A SunCam online continuing education course

Python Programming for Engineers - Part 3: Graphical User Interfaces I

by

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

Computer Programming in *Python* – Part 3
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 third 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 labels, text and entry widgets, click buttons, check buttons, radio buttons, listboxes, spinboxes, menus and frames, as well as message and canvas 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.

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

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*.

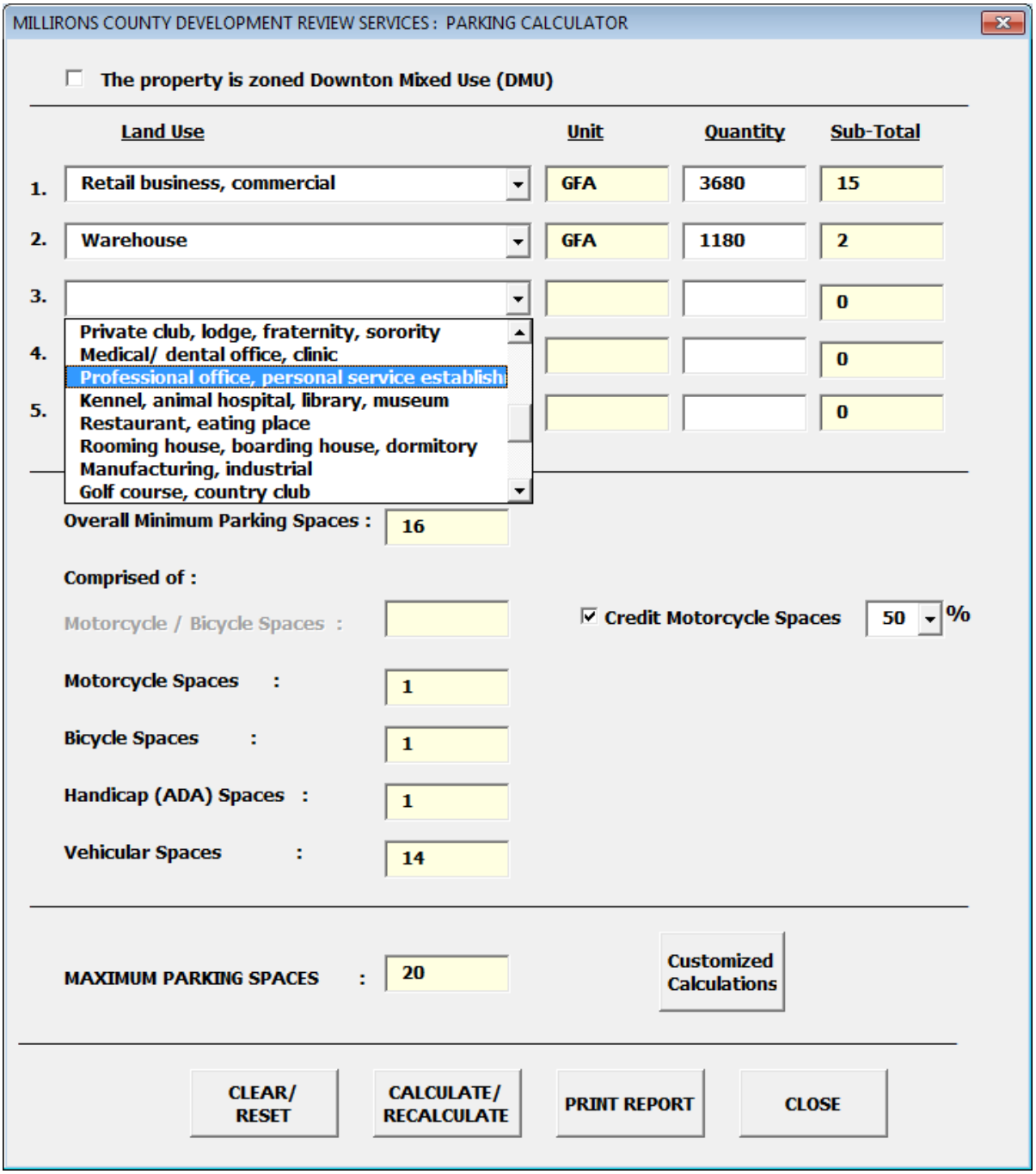


Figure 1. 1: A graphical user interface (GUI)

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.

GUI_1.py - C:\Users\Kwabena\Downloads\GUI_1.py (3.7.2)

File Edit Format Run Options Window Help

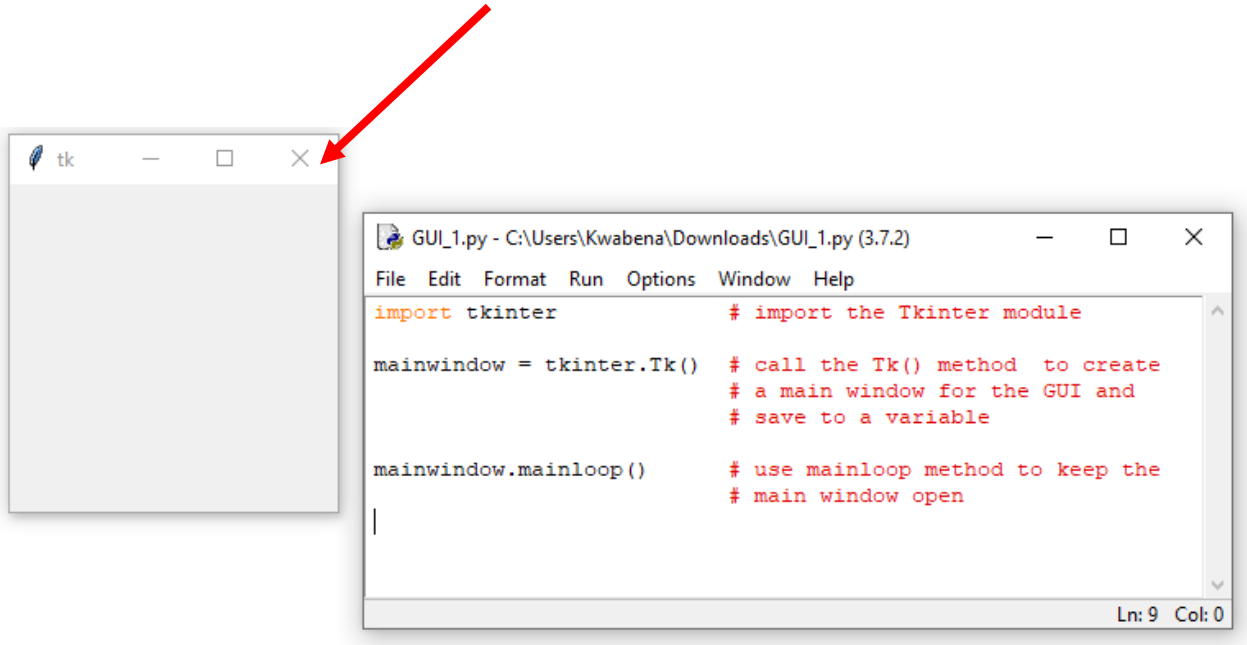
```
import tkinter              # import the Tkinter module

mainwindow = tkinter.Tk()   # call the Tk() method  to create
                            # a main window for the GUI and
                            # save to a variable

mainwindow.mainloop()       # use mainloop method to keep the
                            # main window open
```

Ln: 9 Col: 0

(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.

GUI_2.py - C:\Users\Kwabena\Downloads\GUI_2.py (3.7.2)

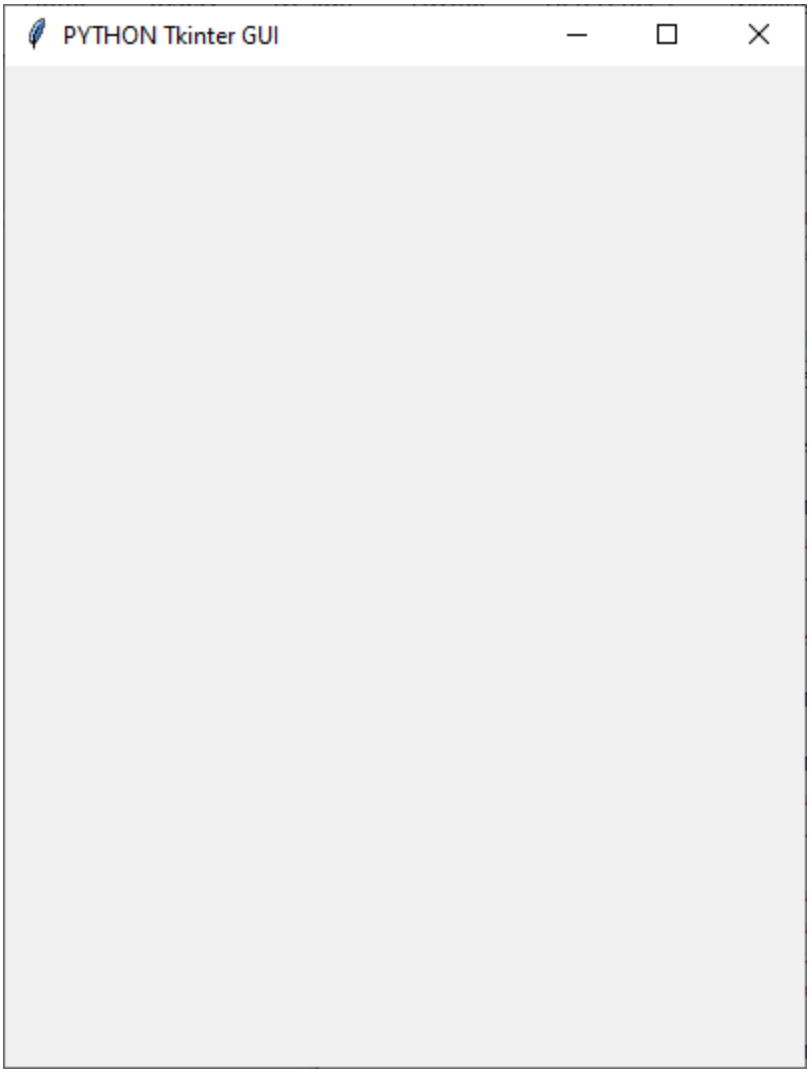
File Edit Format Run Options Window Help

```
import tkinter as tk        # import the Tkinter module and give
                            # it an alias

root = tk.Tk()              # using the alias, call the Tk() method  to
                            # create a main window called root, of  the GUI


root.title('PYTHON Tkinter GUI' )    # features or attributes of root
root.geometry("400x500")             # width by height
                                     # you may play with the numbers



root.mainloop()             # use mainloop method to keep root open
```

Ln: 16 Col: 0

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:

- Label
- Text
- Entry
- Button
- Checkbutton
- Radiobutton
- Listbox
- Spinbox
- Menubutton
- Menu
- Message
- Frame
- LabelFrame
- Canvas

The other widgets are presented in a subsequent part of this course series.

3. THE LABEL WIDGET

3.1 Label

The Label widget is used to display a single line of text or a static image. A label is used for display, thus a user does not interact with it. However, the properties (options) may be changed programmatically at any time.

The syntax is of the form,

< *variable* > = *Label* (< *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

The widget options are summarized in Table 3.1 below.

Table 3. 1: *tkinter* widget options

Option	Description	Values
anchor	controls where the text is positioned if the widget has more space than needed.	N, NE, E, SE, S, SW, W, NW, CENTER, default value = 'CENTER'
aspect	ratio of width by height as a percent	default = 150, if width option is set the option is ignored

Table 3.1 (Continued): *tkinter* widget options

Option	Description	Values
background or bg	background color of the widget	default depends on the operating system
bitmap	used to display an image	assign image object to the bitmap key
borderwidth or bd	border width	default = 2
cursor	controls the mouse cursor display when over the widget	arrow, dot, etc., default is standard cursor
font	controls the text font	default depends on the operating system
foreground or fg	controls the text color	default depends on the operating system
height	sets the vertical dimension of the widget	
image	used to display an image in the widget	set option key to image object
highlightbackground	controls how to draw the highlight region	
justify	specifies the alignment of multiple lines of text in a widget	LEFT, RIGHT, CENTER, default is CENTER
padx	controls horizontal padding (extra space to the left and right of text within the widget)	default = -1 i.e. no padding
pady	controls vertical padding	default = -1 i.e. no padding
relief	specifies a decorative border around the widget	SUNKEN, RAISED, GROOVE, RIDGE, FLAT, default is FLAT
takefocus	puts the widget is on data input focus	true, false, default = false
text	displays one or more lines of text line breaks	set key to the text string with line breaks

Table 3.1 (Continued): *tkinter* widget options

Option	Description	Values
textvariable	used with a variable and the *StringVar( )* method to display a message text	if variable is changed message text will be updated
underline	underline up to the *n*th character of a text string counting from character at position zero	set key to *n*, default = -1, i.e. no underline
width	specifies the width of the widget in character units	default is to size the widget to fit the contents
wraplength	specifies number of characters allowed per line	set key to desired value, default = 0, i.e. lines broken only by line breaks

3.2 Layout Management

All *tkinter* widgets can be arranged and managed on a master (or parent) widget by using any of the three (3) layout (or geometry) management methods (or **layout manager**). A layout manager must be used exclusively and shall not intermixed with another layout manager in the same master window.

3.2.1 The *pack( )* Method

The *pack( )* method "packs" widgets in rows or in columns on a master (or parent) widget in such a manner as to optimize the master or parent area.

The syntax is of the form,

< variable > . pack( )

The following options may be passed to the *pack* manager.

Table 3. 2: *pack( )* options

Option	Description	Values
anchor	controls where the widget is placed inside the master	default = 'CENTER'
expand	specifies whether the widgets should be expanded to fill any extra space in the master	default = false
fill	controls whether the widget should occupy all the space available from the master	X is to fill horizontally, Y is to fill vertically, BOTH, NONE keeps the original size, default = NONE
ipadx	internal padding	default = 0
ipady	internal padding	default = 0
padx	external padding	default = 0
pady	external padding	default = 0
side	determines which side to pack the widget against	TOP packs vertically, LEFT packs horizontally, default = TOP

3.2.2 The *grid( )* Method

The *grid( )* method is used to arrange widgets into a 2-dimensional tabular structure. The master widget is split into rows and columns where each cell of the table can hold a widget.

The syntax is of the form,

< variable > . grid(< options >)

The options that may be passed to the *grid* manager are shown in Table 3.3.

Table 3. 3: *grid( )* options

Option	Description	Values
column	specifies column in which widget is inserted, column numbers start from 0	if omitted, defaults to 0
columnspan	optional, used to make cell span multiple columns	default = 1
ipadx	optional, internal padding	default = 0
ipady	optional, internal padding	default = 0
padx	optional, external padding	default = 0
pady	optional, external padding	default = 0
row	specifies row in which widget is inserted, row numbers start from 0	defaults to first empty row in the grid if omitted
rowspan	optional, used to make cell span multiple rows	default = 1
sticky	determines how to expand the widget if the holding cell is larger than the widget	combination of S, N, E and W, or NW, NE, SW and SE

3.2.2 The *place( )* Method

The *place( )* method is used to explicitly set the position and size of a widget on a master, either in absolute terms or relative to another widget.

The syntax is of the form,

< variable > . place(< options >)

The options that may be passed to the *place* manager are shown in Table 3.4.

Table 3. 4: *place( )* options

Option	Description	Values
anchor	controls exact point where the widget is placed inside the master	default = NW i.e. upper left corner
bordermode	refers to the master's border	INSIDE, OUTSIDE, default = INSIDE
height	height in pixels	
relheight	height as a float between 0 and 1.0 as a fraction of the height of the master window	
relwidth	width as a float between 0 and 1.0 as a fraction of the height of the master window	
relx	horizontal offset as a float between 0 and 1.0 as a fraction of the height of the master window	
rely	vertical offset as a float between 0 and 1.0 as a fraction of the height of the master window	
width	width in pixels	
x	horizontal offset in pixels	
y	vertical offset in pixels	

3.3 Label Widget Examples

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

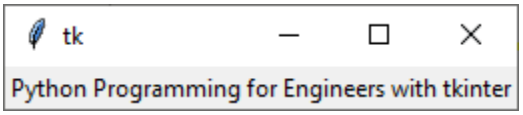
label_pack.py - E:\Python\Python Course Materials\Tutorial Files\3.3_Label\label_pack.py (3.7...

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


                         # create Label widget
                         # with the following text across the label
label1 = tk.Label(root, text = 'Python Programming for Engineers with tkinter')


label1.pack()            # how the lable1 will be geometrically set up on
                         # the root, using the pack() geometry manager


root.mainloop()
```

Ln: 15 Col: 0

Save the file.
Run the file
Look around your screen and locate the GUI.



Close the GUI.
Close the code file.
We shall now trial the other geometry manager namely *grid( )* and *place( ).*

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

Replicate the following code.

label_grid.py - E:\Python\Python Course Materials\Tutorial Files\3.3_Label\label_grid.py (3.7.2)

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


                        # create Label widgets
                        # with the following text across the label
label1 = tk.Label(root, text = ' Welcome to ')
label2 = tk.Label(root, text = ' Python Programming ')
label3 = tk.Label(root, text = ' for Engineers ')


                  # how a lablel will be geometrically set up on the root
                  # using the grid() geometry manager to place them
                  # in a rectangulay array
label1.grid(row = 0, column = 0)
label2.grid(row = 1, column = 1)
label3.grid(row = 2, column = 2)

root.mainloop()
```

Ln: 22 Col: 0

Save the file.
Run the file
Review the GUI.

Close the GUI.
Close the code file.
We shall now trial the other geometry manager namely *grid( )* and *place( ).*

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

Replicate the following code.

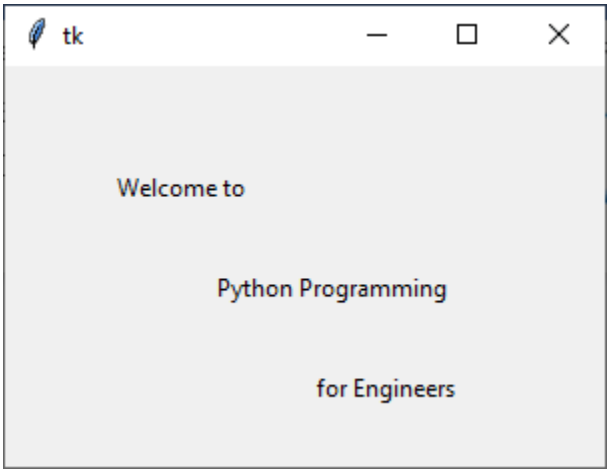
label_place.py - E:\Python\Python Course Materials\Tutorial Files\3.3_Label\label_place.py (3....

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.geometry("300x200")  # size the main window


                        # create Label widgets
                        # with the following text across the label
label1 = tk.Label(root, text = ' Welcome to ')
label2 = tk.Label(root, text = ' Python Programming ')
label3 = tk.Label(root, text = ' for Engineers ')


                  # how a lablel will be geometrically set up on the root
                  # using the place() geometry manager to place them
                  # at specified coordinates
label1.place(x = 50, y = 50)
label2.place(x = 100, y = 100)
label3.place(x = 150, y = 150)


root.mainloop()
```

Ln: 27 Col: 0

Save the file.
Run the file
Review the GUI.



Close the GUI.
Close the code file.

4. THE TEXT WIDGET

4.1 Text

The Text widget is used to display multiline text or an image. A text widget can also be used to display links and images.

The syntax is of the form,

< *variable* > = *Text* (< *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, other Text widget options are summarized in Table 4.1 below.

Table 4. 1: Text widget options

Option	Description	Values
exportselection	to export text selected in the widget to be the selection in the window manager	set to 0 if this behavior is not desired
height	sets the height of the widget in lines based on the current font size	
insertbackground	sets the color of the insertion cursor	default is black

Table 4.1 (Continued): *tkinter* widget options

Option	Description	Values
insertborderwidth	sets the width of the 3-D border around the insertion cursor	default = 0
insertofftime	sets the number of milliseconds the insertion cursor is off during its blinking cycle	default = 300, a value of 0 will suppress blinking
insertontime	sets the number of milliseconds the insertion cursor is on during its blinking cycle	default = 600
lmargin1	sets the left margin for the first line in a block of text	default = 0
lmargin2	sets the left margin for all lines except the first line in a block of text	default = 0
spacing1	controls how much extra vertical spacing is above each line of text	default = 0
spacing2	controls how much extra vertical spacing is added between displayed lines of text when a logical line wraps	default = 0
spacing3	controls how much extra vertical spacing is added below each line of text	default = 0
state	controls whether text widget responds to keyboard and mouse events	NORMAL – widget will respond, DISABLED – widget will not respond and also the contents cannot be changed programmatically
tabs	controls how tab characters can position text	
width	sets the width of the widget in characters based on the current font size	

Table 4.1 (Continued): *tkinter* widget options

Option	Description	Values
wrap	controls the display of lines of text that are too wide to fit in the widget	default is CHAR that breaks the line after the last fitting character, WORD breaks the line after the last fitting word
xscrollcommand	makes the text widget horizontally scrollable	set to *set( )* method
yscrollcommand	makes the text widget vertically scrollable	set to *set( )* method
wraplength	specifies number of characters allowed per line	set key to desired value, default = 0, i.e. lines broken only by line breaks

4.2 Text widget methods

The Text widget has the following methods summarized in Table 4.2.

Table 4. 2: Text widget methods

Method	Description
delete(<startindex> [, <endindex>])	deletes a character or range of characters of text
get(<startindex> [, <endindex>])	returns a character or range of characters of text
index(<index>)	returns the absolute value of an index based on the specified index

Table 4.2 (Continued): Text widget methods

Method	Description
insert(<index> [, <string>])	inserts string(s) at the specified index position
see(<index>)	returns true if the text located at the index position is visible, otherwise returns false

The Text widget supports **marks**, **tags** and **indexes**, collectively known as **helper structures**. A mark is used to bookmark the position between two within a given text. Methods for working with marks are summarized in Table 4.3.

Table 4. 3: Methods for marks

Method	Description
index(<mark>)	returns the line and column position of a mark
mark_gravity(<mark> [, <gravity>])	returns the gravity of the mark; if the second parameter is supplied, the gravity is set for the mark
mark_names( )	returns all marks from the widget
mark_set(<mark>, <index>)	informs a new position to the given mark
mark_unset(<mark>)	removes the specified mark from the widget

Tags are used to assign names to specific areas of text. This simplifies tasks such as modifying the display settings of specific areas of text. The methods for handling tags are summarized in Table 4.4.

Table 4. 4: Methods for tags

Method	**Description**
tag_add(<tagname>, <startindex> [,<endindex>] ...)	tags a position or a range of positions
tag_config	used to configure the tag's attributes
tag_delete(<tagname>)	deletes a specified tag
tag_remove(<tagname> [,<startindex>[.<endindex>]] ...)>)	Removes the tag from the specified area but the tag definition remains

4.3 Text Widget Example

Review your files supplied with your course materials from Suncam.
Locate a file named ***Chrysanthemum.gif***.
Copy and paste the file ***Chrysantehmum.gif*** to your Downloads folder.
(The proceeding exercise can be completed by copying and pasting the ***Chrysanthemum.gif*** to any folder of your choice, however for consistency of results let us all agree to run it out of the Downloads folder.)

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

Replicate the following code.

text.py - E:\Python\Python Course Materials\Tutorial Files\3.4_Text\text.py (3.7.2)

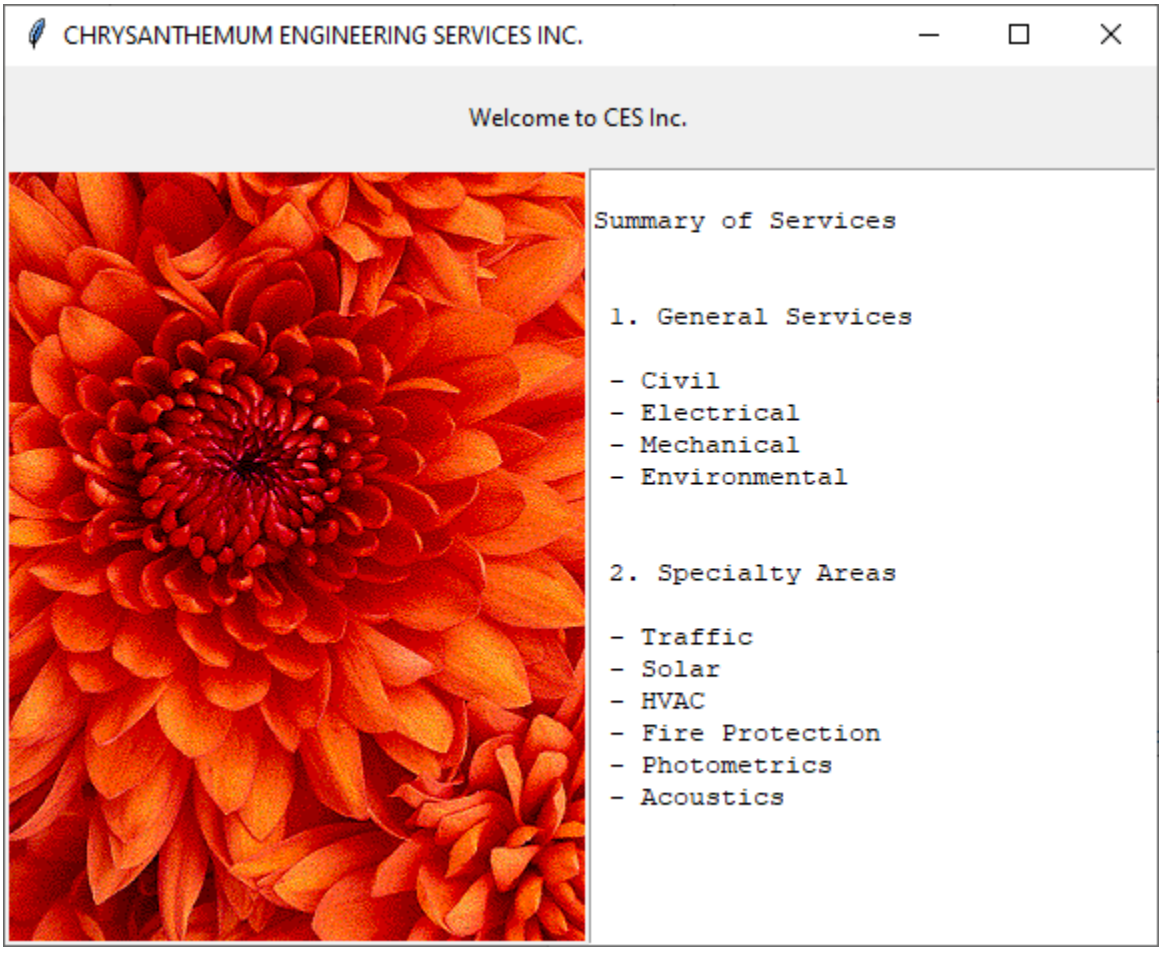
File Edit Format Run Options Window Help

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

root.title('CHRYSANTHEMUM ENGINEERING SERVICES INC.')

# we shall add add label and text widgets
# and place them on the root window using the grid layout manager

# create a Label
label1 = tk.Label(root, text = 'Welcome to CES Inc. ', height= 3)
label1.grid(row = 0, column = 0, columnspan = 2)  # to span over 2 columns

# set up tkinter acceptable PhotoImage class
photo = tk.PhotoImage(file = 'C:\\Users\\Kwabena\\Downloads\\Chrysanthemum.gif')

# create a Label
label2 = tk.Label(root, image = photo)  # insert PhotoImage to a Label
label2.grid(row = 1, column = 0)

# Text widget will hold the following multine line text
strtext = '\nSummary of Services\n\n\n \
1. General Services \n\n - Civil \n - Electrical \n - Mechanical\
\n - Environmental \n\n\n 2. Specialty Areas\n\n - Traffic\n - Solar\n - HVAC\n\
 - Fire Protection\n - Photometrics\n - Acoustics'

# create Text widget
text = tk.Text(root, width = 35)
text.insert(tk.END,strtext) # insert the multiline text into the Text widget
text.grid(row = 1, column = 1)


root.mainloop()
```

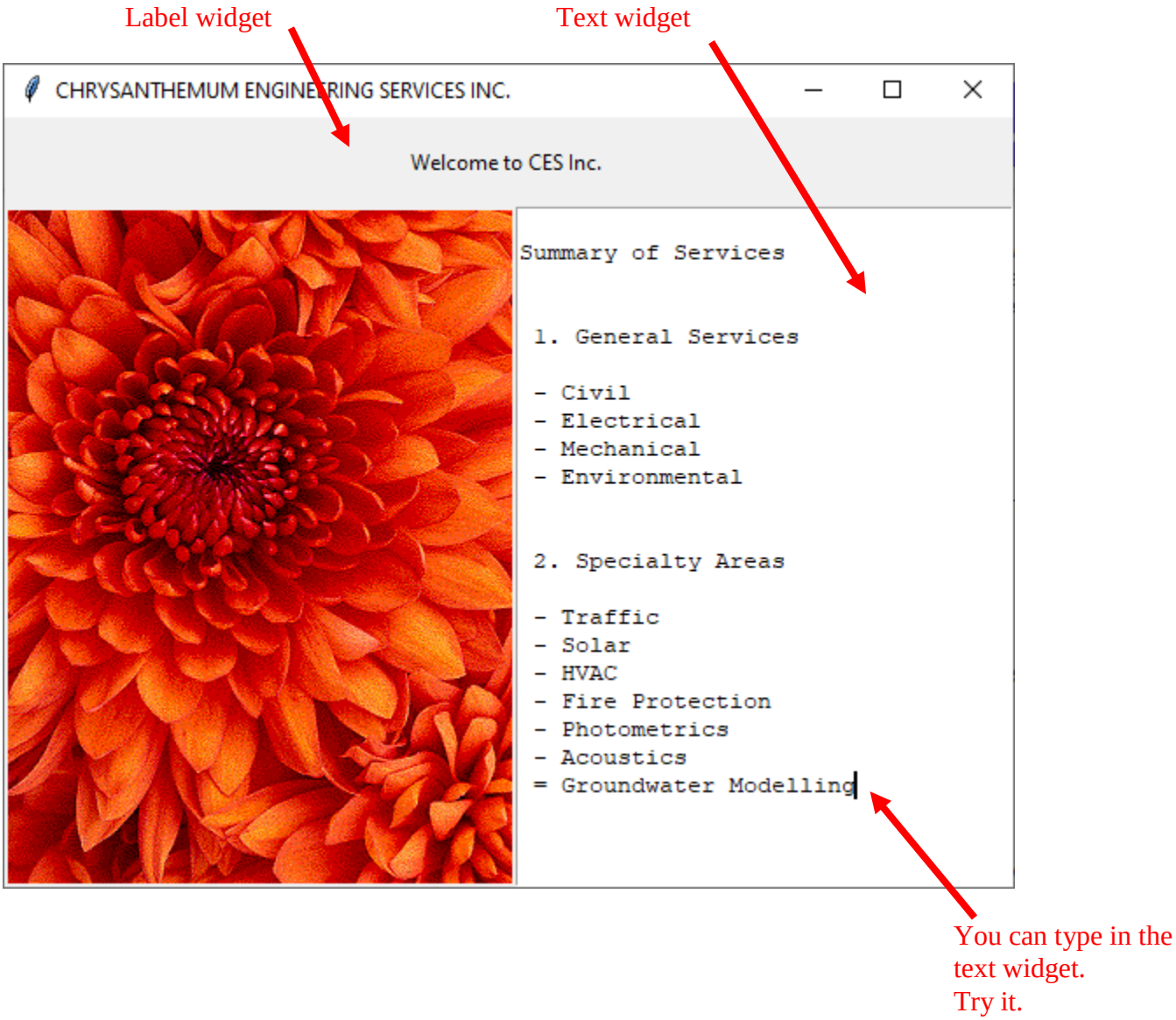
Ln: 34 Col: 0

Type the path to the file as it exists on your specific computer

Save the file.
Run the file
Review the GUI.



Note the contrasting features of a Label widget versus a Text widget.



Close the GUI.
Close the code file.

5. THE ENTRY WIDGET

5.1 Entry

The Entry widget is used to accept a single-line text string entered by a user.

The syntax is of the form,

< *variable* > = *Entry* (< *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, other Entry widget options are summarized in Table 5.1 below.

Table 5. 1: Entry widget options

Option	Description	Values
command	set to a procedure that is called anytime the user changes the state of the widget	the procedure that is called
show	controls whether the characters typed in by the user appear as-is, or they are masked by some specified character; commonly used to conceal passwords	character of choice, *, +, =, #, etc., etc.

5.2 Entry widget methods

The Entry widget has the following methods summarized in Table 5.2.

Table 5. 2: Entry widget methods

Method	Description
delete (<first>, <last>=None)	deletes characters from the first index up to but not including the last index
get( )	returns the current content of the widget as a string
index (<index>)	shifts the contents of the widget such that the character at the specified index is the left-most character visible
insert (<index>, <s>)	inserts a string before the character at the index specified
select_adjust (<index>)	ensures the selection includes the character at the given index
select_clear( )	clears the current selection
select_from (<index>)	sets the anchor to the index specified and selects that character
select_present( )	returns true if there is a selection, otherwise returns false
select_range (<first>, <last>)	during program execution, selects characters from the first index up to but not including the last index
select_to (<index>)	selects characters from the anchor up to but not including the specified index
xview (<index>)	used to connect the widget to an horizontal scrollbar
xview_scroll (<number>, <units>)	used to scroll in character widths of units, a positive number scrolls from left to right, a negative number scrolls from right to left

6. THE BUTTON WIDGET

6.1 Button

The Button widget is a "click" button that is clicked on to trigger some event. Typically, a function or a method (or some combination of both) is incorporated into the code structure of the Button. The function (or method) is assigned to the *command* option. The function (or method) is called when the Button is clicked on. The Button caption may consist of text, or images.

The syntax is of the form,

< *variable* > = *Button* (< *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

The widget options summarized Chapter 3 are applicable to the Button widget.

6.1 Button Widget Example

In this exercise we shall create a log-in window where the user enters a username and password to gain access a portal where the companies projects are managed.

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

Replicate the following code.

entry_button.py - E:\Python\Python Course Materials\Tutorial Files\3.6_Entry_Button\entry_...

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('COMPANY ACCOUNT LOGIN')
root.geometry('350x220')
# we shall add add label and text widgets
# and place them on the root window using the grid layout manager

# create a Label
label = tk.Label(root, text = '   Enter your company credentials to \
access all projects', height= 3)
label.grid(row = 0, column = 0, columnspan = 2)  # to span over 2 columns

# create a Label
label1 = tk.Label(root, text = 'Username : ')
label1.grid(row = 1, column = 0)

# create a Entry widget for user to enter their data
entry1 = tk.Entry(root)
entry1.grid(row = 1, column = 1)

# create a Label
label2 = tk.Label(root, text = 'Password : ')
label2.grid(row = 2, column = 0)

# create a Entry widget for user to enter their data
entry2 = tk.Entry(root, show = '*')  # password entries will be masked by '*'
entry2.grid(row = 2, column = 1)

# create a Label
# this label will display a message of access approval
# or denial if the user enters the correct login information
# the message will be implemented via the variable the text
# text option is set to
label3 = tk.Label(root, height = 5, text = ' ')
label3.grid(row = 3, column = 0, columnspan = 2)

# create click button the user clicks on after entereing
# login information. the command option will call the
```

Ln: 36 Col: 36

Continue as follows.

entry_button.py - E:\Python\Python Course Materials\Tutorial Files\3.6_Entry_Button\entry_...

File Edit Format Run Options Window Help

```
# and place them on the root window using the grid layout manager

# create a Label
label = tk.Label(root, text = '   Enter your company credentials to \
access all projects', height= 3)
label.grid(row = 0, column = 0, columnspan = 2)  # to span over 2 columns

# create a Label
label1 = tk.Label(root, text = 'Username : ')
label1.grid(row = 1, column = 0)

# create a Entry widget for user to enter their data
entry1 = tk.Entry(root)
entry1.grid(row = 1, column = 1)

# create a Label
label2 = tk.Label(root, text = 'Password : ')
label2.grid(row = 2, column = 0)

# create a Entry widget for user to enter their data
entry2 = tk.Entry(root, show = '*')  # password entries will be masked by '*'
entry2.grid(row = 2, column = 1)

# create a Label
# this label will display a message of access approval
# or denial if the user enters the correct login information
# the message will be implemented via the variable the text
# text option is set to
label3 = tk.Label(root, height = 5, text = ' ')
label3.grid(row = 3, column = 0, columnspan = 2)

# create click button the user clicks on after entereing
# login information. the command option will call the
# function which shall check if the correct login
# information has been entered and prompt the user accordingly
button = tk.Button(root, text = 'LOG IN', command = login)
button.grid(row = 4, column = 0, columnspan = 2)

root.mainloop()
```

Ln: 36 Col: 36

Now we need to add code that defines the function *login( )* that the Button *command* option is set to. The function runs when the Button widget is clicked on.

Above the *import tkinter* line, add the following code.

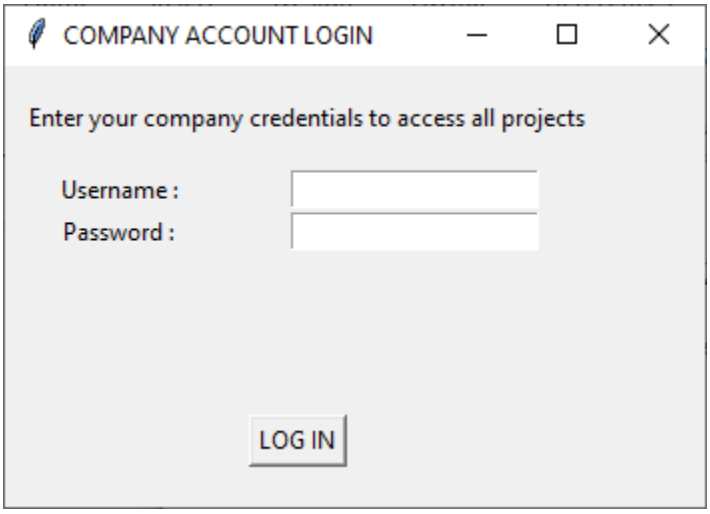
*entry_button.py - E:\Python\Python Course Materials\Tutorial Files\3.6_Entry_Button\entry...

File Edit Format Run Options Window Help

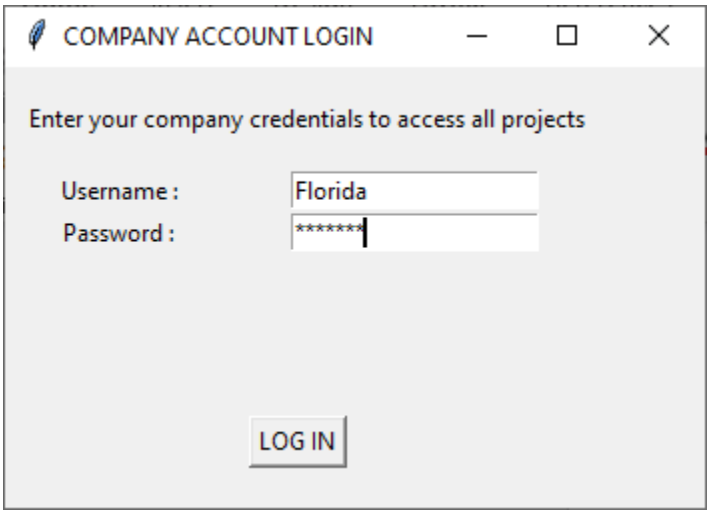
```
# the function that is called when the button is clicked on
def login():

    # initilize/ reset the label3 text variable
    p = ' '

    username = entry1.get() # pull the user name entry and
                            # assign to a variable
    password = entry2.get() # pull and assign the password

    # the correct username for this company is : Tiktaalik
    # the correct password for this account is : Obajina_123

    # now we use an if-statement to check if the entries
    # match the correct login information

    if username == 'Tiktaalik':
        pass # dont do anything
    else:
        p = p + '\nIncorrect Username! '

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


    if username == 'Tiktaalik' and password == 'Obagina_123':
        p = p + '\nAccess Granted. Proceed '
        label3.configure(fg = 'green',
                         text = p)  # modify option of existing widget
    else:
        p = p + '\nAccess Denied! Please Reenter. '
        label3.configure(fg = 'red',
                         text = p)  # modify option of existing widget
# ============================================================
import tkinter as tk    # import the tkinter module
root = tk.Tk()          # create the main window, call it root
```

Ln: 38 Col: 0

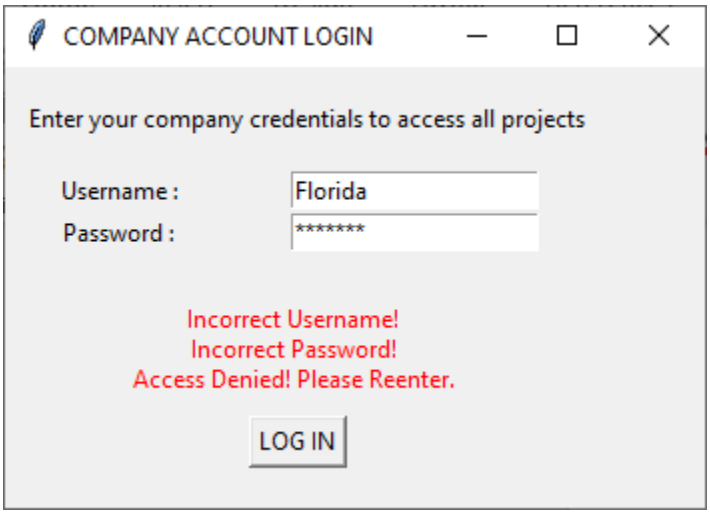
Save the file.
Run the file.
The GUI opens.



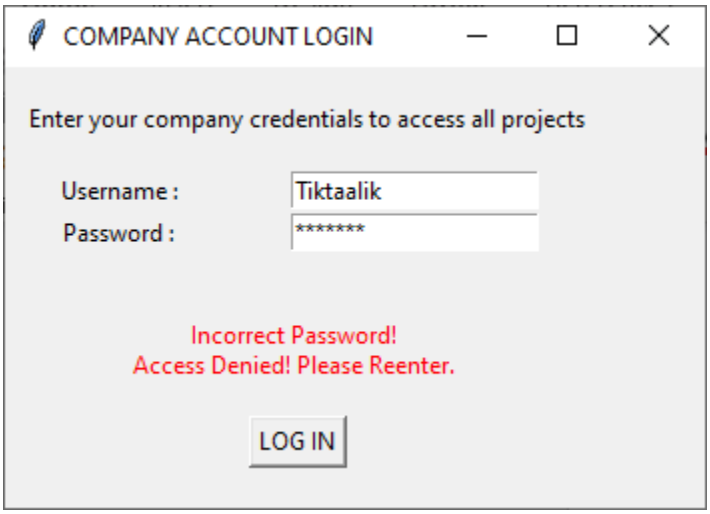
For the username enter “Florida”, which is intentionally incorrect.
For the password, enter “kmnjh78”, which is intentionally incorrect.



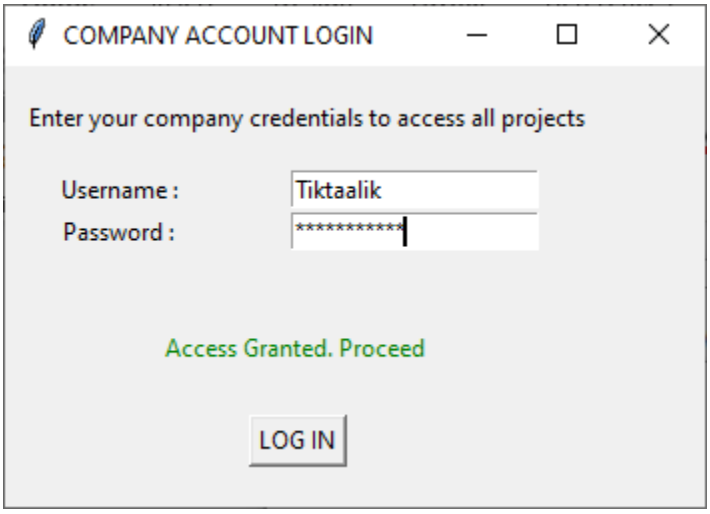
Click on the ***LOG IN*** button.
The applicable message is displayed in the label widget.



Enter the correct username, "Tiktaalik".
Click on the ***LOG IN*** button.
The applicable message is displayed in the label widget.



Enter the correct password, "Obagina_123".
Click on the ***LOG IN*** button.
The applicable message is displayed in the label widget.



Success!
Keep this file as-is. We shall update it and add more features in the next chapters.

7. THE CHECKBUTTON WIDGET

7.1 Checkbutton

The Checkbutton widget consists of a toggle button (check box) and a caption. The user clicks on the check box to select the option. The caption is typically text but may also be an image. Multiple Checkbuttons may be used and multiple selections may be made. A Checkbutton may be *on* (checked) or *off* (unchecked). A Checkbutton may be associated with a function or method which is called when the state of a Checkbutton is changed by the user.

The syntax is of the form,

< *variable* > = *Checkbutton* (< *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, other Checkbutton widget options are summarized in Table 7.1 below.

Table 7. 1: Checkbutton widget options

Option	Description	Values
command	set to a procedure that is called when the state of a check box is changed by the user	set to 0 if this behavior is not desired
disabledforeground	sets the color of the text of a disabled checkbox	the default is the stippled version of the default text color
offvalue	sets a value for the off (unchecked) state of a checkbox	default is 0, may set to an alternate value
onvalue	sets a value for the on (checked) state of a checkbox	default is 0, may set to an alternate value
state	controls whether the widget is responsive or disabled	default is NORMAL which is responsive, DISABLED makes it unresponsive and appearance will be grayed out

7.2 Checkbutton widget methods

The Checkbutton widget has the following methods summarized in Table 7.2.

Table 7. 2: Checkbutton widget methods

Method	Description
deselect( )	turns off (unchecks) the checkbutton
select( )	turns on (checks) the checkbutton
toggle( )	tums off the checkbutton if on, and turns on if off

7.3 Checkbutton Example

In this exercise we shall add a Checkbutton to the log-in window we previously created. Conduct the following updates to your log-in window code.

*check_button.py - E:\Python\Python Course Materials\Tutorial Files\3.7_Checkbutton\check_...

File Edit Format Run Options Window Help

```
entry1.grid(row = 1, column = 1)

# create a Label
label2 = tk.Label(root, text = 'Password : ')
label2.grid(row = 2, column = 0)

# create a Entry widget for user to enter their data
entry2 = tk.Entry(root, show = '*')  # password entries will be masked by '*'
entry2.grid(row = 2, column = 1)

# create a Label
# this label will display a message of access approval
# or denial if the user enters the correct login information
# the message will be implemented via the variable the text
# text option is set to
label3 = tk.Label(root, height = 5, text = ' ')
label3.grid(row = 3, column = 0, columnspan = 2)

# create a Checkbutton
# the Checkbutton will appear with a caption statement of terms of
# use of the app. the user must check to accept the terms
# before being acess to proceed into the portal
check1 = tk.IntVar()
checkbutton1 = tk.Checkbutton(root, height = 4, text =\
'I agree to the Terms and Conditions of use of this application.',\
onvalue = 1, offvalue = 0, padx = 25, variable = check1)
checkbutton1.grid(row = 4, column = 0, columnspan = 2)

# create click button the user clicks on after entereing
# login information. the command option will call the
# function which shall check if the correct login
# information has been entered and prompt the user accordingly
button = tk.Button(root, text = 'LOG IN', command = login)
button.grid(row = 5, column = 0, columnspan = 2)

root.mainloop()
```

Ln: 102 Col: 0

variable to hold onvalue/ offvalue

Adjust the *root.geometry*.

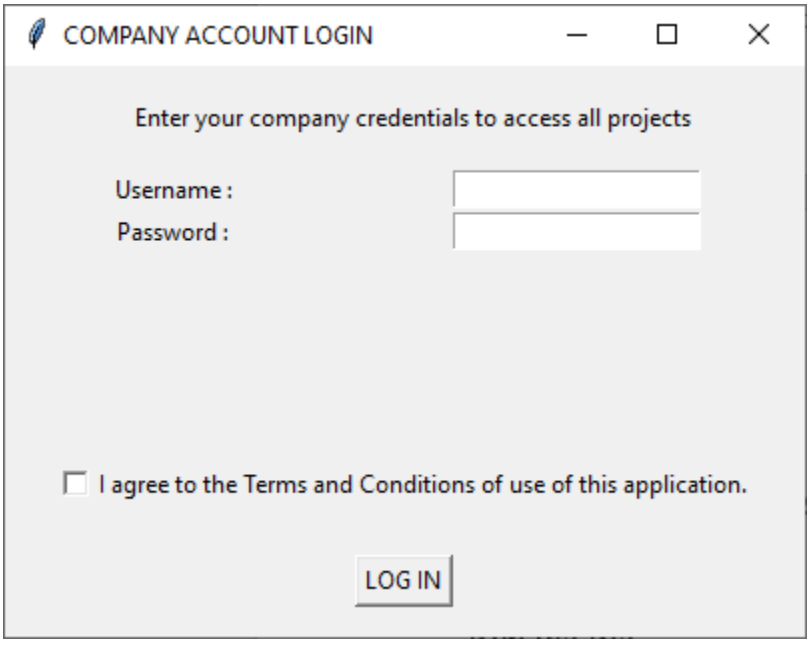
check_button.py - E:\Python\Python Course Materials\Tutorial Files\3.7_Checkbutton\check_...

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('COMPANY ACCOUNT LOGIN')
root.geometry('400x285')
# we shall add add label and text widgets
# and place them on the root window using the grid layout manager

# create a Label
label = tk.Label(root, text = '   Enter your company credentials to \
access all projects', height= 3)
label.grid(row = 0, column = 0, columnspan = 2)  # to span over 2 columns

# create a Label
label1 = tk.Label(root, text = 'Username : ')
label1.grid(row = 1, column = 0)

# create a Entry widget for user to enter their data
entry1 = tk.Entry(root)
entry1.grid(row = 1, column = 1)

# create a Label
label2 = tk.Label(root, text = 'Password : ')
label2.grid(row = 2, column = 0)

# create a Entry widget for user to enter their data
entry2 = tk.Entry(root, show = '*')  # password entries will be masked by '*'
entry2.grid(row = 2, column = 1)

# create a Label
# this label will display a message of access approval
# or denial if the user enters the correct login information
# the message will be implemented via the variable the text
# text option is set to
label3 = tk.Label(root, height = 5, text = ' ')
label3.grid(row = 3, column = 0, columnspan = 2)
```

Ln: 95 Col: 0

Save the file.
Run the file.
The GUI opens.



Close the GUI.
We shall now update the *login( )* function such that a user must accept the Terms and Conditions before they can proceed.

Conduct the following code updates.
Select the *login( )* function code.

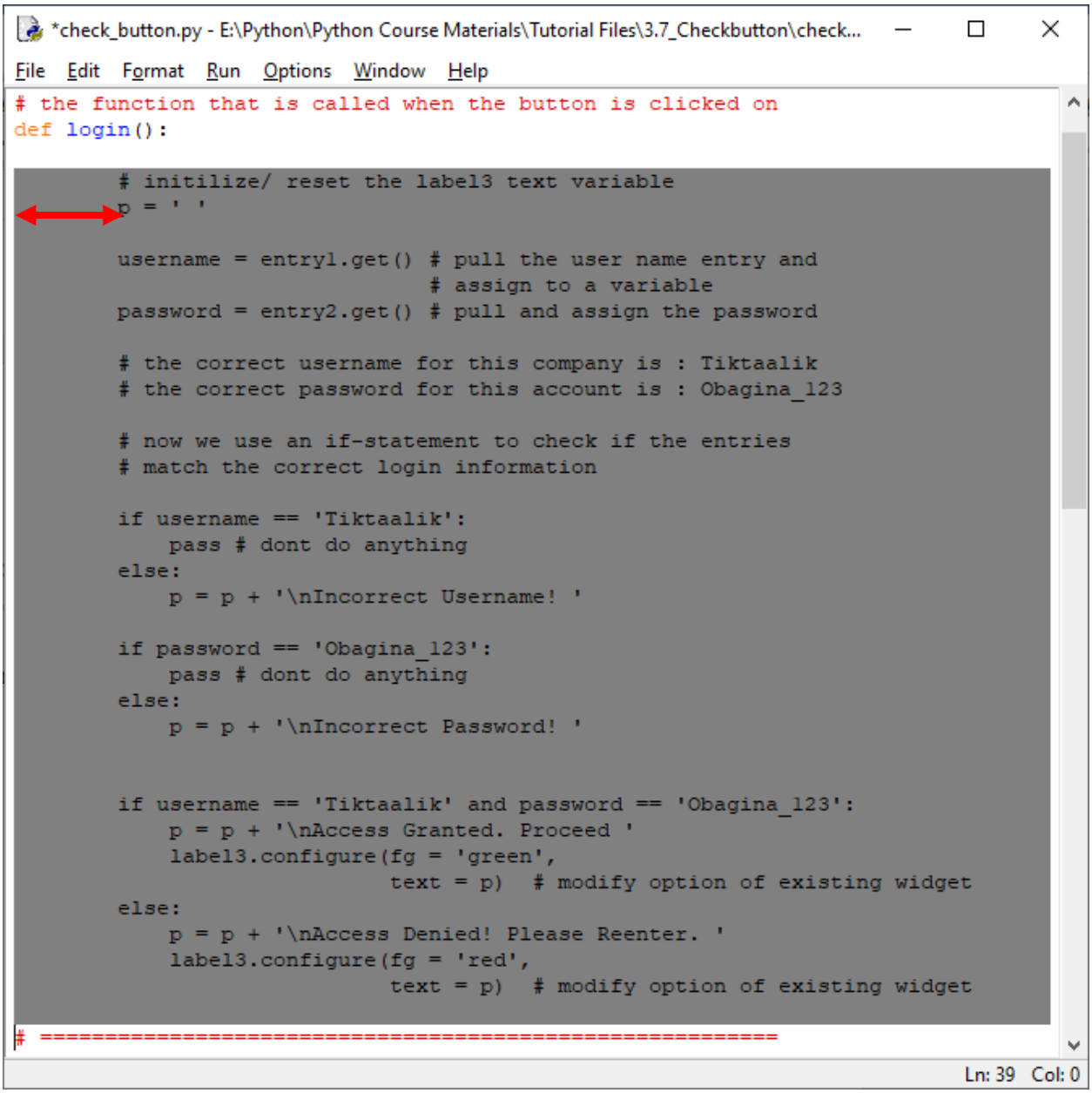
*check_button.py - E:\Python\Python Course Materials\Tutorial Files\3.7_Checkbutton\check...

File Edit Format Run Options Window Help

```
# the function that is called when the button is clicked on
def login():

    # initilize/ reset the label3 text variable
    p = ' '

    username = entry1.get() # pull the user name entry and
                            # assign to a variable
    password = entry2.get() # pull and assign the password

    # the correct username for this company is : Tiktaalik
    # the correct password for this account is : Obagina_123

    # now we use an if-statement to check if the entries
    # match the correct login information

    if username == 'Tiktaalik':
        pass # dont do anything
    else:
        p = p + '\nIncorrect Username! '

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


    if username == 'Tiktaalik' and password == 'Obagina_123':
        p = p + '\nAccess Granted. Proceed '
        label3.configure(fg = 'green',
                         text = p)  # modify option of existing widget
    else:
        p = p + '\nAccess Denied! Please Reenter. '
        label3.configure(fg = 'red',
                         text = p)  # modify option of existing widget

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

Ln: 38 Col: 8

Press the **Tab** button on your keyboard.
The indent increases.



Replicate the following code updates.

check_button.py - E:\Python\Python Course Materials\Tutorial Files\3.7_Checkbutton\check_b...

File Edit Format Run Options Window Help

```
# ============================================================
# the function that is called when the button is clicked on
def login():

    if check1.get() == 1:

        # initilize/ reset the label3 text variable
        p = ' '

        username = entry1.get() # pull the user name entry and
                                # assign to a variable
        password = entry2.get() # pull and assign the password

        # the correct username for this company is : Tiktaalik
        # the correct password for this account is : Obagina_123

        # now we use an if-statement to check if the entries
        # match the correct login information

        if username == 'Tiktaalik':
            pass # dont do anything
        else:
            p = p + '\nIncorrect Username! '

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


        if username == 'Tiktaalik' and password == 'Obagina_123':
            p = p + '\nAccess Granted. Proceed '
            label3.configure(fg = 'green',
                             text = p)  # modify option of existing widget
        else:
            p = p + '\nAccess Denied! Please Reenter. '
            label3.configure(fg = 'red',
```

Note that *get* is applied to the variable, whereas for say Entry widget, it was applied to the widget

Ln: 102 Col: 0

Complete the *if* statement for the Checkbutton response requirement.

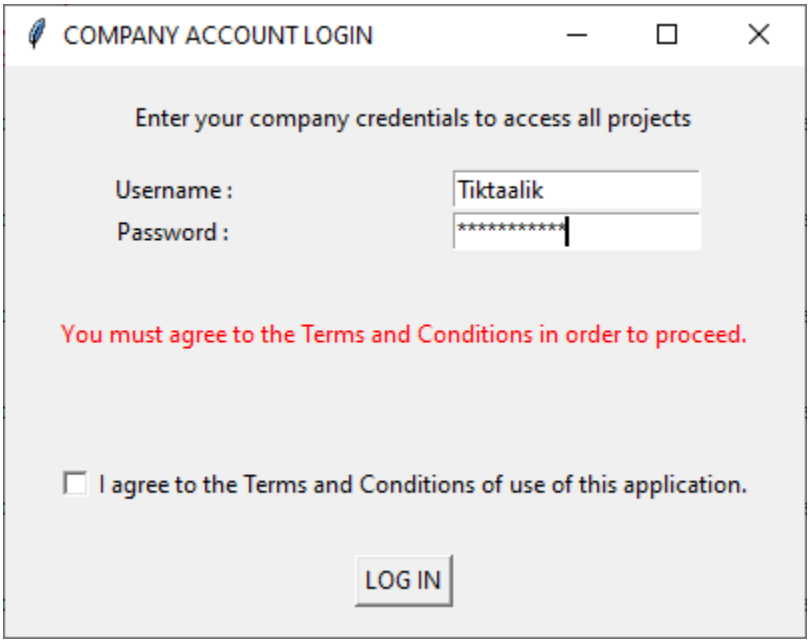
*check_button.py - E:\Python\Python Course Materials\Tutorial Files\3.7_Checkbutton\check_...

File Edit Format Run Options Window Help

```
        # the correct username for this company is : Tiktaalik
        # the correct password for this account is : Obagina_123

        # now we use an if-statement to check if the entries
        # match the correct login information

        if username == 'Tiktaalik':
            pass # dont do anything
        else:
            p = p + '\nIncorrect Username! '

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


        if username == 'Tiktaalik' and password == 'Obagina_123':
            p = p + '\nAccess Granted. Proceed '
            label3.configure(fg = 'green',
                             text = p)  # modify option of existing widget
        else:
            p = p + '\nAccess Denied! Please Reenter. '
            label3.configure(fg = 'red',
                             text = p)  # modify option of existing widget

    else:
        label3.configure(fg = 'red',\
        text = 'You must agree to the Terms and Conditions in order to proceed.')

# ============================================================

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

root.title('COMPANY ACCOUNT LOGIN')
```

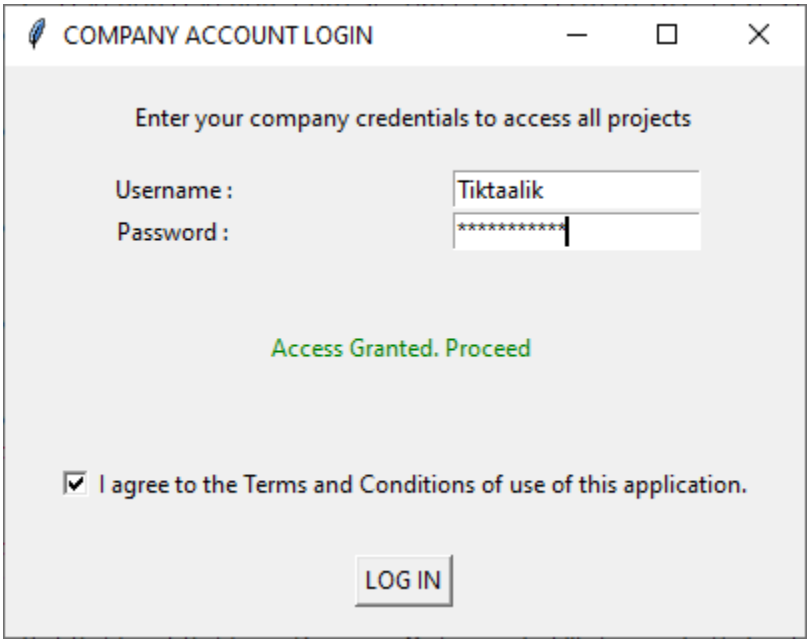
Ln: 46 Col: 0

Save the file.
Run the file.
Enter the correct username and password.
Leave the Checkbutton unchecked.
Click on the ***LOG IN*** button



Click on the Checkbutton to check it.
Click on the ***LOG IN*** button

Success!



Save your file.
We shall add other widgets to it in later chapters of this course.

8. THE RADIOBUTTON WIDGET

8.1 Radiobutton

The Radiobutton is also called an **option button**. This widget is similar to a Checkbutton, however it consists of multiple toggle button – option pairs and only one option can be selected. It is analogous to making a selection to a multiple-choice test question. The structure of the syntax consists of several Radiobutton calls assigned to the same variable.

The syntax is of the form,

< *var* > = *Radiobutton* (< *master* > , < *option* > = < *value*> , < *option*> = < *value*> , …)
< *var* > = *Radiobutton* (< *master* > , < *option* > = < *value*> , < *option*> = < *value*> , …)
< *var* > = *Radiobutton* (< *master* > , < *option* > = < *value*> , < *option*> = < *value*> , …)
:
:
:

where
< *var* > is the common variable each button is associated with
< *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

A Radiobutton may be associated with a function or method which is called when the state of a Radiobutton is changed by the user. The attributes, options and methods for Checkbuttons apply similarly to Radiobuttons.

8.2 Radiobutton Example

In this exercise we shall add Radiobuttons to the log-in window we previously created.

Conduct the following updates to your log-in window code.

*radio_button.py - E:\Python\Python Course Materials\Tutorial Files\3.8_Optionbutton\radio_...

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('PROJECT MANAGEMENT PORTAL')
root.geometry('400x450')
# we shall add add label and text widgets
# and place them on the root window using the grid layout manager

# create a Label
label = tk.Label(root, text = '   Enter your company credentials to \
access all projects', height= 3)
label.grid(row = 0, column = 0, columnspan = 2)

# create a Label
label1 = tk.Label(root, text = 'Username : ')
label1.grid(row = 1, column = 0)

# create a Entry widget for user to enter their data
entry1 = tk.Entry(root)
entry1.grid(row = 1, column = 1, sticky = tk.W)
                                  # sticky aligns the widget in the grid cell

# create a Label
label2 = tk.Label(root, text = 'Password : ')
label2.grid(row = 2, column = 0)

# create a Entry widget for user to enter their data
entry2 = tk.Entry(root, show = '*')  # password entries will be masked by '*'
entry2.grid(row = 2, column = 1, sticky = tk.W)

# create a Label
# this label will display a message of access approval
# or denial if the user enters the correct login information
# the message will be implemented via the variable the text
```

Ln: 61 Col: 65

Continuing.

radio_button.py - E:\Python\Python Course Materials\Tutorial Files\3.8_Optionbutton\radio_...

File Edit Format Run Options Window Help

```
# create a Entry widget for user to enter their data
entry1 = tk.Entry(root)
entry1.grid(row = 1, column = 1, sticky = tk.W)
                                  # sticky aligns the widget in the grid cell

# create a Label
label2 = tk.Label(root, text = 'Password : ')
label2.grid(row = 2, column = 0)

# create a Entry widget for user to enter their data
entry2 = tk.Entry(root, show = '*')  # password entries will be masked by '*'
entry2.grid(row = 2, column = 1, sticky = tk.W)

# create a Label
# this label will display a message of access approval
# or denial if the user enters the correct login information
# the message will be implemented via the variable the text
# text option is set to
label3 = tk.Label(root, height = 6, text = ' ')
label3.grid(row = 3, column = 0, columnspan = 2)

# create a label
label4 = tk.Label(root, text = '     Select your access level : ')
label4.grid(row = 4, column = 0)

# create radiobuttons
rb = tk.IntVar()
radio = tk.Radiobutton(root, text = 'Project Manager', value = 1,
                       variable = rb)
radio.grid(row = 5, column = 1, sticky = tk.W)
radio = tk.Radiobutton(root, text = 'Design Professional', value = 2,
```

Ln: 47 Col: 47

Continuing.

radio_button.py - E:\Python\Python Course Materials\Tutorial Files\3.8_Optionbutton\radio_...

File Edit Format Run Options Window Help

```
# create radiobuttons
rb = tk.IntVar()
radio = tk.Radiobutton(root, text = 'Project Manager', value = 1,
                       variable = rb)
radio.grid(row = 5, column = 1, sticky = tk.W)
radio = tk.Radiobutton(root, text = 'Design Professional', value = 2,
                       variable = rb)
radio.grid(row = 6, column = 1, sticky = tk.W)
radio = tk.Radiobutton(root, text = 'General Contractor', value = 3,
                       variable = rb)
radio.grid(row = 7, column = 1, sticky = tk.W)
radio = tk.Radiobutton(root, text = 'Sub Contractor', value = 4,
                       variable = rb)
radio.grid(row = 8, column = 1, sticky = tk.W)
radio = tk.Radiobutton(root, text = 'Regulator', value = 5,
                       variable = rb)
radio.grid(row = 9, column = 1, sticky = tk.W)

# create a Checkbutton
# the Checkbutton will appear with a caption statement of terms of
# use of the app. the user must check to accept the terms
# before being acess to proceed into the portal
check1 = tk.IntVar()
checkbutton1 = tk.Checkbutton(root, height = 4, text =\
'I agree to the Terms and Conditions of use of this application.',\
onvalue = 1, offvalue = 0, padx = 25, variable = check1)
checkbutton1.grid(row = 10, column = 0, columnspan = 2)

# create click button the user clicks on after entereing
# login information. the command option will call the
# function which shall check if the correct login
# information has been entered and prompt the user accordingly
button = tk.Button(root, text = 'LOG IN', command = login)
button.grid(row = 11, column = 0, columnspan = 2)

root.mainloop()
```

Ln: 47 Col: 47

Now, scroll up to the *login( )* function and make the following updates.

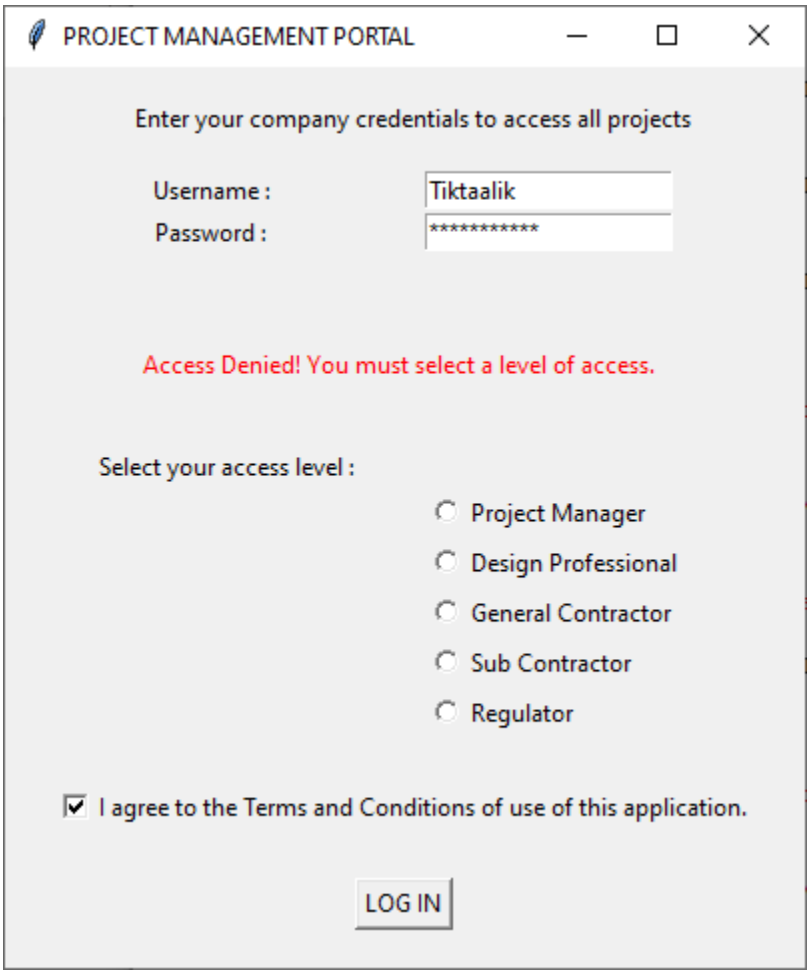
*radio_button.py - E:\Python\Python Course Materials\Tutorial Files\3.8_Optionbutton\radio_...

File Edit Format Run Options Window Help

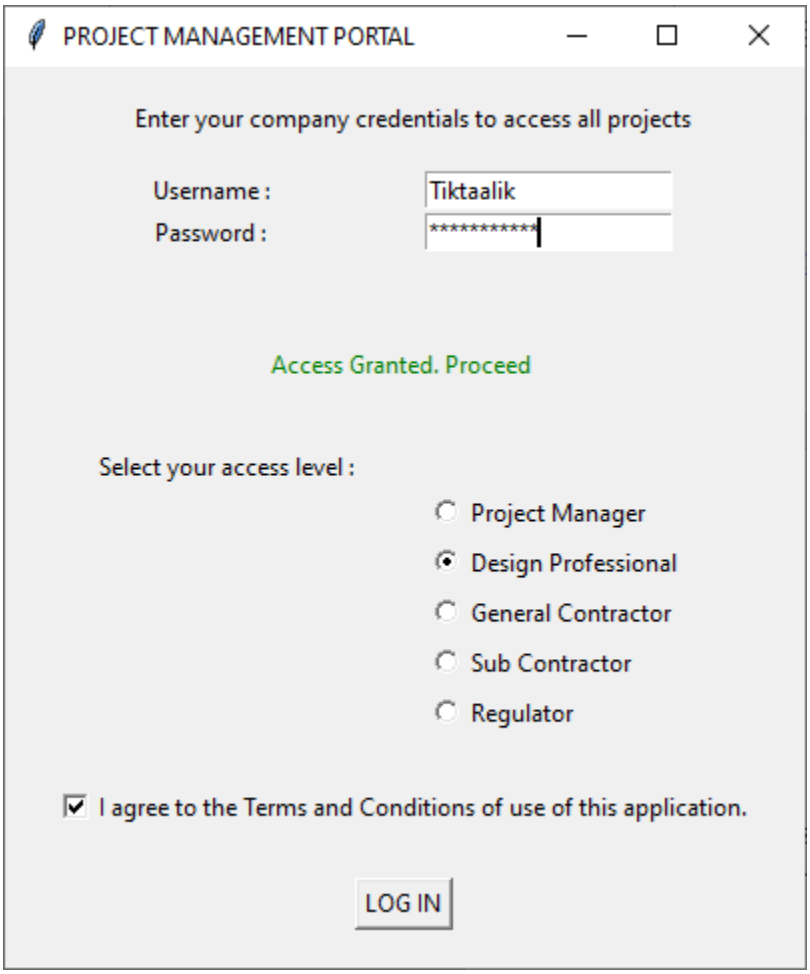
```
        if password == 'Obagina_123':
            pass # dont do anything
        else:
            p = p + '\nIncorrect Password! '


        if username == 'Tiktaalik' and password == 'Obagina_123':
            p = p + '\nAccess Granted. Proceed '
            label3.config(fg = 'green', text = p)
                                        # modify option of existing widget
        else:
            p = p + '\nAccess Denied! Please Reenter. '
            label3.config(fg = 'red', text = p)

        # enforce selection of access level with if-statement
        radiob = rb.get()
        if radiob == 1 or radiob == 2 or radiob == 3 \
           or radiob == 4 or radiob == 5:
            pass
        else:
            p = '\nAccess Denied! You must select a level of access.  '
            label3.config(fg = 'red', text = p)

    else:
        label3.configure(fg = 'red',\
        text = 'You must agree to the Terms and Conditions in order to proceed.'

# ========================================================

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

root.title('PROJECT MANAGEMENT PORTAL')
root.geometry('400x450')
# we shall add add label and text widgets
# and place them on the root window using the grid layout manager
```

Ln: 61 Col: 65

Save the file.
Run the file.
Enter the same old correct username and password.
Leave the Radiobuttons unselected.
Accept the terms and conditions of use.
Click on the ***LOG IN*** button



So, let's select an access level.
Click on the ***LOG IN*** button



Success!

9. THE LISTBOX WIDGET

9.1 Listbox

A Listbox widget is used for data entry. The user must select an item(s) from the list of items presented. The widget has four (4) modes that govern how many items may be selected. The lines of the items are indexed. A Listbox may be associated with a function or method which is called when the state of the widget is changed by the user.

The syntax to create the empty Listbox is of the form,

< var > = *Listbox* (*< master >* , *< option >* = *< value>* , *< option>* = *< value>* , …)

where
< var > 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

The syntax to add line items to the list by the *insert* method, is as of the form,

< var > . insert(< index > , < item >)
< var > . insert(< index > , < item >)
< var > . insert(< index > , < item >)
:
:
:

where the item is inserted before the line of the specified index.

In addition to the applicable widget options presented in Chapter 3, as well as the Entry widget options in Chapter 5, other Listbox widget options are summarized in Table 9.1 below.

Table 9. 1: Listbox widget options

Option	Description	Values
selectmode	governs how many items can be selected	default = BROWSE which allows one item to be selected, SINGLE, MULTIPLE - clicking on any line toggles its selection on or off, EXTENDED – enables selection of group of adjacent items
width	width of the widget by number of characters	default = 20
xscrollcommand	enables horizontal scrolling of the Listbox	
yscrollcommand	Enables vertical scrolling of the Listbox	

9.2 Listbox widget methods

The Listbox widget methods are similar to that of the Entry widget summarized in Chapter 5. Other Listbox methods are summarized in Table 9.2.

Table 9. 2: Listbox widget methods

Method	Description
activate (<index>)	selects the line specified by the index
delete (<first> [, <last>=None])	deletes lines with indexes in the range of indexes given
get (<first> [, <last>=None])	returns a tuple containing the text of the lines in the range of indexes given
index (<index>)	positions the visible items of the Listbox such that the item on the line with the index specified is at the top of the widget
insert (<index>, <elements>)	inserts new line(s) into the Listbox before the line of the specified index, or use keyword END as first parameter to add items to end of the list
size ( )	returns the number of lines of the Listbox
see (<index>)	adjusts the view such that the line at the specified index is visible

9.3 Listbox Example

In this exercise we shall update the project management portal developed in the previous chapter by replacing the Radiobuttons with a Listbox as a means for the user to select their level of access to the portal.

Conduct the following updates to your project management portal code.

Open your project management portal file that you built earlier in this course and applied Radiobuttons.

Locate the code for the Radiobuttons.

radio_button.py - E:\Python\Python Course Materials\Tutorial Files\3.8_Optionbutton\radio_...

File Edit Format Run Options Window Help

```
# create radiobuttons
rb = tk.IntVar()
radio = tk.Radiobutton(root, text = 'Project Manager', value = 1,
                       variable = rb)
radio.grid(row = 5, column = 1, sticky = tk.W)
radio = tk.Radiobutton(root, text = 'Design Professional', value = 2,
                       variable = rb)
radio.grid(row = 6, column = 1, sticky = tk.W)
radio = tk.Radiobutton(root, text = 'General Contractor', value = 3,
                       variable = rb)
radio.grid(row = 7, column = 1, sticky = tk.W)
radio = tk.Radiobutton(root, text = 'Sub Contractor', value = 4,
                       variable = rb)
radio.grid(row = 8, column = 1, sticky = tk.W)
radio = tk.Radiobutton(root, text = 'Regulator', value = 5,
                       variable = rb)
radio.grid(row = 9, column = 1, sticky = tk.W)

# create a Checkbutton
# the Checkbutton will appear with a caption statement of terms of
# use of the app. the user must check to accept the terms
# before being acess to proceed into the portal
checkl = tk.IntVar()
checkbutton1 = tk.Checkbutton(root, height = 4, text =\
'I agree to the Terms and Conditions of use of this application.',\
onvalue = 1, offvalue = 0, padx = 25, variable = checkl)
checkbutton1.grid(row = 10, column = 0, columnspan = 2)

# create click button the user clicks on after entereing
# login information. the command option will call the
# function which shall check if the correct login
# information has been entered and prompt the user accordingly
button = tk.Button(root, text = 'LOG IN', command = login)
button.grid(row = 11, column = 0, columnspan = 2)

root.mainloop()
```

Ln: 47 Col: 47

Replace the Radiobuttons code with the following.

listbox.py - E:\Python\Python Course Materials\Tutorial Files\3.9_Listbox\listbox.py (3.7.2)

File Edit Format Run Options Window Help

```
# create Listbox
L = tk.Listbox(root, height = 7, selectmode = tk.SINGLE)
                                  # 7 rows tall, one selection at a time
# add the Listbox elements
L.insert(1, ' ')
L.insert(2, 'Project Manager')
L.insert(3, 'Design Professional')
L.insert(4, 'General Contractor')
L.insert(5, 'Sub Contractor')
L.insert(6, 'Regulator')
L.selection_set(0)  # select first line by default
# add listbox widget to the grid
L.grid(row = 5, column = 1, sticky = tk.W)

# create a Checkbutton
# the Checkbutton will appear with a caption statement of terms of
# use of the app. the user must check to accept the terms
# before being acess to proceed into the portal
check1 = tk.IntVar()
checkbutton1 = tk.Checkbutton(root, height = 4, text =\
'I agree to the Terms and Conditions of use of this application.',\
onvalue = 1, offvalue = 0, padx = 25, variable = check1)
checkbutton1.grid(row = 6, column = 0, columnspan = 2)

# create click button the user clicks on after entereing
# login information. the command option will call the
# function which shall check if the correct login
# information has been entered and prompt the user accordingly
button = tk.Button(root, text = 'LOG IN', command = login)
button.grid(row = 7, column = 0, columnspan = 2)

root.mainloop()
```

Ln: 127 Col: 0

Adjust the grid parameter of the subsequent widgets.

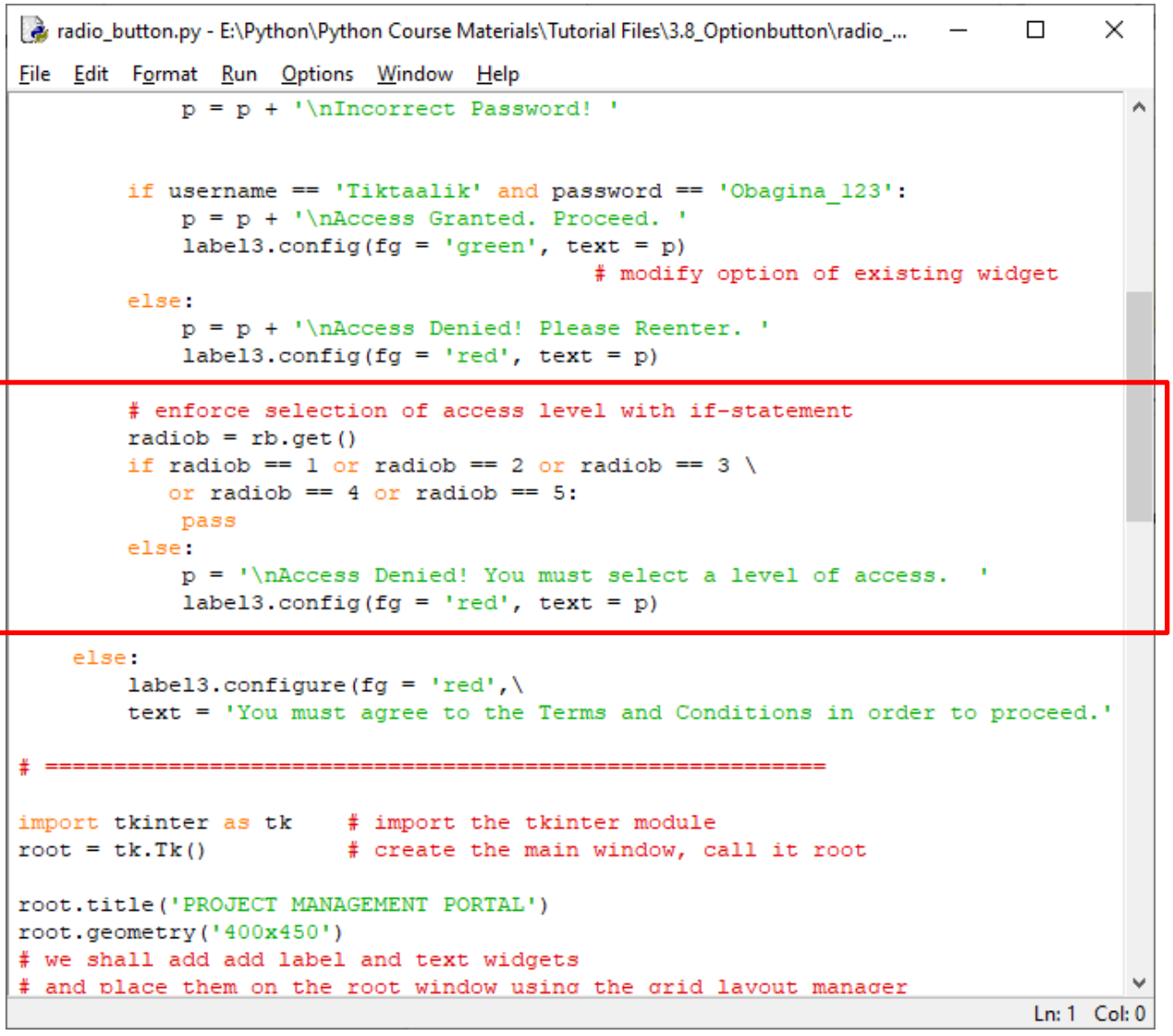
listbox.py - E:\Python\Python Course Materials\Tutorial Files\3.9_Listbox\listbox.py (3.7.2)

File Edit Format Run Options Window Help

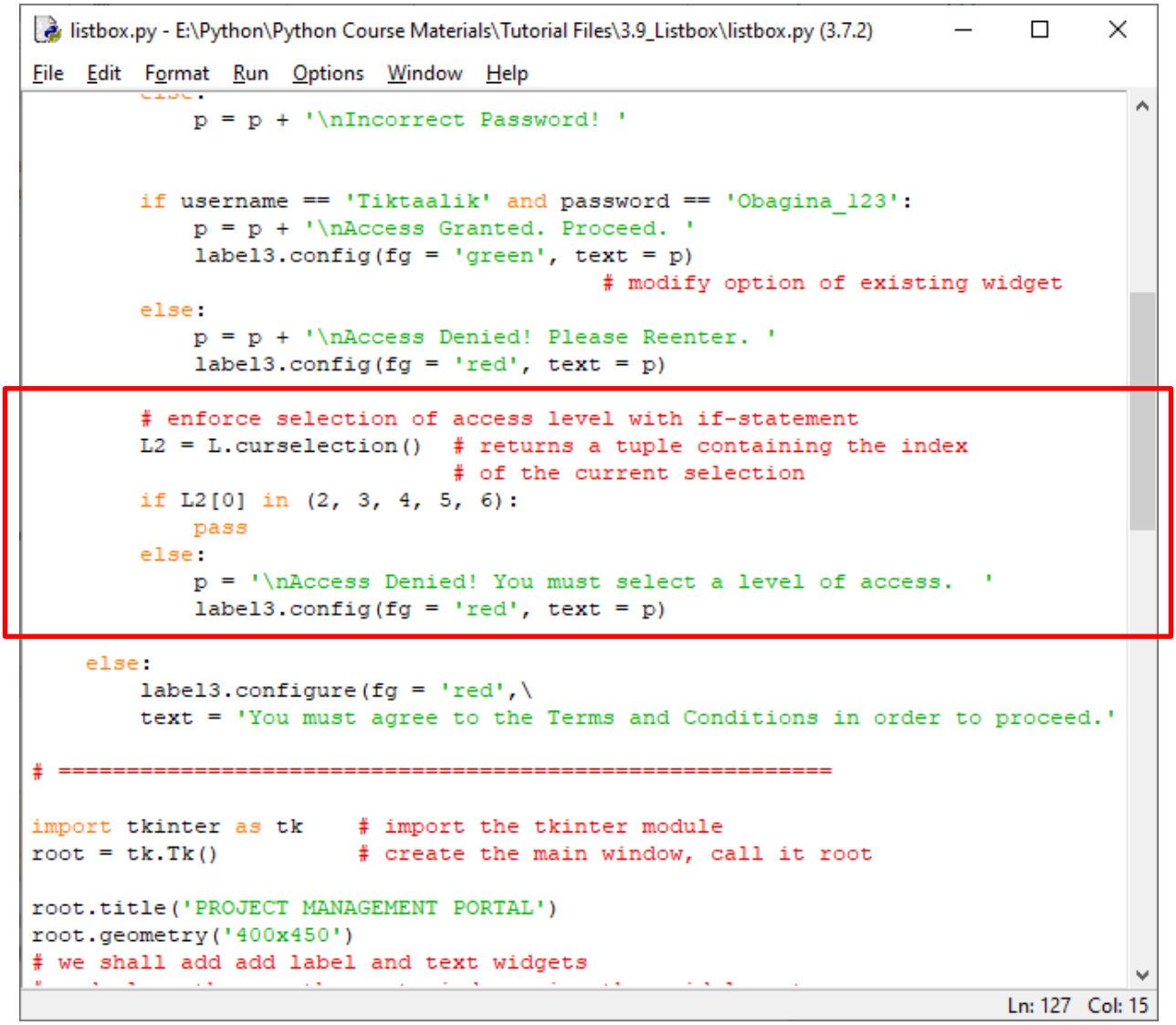
```
# create Listbox
L = tk.Listbox(root, height = 7, selectmode = tk.SINGLE)
                                  # 5 rows tall, one selection at a time
# add the Listbox elements
L.insert(1, ' ')
L.insert(2, 'Project Manager')
L.insert(3, 'Design Professional')
L.insert(4, 'General Contractor')
L.insert(5, 'Sub Contractor')
L.insert(6, 'Regulator')
# add listbox widget to the grid
L.grid(row = 5, column = 1, sticky = tk.W)

# create a Checkbutton
# the Checkbutton will appear with a caption statement of terms of
# use of the app. the user must check to accept the terms
# before being acess to proceed into the portal
check1 = tk.IntVar()
checkbutton1 = tk.Checkbutton(root, height = 4, text =\
'I agree to the Terms and Conditions of use of this application.',\
onvalue = 1, offvalue = 0, padx = 25, variable = check1)
checkbutton1.grid(row = 6, column = 0, columnspan = 2)

# create click button the user clicks on after entereing
# login information. the command option will call the
# function which shall check if the correct login
# information has been entered and prompt the user accordingly
button = tk.Button(root, text = 'LOG IN', command = login)
button.grid(row = 7, column = 0, columnspan = 2)

root.mainloop()
```

Ln: 127 Col: 15

Next we shall update the *login( )* function.
Locate the area of code that implemented the requirement for the user to make a selection from the Radiobuttons.

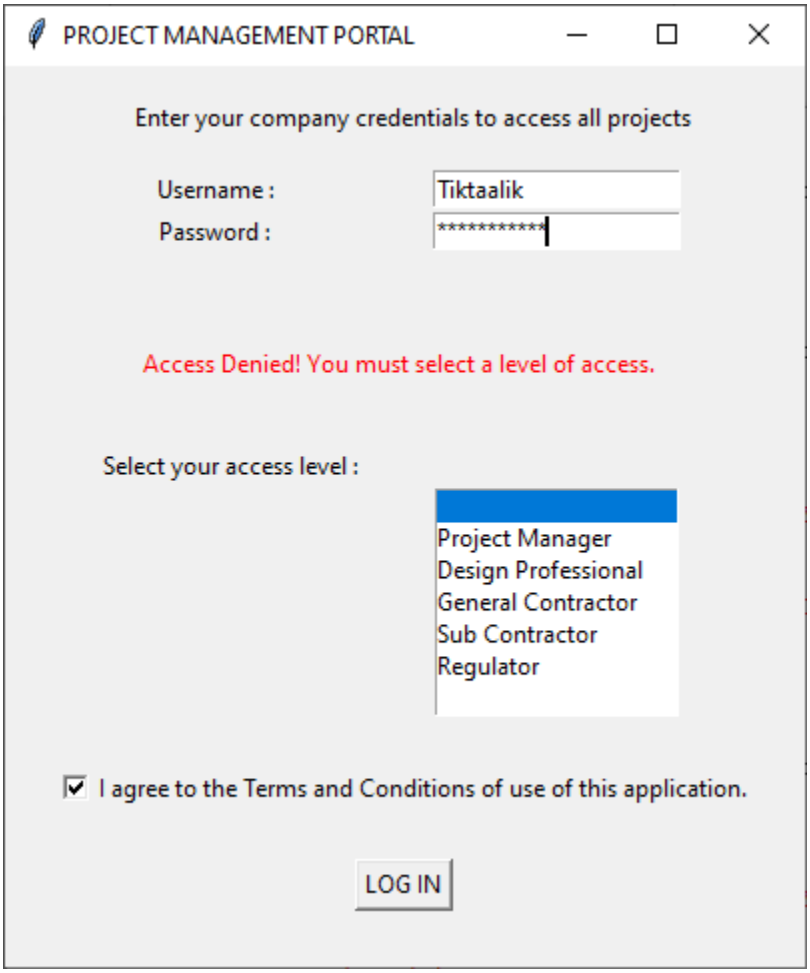


Replace the code with the following.



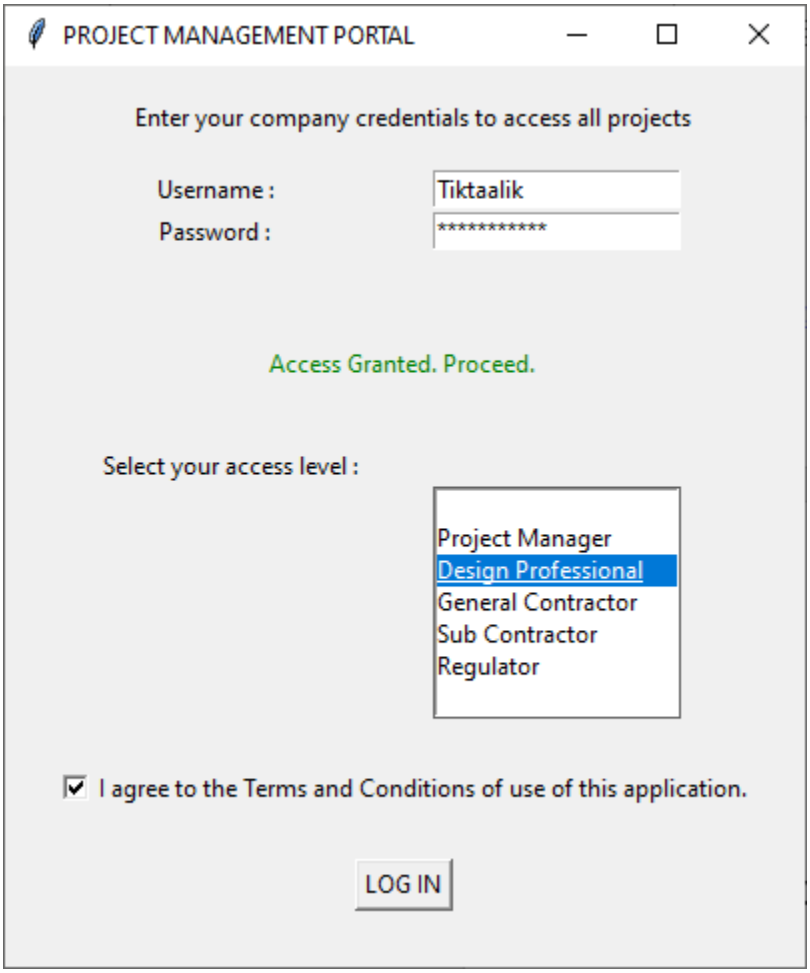
The variable *L2* is a tuple that contains the elements of the Listbox that were selected. In this case the Listbox *selectmode* restricted selections to only one (1) at a time. The *curselection( )* method pulls the selected element from the Listbox to create the tuple *L2*. The *if* statement then checks if the element at index *[0]* of tuple *L2* can be found in the list of values corresponding to the indexes of the Listbox.

Save the file.
Run the file.
Enter the appropriate information.
Make all the appropriate selections.
Click on the ***LOG IN*** button



Oops! Try again.

Select any access level.



Success!

On a side note, one common style for Listboxes is to make the widget height less than the number of lines of items. For such a Listbox, only that height (number of lines) of items will be visible to the user. The user can view the other items by hovering over the Listbox and scrolling, using the scroll wheel on your mouse.

10. THE SPINBOX WIDGET

10.1 Spinbox

The Spinbox widget is used for data entry. It is a variant of the Entry widget. The user makes a selection from a fixed number of ordered values.

The syntax is of the form,

< *variable* > = *Spinbox* (< *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, as well as the Entry widget options in Chapter 5 and the Listbox options in Chapter 9, other Spinbox widget options are summarized in Table 10.1 below.

Table 10. 1: Spinbox widget options

Option	Description	Values
from_	the minimum value of the range of ordered values available for selection	
to	the maximum value of the range of ordered values available for selection	
values	tuple that defines the values of the Spinbox available for selection, overrides the *from_*, *to* attributes	

10.2 Spinbox widget methods

The Spinbox methods are summarized in Table 10.2.

Table 10. 2: Spinbox widget methods

Method	Description
delete(<startindex> [, <endindex>])	deletes the specified character or range of text
get(<startindex> [, <endindex>])	returns the specified character or range of characters
identify(<x>, <y>)	returns widget at the specified location on the GUI
index(<index>)	returns the absolute value of an index based on the specified index

Table 10.2 (Continued): Spinbox widget methods

Method	Description
insert(<index> [, <string>]...)	inserts a string at the specified index
invoke(<element>)	invokes a Spinbox

10.3 Spinbox Example

In this exercise we shall update the project management portal developed in Chapter 8 by replacing the Radiobuttons with a Spinbox as a means for the user to select their level of access to the portal.

Conduct the following updates to your project management portal code.

Open your project management portal file that you built earlier in this course and applied Radiobuttons.

Locate the code for the Radiobuttons.

radio_button.py - E:\Python\Python Course Materials\Tutorial Files\3.8_Optionbutton\radio_...

File Edit Format Run Options Window Help

```
# create radiobuttons
rb = tk.IntVar()
radio = tk.Radiobutton(root, text = 'Project Manager', value = 1,
                       variable = rb)
radio.grid(row = 5, column = 1, sticky = tk.W)
radio = tk.Radiobutton(root, text = 'Design Professional', value = 2,
                       variable = rb)
radio.grid(row = 6, column = 1, sticky = tk.W)
radio = tk.Radiobutton(root, text = 'General Contractor', value = 3,
                       variable = rb)
radio.grid(row = 7, column = 1, sticky = tk.W)
radio = tk.Radiobutton(root, text = 'Sub Contractor', value = 4,
                       variable = rb)
radio.grid(row = 8, column = 1, sticky = tk.W)
radio = tk.Radiobutton(root, text = 'Regulator', value = 5,
                       variable = rb)
radio.grid(row = 9, column = 1, sticky = tk.W)

# create a Checkbutton
# the Checkbutton will appear with a caption statement of terms of
# use of the app. the user must check to accept the terms
# before being acess to proceed into the portal
checkl = tk.IntVar()
checkbutton1 = tk.Checkbutton(root, height = 4, text =\
'I agree to the Terms and Conditions of use of this application.',\
onvalue = 1, offvalue = 0, padx = 25, variable = checkl)
checkbutton1.grid(row = 10, column = 0, columnspan = 2)

# create click button the user clicks on after entereing
# login information. the command option will call the
# function which shall check if the correct login
# information has been entered and prompt the user accordingly
button = tk.Button(root, text = 'LOG IN', command = login)
button.grid(row = 11, column = 0, columnspan = 2)

root.mainloop()
```

Ln: 47 Col: 47

Replace the Radiobuttons code with the following.

spinbox.py - E:\Python\Python Course Materials\Tutorial Files\3.10_Spinbox\spinbox.py (3.7.2)

File Edit Format Run Options Window Help

```
# create a Label
# this label will display a message of access approval
# or denial if the user enters the correct login information
# the message will be implemented via the variable the text
# text option is set to
label3 = tk.Label(root, height = 6, text = ' ')
label3.grid(row = 3, column = 0, columnspan = 2)

# ------------------------------------------------------------
# create a Text
text5 = tk.Text(root, height = 8, width = 25)
text5.insert(tk.END, 'Access level codes :\n\n')
text5.insert(tk.END, ' 1 : Project Manager\n')
text5.insert(tk.END, ' 2 : Design Professional\n')
text5.insert(tk.END, ' 3 : General Contractor\n')
text5.insert(tk.END, ' 4 : Sub Contractor\n')
text5.insert(tk.END, ' 5 : Regulator\n')

text5.grid(row = 4, column = 0, rowspan = 2)

# create a label
label4 = tk.Label(root, height = 3, text = 'Select your access level :')
label4.grid(row = 4, column = 1, sticky = tk.S)

# create Spinbox
S = tk.Spinbox(root, from_ = 0, to = 5)  # values are 0 through 5
# add Spinbox widget to the grid
S.grid(row = 5, column = 1, sticky = tk.N)
#------------------------------------------------------------

# create a Checkbutton
# the Checkbutton will appear with a caption statement of terms of
# use of the app. the user must check to accept the terms
# before being acess to proceed into the portal
```

Ln: 61 Col: 16

Adjust the grid parameter of the subsequent widgets.

spinbox.py - E:\Python\Python Course Materials\Tutorial Files\3.10_Spinbox\spinbox.py (3.7.2)

File Edit Format Run Options Window Help

```
text5.insert(tk.END, ' 4 : Sub Contractor\n')
text5.insert(tk.END, ' 5 : Regulator\n')

text5.grid(row = 4, column = 0, rowspan = 2)

# create a label
label4 = tk.Label(root, height = 3, text = 'Select your access level :')
label4.grid(row = 4, column = 1, sticky = tk.S)

# create Spinbox
S = tk.Spinbox(root, from_ = 0, to = 5)  # values are 0 through 5
# add Spinbox widget to the grid
S.grid(row = 5, column = 1, sticky = tk.N)
#------------------------------------------------------------

# create a Checkbutton
# the Checkbutton will appear with a caption statement of terms of
# use of the app. the user must check to accept the terms
# before being acess to proceed into the portal
check1 = tk.IntVar()
checkbutton1 = tk.Checkbutton(root, height = 4, text =\
'I agree to the Terms and Conditions of use of this application.',\
onvalue = 1, offvalue = 0, padx = 25, variable = check1)
checkbutton1.grid(row = 6, column = 0, columnspan = 2)

# create click button the user clicks on after entereing
# login information. the command option will call the
# function which shall check if the correct login
# information has been entered and prompt the user accordingly
button = tk.Button(root, text = 'LOG IN', command = login)
button.grid(row = 7, column = 0, columnspan = 2)

root.mainloop()
```

Ln: 61 Col: 16

Next we shall update the *login( )* function.
Locate the area of code that implemented the requirement for the user to make a selection from the Radiobuttons.

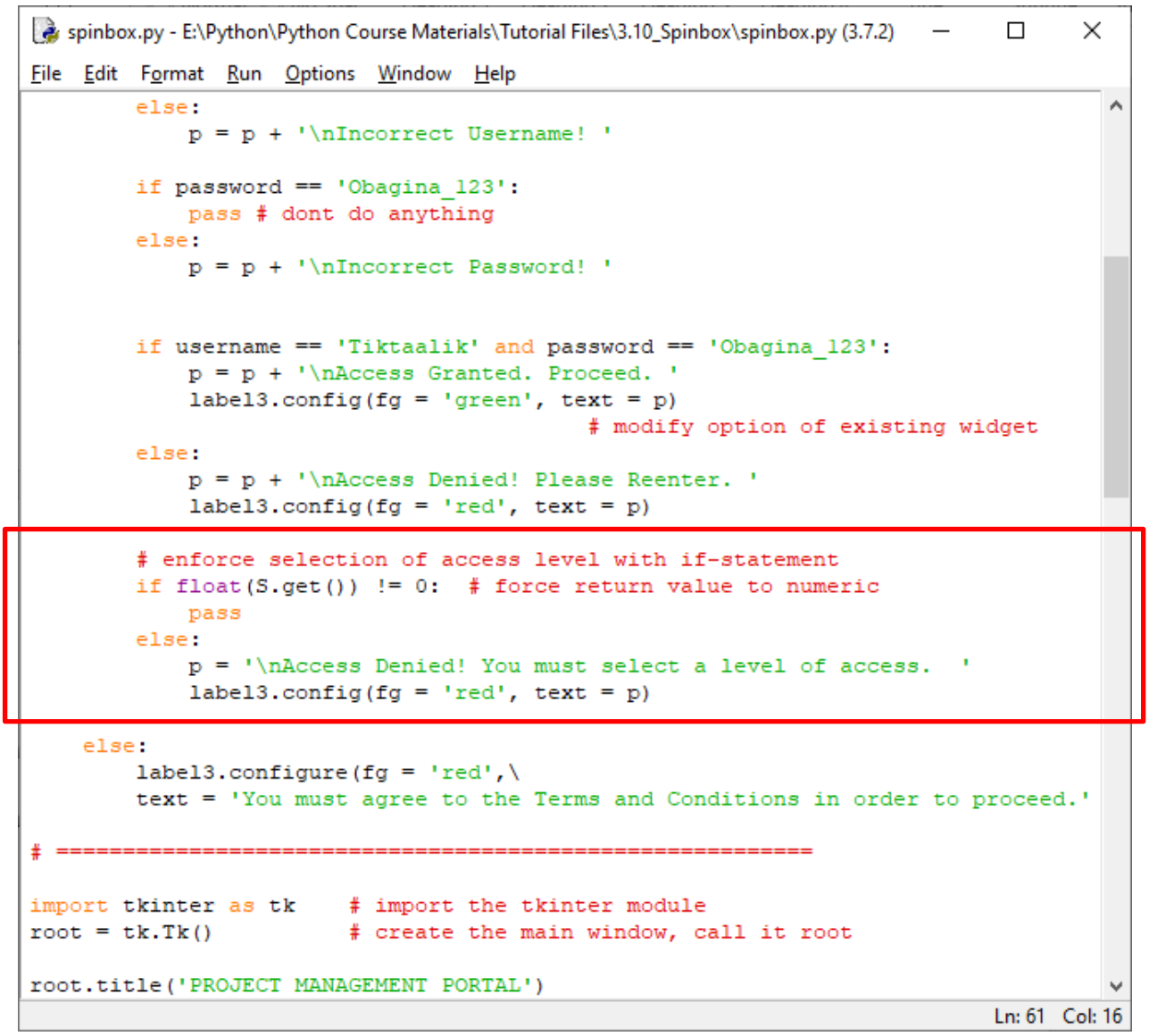
radio_button.py - E:\Python\Python Course Materials\Tutorial Files\3.8_Optionbutton\radio_...

File Edit Format Run Options Window Help

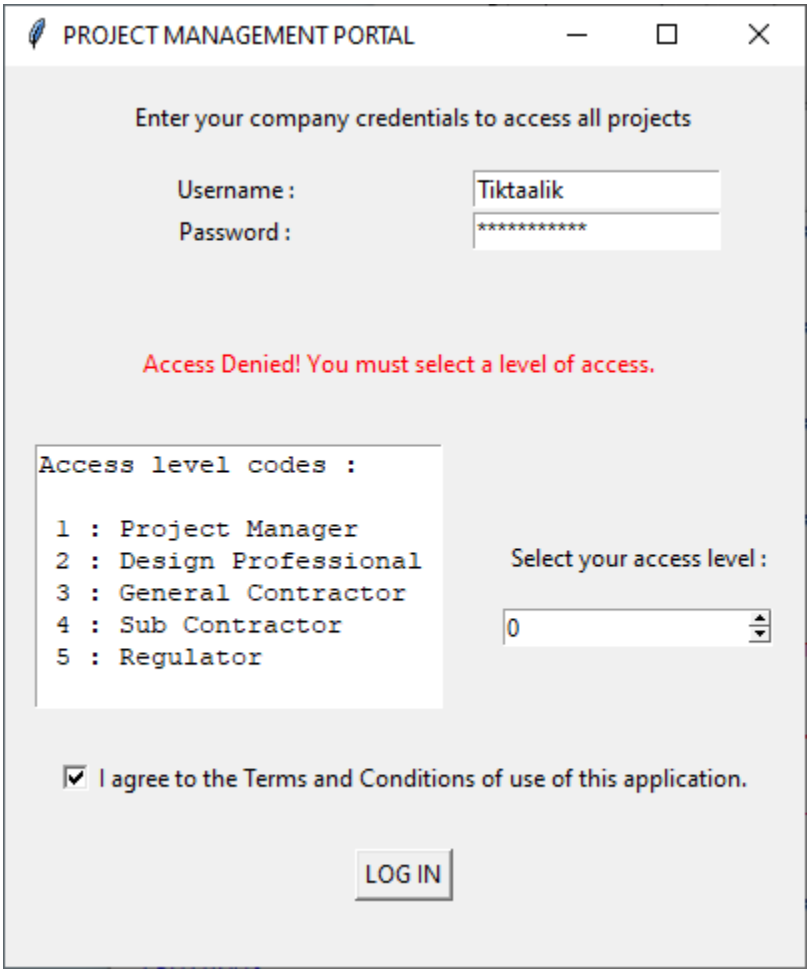
```
            p = p + '\nIncorrect Password! '


        if username == 'Tiktaalik' and password == 'Obagina_123':
            p = p + '\nAccess Granted. Proceed. '
            label3.config(fg = 'green', text = p)
                                        # modify option of existing widget
        else:
            p = p + '\nAccess Denied! Please Reenter. '
            label3.config(fg = 'red', text = p)

        # enforce selection of access level with if-statement
        radiob = rb.get()
        if radiob == 1 or radiob == 2 or radiob == 3 \
           or radiob == 4 or radiob == 5:
            pass
        else:
            p = '\nAccess Denied! You must select a level of access.  '
            label3.config(fg = 'red', text = p)

    else:
        label3.configure(fg = 'red',\
        text = 'You must agree to the Terms and Conditions in order to proceed.'

# ============================================================

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

root.title('PROJECT MANAGEMENT PORTAL')
root.geometry('400x450')
# we shall add add label and text widgets
# and place them on the root window using the grid layout manager
```

Ln: 1 Col: 0

Replace the code with the following.

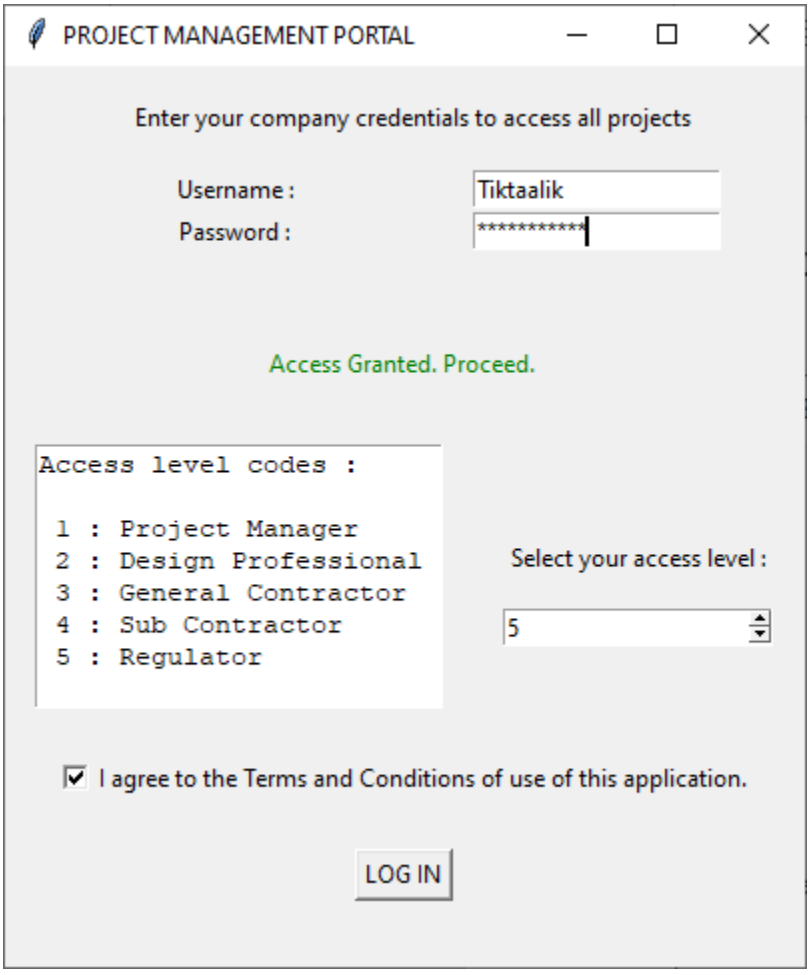


Save the file.
Run the file.
Enter the appropriate information.
Make all the appropriate selections.
Click on the ***LOG IN*** button



Oops! Try again.

Select any access level.



Success!

11. THE MENUBUTTON WIDGET

11.1 Menubutton

A Menubutton is a button widget that opens a drop-down Menu widget when clicked on. The Menu widget displays the multiple choices available for its associated Menubutton when the Menubutton is clicked on. Thus, the Menubutton is the element that remains visible regardless of whether the drop-down Menu is open and visible or not.

The syntax is of the form,

< *variable* > = *Menubutton* (< *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

The Menubutton caption may is typically text but may also be an image.

In addition to the applicable widget options presented in Chapter 3, other Menubutton widget options are summarized in Table 11.1 below.

Table 11. 1: Menubutton widget options

Option	Description	Values
direction	controls on what side of the Menubutton the Menu appears	LEFT, RIGHT ABOVE, default is BELOW
disabledforeground	sets the color of the text of a disabled Menubutton	the default is the stippled version of the default text color
menu	this option is set to the Menu widget that contains the multiple choices available to the user, note that the Menu must also have been set to the master window	the name of the Menu widget

12. THE MENU WIDGET

12.1 Menu

A Menu provides the list of multiple choices associated with a Menubutton.

The syntax to create the empty Menu is of the form,

< var > = Menu (< master > , < option > = < value> , < option> = < value> , …)

where
< var > 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

The syntax to add items to the Menu, is as of the form,

< var > . add_command (< option > = < value> , < option> = < value> , …)
< var > . add_command (< option > = < value> , < option> = < value> , …)
< var > . add_command (< option > = < value> , < option> = < value> , …)
:
:
:

where the items are added to the same variable.

In addition to the applicable widget options presented in Chapter 3, as well as the Menubutton widget options in Chapter 11, other Menu widget options are summarized in Table 12.1 below.

Table 12. 1: Menu widget options

Option	Description	Values
postcommand	set to a procedure that is called anytime the Menu is opened	set to the name of the procedure
tearoff	controls whether a Menu can be detached from the Menubutton to form a floating Menu	
title	the title of a tearoff Menu window	

12.2 Menu widget methods

The Menu widget has the following methods summarized in Table 12.2.

Table 12. 2: Menu widget methods

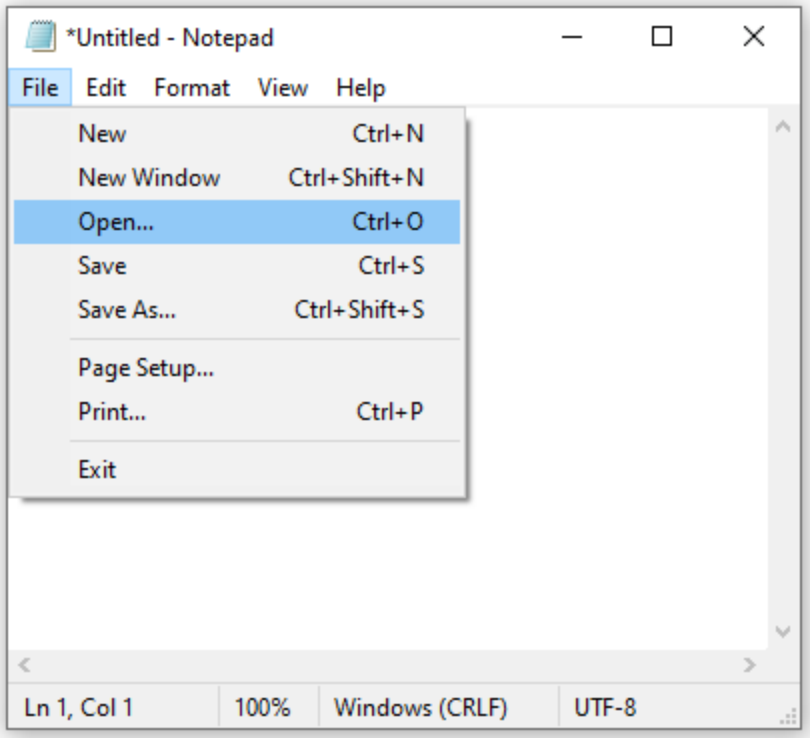
Method	Description
add_command (<options>)	adds an item to a Menu widget
add_radiobutton(<options>)	adds a Radiobutton item to a Menu widget
add_checkbutton(<options>)	adds a Checkbutton item to a Menu widget
add_cascade(<options>)	associates a menu to a parent Menu, to create a hierarchy of Menus.
add_separator( )	creates a line separator in the Menu

Table 12.2 (Continued): Menu widget methods

Method	**Description**
add(<type>, <options>)	adds a specified type of item to a Menu
delete(<startindex> [,<endindex>])	deletes a range of indexes of Menu items
entryconfig(<index>, <options>)	to change an option(s) of a Menu item at the specified index
index(<item>)	returns the index of a given menu item
insert_separator (<index>)	inserts a line separator in the indexed item

12.3 Menu Example

In this exercise we shall replicate drop-down menus commonly encountered in many desktop applications. For example,



Replicate the following code.

menu_button.py - E:\Python\Python Course Materials\Tutorial Files\3.12_Menus\menu_butt...

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('MAIN WINDOW')
root.geometry('300x350')

# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
# Create a Menu
# when clicked on it displays a multiple
# choice list or menu called menu2 that
# we click on to select an item which typically
# calls some function

menu1 = tk.Menu(root)
# attach the "main" menu to the main window
# kind of like "layout geomtery"
root.config(menu = menu1)

# set up menu2 which is a menu
# attached to menu1
menu2 = tk.Menu(menu1)

# set up header for menu2 that will
# always be visible on menu1
menu1.add_cascade(label = 'File', menu = menu2)
# add the menu2 items
menu2.add_command(label = 'New', command = NewFunction)
menu2.add_command(label = 'Open', command = OpenFunction)
menu2.add_command(label = 'Close', command = CloseFunction)
menu2.add_separator()  # a fancy seperator line
menu2.add_command(label = 'Exit', command = ExitFunction)

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

Ln: 16 Col: 15

Continue as follows.

menu_button.py - E:\Python\Python Course Materials\Tutorial Files\3.12_Menus\menu_butt...

File Edit Format Run Options Window Help

```
# kind of like "layout geomtery"
root.config(menu = menu1)

# set up menu2 which is a menu
# attached to menu1
menu2 = tk.Menu(menu1)

# set up header for menu2 that will
# always be visible on menu1
menu1.add_cascade(label = 'File', menu = menu2)
# add the menu2 items
menu2.add_command(label = 'New', command = NewFunction)
menu2.add_command(label = 'Open', command = OpenFunction)
menu2.add_command(label = 'Close', command = CloseFunction)
menu2.add_separator()  # a fancy seperator line
menu2.add_command(label = 'Exit', command = ExitFunction)

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

# lets add header and elements for a menu3
# remember we are attaching this menu3 to main menu3
menu3 = tk.Menu(menu1)
# menu3 header
menu1.add_cascade(label = 'Edit', menu = menu3)
# menu3 items
menu3.add_command(label = 'Undo', command = UndoFunction)
menu3.add_command(label = 'Redo', command = RedoFunction)
menu3.add_command(label = 'Cut', command = CutFunction)
menu3.add_command(label = 'Copy', command = CopyFunction)
menu3.add_command(label = 'Paste', command = PasteFunction)
menu3.add_command(label = 'Delete', command = DeleteFunction)

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

root.mainloop()
```

Ln: 16 Col: 15

At the top of the code, add the code for the functions associated with the menus' items command options, as follows.

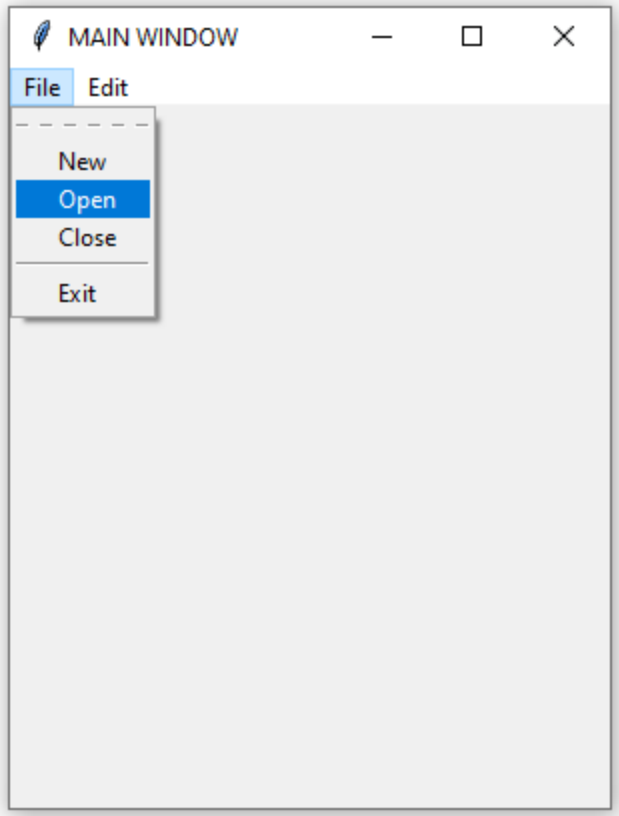
menu_button.py - E:\Python\Python Course Materials\Tutorial Files\3.12_Menus\menu_butt...

File Edit Format Run Options Window Help

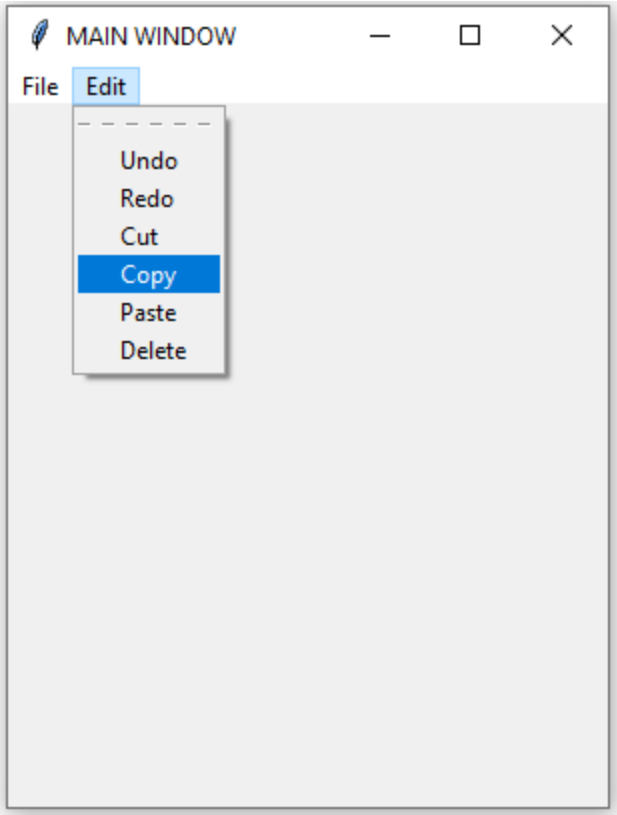
```
# ============================================================
# the functions called when the menu items are clicked on
# these very simple functions will print a statement in your
# IDLE window confirming the menu works as desogned
def NewFunction():
    print('New is working')

def OpenFunction():
    print('Open is working')

def CloseFunction():
    print('Close is working')

def ExitFunction():
    print('Exit is working')
    root.destroy() # PAY ATTENTION TO THIS ONE. SEE WHAT IT DOES.
                   # IT WILL SHUT DOWN YOUR PYTHON APP !!
                   # or try root.quit()
def UndoFunction():
    print('Undo is working')

def RedoFunction():
    print('Redo is working')

def CutFunction():
    print('Cut is working')

def CopyFunction():
    print('Copy is working')

def PasteFunction():
    print('Paste is working')

def DeleteFunction():
    print('Delete is working')
# ============================================================

import tkinter as tk    # import the tkinter module
```

Ln: 18 Col: 57

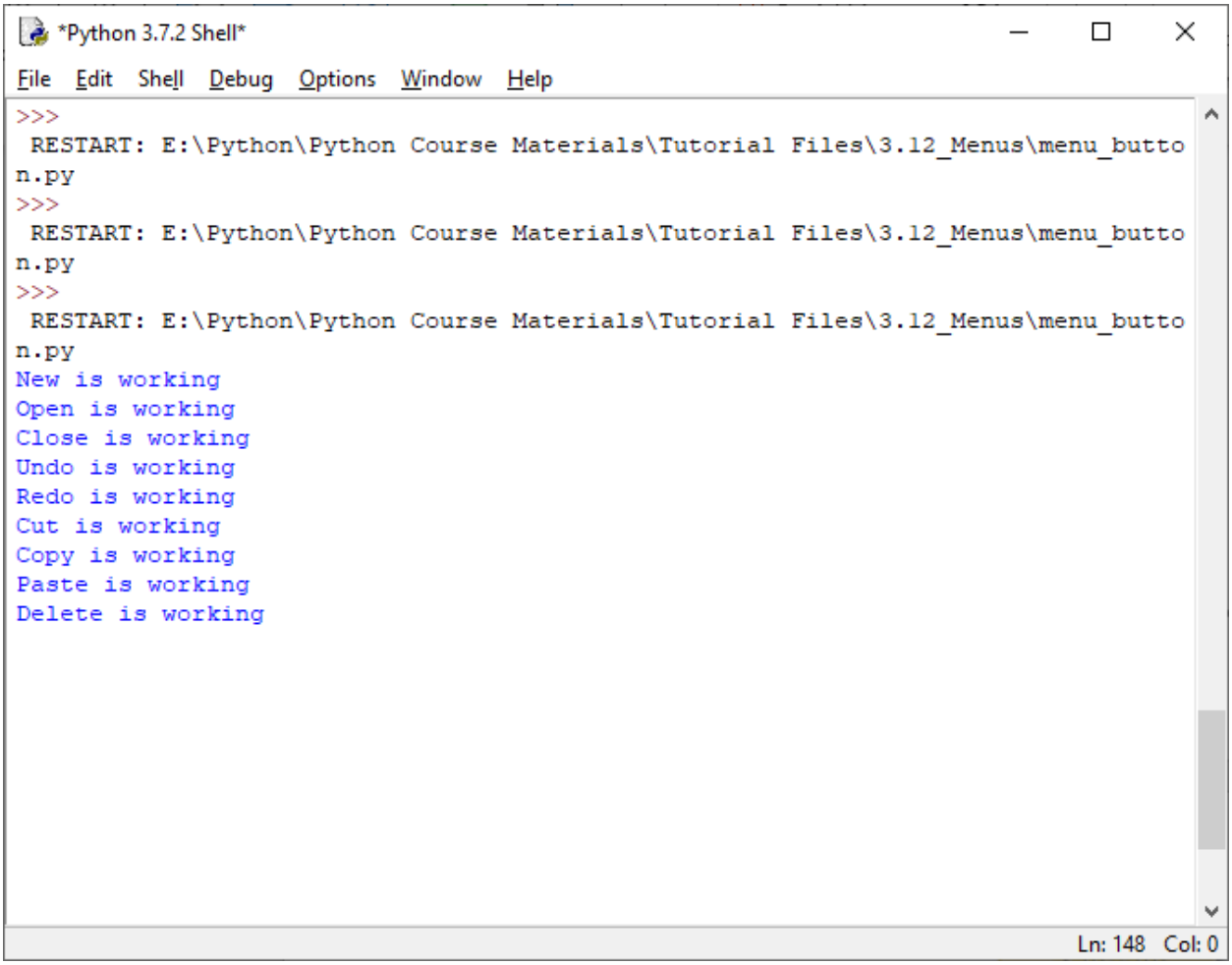
Save the file.
Run the file.
Review the menu items.
Click on some of the menu items under the **File** header.
(Do not click on the **Exit** option yet).
Note the separator line between **Close** and **Exit**.



Click on some of the menu items under the **Edit** header.



Review the output in the IDLE window.



Click **File** – **Exit** to close the application.

13. THE MESSAGE WIDGET

13.1 Message

The message widget is similar in functionality to the Label widget, however, it provides a multiline text that is automatically broken into lines and justified. Also, the text will be automatically wrapped to maintain a given width or aspect ratio.

The syntax is of the form,

< variable > = Message (< 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

The options and methods are similar to those of the Label widget.

14. THE FRAME WIDGET

14.1 Frame

The Frame widget is used as a container to organize and manage other widgets. It is particularly useful for large or complex GUI windows. Frames can be used to "split" the widgets into groups and a group can be managed as a single unit through the Frame. The Frame serves as the "master" for the widgets that are organized on it. The geometry management methods can be applied to individual Frames to achieve versatile organization and display of the widgets.

The syntax is of the form,

< variable > = Frame (< 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

A widget can then be added to the window, directly to the Frame, as follows,

< variable > = < Widget > (< Frame > , < options >)

The general widget options discussed in Chapter 3, generally apply to Frames.

15. THE LABELFRAME WIDGET

15.1 LabelFrame

The LabelFrame is used as a container widget, and as a spacer and organizer for GUIs with complex widget layouts. Widgets are added to the LabelFrame in the same manner as for Frames.

The syntax is of the form,

< variable > = LabelFrame (< 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

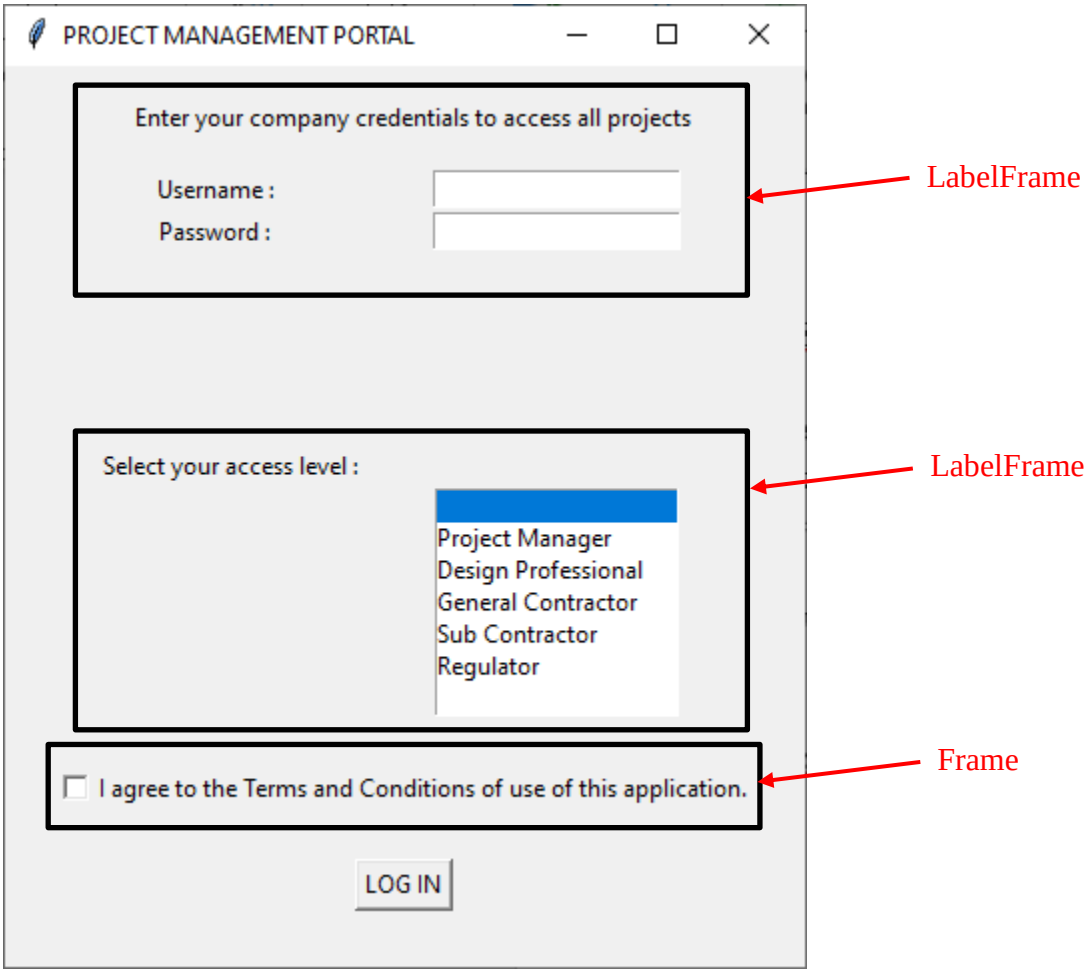
The LabelFrame widget has the same features as the Frame but it also has the capability to display a label. The label is implemented by assigning a string of the desired text to the *text* option.

15.2 Frame and LabelFrame Example

In this exercise we shall add a Frame(s) and LabelFrame(s) to the project management portal app we developed in Chapter 9 of this course.

Open your project management portal file that you built earlier in this course and applied a Listbox.

We shall organize the widgets onto Frames/ LabelFrames as follows.



We must create the Frames/ LabelFrames and associate each widget with its new relevant Frame/ LabelFrame.

Update the code as follows.

frame_lblframe.py - E:\Python\Python Course Materials\Tutorial Files\3.15_Frames_LblFrames...

File Edit Format Run Options Window Help

```
root.title('PROJECT MANAGEMENT PORTAL')
root.geometry('400x480')
# we shall add add label and text widgets
# and place them on the root window using the grid layout manager

# create labelframe1
lblframe1 = tk.LabelFrame(root, text = 'Log In')
lblframe1.grid(row = 0, column = 0, columnspan = 2, rowspan = 2)

# create a Label
label = tk.Label(lblframe1, text = '   Enter your company credentials to \
access all projects', height= 3)
label.grid(row = 0, column = 0, columnspan = 2)

# create a Label
label1 = tk.Label(lblframe1, text = 'Username : ')
label1.grid(row = 1, column = 0)

# create a Entry widget for user to enter their data
entry1 = tk.Entry(lblframe1)
entry1.grid(row = 1, column = 1, sticky = tk.W)
                                  # sticky aligns the widget in the grid cell

# create a Label
label2 = tk.Label(lblframe1, text = 'Password : ')
label2.grid(row = 2, column = 0)

# create a Entry widget for user to enter their data
entry2 = tk.Entry(lblframe1, show = '*')  # password entries will be masked by '
entry2.grid(row = 2, column = 1, sticky = tk.W)

# create a Label
# this label will display a message of access approval
# or denial if the user enters the correct login information
# the message will be implemented via the variable the text
# text option is set to
label3 = tk.Label(root, height = 6, text = ' ')
label3.grid(row = 3, column = 0, columnspan = 2)
```

Ln: 96 Col: 0

Continue.

frame_lblframe.py - E:\Python\Python Course Materials\Tutorial Files\3.15_Frames_LblFrames...

File Edit Format Run Options Window Help

```
                                    # sticky aligns the widget in the grid cell

# create a Label
label2 = tk.Label(lblframe1, text = 'Password : ')
label2.grid(row = 2, column = 0)

# create a Entry widget for user to enter their data
entry2 = tk.Entry(lblframe1, show = '*')  # password entries will be masked by '
entry2.grid(row = 2, column = 1, sticky = tk.W)

# create a Label
# this label will display a message of access approval
# or denial if the user enters the correct login information
# the message will be implemented via the variable the text
# text option is set to
label3 = tk.Label(root, height = 6, text = ' ')
label3.grid(row = 3, column = 0, columnspan = 2)

# create LabelFrame2
lblframe2 = tk.LabelFrame(root)
lblframe2.grid(row = 4, column = 0, columnspan = 2, rowspan = 2)

# create a labe2
label4 = tk.Label(lblframe2, text = '     Select your access level : ')
label4.grid(row = 4, column = 0)

# create Listbox
L = tk.Listbox(lblframe2, height = 7, selectmode = tk.SINGLE)
                                   # 7 rows tall, one selection at a time
# add the Listbox elements
L.insert(1, ' ')
L.insert(2, 'Project Manager')
L.insert(3, 'Design Professional')
L.insert(4, 'General Contractor')
L.insert(5, 'Sub Contractor')
L.insert(6, 'Regulator')
L.selection_set(0)  # select first line by default
# add listbox widget to the grid
L.grid(row = 5, column = 1, sticky = tk.W)
```

Ln: 96 Col: 0

Continue.

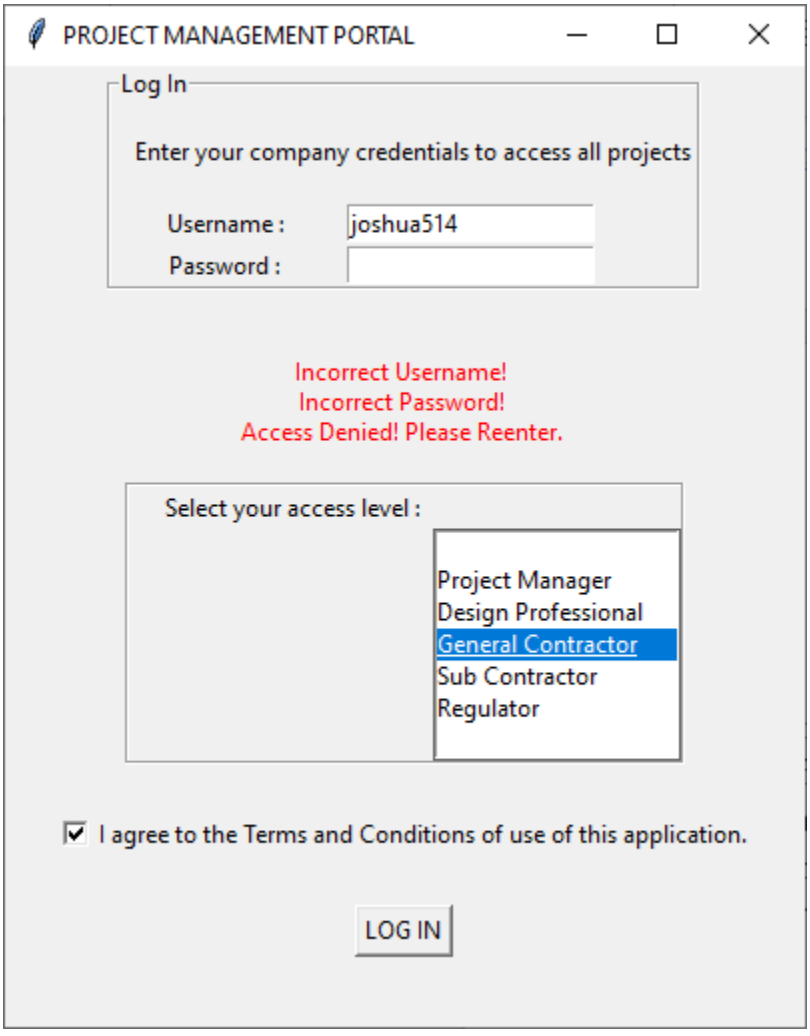
frame_lblframe.py - E:\Python\Python Course Materials\Tutorial Files\3.15_Frames_LblFrames...

File Edit Format Run Options Window Help

```
label4 = tk.Label(lblframe2, text = '     Select your access level : ')
label4.grid(row = 4, column = 0)

# create Listbox
L = tk.Listbox(lblframe2, height = 7, selectmode = tk.SINGLE)
                              # 7 rows tall, one selection at a time
# add the Listbox elements
L.insert(1, ' ')
L.insert(2, 'Project Manager')
L.insert(3, 'Design Professional')
L.insert(4, 'General Contractor')
L.insert(5, 'Sub Contractor')
L.insert(6, 'Regulator')
L.selection_set(0)  # select first line by default
# add listbox widget to the grid
L.grid(row = 5, column = 1, sticky = tk.W)

# create labelframe3
frame3 = tk.Frame(root)
frame3.grid(row = 6, column = 0, columnspan = 2)

# create a Checkbutton
# the Checkbutton will appear with a caption statement of terms of
# use of the app. the user must check to accept the terms
# before being acess to proceed into the portal
check1 = tk.IntVar()
checkbutton1 = tk.Checkbutton(frame3, height = 4, text =\
'I agree to the Terms and Conditions of use of this application.',\
onvalue = 1, offvalue = 0, padx = 25, variable = check1)
checkbutton1.grid(row = 6, column = 0, columnspan = 2)

# create click button the user clicks on after entereing
# login information. the command option will call the
# function which shall check if the correct login
# information has been entered and prompt the user accordingly
button = tk.Button(root, text = 'LOG IN', command = login)
button.grid(row = 7, column = 0, columnspan = 2)

root.mainloop()
```

Ln: 96 Col: 0

Save the file.
Run the file.
Enter some information.
Make some selections.
Click on the ***LOG IN*** button



Consider ways to make the LabelFrames more artistically pleasing. This can be achieved by playing with combinations of options such as *sticky*, *anchor*, *padding*, *ipadx*, *ipady*, margins, Frames/ LabelFrames as spacers, etc., etc., and the list goes on.

16. THE CANVAS WIDGET

16.1 Canvas

A canvas is a widget on which sketches, diagrams etc. are drawn. Also, a canvas may hold graphics, text, Frames and other widgets.

The syntax is of the form,

< variable > = Canvas (< 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

The general widget options discussed in Chapter 3, generally apply to Canvases.

16.2 Drawing Objects

A summary of selected drawing objects supported by the Canvas widget is as follows.

16.2.1 Line

The syntax to create a line is of the form,

line = canvas.create_line (x_0, y_0, x_1, y_1, x_2, y_2, ... , x_n, y_n, < options >)

where the x-y pairs are the coordinates of points on the line

16.2.2 Polygon

The syntax to create a polygon is of the form,

polygon = canvas.create_polygon (x_0, y_0, x_1, y_1, x_2, y_2, … , x_n, y_n, < options >)

where the x-y pairs are the coordinates of the vertices of the polygon, and at least three (3) vertices must be specified

16.2.3 Arc

The syntax to create an arc of a circle is of the form,

arc = canvas.create_arc (< coordinates > , < options >)

The options such as *fill*, *start*, *extent*, may be manipulated to produce an arc, a chord, or a sector of a circle.

16.2.4 Oval

The syntax to create a circle, or an ellipse, is of the form,

oval = canvas.create_oval (< coordinates > , < options >)

where two pairs are the coordinates are required, for an ellipse the coordinates of the top left and bottom right corners of the bounding rectangle must be specified.

16.2.5 Image

The syntax to create an image item – BitmapImage or PhotoImage, is of the form,

image = canvas.create_image (< coordinates > , < options >)

where the coordinates shall be for the insertion point of the object, and the options include *filename* which is set to the name of the file containing the image or the full path to the file.

16.3 Simple Canvas Example

In this exercise we shall add a very simple Canvas widget to project management portal developed in Chapter 8 using the Radiobuttons. We shall add a very simple Canvas widget as the company logo place it next to the ***LOG IN*** button.

Conduct the following updates to your project management portal code.

Open your project management portal file that you built earlier in this course and applied Radiobuttons.

Make the following updates to the code.

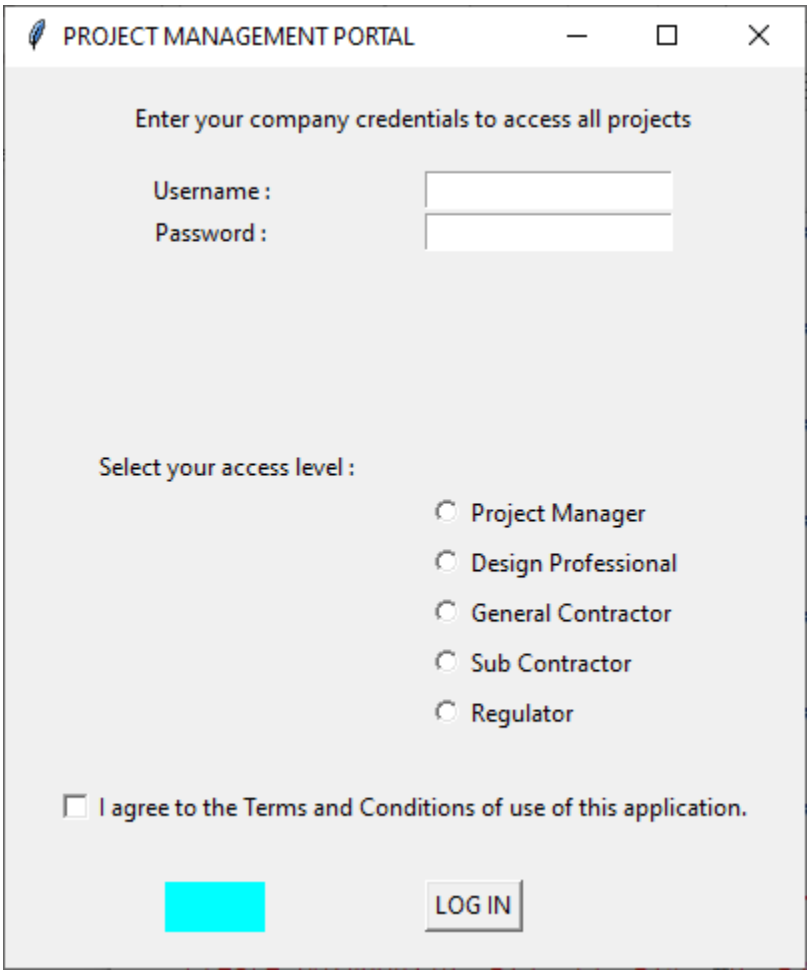
canvas_logo.py - E:\Python\Python Course Materials\Tutorial Files\3.16_Canvas\canvas_logo....

File Edit Format Run Options Window Help

```
radio = tk.Radiobutton(root, text = 'Sub Contractor', value = 4,
                       variable = rb)
radio.grid(row = 8, column = 1, sticky = tk.W)
radio = tk.Radiobutton(root, text = 'Regulator', value = 5,
                       variable = rb)
radio.grid(row = 9, column = 1, sticky = tk.W)

# create a Checkbutton
# the Checkbutton will appear with a caption statement of terms of
# use of the app. the user must check to accept the terms
# before being acess to proceed into the portal
checkl = tk.IntVar()
checkbuttonl = tk.Checkbutton(root, height = 4, text =\
'I agree to the Terms and Conditions of use of this application.',\
onvalue = 1, offvalue = 0, padx = 25, variable = checkl)
checkbuttonl.grid(row = 10, column = 0, columnspan = 2)

# create a canvas widget
canvasl = tk.Canvas(root, width = 50, height = 25, bg = "cyan")
canvasl.grid(row = 11, column = 0)

# create click button the user clicks on after entereing
# login information. the command option will call the
# function which shall check if the correct login
# information has been entered and prompt the user accordingly
button = tk.Button(root, text = 'LOG IN', command = login)
button.grid(row = 11, column = 1, sticky = tk.W)

root.mainloop()
```

Ln: 1 Col: 0

Save the file.
Run the file.



Obviously, this is the simplest company logo you have ever seen. Consider ways you may add features such as polygons, lines, circles etc. etc.

Successful completion!

17. DEBUGGING AND GETTING HELP

17.1 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.

17.2 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.

18. 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, applications and programming structures of the *Python tkinter* widgets for GUI programming.

In this course the following *Python tkinter* widgets were presented in detail: Label, Text, Entry, Button, Checkbutton, Radiobutton, Listbox, Spinbox, Menubutton, Menu, Message, Frame, LabelFrame, and Canvas. 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

