Buttons and Other Widgets

Ch 9 in Perkovic...

Recap

Next two projects: using Python’s tkinter library to create graphical user interfaces (GUIs)

- Apr 28: hello world, payment calculator, labels with images (cards)
- May 5: Blackjack

[demo: programs/flipper.py]
[.demo: programs/blackjack.py]

Program Structure

Main elements in a tkinter app:

- create a “root” object, an instance of the Tk class
- create widget objects (labels, buttons, menus, …)
- widgets are organized in a hierarchy, with Tk object at the root of the tree
- use a layout manager to place widgets inside the parent widget
- start the event loop (call root.mainloop)

Hello Button

Let’s look at a second version of the “hello, world” program

This one has a button instead of a label

[demos/hello-button.py]

Callback Function

The important concept here: callback (a function called when the button is clicked)

Note how the function is specified: we pass the name of the function as the value of command when the button is created

Where have we seen something like this before?
Example Program

temps.py is similar to your payment calculator

Display two boxes, one labeled Fahrenheit and the other Celsius

Users enter a temperature into one box, click a button; the app converts the value to the other scale and displays the result in the other box

[demos/temps.py]

Things to notice about this app:
• there are 5 widgets inside root: two labels, two text entry boxes, and a button
• the widgets place themselves using the grid layout manager
• 3 x 2 grid (3 rows, 2 columns); note how bottom row spans both columns

<table>
<thead>
<tr>
<th>Label</th>
<th>Entry</th>
</tr>
</thead>
<tbody>
<tr>
<td>Label</td>
<td>Entry</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

• the calls to grid specify additional options (spacing, etc; there are lots more)
• there is a single callback function named convert, called from both buttons
• strategy (not foolproof, but it’s OK for this demo): record most recent value of each temp in global vars; when button clicked, see which one changed, use it as value to convert…
Exploring

Use an interactive session get explore how widgets work:

```
[sasquatch:GUI] ➤ python3
>>> from temps import *
```

Note the `if __name__ == '__main__'` statement at the end of the file....

Ask to see the value of one of the widgets:

```
>>> fahr
<tkinter.Entry object at 0x101981390>
```

```
>>> dir(fahr)
[way too much stuff]
```

Attributes that can be configured:

```
>>> fahr.keys()
['background', 'bd' ... ]
```

To get the current value of a configuration option: `cget`

```
>>> fahr.cget('width')
20
```

(unit is characters)

To change a value call `configure`:

```
>>> fahr.configure(width = 25)
>>> fahr.configure(bg = 'red')
```

In tkinter widgets are “dictionary-like” objects; instead of `cget` and `configure`:

```
>>> fahr['width']
20
```

```
>>> fahr['width'] = 15
>>> fahr['background'] = 'lightgray'
```

Test a button’s callback function:

```
>>> button.invoke()
```
**Working with Text**

To get the contents of a text entry widget:

```python
>>> cels.get()
```

Text is updated using editing commands (delete, insert, etc)

To change the text in a box:

- call `delete` to erase the current contents
- call `insert` to add new contents

Arguments to `delete` and `insert` are starting and ending locations
- to work on the entire box pass 0 and END (a constant defined in the library)

See `f2c` and `c2f` in `temps.py`
**Development Strategy**

Start small, grow into bigger program

[Aside: “How to Succeed”]

Example: loan payment program suggests putting output in a disabled entry box. How to disable?

Poke around, discover ‘state’ is configurable:

```python
>>> fahr['state']
'normal'
```

To change state:

```python
>>> fahr['state'] = 'disabled'
```

** Experiment in shell session, learn what works, learn what methods to call, what parameters to use
** Copy and paste into app
** Repeat with the next behavior

**IEP and IDLE**

Unfortunately it’s hard to test GUI code in an interactive session in IEP or IDLE

Both IDEs have their own event loops

But you can run an interactive session from a shell (aka command prompt) window

**Breadth-First vs Depth-First**

Breadth: lay out all widgets, to see how they look, then fill in behaviors (e.g. buttons start out doing nothing, or printing message, then add real callback)

Depth: start with one widget, figure out how to use it, how to fill it with text, how to initialize it, etc, then move on to next one.

** Both take advantage of incremental development style**