Lesson 7 –
GUI (Graphical User Interface) using Python

- S.P. Chong

Objectives

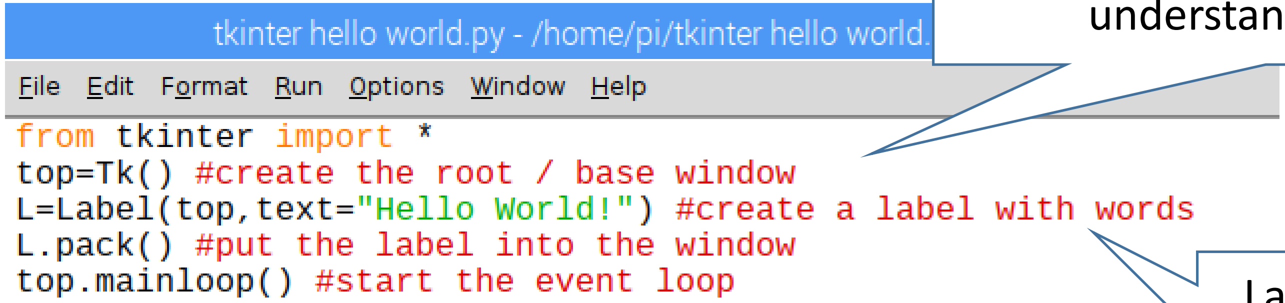
- In this lesson, you will the basics of **Tkinter**, a **graphical programming package** for Python, that allows you to add a **user interface** to your project.

- You will learn to add **buttons** and **message boxes**, create a **virtual button**, a **virtual gauge** and a **virtual** (blinking) **LED**, and draw on a **canvas**.
- You will also learn to add **MatPlotLib** graph to TKinter frame, to arrange graphic elements in a grid, and create a simple **survey form** and **menu system**.

- BTW, TKinter is pronounced as T-K-inter.

Hello world

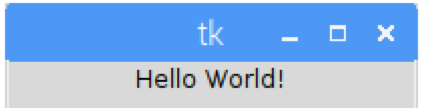
- Type the following Python program and run it:


tkinter hello world.py - /home/pi/tkinter hello world.
File Edit Format Run Options Window Help

```
from tkinter import *
top=Tk() #create the root / base window
L=Label(top,text="Hello World!") #create a label with words
L.pack() #put the label into the window
top.mainloop() #start the event loop
```

The comments will help you to understand the program.

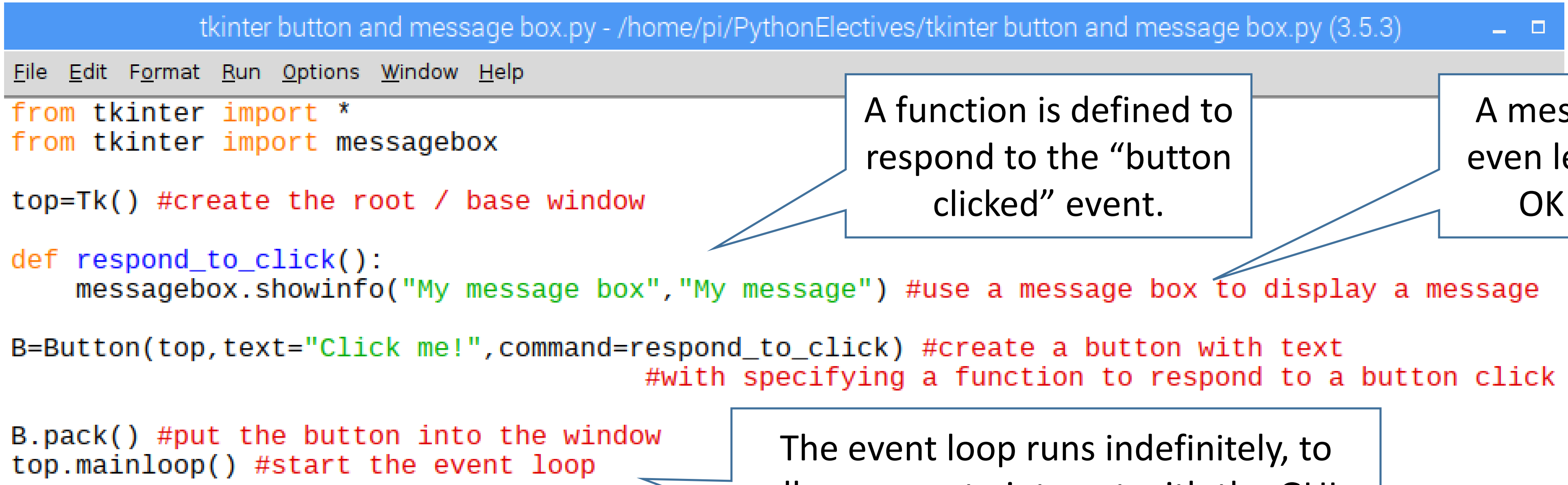
Label is one of the many useful “widgets”.



Results? A “label” with the text “Hello World!” in the window called “top”.

Button & MessageBox

- Let’s try this:

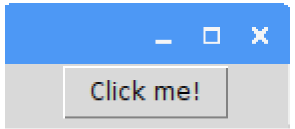

tkinter button and message box.py - /home/pi/PythonElectives/tkinter button and message box.py (3.5.3)
File Edit Format Run Options Window Help

```
from tkinter import *
from tkinter import messagebox

top=Tk() #create the root / base window

def respond_to_click():
    messagebox.showinfo("My message box","My message") #use a message box to display a message

B=Button(top,text="Click me!",command=respond_to_click) #create a button with text
                                      #with specifying a function to respond to a button click

B.pack() #put the button into the window
top.mainloop() #start the event loop
```

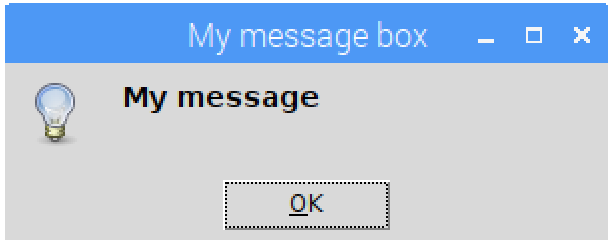
A function is defined to respond to the “button clicked” event.

A message box can even let you choose OK or Cancel.

The event loop runs indefinitely, to allow a user to interact with the GUI.



Results? A “button” with the text “Click me!”.

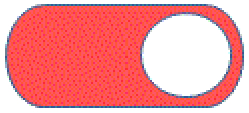


My message

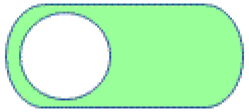
When you click on the button, a message box will be displayed.

A virtual button

Red button: off_button.gif



Green button: on_button.gif

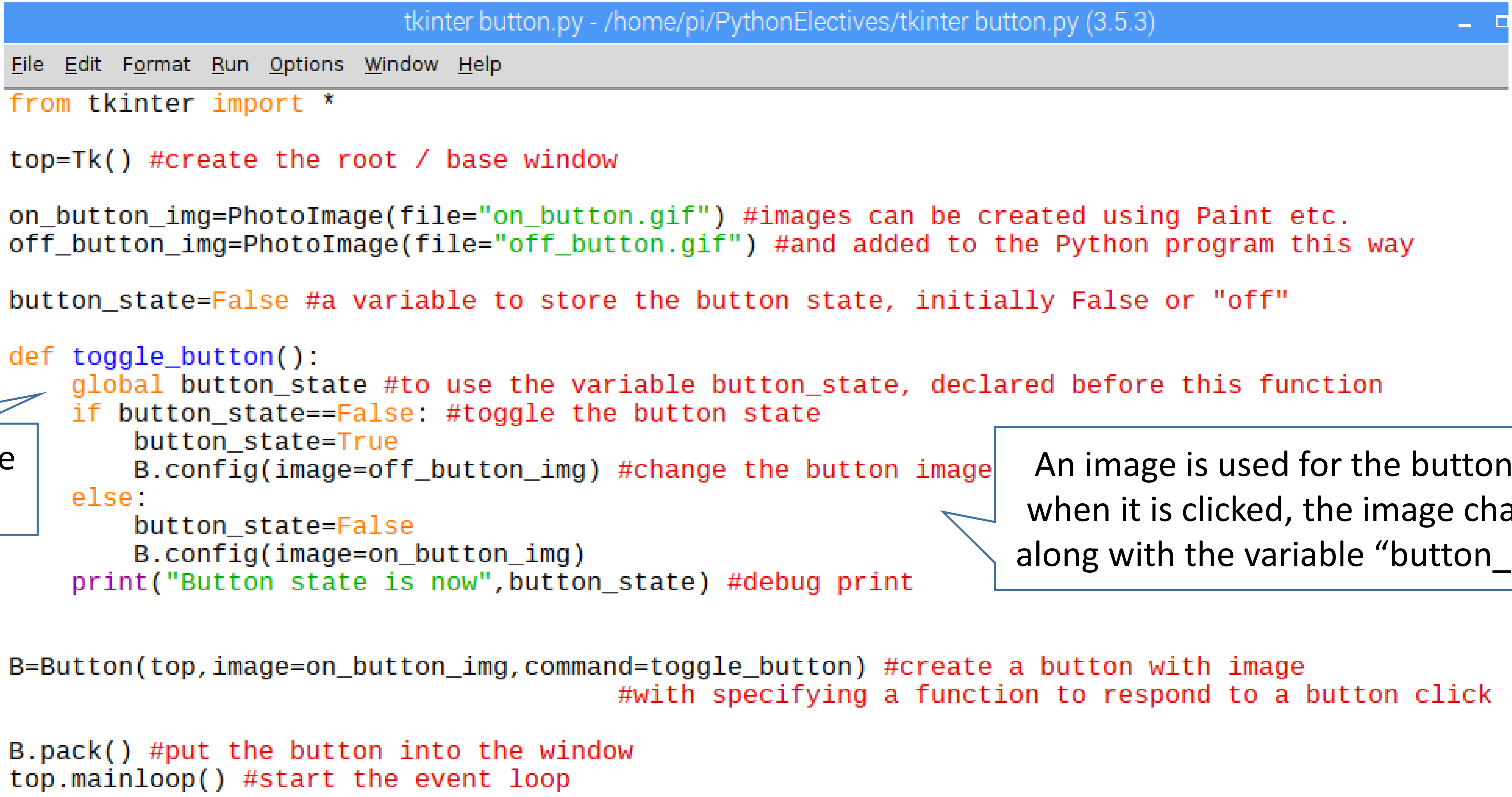


- Using suitable software tools e.g. Powerpoint + Paint, create the button images as shown, and save them as on_button.gif (green) and off_button.gif (red).
- The size can be 64 pixels x 128 pixels, for instance.

- Note that initially, the button is green (or OFF state!). It can be pressed to turn ON something.

A virtual button (cont.)

- Write the Python program shown, and run it.


tkinter button.py - /home/pi/PythonElectives/tkinter button.py (3.5.3)
File Edit Format Run Options Window Help

```
from tkinter import *

top=Tk() #create the root / base window

on_button_img=PhotoImage(file="on_button.gif") #images can be created using Paint etc.
off_button_img=PhotoImage(file="off_button.gif") #and added to the Python program this way

button_state=False #a variable to store the button state, initially False or "off"

def toggle_button():
    global button_state #to use the variable button_state, declared before this function
    if button_state==False: #toggle the button state
        button_state=True
        B.config(image=off_button_img) #change the button image
    else:
        button_state=False
        B.config(image=on_button_img)
    print("Button state is now",button_state) #debug print


B=Button(top,image=on_button_img,command=toggle_button) #create a button with image
                                       #with specifying a function to respond to a button click

B.pack() #put the button into the window
top.mainloop() #start the event loop
```

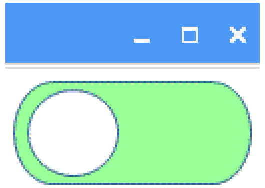
Try to understand the lines of code shown.

An image is used for the button, and when it is clicked, the image changes, along with the variable “button_state”.

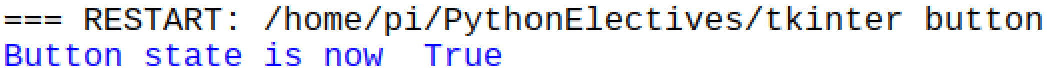
A virtual button (cont.)

- Results of running the program…

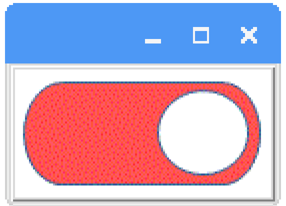
Initially…



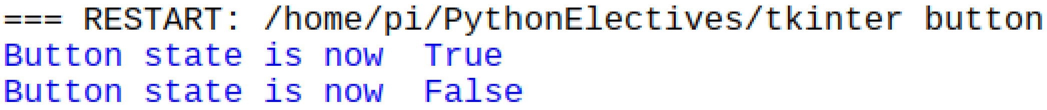
After (green) button is clicked…



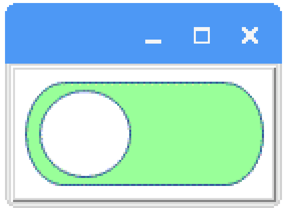
```
=== RESTART: /home/pi/PythonElectives/tkinter button
Button state is now  True
```



After (red) button is clicked…

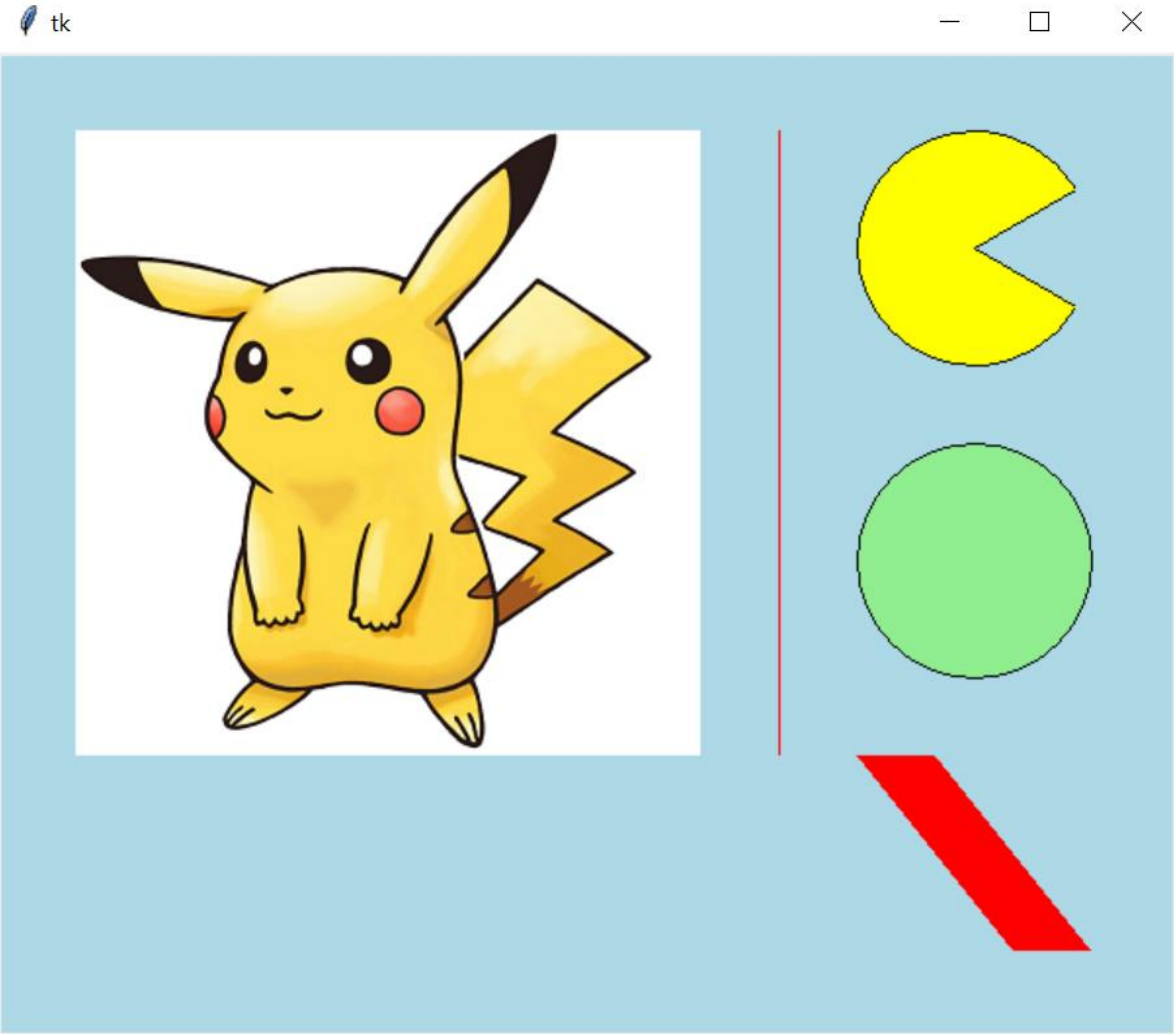


```
=== RESTART: /home/pi/PythonElectives/tkinter button
Button state is now  True
Button state is now  False
```

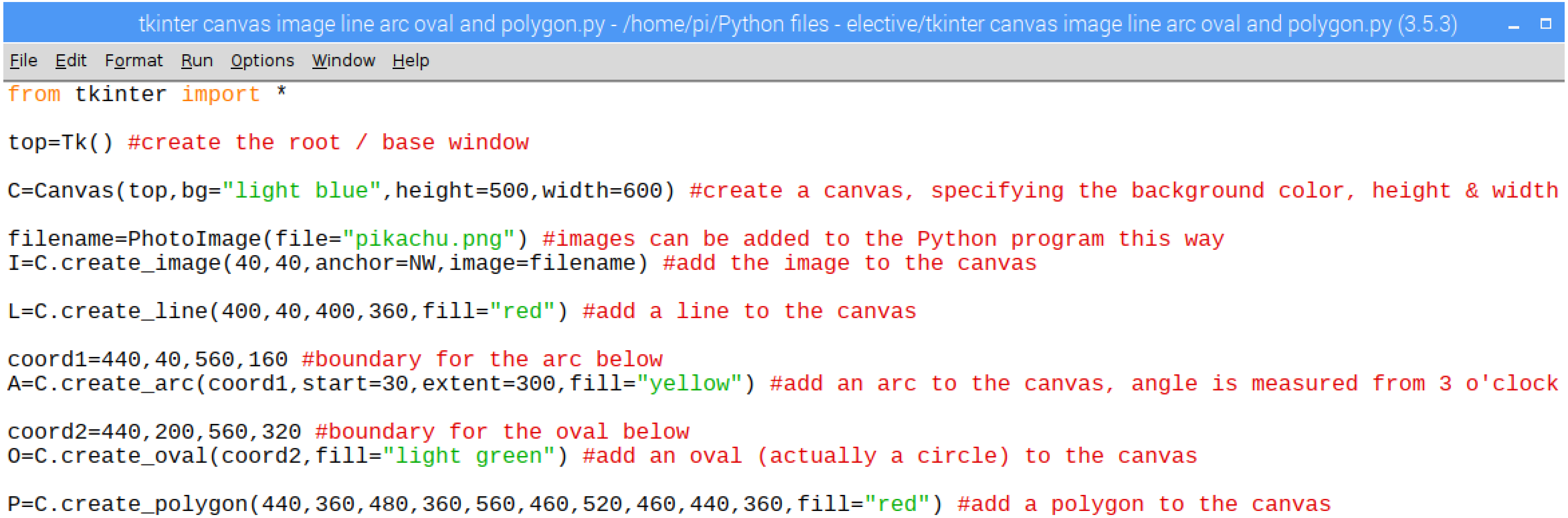


Canvas, image, line, arc, oval, polygon...

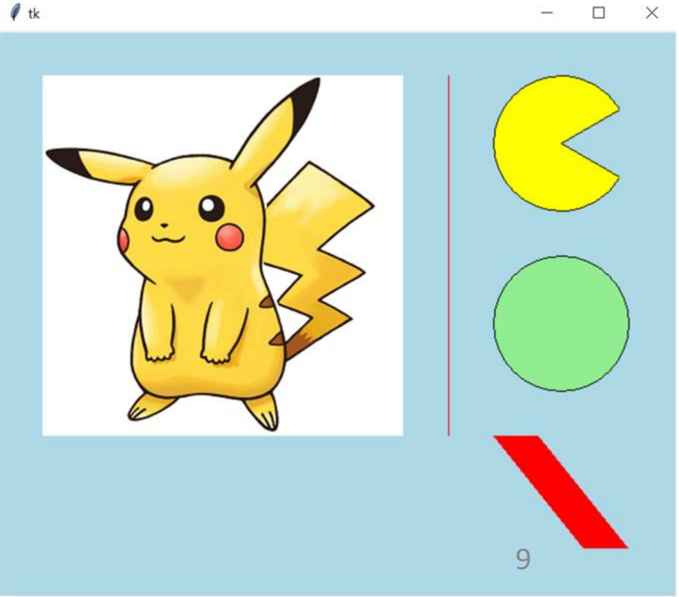
- A canvas can be used to draw various interesting things...



The program on the next slide shows how an image and some common shapes (line, arc, oval, polygon) can be put on a canvas...


tkinter canvas image line arc oval and polygon.py - /home/pi/Python files - elective/tkinter canvas image line arc oval and polygon.py (3.5.3)
File
Edit
Format
Run
Options
Window
Help

```
from tkinter import *

top=Tk() #create the root / base window

C=Canvas(top,bg="light blue",height=500,width=600) #create a canvas, specifying the background color, height & width

filename=PhotoImage(file="pikachu.png") #images can be added to the Python program this way
I=C.create_image(40,40,anchor=NW,image=filename) #add the image to the canvas

L=C.create_line(400,40,400,360,fill="red") #add a line to the canvas

coord1=440,40,560,160 #boundary for the arc below
A=C.create_arc(coord1,start=30,extent=300,fill="yellow") #add an arc to the canvas, angle is measured from 3 o'clock

coord2=440,200,560,320 #boundary for the oval below
O=C.create_oval(coord2,fill="light green") #add an oval (actually a circle) to the canvas

P=C.create_polygon(440,360,480,360,560,460,520,460,440,360,fill="red") #add a polygon to the canvas

C.pack() #put the canvas into the window
top.mainloop() #start the event loop
```



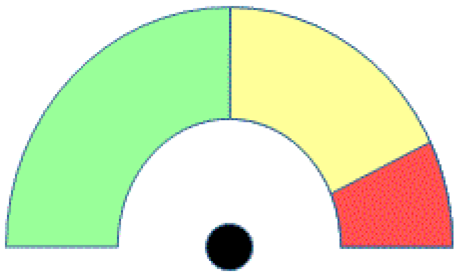
Canvas, image, line, arc, oval, polygon… (cont.)

A virtual gauge

- Using suitable software tools e.g. Powerpoint + Paint, create the gauge background image as shown, and save it as gauge.gif.
- The size can be 256 pixels x 256 pixels, for instance.

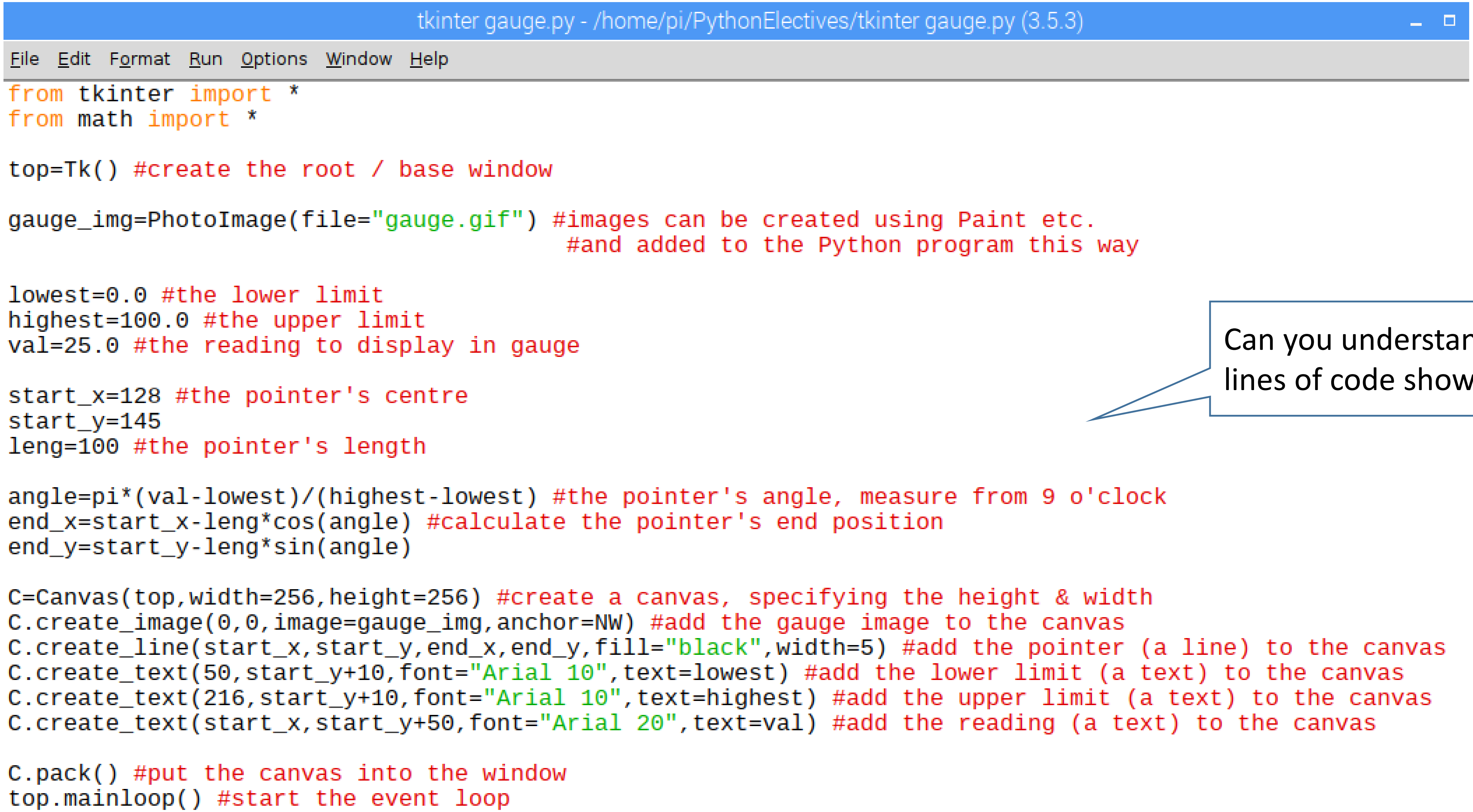
- Note that Python code will be used to add the needle, the value being displayed, as well as the lower & upper limits.

Gauge: gauge.gif



A virtual gauge (cont.)

- Write the Python program shown, and run it.

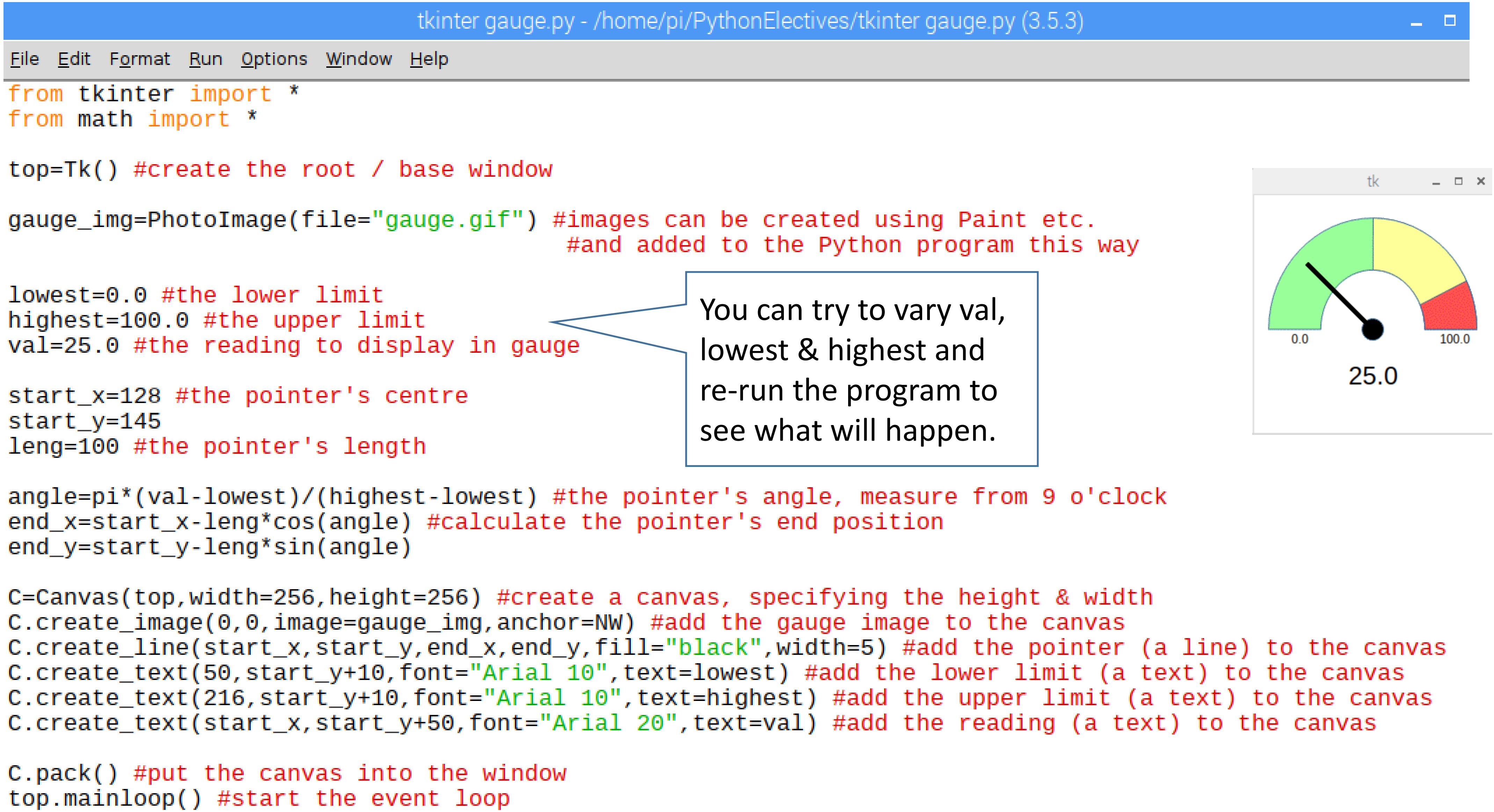

tkinter gauge.py - /home/pi/PythonElectives/tkinter gauge.py (3.5.3)
File Edit Format Run Options Window Help

```
from tkinter import *
from math import *

top=Tk() #create the root / base window

gauge_img=PhotoImage(file="gauge.gif") #images can be created using Paint etc.
                                        #and added to the Python program this way

lowest=0.0 #the lower limit
highest=100.0 #the upper limit
val=25.0 #the reading to display in gauge

start_x=128 #the pointer's centre
start_y=145
leng=100 #the pointer's length

angle=pi*(val-lowest)/(highest-lowest) #the pointer's angle, measure from 9 o'clock
end_x=start_x-leng*cos(angle) #calculate the pointer's end position
end_y=start_y-leng*sin(angle)

C=Canvas(top,width=256,height=256) #create a canvas, specifying the height & width
C.create_image(0,0,image=gauge_img,anchor=NW) #add the gauge image to the canvas
C.create_line(start_x,start_y,end_x,end_y,fill="black",width=5) #add the pointer (a line) to the canvas
C.create_text(50,start_y+10,font="Arial 10",text=lowest) #add the lower limit (a text) to the canvas
C.create_text(216,start_y+10,font="Arial 10",text=highest) #add the upper limit (a text) to the canvas
C.create_text(start_x,start_y+50,font="Arial 20",text=val) #add the reading (a text) to the canvas

C.pack() #put the canvas into the window
top.mainloop() #start the event loop
```

Can you understand the lines of code shown?

A virtual gauge (cont.)

- This is what you get when the program runs.


tkinter gauge.py - /home/pi/PythonElectives/tkinter gauge.py (3.5.3)
File Edit Format Run Options Window Help
tk
0.0
100.0
25.0

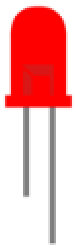
```
from tkinter import *
from math import *

top=Tk() #create the root / base window

gauge_img=PhotoImage(file="gauge.gif") #images can be created using Paint etc.
                                        #and added to the Python program this way

lowest=0.0 #the lower limit
highest=100.0 #the upper limit
val=25.0 #the reading to display in gauge

start_x=128 #the pointer's centre
start_y=145
leng=100 #the pointer's length

angle=pi*(val-lowest)/(highest-lowest) #the pointer's angle, measure from 9 o'clock
end_x=start_x-leng*cos(angle) #calculate the pointer's end position
end_y=start_y-leng*sin(angle)

C=Canvas(top,width=256,height=256) #create a canvas, specifying the height & width
C.create_image(0,0,image=gauge_img,anchor=NW) #add the gauge image to the canvas
C.create_line(start_x,start_y,end_x,end_y,fill="black",width=5) #add the pointer (a line) to the canvas
C.create_text(50,start_y+10,font="Arial 10",text=lowest) #add the lower limit (a text) to the canvas
C.create_text(216,start_y+10,font="Arial 10",text=highest) #add the upper limit (a text) to the canvas
C.create_text(start_x,start_y+50,font="Arial 20",text=val) #add the reading (a text) to the canvas

C.pack() #put the canvas into the window
top.mainloop() #start the event loop
```

You can try to vary val, lowest & highest and re-run the program to see what will happen.

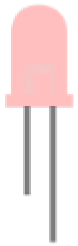
A virtual LED (& “after”)

- Using suitable software tools e.g. Powerpoint + Paint, create the LED images as shown, and save them as on_LED.gif & off_LED.gif.
- The size can be 128 pixels x 64 pixels, for instance.

On LED: on_LED.gif



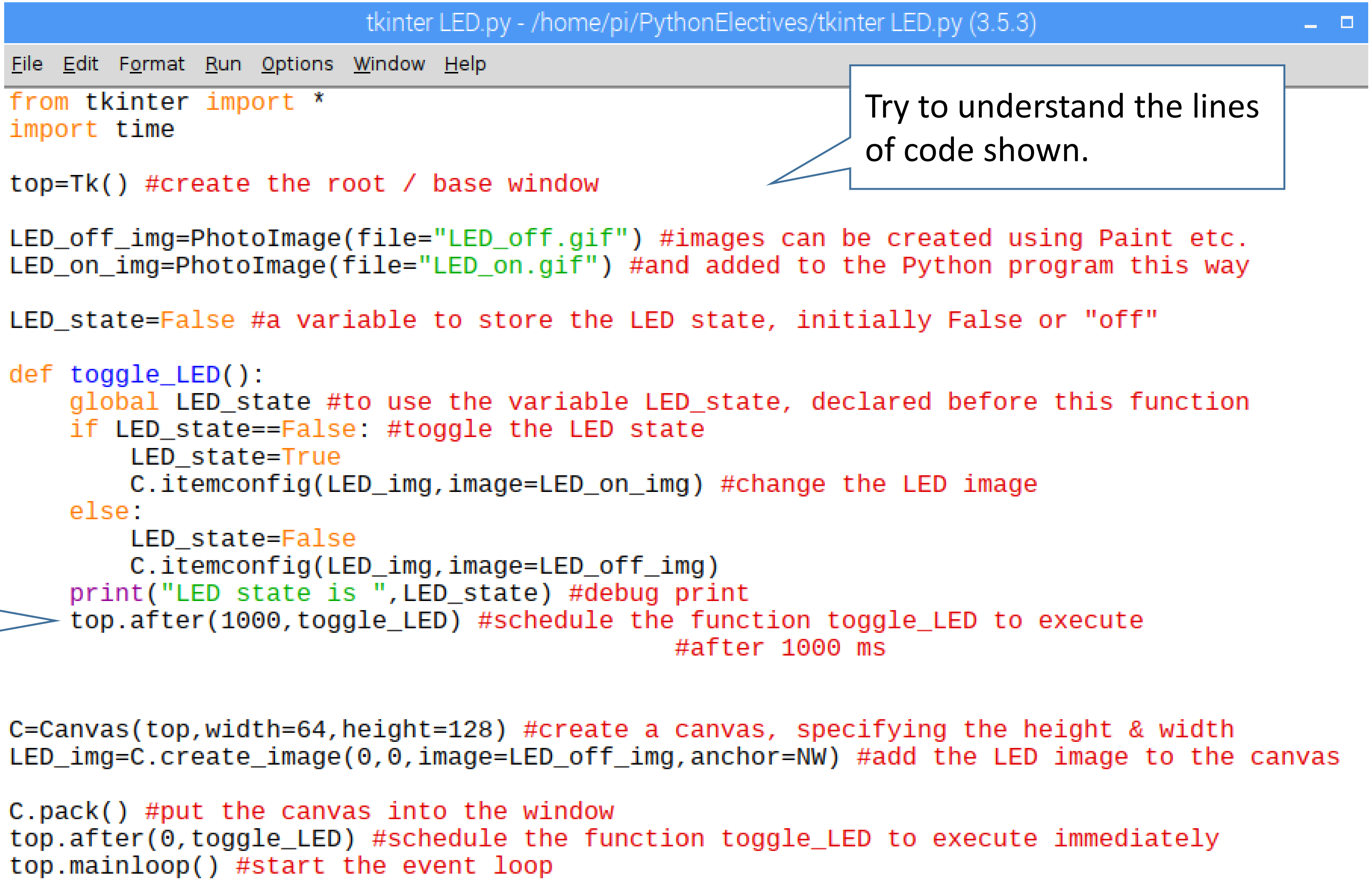
Off LED: off_LED.gif



A virtual LED (cont.)

- Write the Python program shown, and run it.
- Learn how “**after**” can be used to schedule an event.

“after” is used to schedule an event (a function call) in the future.

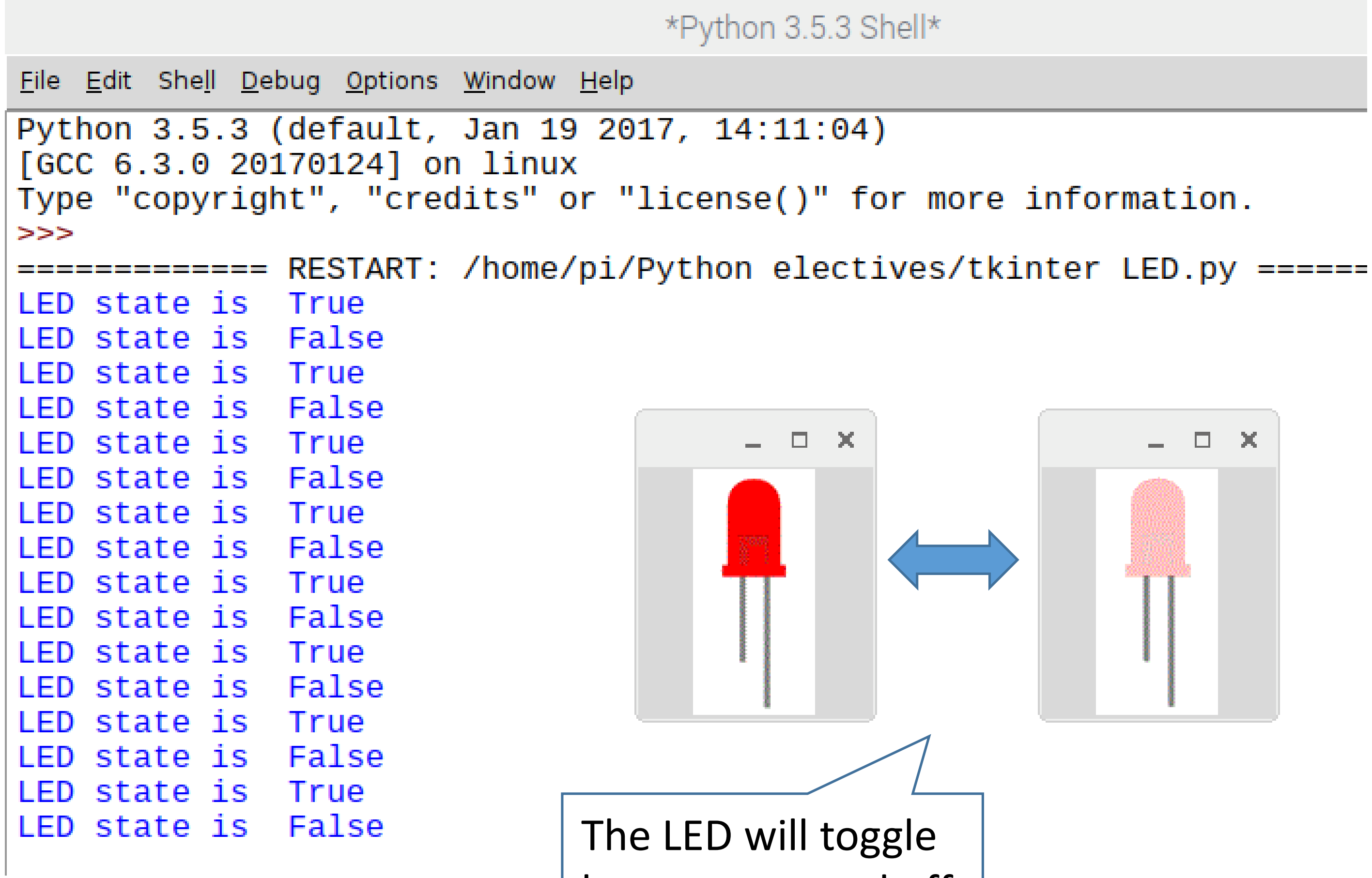

tkinter LED.py - /home/pi/PythonElectives/tkinter LED.py (3.5.3)
File Edit Format Run Options Window Help

```
from tkinter import *
import time

top=Tk() #create the root / base window

LED_off_img=PhotoImage(file="LED_off.gif") #images can be created using Paint etc.
LED_on_img=PhotoImage(file="LED_on.gif") #and added to the Python program this way

LED_state=False #a variable to store the LED state, initially False or "off"

def toggle_LED():
    global LED_state #to use the variable LED_state, declared before this function
    if LED_state==False: #toggle the LED state
        LED_state=True
        C.itemconfig(LED_img,image=LED_on_img) #change the LED image
    else:
        LED_state=False
        C.itemconfig(LED_img,image=LED_off_img)
    print("LED state is ",LED_state) #debug print
    top.after(1000,toggle_LED) #schedule the function toggle_LED to execute
                                            #after 1000 ms


C=Canvas(top,width=64,height=128) #create a canvas, specifying the height & width
LED_img=C.create_image(0,0,image=LED_off_img,anchor=NW) #add the LED image to the canvas

C.pack() #put the canvas into the window
top.after(0,toggle_LED) #schedule the function toggle_LED to execute immediately
top.mainloop() #start the event loop
```

Try to understand the lines of code shown.

A virtual LED (cont.)

- This is what you get when the program runs.

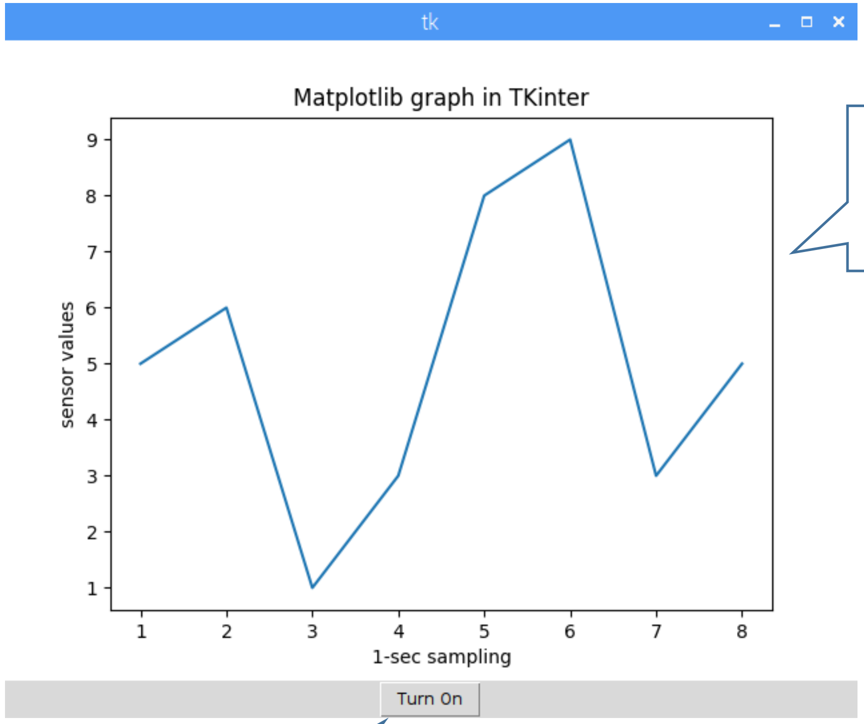


```
Python 3.5.3 (default, Jan 19 2017, 14:11:04)
[GCC 6.3.0 20170124] on linux
Type "copyright", "credits" or "license()" for more information.
>>>
============= RESTART: /home/pi/Python electives/tkinter LED.py ======
LED state is  True
LED state is  False
LED state is  True
LED state is  False
LED state is  True
LED state is  False
LED state is  True
LED state is  False
LED state is  True
LED state is  False
LED state is  True
LED state is  False
LED state is  True
LED state is  False
LED state is  True
LED state is  False
```

The LED will toggle between on and off, at 1 second interval.

MatPlotLib graph (& Button) in Tkinter window

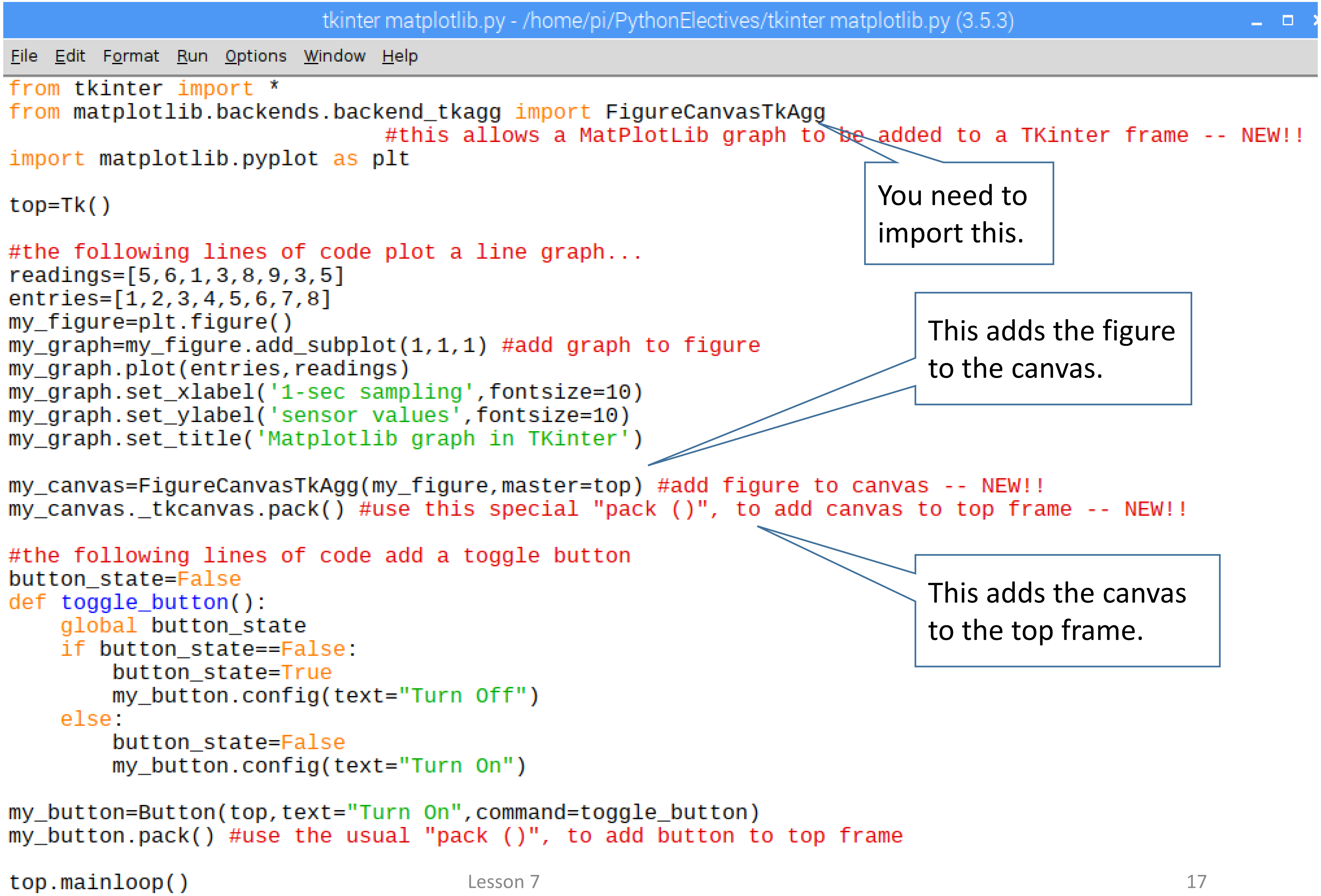
- Eventually, we are going to create a “dashboard”, where sensor data (e.g. temperature) can be **monitored**, and actuator (e.g. motor) can be **controlled**.
- Let’s see how a graph (of sensor data) can be displayed with a button.
- The program is shown on the next slide.



MatPlotLib graph.

Button.

MatPlotLib graph (& Button) in Tkinter window (cont.)


tkinter matplotlib.py - /home/pi/PythonElectives/tkinter matplotlib.py (3.5.3)
File Edit Format Run Options Window Help

```
from tkinter import *
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
                             #this allows a MatPlotLib graph to be added to a TKinter frame -- NEW!!
import matplotlib.pyplot as plt

top=Tk()

#the following lines of code plot a line graph...
readings=[5,6,1,3,8,9,3,5]
entries=[1,2,3,4,5,6,7,8]
my_figure=plt.figure()
my_graph=my_figure.add_subplot(1,1,1) #add graph to figure
my_graph.plot(entries,readings)
my_graph.set_xlabel('1-sec sampling',fontsize=10)
my_graph.set_ylabel('sensor values',fontsize=10)
my_graph.set_title('Matplotlib graph in TKinter')

my_canvas=FigureCanvasTkAgg(my_figure,master=top) #add figure to canvas -- NEW!!
my_canvas._tkcanvas.pack() #use this special "pack ()", to add canvas to top frame -- NEW!!

#the following lines of code add a toggle button
button_state=False
def toggle_button():
    global button_state
    if button_state==False:
        button_state=True
        my_button.config(text="Turn Off")
    else:
        button_state=False
        my_button.config(text="Turn On")

my_button=Button(top,text="Turn On",command=toggle_button)
my_button.pack() #use the usual "pack ()", to add button to top frame

top.mainloop()
```

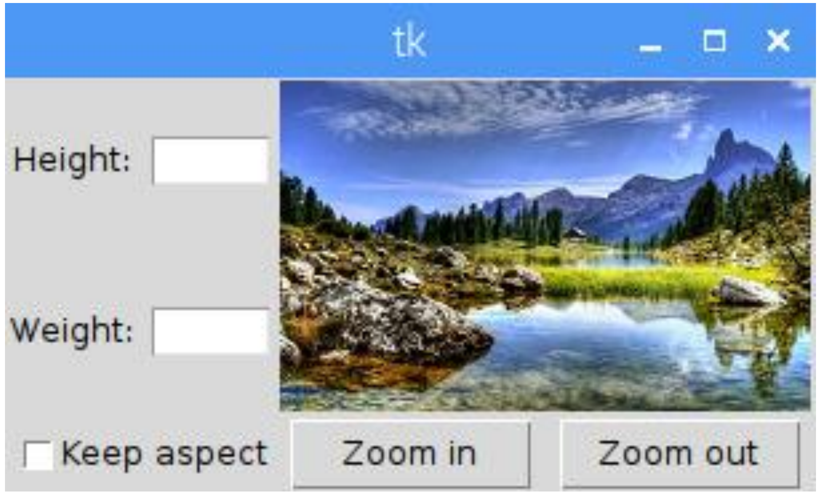
You need to import this.

This adds the figure to the canvas.

This adds the canvas to the top frame.

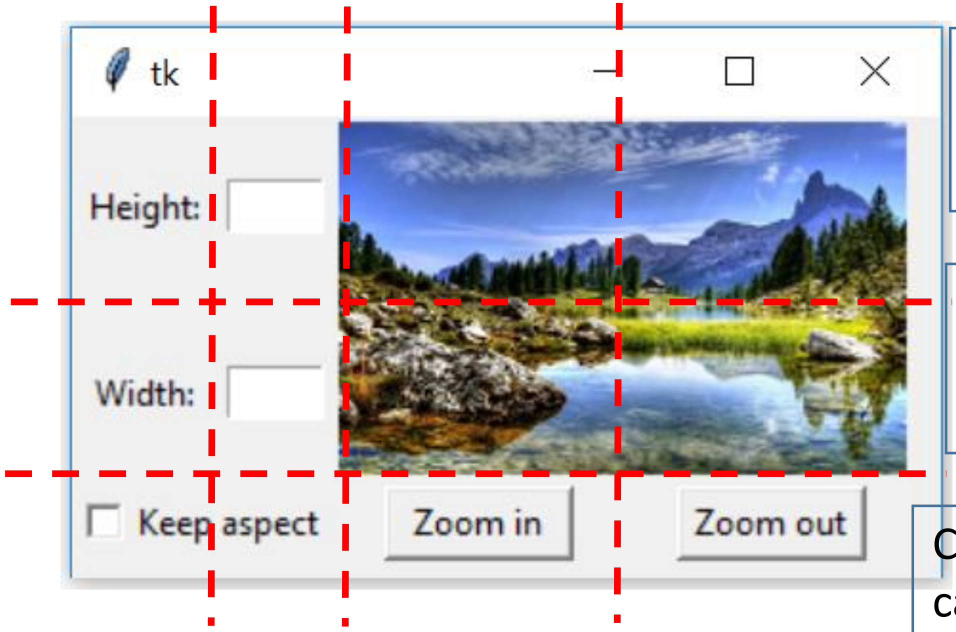
Arranging graphic elements using “grid”

- When creating a GUI, many graphic elements (labels, entries, check button, image, buttons etc.) need to be arranged nicely on the computer screen.
- The next slide shows how “grid” can be used to do this.



Arranging graphic elements using “grid” (cont.)

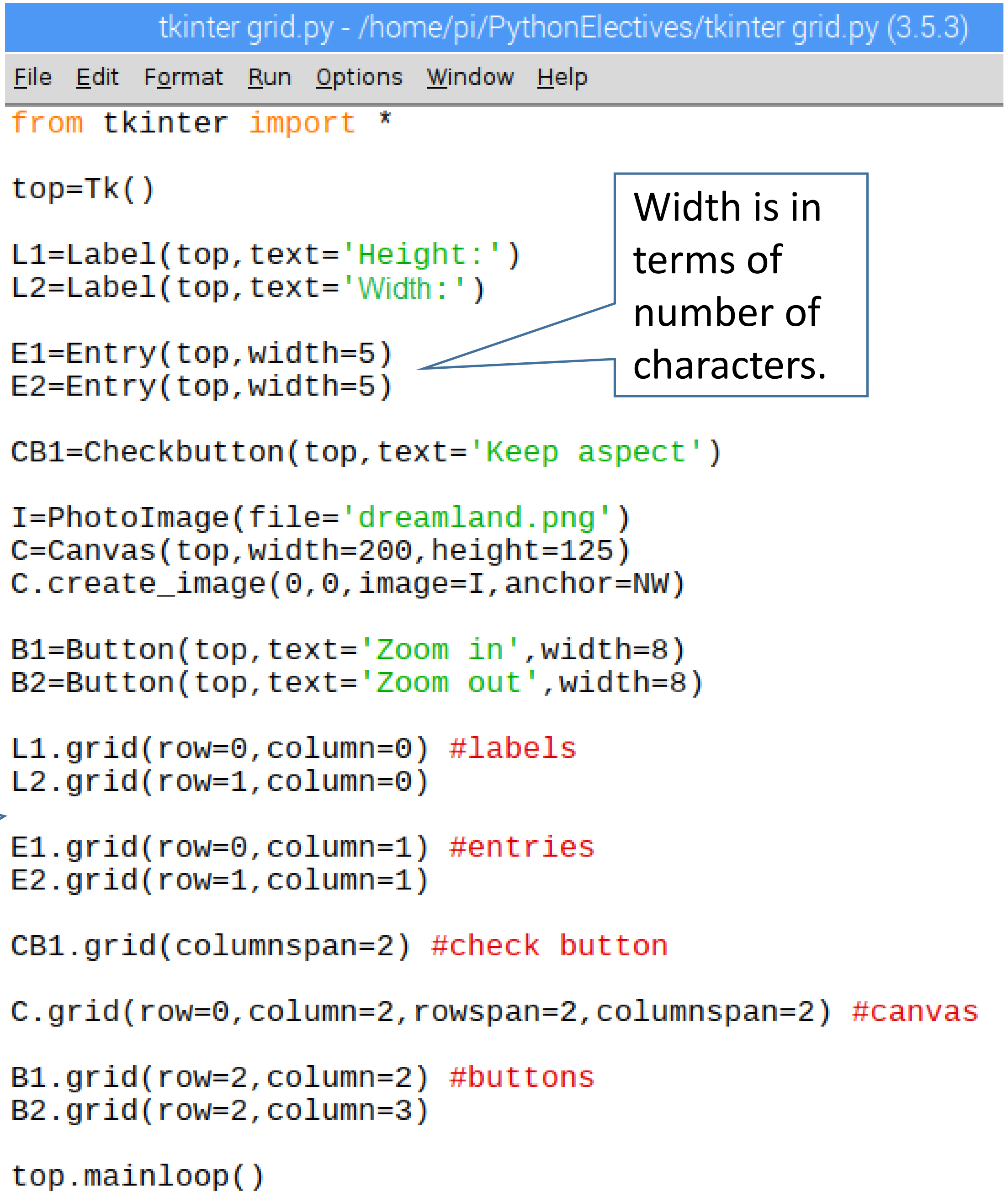
- Note that when grid( )is used, pack( ) will not be used.
- The row & column sizes are not fixed.



row 0 column 0 refers to top left position.

Labels are on column 0, entries are on column 1.

Column / row span can be used for bigger elements.

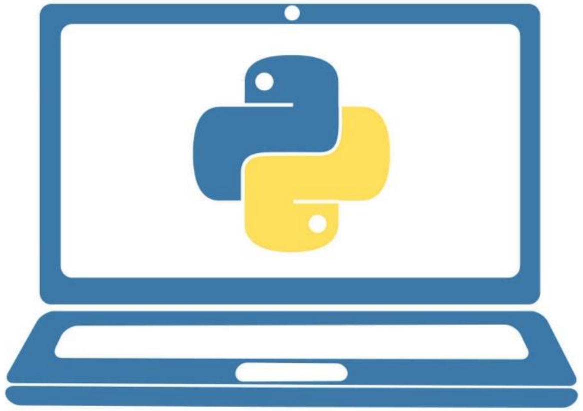

tkinter grid.py - /home/pi/PythonElectives/tkinter grid.py (3.5.3)
File Edit Format Run Options Window Help

```
from tkinter import *

top=Tk()

L1=Label(top,text='Height:')
L2=Label(top,text='Width:')

E1=Entry(top,width=5)
E2=Entry(top,width=5)

CB1=Checkbutton(top,text='Keep aspect')

I=PhotoImage(file='dreamland.png')
C=Canvas(top,width=200,height=125)
C.create_image(0,0,image=I,anchor=NW)

B1=Button(top,text='Zoom in',width=8)
B2=Button(top,text='Zoom out',width=8)

L1.grid(row=0,column=0) #labels
L2.grid(row=1,column=0)

E1.grid(row=0,column=1) #entries
E2.grid(row=1,column=1)

CB1.grid(columnspan=2) #check button

C.grid(row=0,column=2,rowspan=2,columnspan=2) #canvas

B1.grid(row=2,column=2) #buttons
B2.grid(row=2,column=3)

top.mainloop()
```

Width is in terms of number of characters.

THANK
YOU

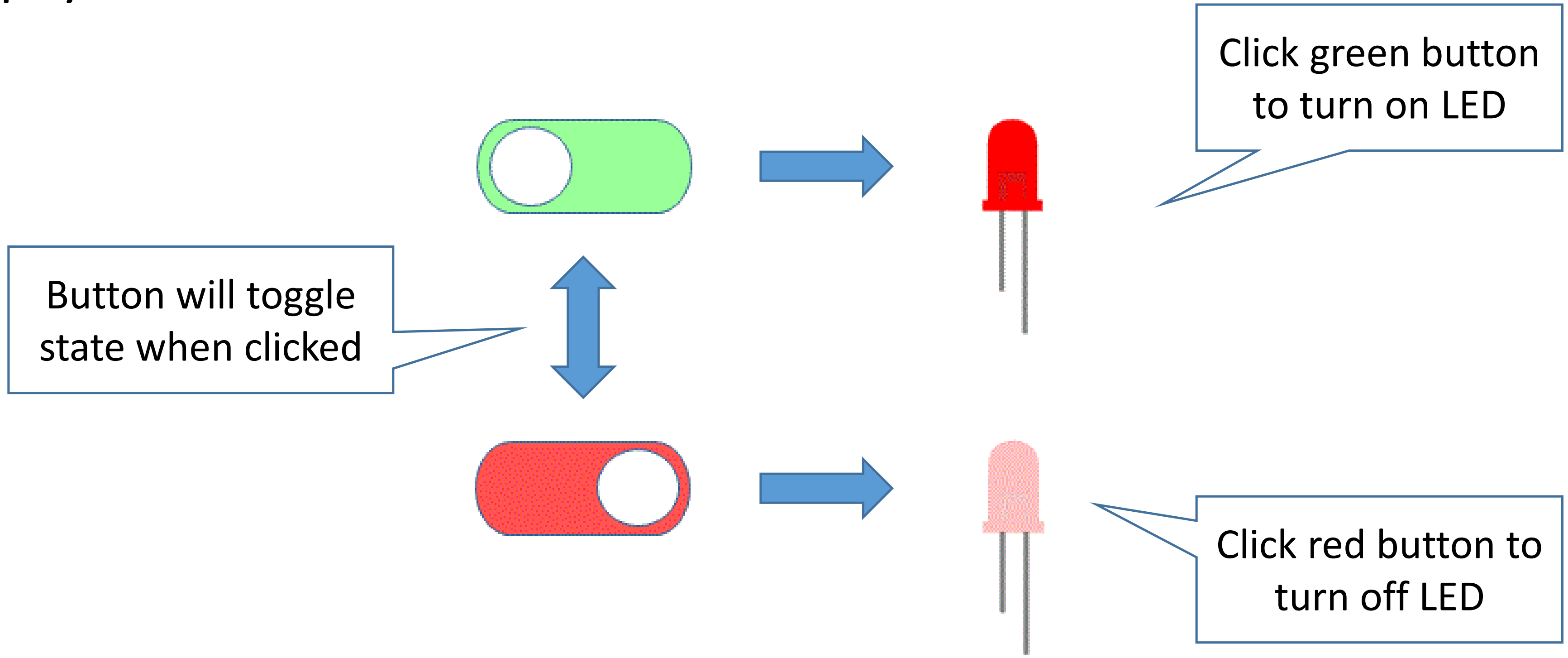
Lab Exercises



- Exercise 7.1 – Using virtual button to control physical LED
- Exercise 7.2 – Using physical button to control virtual LED
- Exercise 7.3 – After you…
- Exercise 7.4 – Dashboard

Exercise 7.1 – Using virtual button to control physical LED

Based on what you have learnt in the lecture, how can you use a virtual button to control a physical LED?



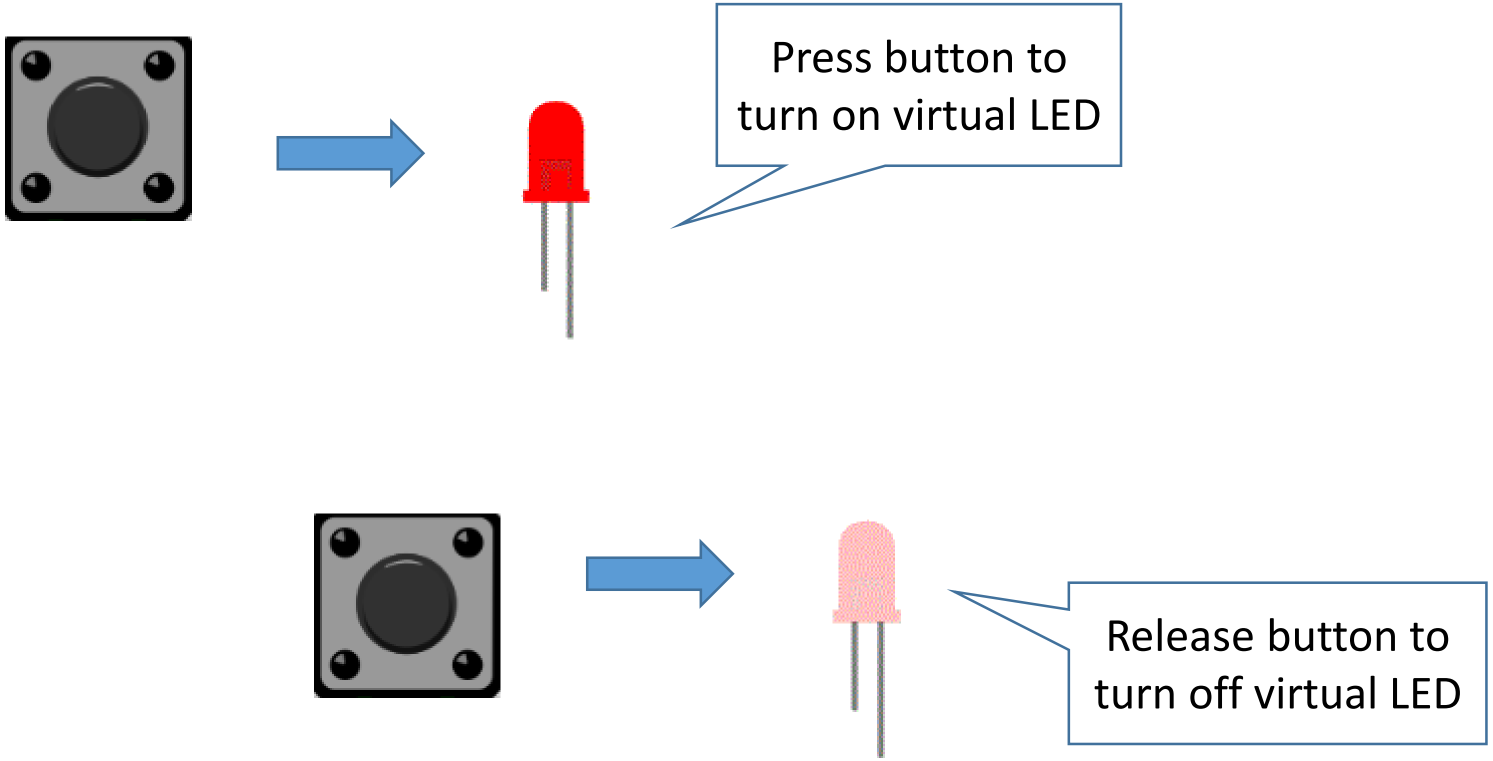
Click green button to turn on LED

Button will toggle state when clicked

Click red button to turn off LED

Exercise 7.2 – Using physical button to control virtual LED

Based on what you have learnt in the lecture, how can you use a physical button to control a virtual LED?

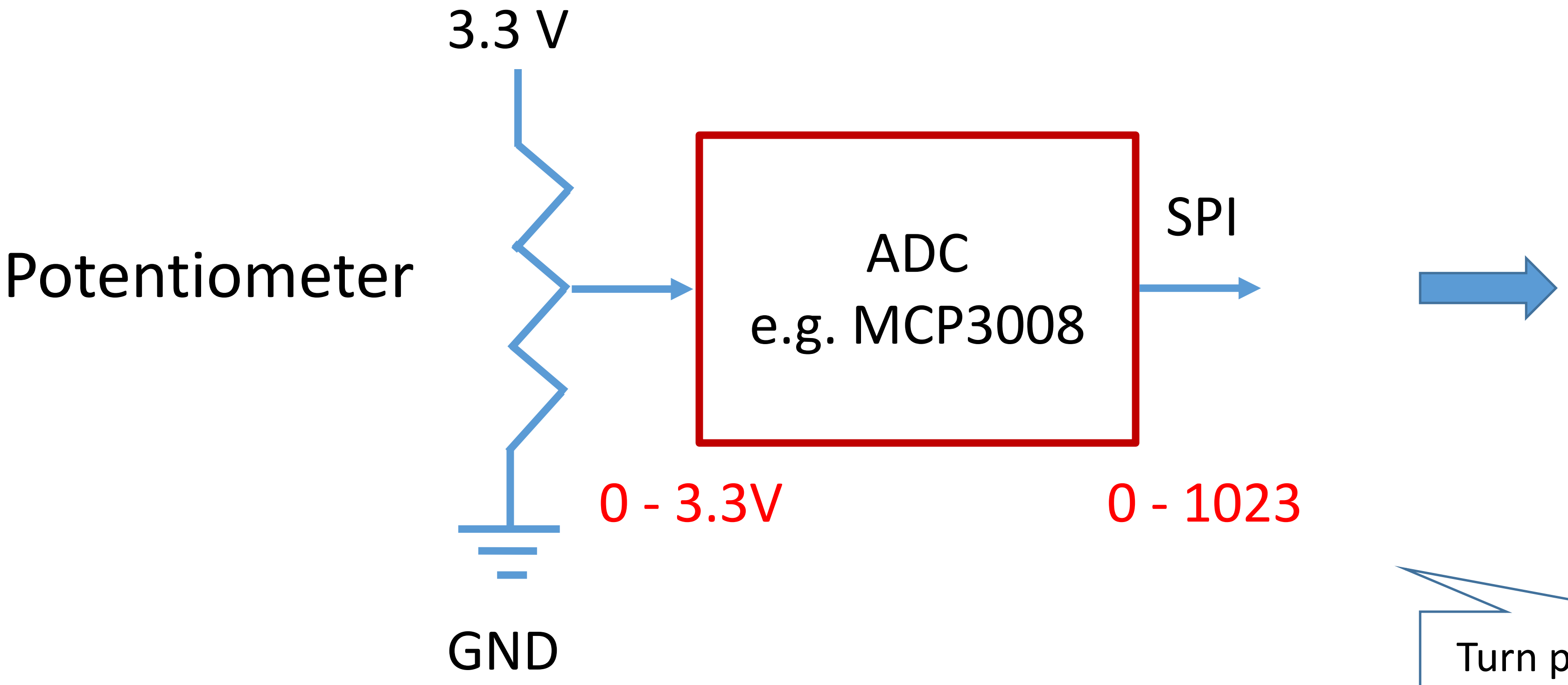


Press button to turn on virtual LED

Release button to turn off virtual LED

Exercise 7.3 – After you…

Create a virtual gauge to display the potentiometer reading (range 0-1023, which can be scaled to 0-100), and use “after” to update this display every second.



3.3 V

Potentiometer

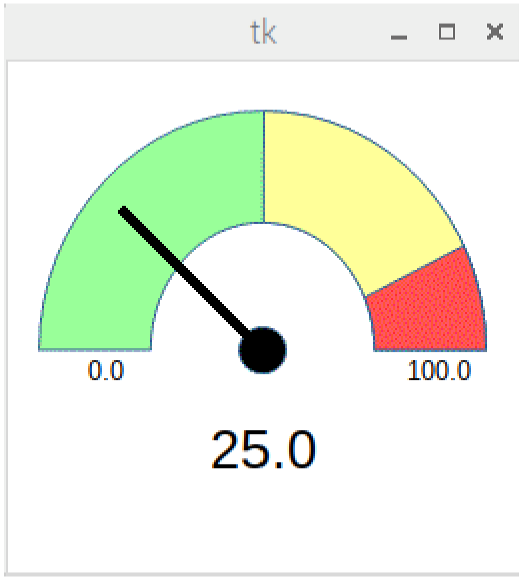
ADC
e.g. MCP3008

SPI

0 - 3.3V

0 - 1023

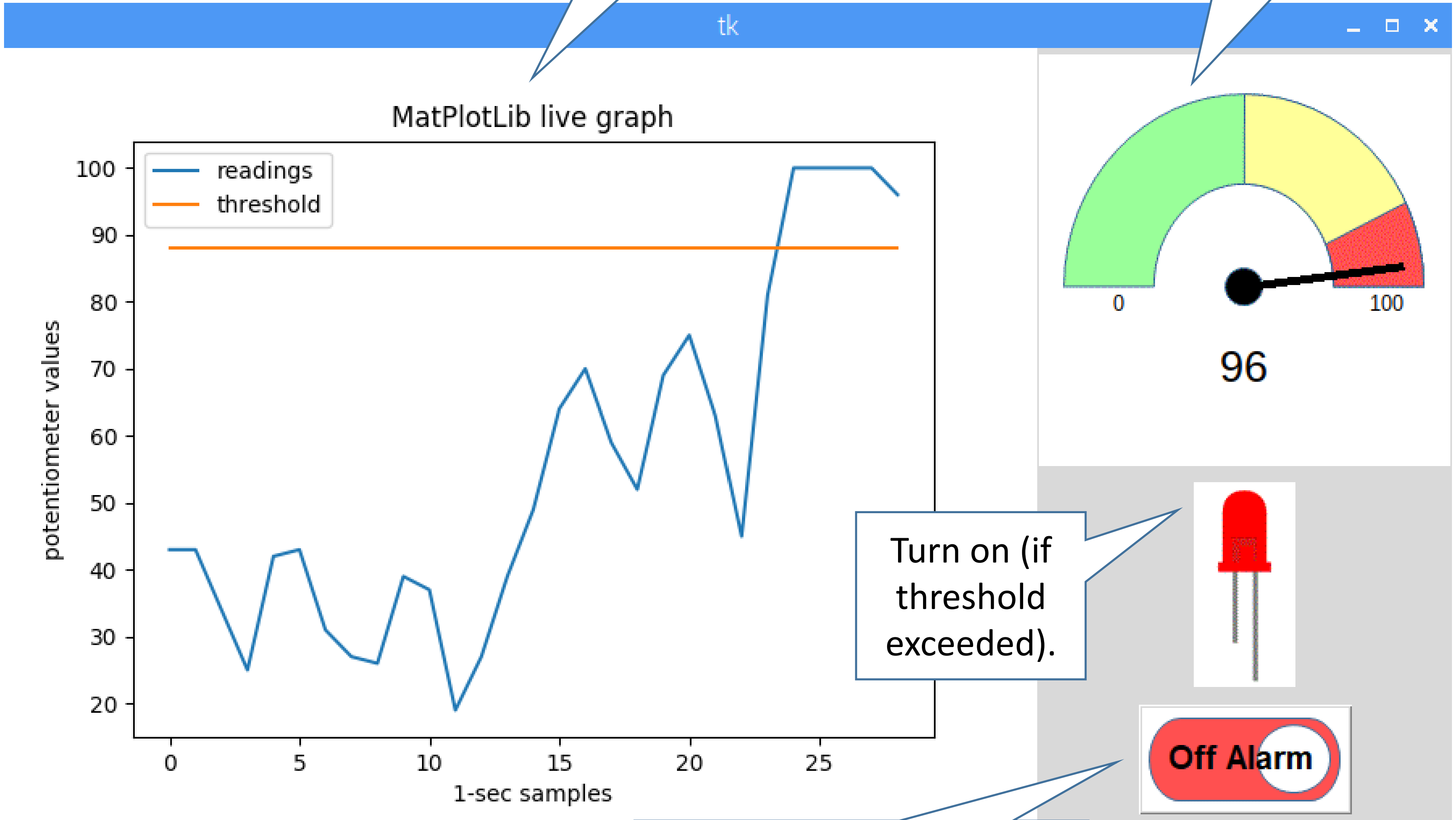
GND



Turn potentiometer, and needle should move.

Exercise 7.4 – Dashboard

Fill in the blanks in the Python program given (Elective1_Exercise 7_4.py) to use “grid” to arrange the widgets on the Dashboard:



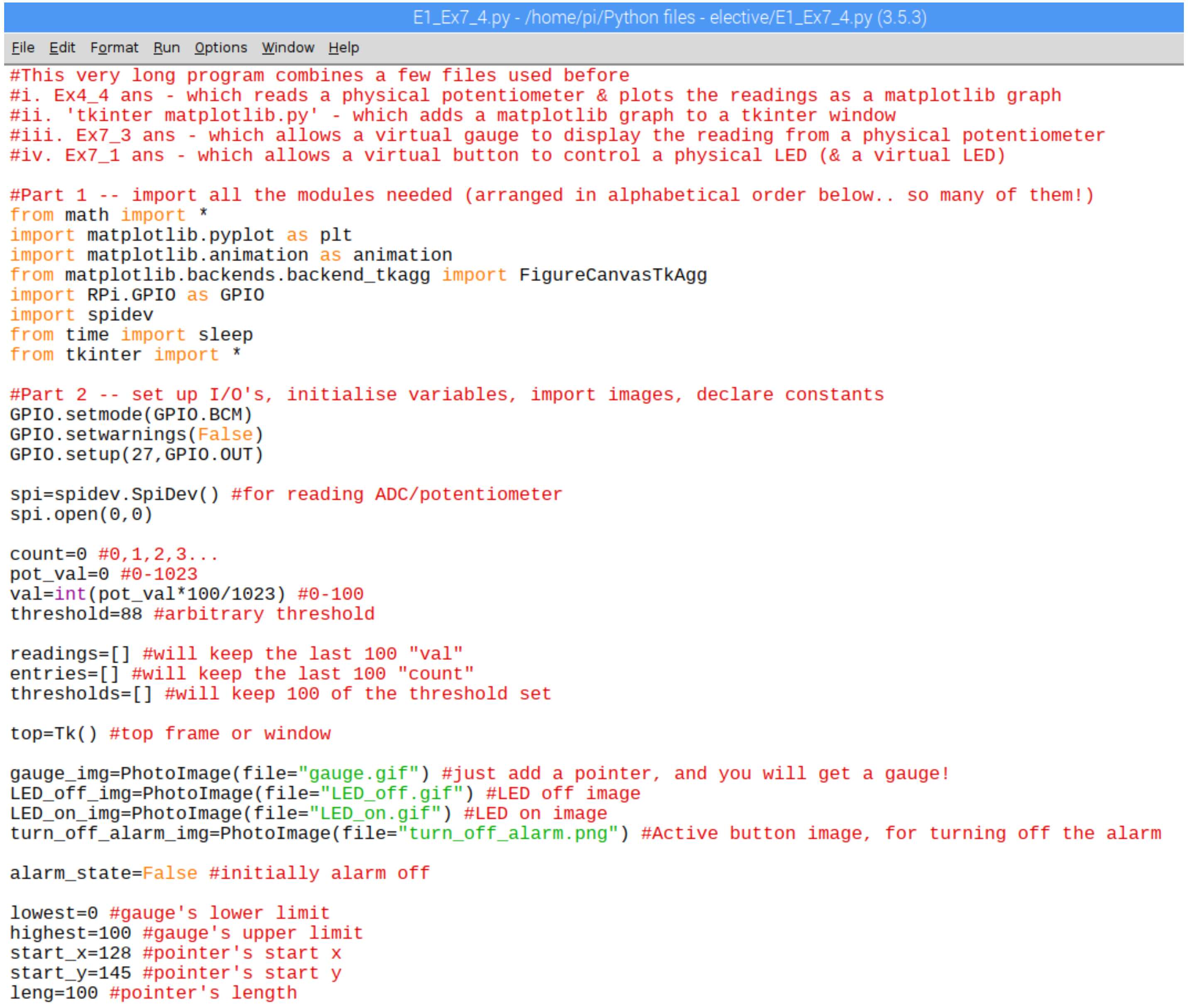
Historical values.

Instantaneous values.

Turn on (if threshold exceeded).

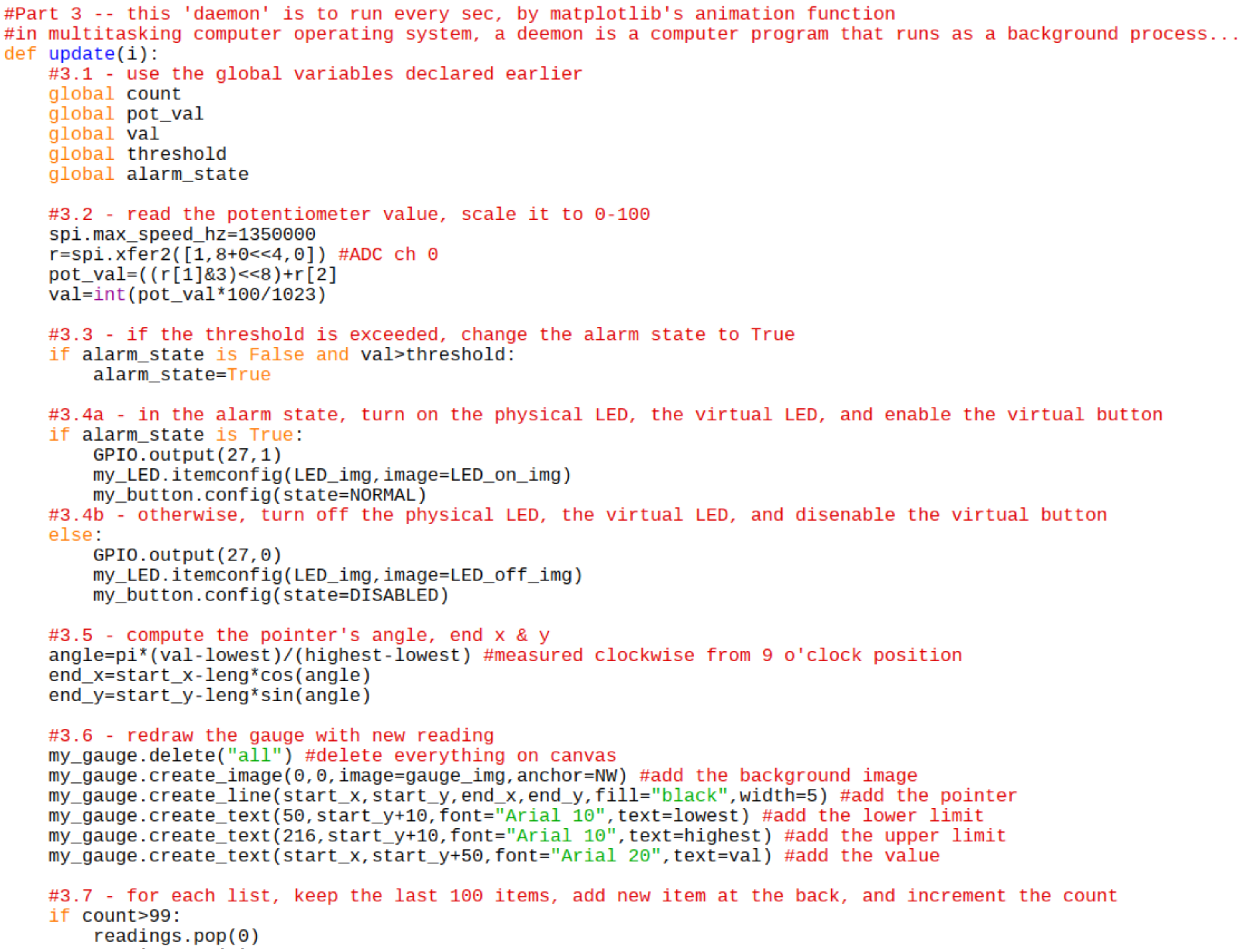
Click to turn off “alarm”.

Exercise 7.4 – Dashboard (cont.)


E1_Ex7_4.py - /home/pi/Python files - elective/E1_Ex7_4.py (3.5.3)
File Edit Format Run Options Window Help

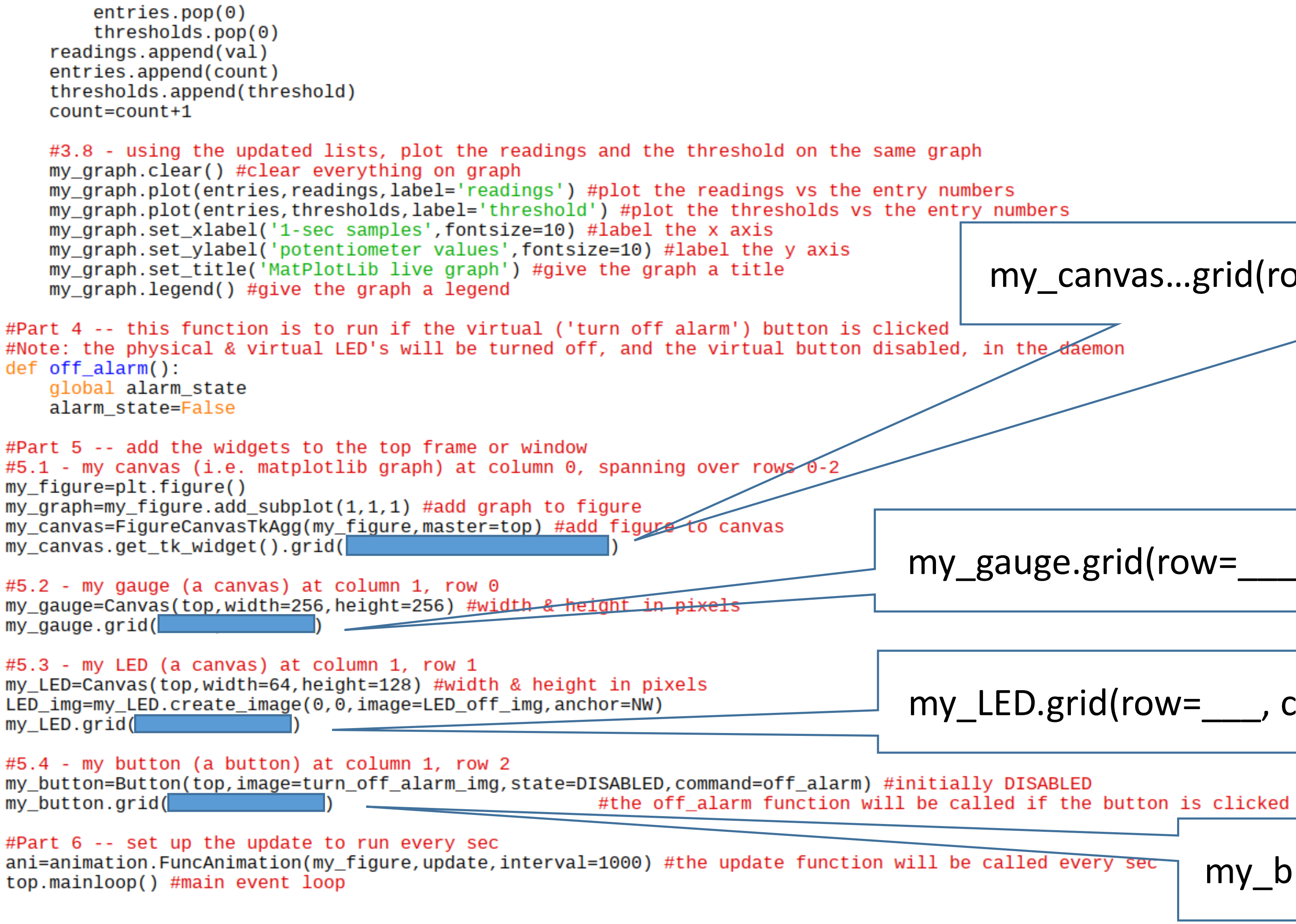
```
#This very long program combines a few files used before
#i. Ex4_4 ans - which reads a physical potentiometer & plots the readings as a matplotlib graph
#ii. 'tkinter matplotlib.py' - which adds a matplotlib graph to a tkinter window
#iii. Ex7_3 ans - which allows a virtual gauge to display the reading from a physical potentiometer
#iv. Ex7_1 ans - which allows a virtual button to control a physical LED (& a virtual LED)

#Part 1 -- import all the modules needed (arranged in alphabetical order below.. so many of them!)
from math import *
import matplotlib.pyplot as plt
import matplotlib.animation as animation
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
import RPi.GPIO as GPIO
import spidev
from time import sleep
from tkinter import *

#Part 2 -- set up I/O's, initialise variables, import images, declare constants
GPIO.setmode(GPIO.BCM)
GPIO.setwarnings(False)
GPIO.setup(27,GPIO.OUT)

spi=spidev.SpiDev() #for reading ADC/potentiometer
spi.open(0,0)

count=0 #0,1,2,3...
pot_val=0 #0-1023
val=int(pot_val*100/1023) #0-100
threshold=88 #arbitrary threshold

readings=[] #will keep the last 100 "val"
entries=[] #will keep the last 100 "count"
thresholds=[] #will keep 100 of the threshold set

top=Tk() #top frame or window

gauge_img=PhotoImage(file="gauge.gif") #just add a pointer, and you will get a gauge!
LED_off_img=PhotoImage(file="LED_off.gif") #LED off image
LED_on_img=PhotoImage(file="LED_on.gif") #LED on image
turn_off_alarm_img=PhotoImage(file="turn_off_alarm.png") #Active button image, for turning off the alarm

alarm_state=False #initially alarm off

lowest=0 #gauge's lower limit
highest=100 #gauge's upper limit
start_x=128 #pointer's start x
start_y=145 #pointer's start y
leng=100 #pointer's length
```

Exercise 7.4 – Dashboard (cont.)



```
#Part 3 -- this 'daemon' is to run every sec, by matplotlib's animation function
#in multitasking computer operating system, a deemon is a computer program that runs as a background process...
def update(i):
    #3.1 - use the global variables declared earlier
    global count
    global pot_val
    global val
    global threshold
    global alarm_state

    #3.2 - read the potentiometer value, scale it to 0-100
    spi.max_speed_hz=1350000
    r=spi.xfer2([1,8+0<<4,0]) #ADC ch 0
    pot_val=((r[1]&3)<<8)+r[2]
    val=int(pot_val*100/1023)

    #3.3 - if the threshold is exceeded, change the alarm state to True
    if alarm_state is False and val>threshold:
        alarm_state=True

    #3.4a - in the alarm state, turn on the physical LED, the virtual LED, and enable the virtual button
    if alarm_state is True:
        GPIO.output(27,1)
        my_LED.itemconfig(LED_img,image=LED_on_img)
        my_button.config(state=NORMAL)
    #3.4b - otherwise, turn off the physical LED, the virtual LED, and disenable the virtual button
    else:
        GPIO.output(27,0)
        my_LED.itemconfig(LED_img,image=LED_off_img)
        my_button.config(state=DISABLED)

    #3.5 - compute the pointer's angle, end x & y
    angle=pi*(val-lowest)/(highest-lowest) #measured clockwise from 9 o'clock position
    end_x=start_x-leng*cos(angle)
    end_y=start_y-leng*sin(angle)

    #3.6 - redraw the gauge with new reading
    my_gauge.delete("all") #delete everything on canvas
    my_gauge.create_image(0,0,image=gauge_img,anchor=NW) #add the background image
    my_gauge.create_line(start_x,start_y,end_x,end_y,fill="black",width=5) #add the pointer
    my_gauge.create_text(50,start_y+10,font="Arial 10",text=lowest) #add the lower limit
    my_gauge.create_text(216,start_y+10,font="Arial 10",text=highest) #add the upper limit
    my_gauge.create_text(start_x,start_y+50,font="Arial 20",text=val) #add the value

    #3.7 - for each list, keep the last 100 items, add new item at the back, and increment the count
    if count>99:
        readings.pop(0)
```

Exercise 7.4 – Dashboard (cont.)



```
        entries.pop(0)
        thresholds.pop(0)
    readings.append(val)
    entries.append(count)
    thresholds.append(threshold)
    count=count+1

    #3.8 - using the updated lists, plot the readings and the threshold on the same graph
    my_graph.clear() #clear everything on graph
    my_graph.plot(entries,readings,label='readings') #plot the readings vs the entry numbers
    my_graph.plot(entries,thresholds,label='threshold') #plot the thresholds vs the entry numbers
    my_graph.set_xlabel('1-sec samples',fontsize=10) #label the x axis
    my_graph.set_ylabel('potentiometer values',fontsize=10) #label the y axis
    my_graph.set_title('MatPlotLib live graph') #give the graph a title
    my_graph.legend() #give the graph a legend

#Part 4 -- this function is to run if the virtual ('turn off alarm') button is clicked
#Note: the physical & virtual LED's will be turned off, and the virtual button disabled, in the daemon
def off_alarm():
    global alarm_state
    alarm_state=False

#Part 5 -- add the widgets to the top frame or window
#5.1 - my canvas (i.e. matplotlib graph) at column 0, spanning over rows 0-2
my_figure=plt.figure()
my_graph=my_figure.add_subplot(1,1,1) #add graph to figure
my_canvas=FigureCanvasTkAgg(my_figure,master=top) #add figure to canvas
my_canvas.get_tk_widget().grid(                        )

#5.2 - my gauge (a canvas) at column 1, row 0
my_gauge=Canvas(top,width=256,height=256) #width & height in pixels
my_gauge.grid(              )

#5.3 - my LED (a canvas) at column 1, row 1
my_LED=Canvas(top,width=64,height=128) #width & height in pixels
LED_img=my_LED.create_image(0,0,image=LED_off_img,anchor=NW)
my_LED.grid(              )

#5.4 - my button (a button) at column 1, row 2
my_button=Button(top,image=turn_off_alarm_img,state=DISABLED,command=off_alarm) #initially DISABLED
my_button.grid(              )
                                        #the off_alarm function will be called if the button is clicked

#Part 6 -- set up the update to run every sec
ani=animation.FuncAnimation(my_figure,update,interval=1000) #the update function will be called every sec
top.mainloop() #main event loop
```

my_canvas…grid(row=___, column=___, rowspan=___)

my_gauge.grid(row=___, column=___)

my_LED.grid(row=___, column=___)

my_button.grid(row=___, column=___)

Appendix 1 – List of TKinter Widgets

Widgets	Applications
Button	to display buttons in your application
Canvas	to draw shapes, such as lines, ovals, polygons and rectangles, in your application.
Checkbutton	to display a number of options as checkboxes. The user can select multiple options at a time.
Entry	to display a single-line text field for accepting values from a user.
Frame	a container widget, to organize other widgets.
Label	to provide a single-line caption for other widgets. It can also contain images.
Listbox	to provide a list of options to a user.
Menubutton	to display menus in your application.
Menu	to provide various commands to a user. These commands are contained inside Menubutton.
Message	to display multiline text fields for accepting values from a user.
Radiobutton	to display a number of options as radio buttons. The user can select only one option at a time.

Appendix 1 – List of TKinter Widgets (cont.)

Widgets	Applications
Scale	to provide a slider widget.
Scrollbar	to add scrolling capability to various widgets, such as list boxes.
Text	to display text in multiple lines.
Toplevel	to provide a separate window container.
Spinbox	a variant of the standard Tkinter Entry widget, to select from a fixed number of values.
PanedWindow	a container widget that may contain any number of panes, arranged horizontally or vertically.
LabelFrame	a simple container widget, to act as a spacer or container for complex window layouts.
tkMessageBox	to display message boxes in your applications.

Appendix 1 – List of TKinter Widgets (cont.)

- These widgets can be organized into a few categories:

➢ The **containers**: frame, toplevel, paned window.

➢ The **buttons**: button, radiobutton, checkbutton (checkbox), menubutton (combobox).

➢ The **text widgets**: label, labelframe, message, text.

➢ The **entry widgets**: scale, scroll, listbox, slider, spinbox, entry (singleline), text (multiline), and canvas (vector and pixel graphics).

Appendix 2 – List of TKinter resources

- Recommended **online learning resources**:
  - ✓https://www.tutorialspoint.com/python/python_gui_programming.htm
  - ✓http://effbot.org/tkinterbook/grid.htm
  - ✓https://en.wikipedia.org/wiki/Tkinter

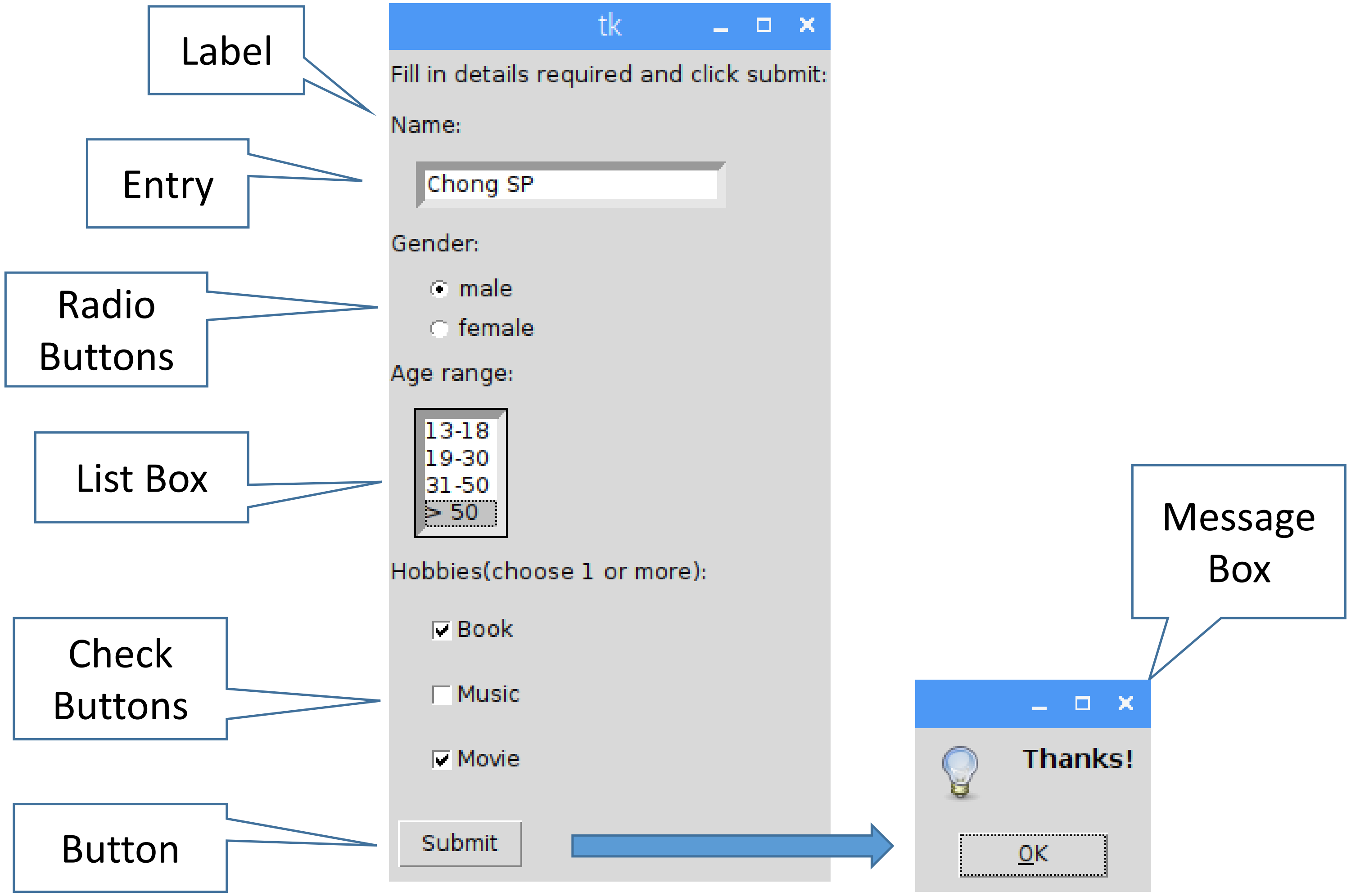
Appendix 3 – Adding I/O’s to your project

- You can use any one or a combination of these methods to add various I / O (input / output) devices to your RPi project:

  - ❑ use “**shield**” – snap a shield with the I / O devices needed to the RPi.
  - ❑ **wire things up** – using jumper wires & breadboard, connect the I / O devices to the RPi.
  - ❑ via **USB** – make use of I / O devices that have USB interface e.g. keyboard, mouse, webcam, finger print reader etc.
  - ❑ use **GUI** – use TK-inter to draw buttons (virtual digital inputs), sliders (virtual analogue inputs) LEDs (virtual digital outputs), gauges (virtual analogue outputs) and add Matplotlib graphs to display outputs over a period of time.

Appendix 4 – A survey form (using Entry, Radio Buttons, List Box, Check Buttons)

This simple survey form asks a person to fill in his or her name, gender, age range and hobbies.



Fill in details required and click submit:

Name:

Chong SP

Gender:

● male
○ female

Age range:

13-18
19-30
31-50
> 50

Hobbies(choose 1 or more):

☑ Book
□ Music
☑ Movie

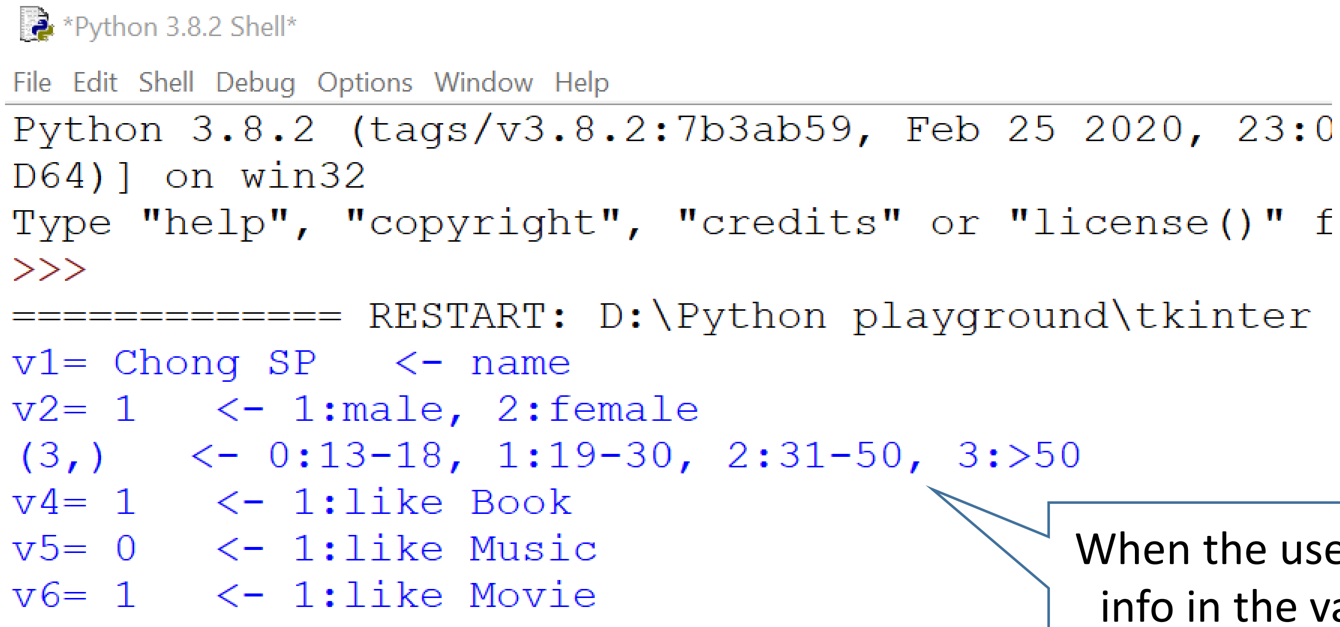
Submit

Thanks!

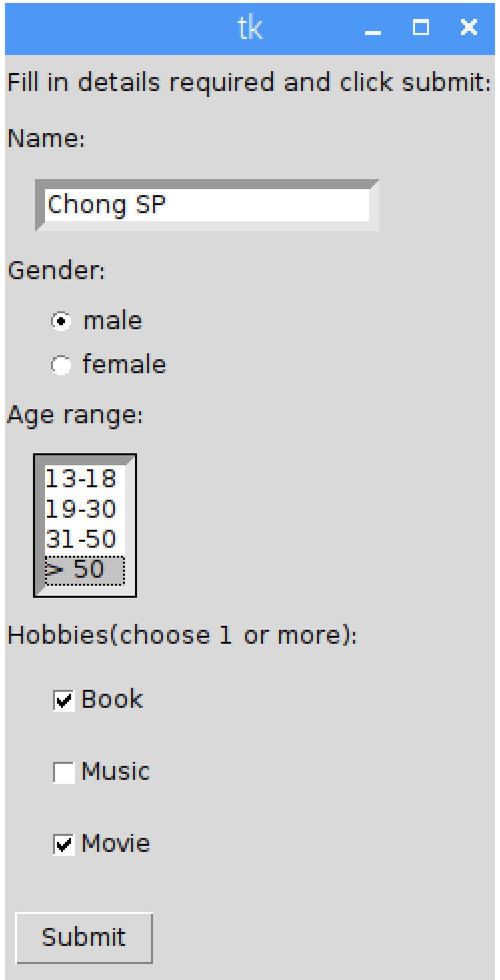
OK

Appendix 4 – A survey form (cont.)

- This is what you get when the program runs.



```
Python 3.8.2 (tags/v3.8.2:7b3ab59, Feb 25 2020, 23:0
D64)] on win32
Type "help", "copyright", "credits" or "license()" f
>>>
============= RESTART: D:\Python playground\tkinter
v1= Chong SP   <- name
v2= 1   <- 1:male, 2:female
(3,)   <- 0:13-18, 1:19-30, 2:31-50, 3:>50
v4= 1   <- 1:like Book
v5= 0   <- 1:like Music
v6= 1   <- 1:like Movie
```



Fill in details required and click submit:

Name:

Chong SP

Gender:

● male

○ female

Age range:

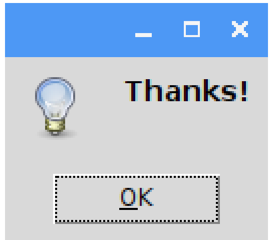
13-18
19-30
31-50
> 50

Hobbies(choose 1 or more):

☑ Book

□ Music

☑ Movie



Thanks!

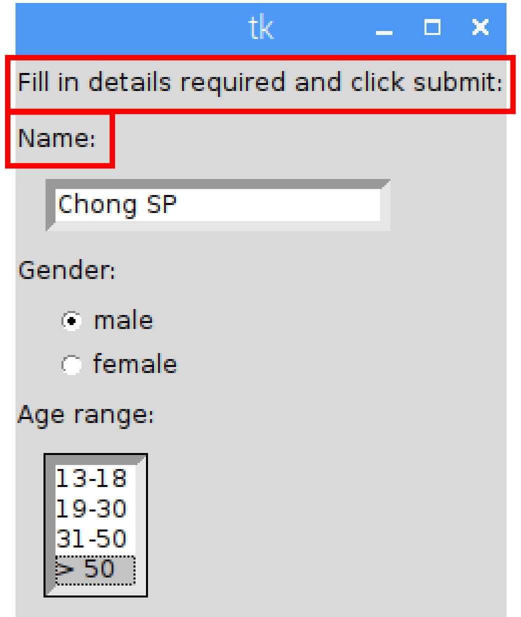
When the user clicks Submit, the info in the various fields in the survey form will be extracted.

The program is shown on the next pages.

Appendix 4 – A survey form (cont.)

- Labels…

Labels are used to display the boxed items.



Fill in details required and click submit:

Name:

Chong SP

Gender:

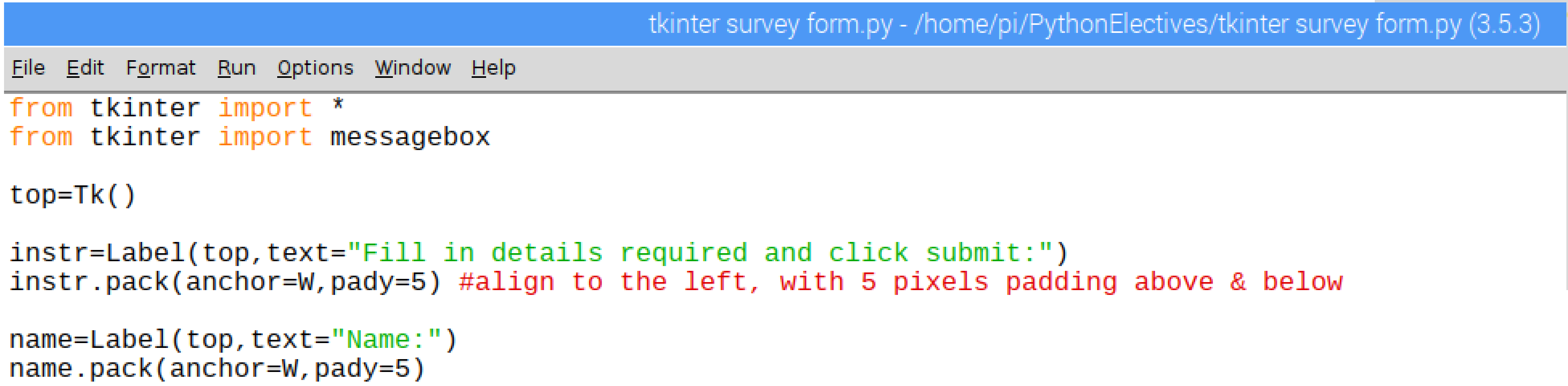
● male

○ female

Age range:

13-18
19-30
31-50
> 50

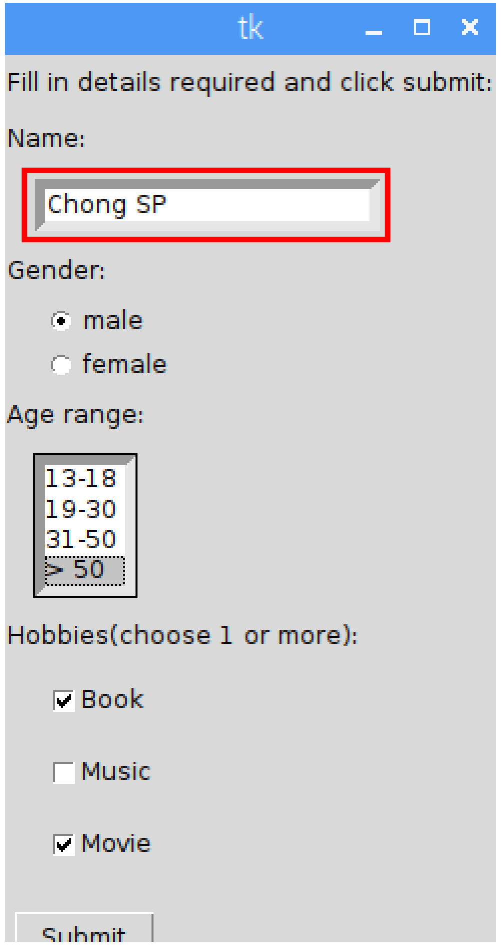
or more):


tkinter survey form.py - /home/pi/PythonElectives/tkinter survey form.py (3.5.3)
File Edit Format Run Options Window Help

```python
from tkinter import *
from tkinter import messagebox

top=Tk()

instr=Label(top,text="Fill in details required and click submit:")
instr.pack(anchor=W,pady=5) #align to the left, with 5 pixels padding above & below

name=Label(top,text="Name:")
name.pack(anchor=W,pady=5)
```

Appendix 4 – A survey form (cont.)

- Entry…



Fill in details required and click submit:

Name:

Chong SP

Gender:

● male

○ female

Age range:

13-18
19-30
31-50
> 50

Hobbies(choose 1 or more):

☑ Book

□ Music

☑ Movie

Entry is used to allow text entry.

The entry can be extracted by v1.get( )

```
v1=StringVar()
name_entry=Entry(top,bd=5,textvariable=v1) #5-pixel border, & the entry will be stored into v1, a string variable
name_entry.pack(anchor=W,padx=15,pady=5) #15-pixel indentation from left
```

Appendix 4 – A survey form (cont.)

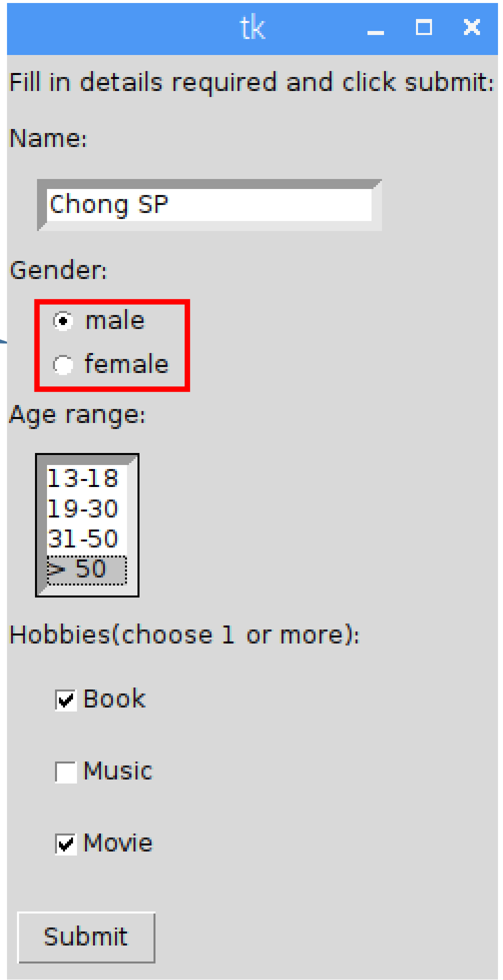
- Radio Buttons…

```
gender=Label(top,text="Gender:")
gender.pack(anchor=W,pady=5)

v2=IntVar()
male=Radiobutton(top,text="male",variable=v2,value=1) #v2=1 if male selected
male.pack(anchor=W,padx=15,pady=1)
female=Radiobutton(top,text="female",variable=v2,value=2) #v2=2 if female selected
female.pack(anchor=W,padx=15,pady=1)
```

Radio Buttons is used to allow selection (one choice out of several).

Note that each selection results in a different value, in the shared variable v2.

The selection can be extracted by v2.get( )



Fill in details required and click submit:

Name:

Chong SP

Gender:

● male

○ female

Age range:

13-18
19-30
31-50
> 50

Hobbies(choose 1 or more):

☑ Book

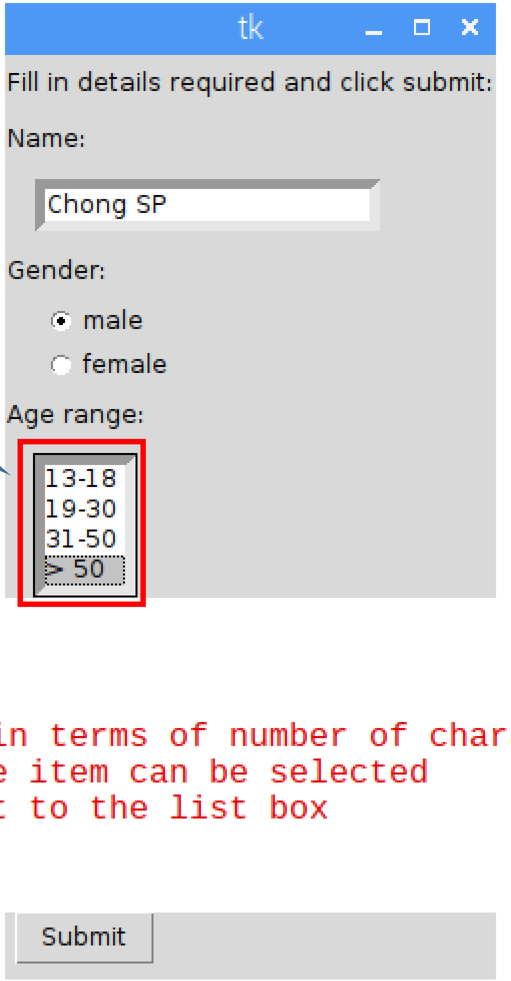
□ Music

☑ Movie

Appendix 4 – A survey form (cont.)

- List Box…

A List Box is used to allow selection (one or more choices out of several).



Fill in details required and click submit:

Name:

Chong SP

Gender:

● male

○ female

Age range:

13-18
19-30
31-50
> 50

```
age=Label(top,text="Age range:")
age.pack(anchor=W,pady=5).

age_listbox=Listbox(top,width=5,height=4,bd=5,selectmode=SINGLE) #the width & height are in terms of number of characters
                                                                    #SINGLE means only one item can be selected
for item in ["13-18","19-30","31-50","> 50"]: #use a for loop to add all items in the list to the list box
    age_listbox.insert(END,item)
age_listbox.pack(anchor=W,padx=15,pady=5)
```

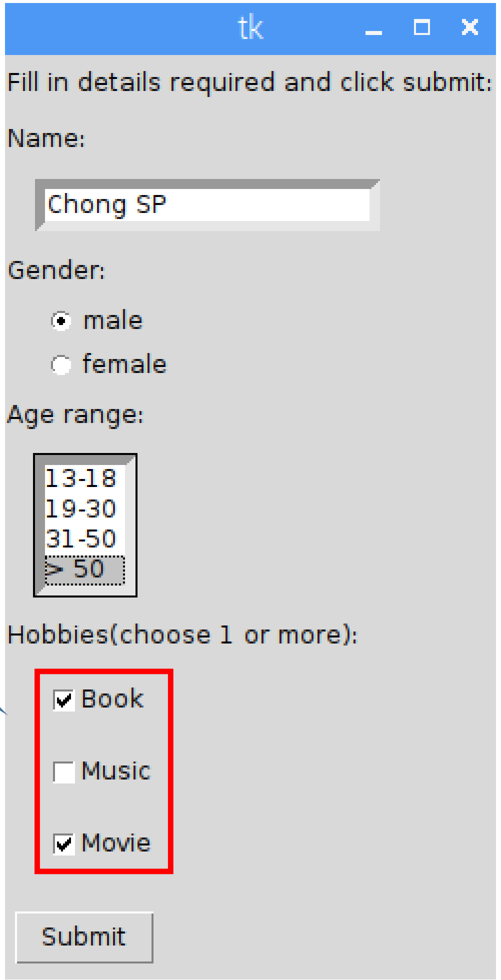
Items can be inserted into the List Box this way.

The selection can be extracted by age_listbox.curselection( ) - We will see this later.

This gives a tuple containing the line numbers of the selected items.

Appendix 4 – A survey form (cont.)

- Check Buttons…



tk

Fill in details required and click submit:

Name:

Chong SP

Gender:

● male

○ female

Age range:

13-18
19-30
31-50
> 50

Hobbies(choose 1 or more):

☑ Book

□ Music

☑ Movie

Check Buttons are used to allow selection (one or more choices out of several).

The selection can be extracted by v4.get( ) etc.

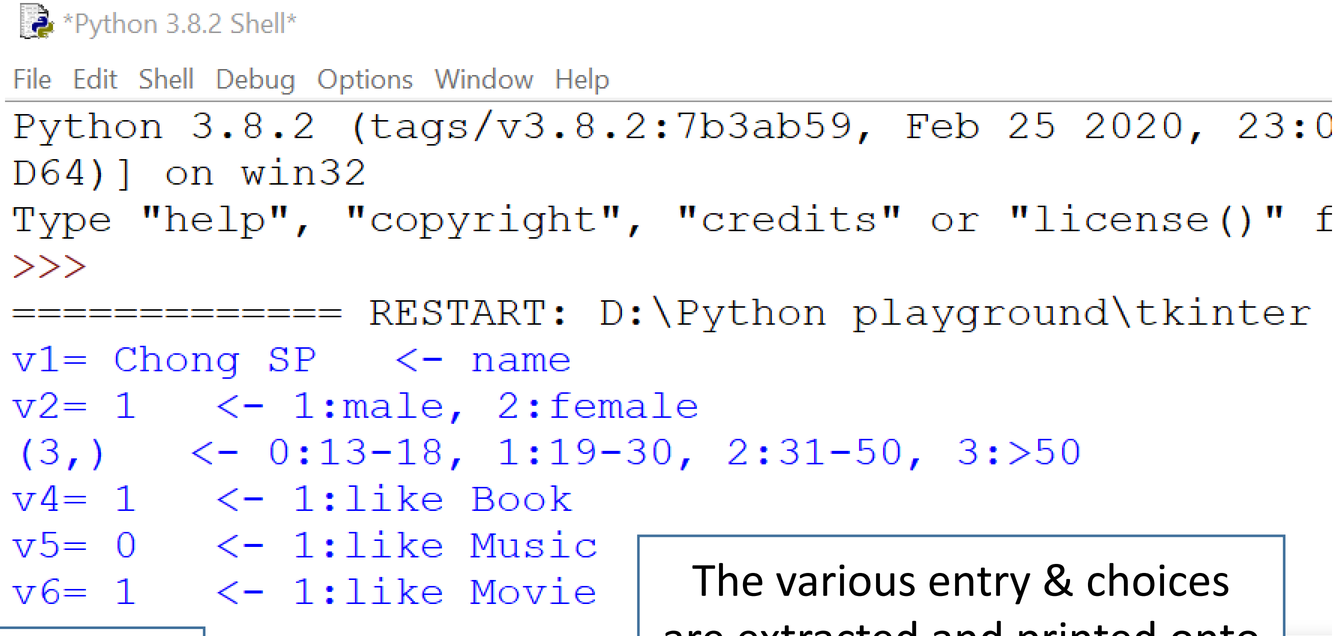
A “binary” variable is used for each selection.

If an item is selected, the variable takes on a certain value.

```
hobbies=Label(top,text="Hobbies(choose 1 or more):")
hobbies.pack(anchor=W,pady=5)

v4=IntVar()
v5=IntVar()
v6=IntVar()
book=Checkbutton(top,text="Book ",variable=v4,onvalue=1,offvalue=0,height=2,width=5) #v4=1 if "Book " selected, 0 otherwise
music=Checkbutton(top,text="Music",variable=v5,onvalue=1,offvalue=0,height=2,width=5) #similarly v5 for Music
movie=Checkbutton(top,text="Movie",variable=v6,onvalue=1,offvalue=0,height=2,width=5) #and v6 for Movie
book.pack(anchor=W,padx=15,pady=1)
music.pack(anchor=W,padx=15,pady=1)
movie.pack(anchor=W,padx=15,pady=1)
```

Appendix 4 – A survey form (cont.)

- Extracting values from form…

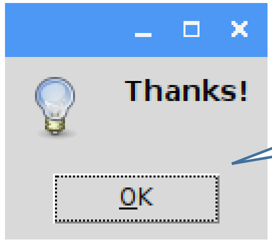


```
Python 3.8.2 (tags/v3.8.2:7b3ab59, Feb 25 2020, 23:0
D64)] on win32
Type "help", "copyright", "credits" or "license()" f
>>>
============== RESTART: D:\Python playground\tkinter
v1= Chong SP   <- name
v2= 1   <- 1:male, 2:female
(3,)   <- 0:13-18, 1:19-30, 2:31-50, 3:>50
v4= 1   <- 1:like Book
v5= 0   <- 1:like Music
v6= 1   <- 1:like Movie
```

The various entry & choices are extracted and printed onto the monitor by these lines.

When user clicks submit, the on_submit function is called.

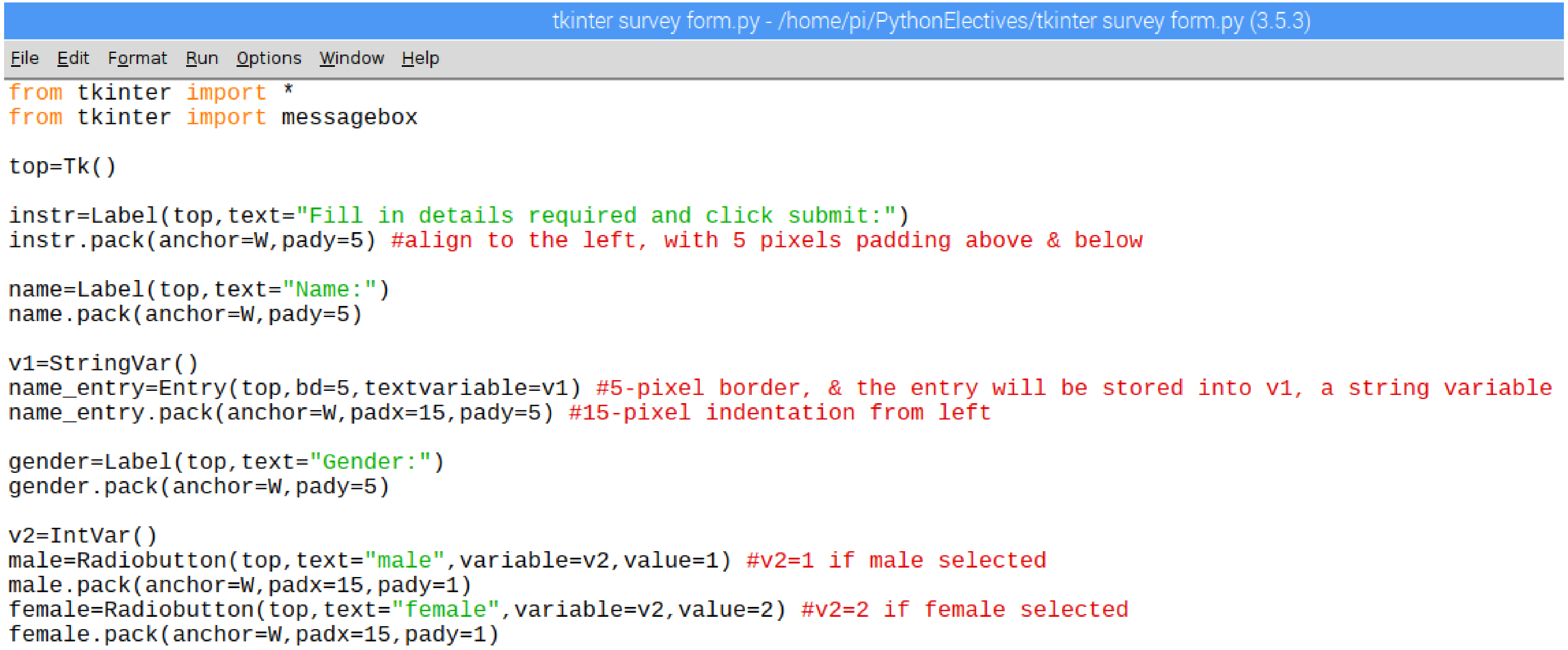
```
def on_submit():
    print("v1=",v1.get(),"  <- name") #the entry, radio buttons selection, list box selection, check buttons selection are displayed
    print("v2=",v2.get(),"  <- 1:male, 2:female") #get is used to get all the results, except for listbox which uses 'curselection'
    print(age_listbox.curselection(),"  <- 0:13-18, 1:19-30, 2:31-50, 3:>50")
    print("v4=",v4.get(),"  <- 1:like Book")
    print("v5=",v5.get(),"  <- 1:like Music")
    print("v6=",v6.get(),"  <- 1:like Movie")
    messagebox.showinfo("","Thanks!") #a message box to thank user for submitting the 'survey'

submit=Button(top,text="Submit",command=on_submit) #if the submit button is clicked, the on_submit function will execute
submit.pack(anchor=W,padx=5,pady=15)

top=mainloop()
```



The message box pops up as an acknowledgement.

Appendix 4 – A survey form (cont.)

- Code listing…

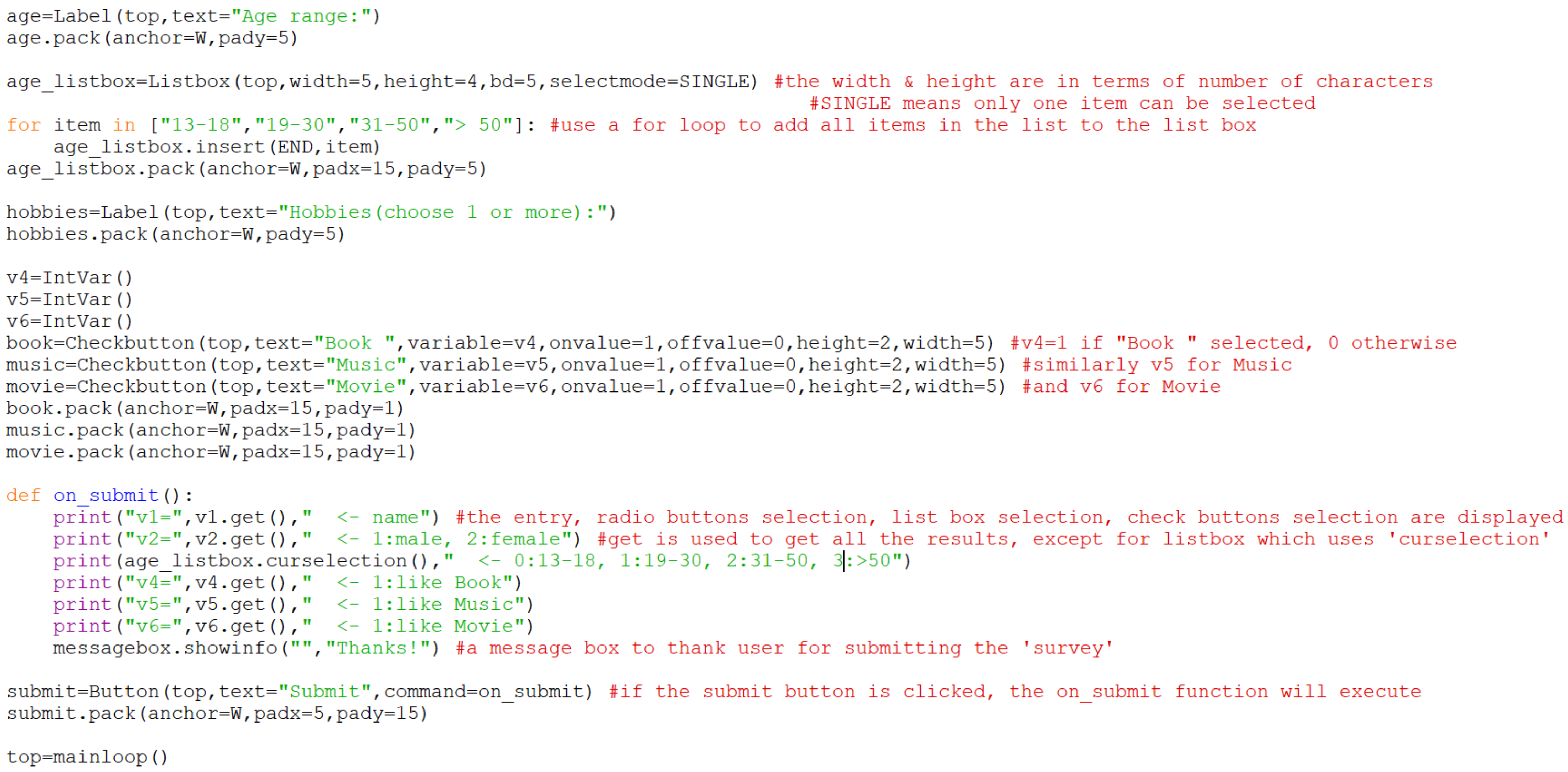


tkinter survey form.py - /home/pi/PythonElectives/tkinter survey form.py (3.5.3)
File Edit Format Run Options Window Help

```
from tkinter import *
from tkinter import messagebox

top=Tk()

instr=Label(top,text="Fill in details required and click submit:")
instr.pack(anchor=W,pady=5) #align to the left, with 5 pixels padding above & below

name=Label(top,text="Name:")
name.pack(anchor=W,pady=5)

v1=StringVar()
name_entry=Entry(top,bd=5,textvariable=v1) #5-pixel border, & the entry will be stored into v1, a string variable
name_entry.pack(anchor=W,padx=15,pady=5) #15-pixel indentation from left

gender=Label(top,text="Gender:")
gender.pack(anchor=W,pady=5)

v2=IntVar()
male=Radiobutton(top,text="male",variable=v2,value=1) #v2=1 if male selected
male.pack(anchor=W,padx=15,pady=1)
female=Radiobutton(top,text="female",variable=v2,value=2) #v2=2 if female selected
female.pack(anchor=W,padx=15,pady=1)
```

Appendix 4 – A survey form (cont.)

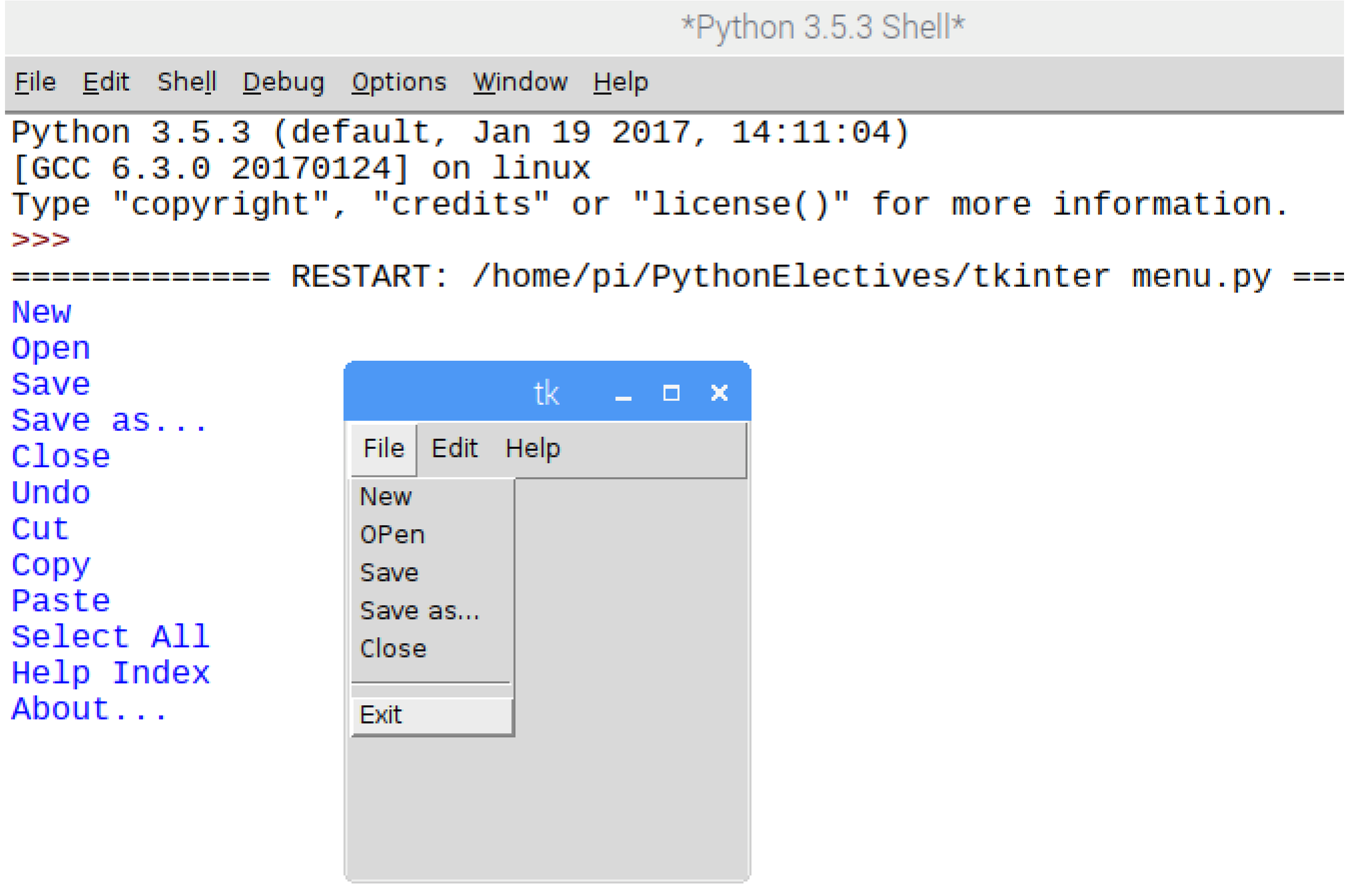
- Code listing…



```
age=Label(top,text="Age range:")
age.pack(anchor=W,pady=5)

age_listbox=Listbox(top,width=5,height=4,bd=5,selectmode=SINGLE) #the width & height are in terms of number of characters
                                                                    #SINGLE means only one item can be selected
for item in ["13-18","19-30","31-50","> 50"]: #use a for loop to add all items in the list to the list box
    age_listbox.insert(END,item)
age_listbox.pack(anchor=W,padx=15,pady=5)

hobbies=Label(top,text="Hobbies(choose 1 or more):")
hobbies.pack(anchor=W,pady=5)

v4=IntVar()
v5=IntVar()
v6=IntVar()
book=Checkbutton(top,text="Book ",variable=v4,onvalue=1,offvalue=0,height=2,width=5) #v4=1 if "Book " selected, 0 otherwise
music=Checkbutton(top,text="Music",variable=v5,onvalue=1,offvalue=0,height=2,width=5) #similarly v5 for Music
movie=Checkbutton(top,text="Movie",variable=v6,onvalue=1,offvalue=0,height=2,width=5) #and v6 for Movie
book.pack(anchor=W,padx=15,pady=1)
music.pack(anchor=W,padx=15,pady=1)
movie.pack(anchor=W,padx=15,pady=1)

def on_submit():
    print("v1=",v1.get()," <- name") #the entry, radio buttons selection, list box selection, check buttons selection are displayed
    print("v2=",v2.get()," <- 1:male, 2:female") #get is used to get all the results, except for listbox which uses 'curselection'
    print(age_listbox.curselection()," <- 0:13-18, 1:19-30, 2:31-50, 3:>50")
    print("v4=",v4.get()," <- 1:like Book")
    print("v5=",v5.get()," <- 1:like Music")
    print("v6=",v6.get()," <- 1:like Movie")
    messagebox.showinfo("","Thanks!") #a message box to thank user for submitting the 'survey'

submit=Button(top,text="Submit",command=on_submit) #if the submit button is clicked, the on_submit function will execute
submit.pack(anchor=W,padx=5,pady=15)

top=mainloop()
```

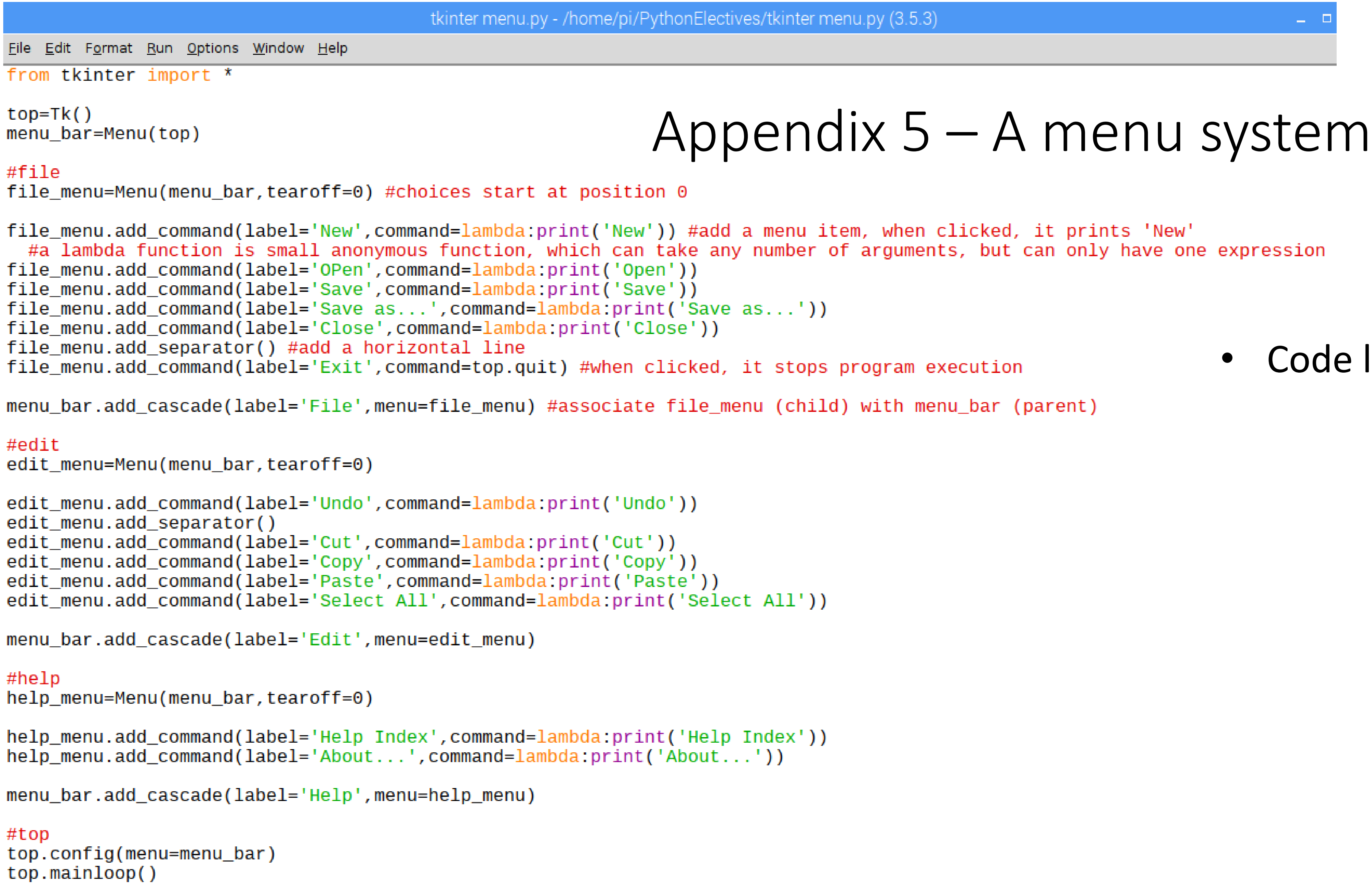
Appendix 5 – A menu system (using Menu)

- This is what you get when the program on the following slides runs.
- When you click File->New, 'New' will be printed onto the screen etc.
- The Edit menu has the menu items Undo, Cut, Copy, Paste and Select All.
- The Help menu has the menu items Help Index and About…



```
Python 3.5.3 (default, Jan 19 2017, 14:11:04)
[GCC 6.3.0 20170124] on linux
Type "copyright", "credits" or "license()" for more information.
>>>
============= RESTART: /home/pi/PythonElectives/tkinter menu.py ===
New
Open
Save
Save as...
Close
Undo
Cut
Copy
Paste
Select All
Help Index
About...
```

Appendix 5 – A menu system (cont.)


tkinter menu.py - /home/pi/PythonElectives/tkinter menu.py (3.5.3)
File Edit Format Run Options Window Help

```
from tkinter import *

top=Tk()
menu_bar=Menu(top)

#file
file_menu=Menu(menu_bar,tearoff=0) #choices start at position 0

file_menu.add_command(label='New',command=lambda:print('New')) #add a menu item, when clicked, it prints 'New'
  #a lambda function is small anonymous function, which can take any number of arguments, but can only have one expression
file_menu.add_command(label='OPen',command=lambda:print('Open'))
file_menu.add_command(label='Save',command=lambda:print('Save'))
file_menu.add_command(label='Save as...',command=lambda:print('Save as...'))
file_menu.add_command(label='Close',command=lambda:print('Close'))
file_menu.add_separator() #add a horizontal line
file_menu.add_command(label='Exit',command=top.quit) #when clicked, it stops program execution

menu_bar.add_cascade(label='File',menu=file_menu) #associate file_menu (child) with menu_bar (parent)

#edit
edit_menu=Menu(menu_bar,tearoff=0)

edit_menu.add_command(label='Undo',command=lambda:print('Undo'))
edit_menu.add_separator()
edit_menu.add_command(label='Cut',command=lambda:print('Cut'))
edit_menu.add_command(label='Copy',command=lambda:print('Copy'))
edit_menu.add_command(label='Paste',command=lambda:print('Paste'))
edit_menu.add_command(label='Select All',command=lambda:print('Select All'))

menu_bar.add_cascade(label='Edit',menu=edit_menu)

#help
help_menu=Menu(menu_bar,tearoff=0)

help_menu.add_command(label='Help Index',command=lambda:print('Help Index'))
help_menu.add_command(label='About...',command=lambda:print('About...'))

menu_bar.add_cascade(label='Help',menu=help_menu)

#top
top.config(menu=menu_bar)
top.mainloop()
```

- Code listing…