Projekt Palindrom mit GUI

Ziele

- einfache Anwendung von String-Behandlung
- Trennung von Modell und Gui in Module
- Übung für das Arbeiten mit der Demo und der Hilfe zu wxPython

Die Modellkomponente behandelt das Fachproblem

Eine Zeichenkette, die symmetrisch ist, heißt Palindrom. Es geht hier nicht darum, sich über sinnvolle oder weniger sinnvolle Palindrome Gedanken zu machen oder ob diese überhaupt ein sinnvolles Problem darstellen.
An diesem Beispiel können auf einfache Weise der Datentyp string und elementare Algorithmen kennengelernt werden.

Rekursion bei funktionaler Sprache

Bei einer funktionalen Sprache ist ein Ansatz mit einer rekursiven Lösung absolut naheliegend. Da zudem bei Palindromen nicht unbedingt mit sehr großer Tiefe der Rekursion zu rechnen ist, kann man auch bei Python von einer funktionierenden Lösung ausgehen.

Ansatz mit Rekursion

Der Ansatz mit Rekursion muss einmal den auf jeder Stufe zu entscheidenden Fall, den Abbau und den Abbruchfall beschreiben.

- Fall: Vergleiche den ersten mit dem letzten Buchstaben.
- Schritt: Bearbeite den String ohne den ersten und den letzten Buchstaben weiter.
- Abbruch (Elementarfall): Der String besteht aus höchstens einem Zeichen.

Eine rekursive Lösung

```
1 def istPalindrom(text):
2     if len(text)<2: return True
3     if text[0].upper() != text[len(text)-1].upper(): return False
4     return istPalindrom(text[1:len(text)-1])
```

Hinweise

- len(...) berechnet die Länge von sequenziellen Typen (Sequence Types), zu denen der Datentyp string gehört.
- Wegen der null-basierenden Indexzählung muss beim Zugriff auf das letzte Zeichen -1 gerechnet werden.
- Wegen der von Computer unterschiedenen Darstellung von Klein- und Großbuchstaben muss der Typ umgewandelt werden.
- Die Funktion von slices bei sequenziellen Typen muss den Schülerinnen und Schülern erläutert werden.
- Die Version ist nicht logisch "sauber", da ausgenutzt wird, dass return die Bearbeitung der Funktion beendet.
  "Besser" also: in Zeile 3 **`elif`** statt **`if`** und in Zeile 4 **`else: return...`** .

Eine iterative Lösung

Iterativ wird man Indexbasiert zugreifen, da man die Zugriffsposition berechnen will. Das

Programm ist ein Dreizeiler, sicher nicht komplizierter als die rekursive Variante.

```
1 def istPalindrom(text):
2     for i in range(len(text)/2):
3         if text[i].upper() != text[len(text)-i-1].upper(): return False
4     return True
```

Die Entwicklung der GUI mit wxPython

Zunächst benötigen wir eine Application und ein wxFrame-Objekt. Dessen Code holen wir uns aus den wxPython Demos. Im Abschnitt **Frames and Dialogs** finden wir das Beispiel **Frame**. Der Code wird zunächst nur geringfügig an unser Problem angepasst.

```
 1 import  wx
 2 #---------------------------------------------------------------------------
 3 class Fenster(wx.Frame):
 4     def __init__(
 5             self, parent, ID, title, pos=wx.DefaultPosition,
 6             size=wx.DefaultSize, style=wx.DEFAULT_FRAME_STYLE
 7             ):
 8
 9         wx.Frame.__init__(self, parent, ID, title, pos, size, style)
10         panel = wx.Panel(self, -1)
11
12         button = wx.Button(panel, 1003, "Beenden")
13         button.SetPosition((15, 15))
14         self.Bind(wx.EVT_BUTTON, self.OnCloseMe, button)
15         self.Bind(wx.EVT_CLOSE, self.OnCloseWindow)
16
17
18     def OnCloseMe(self, event):
19         self.Close(True)
20
21     def OnCloseWindow(self, event):
22         self.Destroy()
23 #---------------------------------------------------------------------------
24 # die Anwendungsklasse
25 class Anwendung(wx.App):
26     def OnInit(self):
27         frame = Fenster(None, -1, "Palindrome")
28         self.SetTopWindow(frame)
29         frame.Show(True)
30         return True
31
32 app = Anwendung(redirect=False)
33 app.MainLoop()
```

Kommentare zu den Programmzeilen der Fensterklasse

ZN 3: Die von uns [übernommene und leicht modifizierte] Klasse erbt von wxFrame.
ZN 4: Hier beginnt der Konstruktor, dessen Parameter (nach self) das nicht vorhandene Elternobjekt, die vom System frei wählbare Komponenten-ID und der in der Window-Zeile anzuzeigende Name sind. Position, Größe und Stil werden mit Standardwerten vordefiniert.
ZN 9: der Konstruktor ruft zunächst explizit den Konstruktor von wxFrame auf und …
ZN 10: … definiert das Panel-Objekt, auf dem die Elemente der GUI positioniert werden sollen.
ZN 12: Es wird ein Button-Objekt erzeugt, dem Panel hinzugefügt und mit Beenden beschriftet. Der Parameter dazwischen ist der Parameter für die ID, der hier explizit definiert wird, damit bei der Ereignisbehandlung möglicherweise das auslösende Objekt identifiziert werden kann.
ZN 13: Die Position des Buttons auf der Oberfläche wird bestimmt.
ZN 14: Der Button mit dem Namen button (es kann ja noch weitere geben) wird an die

Methode **`self.OnBeenden`** zur Behandlung des Ereignisses **`wx.EVT_BUTTON`** gebunden.
ZN 15: Entsprechend für das Fenster.
ZN 18-22: Hier sind die entsprechenden Ereignisbehandlungsmethoden definiert.
Beachten Sie, dass diese nicht mehr Teil des Konstruktors sein dürfen.
Bevor weitere Objekte der Oberfläche hinzugefügt werden, sollten die Aufgaben beschrieben werden, welche die GUI erfüllen soll.

Aufgaben der GUI

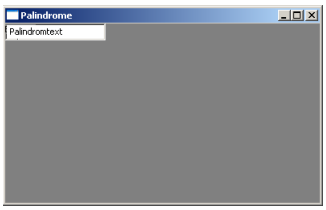
- Die GUI soll dem Benutzer die Möglichkeit geben,
  - den Text einzugeben,
  - zu ändern,
  - die Prüfung auszulösen und
  - die Anwendung zu beenden.
- Die GUI soll dem Benutzer anzeigen,
  - welchen Zustand der eingegebene Text hat,
  - welches Ergebnis die Palindromprüfung hat,
  - ggf. [als erweiterung] Abweichungen vom Palindrom im Text kennzeichnen

Aus diesen Aufgaben ergibt sich nun, welche Komponenten benötigt werden.

Ein Eingabefeld hinzufügen

Für die Texteingabe benötigen wir eine Eingabefeld. In den wxPython Demos finden wir im Abschnitt **Core Windows/Controls** dazu das Beispiel **TextCtrl**. Den Code zum oberen Beispiel mit der Beschriftung **wx.TextCtrl** übernehmen wir, entfernen aber alles, was mit Ereignisbehandlung zu tun hat. da wir nicht auf Ereignisse des Eingabefeldes reagieren wollen. Daher verbleiben nur die ersten beiden Zeilen.

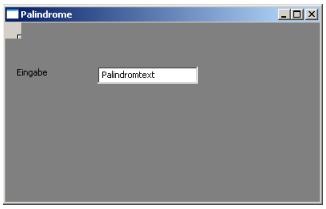
```
        l1 = wx.StaticText(self, -1, "Eingabe")
        t1 = wx.TextCtrl(self, -1, "Palindromtext", size=(125, -1))
```



Wir fügen Sie hinter der Zeilennummer ZN 16 ein. Ein Test führt allerdings zu einem frustrierenden Ergebnis. Woran liegt das?

Sizer und panel

Dafür gibt es zwei Gründe.
Dieses zweite Beispiel aus den wxPython Demos arbeitet zur Positionierung der Elemente mit einem Sizer. Damit kann man Fensterlemente durch das System selbstständig positionieren lassen. Wr haben aber im o.a. Beispiel den Button absolut positioniert. Das passt nicht zusammen. Die Obejekte werden so einfach in die obere Ecke gesetzt.



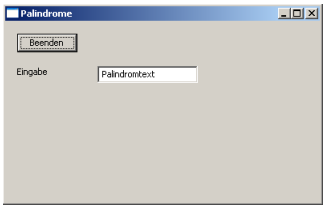
Entscheiden wir uns für eine absolute Positionierung, müssen wir diese einfügen. Die beiden Zeilen ...

```
        l1.SetPosition((15, 55))
        t1.SetPosition((115, 55))
```

… lösen das Problem aber immer noch nicht ganz. Es bleibt immer noch ein grauer Hintergrund.
Dies liegt daran, dass im zweiten Beispiel aus den wxPython Demos der Code Teil eines wxPanel-Objektes ist, wir aber einen wxFrame verwenden. Wir müssen deswegen self durch panel ersetzen. Die vier Zeilen lauten nun ...

```
        l1 = wx.StaticText(panel, -1, "Eingabe")
        t1 = wx.TextCtrl(panel, -1, "Palindromtext", size=(125, -1))
        l1.SetPosition((15, 55))
        t1.SetPosition((115, 55))
```

… und das Ergebnis ist wie gewünscht verändert.



Text ändern

Den Text können wir problemlos ändern, wie ein einfacher Test zeigt. Allerdings passiert dann nichts, da wir die Ereignisbehandlung dafür nicht mit übernommen haben.
Um die Prüfung auszulösen, benutzen wir einen weiteren Button. Das ist natürlich nicht zwingend, genau so sinnvoll wäre es, dem Programm eine Menüleiste hinzuzufügen. Wir bleiben aber beim Button und können den Code vom Beenden-Button entsprechend modifiziert übernehmen. Die entsprechende Ereignisbehandlungsmethode lassen wir zunächst leer. Um keine Verwechslungen zu haben, benennen wir den ersten Button sinngemäß um. Die entsprechenden Codezeilen lauten nun:

```
        beendenButton = wx.Button(panel, 1003, "Beenden")
        beendenButton.SetPosition((15, 15))
        self.Bind(wx.EVT_BUTTON, self.OnBeenden, beendenButton)
        self.Bind(wx.EVT_CLOSE, self.OnCloseWindow)

        l1 = wx.StaticText(panel, -1, "Eingabe")
        t1 = wx.TextCtrl(panel, -1, "Palindromtext", size=(125, -1))
        l1.SetPosition((15, 55))
        t1.SetPosition((115, 55))

        pruefenButton = wx.Button(panel, 1004, "Pruefen")
        pruefenButton.SetPosition((115, 15))
        self.Bind(wx.EVT_BUTTON, self.OnPruefen, pruefenButton)

    def OnPruefen(self, event):
        print "Der Text wird geprueft."
```

Schnittstelle definieren

Wir sind hier an einer entscheidenden Stelle der OO Modellierung. Die Behandlung des Ereignisses, dass der Benutzer den Pruefen-Button angeklickt hat, gehört nicht in die Gui-Klasse. Dies ist eine Methode, die uns die entsprechende Modell-Klasse zur Verfügung stellen muss, also die Klasse, die Palindrome behandelt. Dies ist aber bisher nur eine einzelne Methode. Wir müssen sie einer neu zu erstellenden Klasse zuordnen und zu der zunächst einmal die Schnittstelle definieren.
Objekte der Klasse

- müssen den Text übergeben bekommen und ihn speichern,
- sollten dann auch eine get-Methode und
- müssen eine Methode zum Prüfen bereitstellen

Palindrom-Behandlungs-Klasse

Eine Lösung wäre

```
class PalindromPruefer:
    def __init__(self):
        self.__text=""

    def setzeText(self, neuerText):
        self.__text = neuerText

    def zeigeText(self):
        return self.__text
```

```
    def pruefe(self):
        # Wir verwenden die rekursive Variante
        def istPalindrom(text):
            if len(text)<2: return True
            if text[0].upper() != text[len(text)-1].upper(): return False
            return istPalindrom(text[1:len(text)-1])

        return istPalindrom(self.__text)
```

Änderungen in der GUI-Methode

Wichtig ist, der GUI-Klasse die Modell-Klasse "zur Kenntnis zu geben". Dazu bauen wir im Kopf die Importzeile ein.

```
from palindromKlasse import *
```

In der GUI-Klasse muss nur noch der Teil der Ereignisbehandlung eingebaut werden, der mit der Darstellung zu tun hat.
Die Methode soll nach dem Prüfen das Ergebnis anzeigen. Dazu bietet sich auch das Textfeld an, dessen Inhalt um eine entsprechende Meldung verlängert wird.
Dazu müssen wir aus der wxPython-Hilfe für das Element TextCtrl die Methode zum Setzen eines Inhalts herausfinden. Das ist mit **`SetValue(...)`** leicht zu finden, entsprechend gibt es auch **`GetValue()`**.
Bauen wir das entsprechend ein, …

```
    def OnPruefen(self, event):
        self.palindromPruefer.setzeText('anna')
        if self.palindromPruefer.pruefe():
            t1.SetValue(t1.GetValue()+" ist Palindrom")
        else:
            t1.SetValue(t1.GetValue()+" ist kein Palindrom")
```

… zeigt sich ein Fehler der ursprünglichen Konstruktion der GUI-Objekte: Sie sind nur lokale Objekte des Konstruktors und nicht Attribute des Fenster-Objektes. Das können wir mit vorgestelltem self ändern und erhalten eine funktionierende Lösung.
Natürlich könnte man am Layout noch einiges verbessern und sich Gedanken darüber machen, wie der Benutzer z.B. eine Mitteilung bekommen kann, an welcher Stelle der Text vom Palindrom abweicht. Das soll aber nicht Ziel dieses Textes sein.

Der vollständige Text dieser Version folgt auf der nächsten Seite.

```
# palindrom-gui.py

import  wx

from palindromKlasse import *

#---------------------------------------------------------------------------

class Fenster(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)

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

        l1 = wx.StaticText(panel, -1, "Eingabe")
        self.t1 = wx.TextCtrl(panel, -1, "Palindromtext", size=(125, -1))
        l1.SetPosition((15, 55))
        self.t1.SetPosition((115, 55))

        pruefenButton = wx.Button(panel, 1004, "Pruefen")
        pruefenButton.SetPosition((115, 15))
        self.Bind(wx.EVT_BUTTON, self.OnPruefen, pruefenButton)

        # Objekt von palindromKlasse erzeugen
        self.palindromPruefer = PalindromPruefer()


    def OnPruefen(self, event):
        self.palindromPruefer.setzeText(self.t1.GetValue())
        if self.palindromPruefer.pruefe():
            self.t1.SetValue(self.t1.GetValue()+" ist Palindrom")
        else:
            self.t1.SetValue(self.t1.GetValue()+" ist kein Palindrom")

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

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

#---------------------------------------------------------------------------
# die Anwendungsklasse
class Anwendung(wx.App):
    def OnInit(self):
        frame = Fenster(None, -1, "Palindrome")
        self.SetTopWindow(frame)
        frame.Show(True)
        return True

app = Anwendung(redirect=False)
app.MainLoop()
```