A SunCam online continuing education course

Python Programming for Engineers - Part 4: Graphical User Interfaces II

by

Kwabena Ofosu, Ph.D., P.E., PTOE

Computer Programming in *Python* – Part 4
A SunCam online continuing education course

Abstract

Python is a widely used, free, open source, high-level, general purpose computer programming language. *Python* drives some of the internet's most popular websites such as *Google*, *Youtube* and *Instagram*. *Python* can be used to perform complex mathematical calculations, handle big data, build web apps and desktop applications, and manage databases.

This course is the fourth of a series on *Python* programming. This course presents techniques to build graphical user interfaces (GUI) in *Python*. A GUI application or app is an interface that enables a user to interact with a computer program or an electronic device, in certain designed ways, through visual indications and graphical elements. This course presents the details of *Python tkinter* widgets used to build *Python* GUI applications such as messageboxes, slider and scrollbar widgets, as well as widgets for creating additional GUI windows and widgets for organizing other widgets on sub panes of the main GUI window. This course also presents specialized widgets from other *Python* and *tkinter* modules and function libraries such as comboboxes and dialog widgets. This course is tailored to practicing engineers. Practical examples from situations encountered by practicing engineers and scientists are used to illustrate and demonstrate the concepts and methods learned in this course.

On completion of this course, participants will be capable of applying the methods and techniques learned in a desktop application that can be used to manage large data sets and automate complex, repetitive, and tedious engineering calculations and algorithms. Participants will be able to identify professional situations in which programming will be of a strategic advantage to them in their fields of specialty, and to their organizations. Programming continues to be an increasingly relevant and advantageous skill for engineers competing in a global marketplace in the computer age.

There are no required pre-requisites for this course. However, it will be helpful to understand the fundamentals of the *Python* programming language in general, as presented in the earlier parts of this course series.

TABLE OF CONTENTS

List of Figures

List of Tables

Computer Programming in *Python* – Part 3
A SunCam online continuing education course

1. INTRODUCTION

1.1 Python

Python is an interpreted, high-level, general purpose computer programming language. *Python* is easy to use and is increasingly popular for beginners as well as seasoned programmers.

Python can be used to perform complex mathematical and engineering calculations, and to handle big data. *Python* can be used for building GUIs and desktop applications. *Python* can be used on a server for web development and to build web apps. *Python* can be used to connect to database systems and can read and modify files. Since *Python* runs on an interpreter system, the code is executed rapidly which enables quick prototyping or production-ready software development.

As a high-level language, *Python* has a simpler syntax similar to the English language. The syntax of *Python* enables code to be written with fewer lines than some other programming languages. *Python* is increasingly popular as a first programming language for beginners.

As a result of its user-friendliness and versatility, *Python* is the programming language driving some of the internet's most popular websites, namely:

- *Google*
- *Youtube*
- *Quora*
- *Dropbox*
- *Yahoo!*
- *Yahoo Maps*
- *Reddit*
- *Bitly*
- *Instagram*
- *Spotify*
- *SurveyMonkey*
- *Pintrest*
- *Eventbrite*
- *Firefox*
- *and many others*

1.2 Graphical User Interface (GUI)

A graphical user interface or **GUI** (pronounced goo-ee) is an interface that enables a user to interact with a computer program or an electronic device through visual indications and graphical elements (also called **objects** or **controls**) such as a click button, checkbox, textbox, drop-down menu, image, scrollbar, animation etc., etc.

An example of a GUI is shown in Figure 1.1.

Prior to the invention of GUIs, interaction with a computer was by text-based commands whereby a user would type instructions into a command line.

A GUI provides a computer environment that is simple and easy to use, thus enabling significantly higher productivity and accessibility even for an untrained user. A well-designed GUI will enable a non-expert user to navigate through the system with ease, and the user does not have to know or memorize any special codes or commands whatsoever. All user interaction with the GUI is through a human interface device such as a keyboard, mouse, touchscreen etc.

1.3 *Python* GUIs

Among the many attractive features of *Python* are the options to develop GUIs. It can be argued that without the capability to build GUIs, *Python* may never have reached the level of popularity it has attained to date, and we may have never heard of YouTube, Instagram and other popular sites and applications driven by *Python*.

Python GUIs are built from modules (or function libraries) that ship with *Python* or may be downloaded for free. Some of the more popular packages include:

tkinter : This is an interface to the *tk* GUI toolkit that ships with *Python*.
wxPython : This is an open-source interface for *wxWindows*.
JPython : This is a port for *Java* which gives scripts written in *Python* seamless access to the *Java* GUI capabilities on your local machine.

In this course series, all *Python* GUIs shall be developed using *tkinter*.

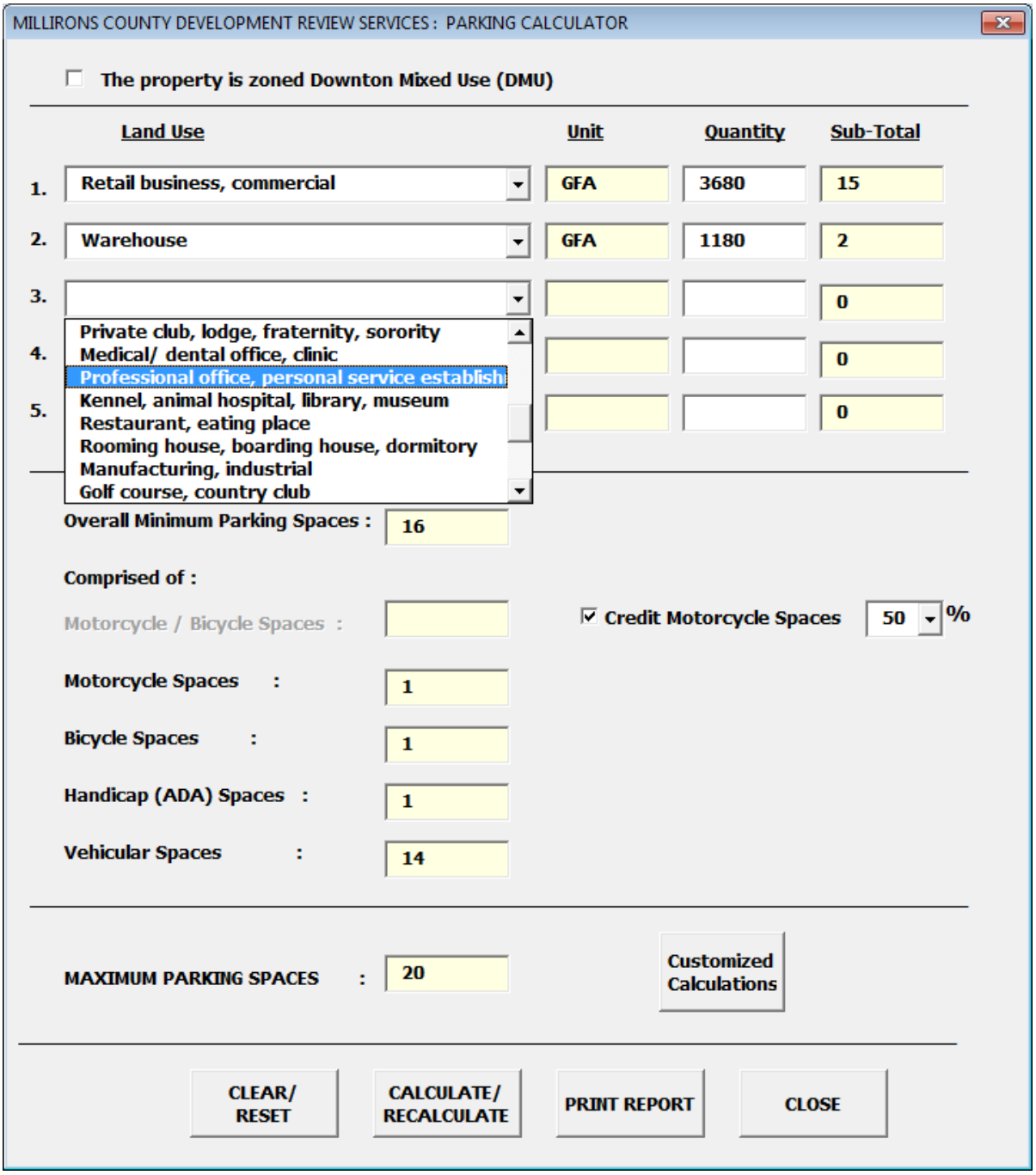


Figure 1. 1: A graphical user interface (GUI)

2. *PYTHON TKINTER*

2.1 *tkinter*

tkinter (pronounced tee-kay-inter) is a built-in module that contains functions and methods for creating *Python* GUIs. The name *tkinter* derives from "tk interface", the interface to the tk GUI tools.

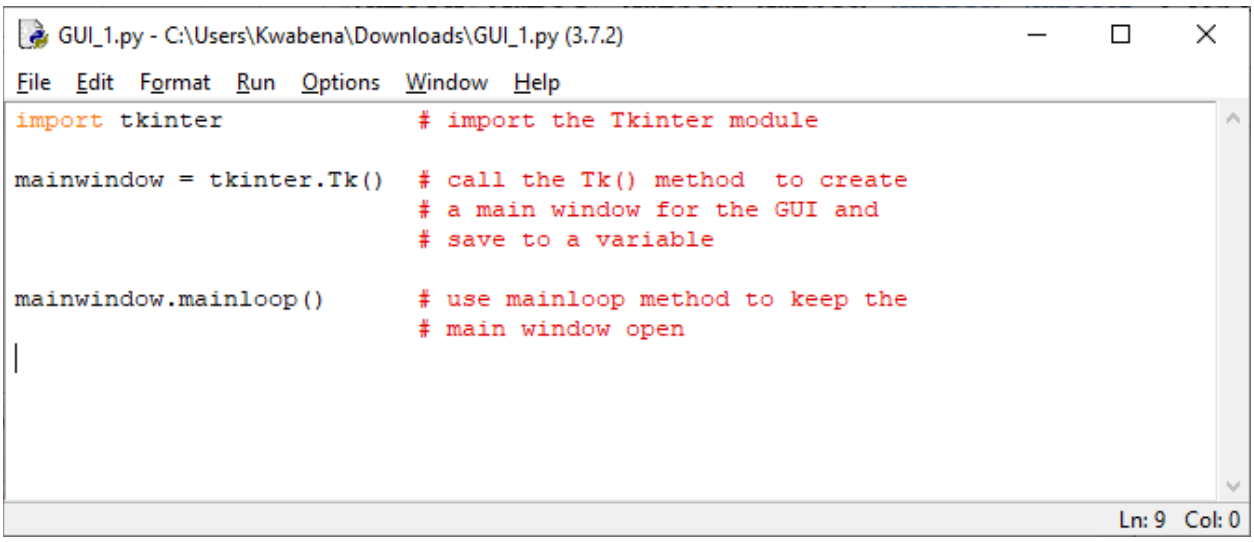
The general steps to create a *Python* GUI using *tkinter* are as follows:

1. import the *tkinter* module
2. create the main window of the GUI
3. add objects (or controls) - click button, checkboxes, scrollbars etc., etc., to the main window of the GUI, as needed
4. insert the code for the main window into a loop that keeps the main window up and available

(Note: Throughout this course, it cannot be over-emphasized that when typing or replicating the *Python* codes please remember to pay attention to spacing, alignment, indentations, margins etc. Remember that *Python* commands are case-sensitive. When modifying existing scripts, please pay particular attention to where exactly within the script the new codes and commands are being inserted and follow suit accordingly.)

Open a new session of IDLE (Python GUI).
Click on **File**.
Click on **New File**, to open the File Editor.

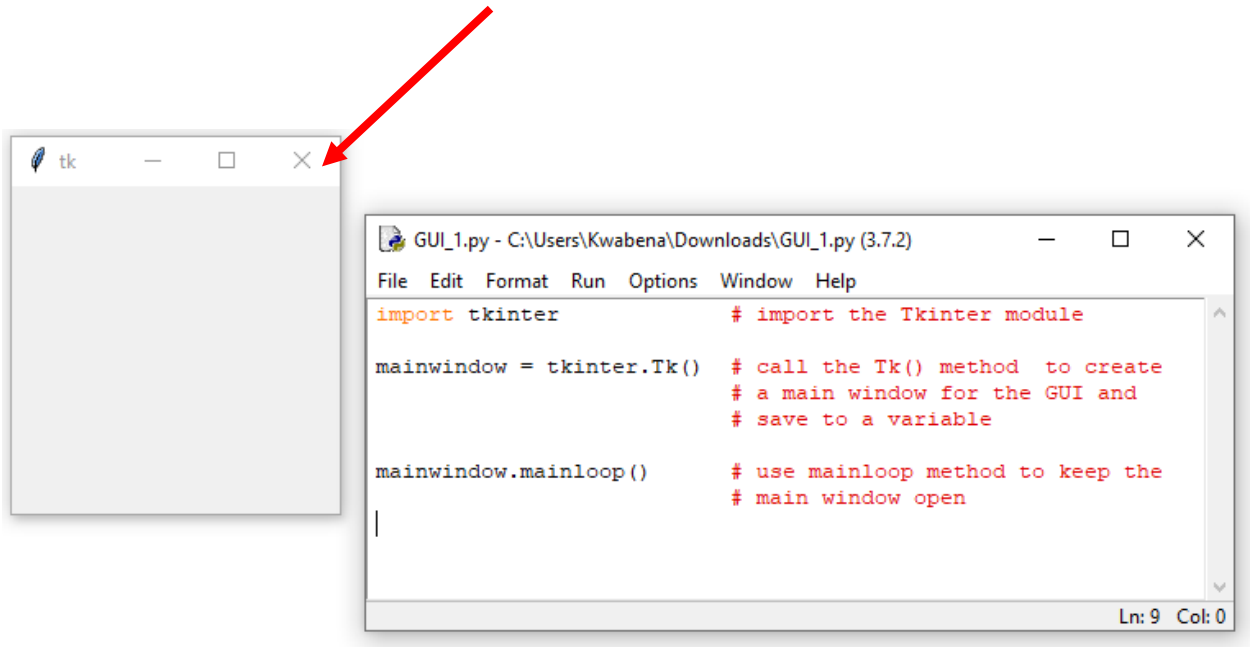
Replicate the following code.



(Note: For Python 2 users, the call is *Tkinter*, whereas for Python 3 and above users the call is *tkinter*.)

Save the file.
Run the file.

Look around your monitor display and locate the GUI window.
(You may have to minimize some other open applications or drag them out of the way to see the *Tk* window).

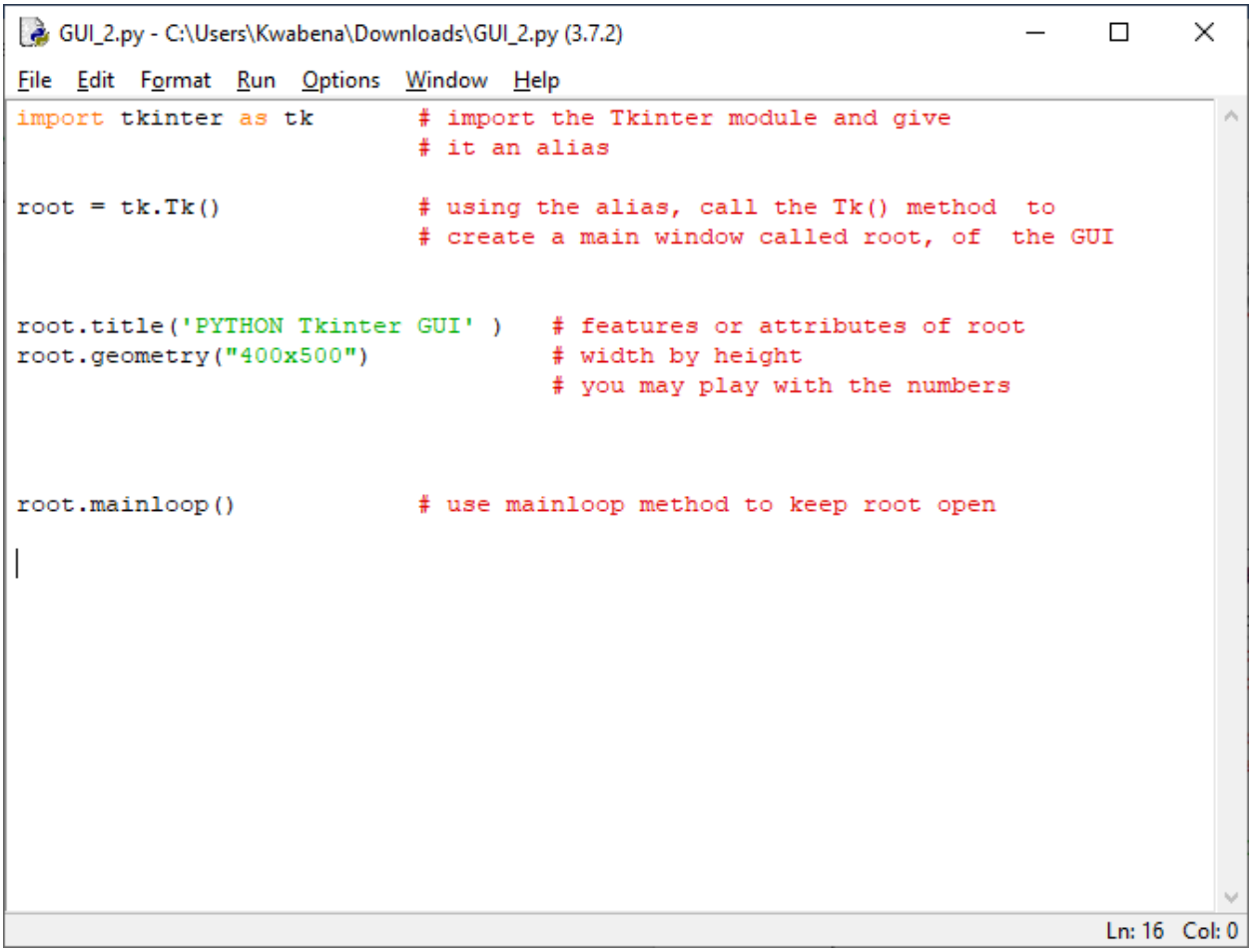


Success. You have created your first *Python* GUI.

Note that without the *mainloop( )* method, the window would show but then disappear. The *mainloop( )* method "reopens" the window continuously, obviously at speeds faster than the human eye can perceive, and this continues infinitely or until the user clicks on the "X" on the window to terminate the *mainloop( )*.

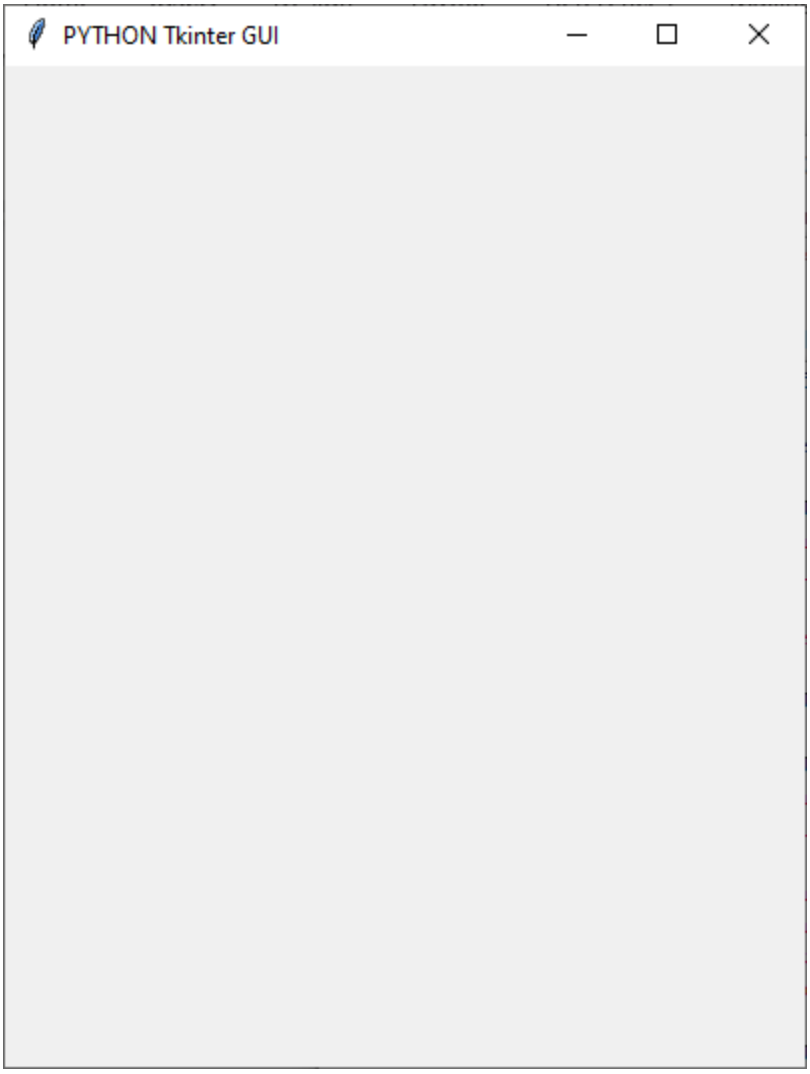
We shall re-write the code incorporating popular naming conventions and common strategies to streamline the code. We shall also modify some features or attributes of the GUI window.

Replicate the following.



Save the file.
Run the file.

We obtain the following titled and geometrically sized GUI window.



Later in this series, we shall look at other ways that attributes can be added or modified. Obviously, the next question is how to add controls – buttons, text, textboxes, checkboxes, scrollbars etc., etc., to the root window.

In *Python*, a control (or object) is called a **widget**.

2.2 *tkinter* Widgets

The widgets currently available are summarized in Table 2.1.

Table 2. 1: *tkinter* Widgets

Widget	Description
Label	used to implement a single line of text, can contain an image
Text	used to implement multiline text
Entry	a single line text field that accepts values from the user
Button	a button that is clicked on to "fire" some instructions and commands
Checkbutton	displays several options each with a checkbox, the user may select multiple options
Radiobutton	displays several options each with a radio button, the user may select one option only, to the exclusion of the other options
Listbox	provides a list of options to the user
Spinbox	used for data entry but data values must be selected from a fixed list of values
Menubutton	displays menus in the application GUI
Menu	provides commands that are contained inside a Menubutton, the user selects a command to implement
Message	a multiline text field that accepts values from the user
Frame	a "container" widget used to organize a group of other widgets
LabelFrame	a widget used as a spacer or container for complex window layouts
Canvas	used to draw shapes in GUI, such as lines, polygons, ellipses etc.
tkMessageBox	used to display message boxes (or popup boxes)
Scale	used to provide a slider widget
Scrollbar	provides scrolling capability within some other widget, e.g. scrolling through a Listbox
Toplevel	used to implement a separate window container
PanedWindow	a "container" widget that holds an array of panes

Each widget has properties (or attributes) called **options** that can be manipulated. Some of the commonly manipulated options include the

- color
- font
- dimensions
- relief
- anchors
- bitmaps
- cursors
- and many others

The following *tkinter* widgets shall be discussed in this course:

- tkMessageBox
- Scale
- Scrollbar
- Toplevel
- PanedWindow

The other *tkinter* widgets are presented in a previous part of this course series.

Other widgets from other *Python* and *tkinter* modules and function libraries discussed in this course are:

- Combobox
- simpledialog
- tkFileDialog

3. THE tkMESSAGEBOX WIDGET

3.1 tkMessageBox

The tkMessageBox is used to display a message box (or pop up box) to the user.

The syntax is of the form,

tkMessageBox.FunctionName (< title > , < massage > [, < options >])

where
FunctionName is the specific type of message box
< *title* > is the title caption displayed in the header bar of the widget
< *message* > is the main text message displayed on the message box
< *options* > optional attributes that are set to customize the message box

As with any kind of pop up box, a tkMessageBox is typically called via a function in the main program if the user interacts with the program in a certain way.

3.2 FunctionName

This is the type of the message box. When the message box opens, an icon of the relevant type is displayed on the message box. The available options are summarized in Table 3.1

Table 3. 1: tkMessageBox FunctionName functions

FunctionName	Description
showinfo( )	places an information icon on the message box
showwarning( )	places an exclamation sign icon on the message box
showerror( )	places a critical error icon on the message box
askquestion( )	places a question mark icon on the message box
askokcancel( )	question mark icon with OK button and Cancel button, clicking on the OK button returns a value of True, whereas clicking the Cancel button returns a value of None
askyesno( )	question mark icon with Yes button and No button, clicking on the Yes button returns a value of True, whereas clicking the No button returns a value of False
askretrycancel( )	question mark icon with Yes button, No button, and Cancel button, clicking on the Yes button returns a value of True, clicking the No button returns a value of False, clicking the Cancel button returns a value of None

3.3 Options

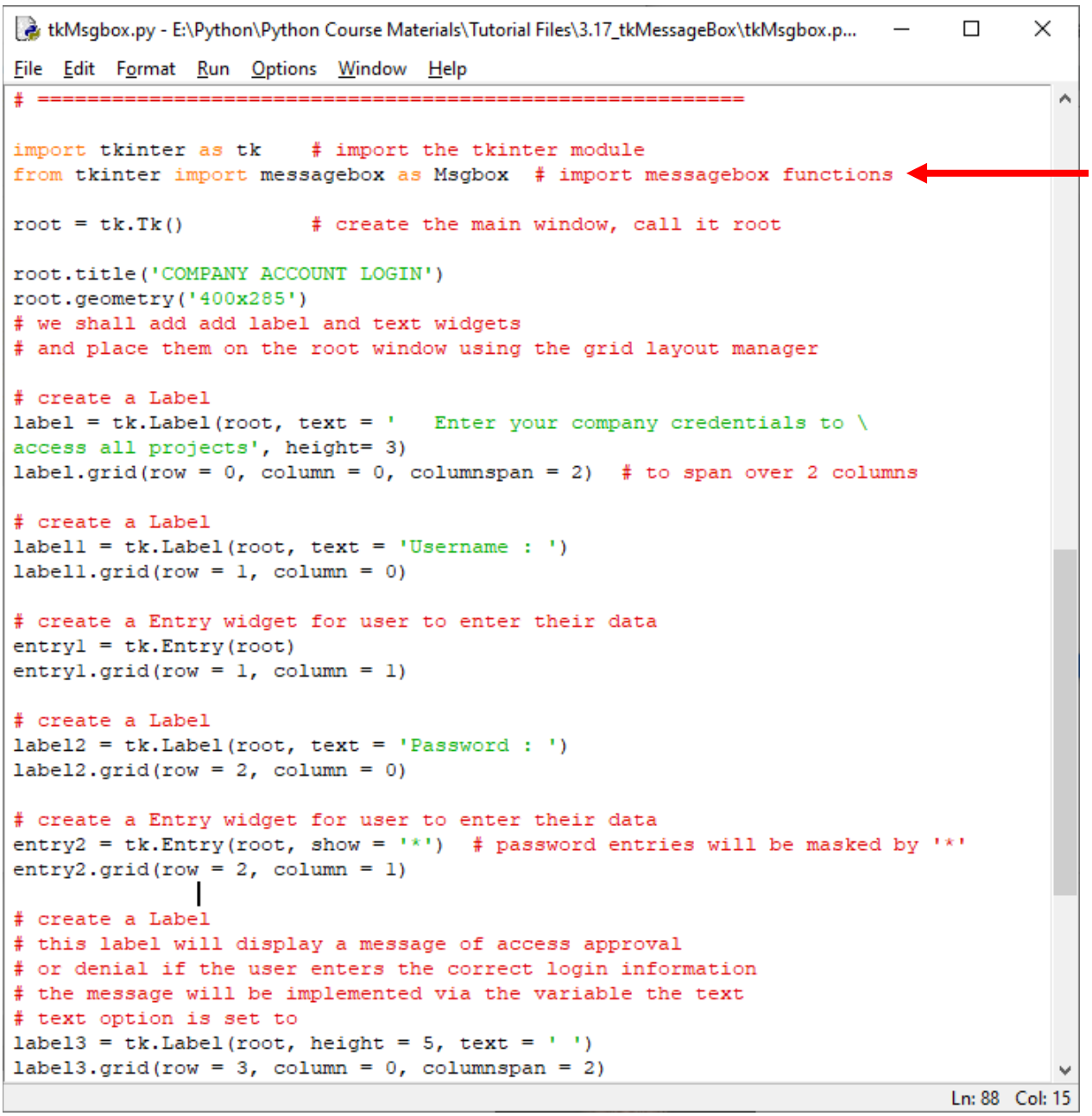
The options fall under *default* and *parent*. The default option is used to specify the default button of the message box such as RETRY, IGNORE, ABORT, etc. The parent option is used to specify the main GUI window on top of which the message box is to be displayed.

3.4 tkMessagBox Example

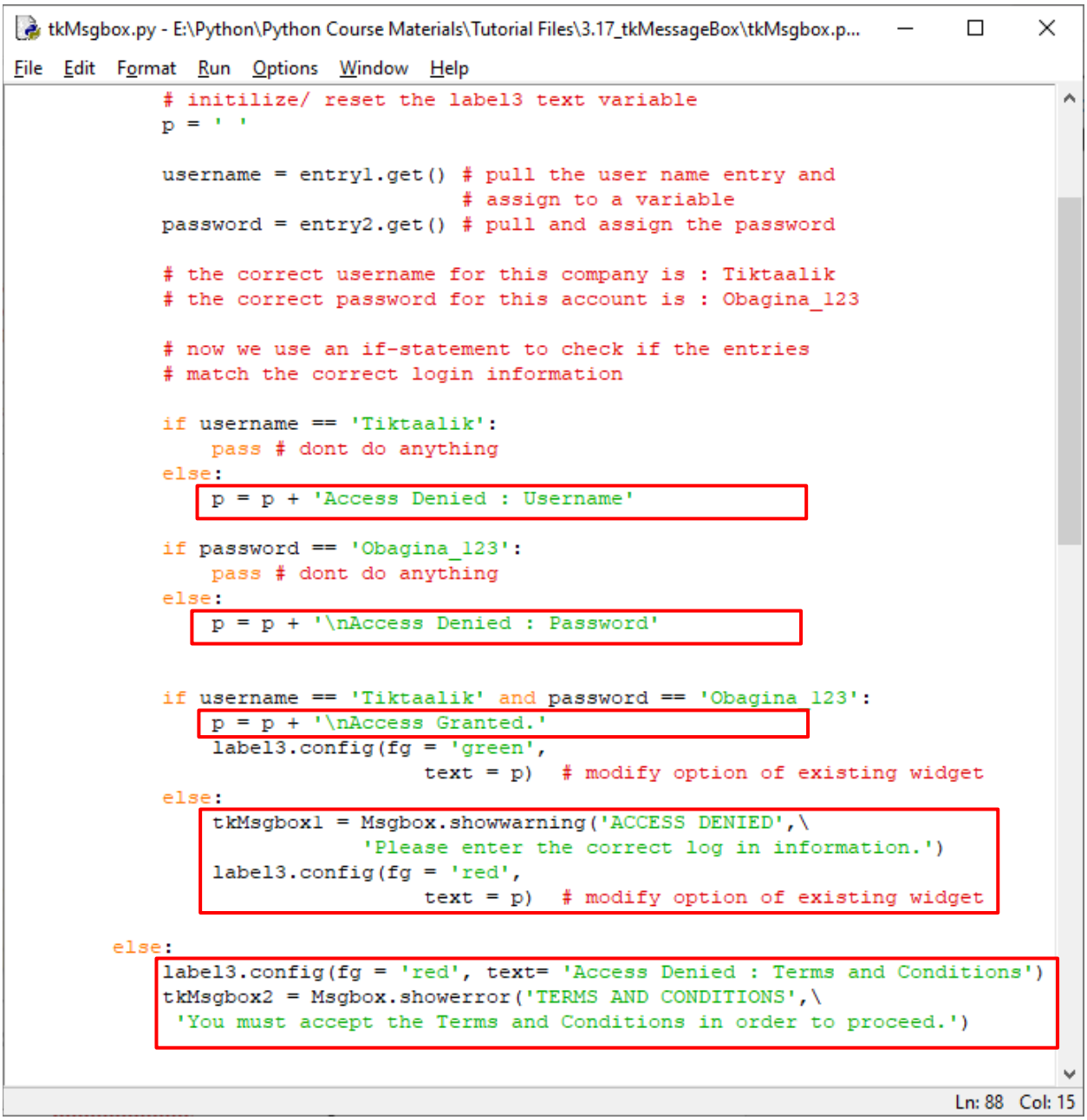
In this exercise we shall update the company account log-in developed in Chapter 7 of part 3 of this course series. We shall add some of the tkMessageBoxes to the app.

Open your company account log-in file and conduct the following updates.

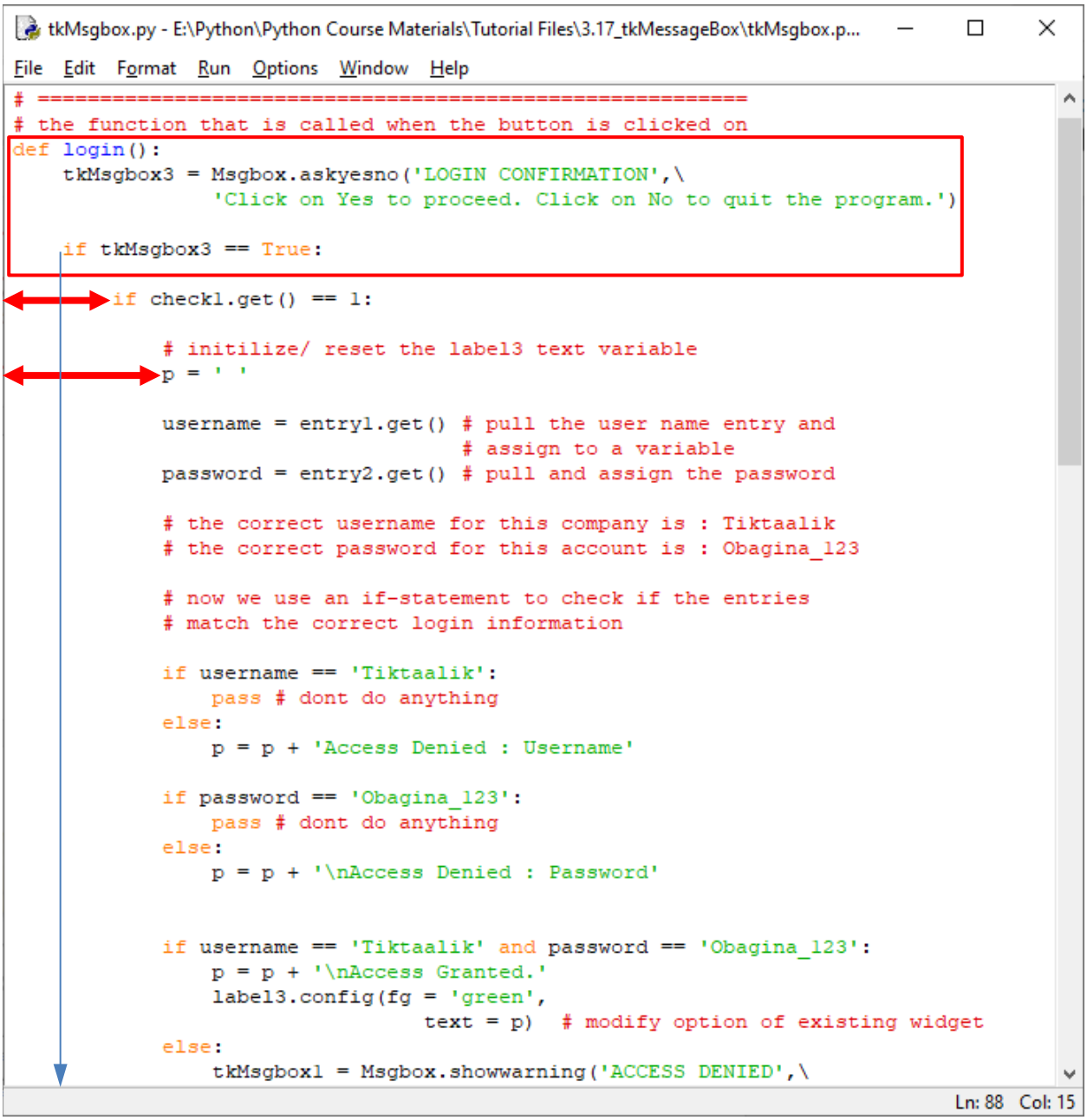
In the main body of the code add the following.



Scroll up to the function called *login( )* which the Checkbutton command option is set to. Conduct the following updates.



Now, indent your entire current code for the *login( )* function.
Wrap your indented current code for the *login( )* function inside the following *if* statement.



Add an *else* clause for the tkMsgbox3 *if* statement.

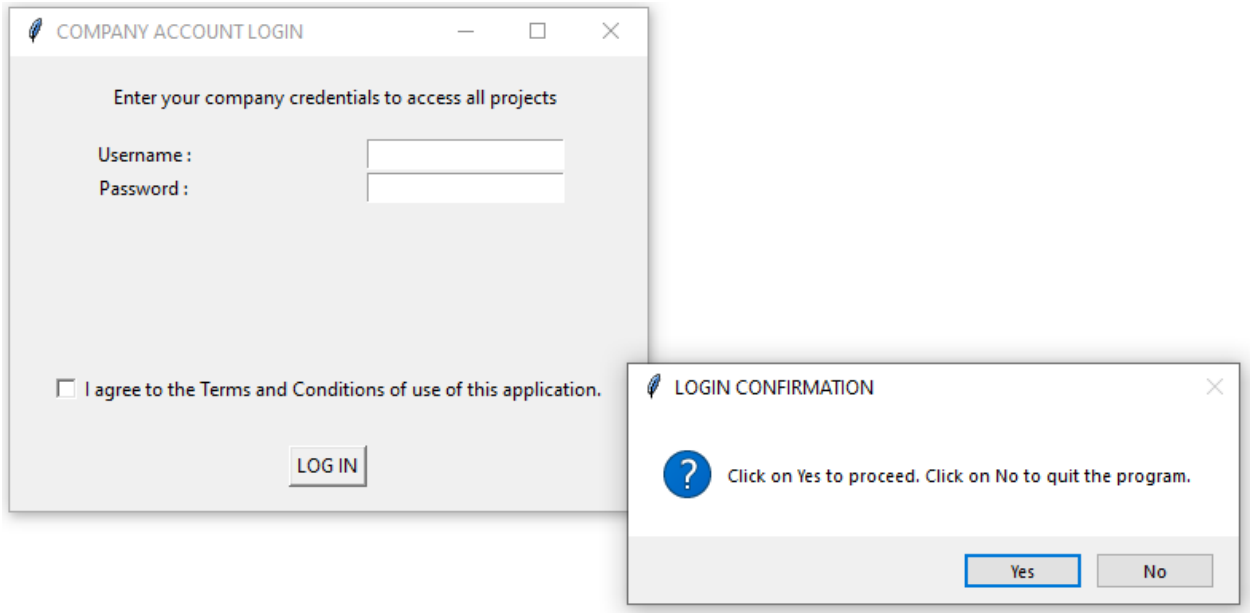
*tkMsgbox.py - E:\Python\Python Course Materials\Tutorial Files\3.17_tkMessageBox\tkMsgbox....

File Edit Format Run Options Window Help

```
            else:
                p = p + 'Access Denied : Username'

            if password == 'Obagina_123':
                pass # dont do anything
            else:
                p = p + '\nAccess Denied : Password'


            if username == 'Tiktaalik' and password == 'Obagina_123':
                p = p + '\nAccess Granted.'
                label3.config(fg = 'green',
                              text = p)  # modify option of existing widget
            else:
                tkMsgbox1 = Msgbox.showwarning('ACCESS DENIED',\
                            'Please enter the correct log in information.')
                label3.config(fg = 'red',
                              text = p)  # modify option of existing widget

        else:
            label3.config(fg = 'red', text= 'Access Denied : Terms and Conditions')
            tkMsgbox2 = Msgbox.showerror('TERMS AND CONDITIONS',\
             'You must accept the Terms and Conditions in order to proceed.')


    else:
        tkMsgbox4 = Msgbox.showinfo('Exit Program',\
                    'The program will now shut down. Good Bye.')
        root.destroy()
# ========================================================

import tkinter as tk     # import the tkinter module
from tkinter import messagebox as Msgbox  # import messagebox functions

root = tk.Tk()           # create the main window, call it root

root.title('COMPANY ACCOUNT LOGIN')
root.geometry('400x285')
# we shall add add label and text widgets
```

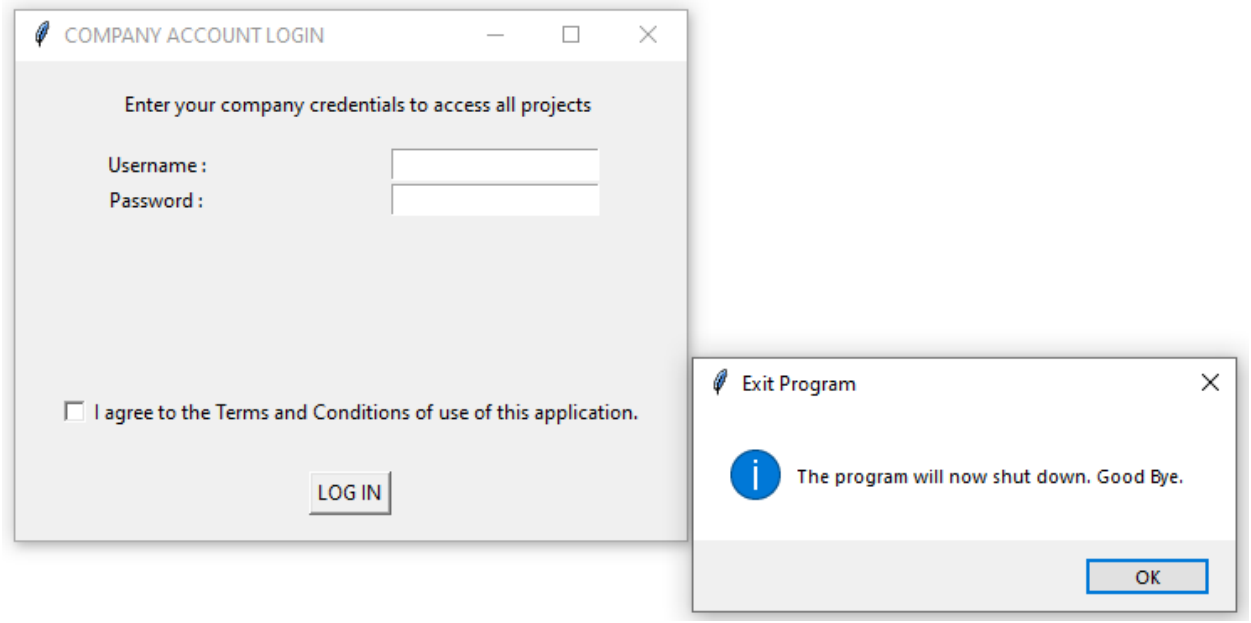
Ln: 56 Col: 8

Save the file.
Run the file.
Without making any entries nor checking to accept the terms and conditions, click the ***LOG IN*** button.



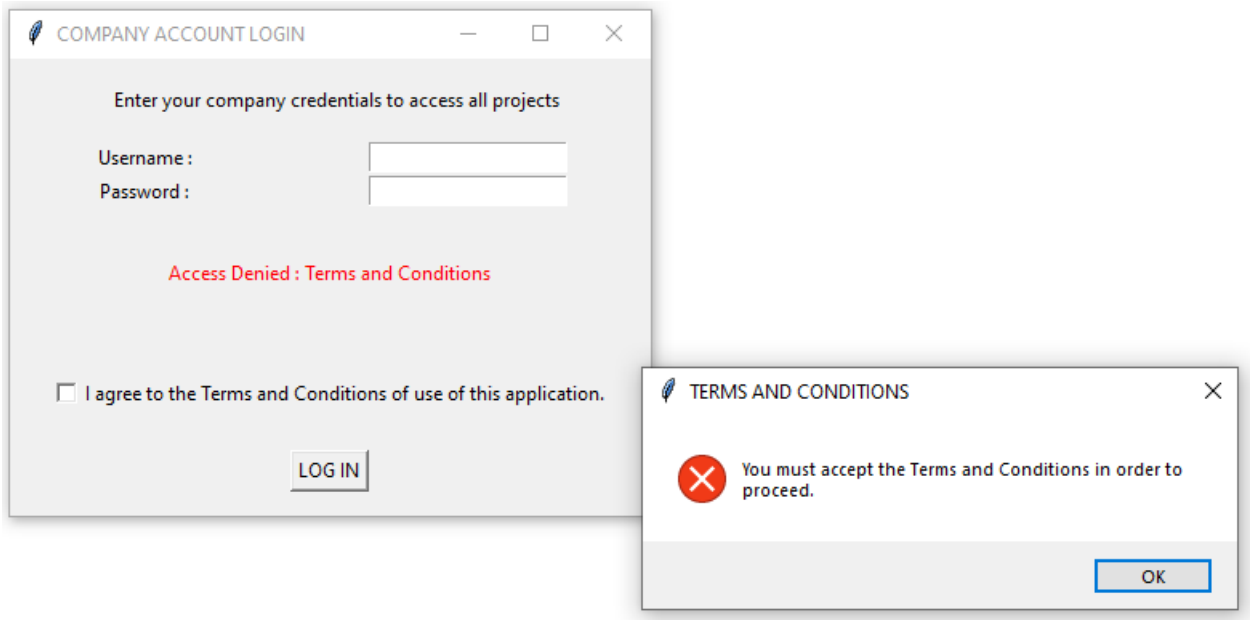
On the LOGIN CONFIRMATION *askyesno( )* tkMessageBox, Click No.

The Exit Program informational *showinfo( )* tkMessageBox appears.



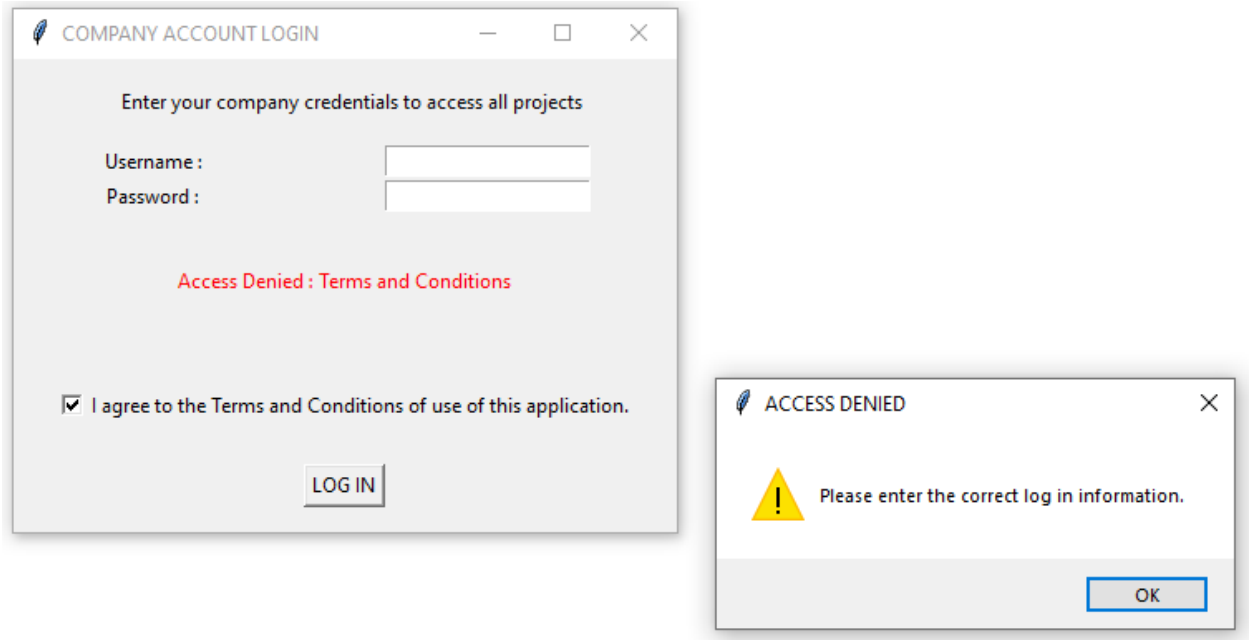
On the Exit Program informational *showinfo( )* tkMessageBox, click OK to dismiss it.
The application shuts off.

Re-run the application.
Without making any entries nor checking to accept the terms and conditions, click the ***LOG IN*** button.
This time, on the LOGIN CONFIRMATION *askyesno( )* tkMessageBox, click Yes.
The TERMS AND CONDITIONS *showerror( )* tkMessageBox pops up.

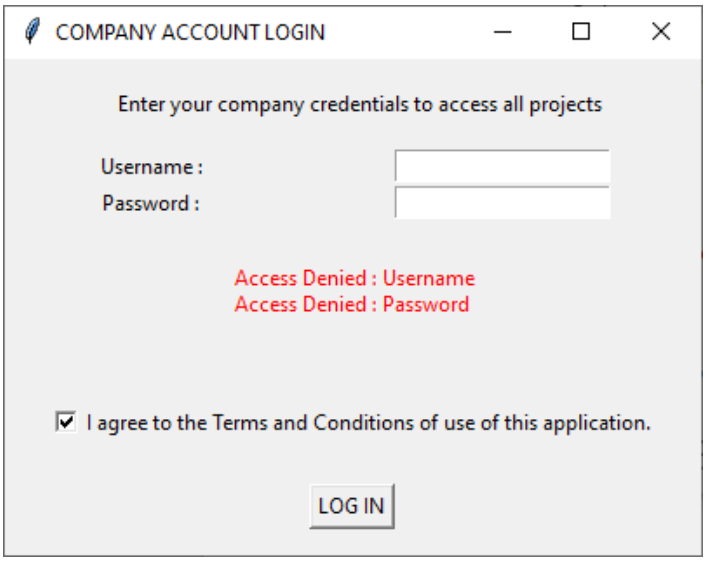


Click OK to dismiss the TERMS AND CONDITIONS *showerror( )* tkMessageBox.
On the main window (root), check the box to accept the Terms and Conditions.
Click the ***LOG IN*** button.
On the LOGIN CONFIRMATION *askyesno( )* tkMessageBox, click Yes.

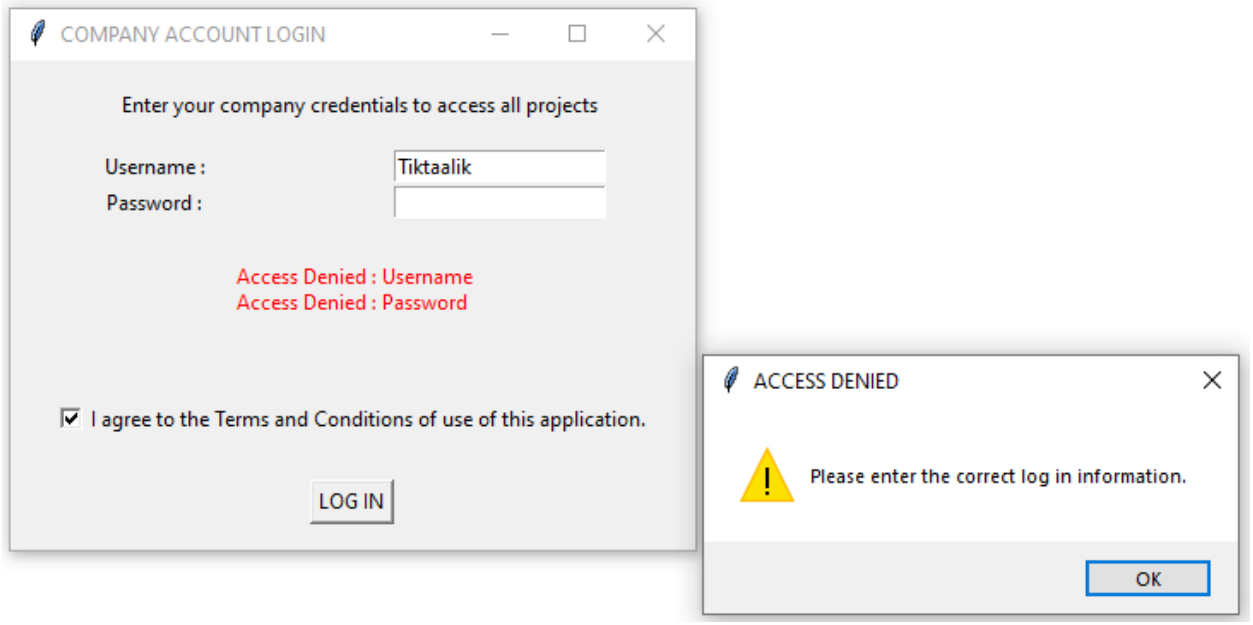
The ACCESS DENIED *showwarning( )* tkMessageBox pops up.



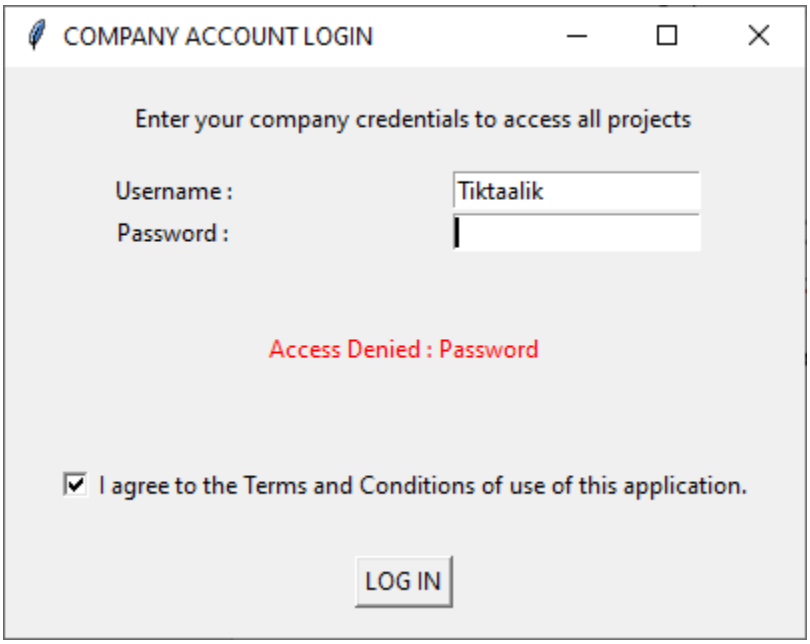
Click OK to dismiss the ACCESS DENIED *showwarning( )* tkMessageBox.
The label text on the main window (root) updates.



Enter the correct username, Tiktaalik.
Leave the password blank or enter an incorrect password.
Click the ***LOG IN*** button.
On the LOGIN CONFIRMATION *askyesno( )* tkMessageBox, click Yes.
The ACCESS DENIED *showwarning( )* tkMessageBox pops up.

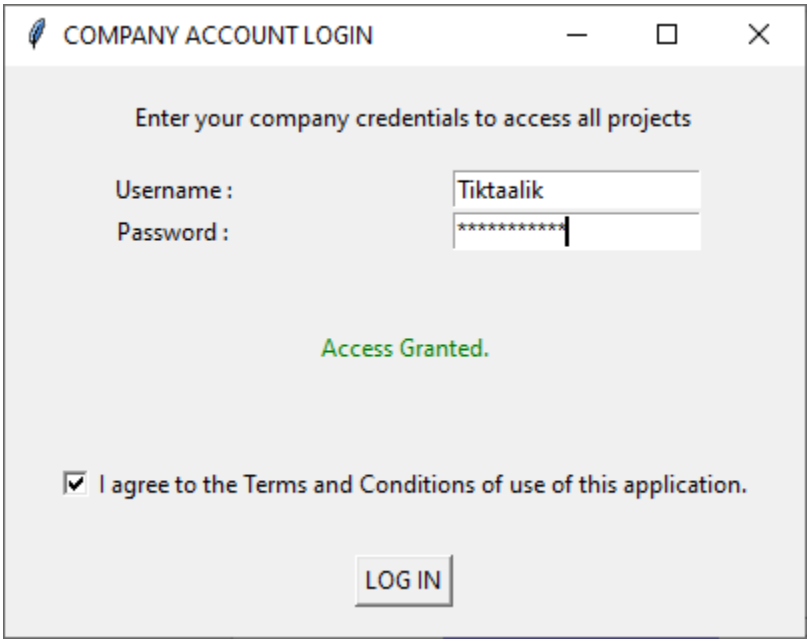


Click OK to dismiss the ACCESS DENIED *showwarning( )* tkMessageBox.
The label text on the root window updates.



Enter the correct password, Obagina_123.
Click the ***LOG IN*** button.
On the LOGIN CONFIRMATION *askyesno( )* tkMessageBox, click Yes.

Successful log in.



Exercise complete!

4. THE SCALE WIDGET

4.1 Scale

The Scale (or Slider) widget gives a graphical slider object that enables a user to select a value from a range.

The syntax is of the form,

< *variable* > = *Scale* (< *master* > , < *option* > = < *value*> , < *option*> = < *value*> , …)

where
< *variable* > is a variable name that the widget is assigned to
< *master* > is the name of the variable the main GUI window has been assigned to, or the full reference to the main window
< *option* > is an attribute
< *value* > is the specific value of the attribute

In addition to the applicable widget options presented in Chapter 3 of part 3 of this course series, other Scale widget options are summarized in Table 4.1 below.

Table 4. 1: Scale widget options

Option	Description	Values
digits	controls how to display the scale's current value through the widgets control variable	
from_	a numeric type – integer or float, that defines the start of the range of the scale	
label	specifies a text label for the scale widget	default is no label
length	specifies length of the slider along the length of the scale	default = 30 pixels
orient	specifies orientation of the scale and thus the sliding	default is HORIZONTAL, VERTICAL
repeatdelay	specifies how long (in milliseconds) the button must be held down before the slider moves repeatedly in that direction	default = 300
showvalue	controls whether the widget's value shall be displayed in a slider label or not	set to 0 to suppress label
to	a numeric type – integer or float, that defines the end of the range of the scale	
troughcolor	the color of the trough	default depends on your operating system
variable	set how the current value of the scale is displayed via the widget's control variable	

4.2 Scale widget methods

The Scrollbar widget methods are summarized in Table 4.2.

Table 4. 2: Scale widget methods

Method	**Description**
get (<options>)	returns the current value of the Scale
set (<value>)	sets the scale widget's value

4.3 Scale Example

In this exercise we shall develop a simple industrial temperature monitor. The user will use the slider to monitor the temperature of the system, and then press a Button to display the value an Entry widget.

Open a new session of IDLE (Python GUI).
Click on **File**.
Click on **New File**, to open the File Editor.

Replicate the following code.

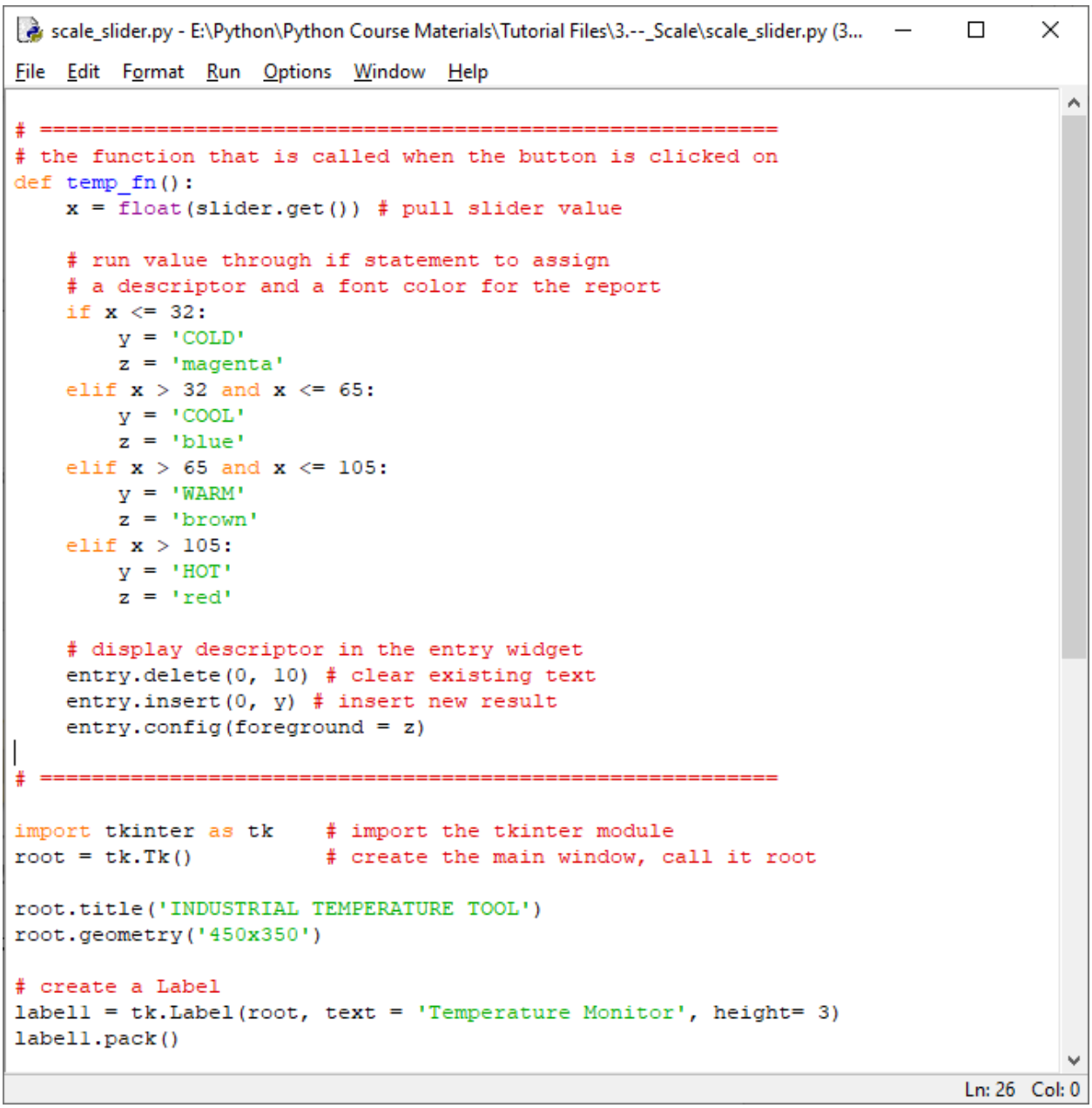
scale_slider.py - E:\Python\Python Course Materials\Tutorial Files\3.--_Scale\scale_slider.py (3...

File Edit Format Run Options Window Help

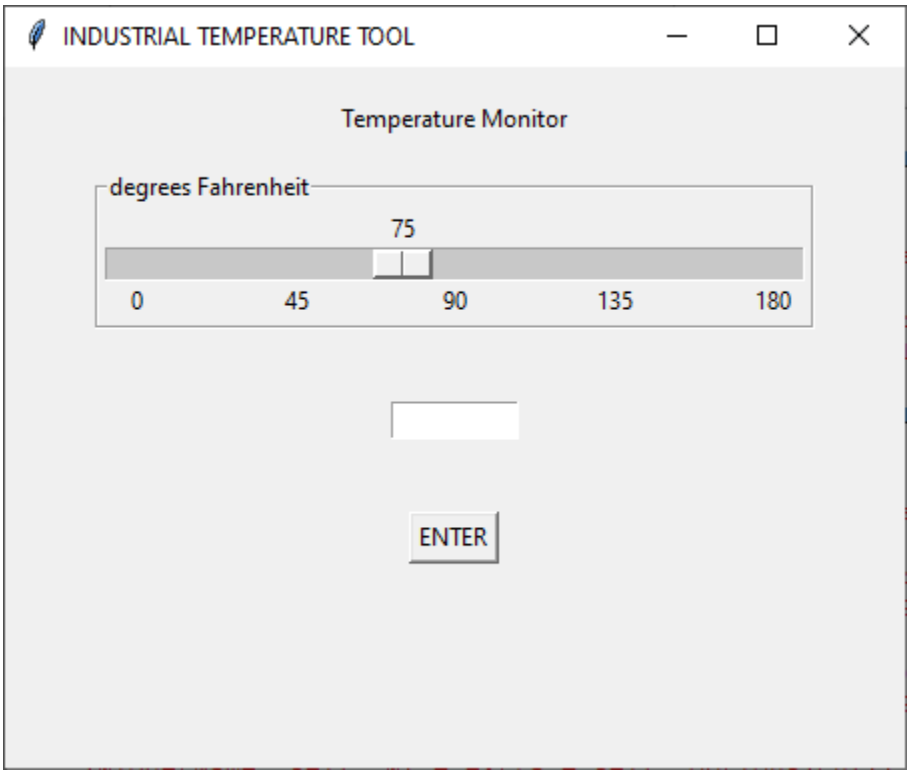
```
# ============================================================

import tkinter as tk     # import the tkinter module
root = tk.Tk()           # create the main window, call it root

root.title('INDUSTRIAL TEMPERATURE TOOL')
root.geometry('450x350')

# create a Label
label1 = tk.Label(root, text = 'Temperature Monitor', height= 3)
label1.pack()

# create LabelFrame to hold the scale/ slider
lblframe = tk.LabelFrame(root, text = 'degrees Fahrenheit')
lblframe.pack()

# create a temp scale/ slider
slider = tk.Scale(lblframe, from_ = 0, to = 180, orient = tk.HORIZONTAL,\
                  length = 350, tickinterval = 45 )
slider.set(75)  # set a value for the slider
slider.pack()

# create a Label
label2 = tk.Label(root, text = ' ', height= 2)  # spacer
label2.pack()

# create a Entry widget
entry = tk.Entry(root, width = 10)
entry.pack()

# create a Label
label3 = tk.Label(root, text = ' ', height= 2)  # spacer
label3.pack()

# create click button
button = tk.Button(root, text = 'ENTER', command = temp_fn)
button.pack()

root.mainloop()
```

Ln: 51 Col: 36

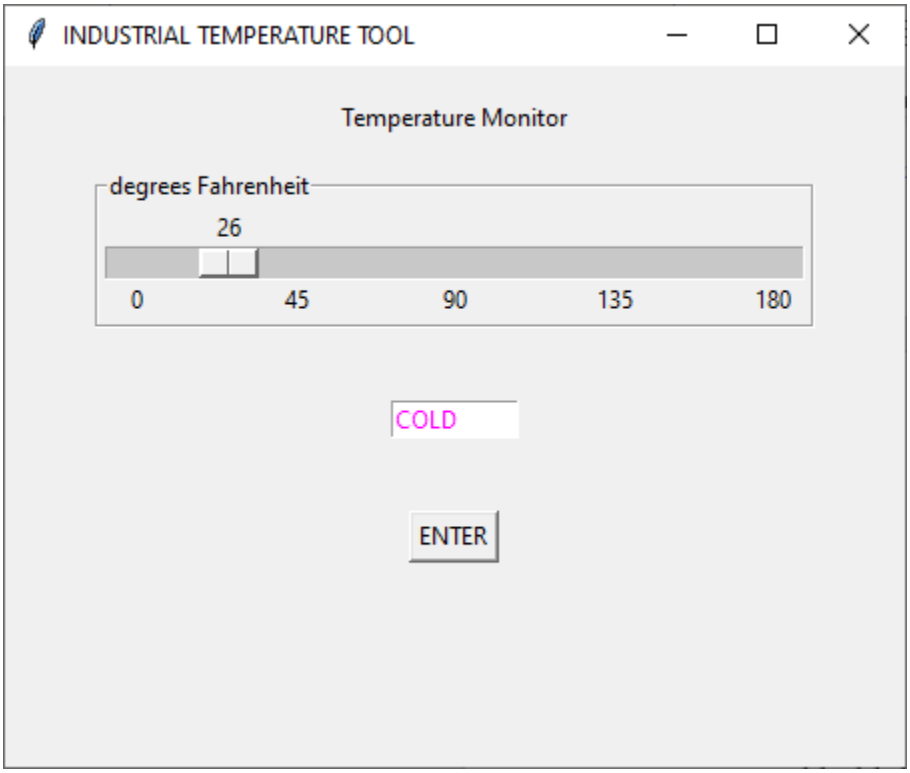
At the very top of the script, add the following code that defines the function for the Button.



Save the file.
Run the file.



Test the slider.
Press and hold down the slider button and while doing so drag the button in either direction.
Release the slider button, select a value.
Click on the Enter button.
The slider value displays on the Entry widget.
Repeat numerous trials.



Successful completion!

5. THE SCROLLBAR WIDGET

5.1 Scrollbar

The Scrollbar widget is used to give the capability of vertical sliding (or scrolling) to other widgets - Text, Canvas, and Listbox.

The syntax is of the form,

< *variable* > = *Scrollbar* (< *master* > , < *option* > = < *value*> , < *option*> = < *value*> , …)

where
< *variable* > is a variable name that the widget is assigned to
< *master* > is the name of the variable the main GUI window has been assigned to, or the full reference to the main window
< *option* > is an attribute
< *value* > is the specific value of the attribute

In addition to the applicable widget options presented in Chapter 3 of part 3 of this course series, other Scrollbar widget options are summarized in Table 5.1 below.

Table 5. 1: Scrollbar widget options

Option	Description	Values
jump	controls what happens when the slider is dragged	default of 0 causes the command callback to be called, with value of 1 the callback is called after the user releases the mouse
orient	specifies the orientation of the scrollbar (but not the scrolling)	HORIZONTAL, VERTICAL
repeatdelay	specifies how long (in milliseconds) the button must be held down before the slider moves repeatedly in that direction	default = 300

5.2 Scrollbar widget methods

The Scrollbar widget methods are summarized in Table 5.2.

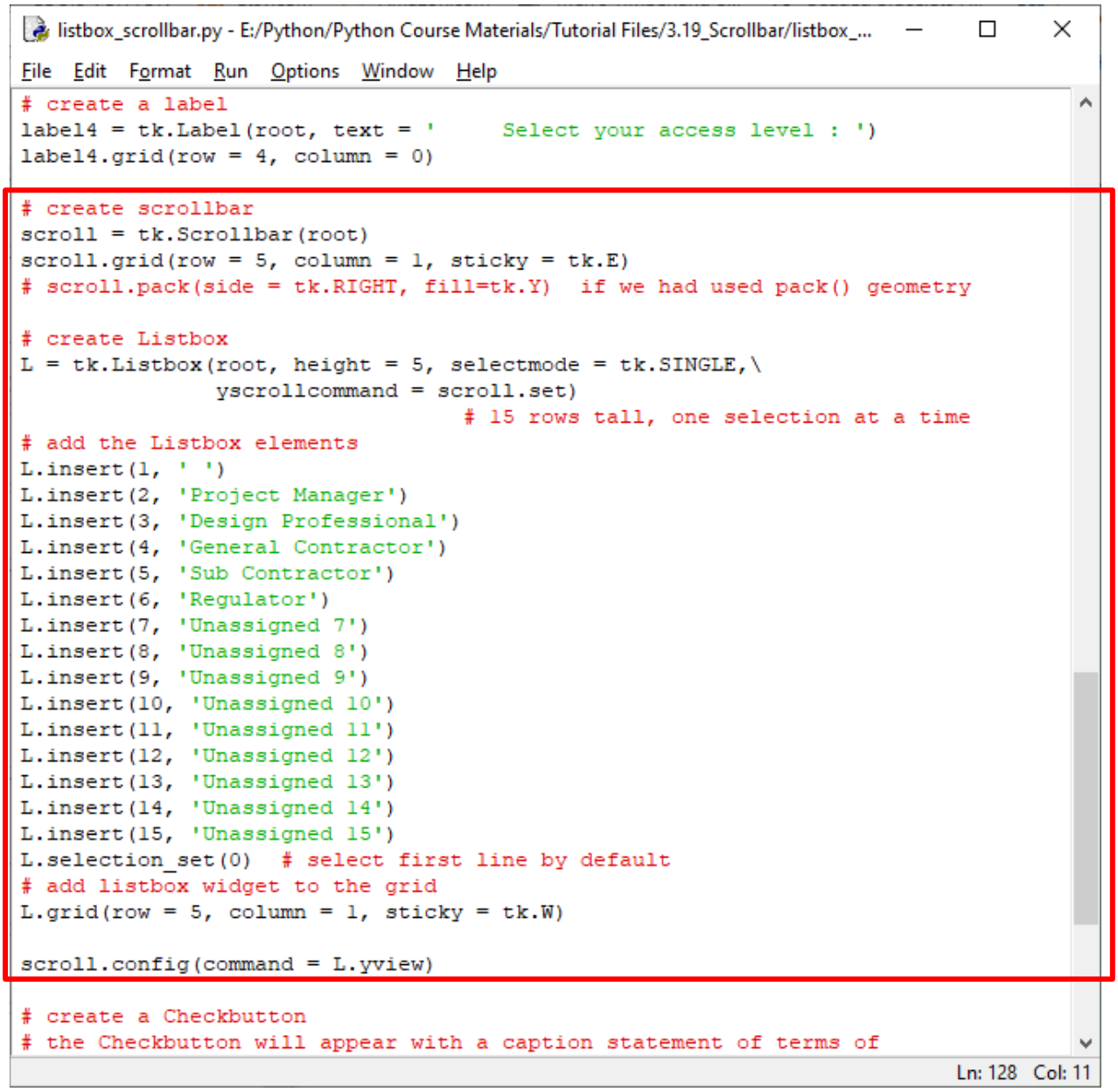
Table 5. 2: Scrollbar widget methods

Method	Description
get (<options>)	returns (p, q) where p is the current position of the left or top edge of the slider for a horizontal and vertical scrollbar respectively, q is the right or bottom edge of the slider
set (<first> , <last>)	to connect the scrollbar to some other widget, set that widget's *xscrollcommand* or *yscrollcommand* to the scrollbar's *set( )* method

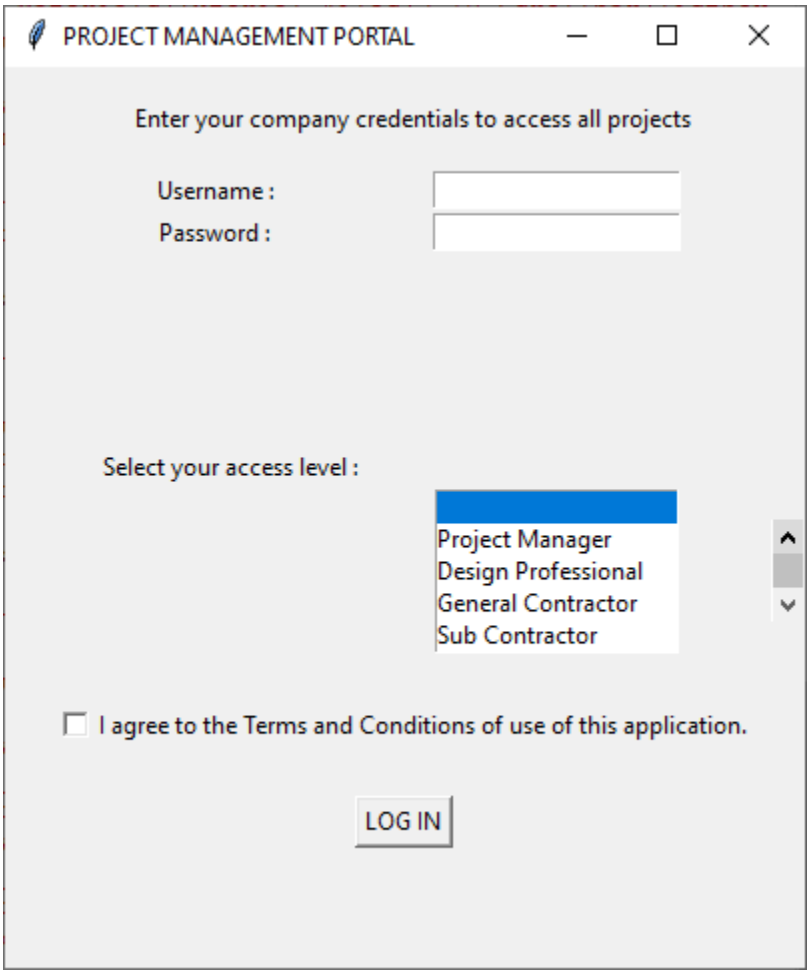
5.3 Scrollbar Example

In this example we shall modify application we built in Chapter 9 of part 3 of this course series. Specifically, we shall modify the Listbox widget and add a vertical scrollbar to it.

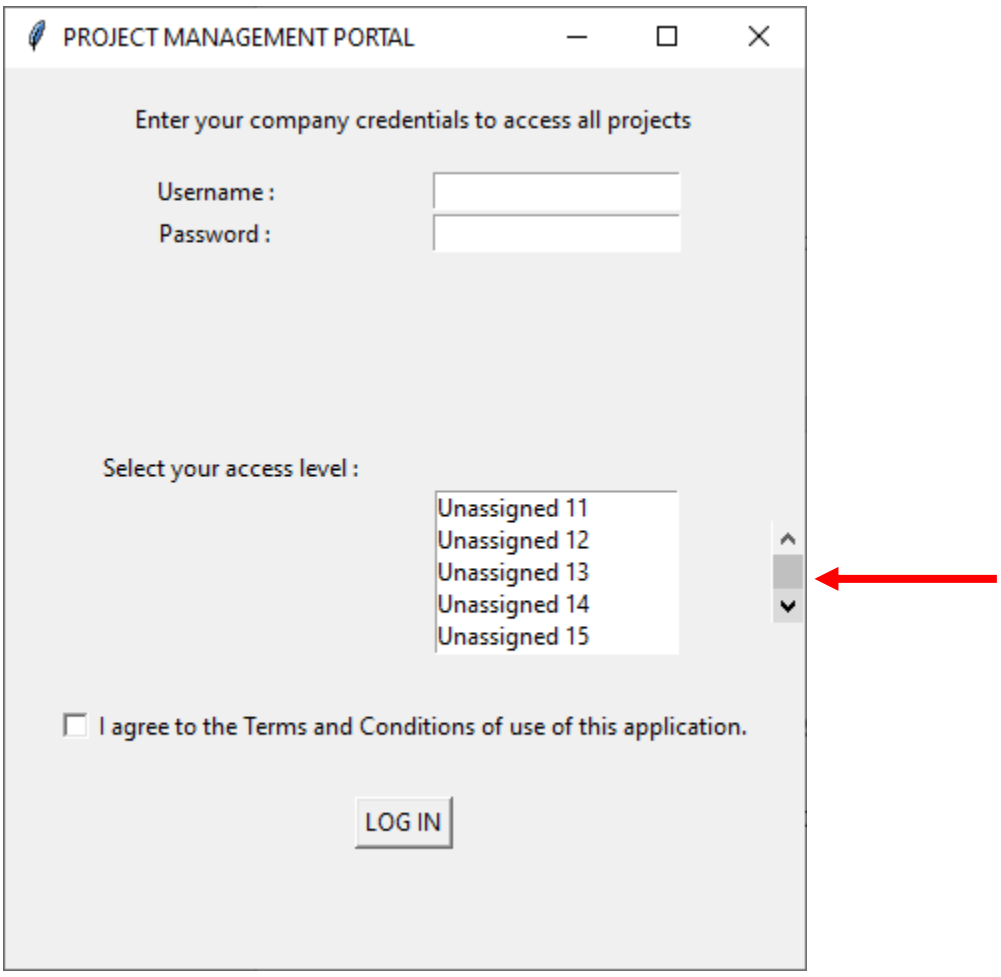
Open your code file for the application we developed in Chapter 9 of part 3 of this course series. Conduct the following updates.



Save the file.
Run the file.



Use the Scrollbar to scroll through the Listbox.



Successful completion!

6. THE TOPLEVEL WIDGET

6.1 Toplevel

A Toplevel widget is used as a container widget in a similar fashion as using a Frame, however the display opens in a separate top-level window. Thus, the Toplevel widget is typically used to display additional application windows, dialogs, and other pop ups.

The syntax is of the form,

< *variable* > = *Toplevel* (< *option* > = < *value*> , < *option*> = < *value*> , …)

where
< *variable* > is a variable name that the widget is assigned to
< *option* > is an attribute
< *value* > is the specific value of the attribute

6.2 Toplevel widget methods

The Toplevel widget methods are summarized in Table 6.1.

Table 6. 1: Toplevel widget methods

Method	**Description**
deiconify ( )	displays the window if the *iconify* or *withdraw* methods were used to remove the window from the screen
frame ( )	returns a string that contains a system-specific window identifier for the window's outermost parent window
group(<window>)	adds window to the window group controlled by the specified window

Table 6.1 (Continued): Toplevel widget methods

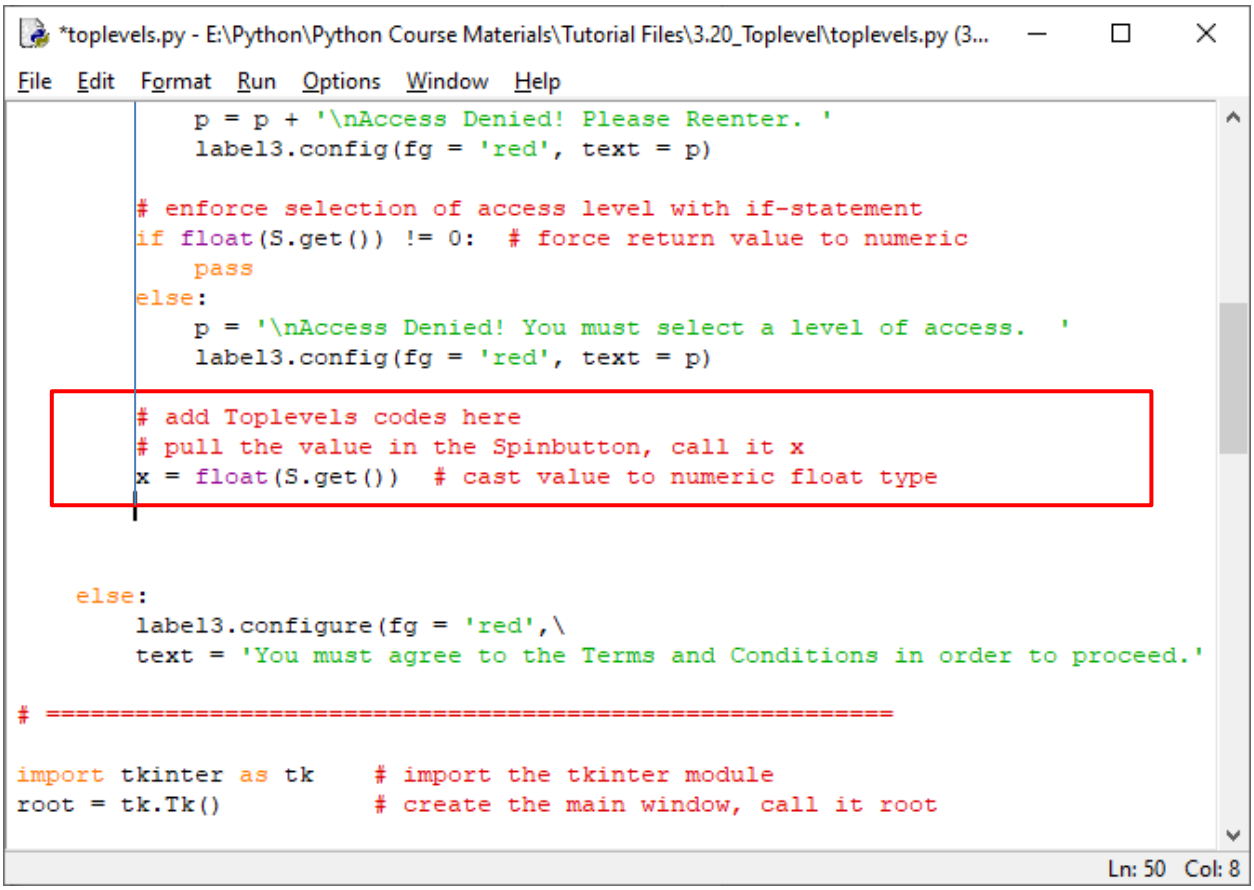
Method	Description
iconify ( )	turns the window into an icon but does not destroy it, the window is reopened by using the *deiconfy* method
maxsize(<width>, <height>)	sets the maximum size of the window
minisize(<width>, <height>)	sets the minimum size of the window
positionfrom(<who>) or *wm_positionfrom(<who>)*	sets or returns the position of the controller
protocol (<name>, <function>)	registers a callback function for the given protocol
resizable(<width>, <height>) or *wm_resizable(<width>, <height>)*	controls whether the window can be resized horizontally or vertically
sizefrom(<who>)	sets or returns the size of the controller
state ( )	returns the current state of the window, values are "normal", "iconic", "withdrawn" or "icon"
title(<string>) or *wm_string(<string>)*	sets or returns the title of the window
transient([<master>])	makes window a transient (or temporary) window for the specified master, if master is not specified, it defaults to self's parent; a transient window is always drawn on top of its master and is hidden when the master is iconified or withdrawn.
withdraw ( )	removes the window from the screen but does not without destroying it. to reopen the window, use *deiconify* method

6.3 Toplevel Example

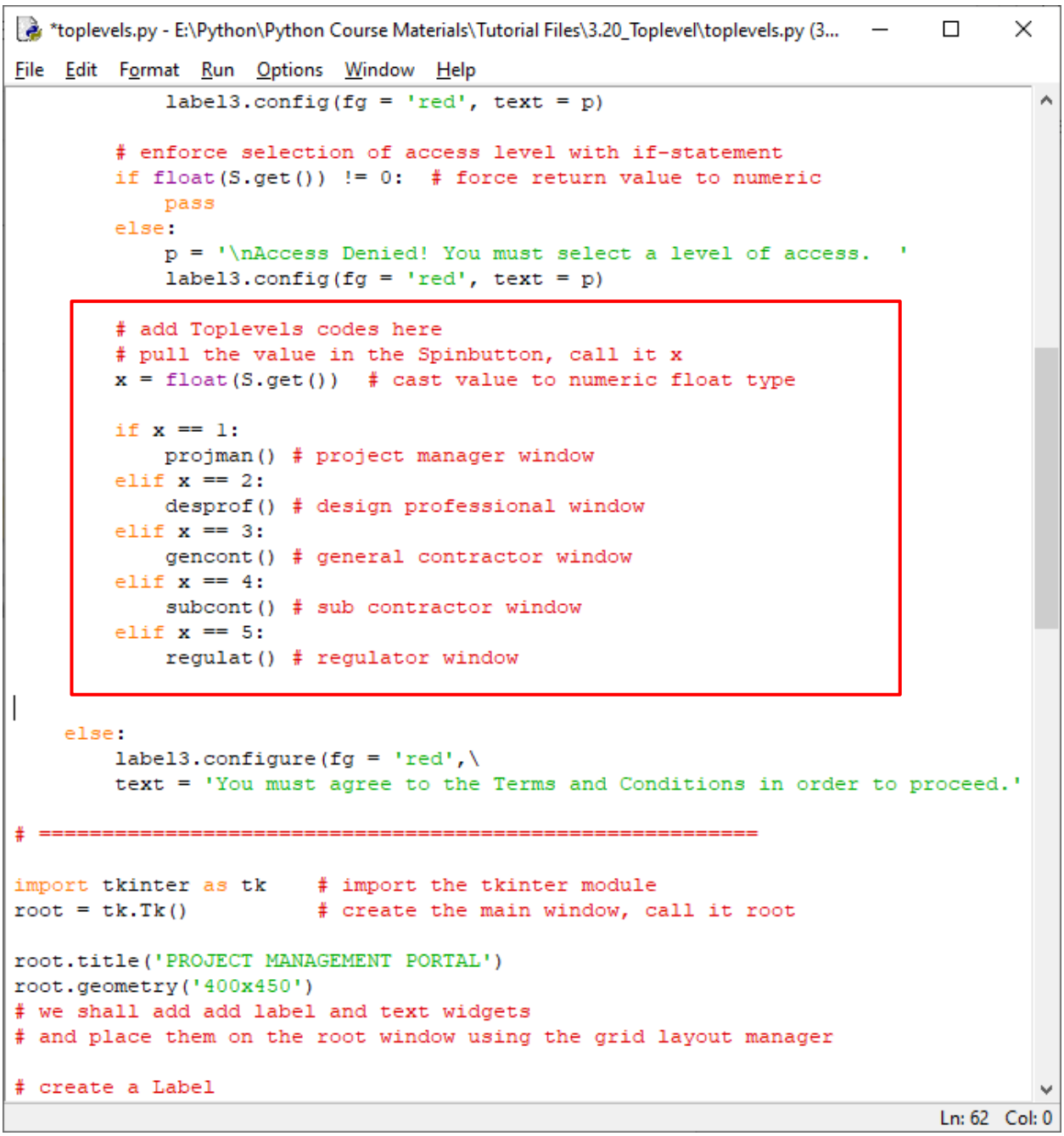
A Toplevel window may be created anywhere in the code of the root. However, in most cases a Toplevel window opens after the user interacts with some widget or the other on the root window. Therefore, typically, a Toplevel window is implemented through the function assigned to the *command* option of the relevant widget.

In this exercise we shall upgrade the project management portal we developed in Chapter 10 of part 3 of this course series that uses the Spinbox. Specifically, we shall add Toplevel windows that open based on what access level is selected by the user.

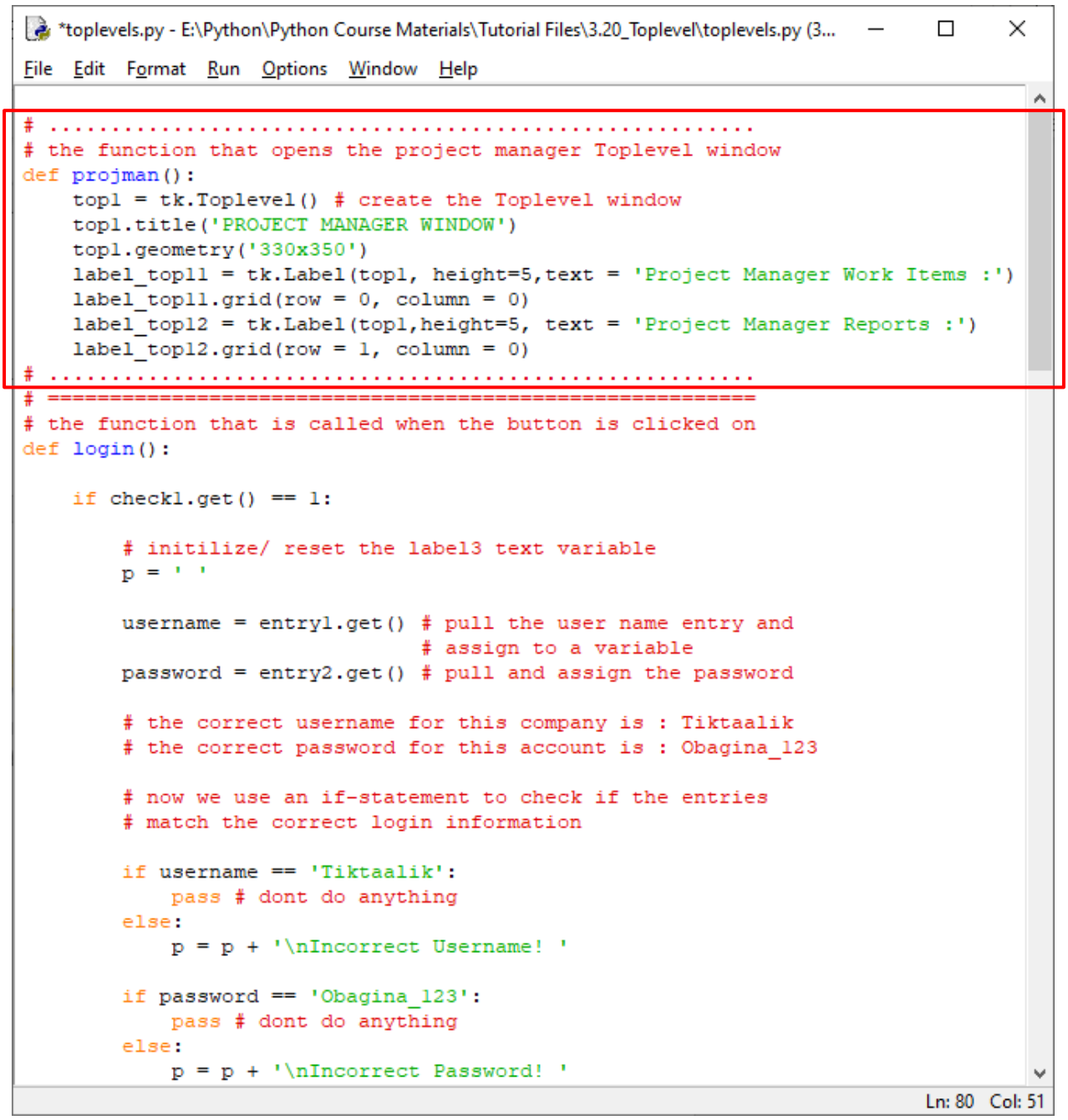
Open your code file for your project management portal with Spinbutton. Within the *login( )* function conduct the following updates.



Continue as follows.



Now, we scroll to the top of our script to add code for the window functions.



Add for the other windows.
(To work smart rather than unwarrantedly hard, copy the first function, paste it and then update it for the subsequent function, and so on and so forth.)

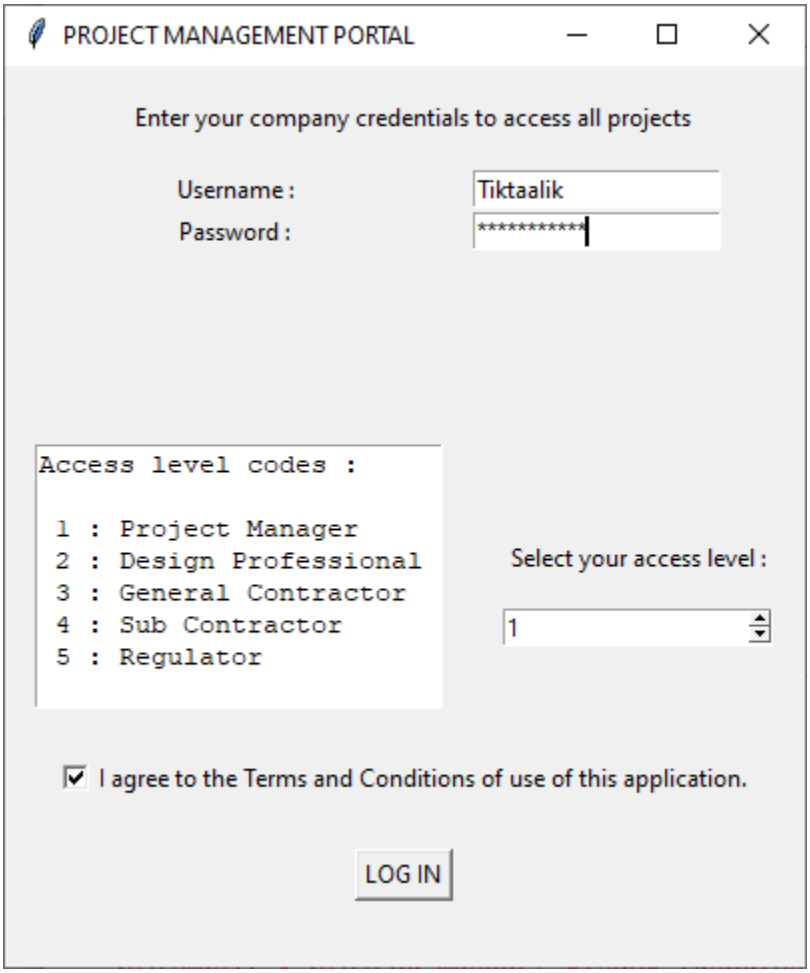
toplevels.py - E:\Python\Python Course Materials\Tutorial Files\3.20_Toplevel\toplevels.py (3.7.2)

File Edit Format Run Options Window Help

```
# the function that opens the design professional Toplevel window
def desprof():
    topl = tk.Toplevel() # create the Toplevel window
    topl.title('DESIGN PROFESSION WINDOW')
    topl.geometry('330x350')
    label_topl1 = tk.Label(topl, height=5,text = 'Design Professional Work Items :')
    label_topl1.grid(row = 0, column = 0)
    label_topl2 = tk.Label(topl,height=5, text = 'Design Professional Reports :')
    label_topl2.grid(row = 1, column = 0)
# .....................................................
# the function that opens the project manager Toplevel window
def gencont():
    topl = tk.Toplevel() # create the Toplevel window
    topl.title('GENERAL CONTRACTOR WINDOW')
    topl.geometry('330x350')
    label_topl1 = tk.Label(topl, height=5,text = 'General Contractor Work Items :')
    label_topl1.grid(row = 0, column = 0)
    label_topl2 = tk.Label(topl,height=5, text = 'General Contractor Reports :')
    label_topl2.grid(row = 1, column = 0)
# .....................................................
# the function that opens the project manager Toplevel window
def subcont():
    topl = tk.Toplevel() # create the Toplevel window
    topl.title('SUB CONTRACTOR WINDOW')
    topl.geometry('330x350')
    label_topl1 = tk.Label(topl, height=5,text = 'Sub Contractor Work Items :')
    label_topl1.grid(row = 0, column = 0)
    label_topl2 = tk.Label(topl,height=5, text = 'Sub Contractor Reports :')
    label_topl2.grid(row = 1, column = 0)
# .....................................................
# the function that opens the project manager Toplevel window
def regulat():
    topl = tk.Toplevel() # create the Toplevel window
    topl.title('REGULATOR WINDOW')
    topl.geometry('330x350')
    label_topl1 = tk.Label(topl, height=5,text = 'Regulator Work Items :')
    label_topl1.grid(row = 0, column = 0)
    label_topl2 = tk.Label(topl,height=5, text = 'Regulator Reports :')
    label_topl2.grid(row = 1, column = 0)
# .....................................................
# =====================================================
# the function that is called when the button is clicked on
```

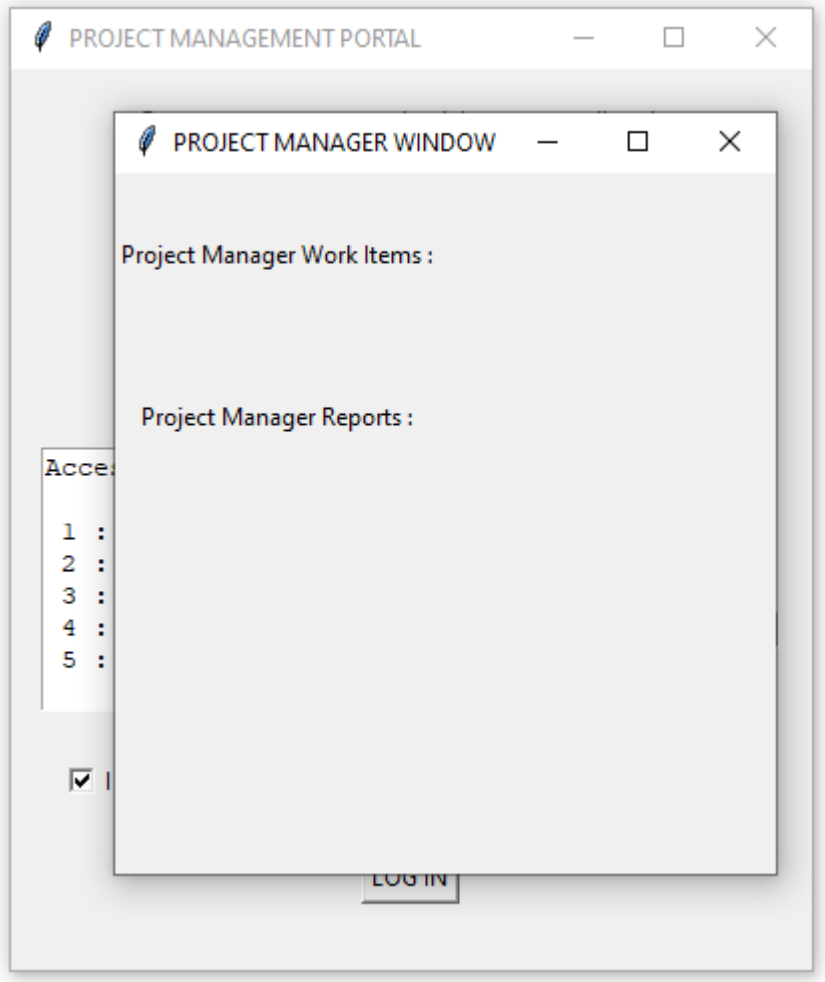
Ln: 28 Col: 68

Save the file.
Run the file.
Enter the login information.
Accept the Terms and Conditions.
Select an access level.



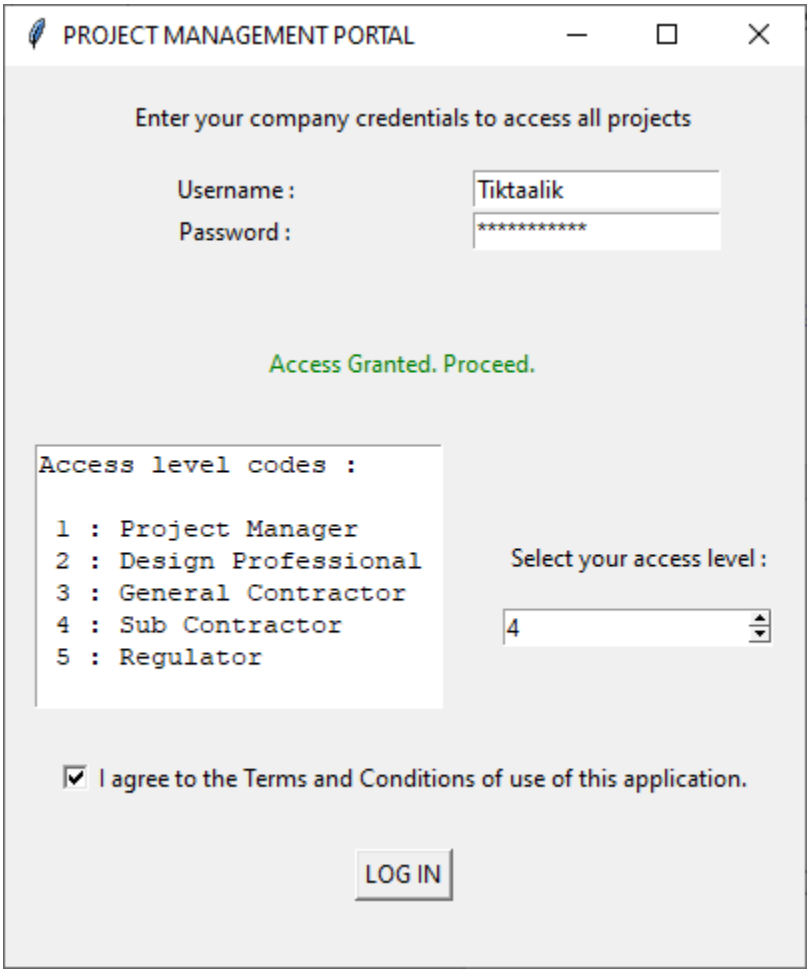
Click on the ***LOG IN*** button

The Project Manager Window Toplevel opens.

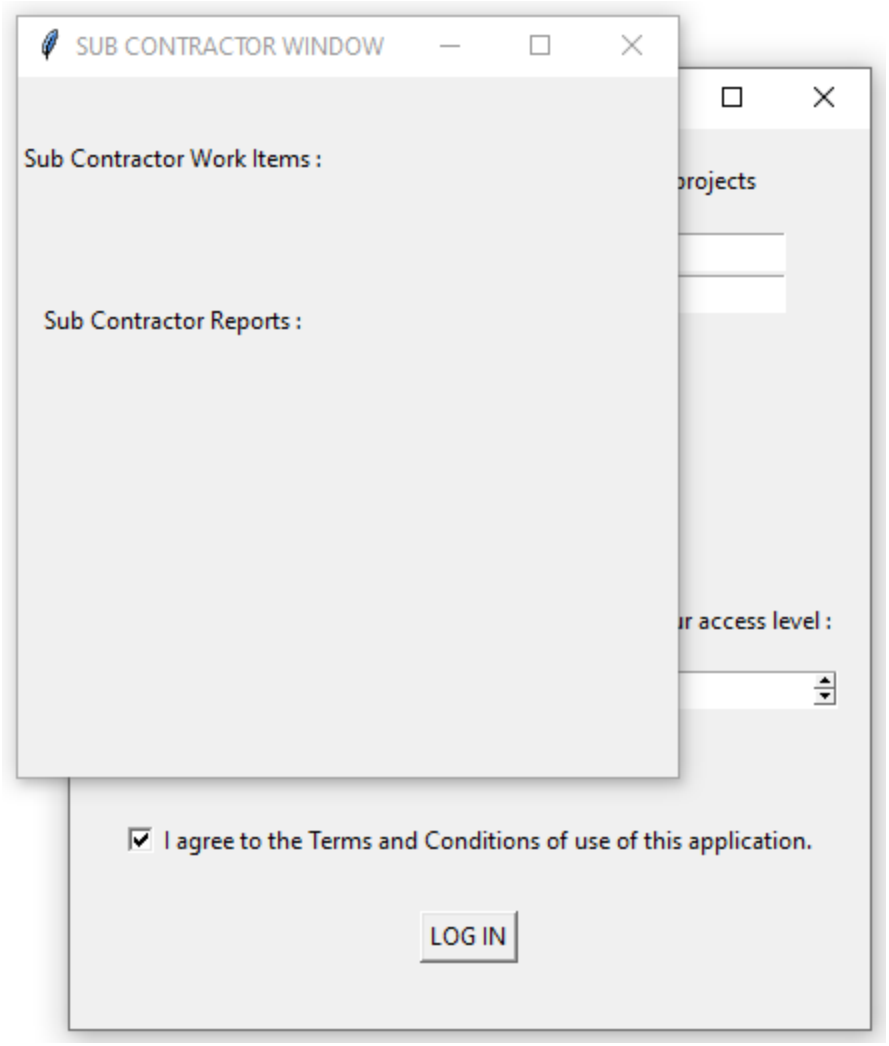


Close the Project Manager Window.

Test another one.



The Toplevel window opens accordingly.



Successful completion!

7. THE PANEDWINDOW WIDGET

7.1 PanedWindow

A PanedWindow is a container widget that may have any number of panes. The panes are "child" widgets of the main PanedWindow and may be arranged horizontally or vertically. Each pair of panes is separated by a moveable separator (or sash) that enables a user to resize a pane as desired.

The syntax to create a PanedWindow is of the form,

< var > = PanedWindow (< option > = <value> , <option> = < value> , …)

where
< var > is a variable name that the widget is assigned to
< option > is an attribute
< value > is the specific value of the attribute

The syntax to add a child panes to the PanedWindow is of the form

< var > . add(< child 1 >)
< var > . add(< child 2 >)
:
:
:
< var > . add(< child n >)

where
< child n > is a variable name the nth child pane is assigned to

In addition to the applicable widget options presented in Chapter 3 of part 3 of this course series, other PanedWindow widget options are summarized in Table 7.1 below.

Table 7. 1: PanedWindow widget options

Option	Description	Values
handlepad		default of 8
handlesize		default of 8
orient		default is HORIZINTAL
sashcursor		
sashpad		
sashrelief		Default is RAISED
sashwidth		default is 2
showhandle		

7.2 PanedWindow widget methods

The PanedWindow widget methods are summarized in Table 7.2.

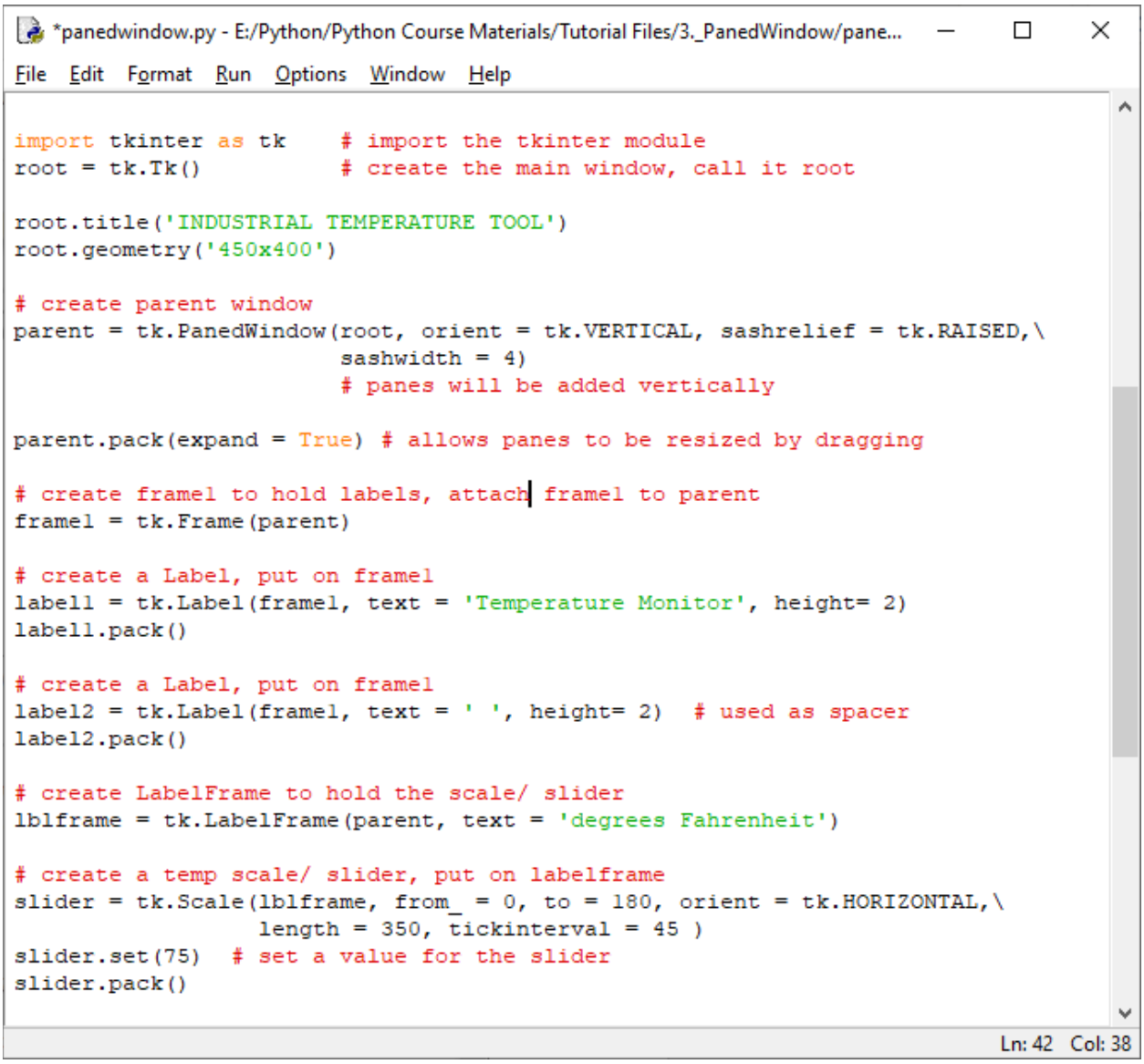
Table 7. 2: PanedWindow widget methods

Method	**Description**
add(child, options)	adds a child pane to the PanedWindow
config(<options>)	modifies specified widget options
forget(<child>) or *remove(<child>)*	deletes a child window
get(<startindex> [, <endindex>])	returns characters from a text for the specified range
panes( )	returns a list of child widgets
sash_coord(<index>)	returns the current position of the sash
sash_dragto(<index>, <x>, <y>)	drags the sash to a new position, relative to the mark
sash_mark(<index>, <x>, <y>)	registers the current mouse position

7.3 PanedWindow Example

In this example we shall create a new version of the industrial temperature tool we developed in Chapter 4 that incorporates PanedWindows.

Open a new session of IDLE (Python GUI).
Click on **File**.
Click on **New File**, to open the File Editor.
Replicate the following code.



Continue as follows.

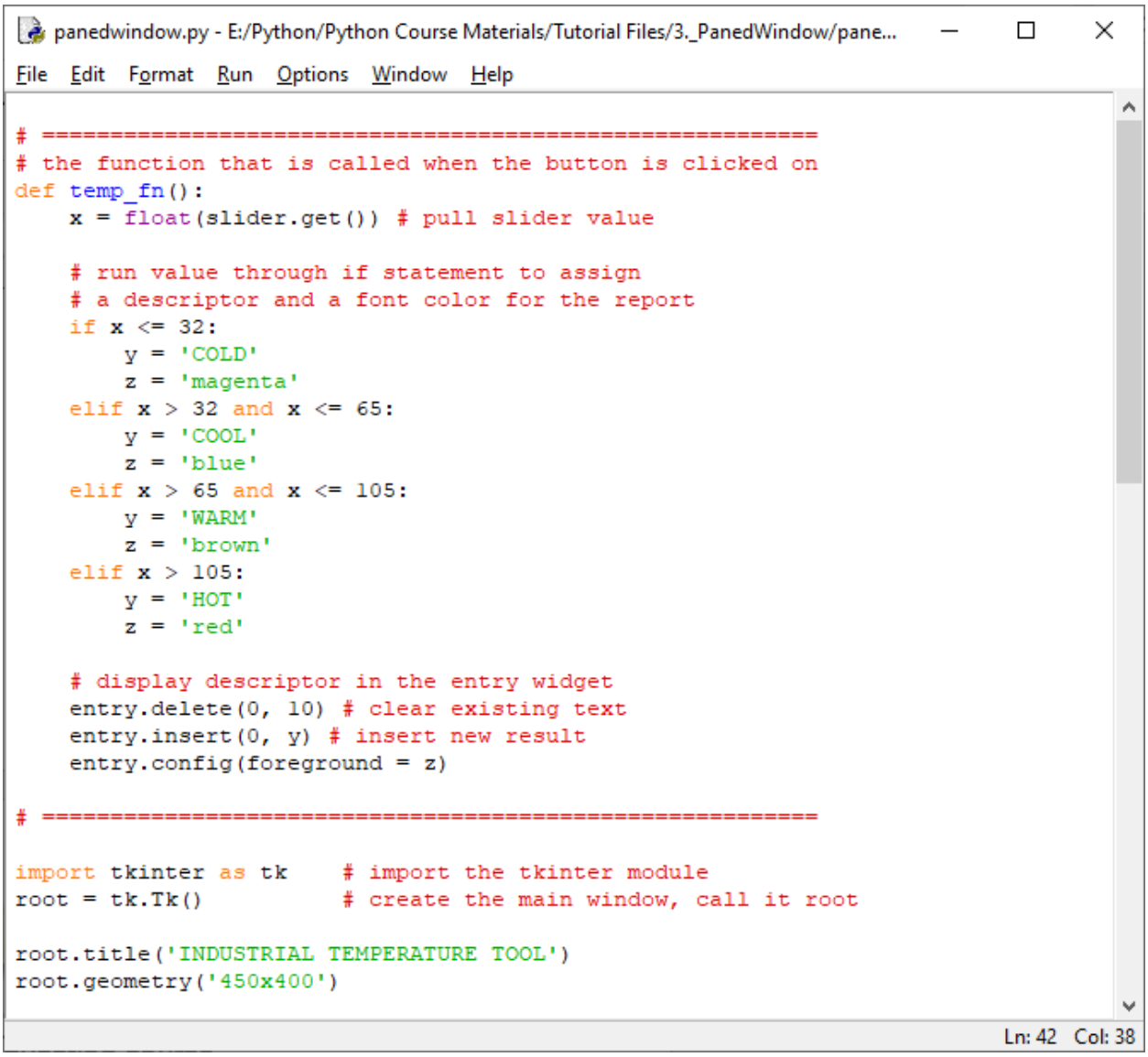
*panedwindow.py - E:/Python/Python Course Materials/Tutorial Files/3._PanedWindow/pane...

File Edit Format Run Options Window Help

```
# create LabelFrame to hold the scale/ slider
lblframe = tk.LabelFrame(parent, text = 'degrees Fahrenheit')

# create a temp scale/ slider, put on labelframe
slider = tk.Scale(lblframe, from_ = 0, to = 180, orient = tk.HORIZONTAL,\
                  length = 350, tickinterval = 45 )
slider.set(75)  # set a value for the slider
slider.pack()

# create frame2 to hold subsequent entry, label and button
frame2 = tk.Frame(parent)

# create a Label, put on frame2
label3 = tk.Label(frame2, text = ' ', height= 1)  # used as spacer
label3.pack()

# create a Entry widget, put on frame2
entry = tk.Entry(frame2, width = 10)
entry.pack()

# create a Label, put on frame2
label4 = tk.Label(frame2, text = ' ', height= 1)  # used as spacer
label4.pack()

# create click button, put on frame2
button = tk.Button(frame2, text = 'ENTER', command = temp_fn)
button.pack()

# create frames to create child panes
parent.add(frame1)
parent.add(lblframe)
parent.add(frame2)
```

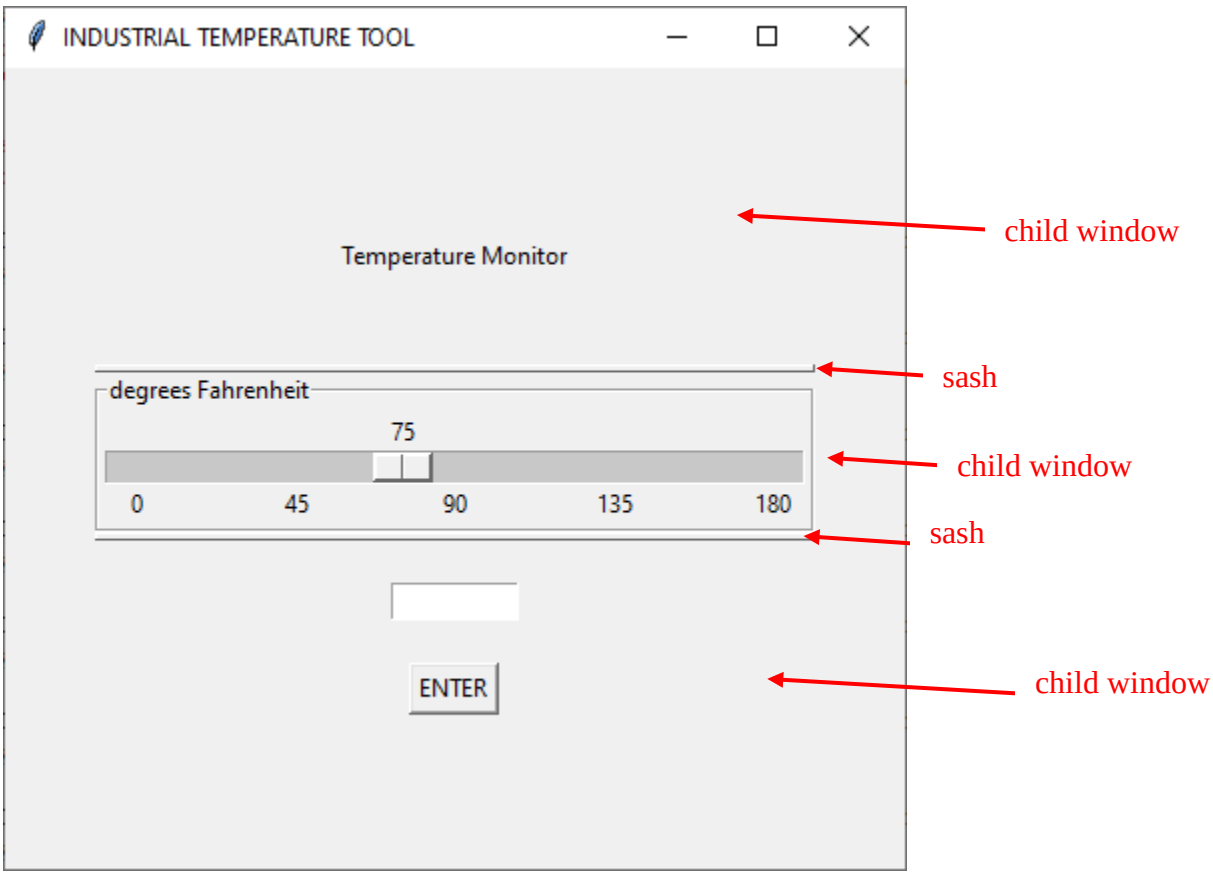
Ln: 42 Col: 38

At the top of the script add the code for the *temp_fn* function that the Button command option is set to.

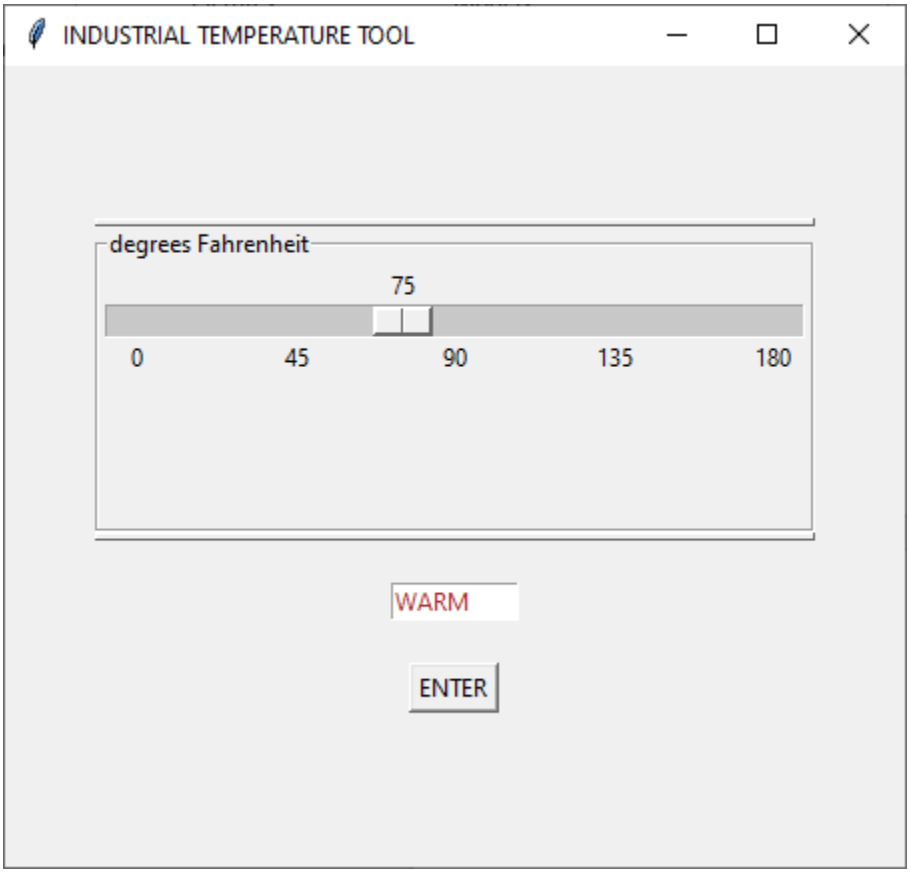


Save the file.
Run the file.

The app opens.



Hover over the upper positioned sash.
Notice the cursor change to a two-headed arrow.
With the two-headed arrow cursor, click on the sash and hold down.
Drag the sash up or down.
Observe the resizing of the panes as you do so.
Repeat for the lower positioned sash.
Play around with the sashes to reconfigure the GUI.
Run a calculation.



Successful completion!

8. THE COMBOBOX WIDGET

8.1 Combobox

A Combobox is a combination of an Entry widget and drop-down menu. When the user is not interacting with the widget, only one item, the current Entry content, is visible. When the user clicks on the drop-down arrow button, the other menu items become visible and the user may make a selection. A selection is made by clicking on the item. The item then replaces the existing item in the Entry and the menu closes.

The *tkinter* Combobox widget is in the *ttk* module of *tkinter*. The *ttk* module must be imported in order to use the widget. The syntax to create a Combobox is of the form,

< variable > = Combobox (*< master>, values = [<item>, <item>, <item>,…] {, < options >}*)

where
< variable > is a variable name that the widget is assigned to
< master > is the name of the variable the main GUI window has been assigned to, or the full reference to the main window
values is an option (a variable) set to a *Python* list - a comma-separated list of items enclosed in straight brackets (*[ ]*), these are the items that appear in the drop-down list
< options > are optional, attributes of the widget

The general widget options as well as those specific to the Entry widget, the Listbox widget, and the Menu are generally applicable to the Combobox widget. Some of the commonly applied Combobox widget options are summarized in Table 8.1.

Table 8. 1: Combobox widget options

Option	Description	Values
justify	controls the alignment of text	LEFT, RIGHT, CENTER
height	specifies the height of the list of items, in rows	
postcommand	set to a function or script that is called immediately before displaying the values	
state	controls whether the widget is enabled or disabled, or whether the list can be edited by the user	NORMAL – the text field is editable by the user, READONLY – the user cannot edit and may only select a value as-is, DISABLED – the widget is disabled completely
values	specifies the list of values to display in the drop-down listbox.	
width	specifies the width of the entry window, in number of characters, at the current font..	

8.2 Combobox widget methods

Some of the commonly used Combobox methods are summarized in Table 8.2.

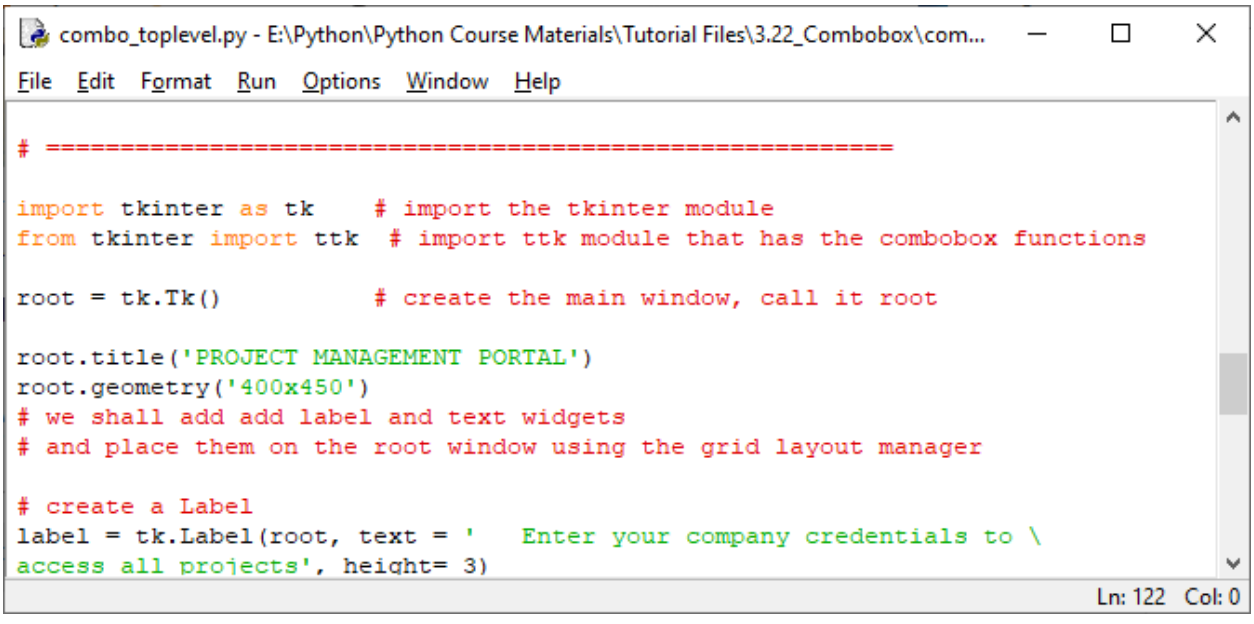
Table 8. 2: Combobox widget methods

Method	Description
current(<index>)	displays the item at the specified index, if not used, default is for Combobox to display item at index[0]
get( )	returns the current value of the Combobox
set(<value>)	sets the value of the combobox to the value specified

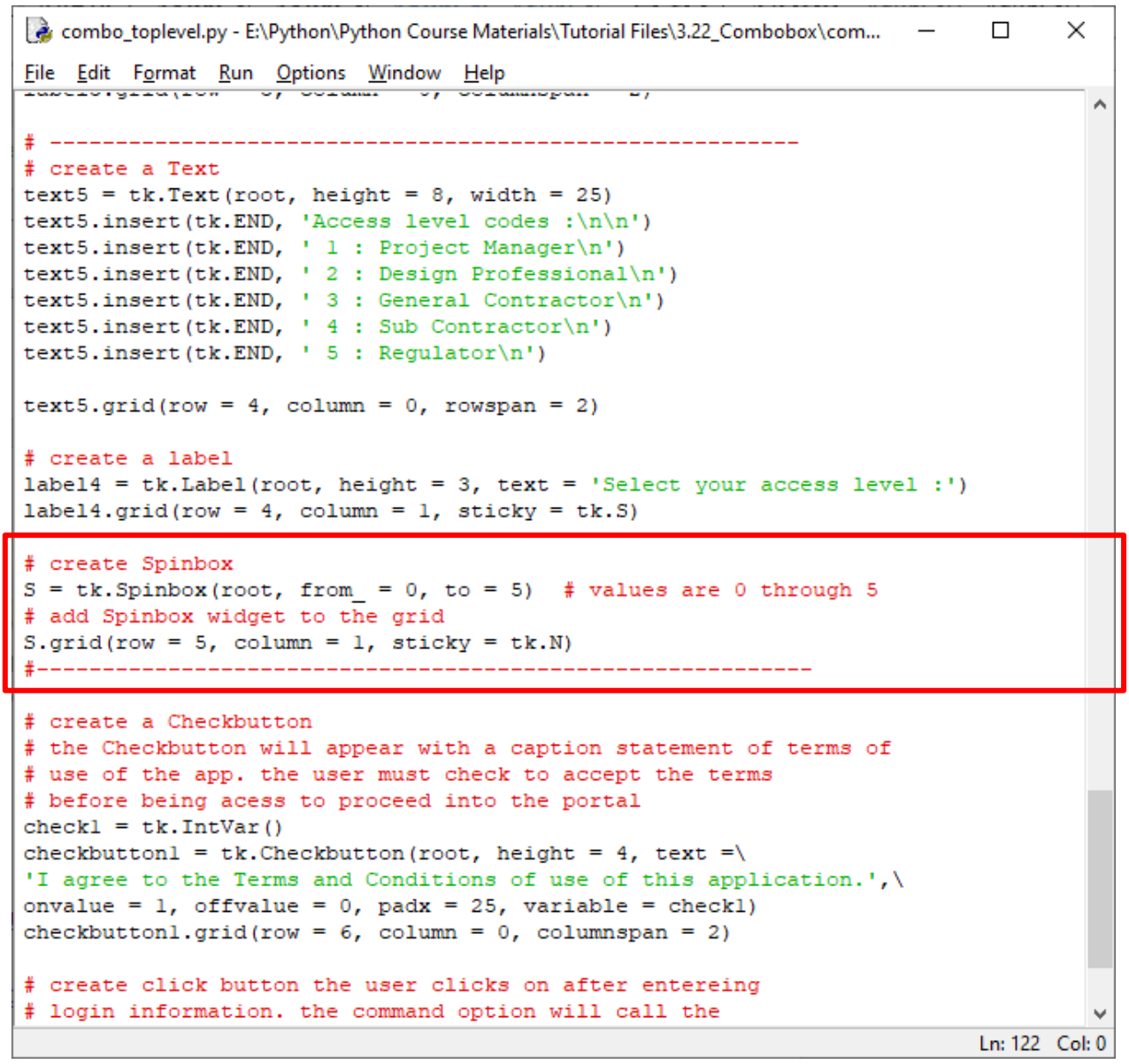
8.3 Combobox Example

In this exercise, we shall modify the application we developed in Chapter 6 (Toplevels). Specifically, we shall replace the Spinbutton with a Combobox for a user to select their access level before proceeding through the project management portal.

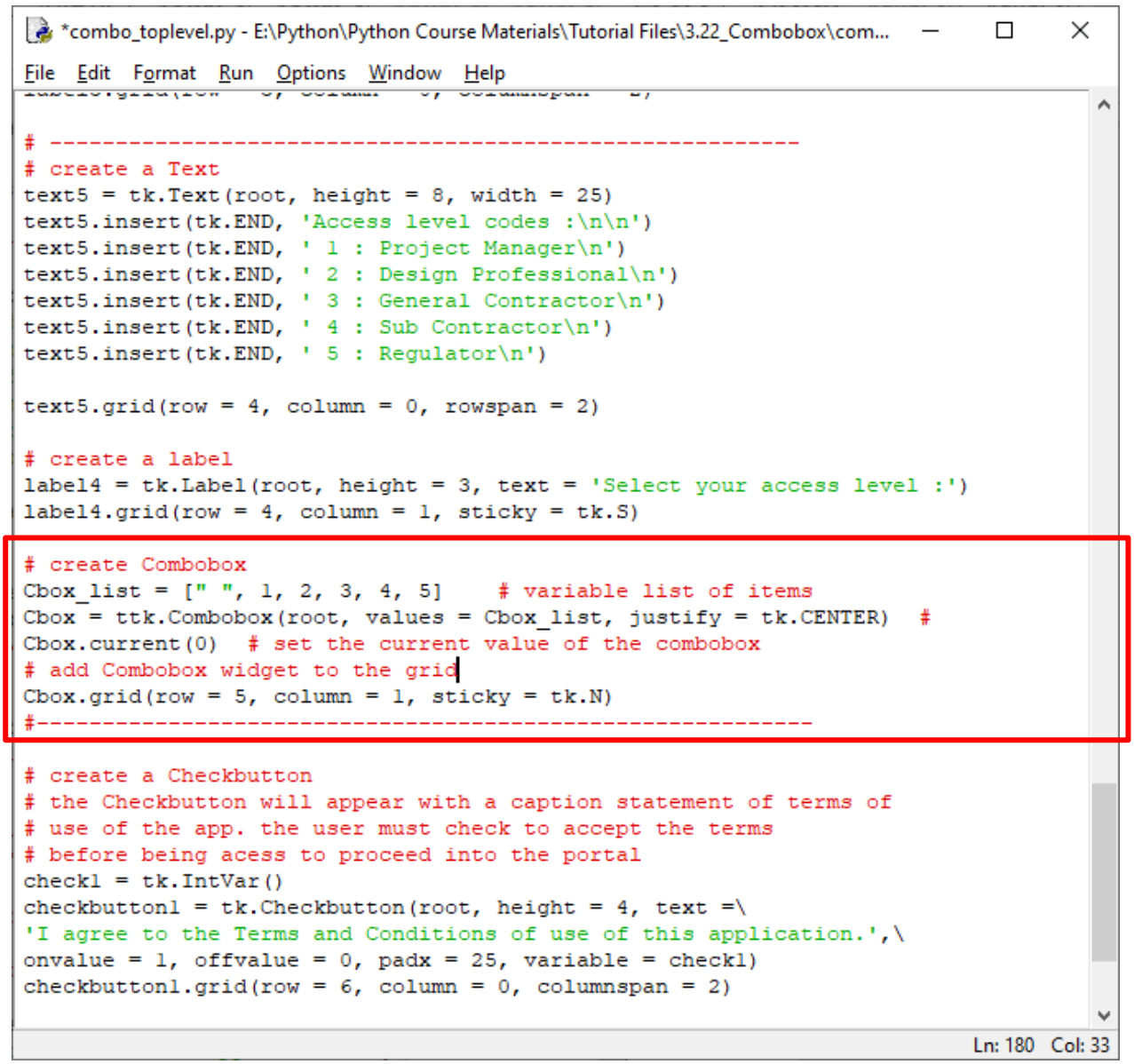
Open your code file that you worked on in Chapter 6 (Toplevels), and conduct the following updates.



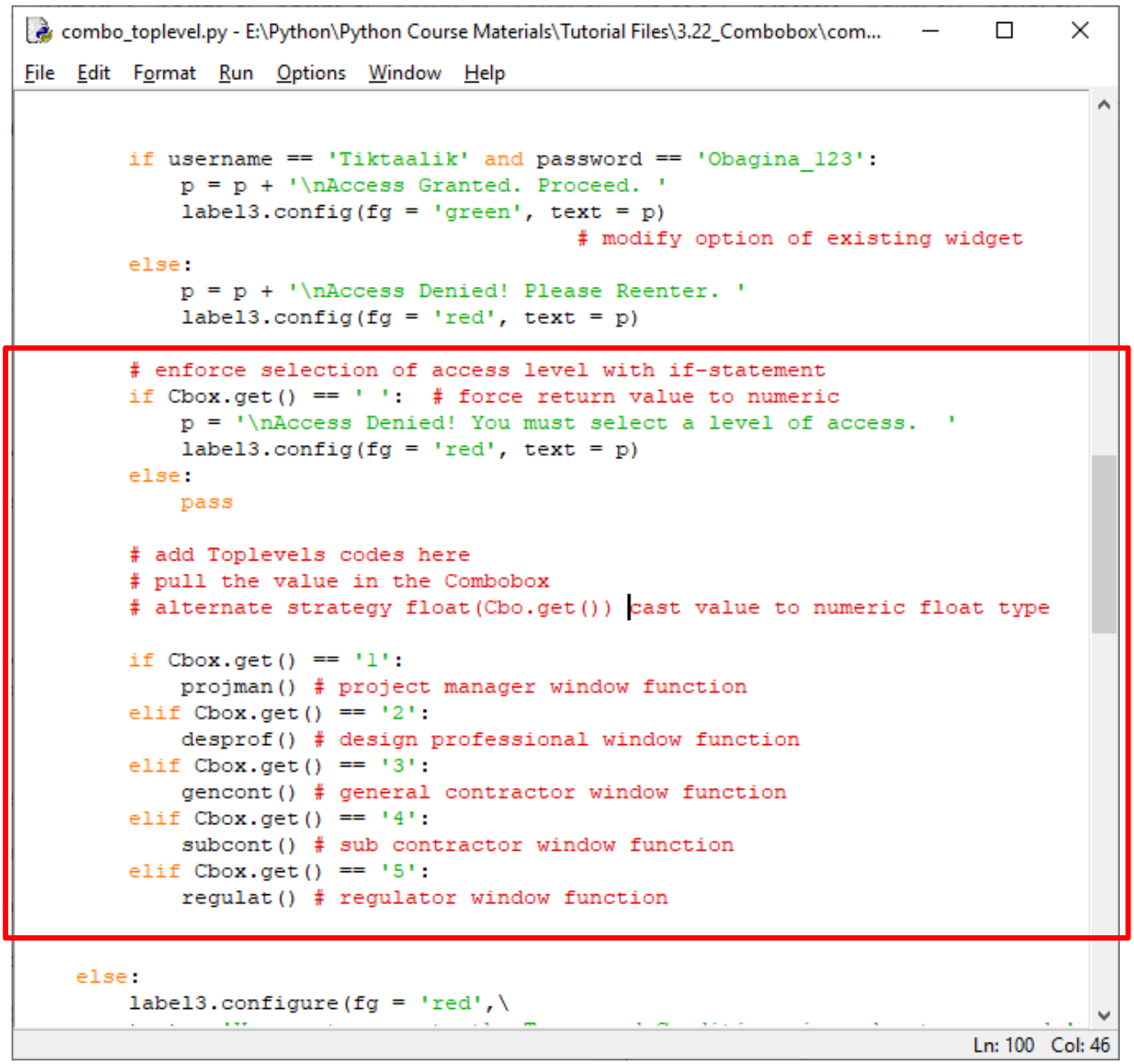
Find the Spinbox code.



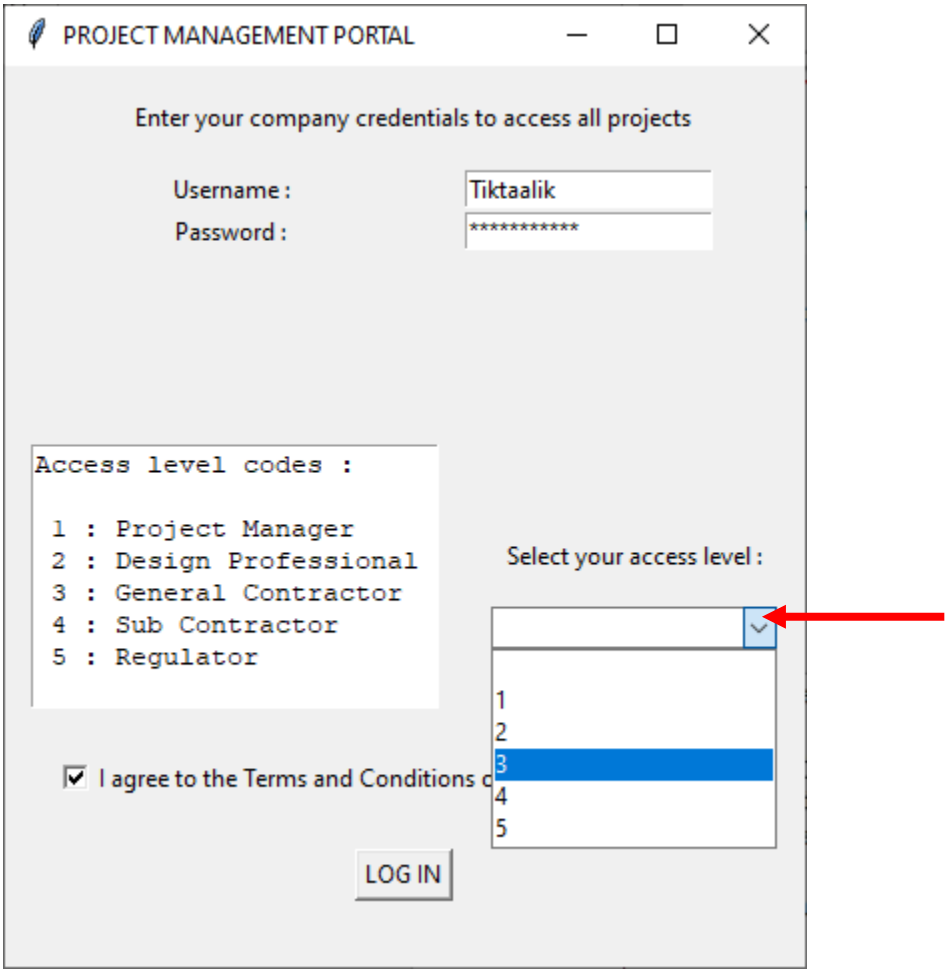
Replace the Spinbox code with the following to implement a Combobox.



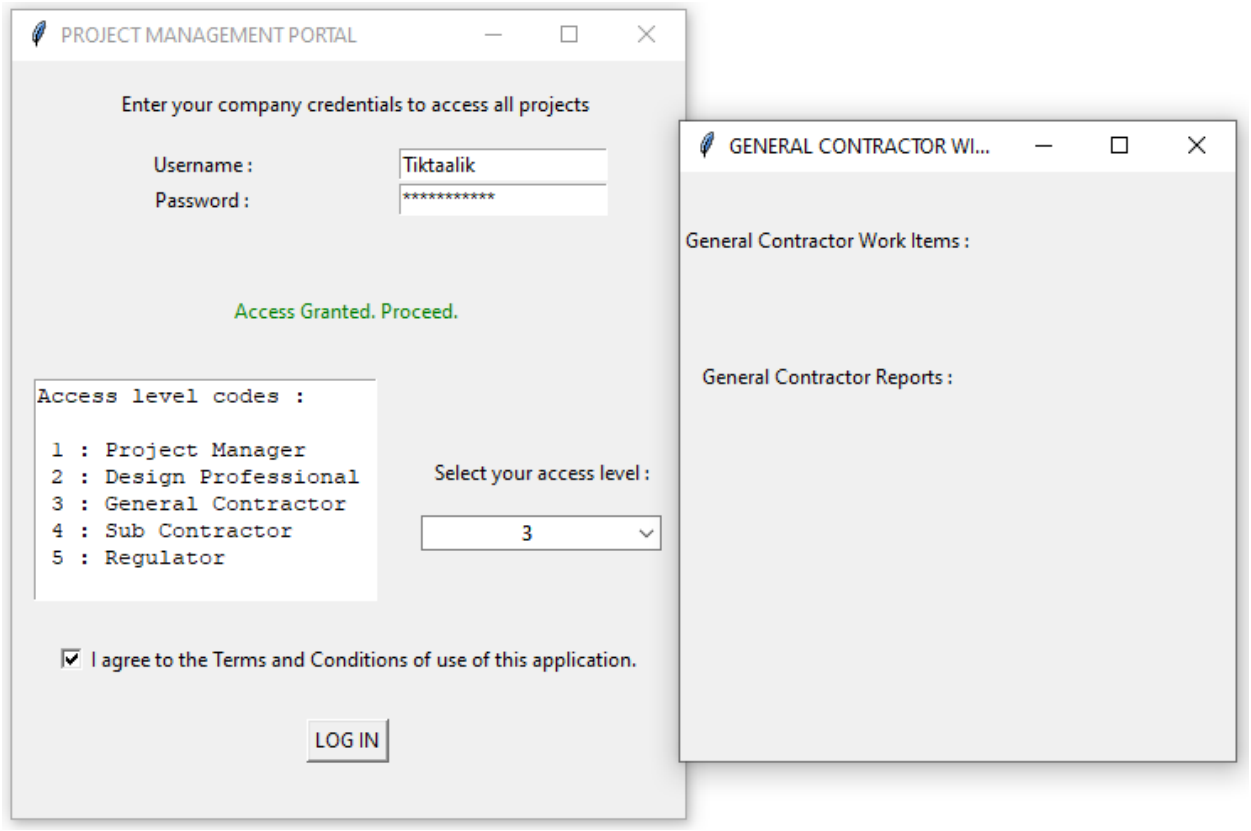
And now update the Button *command* option *login( )* function's *if* statement.



Save the file.
Run the file.
Enter the login information.
Accept the Terms and Conditions.
Click on the Combobox drop-down button.



Select an item from the Combobox list.
Click on the ***LOG IN*** button
The selected Window Toplevel opens.

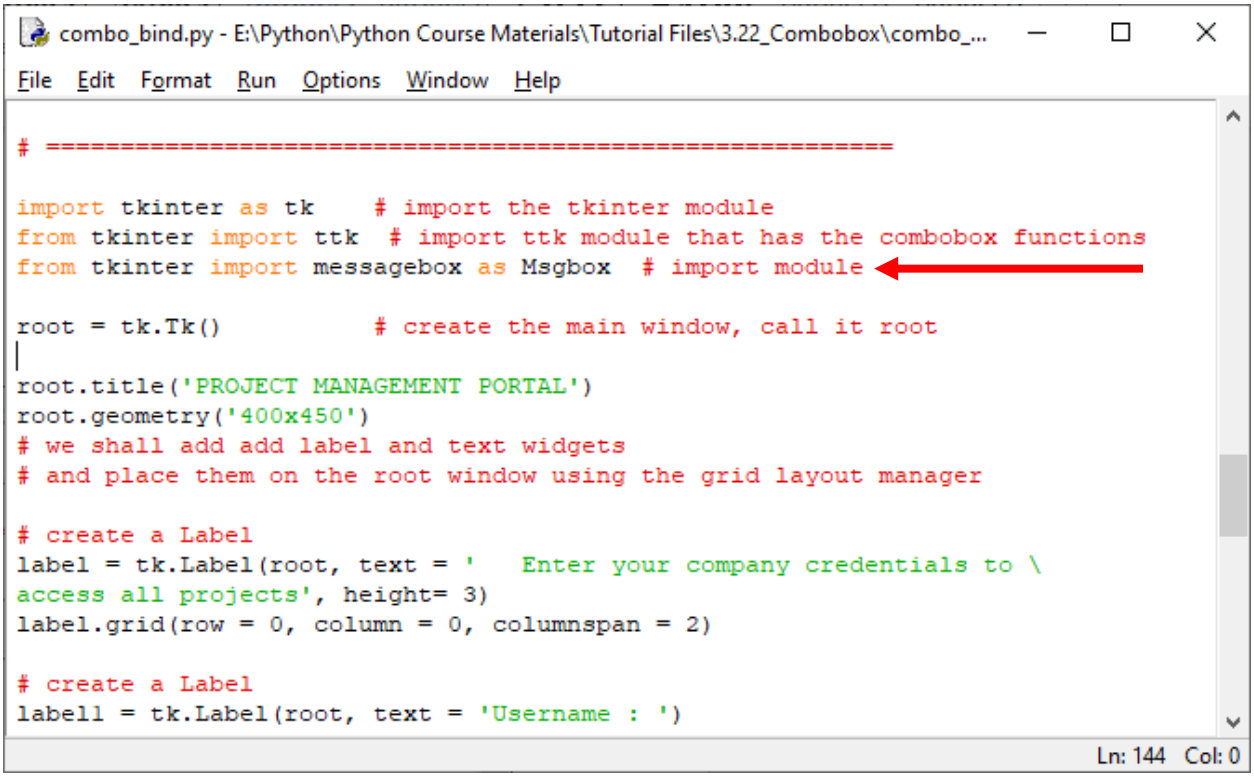


Close the Project Manager Window.

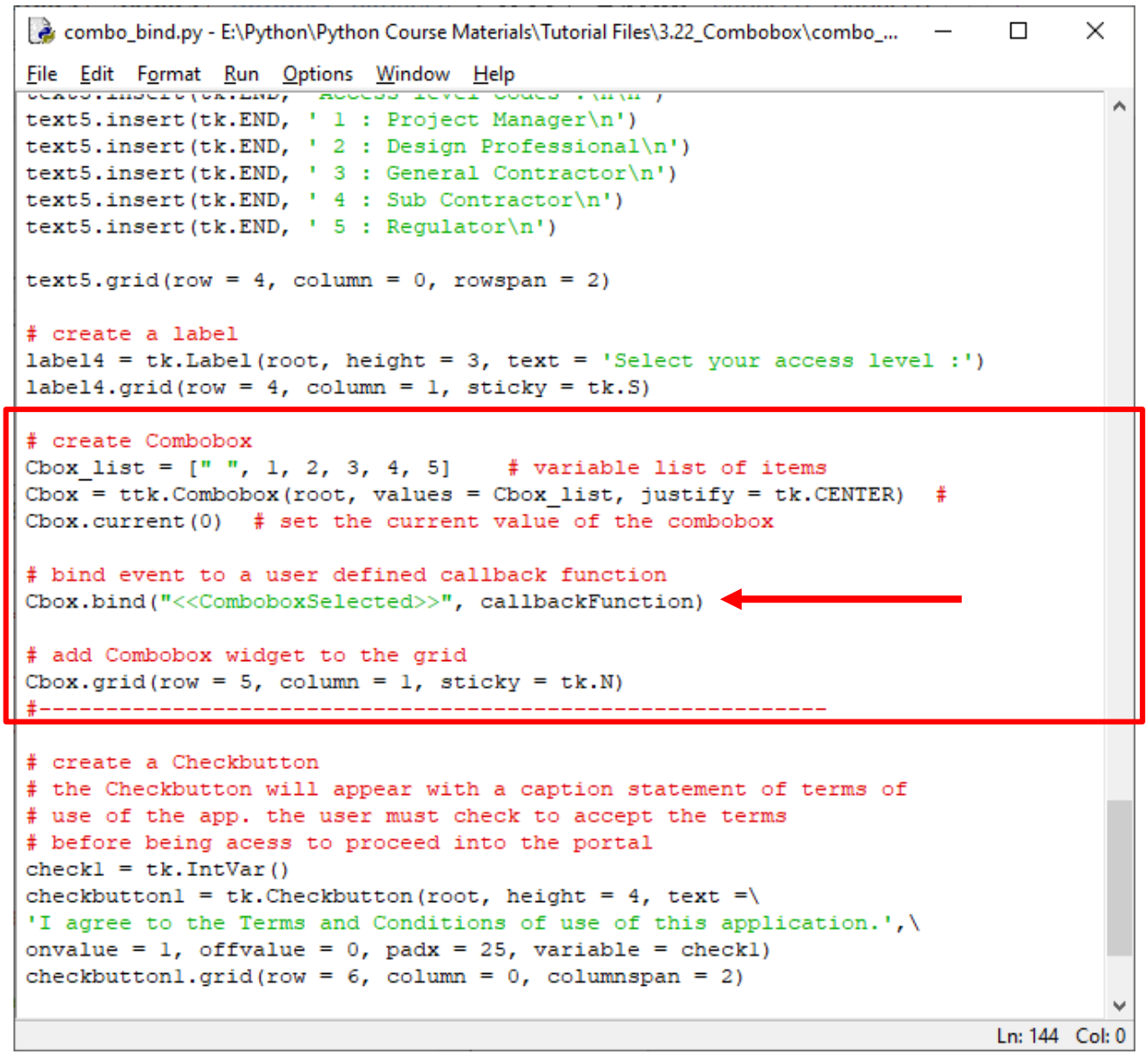
Let us now learn a unique feature of the Combobox.
In many applications, it is desirable that some event is triggered (or some code is executed) immediately a Combobox item is selected. This is implemented by binding (*bind( )*) the Combobox callback function the Combobox virtual event when the user selects an item in the drop-down list.

In the following example we shall have a pop up appear once an item is selected from the Combobox list.

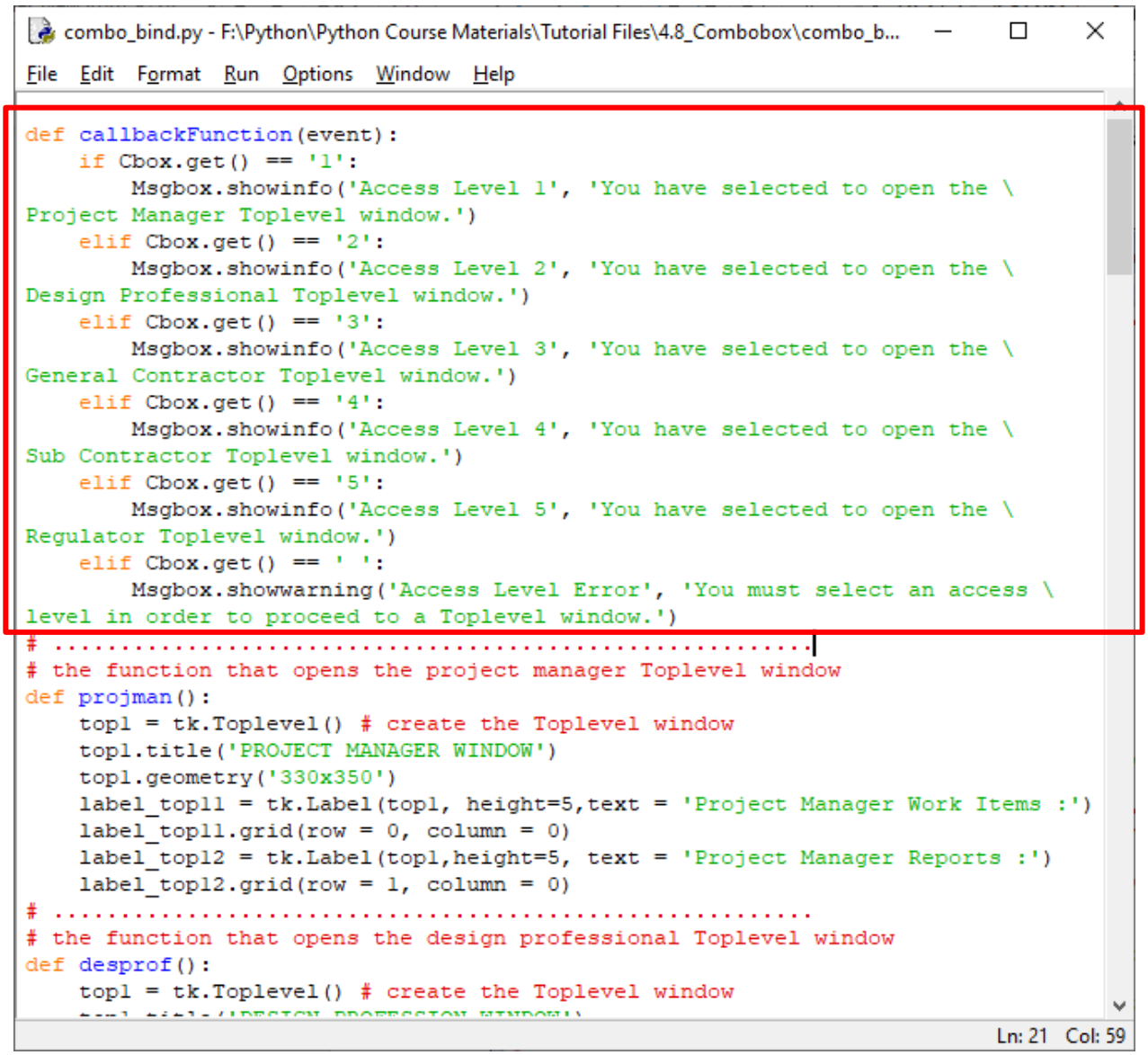
Reopen the code.
Add the following.



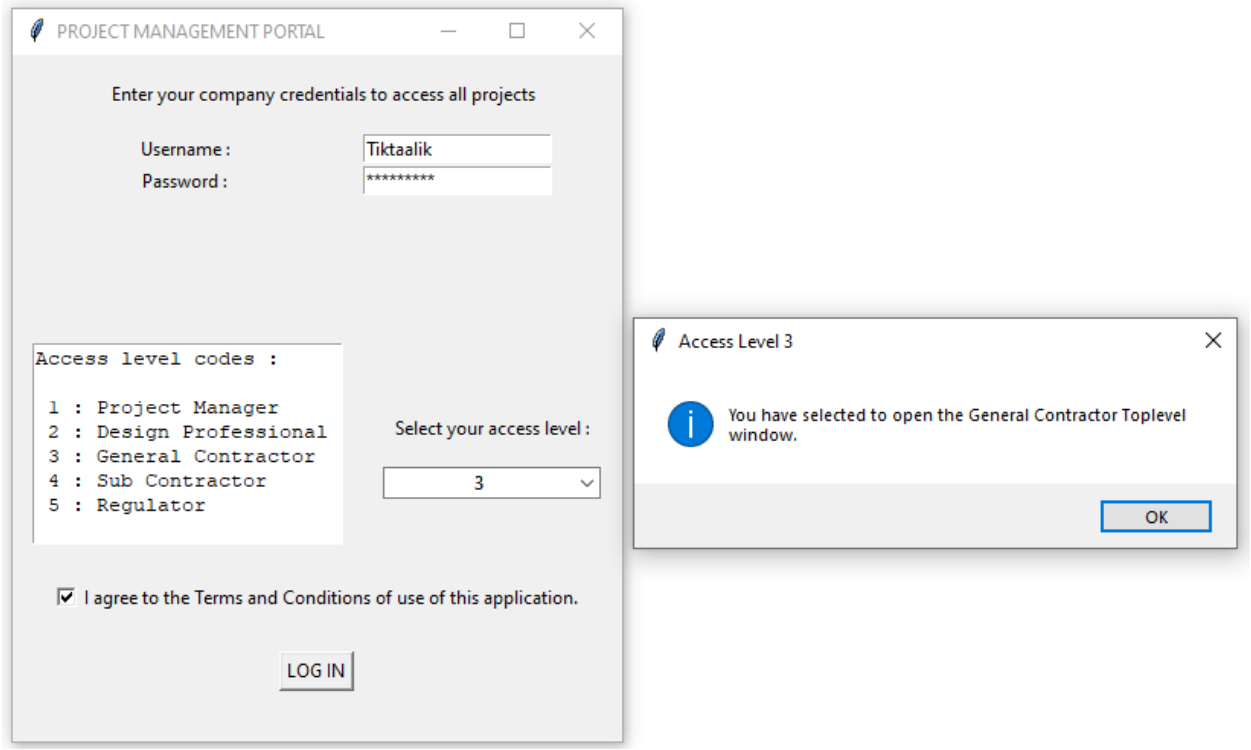
Update the Combobox code as follows.



At the top of your code window add the code to define the callback function, as follows.

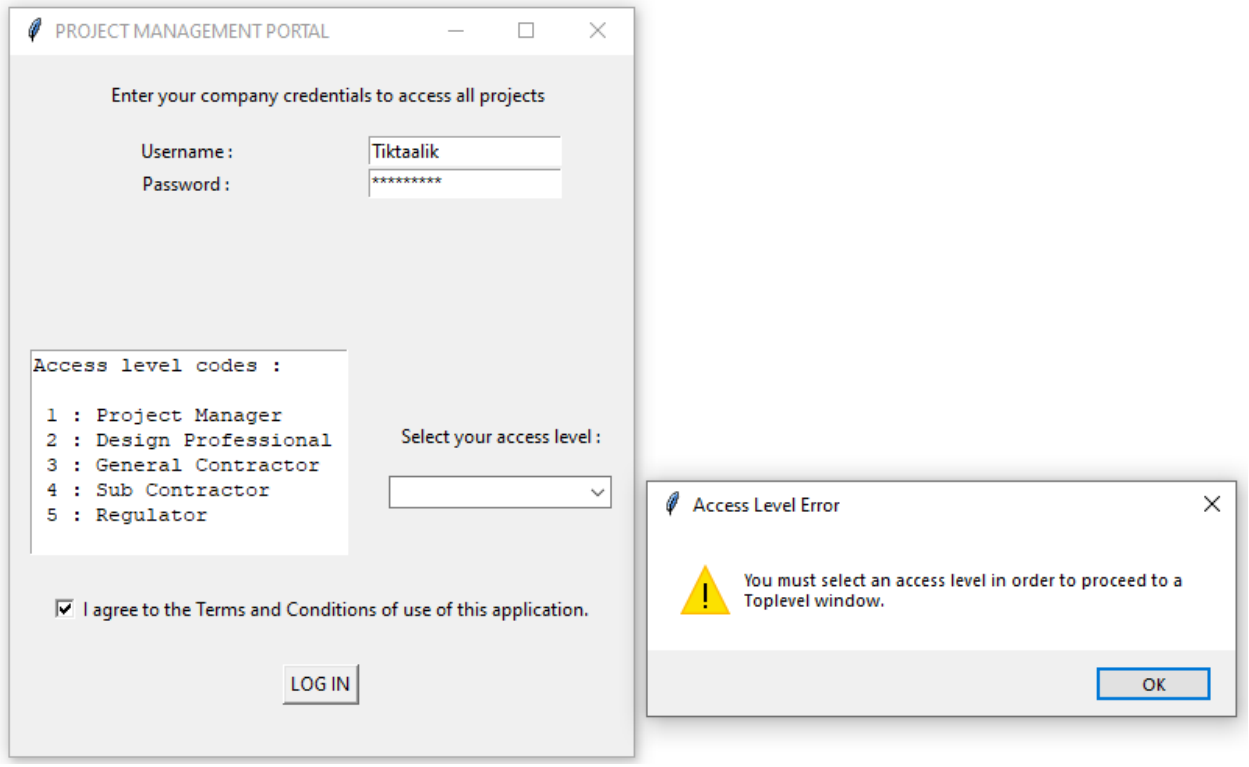


Save the file.
Run the file.
Enter the login information.
Accept the Terms and Conditions.
Select an access level in the Combobox.
A tkmessageBox opens immediately.



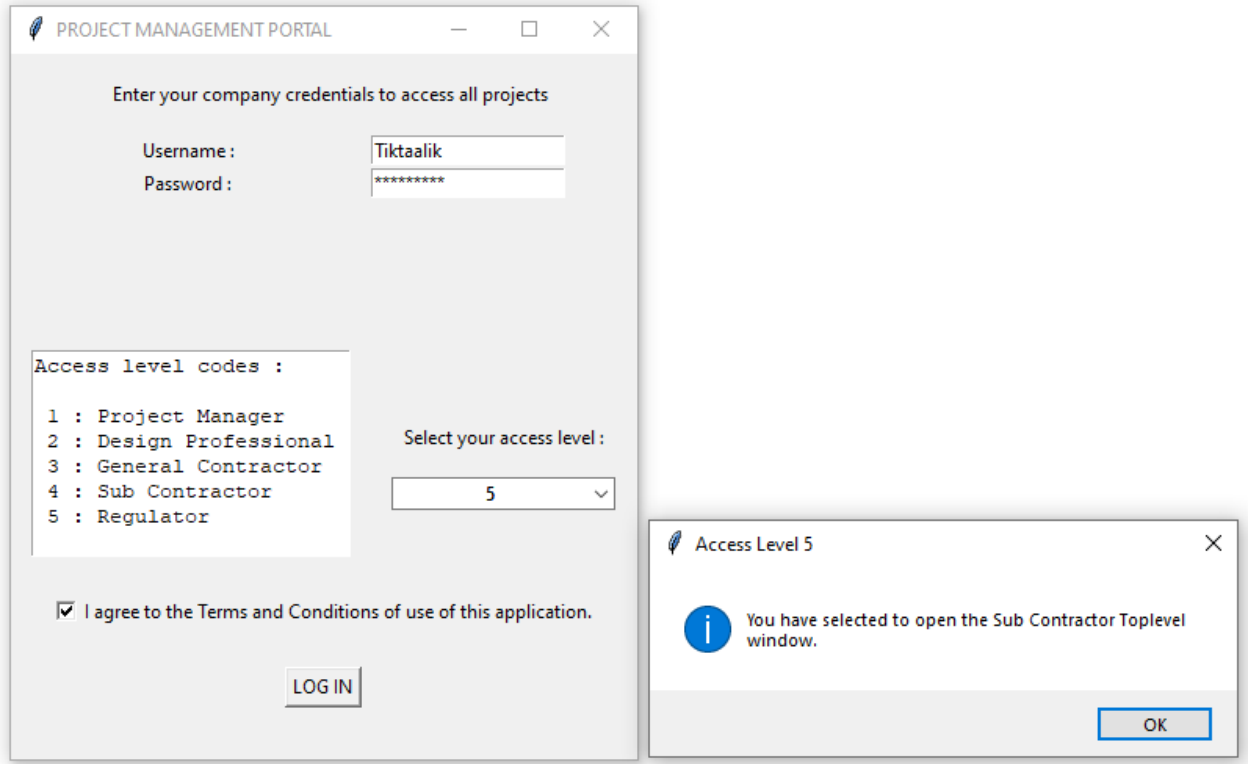
Click on OK to dismiss the tkMessageBox.

Test another access level.

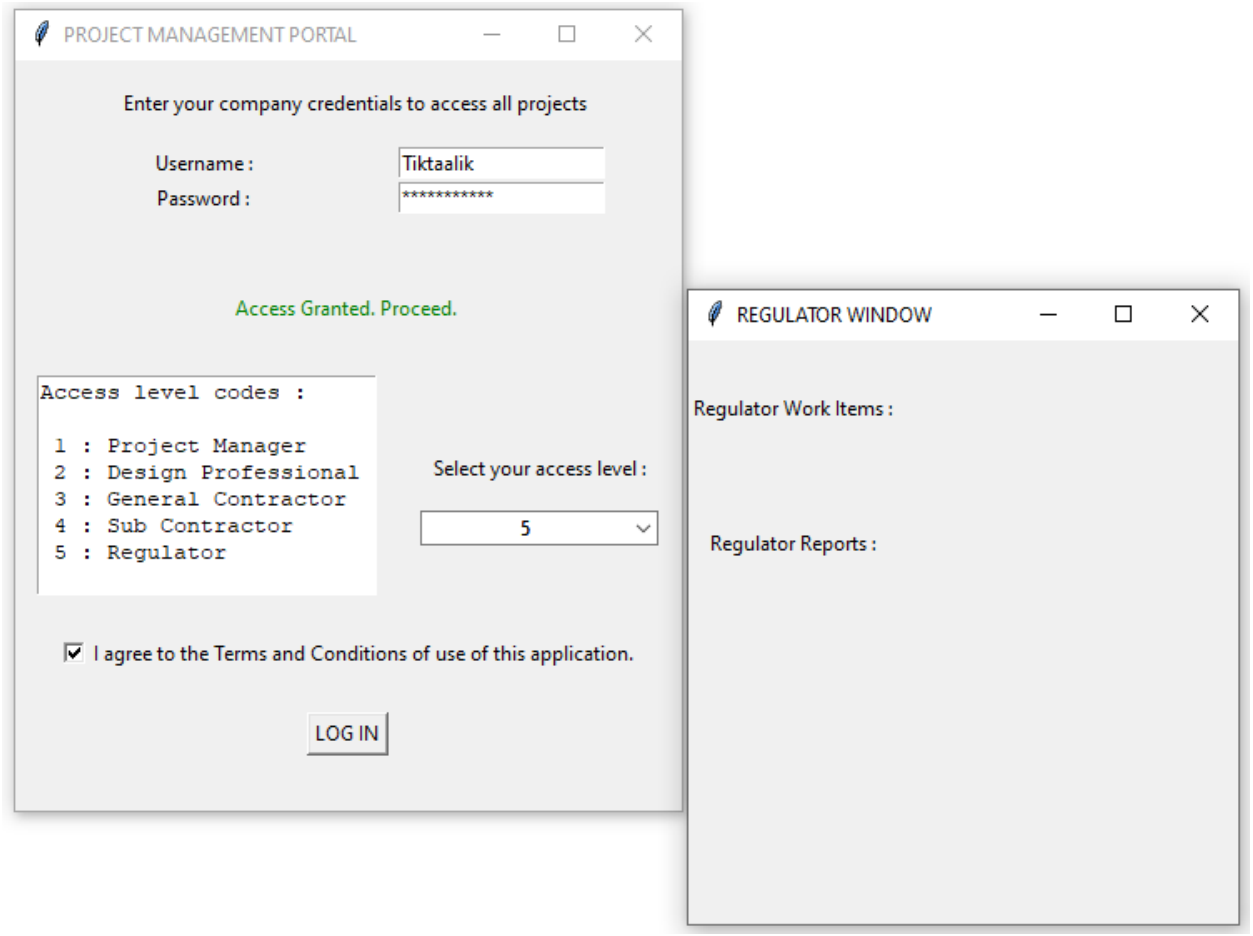


Click on OK to dismiss the tkMessageBox.

Test one more access level.



Click on OK to dismiss the tkMessageBox.
On the project management portal, click ***LOG IN***.
The relevant Toplevel window opens.



Successful completion!

9. THE SIMPLEDIALOG WIDGET

9.1 simpledialog

A simpledialog widget is a pop up that enables an executing program to interact with the user. A simpledialog widget consists of a text field (entry widget) and buttons such as on OK button and a Cancel button. When the simpledialog widget pops up, the program execution pauses, and the user may supply some input data to the program through the simpledialog widget's text field. Upon supplying the data and clicking the OK button, the data is passed to the simpledialog widget's control variable which is then processed by the program. If the Cancel button is clicked on the data input process is aborted, the widget's control variable does not change its value and no new data is supplied to the program.

The *tkinter* simpledialog widget is in the *simpledialog* module of *tkinter*. The *simpledialog* module must be imported in order to use the widget. The syntax for the creation of a simpledialog is of the form,

< var > = simpledialog.FunctionName (< title > , < prompt > [, < options >])

where
FunctionName is the specific type of simpledialog widget
< title > is the title caption displayed in the header bar of the widget
< prompt > is the main text message displayed on the dialog box
< options > optional attributes that are set to customize the dialog box

As with any kind of pop up box, a simpledialog is typically called via a function in the main program if the user interacts with the program in a certain way.

9.2 FunctionName

This is the type of the simpledialog. When the simpledialog opens, an icon of the relevant type is displayed on the widget. The available options are summarized in Table 9.1

Table 9. 1: simpledialog FunctionName functions

FunctionName	Description
askstring ( )	user input is returned as a string upon clicking OK, returns None upon clicking Cancel
askinteger ( )	user input is returned as an integer upon clicking OK, returns None upon clicking Cancel
askfloat ( )	user input is returned as a float upon clicking OK, returns None upon clicking Cancel

9.3 simpledialog Example

In this example we shall use simpledialogs to implement a login process where a username and password must be correctly entered to gain access to an account.

Open a new session of IDLE (Python GUI).
Click on **File**.
Click on **New File**, to open the File Editor.

Replicate the following code.

dialogs.py - E:/Python/Python Course Materials/Tutorial Files/dialogs.py (3.7.2)

File Edit Format Run Options Window Help

```
# ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

import tkinter as tk
from tkinter import simpledialog as Dialog
from tkinter import messagebox as Msgbox


root = tk.Tk()  # create the main window, call it root

root.title('SIGN IN')
root.geometry('300x350')

# create a spacer label
label2 = tk.Label(root, height = 4, text = ' ')
label2.pack()

# button to click on to sign in
button = tk.Button(root, text = 'Sign In', command = signin)
button.pack()


root.mainloop()
```

Ln: 73 Col: 0

At the top of your code window replicate the following code for the *signin( )* function.

dialogs.py - E:/Python/Python Course Materials/Tutorial Files/dialogs.py (3.7.2)

File Edit Format Run Options Window Help

```
# ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
# define signin function
def signin():

    # open dialoguebox, user entry will save to variable x
    x = Dialog.askstring('USERNAME', 'Enter your username.')
    y = Dialog.askstring('PASSWORD', 'Enter your password.')

    # use if statements to verify username
    if x == 'Tiktaalik':
        pass
    else:
        Msgbox1 = Msgbox.showwarning('INCORRECT USERNAME',\
                            'Please enter the correct username.')

    if y == 'Obagina_123':
        pass
    else:
        Msgbox2 = Msgbox.showwarning('INCORRECT PASSWORD',\
                            'Please enter the correct password.')

    if x == 'Tiktaalik' and y == 'Obagina_123':
        open_win()
    else:
        pass

# ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

import tkinter as tk
from tkinter import simpledialog as Dialog
from tkinter import messagebox as Msgbox


root = tk.Tk()  # create the main window, call it root
```

Ln: 47 Col: 0

At the top of your code window replicate the following code for the *open_win( )* function.

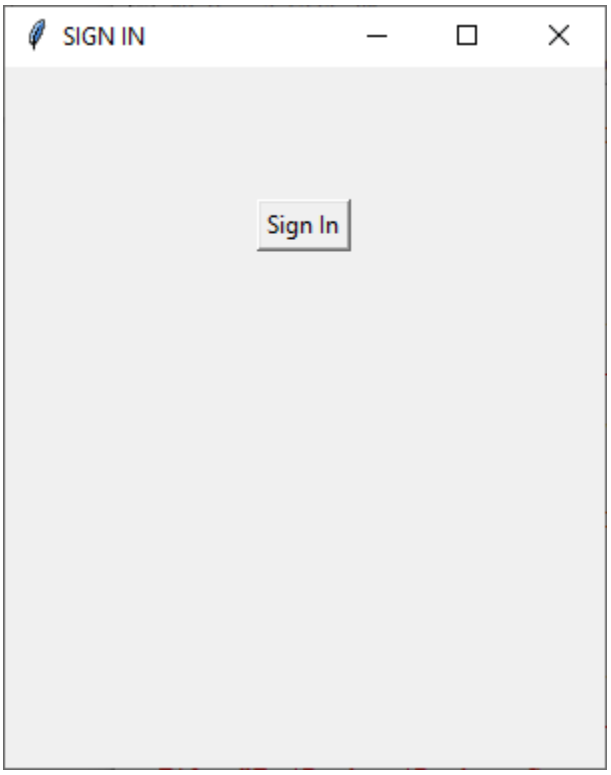
dialogs.py - E:/Python/Python Course Materials/Tutorial Files/dialogs.py (3.7.2)

File Edit Format Run Options Window Help

```
# ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
# define open_win
def open_win():
    top1 = tk.Toplevel() # create the Toplevel window
    top1.title('ACCOUNT MAIN')
    top1.geometry('270x320')
    label_top11 = tk.Label(top1, height=5,text =\
                           '    Welcome to your account.')
    label_top11.grid(row = 0, column = 0)
    label_top12 = tk.Label(top1,height=5, text =\
                           'Please review your account balances.')
    label_top12.grid(row = 1, column = 0)

# ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
# define signin function
def signin():

    # open dialoguebox, user entry will save to variable x
    x = Dialog.askstring('USERNAME', 'Enter your username.')
    y = Dialog.askstring('PASSWORD', 'Enter your password.')

    # use if statements to verify username
    if x == 'Tiktaalik':
        pass
    else:
        Msgbox1 = Msgbox.showwarning('INCORRECT USERNAME',\
                           'Please enter the correct username.')

    if y == 'Obagina_123':
        pass
    else:
        Msgbox2 = Msgbox.showwarning('INCORRECT PASSWORD',\
                           'Please enter the correct password.')

    if x == 'Tiktaalik' and y == 'Obagina_123':
```

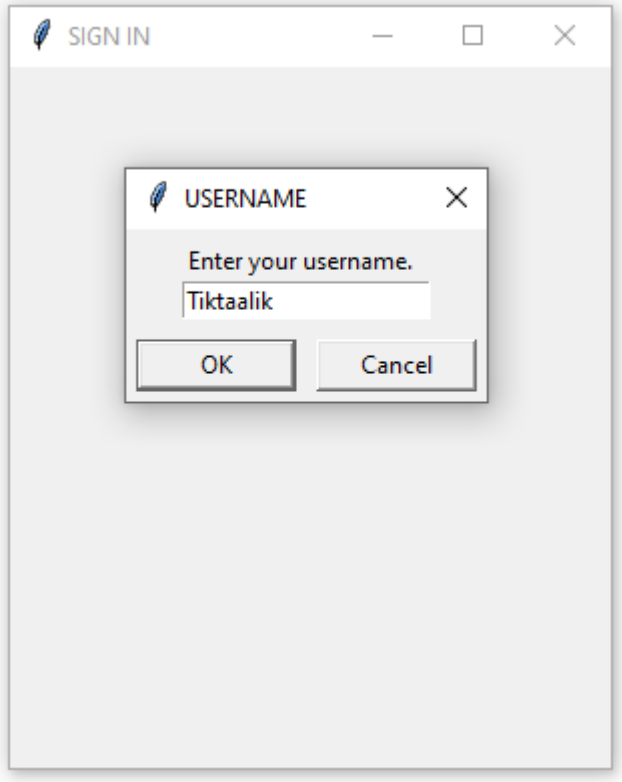
Ln: 73 Col: 0

Save the file.
Run the file.



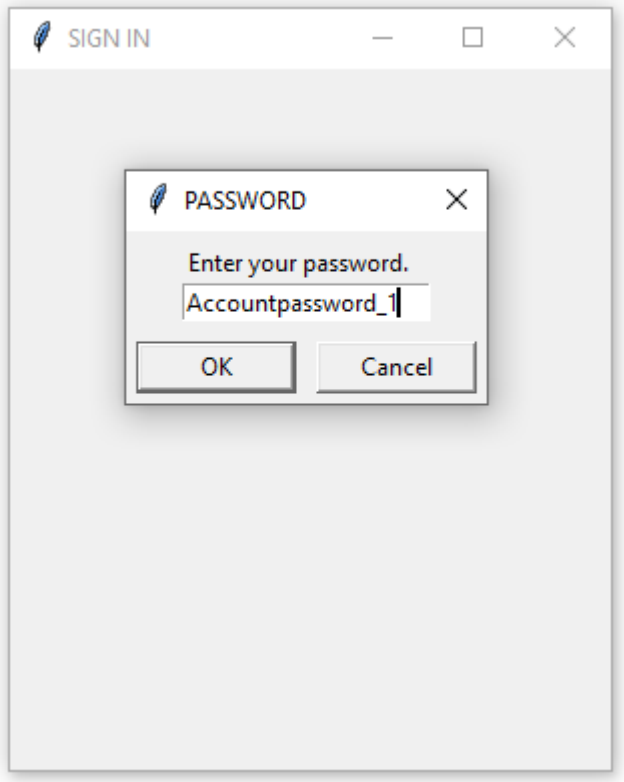
Click the ***Sign In*** button.

The username simpledialog opens.
Enter the correct username.



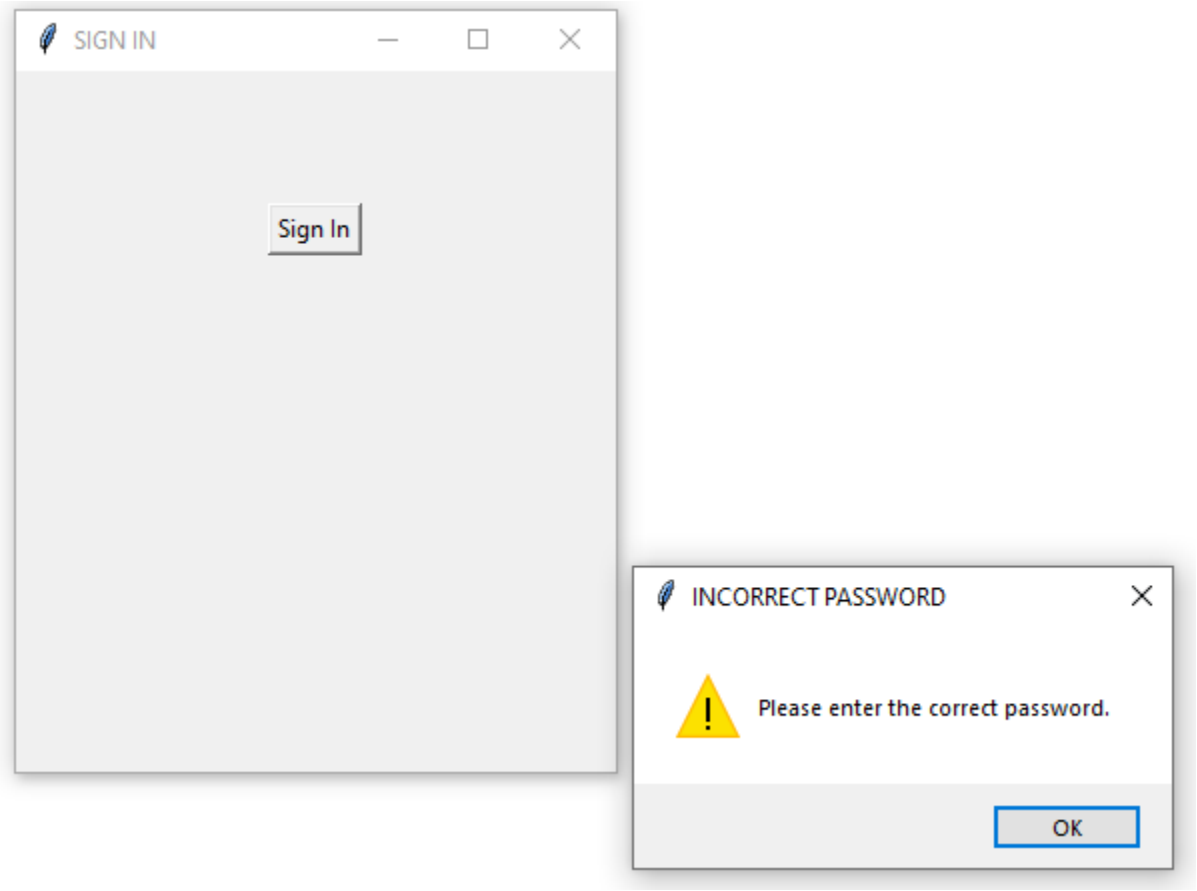
Click the ***OK*** button to dismiss the username simpledialog.
The password simpledialog opens.

Let us intentionally enter an incorrect password.



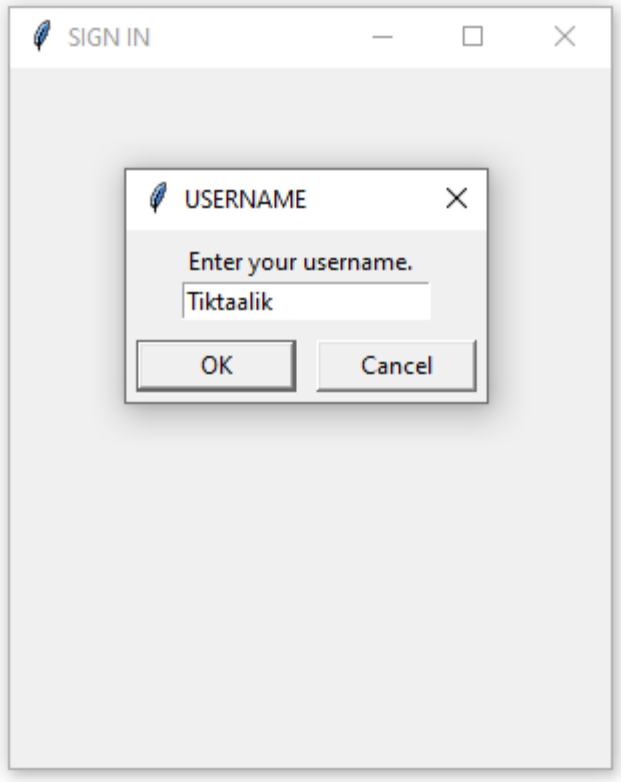
Click the ***OK*** button to dismiss the password simpledialog.

A tkMessageBox pops up.



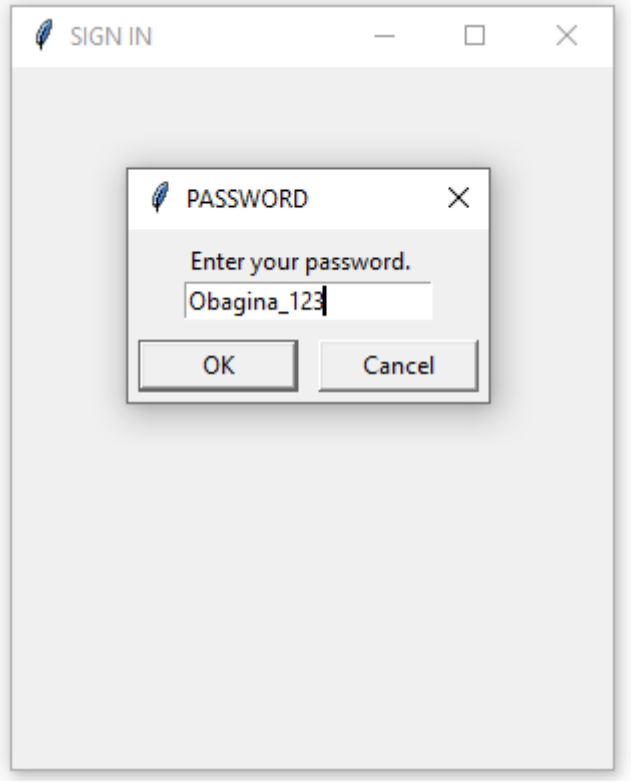
Click the ***OK*** button to dismiss the username tkMessageBox.

Let us start over.
On the root window, click on the ***Sign*** In button.
In the username simpledialog widget, enter the correct username.



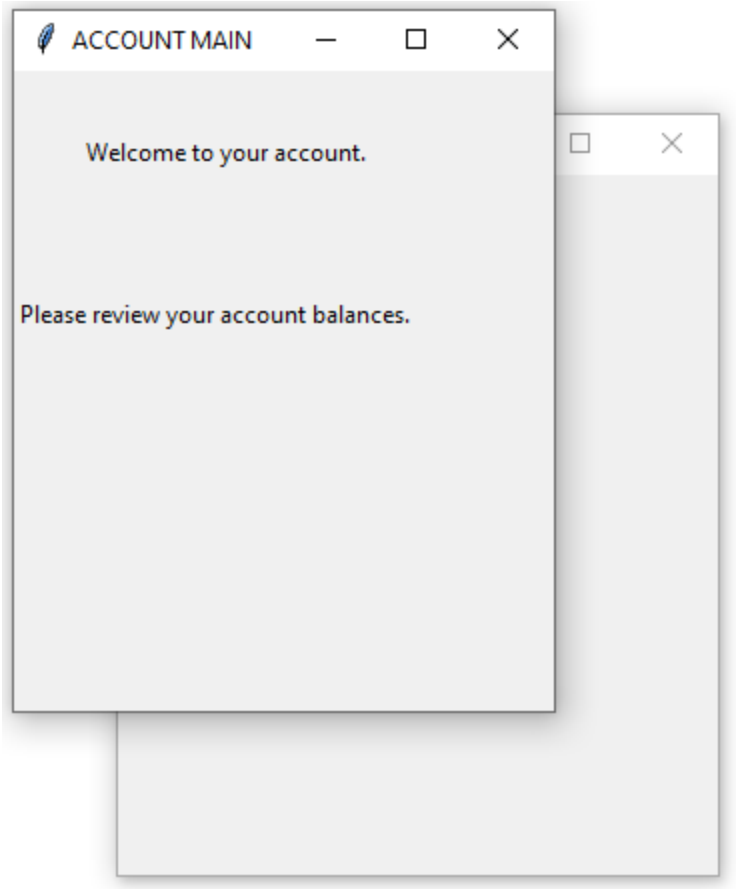
Click the ***OK*** button to dismiss the username simpledialog.
The password simpledialog opens.

Enter the correct password.

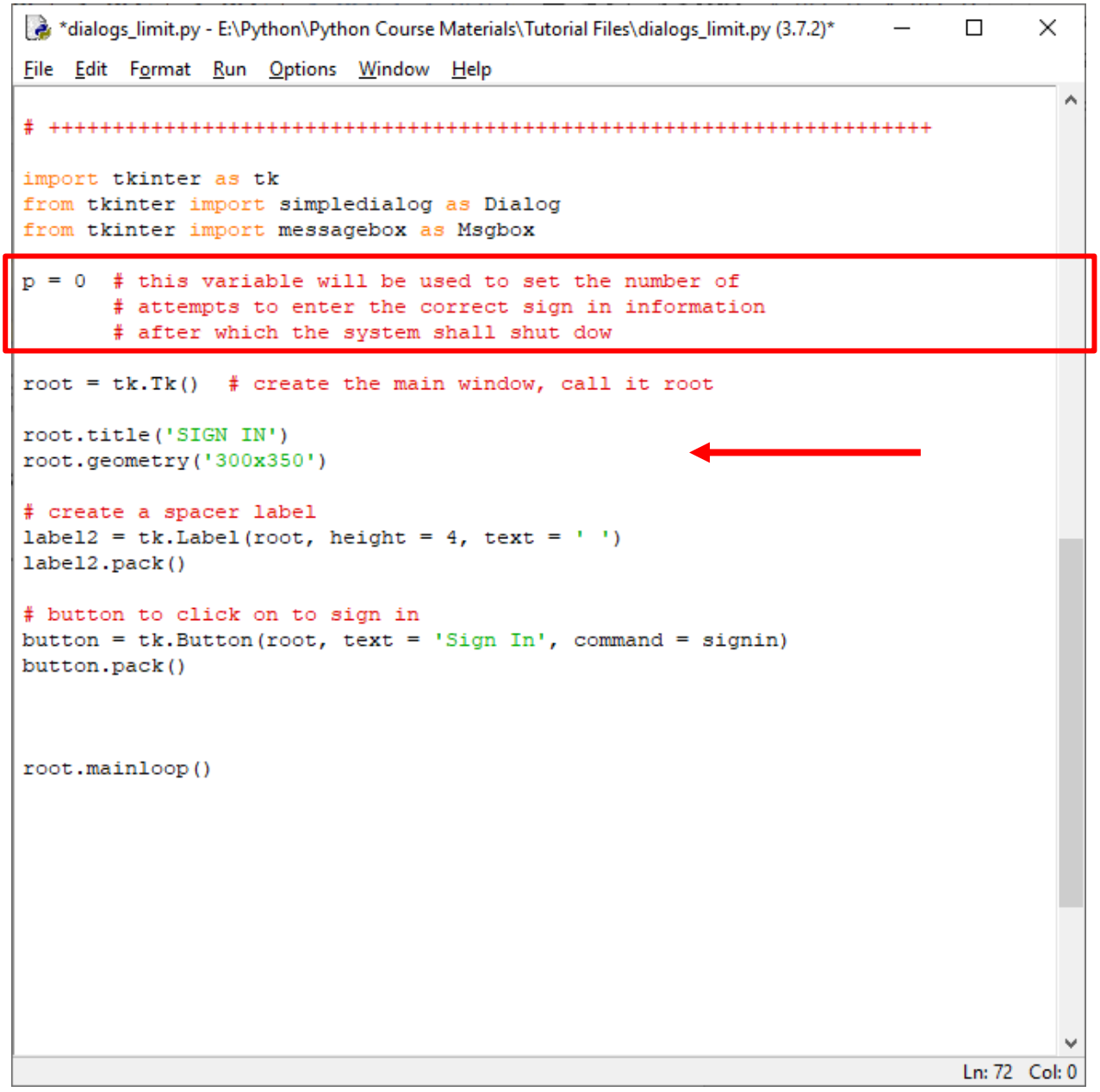


Click the ***OK*** button to dismiss the password simpledialog.

The Account Main window (Toplevel) opens.



Let us add a rule that the user has three (3) to correctly enter the sign in formation, if not, the system shall shut down.
Replicate the following code to implement the rule.



And in the *signin( )* function, add the following.

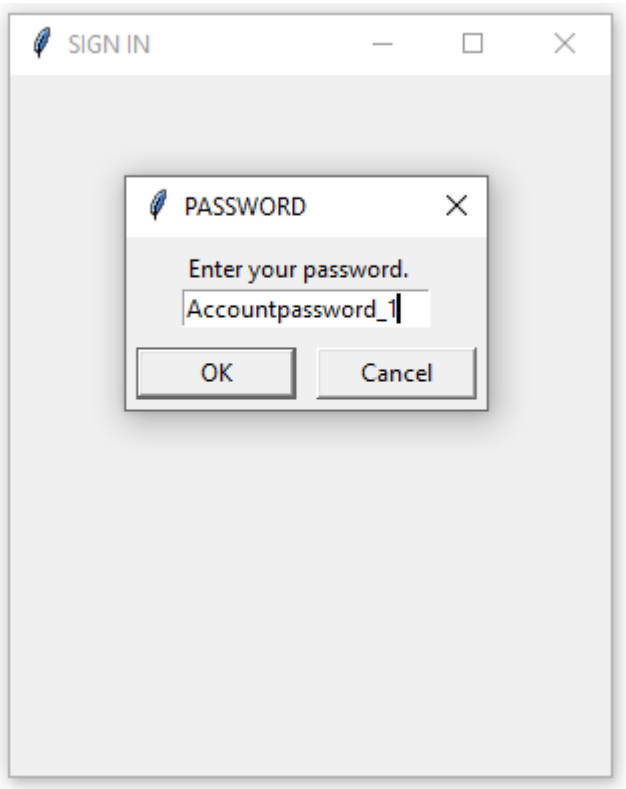
dialogs_limit.py - E:\Python\Python Course Materials\Tutorial Files\dialogs_limit.py (3.7.2)

File Edit Format Run Options Window Help

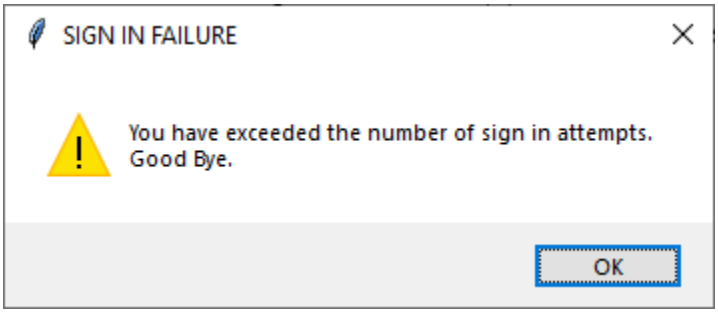
```
# ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
# define signin function
def signin():

    global p  # declare global variable in the local function

    # open dialoguebox, user entry will save to variable x
    x = Dialog.askstring('USERNAME', 'Enter your username.')
    y = Dialog.askstring('PASSWORD', 'Enter your password.')

    # use if statements to verify username
    if x == 'Tiktaalik':
        pass
    else:
        Msgbox1 = Msgbox.showwarning('INCORRECT USERNAME',\
                                     'Please enter the correct username.')

    if y == 'Obagina_123':
        pass
    else:
        Msgbox2 = Msgbox.showwarning('INCORRECT PASSWORD',\
                                     'Please enter the correct password.')

    if x == 'Tiktaalik' and y == 'Obagina_123':
        open_win()
    else:
        p = p + 1

    if p == 3:
        Msgbox3 = Msgbox.showwarning('SIGN IN FAILURE',\
                                     'You have exceeded the number of sign \
in attempts.\nGood Bye.')
        root.destroy()

# ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

import tkinter as tk
```

Ln: 72 Col: 0

Save the file.
Run the file.
Click on the Sign In button.
When prompted, enter an incorrect username or an incorrect password.



The tkMessageBoxes notify you of the incorrect errors accordingly.
Repeat this process of incorrect username or password two (2) more times.
You are now notified that you have exhausted the number of attempts allowed to correctly sign in.



Click on ***OK*** to dismiss the tkMessageBox.
The system shuts off completely.

Successful completion!

10. THE tkFILEDIALOG MODULE

10.1 tkFileDialog

The tkFileDialog module provides functions that enable the user to open the computer's operating system's file management application and browse or navigate through the folders to select a file to open or to save into a new file (Save As).

The syntax for the tkFileDialog function that enables a user to get an existing file name to open the file is of the form

askopenfilename ([< options >])

where
< *options* > optional attributes that are set to customize the dialog box

The syntax for the tkFileDialog function that enables a user to get a new file name to save-as is of the form

asksaveasfilename ([< options >])

where
< *options* > optional attributes that are set to customize the dialog box

It must be noted that the above tkFileDialog commands do not actually save or load the file. They simply enable the user to select or enter a file name. Once a file name is selected or entered, the file may be opened, closed, saved, etc., using the relevant file handling command(s).

The tkFileDialog functions' options are summarized in Table 10.1

Table 10. 1: tkFileDialog options

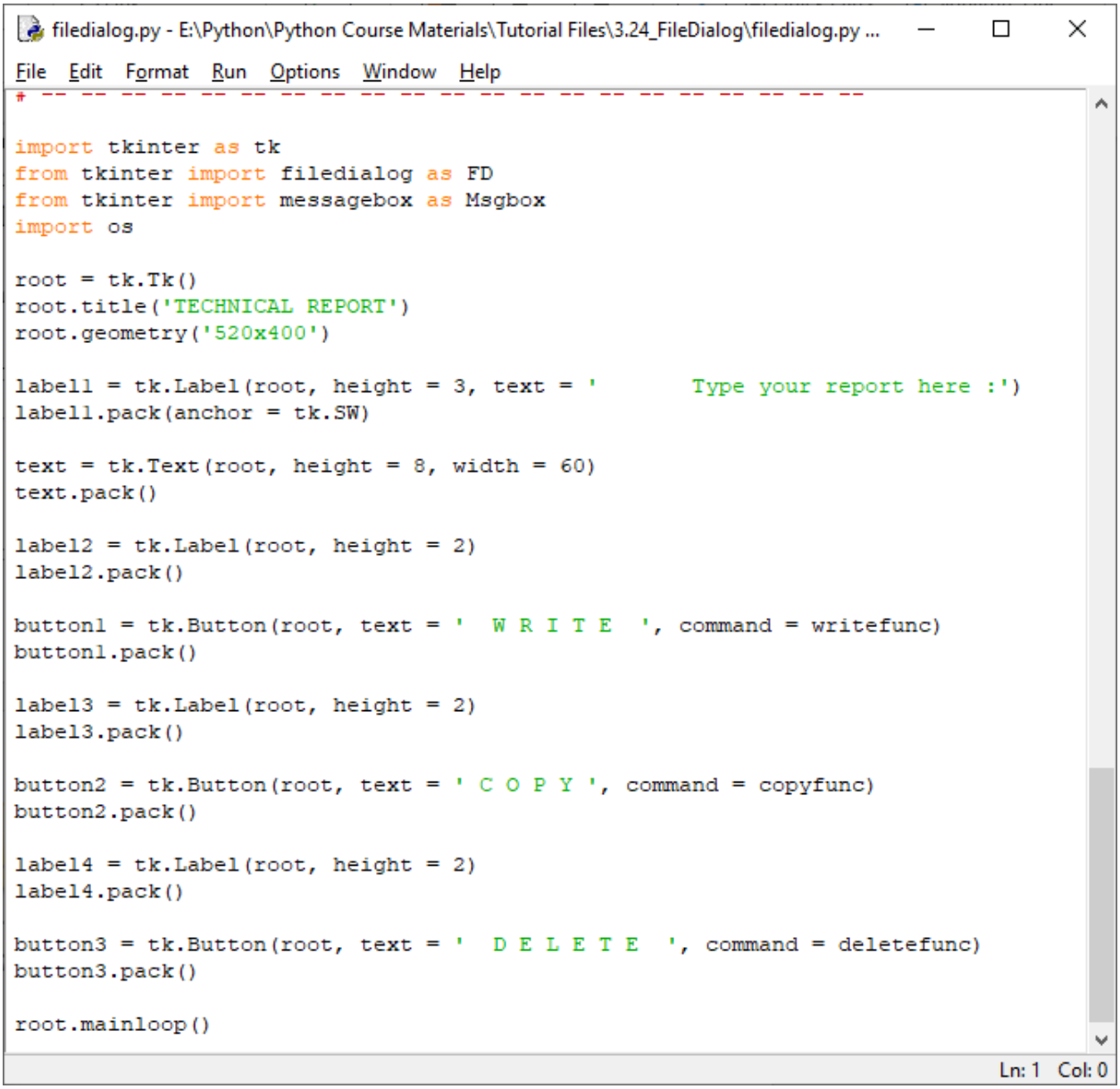
Option	Description	Values
defaultextension	an extension appended to the file name if the user does not provide one	the appropriate file extension – '.py', '.docx', '.xlsm', etc
filetypes		'*' used to show all files in the directory
initialdir	initial directory	selected by user
initialfile	initial file	selected by user
parent	controls which window the dialog box is placed on top of, the window (re)gains focus when the dialog box is closed	selected by user
title	specifies the title in the header bar of the dialog box	selected by user

10.2 tkFileDialog Example

In this exercise we shall create an app through which a user can enter a project report. The report can then be saved as a file of several filetypes to any folder the user chooses, using a tkFileDialog. The app will also have functionality to create copies of the report that can be edited and augmented while the original report remains intact. Finally, the app will enable the user to browse through folders and select a file and delete it.

In order to implement the functionality, we shall use a Text widget for typing in the report. Three (3) Button widgets shall be added. One Button will be used for creating the report file from the Text widget data. One Button will be used to make copies of the report file. The third Button will provide the functionality to browse folders and select a file for deletion.

Open a new session of IDLE (Python GUI).
Click on **File**.
Click on **New File**, to open the File Editor.



Replicate the following code to define the *writefunc( )* function for the ***WRITE*** Button.

filedialog.py - E:\Python\Python Course Materials\Tutorial Files\3.24_FileDialog\filedialog.py ...

File Edit Format Run Options Window Help

```
# -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --

def writefunc():

    # pull the text widget content
    report = text.get(1.0, tk.END) # ie fetch from start index
                                   # through END index

    # file types that can be used must be declared
    file_types = [('All files', '.*'), ('Text files', '.txt'),\
                  ('Word Document', '.docx*'),\
                  ('Word 97-2003 Document', '.doc')]
    # pick directory and give a file name and select file type
    file_name = FD.asksaveasfilename(parent = root,\
                        initialdir = os.getcwd(),\
                        filetypes = file_types,\
                        title = "Navigate and give a file name :")

    # create and open a new file or overwrite an existing one of that name
    report_file = open(file_name, "w")
                        # creates a file named as you entered and the
                        # type that you picked

    # write or overwrite the text widget content into the file
    report_file.write(report)

    # close the file
    report_file.close

    # a tkMessageBox to the user
    tkMsgbox1 = Msgbox.showinfo('Create Report',\
                        'The Tech Report has been created.\n\
Click on Copy to make an appended renamed copy of the report.')

# -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --

import tkinter as tk
from tkinter import filedialog as FD
```

Ln: 1 Col: 0

Replicate the following code to define the *copyfunc( )* function for the ***COPY*** Button.

*filedialog.py - E:\Python\Python Course Materials\Tutorial Files\3.24_FileDialog\filedialog.py...

File Edit Format Run Options Window Help

```
# -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
def copyfunc():

    # makes a copy of the file created to hold the Text report

    # pull the text widget content
    report = text.get(1.0, tk.END) # ie fetch from start index
                                   # through END index

    # file types that can be used must be declared
    file_types = [('All files', '.*'), ('Text files', '.txt'),\
                  ('Word Document', '.docx*'),\
                  ('Word 97-2003 Document', '.doc')]
    # pick directory and give a file name and select file type
    file_name = FD.asksaveasfilename(parent = root,\
                        initialdir = os.getcwd(),\
                        filetypes = file_types,\
                        title = "Navigate and give a new file name :")

    # create and open a new file or overwrite an existing one of that name
    report_file = open(file_name, "w")
                        # creates a file named as you entered and the
                        # type that you picked

    # write or overwrite the text widget content into the file
    report_file.write(report)

    # close the file
    report_file.close

    # reopen the file to have content appended
    report_file = open(file_name, "a")
    report_file.write('\n- - - - - APPENDED - - - - -')
    report_file.close

    # a tkMessageBox to the user
    tkMsgbox2 = Msgbox.showinfo('Copy of Report',\
                        'The appended copy of the Tech Report file has been\n\
saved under the new name.')
# -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
def writefunc():
```

Ln: 69 Col: 64

Replicate the following code to define the *deletefunc( )* function for the ***DELETE*** Button.

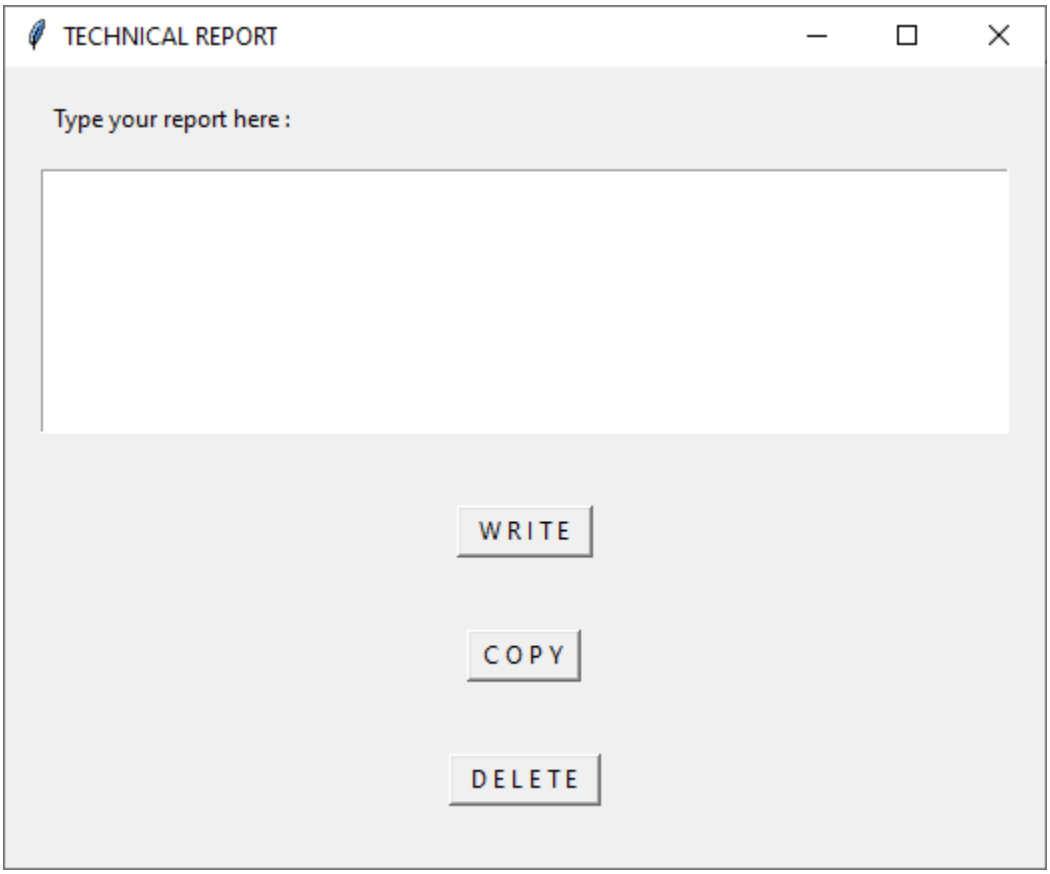
*filedialog.py - E:\Python\Python Course Materials\Tutorial Files\3.24_FileDialog\filedialog.py...

File Edit Format Run Options Window Help

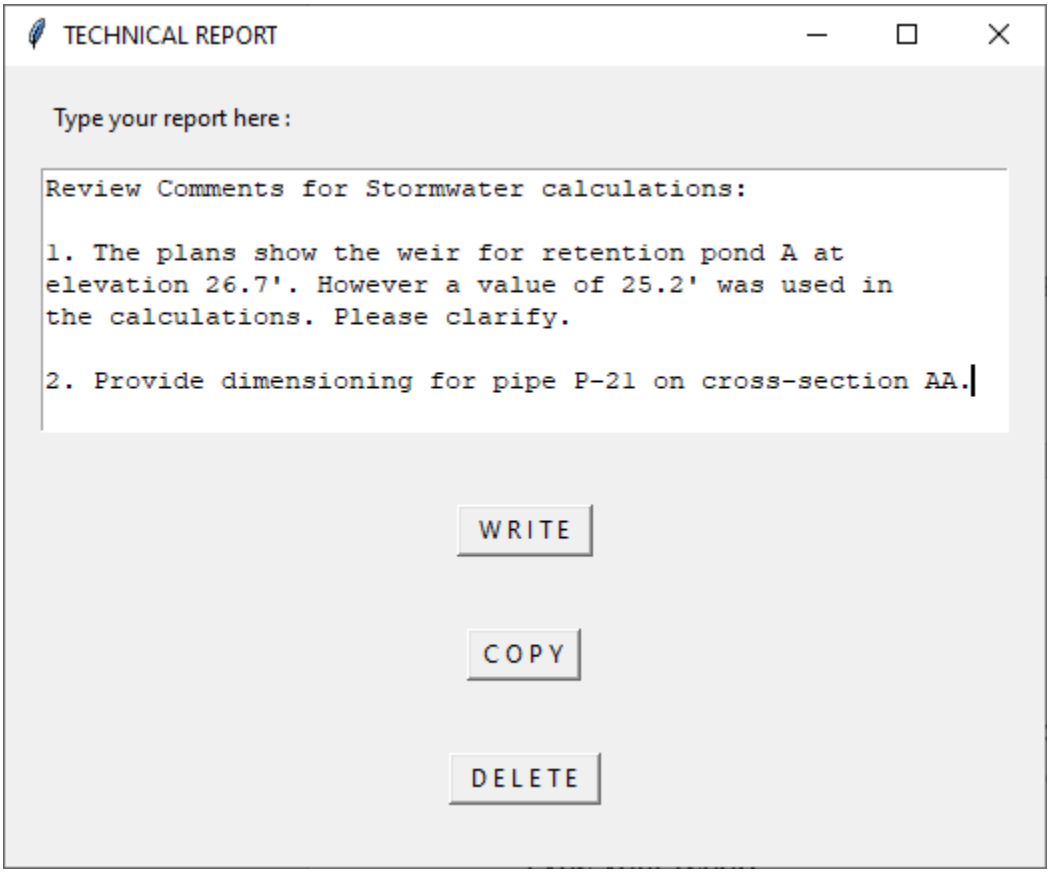
```
def deletefunc():

    tkMsgbox3 = Msgbox.askyesno('File Deletion Pending',\
                        'Are you sure you want to delete a file?')

    if tkMsgbox3 == tk.TRUE:

        # file types that can be selected
        file_types = [('All files', '.*'), ('Text files', '.txt'),\
                      ('Word Document', '.docx*'),\
                      ('Word 97-2003 Document', '.doc')]
        # this code will open Explorer Window for you to navigate to a
        # the file to be deleted and select it
        file_to_delete = FD.askopenfilename(parent = root,\
                                initialdir = os.getcwd(),\
                                filetypes = file_types,\
                                title = "Navigate and select file to delete:")

        # delete the file using the operating system's remove function
        os.remove(file_to_delete)

    else:
        pass



# -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --

def copyfunc():

    # makes a copy of the file created to hold the Text report

    # pull the text widget content
    report = text.get(1.0, tk.END) # ie fetch from start index
                                   # through END index

    # file types that can be used must be declared
    file_types = [('All files', '.*'), ('Text files', '.txt'),\
                  ('Word Document', '.docx*'),\
                  ('Word 97-2003 Document', '.doc')]
```

Ln: 27 Col: 0

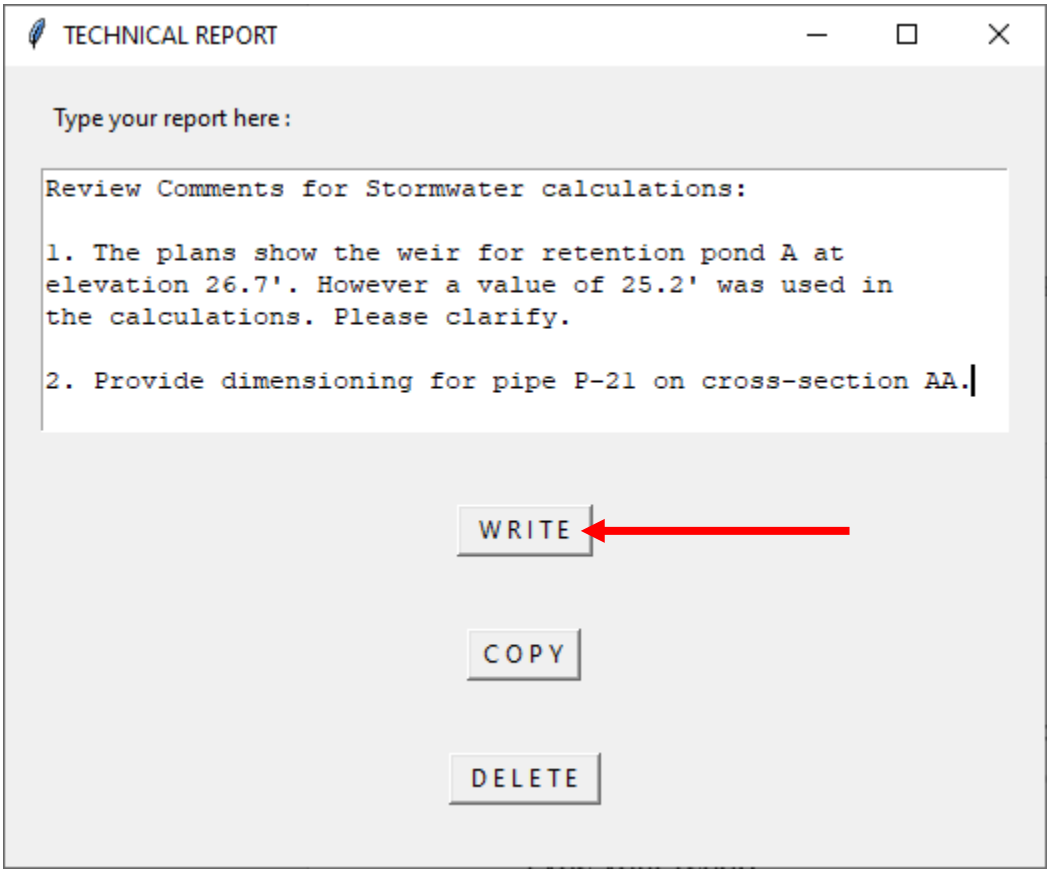
Save the file.
Run the file.



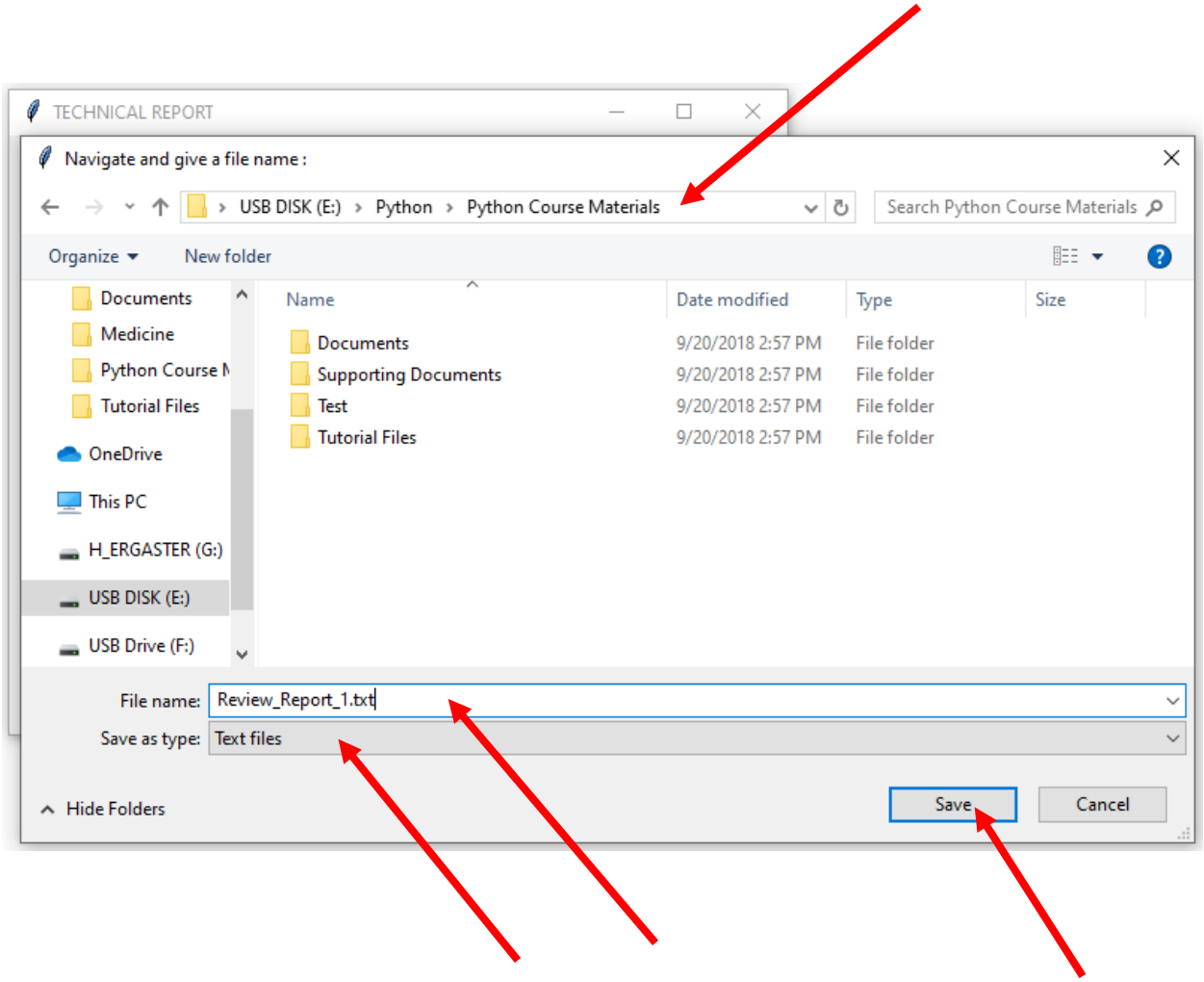
Type your report.



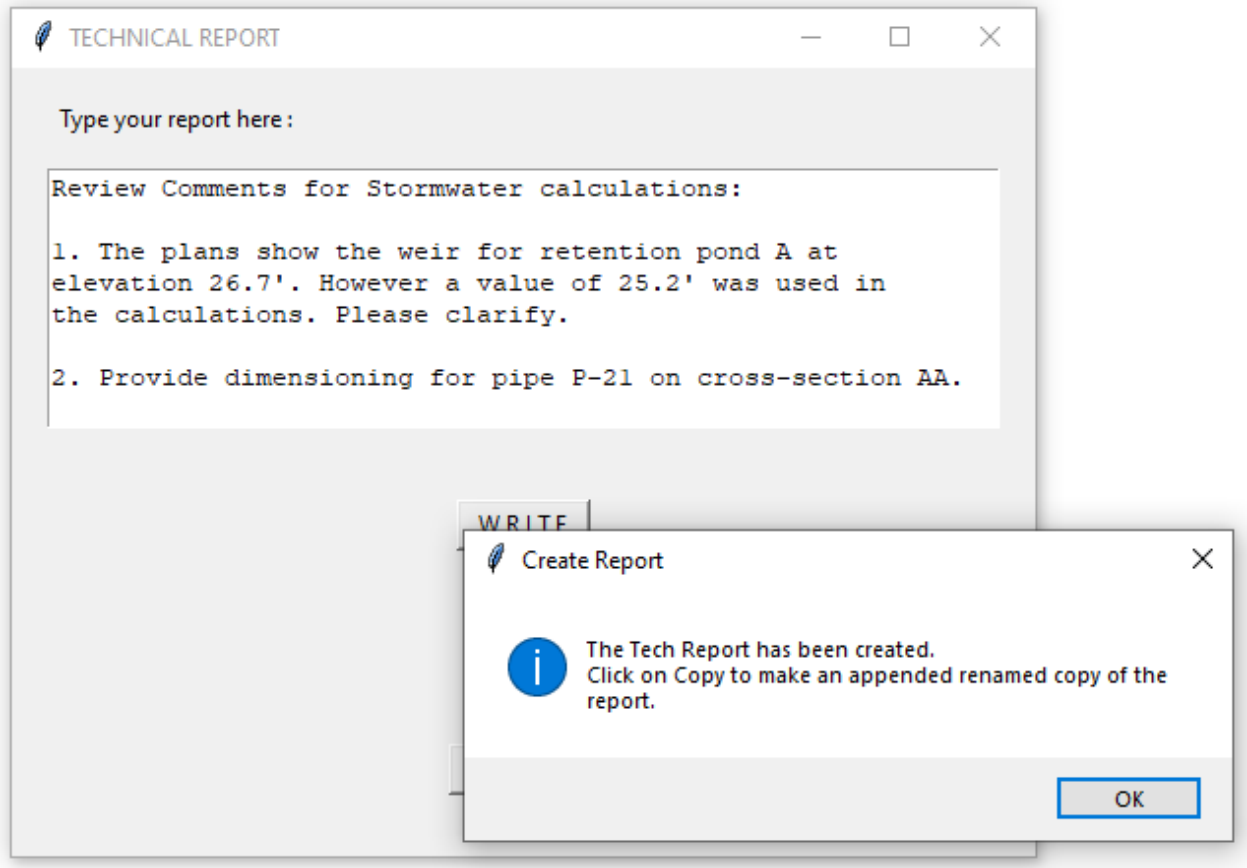
To create your report file, click on ***WRITE***.



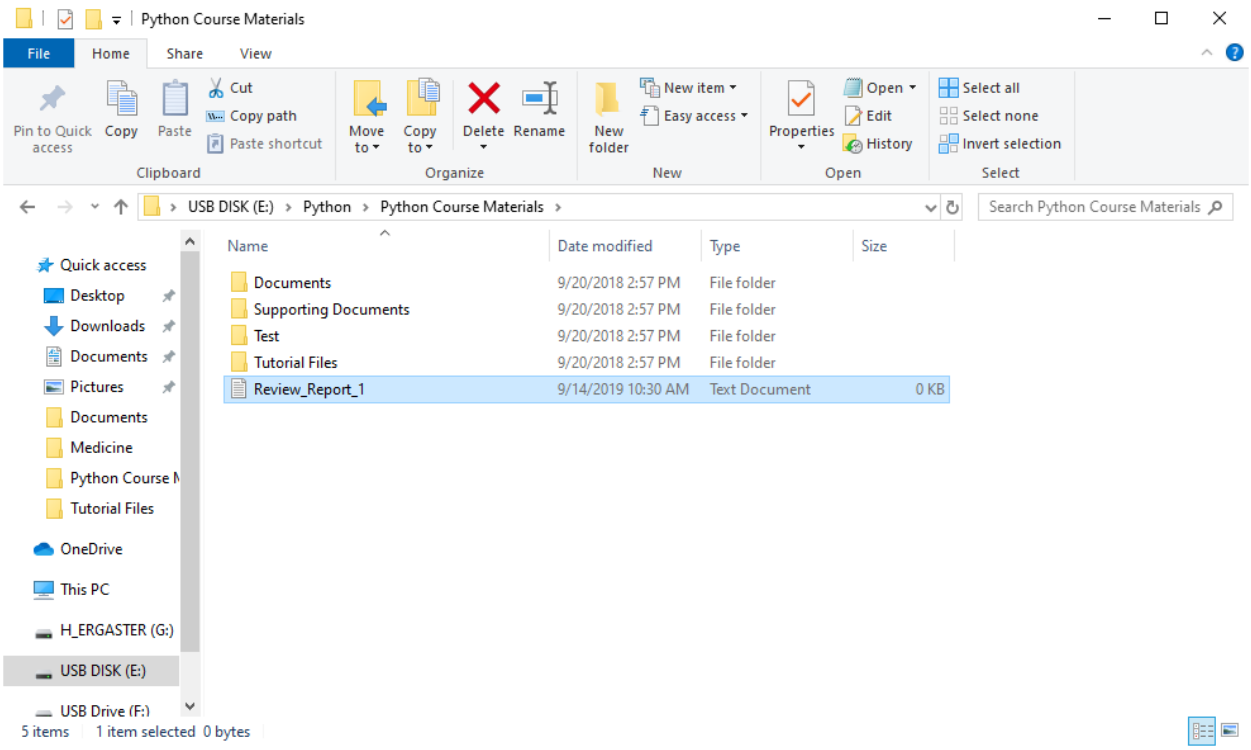
Navigate to a folder of your choice.
Name your file with the relevant suffix. In this exercise we shall save as a text file.
Click ***Save***.



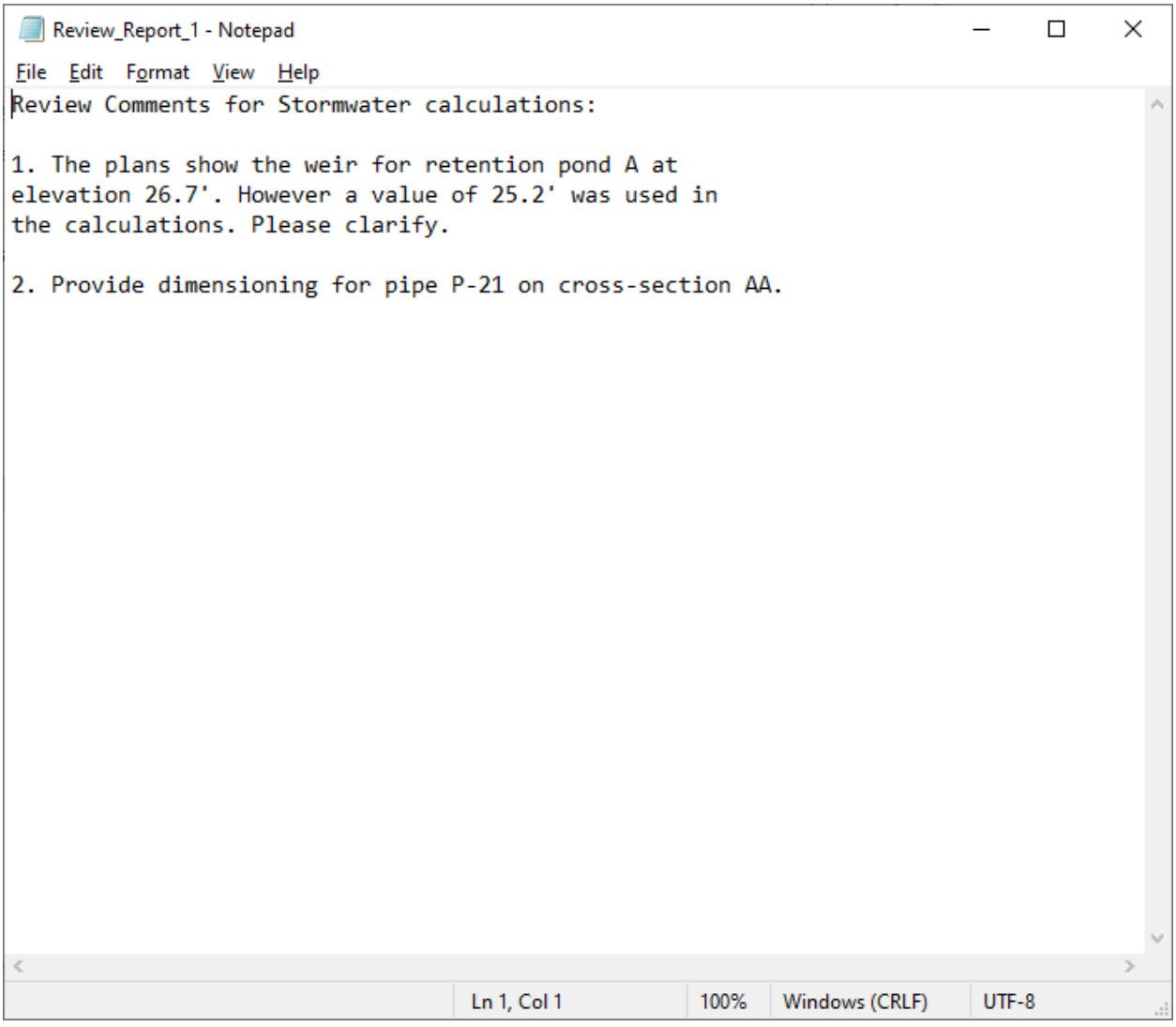
The tkMessageBox appears.
Click ***OK*** to dismiss the pop up.



Use your computer's file explorer to navigate to the folder you saved your report to and confirm the file has been created.

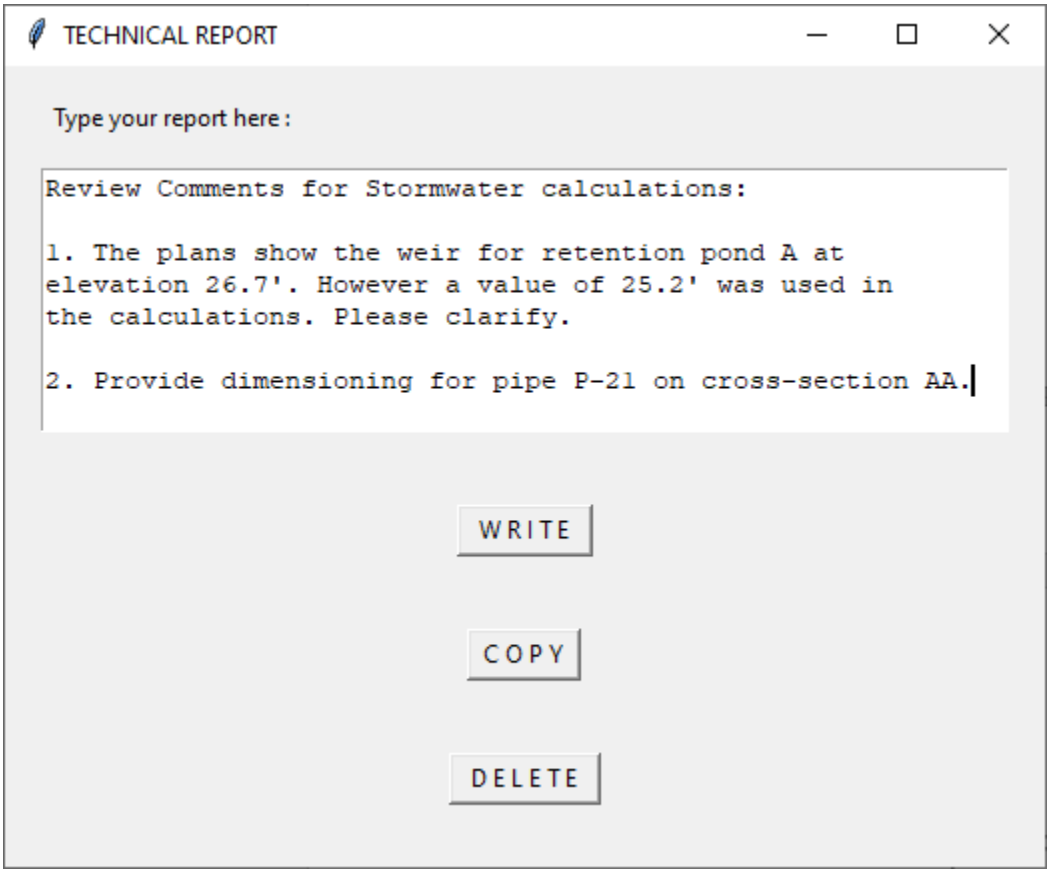


Open the file to confirm the contents.

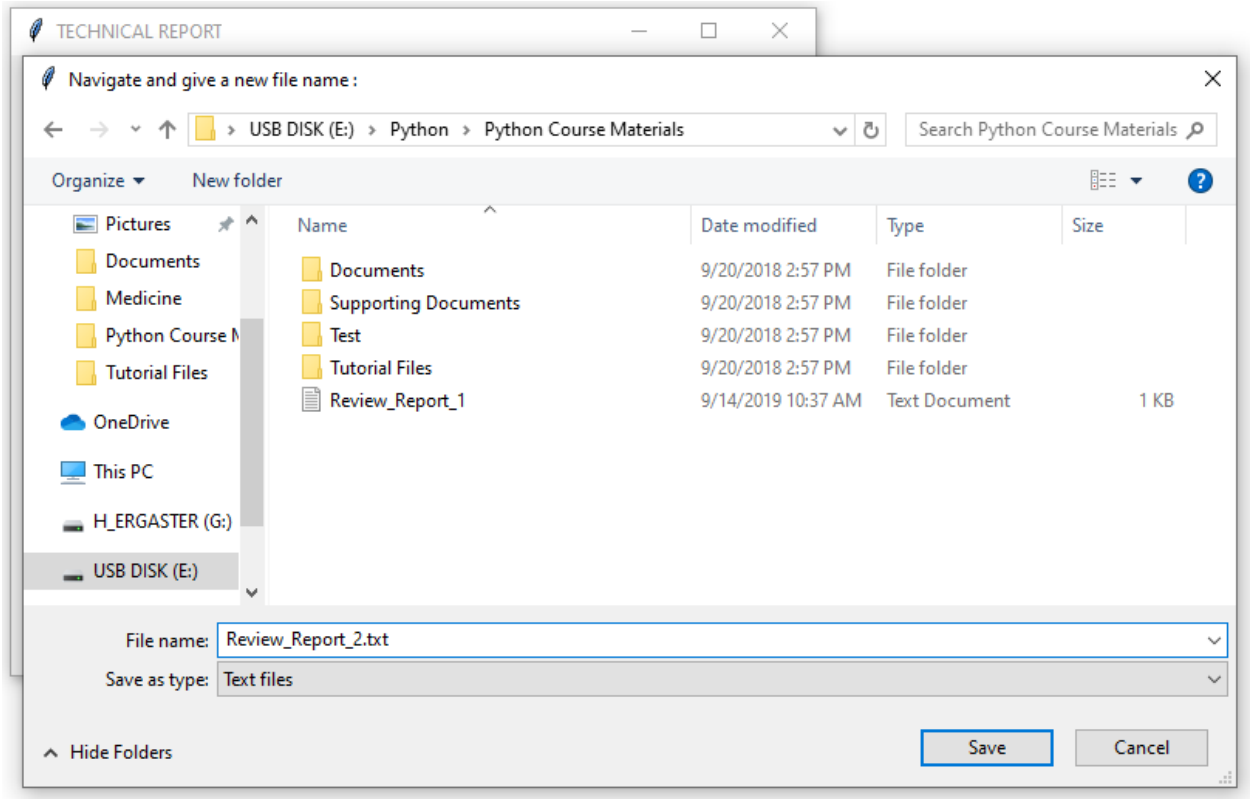


Close the file.
Let's go back to the report app.

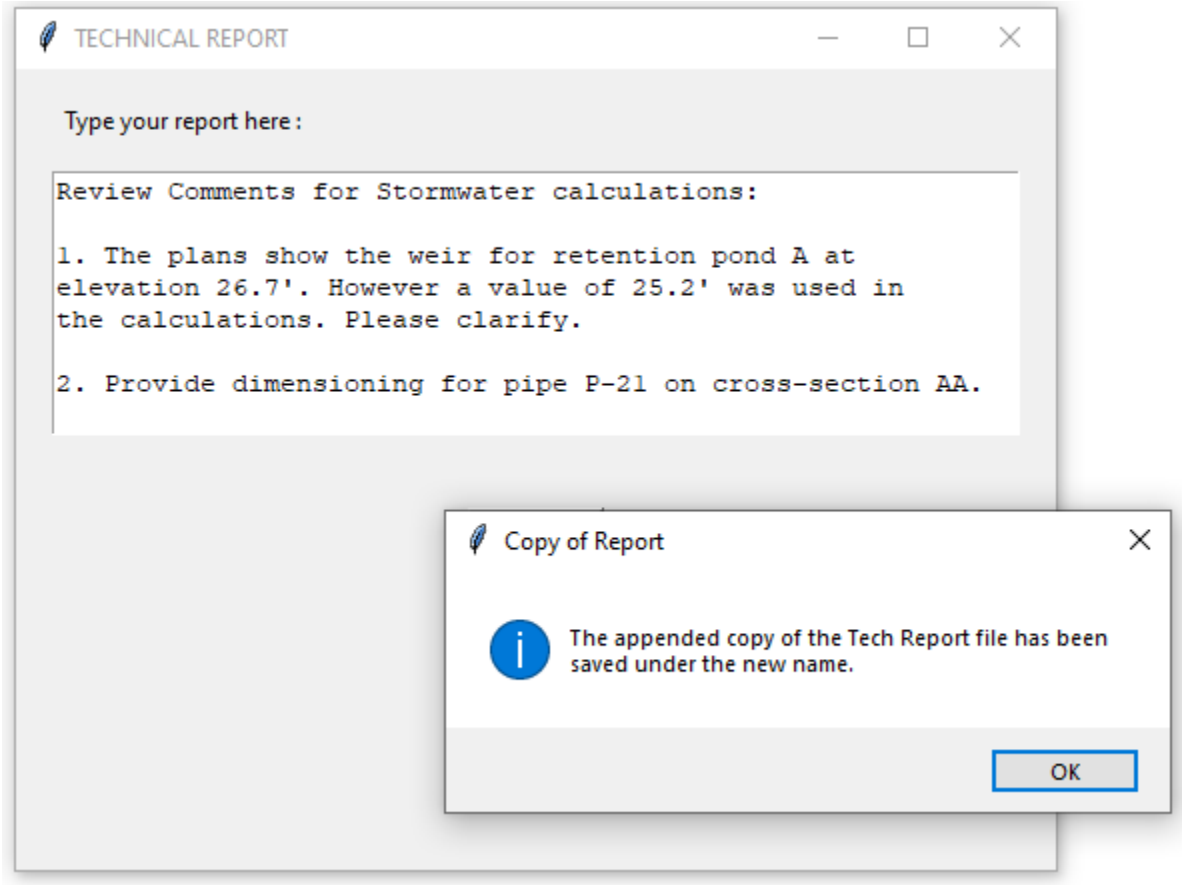
Click on ***COPY***.



Navigate to your folder.
Name your file copy.

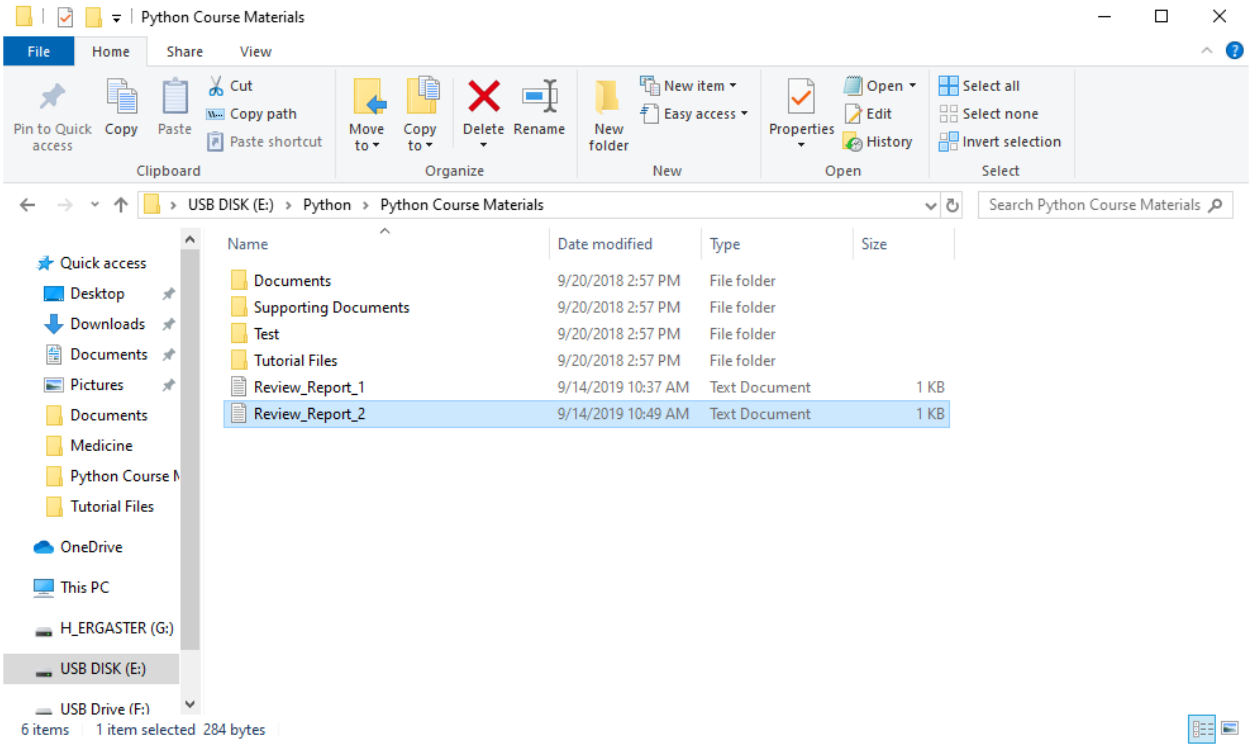


Click on ***Save***.

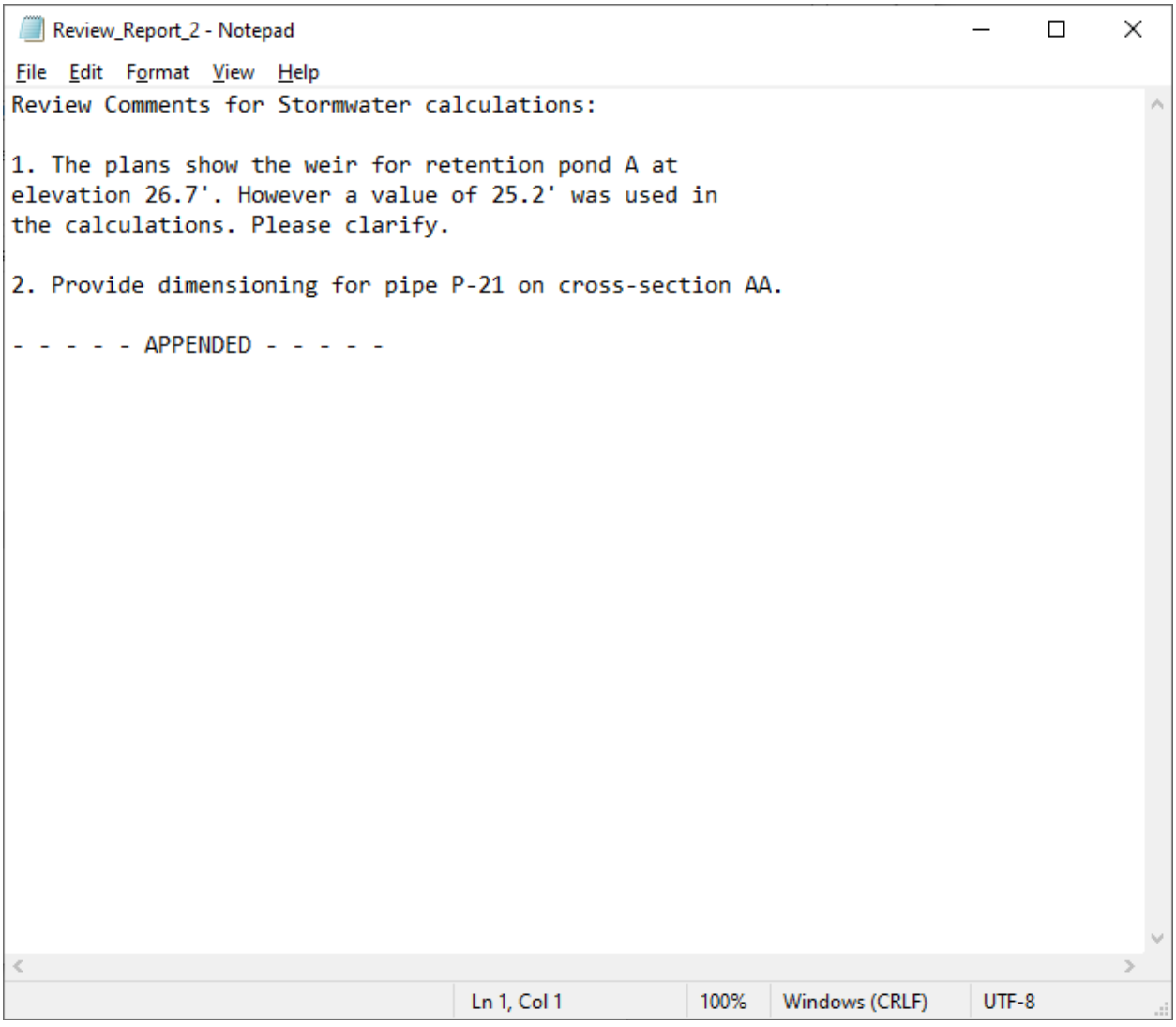


Review the pop up.
Dismiss the pop up.

Review your folder.

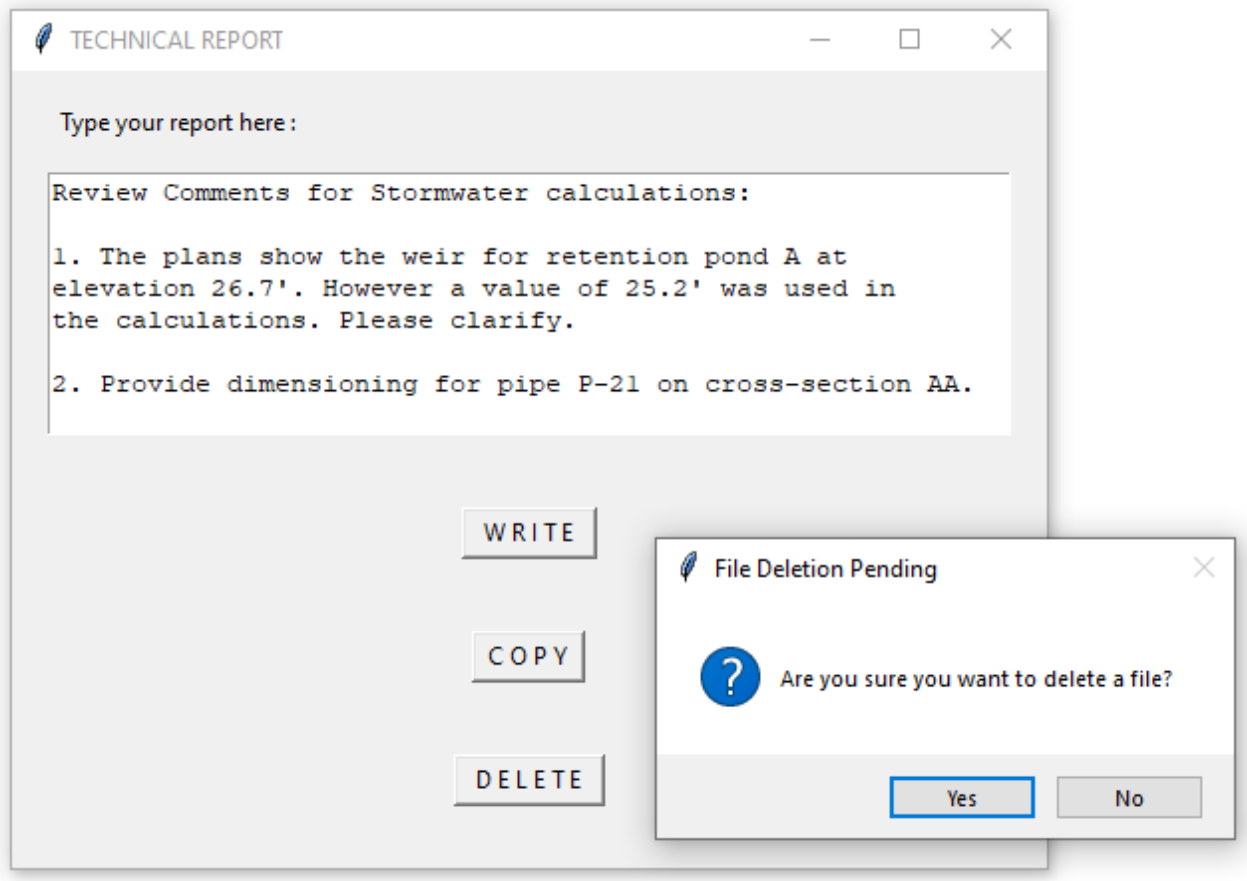


Open the file and review the contents.



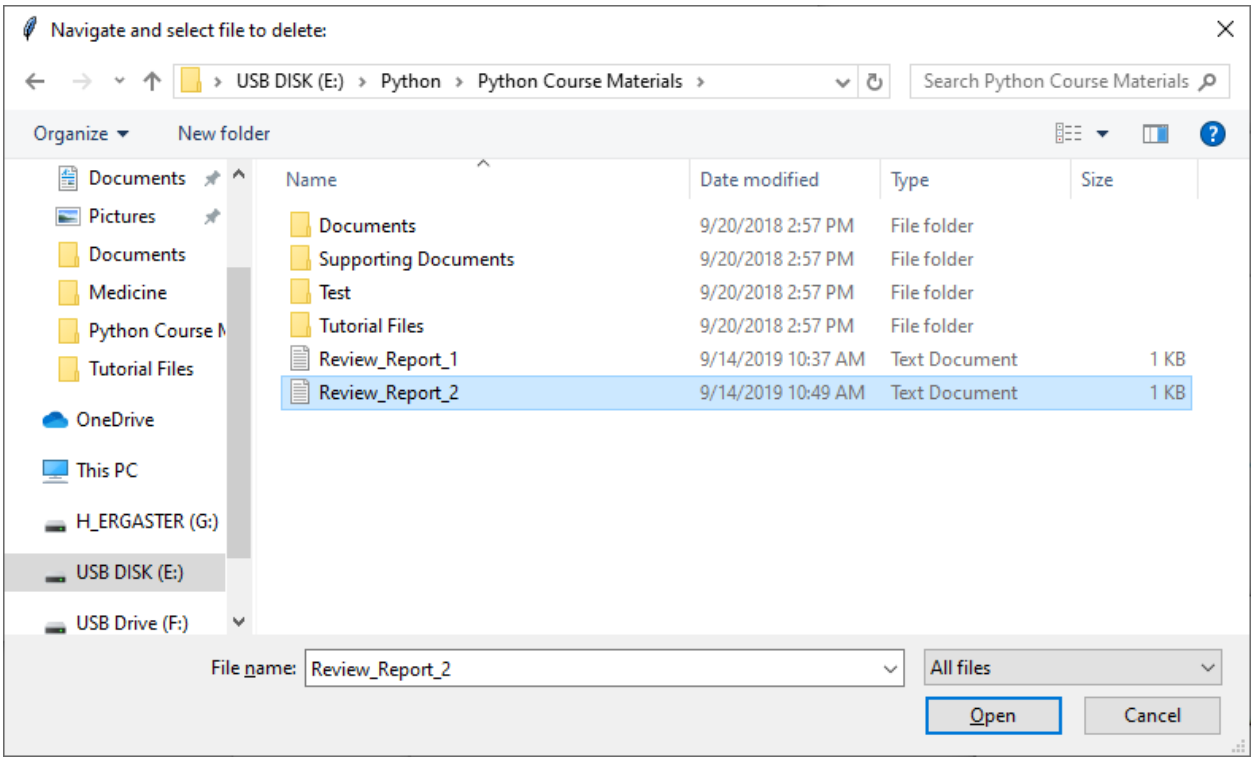
Close the file.
Let's go back to the report app.

Click on ***DELETE***.



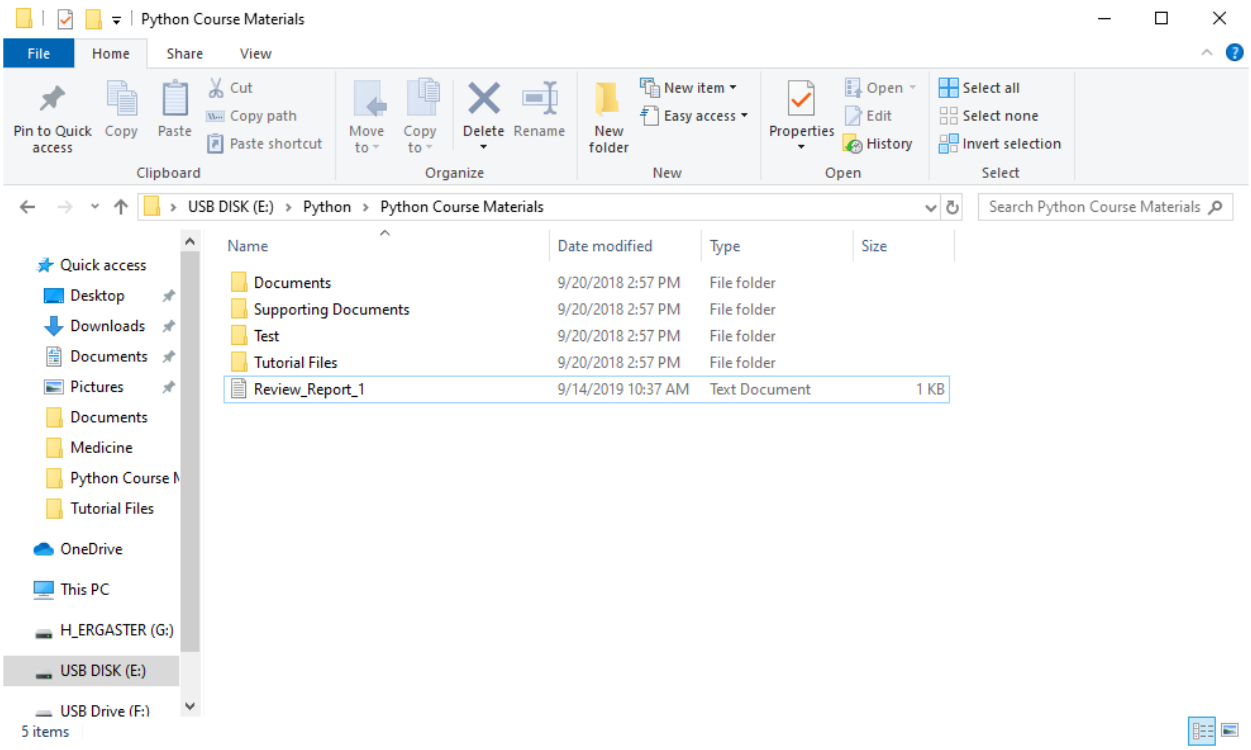
Click ***Yes*** to continue the file deletion exercise.

Navigate to your folder.
Select a file to delete.

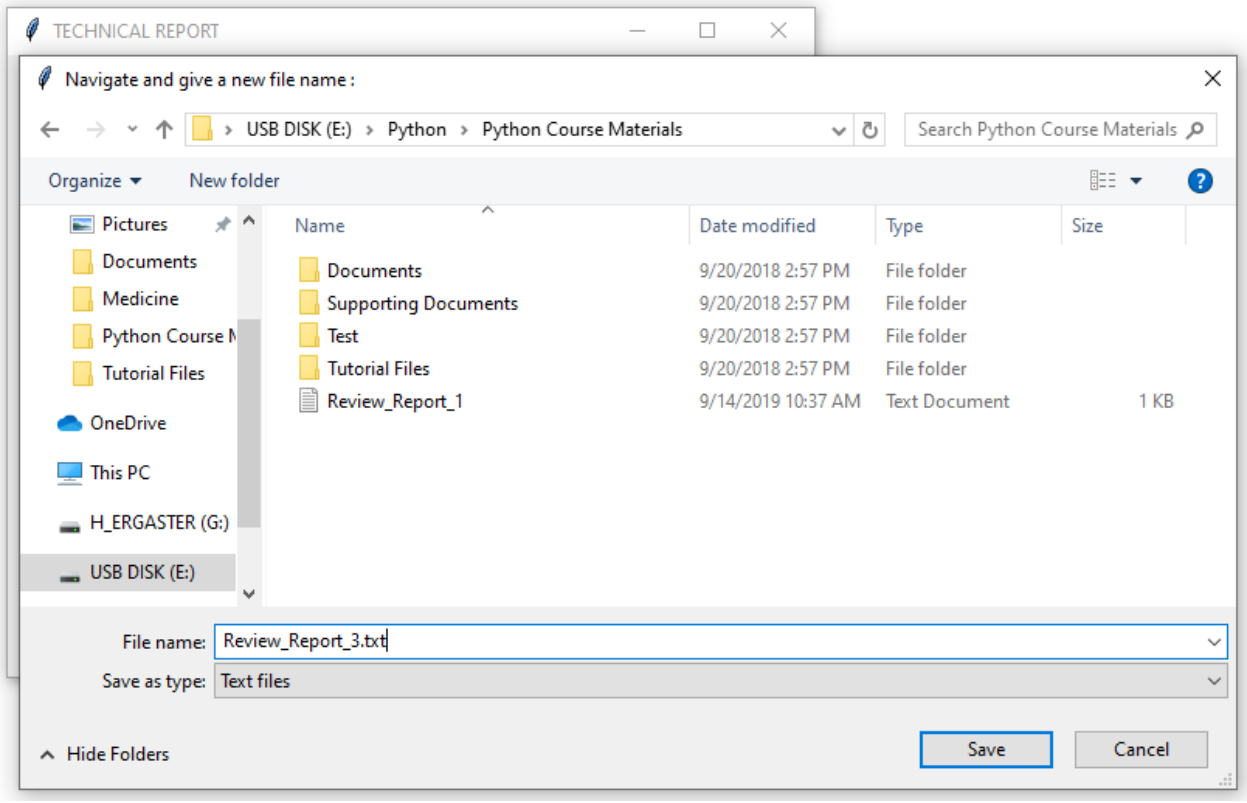


Click on ***Open*** to delete the file.

Review your folder.
Confirm the file has been deleted.

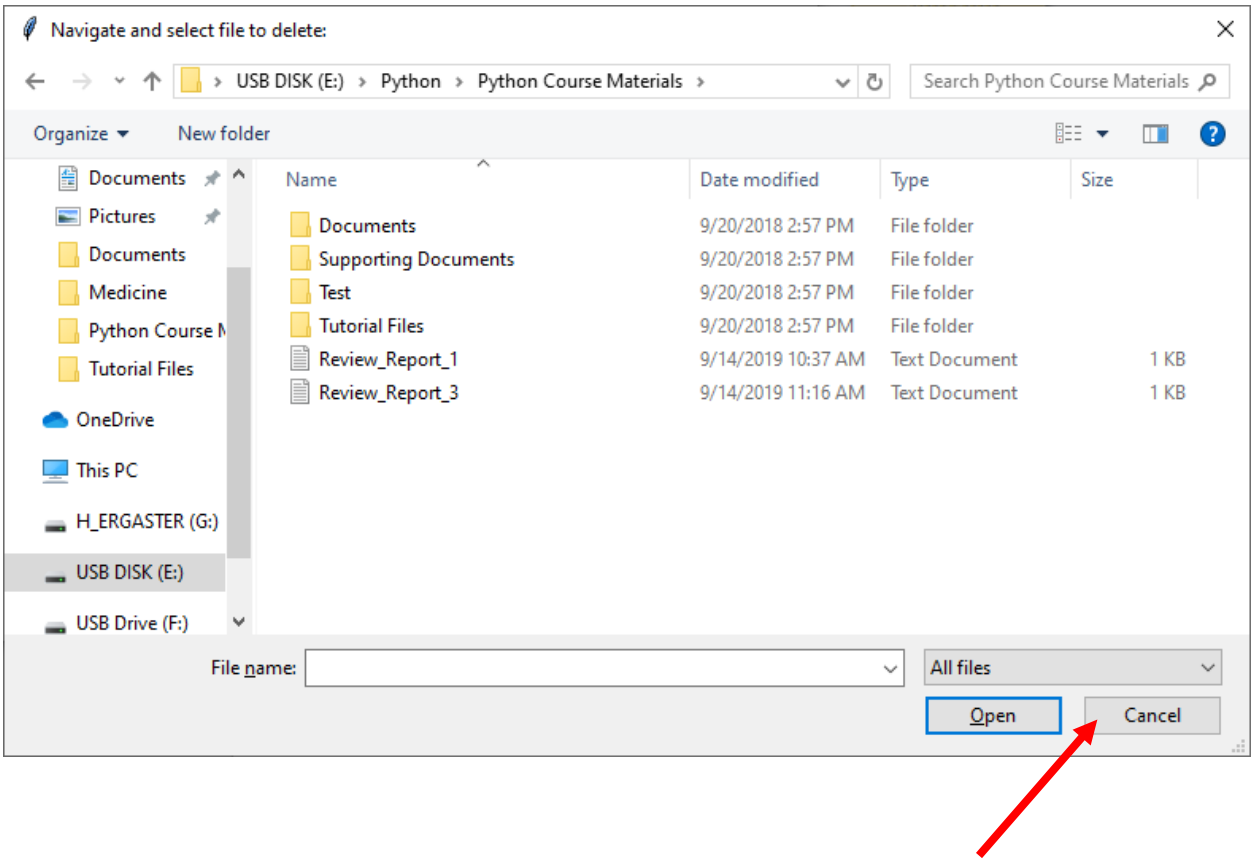


Make a new copy of the report file.

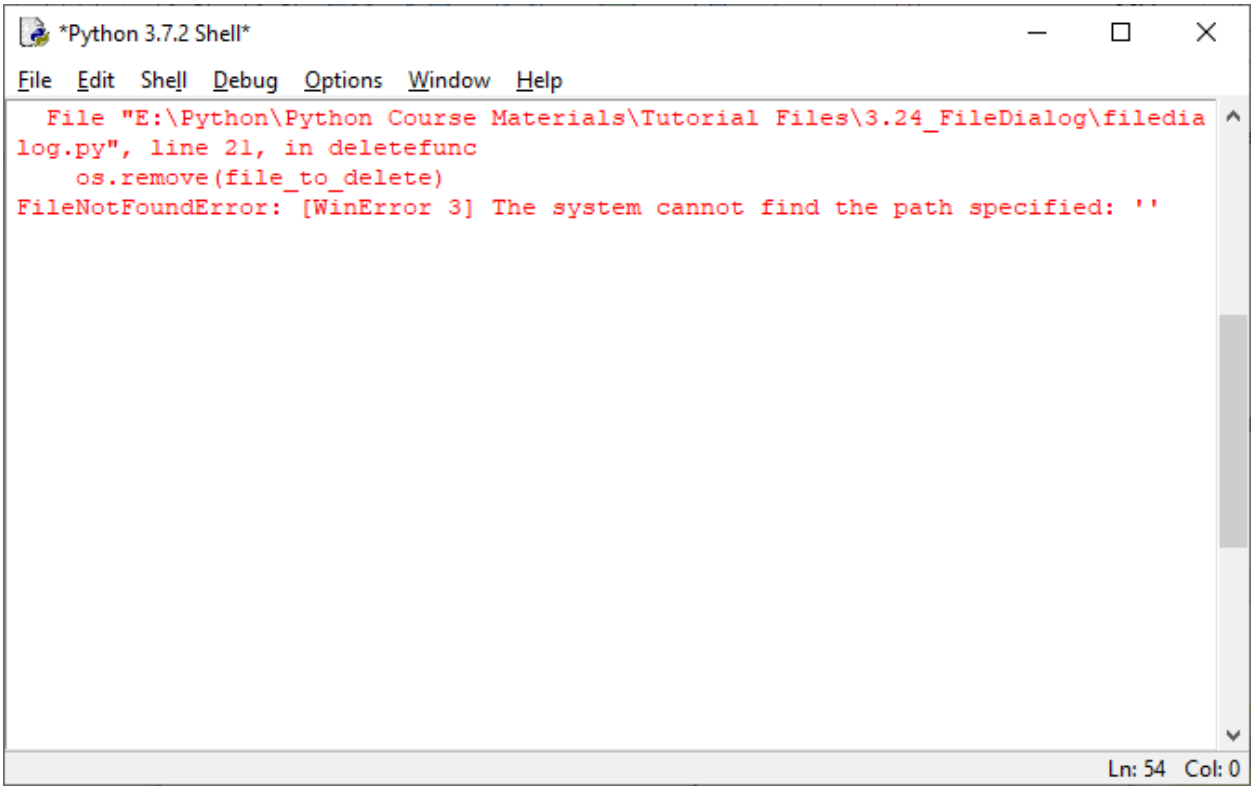


Complete the process.

Proceed with the process to delete the copy file.
But this time, in the tkFileDialog, click on ***Cancel***.



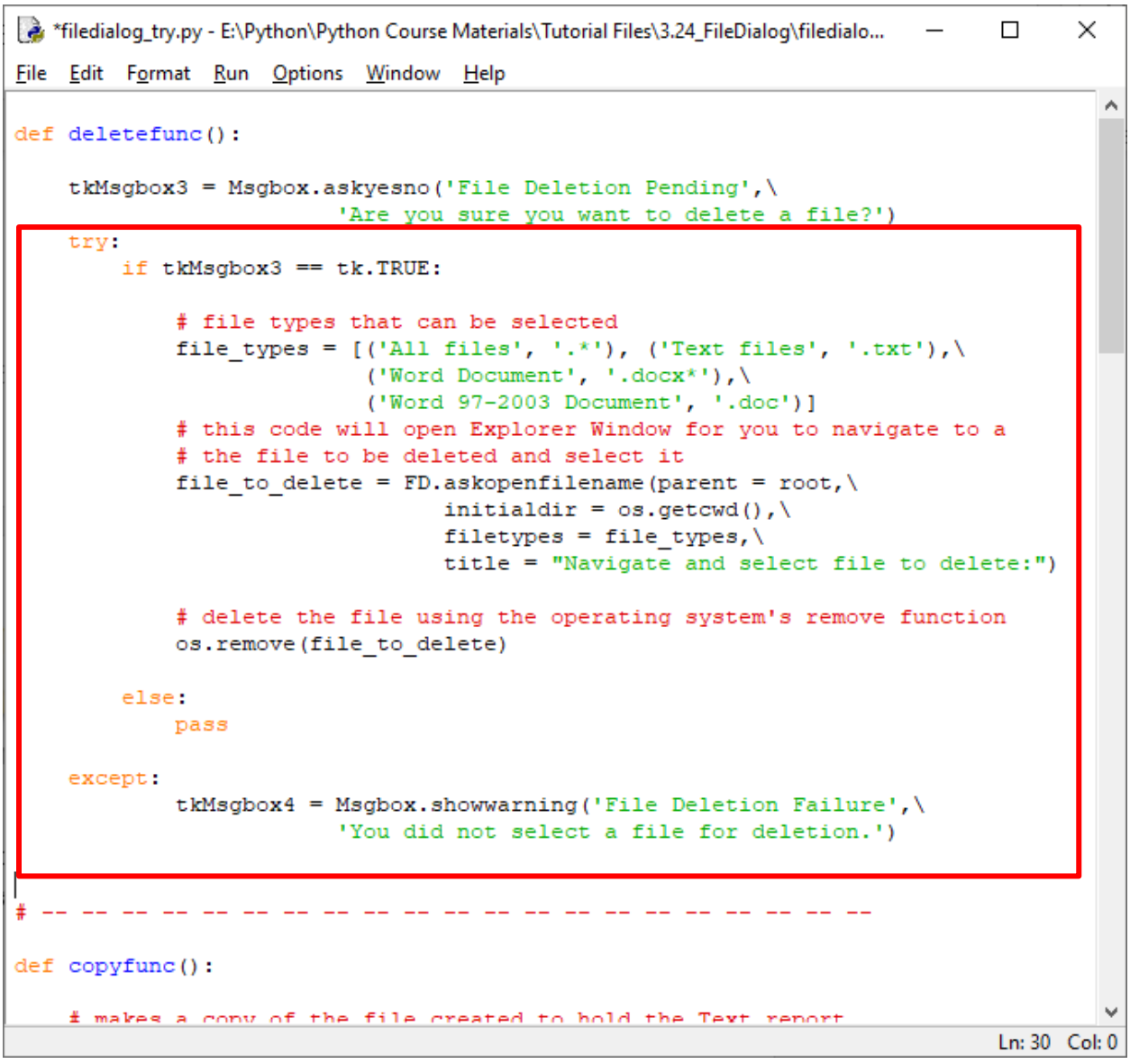
Nothing appears to happen.
Review your IDLE (Python GUI) window.
Review the error message generated by not selecting a file to delete.
The tkFileDialog, once called, requires that a valid file (or path to a file) is selected.
Thus, the program raises an exception.



A non-expert user may be unable to figure out what needs to be done.
Under Exception Handling we learned that it is preferred to incorporate Exception Handlers in the code such that the user does not see the error messages generated by the program. A custom message or some other direction or action can then be taken to assist the user or address the problem "behind the scenes".

Close the report app.

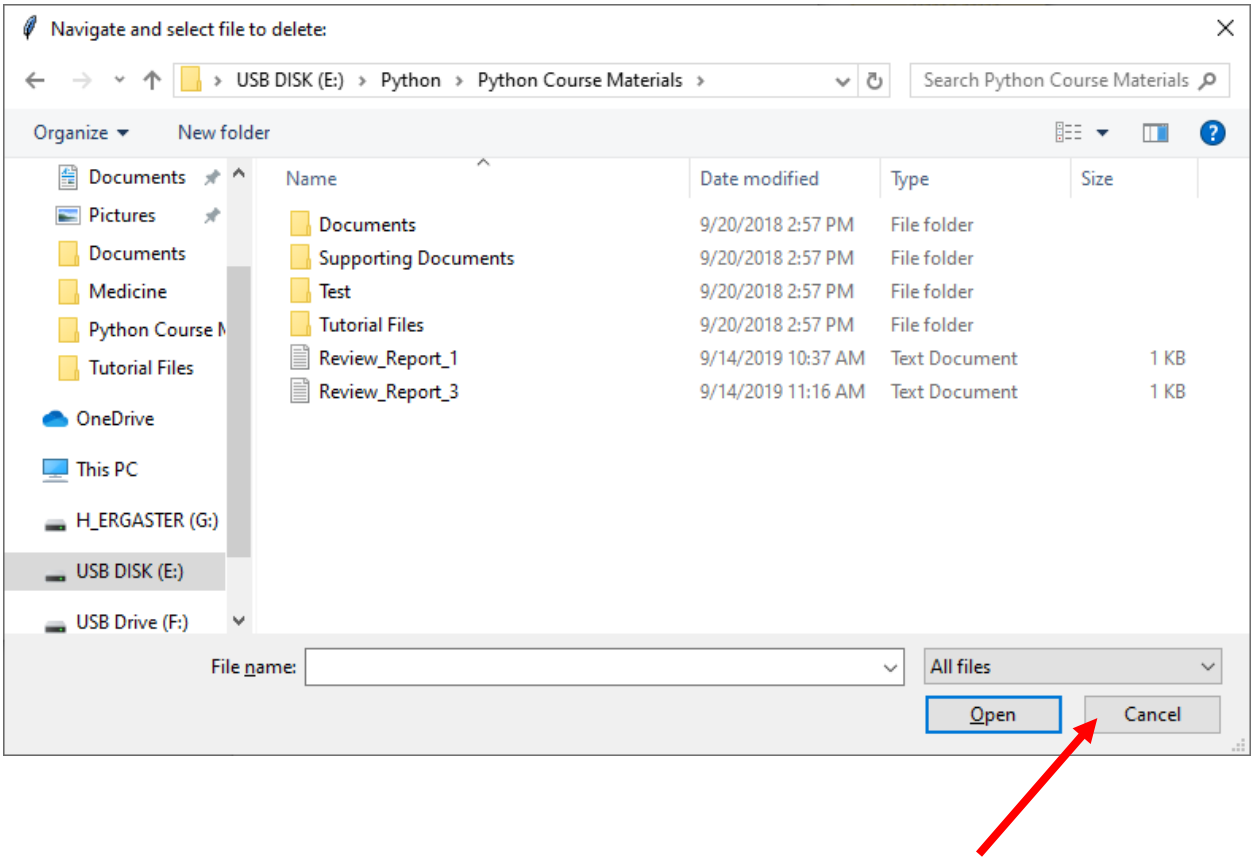
Update your *deletefunc( )* function code as shown below to implement an error handler for the scenario that the user abruptly cancels out of the file deletion process.



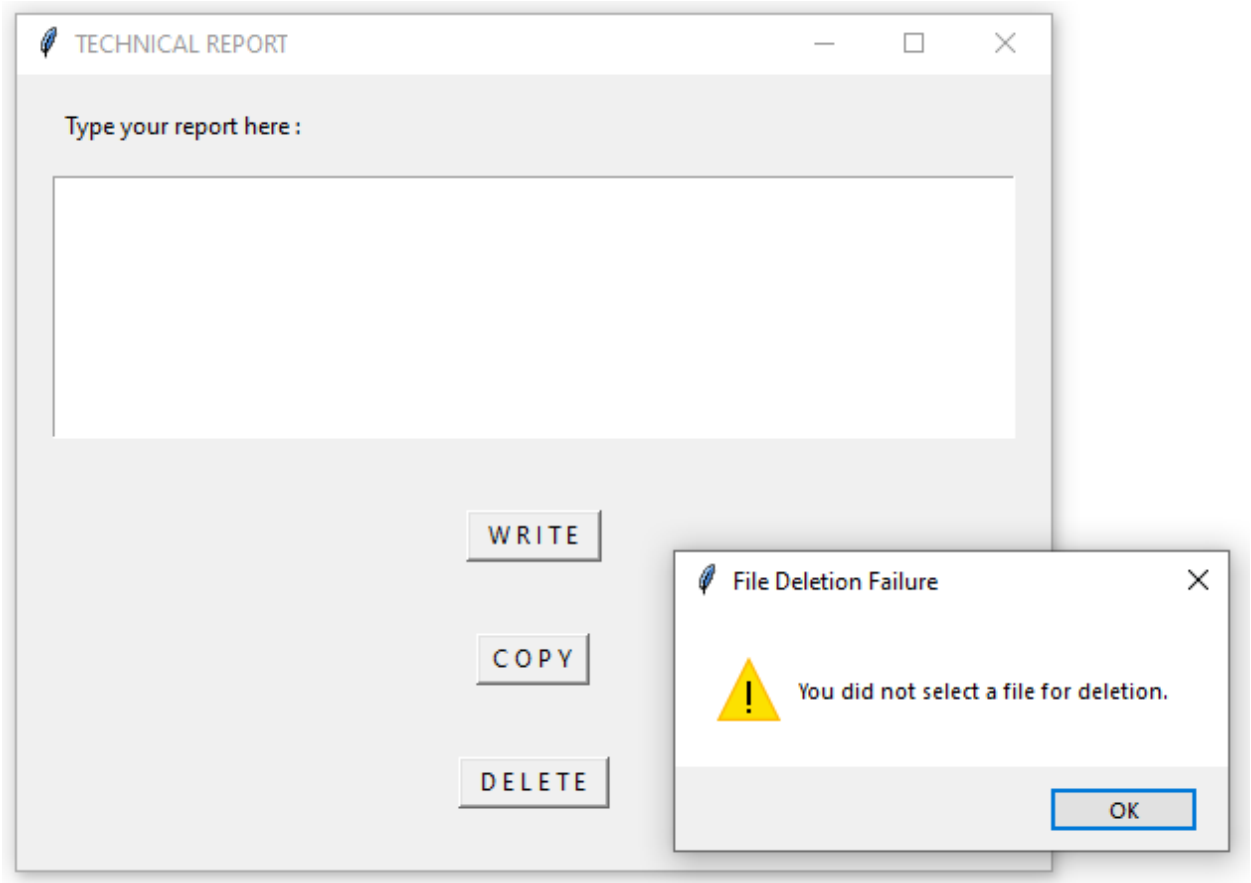
Note the use of the indenting to implement the error hander. Essentially, the existing code is "wrapped" in a *try … except* structure.

The tkMessageBox added under the *except* clause shall appear to the user and the built-in exception raised by the *Python* interpreter will be suppressed from the user's view.

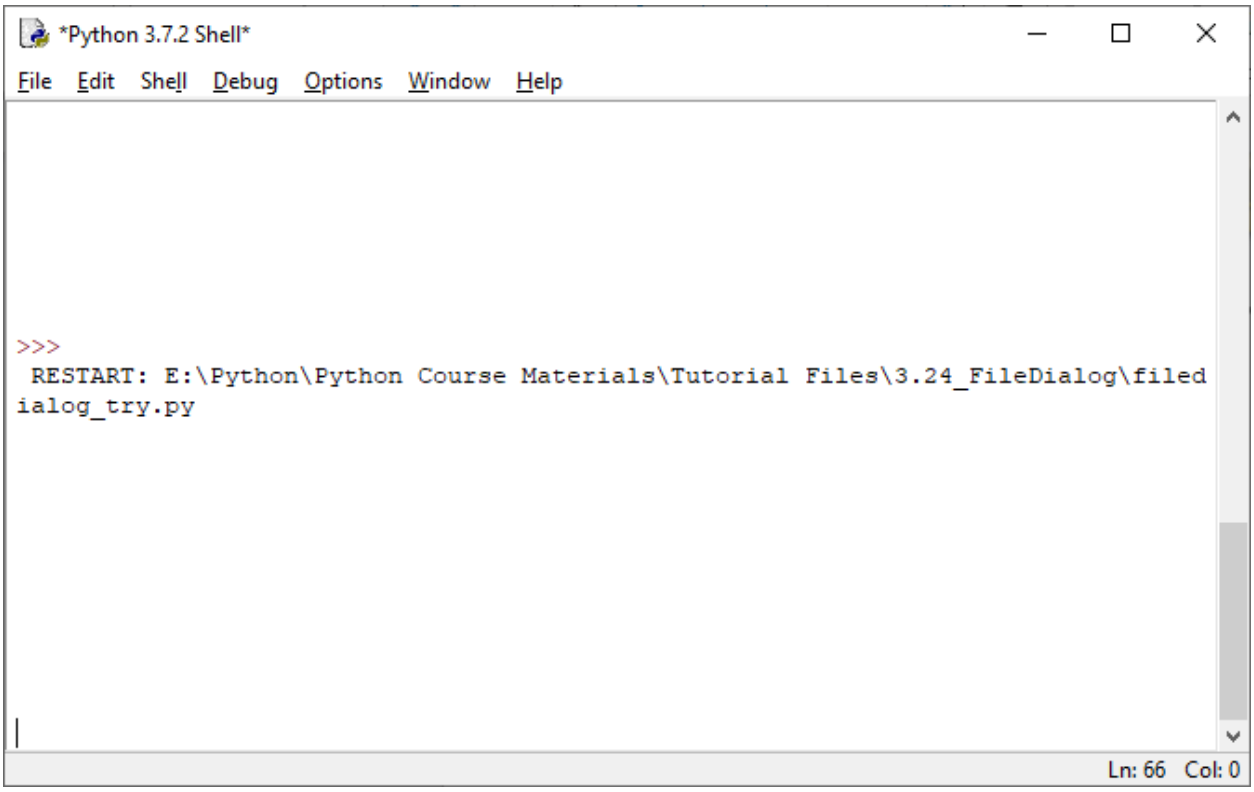
Save the file.
Run the file.
Proceed with the process to delete a file.
In the tkFileDialog, click on ***Cancel***.



The tkMessageBox we added in the *try* … *except* structure opens.



Dismiss the pop up.
Review your IDLE (Python GUI) window.



There are no error messages (or exceptions).
Successful completion!

11. DEBUGGING AND GETTING HELP

11.1 Exceptions

Domain knowledge and experience must be used to design and implement appropriate error handlers. The programmer should test multiple scenarios that a non-expert end-user may encounter and design the error handlers to guide and assist the end-user accordingly while precluding to the maximum extent possible, built-in exceptions being raised to the end-user.

11.2 Testing and Debugging

It cannot be overemphasized that all *Python* scripts should be meticulously reviewed, thoroughly scrutinized, and frequently tested as they are being developed. It is the programmer's responsibility to frequently test the code and address problems as they arise, and to verify or otherwise that the scripts execute as intended (validation). Test your codes and scripts frequently, block by block, line by line, using the IDLE (Python GUI) or the File Editor or any other preferred Python tool. A piecemeal approach to writing and testing code is strongly preferred rather than writing the entire script before testing it. In the latter scenario, it will be significantly more difficult to identify and isolate the relevant problems.

11.3 Getting Help

There is currently an abundance of help information on *Python* and *tkinter* programming on the World Wide Web. These include official (peer-reviewed) and unofficial sources, websites, academic reports, professional presentations, tutorial videos (YouTube, etc.), user groups, online forums, downloadable code snippets, etc., etc. Typing any *Python* or *tkinter* topic in a search engine will typically yield tens if not hundreds of results. It is still strongly recommended, regardless of the source of any contributory or relevant help information, that all codes being developed be tested and validated thoroughly before deployment.

12. CONCLUSION

Python is an interpreted, high-level, general purpose programming language. *Python* is a free and open source and can be used to build a wide range of desktop and web-based applications. This course has presented an overview of the *Python tkinter* libraries for developing graphical user interfaces (GUI). This course presented fundamental concepts, principles, and programming structures of the *Python tkinter* GUI programming.

In this course the following *Python tkinter* widgets were presented in detail: tkMessageBox, Scale, Scrollbar, Toplevel, and PanedWindow. Other widgets from other *Python* and *tkinter* modules and function libraries that were discussed in this course are: Combobox, simpledialog and tkFileDialog. Practical examples from situations encountered by a practicing engineer or scientist were used to illustrate and demonstrate the concepts and methods learned in this class.

This course has prepared participants to now develop their own applications driven by *Python*. This course has enabled participants to identify situations where computer programming is relevant and will be of advantage to the practicing professional competing in the global marketplace.

Practitioners are strongly encouraged to look for situations in their domains of expertise where computer programming solutions are applicable and will be of benefit to their work and their organizations.

All programming requires a careful and meticulous approach and can only be mastered and retained by practice and repetition.

Good Luck and Happy Programming.

REFERENCES

effbot.org. (2019). *An Introduction to Tkinter*. Retrieved August 2019, from effbot.org: http://effbot.org/tkinterbook/tkinter-index.htm

Python Software Foundation. (2019). *Python Software Foundation*. Retrieved July 2019, from Python Software Foundation: http://www.python.org/psf/

Python Tutorial: A Tutorial. (2019). Retrieved September 2019, from Tutorials, Python Courses: Online and On Site: https://www.python-course.eu/python_tkinter.php

tutorialspoint. (2019). *Python - GUI Programming (Tkinter)*. Retrieved August 2019, from tutorialspoint.com: https://www.tutorialspoint.com/python/python_gui_programming

Images were all drawn/prepared by Kwabena. Ofosu

