wx.PaintDC** - Use this to paint on the client area of the window, but *only* from within a `wx.PaintEvent`.
- **wx.WindowDC** - Use this to paint on the whole area of the window, including decorations. This may not be available on non-Windows platforms.

Drawing API of wxPython offers different functions for drawing shape, text and image. Objects required for drawing purpose, like Colour, Pen, Brush and Font can also be constructed using GDI classes.

## wx.Colour Class

---

Colour object represents combination of RGB (RED, Green and Blue) intensity values, each on the scale of 0-255. There are a few predefined colour objects like:

- `wx.BLACK`
- `wx.BLUE`
- `wx.CYAN`
- `wx.GREEN`
- `wx.YELLOW`
- `wx.LIGHT_GREY`
- `wx.RED`
- `wx.WHITE`

Color with custom combination of RGB values is formed as **wx.Colour object**.

`wx.Colour(r,g,b)`

## wx.Pen Class

---

Pen object determines the colour, width and style of the shape of graphics like line, rectangle, circle etc.

**Predefined Pen objects** are:

- wxBLACK\_DASHED\_PEN
- wxBLACK\_PEN
- wxBLUE\_PEN
- wxCYAN\_PEN
- wxGREEN\_PEN
- wxYELLOW\_PEN
- wxGREY\_PEN
- wxLIGHT\_GREY\_PEN
- wxMEDIUM\_GREY\_PEN
- wxRED\_PEN
- wxTRANSPARENT\_PEN
- wxWHITE\_PEN

**Predefined Pen styles** are:

- wx.SOLID
- wx.DOT
- wx.LONG\_DASH
- wx.SHORT\_DASH
- wx.DOT\_DASH
- wx.TRANSPARENT

## wx.Brush Class

---

Brush is another elementary graphics object required to fill the backgrounds of shapes such as rectangle, ellipse, circle etc.

A custom Brush object requires wx.Colour and Brush style parameters. The following is a list of predefined brush styles:

- wx.SOLID
- wx.STIPPLE
- wx.BDIAGONAL\_HATCH
- wx.CROSSDIAG\_HATCH
- wx.FDIAGONAL\_HATCH
- wx.CROSS\_HATCH
- wx.HORIZONTAL\_HATCH

- wx.VERTICAL\_HATCH
- wx.TRANSPARENT

wxPython has a number of functions that facilitate drawing different shapes, text and image.

DrawRectangle()	Draws a rectangle of given dimensions
DrawCircle()	Draws a circle at the given point as center and radius
DrawEllipse()	Draws an ellipse with the given x and y radius
DrawLine()	Draws a line between two wx.Point objects
DrawBitmap()	Draw an image at the given position
DrawText()	Displays the given text at the specified position

## Example

The above functions are implemented in the following example, making use of Pen, Brush, Colour and Font objects.

The complete code is as follows:

```
import wx

class Mywin(wx.Frame):

    def __init__(self, parent, title):
        super(Mywin, self).__init__(parent, title=title, size=(500,300))
        self.InitUI()

    def InitUI(self):
        self.Bind(wx.EVT_PAINT, self.OnPaint)
        self.Centre()
        self.Show(True)
    def OnPaint(self, e):
        dc = wx.PaintDC(self)
        brush = wx.Brush("white")
        dc.SetBackground(brush)
        dc.Clear()

        dc.DrawBitmap(wx.Bitmap("python.jpg"),10,10,True)
        color=wx.Colour(255,0,0)
```

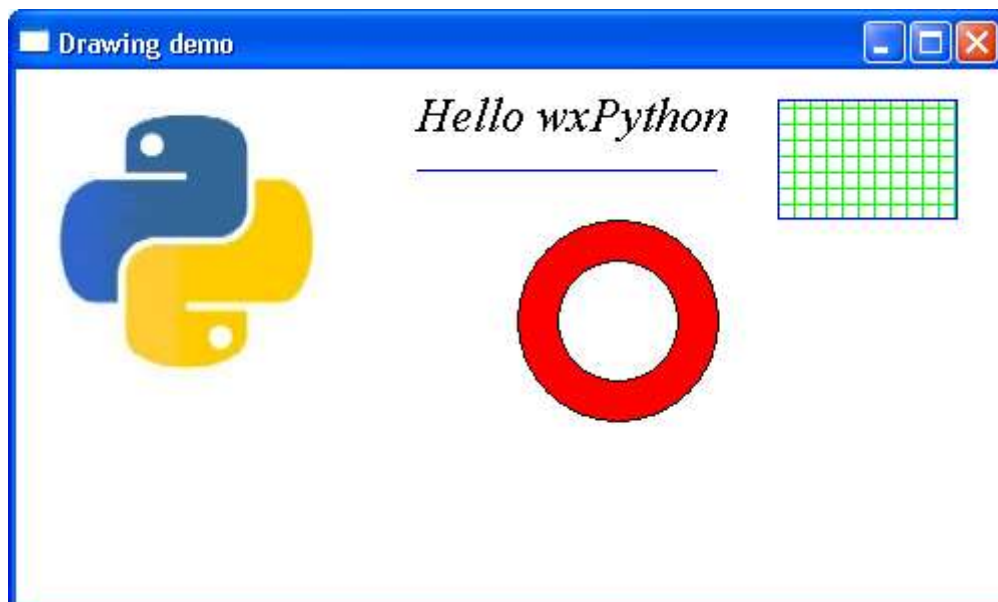
```

b=wx.Brush(color)
dc.SetBrush(b)
dc.DrawCircle(300,125,50)
dc.SetBrush(wx.Brush(wx.Colour(255,255,255)))
dc.DrawCircle(300,125,30)
font = wx.Font(18, wx.ROMAN, wx.ITALIC, wx.NORMAL)
dc.SetFont(font)
dc.DrawText("Hello wxPython",200,10)
pen=wx.Pen(wx.Colour(0,0,255))
dc.SetPen(pen)
dc.DrawLine(200,50,350,50)
dc.SetBrush(wx.Brush(wx.Colour(0,255,0), wx.CROSS_HATCH))
dc.DrawRectangle(380, 15, 90, 60)

ex = wx.App()
Mywin(None,'Drawing demo')
ex.MainLoop()

```

The above code produces the following output:



# 31. wx.HTMLWindow Class

wxHTML library contains classes for parsing and displaying HTML content. Although this is not intended to be a full-featured browser, wx.HtmlWindow object is a generic HTML viewer.

HtmlWindow class constructor takes a familiar look:

(Parent, id, pos, size, style)
--------------------------------

This class supports the following styles:

wxHW_SCROLLBAR_NEVER	Never display scrollbars, not even when the page is larger than the window
wxHW_SCROLLBAR_AUTO	Display scrollbars only if the page size exceeds window's size
wxHW_NO_SELECTION	Don't allow the user to select text

Following event binders are available for this class:

EVT_HTML_CELL_CLICKED	A wxHtmlCell was clicked
EVT_HTML_CELL_HOVER	The mouse passed over a wxHtmlCell
EVT_HTML_LINK_CLICKED	A wxHtmlCell which contains an hyperlink was clicked

Following member functions of this class are frequently used:

AppendToPage()	Appends HTML fragment to currently displayed text and refreshes the window
HistoryBack()	Goes back to the previously visited page
HistoryForward()	Goes to the next page in history
LoadPage()	Loads a HTML file
OnLinkClicked()	Called when a hyperlink is clicked
SetPage()	Sets text tagged with HTML tags as page contents

The following code displays a simple HTML browser. On running the code, a TextEntry Dialog pops up asking a URL to be entered. LoadPage() method of wx.HtmlWindow class displays the contents in the window.

```
import wx
import wx.html

class MyHtmlFrame(wx.Frame):
    def __init__(self, parent, title):
        wx.Frame.__init__(self, parent, -1, title, size=(600,400))
        html = wx.html.HtmlWindow(self)
        if "gtk2" in wx.PlatformInfo:
            html.SetStandardFonts()
        dlg = wx.TextEntryDialog(self, 'Enter a URL','HTMLWindow')
        if dlg.ShowModal() == wx.ID_OK:
            html.LoadPage(dlg.GetValue())

app = wx.App()
frm = MyHtmlFrame(None, "Simple HTML Browser")
frm.Show()
app.MainLoop()
```

## 32. ListBox & ListCtrl Class

A wx.ListBox widget presents a vertically scrollable list of strings. By default, a single item in the list is selectable. However, it can be customized to be multi-select.

ListCtrl widget is a highly enhanced list display and selection tool. List of more than one column can be displayed in Report view, List view or Icon view.

ListBox constructor has the following definition:

```
wx.ListBox(parent, id, pos, size, choices, style)
```

Choices parameter is the list of strings used to populate the list.

wx.ListBox object is customizable with the following style parameters:

wxLB_SINGLE	Single-selection list
wxLB_MULTIPLE	Multiple-selection list: the user can toggle multiple items on and off
wxLB_EXTENDED	Extended-selection list: the user can extend the selection by using SHIFT or CTRL keys together with the cursor movement keys or the mouse
wxLB_HSCROLL	Create horizontal scrollbar if contents are too wide
wxLB_ALWAYS_SB	Always show a vertical scrollbar
wxLB_NEEDED_SB	Only creates a vertical scrollbar if needed
wxLB_SORT	The listbox contents are sorted in an alphabetical order

wx.ListBox class methods:

DeSelect()	Deselects an item in the list box
InsertItem()	Inserts a given string at the specified position
SetFirstItem()	Sets a string at the given index as first in the list
IsSorted()	Returns true if wxLB_SORT style is used
GetString()	Returns the string at the selected index
SetString()	Sets the label for an item at the given index

EVT\_LISTBOX binder triggers the handler when an item in the list is selected or when the selection changes programmatically. Handler function bound by EVT\_LISTBOX\_DCLICK is invoked when a double-click event on the list box item occurs.



## Example

In the following example, a ListBox control and a TextCtrl object are respectively placed in the left and the right portion of a horizontal box sizer. ListBox is populated with strings in languages[] list object.

```
languages = ['C', 'C++', 'Java', 'Python', 'Perl',
             'JavaScript', 'PHP', 'VB.NET', 'C#']

self.text=wx.TextCtrl(panel,style=wx.TE_MULTILINE)

lst = wx.ListBox(panel, size=(100,-1), choices= languages, style=wx.LB_SINGLE)
```

Two objects are placed in a horizontal box sizer.

```
box=wx.BoxSizer(wx.HORIZONTAL)
box.Add(lst,0,wx.EXPAND)
box.Add(self.text, 1, wx.EXPAND)
```

ListBox control is linked to onListBox() handler with EVT\_LISTBOX binder.

```
self.Bind(wx.EVT_LISTBOX, self.onListBox, lst)
```

The handler appends selected string into multiline TextCtrl on the right.

```
def onListBox(self, event):
    self.text.AppendText( "Current selection: "+
event.GetEventObject().GetStringSelection() + "\n")
```

The complete code is as follows:

```
import wx
class Mywin(wx.Frame):

    def __init__(self, parent, title):
        super(Mywin, self).__init__(parent, title=title,size=(350,300))

        panel=wx.Panel(self)
        box=wx.BoxSizer(wx.HORIZONTAL)

        self.text=wx.TextCtrl(panel,style=wx.TE_MULTILINE)

        languages = ['C', 'C++', 'Java', 'Python', 'Perl',
                    'JavaScript', 'PHP', 'VB.NET', 'C#']
        lst = wx.ListBox(panel, size=(100,-1), choices= languages, style=wx.LB_SINGLE)
```

```

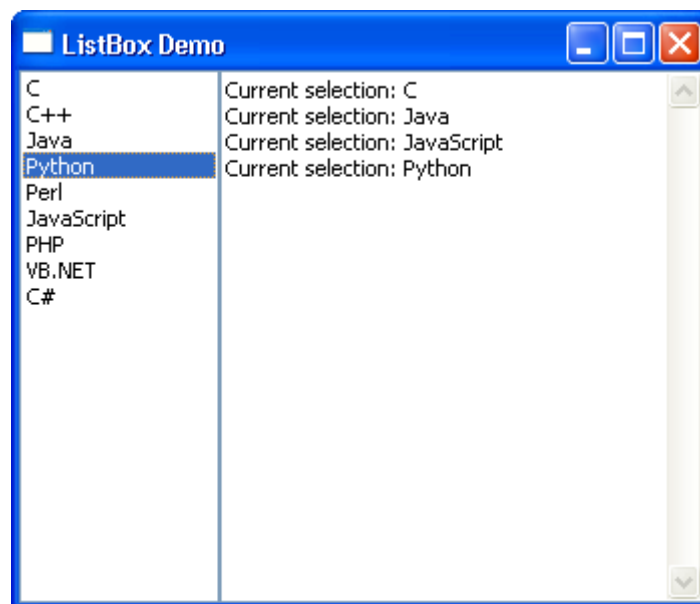
        box.Add(lst,0,wx.EXPAND)
        box.Add(self.text, 1, wx.EXPAND)
        panel.SetSizer(box)
        panel.Fit()
        self.Centre()
        self.Bind(wx.EVT_LISTBOX, self.onListBox, lst)
        self.Show(True)

    def onListBox(self, event):
        self.text.AppendText( "Current selection:
"+event.GetEventObject().GetStringSelection()+"\n")

ex = wx.App()
Mywin(None,'ListBox Demo')
ex.MainLoop()

```

The above code produces the following output:



**wx.ListCtrl** is an enhanced, and therefore, more complex widget. Where a **ListBox** shows only one column, **ListCtrl** can contain multiple columns. The appearance of **ListCtrl** widget is controlled by the following style parameters:

wx.LC_LIST	Multicolumn list view, with optional small icons. Columns are computed automatically
wx.LC_REPORT	Single or multicolumn report view, with optional header

<code>wx.LC_VIRTUAL</code>	The application provides items text on demand. May only be used with <code>wx.LC_REPORT</code>
<code>wx.LC_ICON</code>	Large icon view, with optional labels
<code>wx.LC_SMALL_ICON</code>	Small icon view, with optional labels
<code>wx.LC_ALIGN_LEFT</code>	Icons align to the left
<code>wx.LC_EDIT_LABELS</code>	Labels are editable: the application will be notified when editing starts
<code>wx.LC_NO_HEADER</code>	No header in report mode
<code>wx.LC_SORT_ASCENDING</code>	Sort in ascending order
<code>wx.LC_SORT_DESCENDING</code>	Sort in descending order
<code>wx.LC_HRULES</code>	Draws light horizontal rules between the rows in report mode
<code>wx.LC_VRULES</code>	Draws light vertical rules between the columns in report mode

## Example

A ListCtrl widget in report view is constructed in the following example.

```
self.list = wx.ListCtrl(panel, -1, style=wx.LC_REPORT)
```

Header columns are created by `InsertColumn()` method which takes the column number, caption, style and width parameters.

```
self.list.InsertColumn(0, 'name', width=100)
self.list.InsertColumn(1, 'runs', wx.LIST_FORMAT_RIGHT, 100)
self.list.InsertColumn(2, 'wkts', wx.LIST_FORMAT_RIGHT, 100)
```

A list of tuples, each containing three strings, called `players[]` stores the data which is used to populate columns of the ListCtrl object.

New row starts with `InsertStringItem()` method which returns the index of the current row. Use of `sys.maxint` gives the row number after the last row. Using the index, other columns are filled by `SetStringItem()` method.

```
for i in players:
    index = self.list.InsertStringItem(sys.maxint, i[0])
    self.list.SetStringItem(index, 1, i[1])
    self.list.SetStringItem(index, 2, i[2])
```

The complete code for the example is:

```
import sys
import wx

players = [('Tendulkar', '15000', '100'), ('Dravid', '14000', '1'),
           ('Kumble', '1000', '700'), ('KapilDev', '5000', '400'),
           ('Ganguly', '8000', '50')]

class Mywin(wx.Frame):

    def __init__(self, parent, title):
        super(Mywin, self).__init__(parent, title=title)

        panel=wx.Panel(self)
        box=wx.BoxSizer(wx.HORIZONTAL)

        self.list = wx.ListCtrl(panel, -1, style=wx.LC_REPORT)
        self.list.InsertColumn(0, 'name', width=100)
        self.list.InsertColumn(1, 'runs', wx.LIST_FORMAT_RIGHT, 100)
        self.list.InsertColumn(2, 'wkts', wx.LIST_FORMAT_RIGHT, 100)

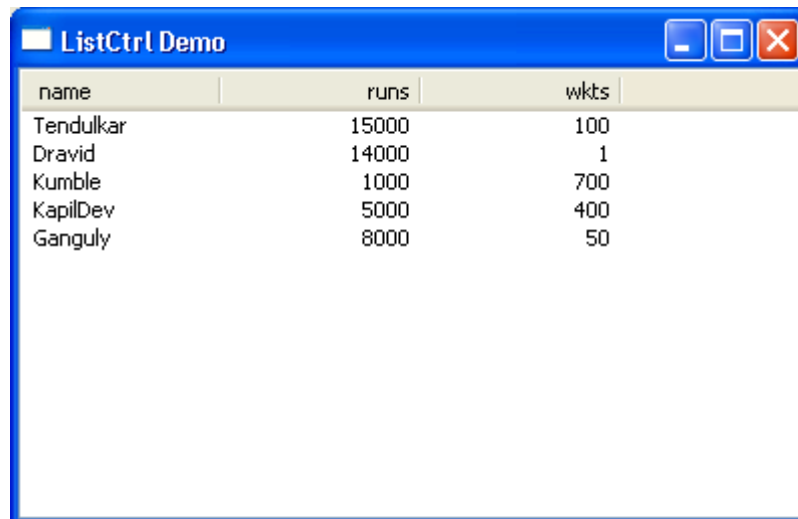
        for i in players:
            index = self.list.InsertStringItem(sys.maxint, i[0])
            self.list.SetStringItem(index, 1, i[1])
            self.list.SetStringItem(index, 2, i[2])
        box.Add(self.list,1,wx.EXPAND)
        panel.SetSizer(box)
        panel.Fit()
        self.Centre()

        self.Show(True)

ex = wx.App()
Mywin(None,'ListCtrl Demo')
ex.MainLoop()
```

The above code produces the following output. Players' data is displayed in report view:

111



A screenshot of a wxPython window titled "ListCtrl Demo". The window contains a list control displaying a table with three columns: "name", "runs", and "wkts". The data is as follows:

name	runs	wkts
Tendulkar	15000	100
Dravid	14000	1
Kumble	1000	700
KapilDev	5000	400
Ganguly	8000	50

## 33. Drag and Drop

Provision of **drag and drop** is very intuitive for the user. It is found in many desktop applications where the user can copy or move objects from one window to another just by dragging it with the mouse and dropping on another window.

Drag and drop operation involves the following steps:

- Declare a drop target
- Create data object
- Create wx.DropSource
- Execute drag operation
- Cancel or accept drop

In wxPython, there are two predefined drop targets:

- wx.TextDropTarget
- wx.FileDropTarget

Many wxPython widgets support drag and drop activity. Source control must have dragging enabled, whereas target control must be in a position to accept (or reject) drag.

Source Data that the user is dragging is placed on the the target object. OnDropText() of target object consumes the data. If so desired, data from the source object can be deleted.

### Example

In the following example, two ListCtrl objects are placed horizontally in a Box Sizer. List on the left is populated with a languages[] data. It is designated as the source of drag. One on the right is the target.

```
languages = ['C', 'C++', 'Java', 'Python', 'Perl', 'JavaScript', 'PHP',
'VB.NET', 'C#']

self.lst1 = wx.ListCtrl(panel, -1, style=wx.LC_LIST)
self.lst2 = wx.ListCtrl(panel, -1, style=wx.LC_LIST)

for lang in languages:
    self.lst1.InsertStringItem(0, lang)
```

The second list control is empty and is an argument for object of TextDropTarget class.

```
class MyTextDropTarget(wx.TextDropTarget):
    def __init__(self, object):
        wx.TextDropTarget.__init__(self)
        self.object = object
```

```
def OnDropText(self, x, y, data):
    self.object.InsertStringItem(0, data)
```

OnDropText() method adds source data in the target list control.

Drag operation is initialized by the event binder.

```
wx.EVT_LIST_BEGIN_DRAG(self, self.lst1.GetId(), self.OnDragInit)
```

OnDragInit() function puts drag data on the target and deletes from the source.

```
def OnDragInit(self, event):
    text = self.lst1.GetItemText(event.GetIndex())
    tobj = wx.PyTextDataObject(text)
    src = wx.DropSource(self.lst1)
    src.SetData(tobj)
    src.DoDragDrop(True)
    self.lst1.DeleteItem(event.GetIndex())
```

The complete code is as follows:

```
import wx

class MyTarget(wx.TextDropTarget):
    def __init__(self, object):
        wx.TextDropTarget.__init__(self)
        self.object = object

    def OnDropText(self, x, y, data):
        self.object.InsertStringItem(0, data)

class Mywin(wx.Frame):

    def __init__(self, parent, title):
        super(Mywin, self).__init__(parent, title=title, size=(-1,300))

        panel=wx.Panel(self)
        box=wx.BoxSizer(wx.HORIZONTAL)

        languages = ['C', 'C++', 'Java', 'Python', 'Perl', 'JavaScript', 'PHP',
'VB.NET', 'C#']
```

```

self.lst1 = wx.ListCtrl(panel, -1, style=wx.LC_LIST)
self.lst2 = wx.ListCtrl(panel, -1, style=wx.LC_LIST)
for lang in languages:
    self.lst1.InsertStringItem(0,lang)

dt = MyTarget(self.lst2)
self.lst2.SetDropTarget(dt)
wx.EVT_LIST_BEGIN_DRAG(self, self.lst1.GetId(), self.OnDragInit)
box.Add(self.lst1,0,wx.EXPAND)
box.Add(self.lst2, 1, wx.EXPAND)
panel.SetSizer(box)
panel.Fit()
self.Centre()
self.Show(True)

def OnDragInit(self, event):
    text = self.lst1.GetItemText(event.GetIndex())
    tobj = wx.PyTextDataObject(text)
    src = wx.DropSource(self.lst1)
    src.SetData(tobj)
    src.DoDragDrop(True)
    self.lst1.DeleteItem(event.GetIndex())

ex = wx.App()
Mywin(None, 'Drag&Drop Demo')
ex.MainLoop()

```

The above code produces the following output:

