

# COMP 120 - Problem Solving Assignment 2

**Due: Thursday, September 24 @ 10:00PM**

## Required Background:

- “Introduction to Python Programming and Data Structures” - Chapter 10, Basic GUI programming using Tkinter.

## Assignment Overview:

For this assignment you will write a GUI program based, but not identical to, programming exercise 10.18,

**Be sure to read this entire document before beginning to work on the assignment.**

This assignment is worth a total of 100 points.

## Initial Setup

Both you and your partner will need to get the starter code for your group using Git.

1. You will need your group number in what follows. Get this from the “PSA Group Numbers” file under the “PSAs” tab on Blackboard.,
2. In VS Code, open the command palette and select the “Git Clone” option.
3. When prompted for the repository URL, enter the following, with X replaced by your group number (e.g. 7 or 12).

```
ssh://git@code.sandiego.edu/comp120-sp20-psa2-groupX
```

4. Choose the “Open Repository” option in the window that pops up in the lower-right corner of the screen. The repository should contain the following files:
  - `addressbook.py`: You will put all of the code you write for the problem in this file.
  - `addresses.txt`: A starter data file containing a couple addresses for the problem.
5. Each programming team will have its own repository that has been initialized with the same starter code, so when you sync your code, you are sharing with your partners, but with no one else in the class. (I can see your repository also.)

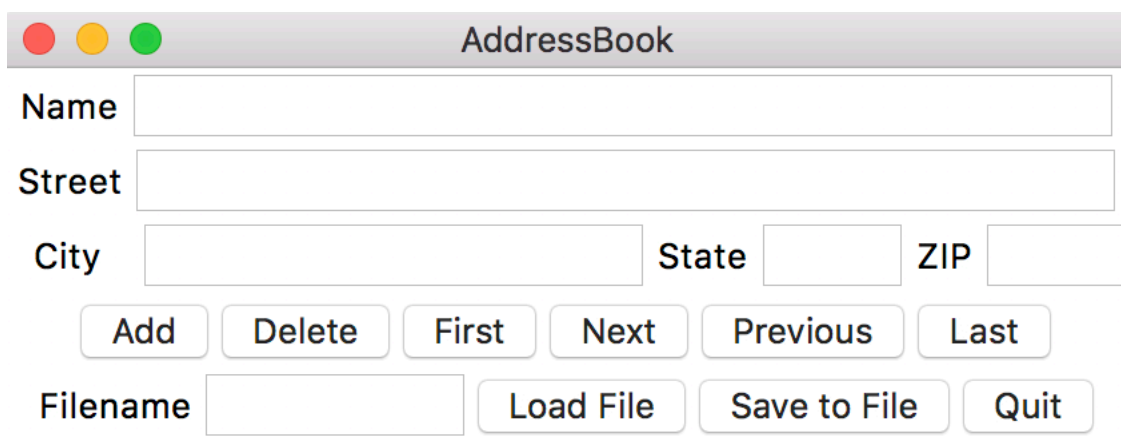
Remember that if you close VS Code, then when you reopen it, you should see your repository. But if you don't see it, then just select File->Open..., and then select the directory containing your repository.

We also recommend that you stage changes, commit those changes, and sync the changes periodically as you are working on the program, and certainly when you are done with a session with your programming partner. This ensures that you won't lose any of your work in case your computer gets lost or a file gets accidentally deleted.

## The Problem

As said above, the problem is based on, but not identical to exercise 10.18 in your textbook. You started working on exercise 10.18 in lab last week, so the first thing you do is copy the code you wrote there into `addressbook.py` to get you started. You will have to edit it, because this problem is different.

The screen your program displays should look like:



The screenshot shows a window titled "AddressBook" with a standard macOS-style title bar (red, yellow, green buttons). The window contains several text input fields and buttons. The input fields are labeled "Name", "Street", "City", "State", and "ZIP". Below these fields are buttons labeled "Add", "Delete", "First", "Next", "Previous", and "Last". At the bottom of the window, there is a "Filename" input field and buttons labeled "Load File", "Save to File", and "Quit".

Your program should maintain an address list that the user of the program can edit. The user can add a new address by typing in the name, street, city, state, and zip of a person into the entry fields at the top of the window, and then hitting the "Add" button. The user can move around the address list using the "First", "Next", "Previous", and "Last" buttons. An address can be deleted using the "Delete" button. A file containing an address list can be loaded using the filename entry field and the "Load File" button. An address list can be saved to a file using the filename entry field and the "Save to File" button. And finally, the "Quit" button quits the program.

Here are the details. Your program must adhere to the following requirements:

1. The program should maintain the address list as a Python list. (The choice of a list becomes the natural choice after seeing the behavior of the buttons in the window.)
2. Each entry in the address list should be an object of type `Address`. So you need to define a class called `Address` that encapsulates (with instance variables) the name, street, city, state, and zip of a single address. The `Address` class should be defined in the same file as the `AddressBook` class.
3. The "First" button displays the first address in the address list. The address is displayed in the entry name, street, city, state, and zip fields. So the entry fields

double as display fields. If there is no first address (the address list is empty), the button should not do anything.

4. The “Next” button displays the next address in the address list (the next address after the one currently being displayed). This suggests that the program should keep track of the index of the address currently being displayed. If there is no next address (the last address is already being displayed, or the list is empty), the button should not do anything.
5. The “Previous” button should display the previous address in the address list. If there is no previous address, the button should not do anything.
6. The “Last” button should display the last address in the address list. If there is no last address (the list is empty), the button should do nothing.
7. If an address is displayed in the entry fields, the user can edit it in any way they want, and then add that address to the address book by hitting the “Add” button. The address should be added directly after the current address that is being displayed. If there is no current address (because the list is empty), then the added address should go at the beginning of the list. After an add, the newly added address should become the current address being displayed.
8. If the user hits the “Delete” button, the currently displayed address should be deleted from the address list. If there is no currently displayed address (because the list is empty), the button should not do anything. After an address is deleted, the currently displayed address should be the address after the deleted address, and if there is not next address, the previous address, and if there is also no previous address (because the list is now empty), nothing should be displayed.
9. The user can load an address list from a file by entering a filename in the “Filename” entry field, and hitting the “Load File” button. If the filename entry field is empty, or if the file cannot be opened, then the button should not do anything. If the file can be opened, the current address list should be deleted, and replaced by the contents of the file. The format of the file is illustrated by the sample file in the repository. (5 lines for each address – one line for each field of the address.) You can assume that an address file has the correct format, so you do not have to do any exception handling while reading the file. After reading in a file, the current address should be the first address that was read in from the file.
10. The user can store an address list to a file by entering a filename in the “Filename” entry field, and hitting the “Save to File” button. If the filename entry field is empty, or if the file cannot be opened, then the button should not do anything. If the file can be opened, the current address list should be written to the file. The format of the file is illustrated by the sample file in the repository. (5 lines for each address – one line for each field of the address.)

11. All of your code must be written in the AddressBook class, and the Address class. (The Address class should be very short – it is just a holding place for the 5 fields of an address.)
12. Use descriptive variable names in your program, and use all lowercase letters with underscores separating words. You should appropriately comment your program. It should have a header (this is started for you - be sure to add your names, the date you started it, and a description); each function should have docstring comments.
13. You should also comment blocks of code within your functions, explaining what the code is doing. How much commenting to add is a judgement call - you don't want too much, or too little. If you have a block of code (say up to 10 lines long), put a brief comment before it saying what is about to happen. Then put blank lines between the blocks of code. Don't comment individual lines of code, unless they are doing something that the reader might not see right away.

Any questions about how the program should behave should be posted to campuswire (category psa2).

## Testing your program

There is not an automated test program for this assignment, so you should test them on your own. Go down the numbered requirements listed above for each problem, and insure that your program satisfies the requirement. Points will be deducted for each requirement that is not satisfied.

## Pair programming requirement

As described in the syllabus, you should write your program using pair programming. Recall that in pair programming, you and your partner work together at one computer, with one of you typing code (the driver), and the other managing (the navigator). To encourage this from you and your partner, when you are the driver for your team, you should be working on your own computer. When you switch roles, the driver should sync her code to the repository, and her partner should then sync onto his computer, and then become the driver. Remember that you should be switching roles every half hour or so.

So when you follow this approach in writing your program, I should see syncs from both of you, with significant differences between the code synced. If I don't see syncs from both of you, there will be a 10 point penalty on your final grade for the assignment.

## Submission Instructions

**Important:** To be safe, you should run your final code on both you and your partner's computers. This will ensure that you are not relying on any special setup on your own computer for the code to work.

To submit your code, you will need to synchronize it using Git. To make sure your changes are saved and synchronized, follow these steps.

1. Open the “Source Control” menu, either by clicking on the 3rd icon on the left (right under the magnifying glass) *or* by going to “View” and “SCM”.
2. Your `addressbook.py` file should show up under the “Changes” section. Click on the “+” icon to the right of the name(s) of the file(s) that you changed to “stage” the changes. This should move the file to a section named “Staged Changes.”
3. Commit your changes by typing in a descriptive message (e.g. “finished problem”) into the “Message” textbox (above the Staged Changes area). After entering the message, click the checkmark icon above the message box to perform the commit. This should remove the changed files from the Staged Changes section.
4. Then, Sync your commit(s) to the server by clicking the “...” (to the right of the checkmark from the last step) and select the “Sync” option. This will likely result in a pop-up notifying you that the sync will do a push and a pull. Select “OK” (or “OK and don’t ask again”) to complete the sync.

If you run into problems, make sure you are properly connected to USD’s VPN (or eduroam wifi if you are on campus), and try the Sync again. If you are still running into problems, check campuswire and ask a new question there if the answer doesn’t already exist.

To make sure that your changes were synced correctly, have your partner do the final step above (namely “...” and then “Sync”). This should fetch the changes you made. You can then test on their computer to make sure it works exactly the same as on your computer. If your partner has trouble accessing and/or running the file, it is likely that the grader will also have problems and your grade will be negatively impacted.

5. When you have finished the problem, go to campuswire, and post to the `psa2_submit` category a message saying that you have submitted `psa2`.

## Grading

When I grade the problem, I will go down the requirements listed above, and up to 10 points will be deducted for each item where you have not met the requirements.

## Late Penalties

1. If you commit by the due date, no penalty. Else,
2. if you commit by 10PM on Tuesday, September 29, 10 points late penalty. Else,
3. if you sync by 10PM on Thursday, October 1, 20 points total late penalty. Else,
4. no points.

## Academic Integrity

Please review the portion of the syllabus that talks about academic integrity with regard to writing programs. In summary, do not share or show your code to any other team in class, and do not turn in any code that you did not write yourself. Don’t look at the code from

another team, or from any other source. The point of these programs is for you to develop your coding skills.