**PROJECT Working with Tkinter 50 points**

**Objective** To modify a GUI program, (**tkContacts.py**) which allows you to add, update, delete and load contacts from a list defined in another py (**contacts.py**) file.

***PROJECT DESCRIPTION***

For this exercise you will modify the existing starter code file included in BB for this lab called **tkContacts.py**. The file if you run it will incorporate a contact list imported into tkContacts.py called **contacts.py** which is a file you also can download from BB.

***Information about This Project***

Tkinter is a powerful package to create GUI applications.

***Steps to Complete This Project***

**STEP 1 Review This Program Description**

Open up the **contacts.py** file to see how the **py** file is set up. You will notice that the file contains a simple list called **contactlist** which is read in your **tkContacts.py** file via the **setList()** function.

Go thru the rather simplistic yet elegant code logic for **tkContacts.py** before you modify things so you become familiar with the widgets introduced namely, Label, Button, ListBox, Scrollbar and Entry widgets. Notice all of the widgets are wrapped into a frame namely **frame1**.

Notice also the various functions included in the **tkContacts.py** file. Functions and their logic are defined as follows:

-The **selection()** function to return the **curselection** value indicating which selection of your ListBox was chosen via its index.

Return value:

**return int(select.curselection()[0])**

Note the value returned here is the index from a tuple.

-The **addContact()** function appends to the variable **contactList**, the values from the Tkinter class variables **nameVar** and **phoneVar**. These variables are special variables used with the two defined Entry widgets (defined in your buildFrame() function) which carry values entered into the Entry widgets when the user enters a name and phone number. The get() method of the class variables is employed to get the text entry for each Entry widget.

The function then ends up with a call to setList() which sorts the contactlist.

-The **updateContact()** function updates the variable contactlist by the selected index of the Listbox with the nameVar and phoneVar values. Notice the call to the selection() function (shown below) which gets the actual selected index from the Listbox by the user’s mouse click.

**contactlist[selection()]**

The function then ends up with a call to setList() which sorts the contactlist.

-The **deleteContact()** function merely deletes the selection from the contactlist, again via the selected index of the Listbox.

The function then ends up with a call to setList() which sorts the contactlist.

-The **loadContact()** function takes the selection from the user, assigns as a tuple the results to (name and phone), the contact values and adds the tuples into the respective nameVar and phoneVar class variables via a special set() method of the class variables.

-The **buildFrame()** function places all the widgets one by one into the frame.

A grid is used to place the widgets in an orderly fashion using parameter values for a given row and column.

Example for the name Entry widget:

**name.grid(row=0, column=1, sticky=W)**

Notice the placement for the widget to be placed in a grid format in row 0 and column 1. The sticky value is assigned a value of W which stands for west meaning the widget should be aligned to the left cell border. Other values may include S,N,E or NW,NE,SW,SE.

Notice also the use of the Scrollbar widget which is added to the Listbox with the value inside the Listbox constructor with the command yscrollcommand=scroll.set, as shown below:

Listbox(frame1, **yscrollcommand=scroll.set**, height=**7**)

Note the height command with a value of **7**. This indicates the allowance of values shown inside the Listbox,visible at start up.

Lastly notice the **pack()** method which ensures the visibility of various widgets placed onto the frame. Parameters include which side the widget may be placed and how the orientation should be filled with the fill command. Sides include top, right, left or bottom. If no side is issued, top is assumed. Example of a fill maybe Y for vertical alignment or H for horizontal alignment of the widget containment or BOTH for both possibilities.

At the tail end of the function, the root object is returned to allow for the root to build its frame.

-Lastly the **setList()** function sorts the contactlist and allows for an update to the Listbox with the sorted values inserted into the ListBox from the newly built select list.

**STEP 2**  **Run and Test the Program**

Run and test the program out well before you modify it by walking thru its features, namely to add, load, update and delete contacts. Run the program and click on one of the names in the list box. Then click on your Load button. The contact name and corresponding phone number of your selection should appear in your Entry widgets. Clicking the Load button (btn4) activates the callback function, loadContact which accesses the index of your listbox selection (via the function selection()) and then accesses the data from the list "contactlist". The code to trigger the callback is defined in the button’s constructor as follows:

btn4 = Button(frame1,text=" Load ",command=**loadContact**)

Now modify the Entry widgets contents and click Update or Add. Play also around with the Delete functionality.

You’ll notice that every rerun of the app shows that the list data contained in contactlist has not changed from the contacts.py file. You will be asked to change that data soon.

**STEP 3**  **Instructions to complete this lab:**

1. Add in a title to the frame. Entitle it as ‘My Contact List’.
2. Add a Save button that will save any updates, deletes and adds, to the contactlist so the next time the program is run, the contact list data will be shown as refreshed. A possibility to save your contactlist, is to have the Save button's callback write a new "contacts" file to import in the next time the program is run. Another possibility is to use the "pickle" module to store the contents of "contactlist" between runs.

\*\*Grads finish up with the following as well:

Add in an exit button that will close out the app at the bottom, center of the GUI. Include

an appropriate callback function to perform this duty. Include a messagebox pop up

telling the user that the program is now exiting and have the option to click Ok or Cancel

within the messagebox. To allow for a module to get a message box working use the

following directive in your program:

**from tkinter import messagebox**

The action to allow the user to Exit or not can be coded as follows:

**if (messagebox.askokcancel(title=app\_title, \**

**message="Are you want to exit, OK or Cancel") == 1) :**

**os.\_exit(1)**

**STEP 4**  **Submit Your Program Code and Your Run Time Output**

Take appropriate snapshots and place them into a Word document as follows for full credit. Label your snapshots accordingly. Include both your .py files as well to BB.

Snapshots must include the following order:

1. Your entire interface at runtime including your save button added on

2. Initial snapshot of listbox at runtime

3. Show a contact being added (add in a contact name by last name, first name)

4. Show a contact deleted

5. Show a contact being updated

6. Show modified contactlist file (contained in contacts.py)

\*\*\*Grads include as snapshots

- Your message box requesting if a user wants to exit the program

- Your GUI with the added exit button