Organizing GUI Apps

Ch 9 in Perkovic...

Globals

Review temps.py: lots of global vars.... Not good

[temps.py]

Alternative (described in Sec 9.4): define classes that extend widget classes

Perkovic: motivation is “reuse”

But better organization has its own benefits (modularity most important)

Example: working on a big project, assign different parts of the interface to separate groups

ClickIt

Example from Perkovic demonstrates basic idea

clickit.py (Sec 9.2) — simple button demo; displays current time, uses globals

repackaged: ClickIt class — frame that contains button

can be imported and used by other apps

[clickit.py, Components.py, test_Clickit.py]

Details to note:

• the __init__ method of ClickIt calls its parent class __init__ function (important!)
• the Button inside a ClickIt frame has self as its parent
• the callback is a method of the ClickIt class, referred to as self.clicked from the __init__ method

Details to note in test_ClickIt.py (app that uses ClickIt class):

• we only need to import the name ClickIt
• the top level module doesn’t refer to Frame, Button, etc; no need to import
• it’s up to the ClickIt module to import the pieces it needs; we don’t need to worry about how the Frame is built
**Date**

Another example from Perkovic

[day.py, Components.py, test_Day.py]

**Notes:**

- since dateEnt is used elsewhere in the class the constructor saves it as an instance variable
- the button and label are never referred to by other code, so no need to save references to them (names go away, objects don't...)

**Background:** localtime, strftime and strptime

Call localtime to get a “time struct”

```python
>>> from time import localtime, strptime, strftime
>>> t = localtime()
```

```python
>>> t.tm_year
2014
```

```python
>>> t.tm_wday
4
```

Call to strftime format a time:

```python
>>> strftime("%Y %b %d", t)
'2014 Apr 25'
```

```python
>>> strftime("%A, %B %d, %Y", t)
'Friday, April 25, 2014'
```

Call strptime to parse a time string, creating a “time struct” object:

```python
>>> strptime("Jan 1 2000", "%b %d %Y")
time.struct_time(tm_year=2000, tm_mon=1, tm_mday=1, tm_hour=0, tm_min=0, tm_sec=0, tm_wday=5, tm_yday=1, tm_isdst=-1)
```

```python
>>> x = strptime("Jan 1 2000", "%b %d %Y")
```

```python
>>> strftime("%A, %B %d, %Y", x)
'Saturday, January 01, 2000'
```

See the online documentation at python.org for information about the formatting options
**Temps**

Components.py also has the definition of a new Frame that encapsulates all the GUI components from the temps program

Notes:
- what used to be global vars (oldf and oldc) are now instance vars
- as in the Day example, references to some widgets are saved as instance vars

**Using 2 Components**

See time_and_temp.py for an example of an app that uses two of our re-packaged components...

**Summary of Programs**

<table>
<thead>
<tr>
<th>Original, with globals</th>
<th>Class (all in Components.py)</th>
<th>App using class</th>
</tr>
</thead>
<tbody>
<tr>
<td>clickit.py</td>
<td>ClickIt</td>
<td>test_ClickIt.py</td>
</tr>
<tr>
<td>day.py</td>
<td>Day</td>
<td>test_Day.py</td>
</tr>
<tr>
<td>temps.py</td>
<td>Temp</td>
<td>test_Temps.py</td>
</tr>
</tbody>
</table>

App that uses two components: time_and_temp.py

**Tweaks**

Let’s do some interactive experiments to draw borders around the frames, offset them from one another

```python
>>> from tkinter import Tk
>>> app = Tk()
>>> from Components import Day
>>> d = Day(app)
```

See Sec 11 (p. 47) of the tkinter reference for Frame widget options: padx, pady, relief, borderwidth

Can we add space around the two frames so they don’t crowd the border of the main app?