Ein einfaches wxFrame-Beispiel

Das eigentliche "Programm", also die Klassendefition für das Fenster ist bis auf kleine Änderungen aus den wxPython-Demos übernommen worden. Ergänzt wird es allein durch die notwendige[1] Application, damit eine unter Python selbst lauffähige Anwendung entsteht.
Auf der zweiten Seite folgt dasselbe Programm noch einmal mit Kommentaren dazu. Die eingefügten Kommentare sollen das Verständnis erleichtern.

```
import  wx

#---------------------------------------------------------------------------
class MeinFenster(wx.Frame):
    def __init__(
            self, parent, ID, title, pos=wx.DefaultPosition,
            size=wx.DefaultSize, style=wx.DEFAULT_FRAME_STYLE
            ):

        wx.Frame.__init__(self, parent, ID, title, pos, size, style)
        panel = wx.Panel(self, -1)

        button = wx.Button(panel, 1003, "Beenden")
        button.SetPosition((15, 15))
        self.Bind(wx.EVT_BUTTON, self.OnCloseMe, button)
        self.Bind(wx.EVT_CLOSE, self.OnCloseWindow)


    def OnCloseMe(self, event):
        self.Close(True)

    def OnCloseWindow(self, event):
        self.Destroy()

# ===== Application-Klasse ====================================================
class MeineAnwendung(wx.App):
    def OnInit(self):
        frame = MeinFenster(None, -1, "Dies ist mein Fenster", size=(350,
200),style = wx.DEFAULT_FRAME_STYLE)
        self.SetTopWindow(frame)
        frame.Show(True)
        return True

# ===== Application-Objekt erzeugen ===========================================
app = MeineAnwendung(redirect=False)
app.MainLoop()
```

1 Siehe dazu auch den teilweise übersetzten Text aus dem wiki zu wxPython am Schluss dieses Textes.

Kommentierte Version

Die eingefügten Kommentare sollen das Verständnis erleichtern.

```
import  wx

#-------------------------------------------------------------------------
```

Ein Frame liefert ein top-level-Window, wir erben daher von wx.Frame

```
class MeinFenster(wx.Frame):
```

Der Konstruktor bekommt die für die Darstellung wichtigen oder gewollten Parameter übergeben. Einige Werte sind mit Standardwerten vordefiniert.

```
  def __init__(
          self, parent, ID, title, pos=wx.DefaultPosition,
          size=wx.DefaultSize, style=wx.DEFAULT_FRAME_STYLE
          ):
```

Zuerst wird der Konstruktor der vererbenden Klasse konkret aufgerufen.

```
      wx.Frame.__init__(self, parent, ID, title, pos, size, style)
```

Auf diesen (self) Frame wird ein panel-Objekt eingefügt. -1 steht für die Objekt-ID, die Python so beliebig wählen kann.

```
      panel = wx.Panel(self, -1)
```

Auf diesem panel können nun Objekt platziert werden, also z.B. ein Button. Seine ID wird definiert, so dass ggf. später abgefragt werden könnte, ob er ein Ereignis ausgelöst hat.

```
      button = wx.Button(panel, 1003, "Beenden")
      button.SetPosition((15, 15))
```

Button und Ereignismethode (folgt unten; darf nicht Teil des Konstruktors sein) werden verknüpft.

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

Auch der Frame selbst kennt – da er von wx.Window erbt, einige Ereignisse, die hier mit den Methoden verknüpft werden.

```
      self.Bind(wx.EVT_CLOSE, self.OnCloseWindow)
```

Ereignismethodefür den Button:

```
  def OnCloseMe(self, event):
      self.Close(True)
```

Ereignismethodefür den Frame:

```
  def OnCloseWindow(self, event):
      self.Destroy()

# ===== Application-Klasse ============================================
```

Die Application-Klasse definiert das Fenster, ...

```
class MeineAnwendung(wx.App):
    def OnInit(self):
```

… erzeugt ein Objekt der oben definierten Fensterklasse ...

```
      frame = MeinFenster(None, -1, "Mein Fenster", size=(350, 200),style =
wx.DEFAULT_FRAME_STYLE)
```

… definert es als top-level-window und ...

```
      self.SetTopWindow(frame)
```

… zeigt es an.

```
      frame.Show(True)
      return True

# ===== Application-Objekt erzeugen ========================================
app = MeineAnwendung(redirect=False)
```

Parameterwert True, wenn Ausgaben nicht in der Shell, sondern in einem eigenen Fenster angezeigt werden sollen

```
app.MainLoop()
```

Hilfetext aus dem wxPython-wiki

Teilweise Übersetzung und Code ...
… von → **http://wiki.wxpython.org/Using%20wxPython%20Demo%20Code**
Quelle wxPython Demos: **http://wxpython.org/download.php#demo**

Titel dort: *How to Use wxPython Demo Code*

Die Erläuterung beschreibt das Erstellen einer selbstständigen unter Python lauffähigen Anwendung aus dem in den Demos angegebenen Code.
Zunächst muss im verwendeten Python-Editor (z.B. IDLE) eine Datei für die Anwendung erstellt werden und der folgende Code eingefügt werden:

```
import wx

class MyApp(wx.App):
    def __init__(self, redirect=False, filename=None):
        wx.App.__init__(self, redirect, filename)
        self.frame = wx.Frame(None, wx.ID_ANY, title='My Title')

        self.panel = wx.Panel(self.frame, wx.ID_ANY)

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

Anschließend wählt man in den wxPython Demos die zu verwendende Komponente [auf der Hilfeseite wird die About-Box verwendet] und kopiert deren inneren Code in die eigene Datei.

Erweiterung um eine Panel-Klasse

Viele Demo-Beispiele[1] arbeiten mit einer gesonderten Panel-Klasse, also nicht allein mit einem internen Panel-Objekt in der Frame-Klasse. Wir verändern unsere Klasse, so dass sie eine Panel-Klasse verwendet. Dazu Kopieren wir den Code aus dem Button – Demo-Beispiel in unser Programm[2].
wir ersetzen

```
        panel = wx.Panel(self, -1)
```

durch

```
        panel = TestPanel(self)
```

Das TestPanel – Objekt muss also sein Elternobjekt – das Frame-Objekt – übergeben bekommen. Die im Button – Demo-Beispiel vorhandenen Buttons und deren Ereignisbehandlung wird durch die Zeilen aus der Frame-Klasse ersetzt. Dabei muss bei der Definition des Buttons das Objekt panel durch self ersetzt werden.
Nun können wir das Projekt starten, allerdings mit einem frustrierenden Ergebnis: Der Beenden-Button führt zu keinem Beenden!

Ein Panel-Objekt kann man nicht schließen

Was geschlossen werden muss, ist das Frame-Objekt. Das ist aber kein Problem, da beim Erzeugen des Panel-Objektes das Elternobjekt übergeben wird. Wir müssen es nur im Konstruktor einem Attribut zuweisen, dann können wir bei der Ereignisbehandlung darauf zugreifen. Der gesamte Programmcode lautet dann:

```
import wx
#---------------------------------------------------------------------------
class TestPanel(wx.Panel):
    def __init__(self, parent):
        wx.Panel.__init__(self, parent, -1,
                          style=wx.NO_FULL_REPAINT_ON_RESIZE)

        self.parent = parent
        button = wx.Button(self, 1003, "Beenden")
        button.SetPosition((15, 15))

        self.Bind(wx.EVT_BUTTON, self.OnCloseMe, button)

    def OnCloseMe(self, event):
        self.parent.Close(True)

#---------------------------------------------------------------------------
class MeinFenster(wx.Frame):
    def __init__(
        self, parent, ID, title, pos=wx.DefaultPosition,
        size=wx.DefaultSize, style=wx.DEFAULT_FRAME_STYLE
        ):
        wx.Frame.__init__(self, parent, ID, title, pos, size, style)
        # Änderung:
        panel = TestPanel(self)

        self.Bind(wx.EVT_CLOSE, self.OnCloseWindow)

    def OnCloseWindow(self, event):
        self.Destroy()
```

1 Beispiel: Core Windows/Controls → Button - Beispiel
2 Alles, was mit dem log zu tun hat, wird nur von der Demo benötigt und sollte entfernt werden.

```
# ===== Application-Klasse ============================================
class MeineAnwendung(wx.App):
    def OnInit(self):
        frame = MeinFenster(None, -1, "Mein Fenster", size=(350, 200),style =
wx.DEFAULT_FRAME_STYLE)
        self.SetTopWindow(frame)
        frame.Show(True)
        return True
# ===== Application-Objekt erzeugen ========================================
app = MeineAnwendung(redirect=False)
app.MainLoop()
```

Alles arbeitet nun wie vorher. Ob dieser Umbau lohnend ist, bleibt jedoch fraglich.