

Erläuterung der Abschnitte der Klasse für die GUI

In diesem Text werden die einzelnen Abschnitte ausführlich erläutert.

Kopf der Klassendefinition

```
import wx
```

- Das Paket wx muss importiert werden, damit die hier verwendeten Klassen benutzt werden können.

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

- Wie der Name MyFrame schon sagt, schreiben wir unsere eigene Frame-Klasse, die aber alles können soll, was Standard-Frame-Klassen können. Was das alles ist, wollen wir nicht vollständig behandeln, auf einige Eigenschaften und Verhalten kommen wir aber. Damit MyFrame das kann, müssen wir die Klasse von wx.Frame „ableiten“ [von ihr „erben“ lassen]. Das kennzeichnen wir mit dem Klassennamen in Klammern.

```
"""
```

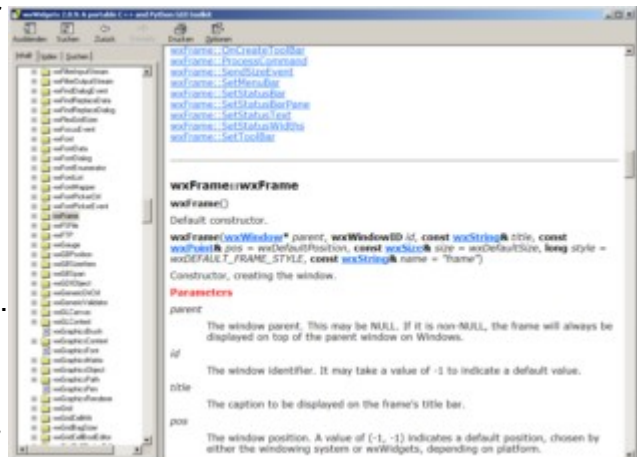
```
Die Klasse MyFrame erzeugt einen Frame fuer die Klasse Bruch.
```

```
"""
```

- Darunter steht ein mehrzeiliger Kommentar.

```
def __init__(self, parent, title):
    wx.Frame.__init__(self, parent, -1, title,
                      pos=(100, 100), size=(350, 350))
```

- Der Konstruktor braucht als Parameter natürlich ersteinmal self, dann das Fenster, zu dem der erzeugte Frame gehören soll und als dritten Parameter die Titelzeile des Fensters.
- Er ruft den Konstruktor der Klasse auf, von der er abgeleitet wurde. In der Hilfe von wxPython kann man nachlesen, welche Parameter er kennt. Die Kopfzeile sieht etwas unverständlich aus, die Erläuterungen kann man aber meistens verstehen. Allerdings muss man sich natürlich auf englischen Text einlassen:



Beispiel einer Übersetzung

parent: The window parent. This may be NULL. If it is non-NULL, the frame will always be displayed on top of the parent window on Windows.

- übersetzt „Eltern“, das Fenster, zu dem dieses gehört (s.o.); steht dort NULL, gibt es keines.

id: The window identifier. It may take a value of -1 to indicate a default value.

- Die Fenster werden durch Zahlen identifiziert. Wenn es egal ist, welchen Wert diese Zahlen haben, darf man -1 verwenden, dann wird eine Zahl zufällig erzeugt.

title: The caption to be displayed on the frame's title bar.

- s.o.

pos: The window position. A value of (-1, -1) indicates a default position, chosen by either the windowing system or wxWidgets, depending on platform.

- die Position, die das Fenster beim Erscheinen haben soll; auch hier kann man das

dem System überlassen.

size: The window size. A value of (-1, -1) indicates a default size, chosen by either the windowing system or wxWidgets, depending on platform.

- die Größe des Fensters in pixeln

style: The window style. See wxFrame.

- Stil; was auch immer → Dokumentation

name: The name of the window. This parameter is used to associate a name with the item, allowing the application user to set Motif resource values for individual windows.

- Name des Fensters; wofür auch immer

Achtung: Wenn man anfängt zu programmieren, muss man nicht alles sofort verstehen. Man konzentriert sich zunächst auf die wesentlichen Aspekte.

Menü

Darum folgen die nächsten Zeilen des Programmcodes, die für das Menü des Fensters zuständig sind, ohne weitere Erläuterungen.

```
# Erzeugt die Komponente Menuezeile [menubar]
menuBar = wx.MenuBar()
# [noch zu klaeren: ist menuBar nur aus Faulheit als globale
Variable definiert
# und nicht mit self. ???]

# und erzeuge fuer die Menuezeile zwei Menuetitel.
menu1 = wx.Menu()
menu2 =wx.Menu()

# Hier wird ein Menuepunkt in den ersten oben erzeugeten
Menuetitel
# eingefuegt. [add an item to the menu]
# Mit \tKeyName wird ein Auswahlbuchstabe definiert, zudem auch
automatisch
# die Ereignisbehandlung generiert wird.
# Der Text im dritten Parameter wird in der Statuszeile
angezeigt.
menu1.Append(wx.ID_EXIT, "B&eenden\tAlt-E", "Schliessen der
Anwendung")

# binde den Menuepunkt an seine Ereignisbehandlung [event
handler]
self.Bind(wx.EVT_MENU, self.OnTimeToClose, id=wx.ID_EXIT)

# Fuege das Menue der Menuezeile hinzu.
menuBar.Append(menu1, "&Datei")
menuBar.Append(menu2, "&Hilfe")
self.SetMenuBar(menuBar)
# Anlegen der Statuszeile [z.B. fuer die oben definierten
Meldungen]
self.CreateStatusBar()
```

panel

Die eigentliche Fensteroberfläche ist eine panel. Auf ihm werden die ganzen Fenster-elemente untergebracht.

```
# Erzeuge die Oberflaeche [abgeleitet von wxPanel]
```

```
# fuer die hinzuzufuegenden Elemente.  
panel = wx.Panel(self)
```

Also zum Beispiel ...

Beschriftungen

```
text = wx.StaticText(panel, -1, "Bruchrechnungsprogramm")  
text.SetFont(wx.Font(10, wx.SWISS, wx.NORMAL, wx.BOLD))  
text.SetSize(text.GetBestSize())  
  
text2 = wx.StaticText(panel, -1, "Ergebnis :")  
text2.SetFont(wx.Font(10, wx.SWISS, wx.NORMAL, wx.BOLD))  
text2.SetSize(text.GetBestSize())
```

Textfelder zur Eingabe und Ausgabe

```
self.textfeld1 = wx.TextCtrl(panel, -1, "11 / 20", size=(125,-1))  
self.textfeld2 = wx.TextCtrl(panel, -1, "7 / 8", size=(125, -1))  
self.textfeld3 = wx.TextCtrl(panel, -1, " ", size=(125, -1))
```

Buttons:

Ein Button auf der Oberflaeche zum Beenden der Anwendung ist immer hilfreich.

```
endeBtn = wx.Button(panel, -1, "Beenden")  
endeBtn.SetToolTipString("Button beendet die Anwendung")
```

Außerdem natürlich je ein Button für jede Berechnungsart, also für die ...

Addition

```
plusBtn = wx.Button(panel, -1, " + ")  
plusBtn.SetToolTipString("addiert die beiden Brueche")
```

Subtraktion

```
minusBtn = wx.Button(panel, -1, " - ")  
minusBtn.SetToolTipString("subtrahiert die beiden Brueche")
```

Multiplikation

```
malBtn = wx.Button(panel, -1, " * ")  
malBtn.SetToolTipString("multipliziert die beiden Brueche")
```

Division

```
durchBtn = wx.Button(panel, -1, " : ")  
durchBtn.SetToolTipString("dividiert die beiden Brueche")
```

Ereignisbehandlung

Alle diese Buttons müssen mit der Ereignisbehandlung [event-handler] verbunden werden.

```
self.Bind(wx.EVT_BUTTON, self.OnTimeToClose, endeBtn)  
self.Bind(wx.EVT_BUTTON, self.OnPlusButton, plusBtn)  
self.Bind(wx.EVT_BUTTON, self.OnMinusButton, minusBtn)  
self.Bind(wx.EVT_BUTTON, self.OnMalBtn, malBtn)  
self.Bind(wx.EVT_BUTTON, self.OnDurchBtn, durchBtn)
```

Layout

Hier wird ein Layoutmanager für ein tabellarisches Layout [GridSizer] verwendet. Das ist sinnvoll, da sehr viele Elemente positioniert werden sollen und alle untereinander sofort eine sehr unübersichtliche Darstellung ergeben.

Die Anzahl der Spalten wird auf 2 festgesetzt, die 0 sorgt dafür, dass die Zahl der Zeilen automatisch festgelegt wird.

```
sizer = wx.GridSizer(0, 2)
# erste Zeile:
sizer.Add(text, 0, wx.ALL, 10)
sizer.Add(endeBtn, 0, wx.ALIGN_CENTER, 10)
# zweite Zeile:
sizer.Add(self.textfeld1, 0, wx.ALL, 10)
sizer.Add(self.textfeld2, 0, wx.ALL, 10)
# dritte Zeile:
sizer.Add(plusBtn, 0, wx.ALL, 10)
sizer.Add(minusBtn, 0, wx.ALL, 10)
# vierte Zeile:
sizer.Add(malBtn, 0, wx.ALL, 10)
sizer.Add(durchBtn, 0, wx.ALL, 10)
# fuenfte Zeile:
sizer.Add(text2, 0, wx.ALL, 10)
sizer.Add(self.textfeld3, 0, wx.ALL, 10)
```

Der sizer muss nun noch dem panel hinzugefügt werden.

```
panel.SetSizer(sizer)
panel.Layout()
```

Ereignismethoden

Jedes Anklicken eines Buttons hat jeweils eine andere Folge. Diese Folge müssen wir programmieren und dazu dienen die Ereignismethoden. Die Methoden sollten selbsterklärend sein.

```
def OnTimeToClose(self, evt):
    """Ereignisbehandlung fuer den Ende-Button."""
    self.Close()

def OnPlusButton(self, evt):
    """Ereignisbehandlung fuer den plus-Button."""
    b1, b2=self.holeBrueche()
    b=b1.addiere(b2)
    self.textfeld3.SetValue(b.gibString())

def OnMinusButton(self, evt):
    """Ereignisbehandlung fuer den minus-Button."""
    b1, b2=self.holeBrueche()
    b=b1.subtrahiere(b2)
    self.textfeld3.SetValue(b.gibString())

def OnMalBtn(self, evt):
    """Ereignisbehandlung fuer den minus-Button."""
    b1, b2=self.holeBrueche()
    b=b1.multipliziere(b2)
    self.textfeld3.SetValue(b.gibString())

def OnDurchBtn(self, evt):
    """Ereignisbehandlung fuer den minus-Button."""
    b1, b2=self.holeBrueche()
    # Division durch 0 verhindern:
    if b2.zaehler!=0:
        b=b1.dividiere(b2)
        self.textfeld3.SetValue(b.gibString())
    else:
        self.textfeld3.SetValue("keine Division durch 0!")
```

Hilfsmethoden

Alle Methoden für die Berechnung müssen zunächst die in die Textfelder eingegebenen Zahlen holen. Diese sind nicht nur von einem anderen Typ, nämlich keine Zahlen sondern strings, sondern auch noch in einer Bruchschreibweise versteckt. Z.B.: "11 / 20" (siehe die vordefinierten Werte aus den Textfeldern. Der eine string muss also aufgeteilt [split] werden und von den Teilen brauchen wir nur den ersten und dritten. Python fängt aber beim Zählen bei 0 an, so dass wir also den nullten und den zweiten brauchen.

```
# Hilfsmethode zum Holen von Zaehler und Nenner
def holeZahlen(self, text):
    text_teile=text.split()
    return int(text_teile[0]), int(text_teile[2])
```

Diese beiden Zahlen braucht nun eine Methode, die daraus die beiden Brüche erzeugt.

```
# Hilfsmethode zum Holen der Brueche
def holeBrueche(self):
    z, n=self.holeZahlen(self.textfeld1.GetValue())
    b1=Bruch(z, n)
    z, n=self.holeZahlen(self.textfeld2.GetValue())
    b2=Bruch(z, n)
    return b1, b2
```

Anwendung [Application]

Unsere Frameklasse wird nun verwendet von einem Application-Objekt, also einer Anwendung. Dieses Objekt muss erzeugt und gestartet werden.

```
class MyApp(wx.App):
    def OnInit(self):
        frame = MyFrame(None, "Bruchrechnungs - Anwendung")
        self.SetTopWindow(frame)
        frame.Show(True)
        return True
```

```
app = MyApp(redirect=True)
app.MainLoop()
```

Auf geht es!

Das war's, reduziert auf die wesentlichen Probleme. Ihr solltet mit dieser Hilfe in der Lage sein, Schritt für Schritt eigene Anwendungen schreiben zu können.