Practical Sheet: GUI Programming

This sheet is a set of exercises for introducing GUI programming in Python using Tkinter, assuming knowledge of basic Python programming.

All materials are at http://www.eecs.qmul.ac.uk/~william/CAS-London-2016.html You need to download the zip file of sample programs.

Contents

These notes are a shortened version of the materials available at http://teachinglondoncomputing.org/free-workshops/minicpd-gui-programming-in-python/ (scroll down from the resources). If you complete all the exercises here, there are more …

1 Exercise 1: A First GUI Program

The following program is available for download (called exercise1.py). Find the program, open it using IDLE and run it.

```
# Import the Tkinter package
#     Note in Python 3 it is all lowercase
from tkinter import *

# This method is called when the button is pressed
def clicked():
    print("Clicked")

# Create a main frame with
#      - a title
#      - size 200 by 200 pixels
app = Tk()
app.title("GUI Example 1")
app.geometry('200x200')

# Create the button with
#     - suitable text
#     - a command to call when the button is pressed
button1 = Button(app, text="Click Here", command=clicked)

# Make the button visible at the bottom of the frame
button1.pack(side='bottom')

# Start the application running
app.mainloop()
```

Exercise 1.1: Run the Program

Find the program (or type it in), open it using IDLE and run it. It should look like this:

Also experiment with 'usual' windows operation such as resizing the window, minimising it and closing it.

Exercise 1.2: Modify the Program

Although it has not been explained yet, see if you can figure out how to make the following modifications:

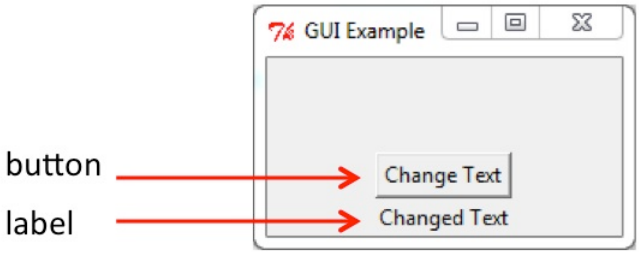
- Change the title
- Change the text in the button
- Change the text printed when the button is pressed
- Change the size (geometry) of the rectangular frame
- Move the button to the top of the frame

2 Exercise 2: Label Widget

This exercise:

- Introduces the label widget
- Practices getting and setting attributes.

Run the given program: exercise2.py. It should display the following:



The widgets are:

- Label: a single line text that is displayed
- Button: (see exercise 1)

The intended behaviour is:

- Text is entered in entry widget

- When the button is pressed the text changes, toggling between two messages.

Exercise 2.1: Getting and Setting Attributes

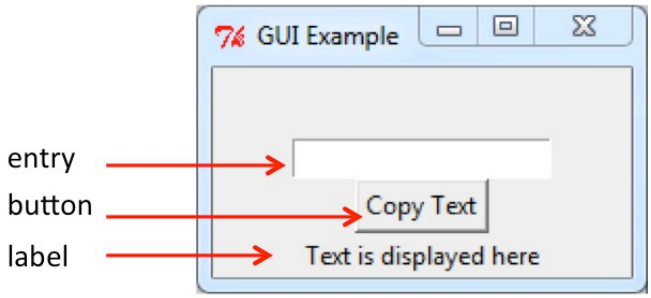
In the given code, pressing the button only changes the label text once. Change it so that it toggles, as described above.

Exercise 2.2: Further Attributes

The background colour of the button is determined by an attribute 'bg' or 'background' (the two are equivalent). Further enhance the program to cycle the background colour of the button through 3 different colours, as well as changing the txt.

3 Exercise 3: Adding an Entry Widget

This exercise adds an 'entry' widget to the previous exercise. Run the given program exercise3.py, displaying:



The widgets are:

- Label: a single line text that is displayed (see exercise 2)
- Button: (see exercise 1)
- Entry: a single line text that can be entered

The intended behaviour is:

- Text is entered in entry widget
- When the button is pressed the label text is updated with the entered text

Exercise 3.1: Complete Program

In the provided program, when you enter text in the box (the Entry widget) and press the button, it only prints the text from the entry. Complete it to give the intended behaviour described above.

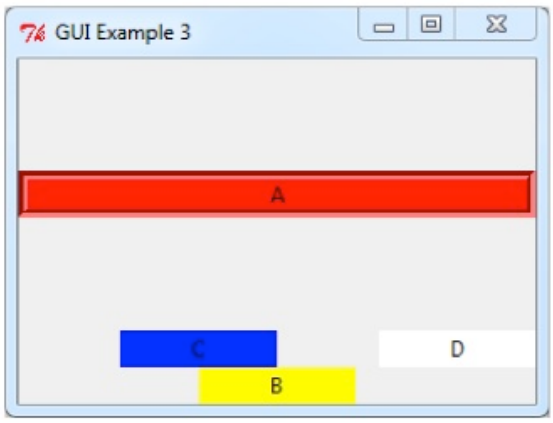
Exercise 3.2: Elaborations

- When the button is pressed, check if the entered text is blank (i.e. has zero length). If so, do not copy it but instead set the background of the button red. Restore the original background colour when the button is pressed and some text has been entered.
- After the button has been pressed and the label changed, make the next press of the button clear the text in the entry widget. Change the button text so that the user understands what is happening.

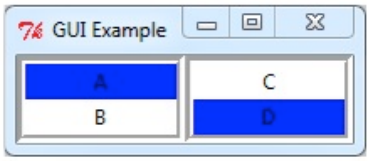
4 Exercise 4: Managing Layout

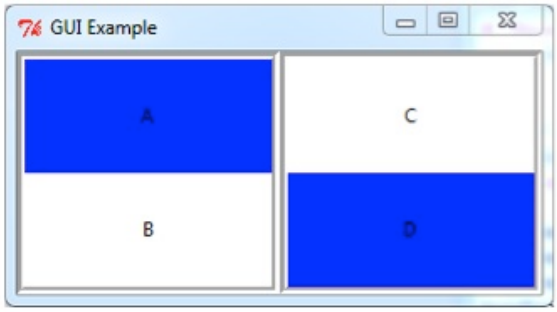
The aim of this exercise is to learn more about layout using the 'pack' layout manager. *The principles of packing are described in the additional notes at the end*.

The program exercise4.py displays the following (left hand picture shows the original display and the right hand side shows what happens when the window is resized):



The desired behaviour is shown below. The four labels are positioned at the corners and the labels fill the space when the window is resized.





<u>Exercise 4.1</u>: Arrange the labels is a square grid

With the pack layout manager this means introduces extra frames so that the labels are in the frames and the frames are in the top-level window. In the diagram above, the frames have a border so they can be seen.

<u>Exercise 4.2</u>: Support resizing

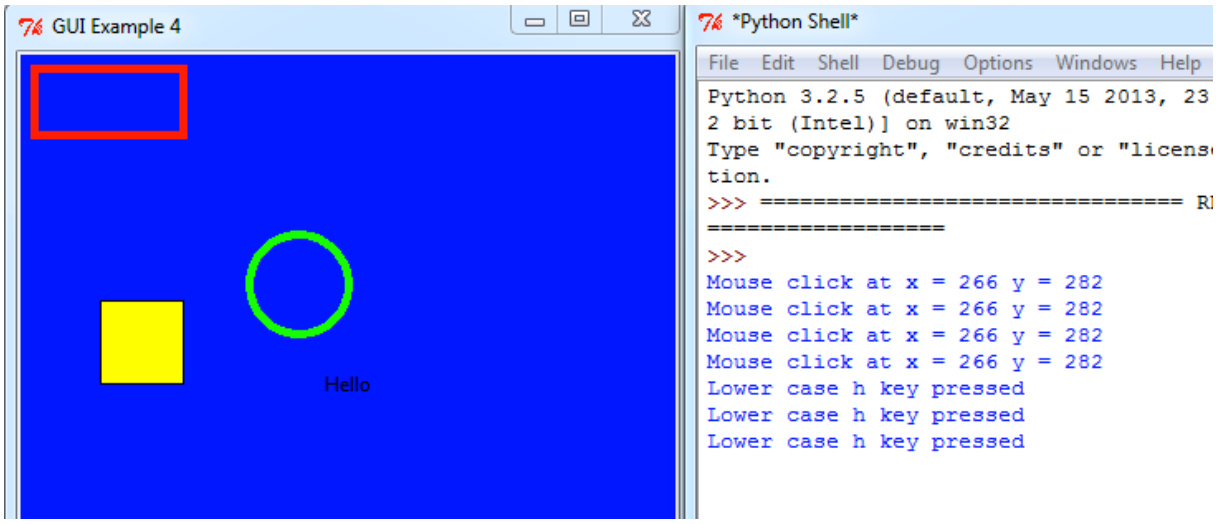
Use the 'expand' and 'fill' attributes of the pack method to make the labels grow and expand into the available space. There is more guidance in code comments.

5 Exercise 5: The Drawing Canvas and Events

The aim of this exercise is to learn about the drawing canvas and also to use two new types of events:

1. Key press events
2. Mouse click events

The program exercise5.py displays a canvas with some shapes:



In addition, two events are bound:

- Pressing the key 'h'
- Clicking the left mouse button

Exercise 5.1: Draw a Square where the mouse is clicked

Instead of always drawing the same shapes, use the mouse to draw a square: the top-left corner of the square goes where the mouse is clicked.

Exercise 5.2: Change the shape, colour and fill

Use keys to specify the shape, colour and whether the shape is filled. For example:

- Shape: 's' for square, 'c' for circle
- Filling: 'F' for filled, 'f' for unfilled
- Colour: 'y' for yellow, 'r' for red

Exercise 5.3: Interface Design

Using a pencil and paper, sketch some better interfaces to draw shapes. Consider either a) how to show what the current drawing options are or b) alternative ways to specify the shape, colour and filling, plus other features that could be useful.

6 Additional Notes on Layout

When a widget is created it needs to be positioned inside its parent. The positioning also needs to consider the possibility that the window may be resized. Layout is the responsibility of a layout manager: Tkinter offers a choice of layout managers.

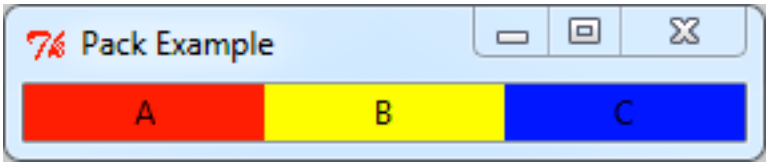
These notes consider the 'pack' layout manager. It is simplest to get started with and also behaves ok for resizing. Other layout managers are 'place' and 'grid'; the latter is often recommended for more complex layouts.

The following two example program fragments illustrate the principles:

```
#
# Create three labels of given width
#
bA = Label(app, text="A", width=12, bg='red')
bB = Label(app, text="B", width=12, bg='yellow')
bC = Label(app, text="C", width=12, bg='blue')

# Pack horizontally
# -----------------
# Horizontal packing with
#   side = "left"
#   side = "right"
bA.pack(side='left')
bB.pack(side='left')
bC.pack(side='left')
```

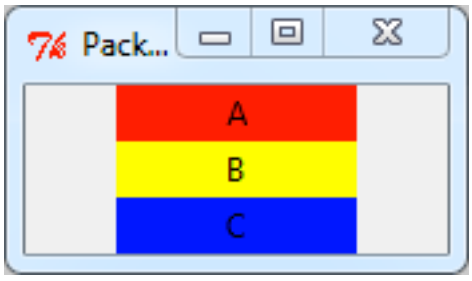
gives the display:



Whereas the vertical packing

```
# Pack vertically
# ---------------
# Vertical packing with
#   side = "top"
#   side = "bottom"
bA.pack(side='top')
bB.pack(side='top')
bC.pack(side='top')
```

displays:



There are two ways to control layout further using pack:

1. Create extra frames to hold widgets. For example, to position widget in the four quadrants of a square, create frames for the two and bottom halves and pack these vertically, then pack the widgets horizontally into the two frames.
2. Use the 'expand' and 'fill' attributes of pack.
    a. Fill (values X, Y, BOTH) determines the direction in which a widget can grow to fill available space.
    b. Expand (integer value) determines whether the containing frame gives more space to the widgets it contains. A zero value means no expansion. A positive value determine the relative expansion of each widget.