Python SQLite Address Book Tutorial

Contents

Time required: 180 minutes

- Comment each line of code as show in the tutorials and other code examples.
- Follow all directions carefully and accurately.
- Think of the directions as minimum requirements.

SQL Tutorial

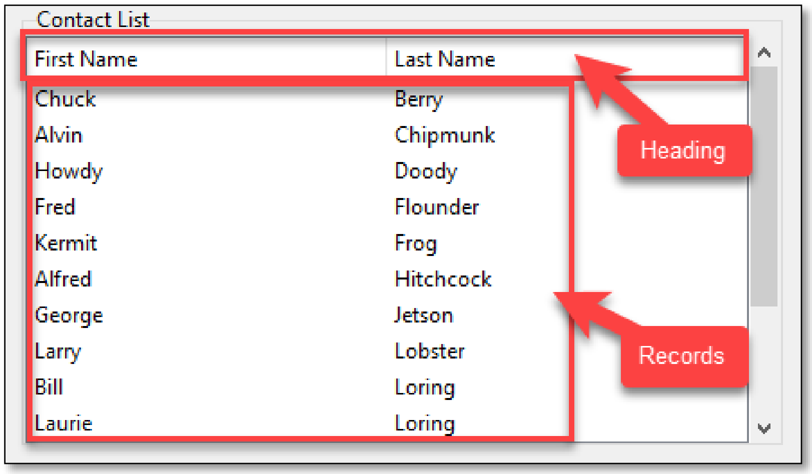
- https://www.w3schools.com/sql/sql_intro.asp
- https://www.w3schools.com/sql/sql_syntax.asp
- https://www.w3schools.com/sql/sql_create_db.asp
- https://www.w3schools.com/sql/sql_create_table.asp

- https://www.w3schools.com/sql/sql_drop_table.asp
- https://www.w3schools.com/sql/sql_insert.asp
- https://www.w3schools.com/sql/sql_update.asp
- https://www.w3schools.com/sql/sql_delete.asp
- https://www.w3schools.com/sql/sql_select.asp

The Treeview Widget

There are many ways to display tabular data in Tkinter. One of the best options is to use a TTK.Treeview widget.

This is an example of a Treeview widget with a Scrollbar widget.



First Name	Last Name
Chuck	Berry
Alvin	Chipmunk
Howdy	Doody
Fred	Flounder
Kermit	Frog
Alfred	Hitchcock
George	Jetson
Larry	Lobster
Bill	Loring
Laurie	Loring

Database

We will use the same database structure from the previous CLI Address Book assignment for this assignment. All the SQL data handling methods and much of the code will stay the same. The interface will be the major change.

These are the fields and data types that are in this tutorial.

id (primary key)	**INTEGER**
first_name	TEXT
last_name	TEXT

phone_number	TEXT
email	TEXT
your_field	TEXT

Tutorial 1: Address Book GUI

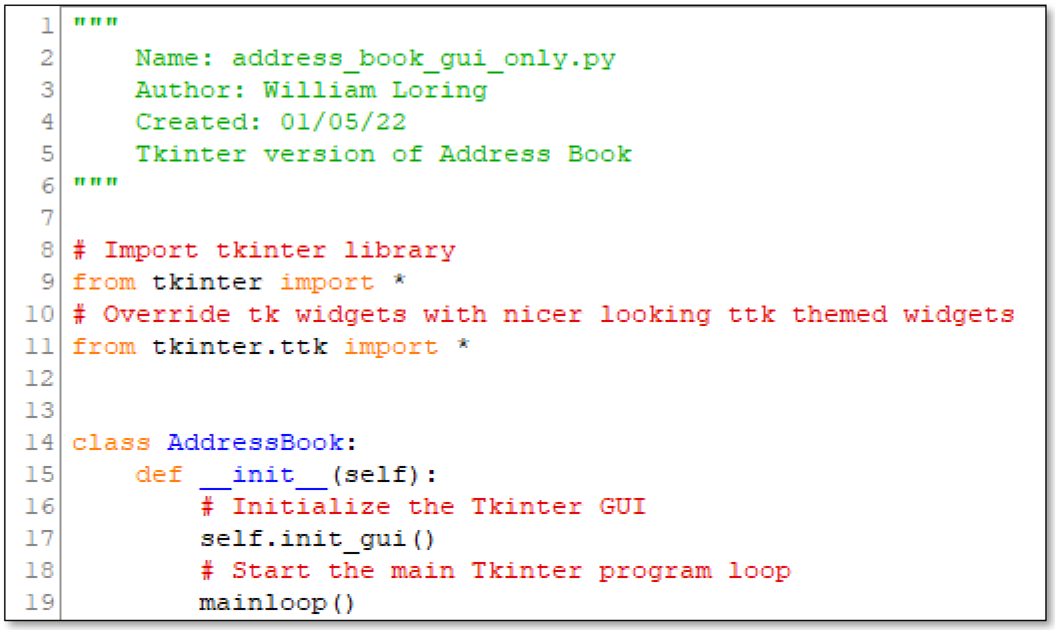
NOTE: Your program does not have to look like the example. It does need to have the same functionality.

NOTE: Add the code needed for the database field you added in Address Book CLI. That field is not shown in this tutorial. The code must match your database fields.

1. Create a Python program named **address_book_gui_only.py**
2. Let's start setting up the Tkinter GUI. Add the following code to your program.

init

Create the **__init__** method.



```python
"""
    Name: address_book_gui_only.py
    Author: William Loring
    Created: 01/05/22
    Tkinter version of Address Book
"""

# Import tkinter library
from tkinter import *
# Override tk widgets with nicer looking ttk themed widgets
from tkinter.ttk import *


class AddressBook:
    def __init__(self):
        # Initialize the Tkinter GUI
        self.init_gui()
        # Start the main Tkinter program loop
        mainloop()
```

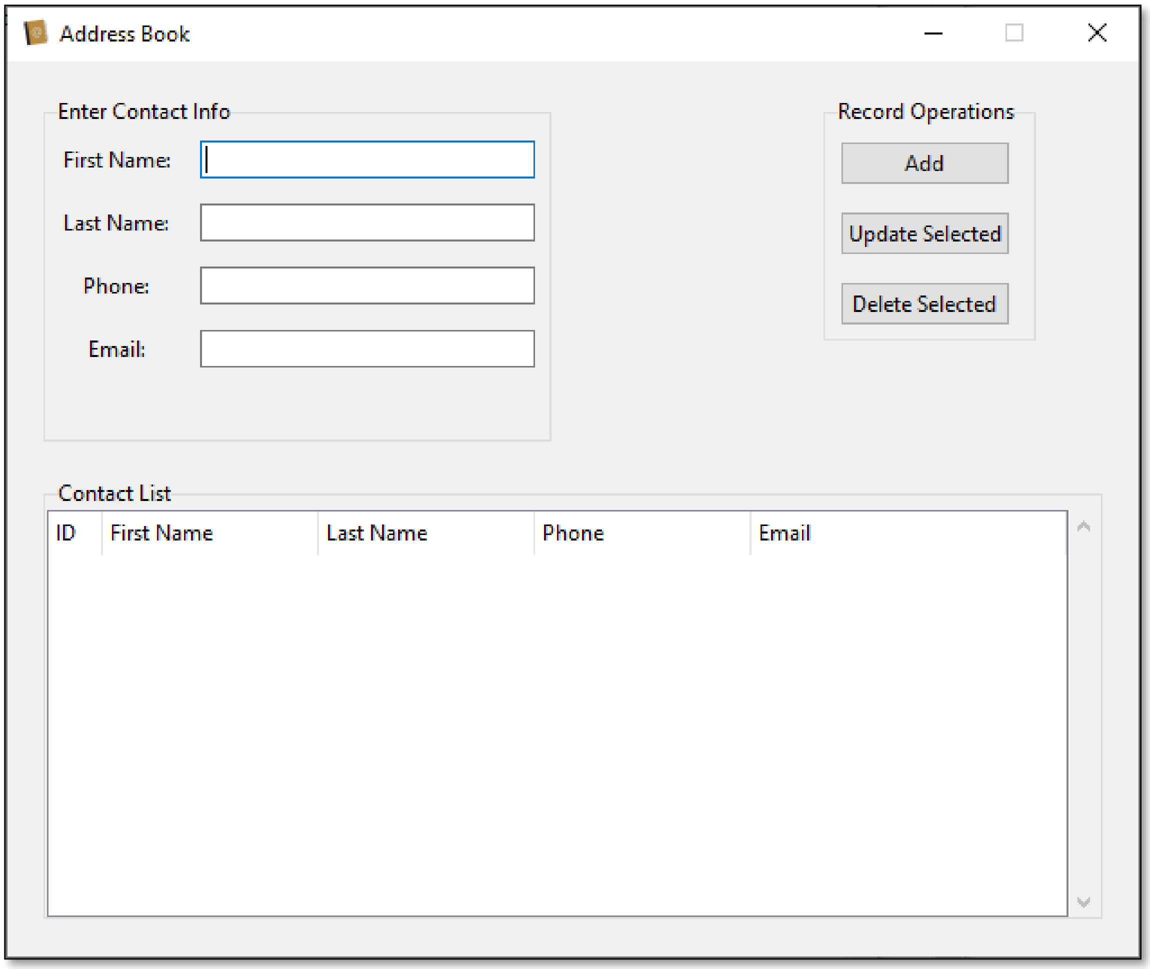
```python
# ------------------- INITIALIZE GUI ----------------------------------------#
    def init_gui(self):
        """Initialize program GUI"""
        self.window = Tk()
        # Set window location on screen 400 pixels right 300 pixels down
        # The window size will change based on the controls
        self.window.geometry("+400+300")
        # Add icon to program title bar
        self.window.iconbitmap("address_book.ico")
        self.window.title("Address Book")
        self.window.resizable(False, False)
        # Create and grid all widgets
        self.create_frames()
        self.create_widgets()
        self.create_treeview()
```

```python
# --------------------------- CREATE FRAMES --------------------------------#
    def create_frames(self):
        self.entry_frame = LabelFrame(
            self.window,
            text="Enter Contact Info",
            relief=GROOVE
        )
        self.operations_frame = LabelFrame(
            self.window,
            text="Record Operations",
            relief=GROOVE
        )
        self.treeview_frame = LabelFrame(
            self.window,
            text="Contact List",
            relief=GROOVE
        )
        # Grid the frames
        self.entry_frame.grid(row=0, column=0, sticky=NW)
        self.operations_frame.grid(row=0, column=1, sticky=N)
        self.treeview_frame.grid(row=1, column=0, columnspan=2, sticky=W)
```

```python
# ------------------------- CREATE WIDGETS --------------------------------#
    def create_widgets(self):
        # ------------------------ CREATE LABELS --------------------------#
        self.lbl_first_name = Label(
            self.entry_frame, text="First Name:", anchor="e")
        self.lbl_last_name = Label(
            self.entry_frame, text="Last Name:", anchor="e")
        self.lbl_phone = Label(self.entry_frame, text="Phone:", anchor="e")
        self.lbl_email = Label(
            self.entry_frame, text="Email:", anchor="e")
        self.lbl_status = Label(self.entry_frame, text=" ", anchor="w")

        # --------------------- CREATE ENTRY BOXES ------------------------#
        self.first_name_entry = Entry(self.entry_frame, width=30)
        # Set focus for data entry
        self.first_name_entry.focus_set()
        self.last_name_entry = Entry(self.entry_frame, width=30)
        self.phone_entry = Entry(self.entry_frame, width=30)
        self.email_entry = Entry(self.entry_frame, width=30)

        # ------------------------ CREATE BUTTONS -------------------------#
        self.btn_add = Button(
            self.operations_frame,
            text="Add",
        )
        self.btn_modify = Button(
            self.operations_frame,
            text="Update Selected"
        )
        self.btn_delete = Button(
            self.operations_frame,
            text="Delete Selected",
        )
        # -------------------------- GRID WIDGETS --------------------------#
        self.lbl_first_name.grid(row=0, column=0)
        self.lbl_last_name.grid(row=1, column=0)
        self.lbl_phone.grid(row=2, column=0)
        self.lbl_email.grid(row=3, column=0)
        self.lbl_status.grid(row=4, column=0, columnspan=2)

        self.first_name_entry.grid(row=0, column=1)
        self.last_name_entry.grid(row=1, column=1)
        self.phone_entry.grid(row=2, column=1)
        self.email_entry.grid(row=3, column=1)

        self.btn_add.grid(row=0, column=0, sticky=EW)
        self.btn_modify.grid(row=1, column=0, sticky=EW)
        self.btn_delete.grid(row=2, column=0, sticky=EW)

        # Set padding between frame and window
        self.entry_frame.grid_configure(padx=20, pady=(20))
        self.operations_frame.grid_configure(padx=20, pady=(20))
        # Even out the padding between frames, leave out y distance on top
        self.treeview_frame.grid_configure(padx=20, pady=(0, 20))

        # Set padding for all widgets inside the frame
        for widget in self.entry_frame.winfo_children():
            widget.grid_configure(padx=7, pady=7)
        for widget in self.treeview_frame.winfo_children():
            widget.grid_configure(padx=7, pady=7)
        for widget in self.operations_frame.winfo_children():
            widget.grid_configure(padx=7, pady=7)
```

```python
# --------------------------- TREEVIEW AND SCROLLBAR ------------------------#
    def create_treeview(self):
        """Setup tree view for record display"""
        # Create treeview
        self.tree = Treeview(
            self.treeview_frame,
            height=10,
            columns=("id", "first_name", "last_name", "phone", "email"),
            style="Treeview",
            show="headings",
            selectmode="browse"
        )
        # Setup the columns
        self.tree.column("id", width=30)
        self.tree.column("first_name", width=120)
        self.tree.column("last_name", width=120)
        self.tree.column("phone", width=120)
        self.tree.column("email", width=175)

        # Setup the heading text visible at the top of the column
        self.tree.heading("id", text="ID", anchor=W)
        self.tree.heading("first_name", text="First Name", anchor=W)
        self.tree.heading("last_name", text="Last Name", anchor=W)
        self.tree.heading("phone", text="Phone", anchor=W)
        self.tree.heading("email", text="Email", anchor=W)

        # Grid the tree
        self.tree.grid(row=0, column=0)

        # Create scrollbar for treeview
        self.scrollbar = Scrollbar(
            self.treeview_frame,
            orient="vertical",
            command=self.tree.yview
        )

        # Set scroll bar to scroll vertically and attach to the tree
        self.tree.configure(yscroll=self.scrollbar.set)

        # Grid scrollbar just to the right of the tree
        # sn (SouthNorth) expands scrollbar to height of tree
        self.scrollbar.grid(row=0, column=1, sticky="sn")


# ------------------ START PROGRAM -----------------------#
address_book = AddressBook()
```

Example run:


Address Book
Enter Contact Info
First Name:
Last Name:
Phone:
Email:
Record Operations
Add
Update Selected
Delete Selected
Contact List
ID
First Name
Last Name
Phone
Email

Tutorial 2: Database

NOTE: Add the code needed for the database field you added in Address Book CLI. That field is not shown in this tutorial. The code must match your database fields.

1. Copy the **db_operations.py** from the Address Book program.
2. Make a copy of your **address_book_gui_only.py** program.
3. Name it **address_book_gui.py**

We will make a couple of minor changes to the **__init__** method to connect to our database. The rest of the code remains the same, we will only add methods to connect to the db_operations.py file.

```python
"""
    Name: address_book_gui.py
    Author: William Loring
    Created: 01/05/22
    Tkinter version of MVC (Model View Controller) Address Book
"""

# Import tkinter library
from tkinter import *
# Override tk widgets with nicer looking ttk themed widgets
from tkinter.ttk import *
# Database operations library
import db_operations


class AddressBook:
    def __init__(self):
        # Create the database controller object
        # If the database doesn't exist, it is created
        self.db_op = db_operations.DBOperations("address_book.db")
        # The controller creates the table if it doesn't exist
        self.db_op.create_table()
        # Initialize Tkinter GUI
        self.init_gui()
        # List the existing records to show on startup
        self.fetch_all_records()
        # Start the main Tkinter program loop
        mainloop()
```

Insert Record

NOTE: Add the code needed for the database field you added in Address Book CLI. That field is not shown in this tutorial.

We are going to add methods to our GUI to work with the db_operations.py file.

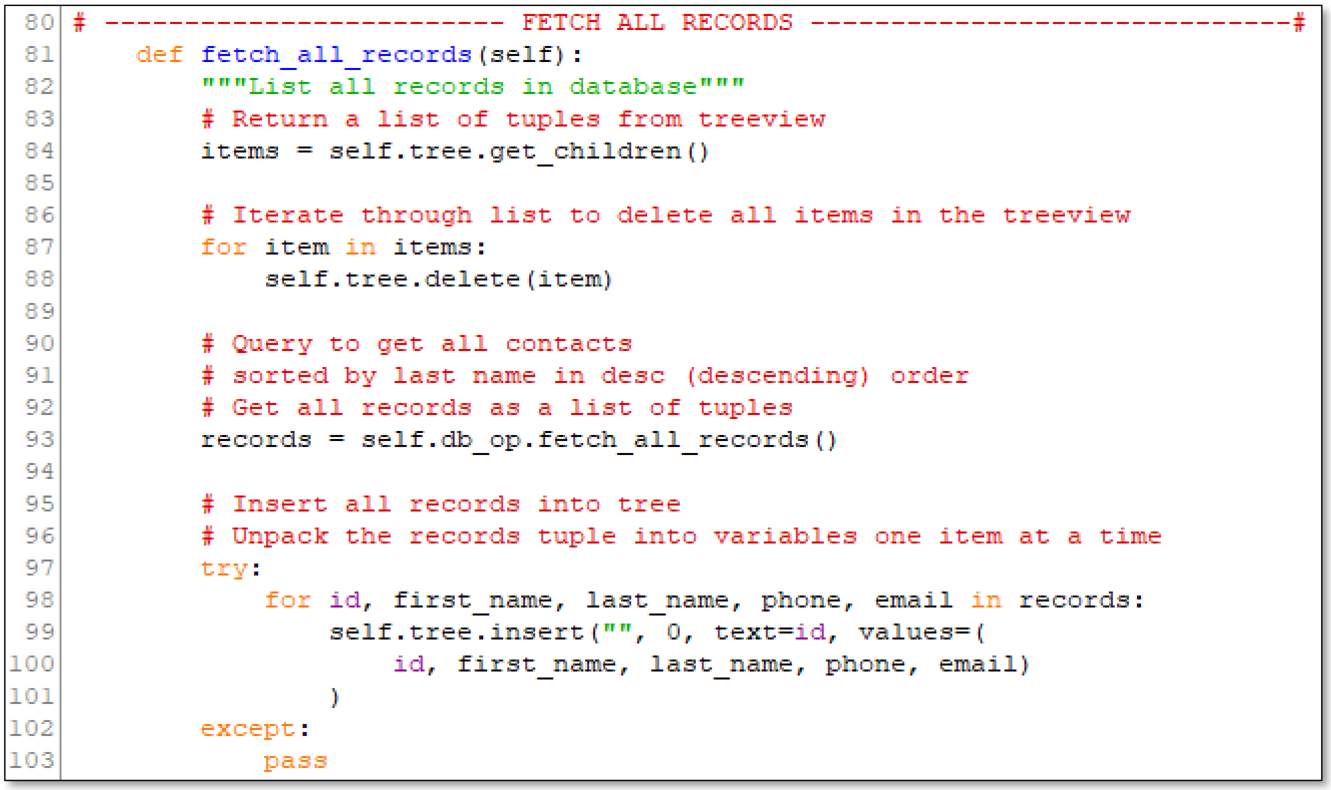
Some of the code is the same as the CLI version. To save time with each of these methods, you can copy and paste the CLI methods and modify them.

The Insert Record method is modified for the GUI interface.

```python
# ---------------------------- INSERT RECORD ---------------------------------#
    def insert_record(self):
        """Add new record to database"""
        # Clear status label
        self.lbl_status.configure(text="")
        # Get input from user
        first_name = self.first_name_entry.get()
        last_name = self.last_name_entry.get()
        phone = self.phone_entry.get()
        email = self.email_entry.get()

        # Ensure the user enters a complete record
        if first_name == "" or last_name == "":
            self.lbl_status.configure(text="Please fill out all entries")
        else:
            # Insert record into database
            self.db_op.insert_record(first_name, last_name, phone, email)
            # Let the user know the add record was successful
            self.lbl_status.configure(
                text=f"{first_name} {last_name} was successfully added."
            )

        # Clear the entry widgets
        self.first_name_entry.delete(0, END)
        self.last_name_entry.delete(0, END)
        self.phone_entry.delete(0, END)
        self.email_entry.delete(0, END)

        # Display all records in treeview
        self.fetch_all_records()

        # Set focus to entry widget for next entry
        self.first_name_entry.focus()
```

List All Records

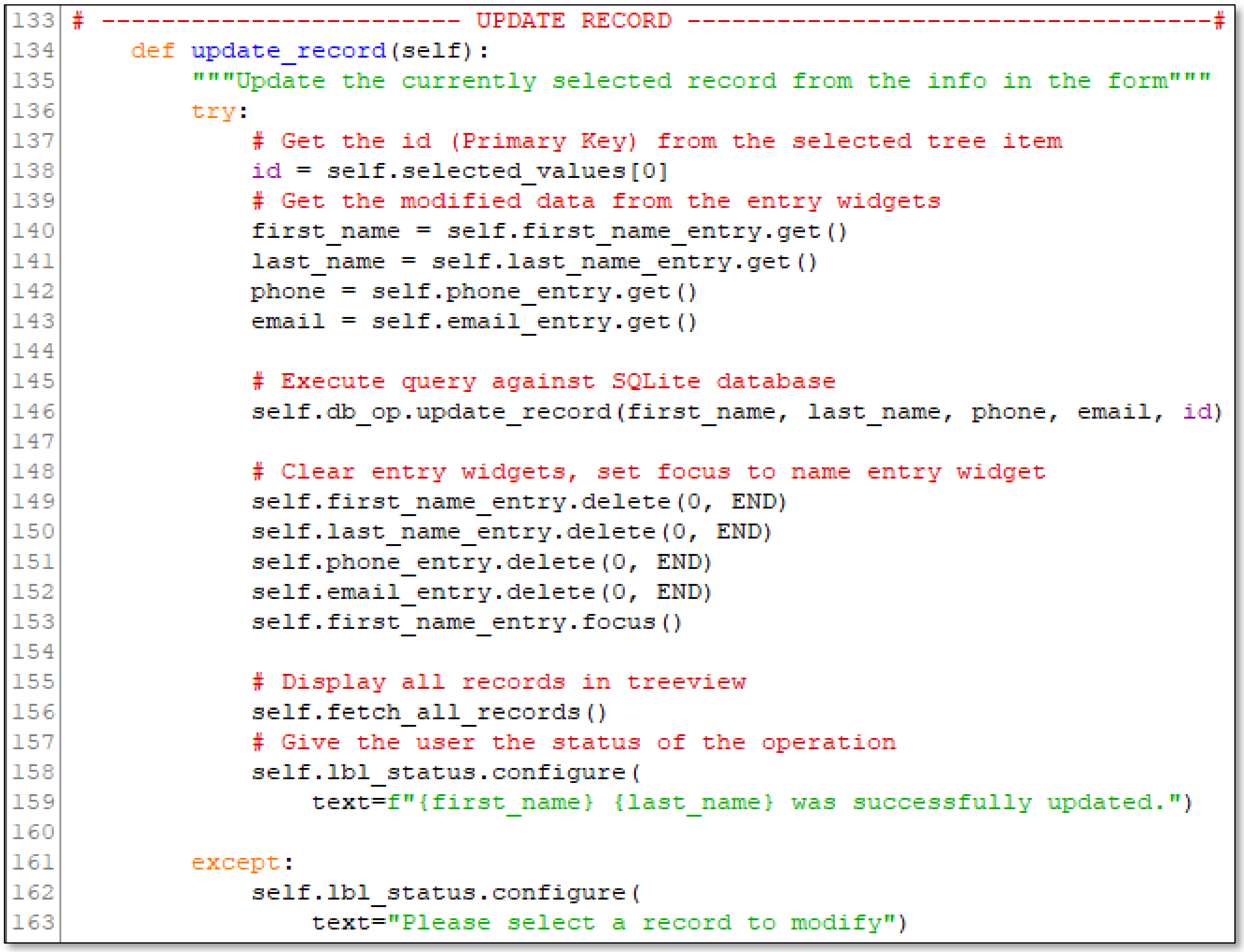
The List All Records method is new and quite a bit different. It uses a **Treeview** and **Scrollbar** widget to display the data.



```python
# ------------------------- FETCH ALL RECORDS ------------------------------#
    def fetch_all_records(self):
        """List all records in database"""
        # Return a list of tuples from treeview
        items = self.tree.get_children()

        # Iterate through list to delete all items in the treeview
        for item in items:
            self.tree.delete(item)

        # Query to get all contacts
        # sorted by last name in desc (descending) order
        # Get all records as a list of tuples
        records = self.db_op.fetch_all_records()

        # Insert all records into tree
        # Unpack the records tuple into variables one item at a time
        try:
            for id, first_name, last_name, phone, email in records:
                self.tree.insert("", 0, text=id, values=(
                    id, first_name, last_name, phone, email)
                )
        except:
            pass
```

Update Record

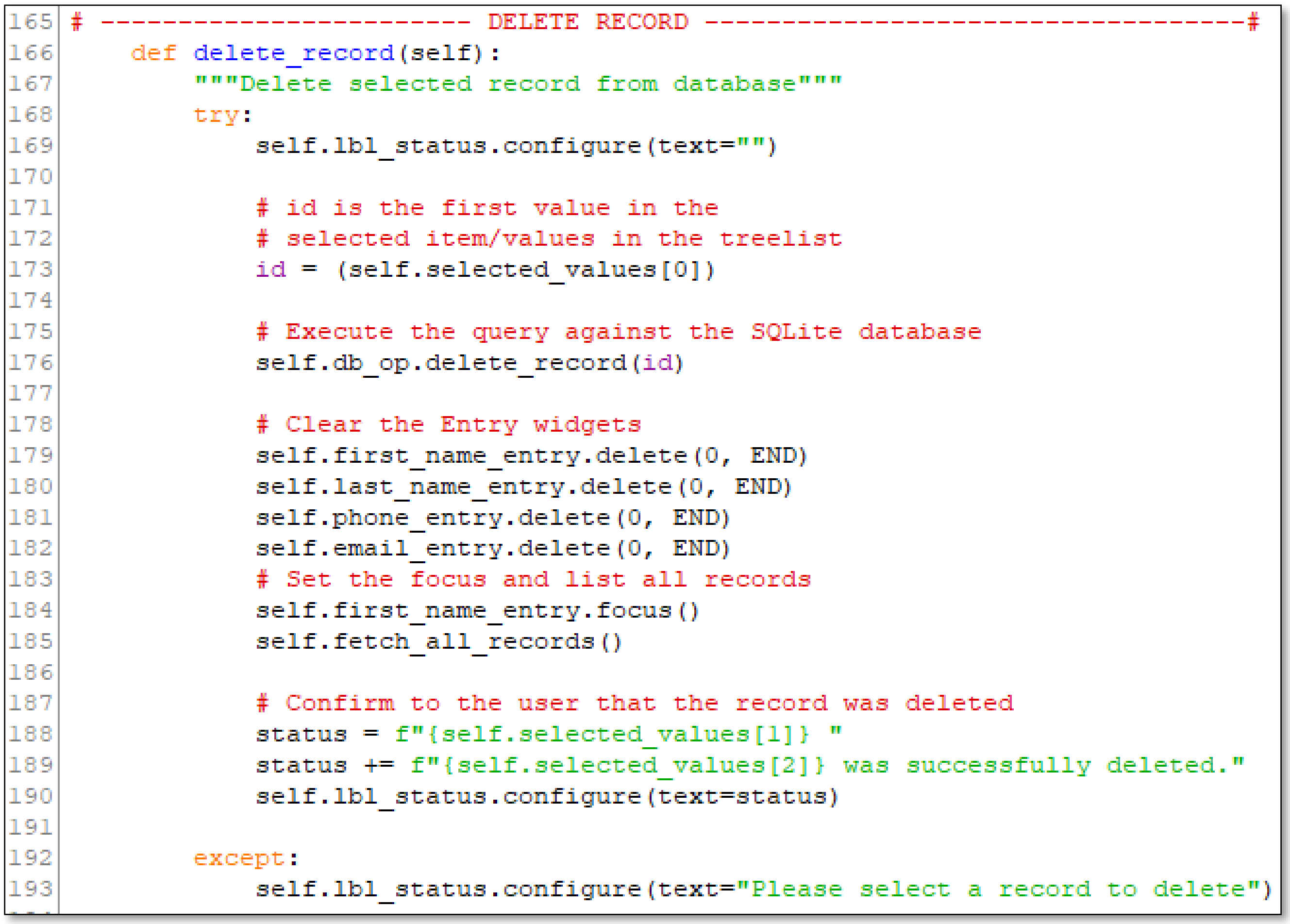
We don’t have to type in the primary key with the Treeview to select a record. We select a record in the Treeview. This populates the entry widgets. Edit the record. Save the entry widgets data to the database.



```python
# ------------------------- UPDATE RECORD -----------------------------------#
    def update_record(self):
        """Update the currently selected record from the info in the form"""
        try:
            # Get the id (Primary Key) from the selected tree item
            id = self.selected_values[0]
            # Get the modified data from the entry widgets
            first_name = self.first_name_entry.get()
            last_name = self.last_name_entry.get()
            phone = self.phone_entry.get()
            email = self.email_entry.get()

            # Execute query against SQLite database
            self.db_op.update_record(first_name, last_name, phone, email, id)

            # Clear entry widgets, set focus to name entry widget
            self.first_name_entry.delete(0, END)
            self.last_name_entry.delete(0, END)
            self.phone_entry.delete(0, END)
            self.email_entry.delete(0, END)
            self.first_name_entry.focus()

            # Display all records in treeview
            self.fetch_all_records()
            # Give the user the status of the operation
            self.lbl_status.configure(
                text=f"{first_name} {last_name} was successfully updated.")

        except:
            self.lbl_status.configure(
                text="Please select a record to modify")
```

Delete Record

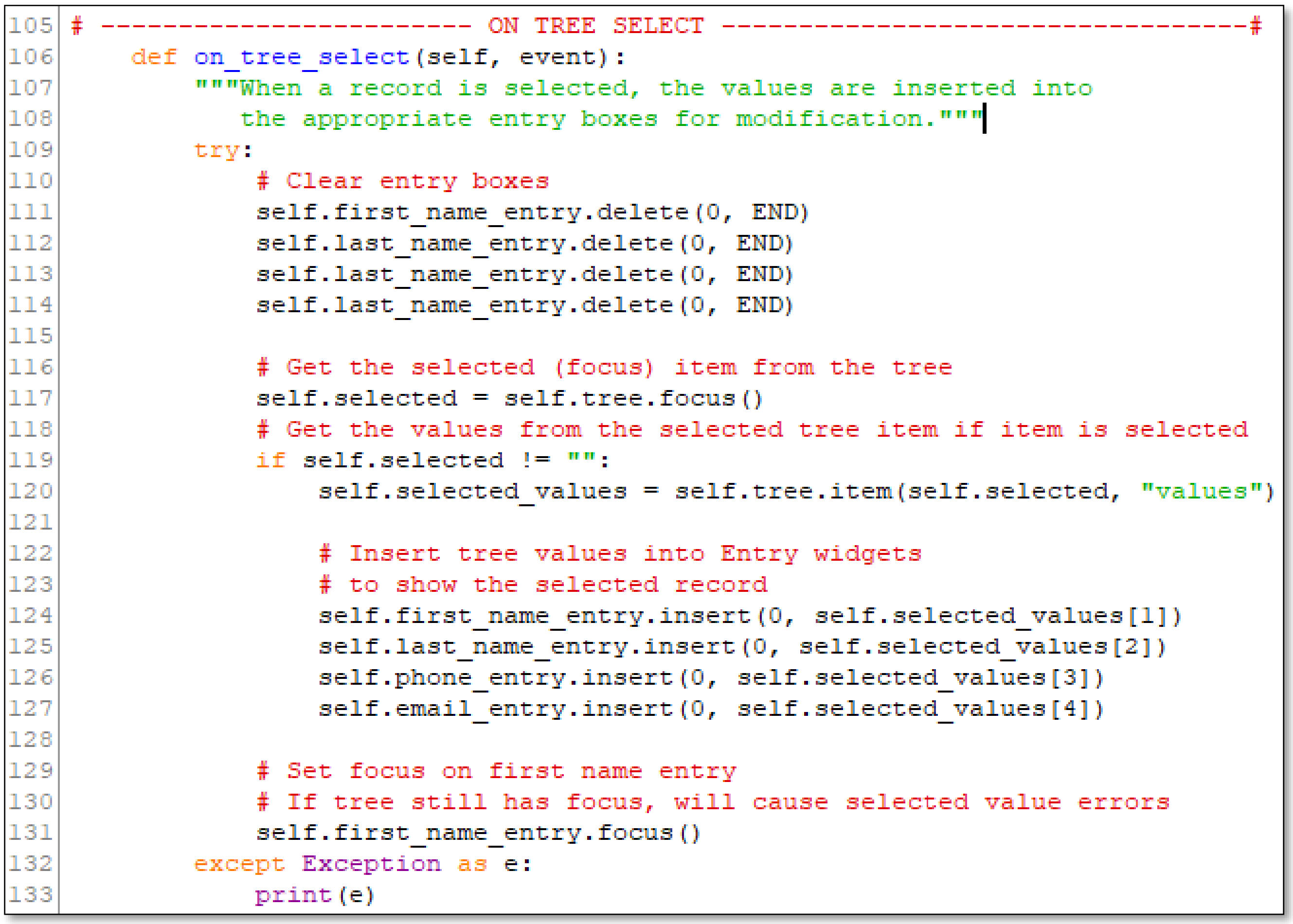
We don’t need to know the primary key to delete a record. We retrieve the primary key when we select the record from the Treeview.



```
# ------------------------- DELETE RECORD ------------------------------------#
    def delete_record(self):
        """Delete selected record from database"""
        try:
            self.lbl_status.configure(text="")

            # id is the first value in the
            # selected item/values in the treelist
            id = (self.selected_values[0])

            # Execute the query against the SQLite database
            self.db_op.delete_record(id)

            # Clear the Entry widgets
            self.first_name_entry.delete(0, END)
            self.last_name_entry.delete(0, END)
            self.phone_entry.delete(0, END)
            self.email_entry.delete(0, END)
            # Set the focus and list all records
            self.first_name_entry.focus()
            self.fetch_all_records()

            # Confirm to the user that the record was deleted
            status = f"{self.selected_values[1]} "
            status += f"{self.selected_values[2]} was successfully deleted."
            self.lbl_status.configure(text=status)

        except:
            self.lbl_status.configure(text="Please select a record to delete")
```

Select Record with on_tree_select Method

This is the method that works its magic to select and display a record in the form. To make it more convenient for our user, we create an **on_tree_select** method to automatically fill the Entry widgets with the record being selected in the Treeview. This makes it easy to update or delete the record.



```python
# ------------------------ ON TREE SELECT ----------------------------------#
    def on_tree_select(self, event):
        """When a record is selected, the values are inserted into
           the appropriate entry boxes for modification."""
        try:
            # Clear entry boxes
            self.first_name_entry.delete(0, END)
            self.last_name_entry.delete(0, END)
            self.last_name_entry.delete(0, END)
            self.last_name_entry.delete(0, END)

            # Get the selected (focus) item from the tree
            self.selected = self.tree.focus()
            # Get the values from the selected tree item if item is selected
            if self.selected != "":
                self.selected_values = self.tree.item(self.selected, "values")

                # Insert tree values into Entry widgets
                # to show the selected record
                self.first_name_entry.insert(0, self.selected_values[1])
                self.last_name_entry.insert(0, self.selected_values[2])
                self.phone_entry.insert(0, self.selected_values[3])
                self.email_entry.insert(0, self.selected_values[4])

            # Set focus on first name entry
            # If tree still has focus, will cause selected value errors
            self.first_name_entry.focus()
        except Exception as e:
            print(e)
```

Finish Up

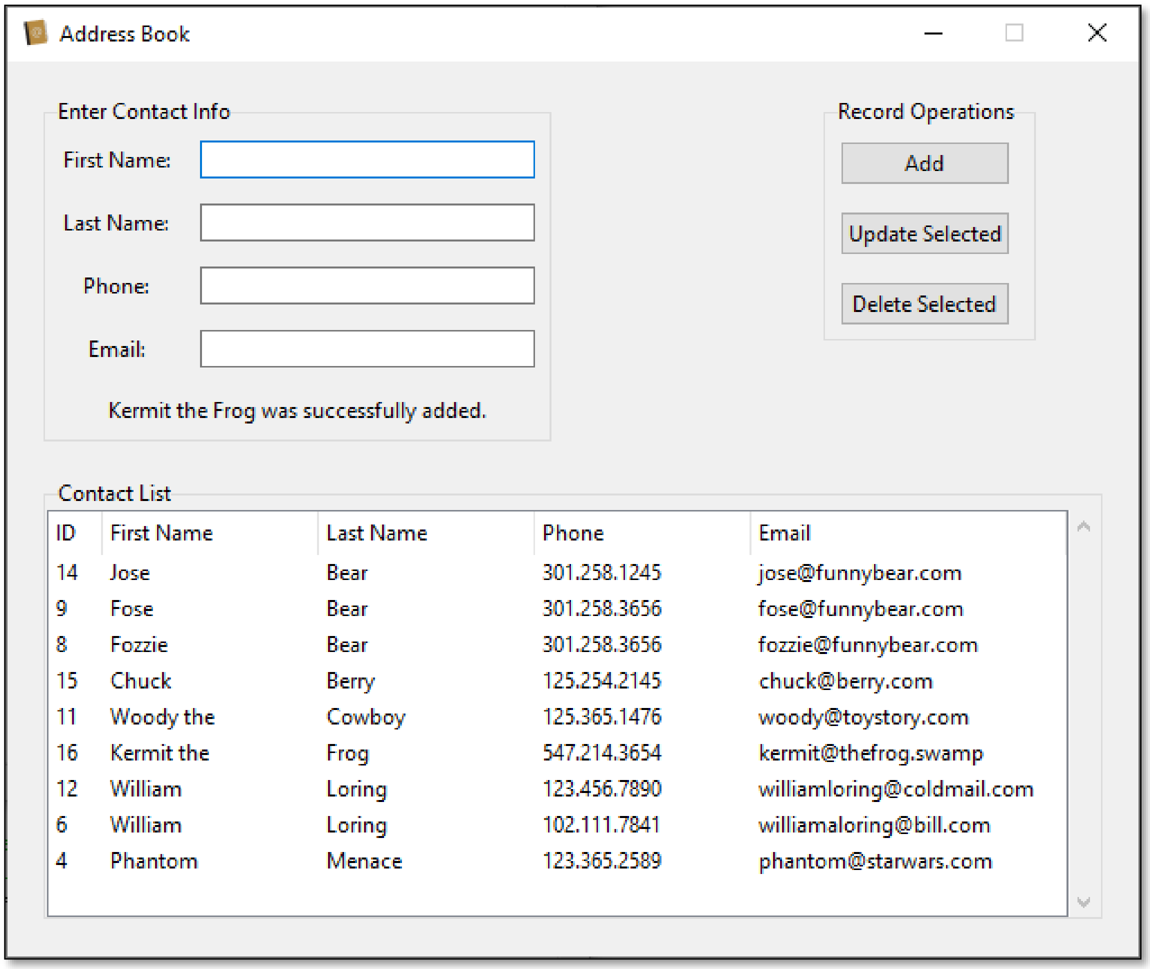
We have a couple of modifications to tie in our methods. This adds our method calls to our buttons.

```
# ------------------------- CREATE BUTTONS ---------------------------#
self.btn_add = Button(
    self.operations_frame,
    text="Add",
    command=self.insert_record
)
self.btn_modify = Button(
    self.operations_frame,
    text="Update Selected",
    command=self.update_record
)
self.btn_delete = Button(
    self.operations_frame,
    text="Delete Selected",
    command=self.delete_record
)
```

The following line goes at the bottom of the **create_treeview** method.

```
# Enable filling from the treeview selection to the entry boxes
self.tree.bind("<<TreeviewSelect>>", self.on_tree_select)
```

Example run:



Kermit the Frog was successfully added.

ID	First Name	Last Name	Phone	Email
14	Jose	Bear	301.258.1245	jose@funnybear.com
9	Fose	Bear	301.258.3656	fose@funnybear.com
8	Fozzie	Bear	301.258.3656	fozzie@funnybear.com
15	Chuck	Berry	125.254.2145	chuck@berry.com
11	Woody the	Cowboy	125.365.1476	woody@toystory.com
16	Kermit the	Frog	547.214.3654	kermit@thefrog.swamp
12	William	Loring	123.456.7890	williamloring@coldmail.com
6	William	Loring	102.111.7841	williamaloring@bill.com
4	Phantom	Menace	123.365.2589	phantom@starwars.com

Assignment: Add Field

If you haven't done this already, modify this program to include the field you added to your Address Book CLI.

Assignment Submission

1. Attach the program files.
2. Attach screenshots showing the successful operation of the program.
3. Submit in Blackboard.