Building user interfaces with Tkinter

using Python 3 on the Raspberry Pi

Mr J. Dent

8th December 2013

ABOUT TKINTER 1

Tkinter is a Standard Python Graphical User Interface (GUI):
Why the name Tkinter? – Tool Kit Interface

Import Tkinter module
```
      from tkinter import *
```

Initialise Tkinter
```
      root = Tk()
```

Add code to draw widgets
```
      # code goes here
```

Run the Tkinter main event loop (processes button presses etc)
```
      root.mainloop()
```

INITIALISING EVERYTHING 2

```
from tkinter import *
root = Tk()
root.resizable(0,0) #fixed size window
app = Frame(root)   #create frame to hold widgets
app.grid()          #grid layout for widgets

root.title("Pythagoras") #window title
root.geometry("400x600") #window size
```

If you do not specify the geometry, a default screen size will be chosen for you.

TKINTER WIDGETS 3

Some Tkinter Widgets that can be used in a graphical user interface:-
Buttons, Labels, Menus, Messages, Frames, Text, Lists, Scrollbars, etc.

You can use the grid manager – `grid()` to place items next to each other using row and column arguments.

Alternatives for managing geometry include `pack()` or `place()`

`Place()` requires x and y coordinates, `pack()` places in a specific window location i.e. top left

If you do not specify the geometry of widgets, they will not display

ADDING PICTURES 4

(See **FACTS ABOUT PHOTO IMAGE** for help about pictures and images)

```
# create a Label and import the picture
picture = Label(app)
picture.grid(row=0, columnspan=3, sticky=W)

# 'sticky = W' means left aligned (West)
pythagoras = PhotoImage(file="pythagoras.gif")

#needs to be in same directory
picture["image"] = pythagoras
```

FACTS ABOUT PHOTO IMAGE 5

The PhotoImage class can only read GIF and PGM/PPM
File names cannot include any numbers or symbols

If you need to work with other file formats, the Python Imaging Library (PIL) supports over 30 image formats.

```
from PIL import Image, ImageTk
image = Image.open("lenna.jpg")
photo = ImageTk.PhotoImage(image)
```

You can use a PhotoImage instance everywhere Tkinter accepts an image object. An example:

```
label = Label(image=photo)
label.image = photo # keep a reference!
```

LABELS AND ENTRY BOXES 6

```
# create a Label and display text
lbl = Label(app, text="This will calculate the ↵
  length of the hypotenuse (c)")
lbl.grid(row=1, columnspan=3, sticky=W)

# create a Label and Entry box for a on row 2
lbl = Label(app, text="Enter length of side a")
lbl.grid(row=2, column=0, sticky=W)
value_a = Entry(app, width=10)
value_a.grid(row=2, column=1, sticky=W)

# create a Label and Entry box for b on row 3
lbl = Label(app, text="Enter length of side B")
lbl.grid(row=3, column=0, sticky=W)
value_b = Entry(app, width=10)
value_b.grid(row=3, column=1, sticky=W)
```

CREATING BUTTONS 7

```
# create a button onscreen on row 4
# when pressed it calls function 'calcHypotenuse'

calc = Button(app, text="Calculate!", ↵
  command=calcHypotenuse)
calc.grid(row=4, column=1, sticky=W)

# run the Tkinter main loop to process events
root.mainloop()
```

<u>This errors – why?</u>
<u>You need to **def** the function!</u>

CALCULATE THE HYPOTENUSE 8

Make sure this function features at the top of your code

```
def calcHypotenuse(*ignore):
  a = value_a.get() #store values from inputs
  b = value_b.get()
  asquared = float(a) * float(a)
  bsquared = float(b) * float(b)
  csquared = asquared + bsquared
  c = math.sqrt(csquared)

  #output result:
  txt = Label(app, text="Hypotenuse is "+str(c))
  txt.grid(row=6, sticky=W)
```

ACCESSING THE MATHS MODULE 9

You also need to include the following library at the top of your code so the `math.sqrt()` in `calcHypotenuse()` works:

```
import math
```

ABOUT PYTHAGORAS' THEOREM 10

Years ago, a Greek mathematician called Pythagoras named Pythagoras found an amazing fact about right-angled (90°) triangles:

If you placed a square fitting on each of the three sides, then the biggest square had the exact same area as the other two squares added together! The area of each square is the length of its side squared.

If we say that a and b are the short sides' lengths and c is the length of the longest side, Pythagoras's theorem is just one short equation:
$a^2 + b^2 = c^2$

So, if we know the lengths of two sides of a right angled triangle, we can find the length of the third side. (But remember it only works on right angled triangles!)

SETTING UP GPIO 11

```
# Import the required module:
import RPi.GPIO as GPIO

# Set the mode of numbering the pins:
GPIO.setmode(GPIO.BOARD)

# define constants for pins
LED = 10
SWITCH = 8

# GPIO pin 10 is the output:
GPIO.setup(LED, GPIO.OUT)

# GPIO pin 8 is the input:
GPIO.setup(SWITCH, GPIO.IN)
```

SWITCH AND LED USING GPIO 12

This example (based on the resources links later) connects the LED so that it is powered from 3V3 and the GPIO pin connects the other pin to ground. This means that a False is required to turn the LED on, and a True is required to turn the LED off.

```
# loop forever, setting LED on/off based on switch
while True:
  if GPIO.input(SWITCH): # pressed
    GPIO.output(LED, False) # on, see above
  else: # released
    GPIO.output(LED, True) # off, see above
```

CONTROLLING GPIO VIA TKINTER 13

You can add a Tkinter Button widget, so that when it is pressed, it controls your LED, try something like this:

```
def ledON():
  GPIO.output(LED, False) # on, see card 12
def ledOFF():
  GPIO.output(LED, True) # off, see card 12

onBtn = Button(app, text='turn it on', ↵
  command=ledON)
onBtn.grid(row=8, column=1, sticky=W)

offBtn = Button(app, text='turn it off', ↵
  command=ledOFF)
offBtn.grid(row=8, column=2, sticky=W)
```

USEFUL WEBSITES 14

http://www.mathsisfun.com/pythagoras.html

http://www.tutorialspoint.com/python/python_gui_programming.htm

http://www.cl.cam.ac.uk/projects/raspberrypi/tutorials/turing-machine/two.html

http://www.pythonware.com/library/

http://www.pythonware.com/products/pil

TKINTER WIDGET ATTRIBUTES 15

As a bonus, you might like to try the two different ways of setting Tkinter widget attributes. The second method can look a little more complex, but it allows you to change attributes after the widget has been created.

```
# set attributes when widget created
calc = Button(app, text="Calculate!", ↵
  command=calcHypotenuse)

# change attributes after widget has been created
calc["command"] = calcHypotenuse
calc["text"] = "yay!"
```