

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.

Contents

1	Exercise 1: A First GUI Program	1
2	Exercise 2: Adding a Label and Entry Widget.....	3
3	Exercise 3: Managing Layout.....	4
4	Exercise 4: The Drawing Canvas and Events.....	5
5	Exercise 5: Dialogs and Menu	6
6	Additional Notes	8
6.1	Layout.....	8
6.2	The Top-Level Window.....	9
6.3	Message Dialogs	9
7	Recommended References and Sources.....	10

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.

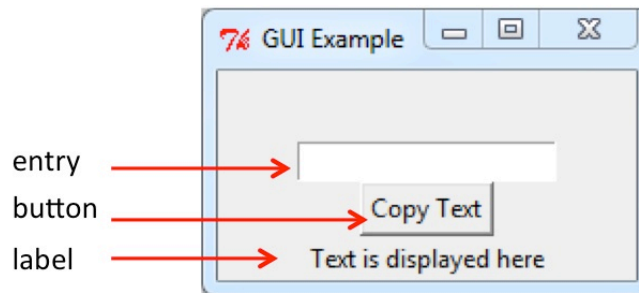
Exercise 1.2: Modify the Program

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: Adding a Label and Entry Widget

The aim of this exercise is to try out getting and setting attributes. The exercise is in two parts: at the end, your program should display the following:



The widgets are:

- Label: a single line text that is displayed
- 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 2.1: Part A: Getting and Setting Attributes

Run the given program (exercise2A.py); this version just has a button and a label. Pressing the button once changes the text of the label. Change it so that the text changes on each press, toggling between two messages.

Exercise 2.2: Part B: Complete Program

Run the given program (exercise2B.py); this part adds the entry widget. When you enter text in the box (the Entry widget) and press the button, it only prints the text from the entry. Complete it so that it behaves as described above.

Exercise 2.3: 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.

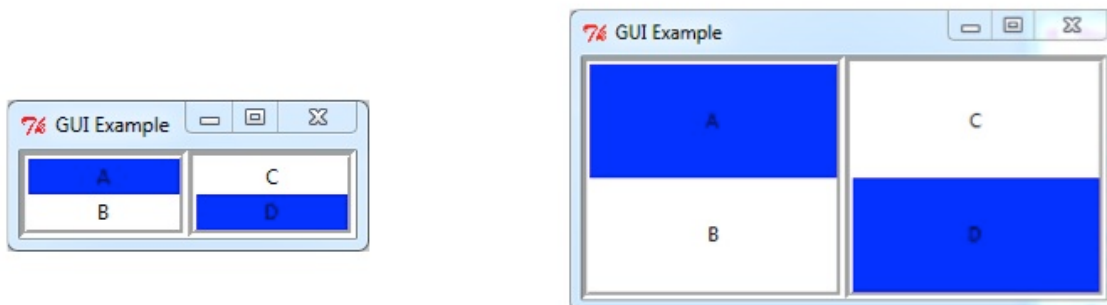
3 Exercise 3: Managing Layout

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

The program exercise3.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.



Exercise 3.1: Arrange the labels in 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.

Exercise 3.2: Support resizing

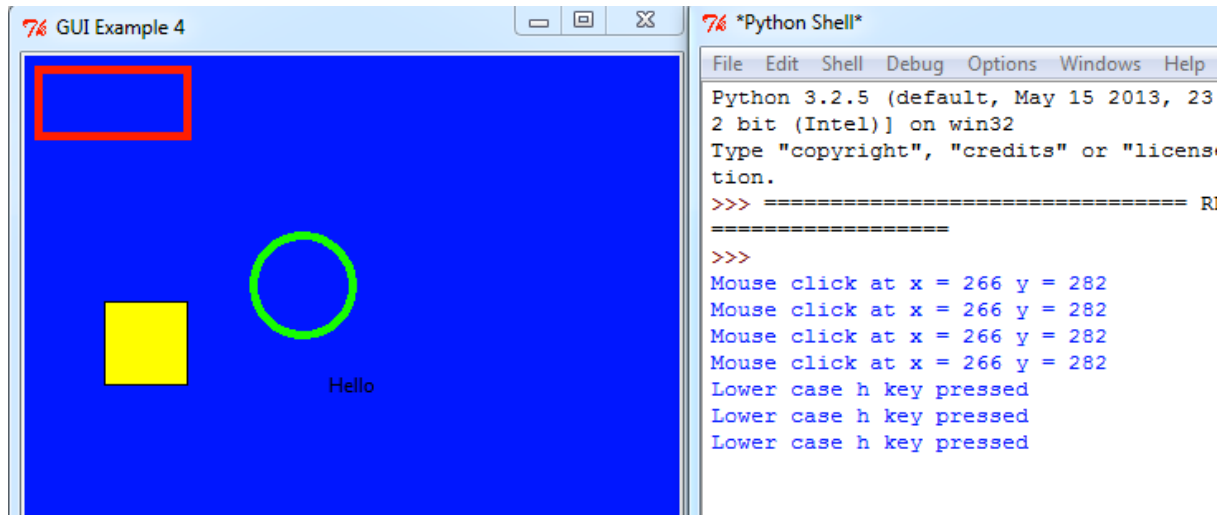
Use the 'expand' and 'fill' attributes of the pack method to make the labels grow and expand into the available space. There is more guidance in code comments.

4 Exercise 4: 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 `exercise4.py` displays a canvas with some shapes:



In addition, two events are bound:

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

Exercise 4.1: 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 4.2: 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 4.3: Interface Design

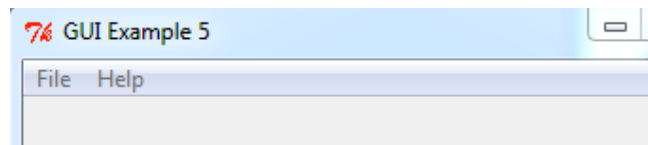
Using a pencil and paper, sketch some better interfaces to draw shapes. Consider either a) how to show what the current drawing options are or b) alternative ways to specify the shape, colour and filling, plus other features that could be useful.

5 Exercise 5: Dialogs and Menu

The aim of this exercise is to use dialogs and menus.

- A dialog displays a window requiring a response before the user can continue. We will look at message boxes and a file choosing dialog.
- A menu shows a menubar with menu items that expand when clicked to offer some command.

The program `example5.py` displays a frame, with two menu items.



It does not do much yet! Change it to a simple application that can read a file, convert it to upper case and save it again. The commands are:

Menu	Command	Description
File	Open	Open a new file
	Save	Save the existing file
	Quit	Quit the application
Edit	Convert to upper	Convert the file contents to upper case
About	Help	Useful help
	About	Information about the application

Exercise 5.1: Add menu items

Add the new menu and menu items. At first, do not give a command.

Exercise 5.2: Implement Functions

Implement the functions to act on the command. You can use

- `open(filename, mode)` to open the file with mode 'r' and 'w'
- `f.close()` to close a file
- `f.read()` to read a whole file
- `s.upper()` to convert a string `s` to uppercase (returns a new string)
- `f.write(string)` to write a string to the file.

Exercise 5.3: Add checks

Add checks so that a) the program never crashes and b) the user does not lose work. The following table suggest which checks are needed. Display suitable messages in each case.

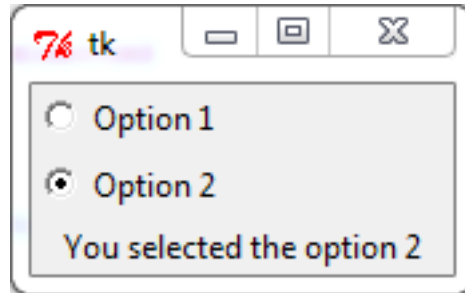
Command	Checks Needed
Open	Check for unsaved changes to the current file (Question)
Save	Check a file is open (Error) Check that changes need saving (Info)
Quit	Check for unsaved changes to the current file (Question)
Convert to upper	Check a file is open (Error)

6 Exercise 6: Radio Button

The aim of this exercise is to use a radio button.

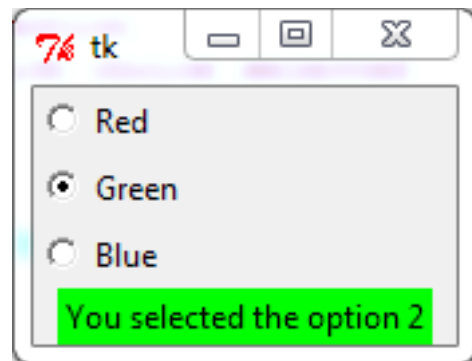
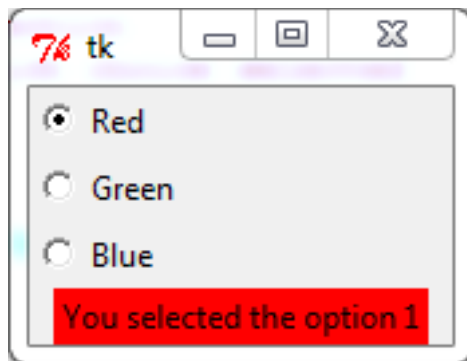
The program exercise6.py is shown below:

- There are three radio buttons
- Selecting one of the buttons changes the text of the label.



Exercise 6.1: Enhanced the program

The complete program should change the colour of the button as well as the text, for a suitable choice of three (or more) colours.



7 Additional Notes

7.1 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.

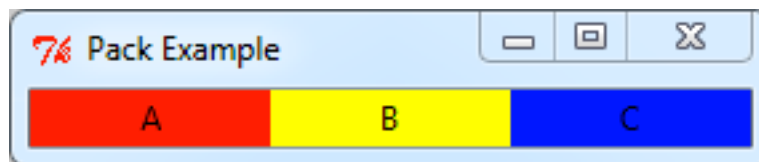
These notes consider the 'pack' layout manager. It is simplest to get started with and also behaves ok for resizing. Other layout managers are 'place' and 'grid'; the latter is often recommended for more complex layouts.

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')
```

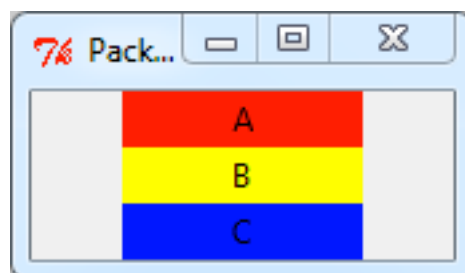
gives the display:



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.

7.2 The Top-Level Window

A *top-level window* is a window that has an independent existence under the window manager. It has the standard icons for closing or minimizing the window, and can be moved and resized independently. Your application can use any number of top-level windows.

A top level window can conveniently be created

Attributes

bg or background	The background color of the window.
menu	To provide this window with a top-level menubar, supply a Menu widget as the value of this option.

Methods

`.geometry(newGeometry=None)`

Set the window geometry. For example:

- 200x400
- 200x400+10+20
- 200x400-10-20

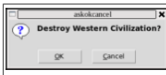
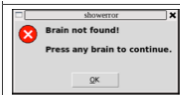
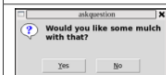

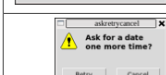
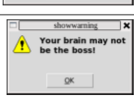
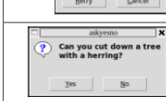
If the argument is omitted, the current geometry string is returned.

`.title(text=None)`

Set the window title. If the argument is omitted, returns the current title.

7.3 Message Dialogs

The following message dialogs are useful:

	<code>.askokcancel(title, message, options)</code>		<code>.showerror(title, message, options)</code>
	<code>.askquestion(title, message, options)</code>		<code>.showinfo(title, message, options)</code>
	<code>.askretrycancel(title, message, options)</code>		<code>.showwarning(title, message, options)</code>
	<code>.askyesno(title, message, options)</code>		

On messages with a choice, the default choice is the first one. Use the default option

```
default = C
```

to change this where C = CANCEL, IGNORE, OK, NO, RETRY, or YES

Each of the “ask...” pop-up functions returns a value that depends on which button the user pushed to remove the pop-up.

- askokcancel, askretrycancel, and askyesno all return a bool value: True for “OK” or “Yes” choices, False for “No” or “Cancel” choices.
- askquestion returns u'yes' for “Yes”, or u'no' for “No”.

8 Recommended References and Sources

1. The most comprehensive summary is ‘Tkinter 8.5 reference: a GUI for Python’, available from <http://infohost.nmt.edu/tcc/help/pubs/tkinter/web/index.html> either as a PDF or online.
2. A good online tutorial is http://www.python-course.eu/python_tkinter.php, part of a larger Python tutorial
3. This tutorial <http://www.tutorialspoint.com/python/index.htm> also has a section on Tkinter but I find it harder to navigate.