ECE 364 Prelab Assignment 11
GUI Programming in Python

Passing this lab will satisfy course objectives CO2, CO3, CO4 and CO6

Instructions

- You must meet all *base requirements* in the syllabus to receive any credit.
- Work in your Prelab11 directory, and copy all files from ~ee364/DataFolder/Prelab11 into your working directory:

  ```
  cp -r ~ee364/DataFolder/Prelab11/* ./
  ```

- Remember to add and commit all **required** files to SVN. **We will grade the version of the file that is in SVN!**
- Do *not* add any file that is *not* required. **You will lose points if your repository contains more files than required!**
- Make sure you file compiles. **You will not receive any credit if your file does not compile.**
- Name and spell the file, and the functions, exactly as instructed. Your scripts will be graded by an automated process. **You will lose some points, per our discretion, for any function that does not match the expected name.**
- Make sure your output from all functions match the given examples. **You will not receive points for a question whose output mismatches the expected result.**
- Unless otherwise specified, you cannot use any external library, but you can use any module in the **Python Standard Library** to solve this lab, i.e. anything under:

  ```
  https://docs.python.org/3.7/library/index.html
  ```

- Make sure you are using Python 3.7 for your Prelab.

Creating the GUI

Required File `Consumer.py`

Description

In this lab, you will create a simple data entry application. The user will be able to modify the data, but you do not need to perform data validation in this lab, i.e. you can assume, when data is present, it is valid. Refer to the PyQt5 documentation for more information on the widgets and their signals.

You are given the UI file `BasicUI.ui`, that looks like Figure 1, along with its Python equivalent `BasicUI.py`. Additionally, you are also given a boilerplate file called `Consumer.py`, that uses the Python GUI file. Your task is to populate this application file with code that fulfills the requirements given below.

The only code file you need to submit is `Consumer.py`.

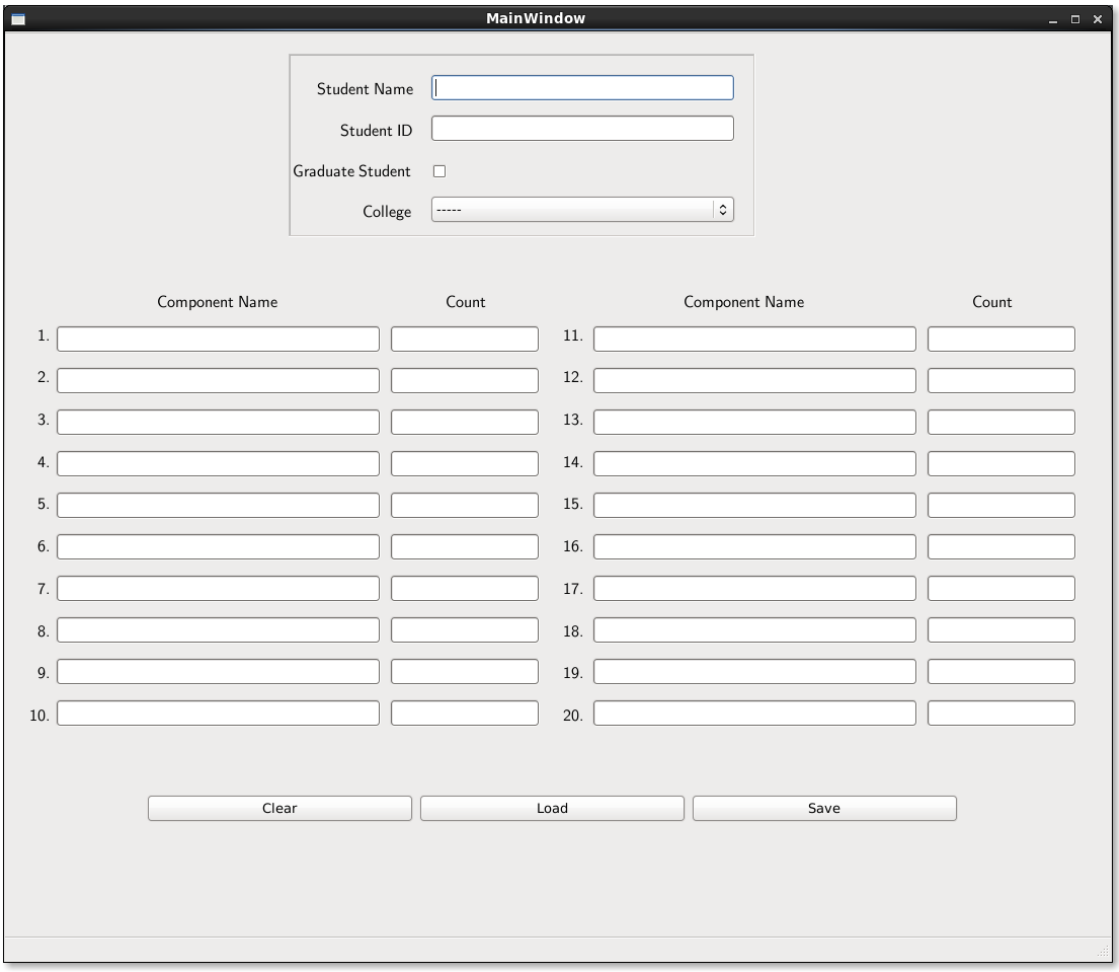

Figure 1: The GUI file with all the widgets shown.

Application Behavior

The application should start with all the text boxes being empty, the check box unchecked, the combo box reset, and the `'Save'` button disabled, as in Figure 2. This is called the 'Initial State' of the GUI, and will help drive the rest of application behavior.

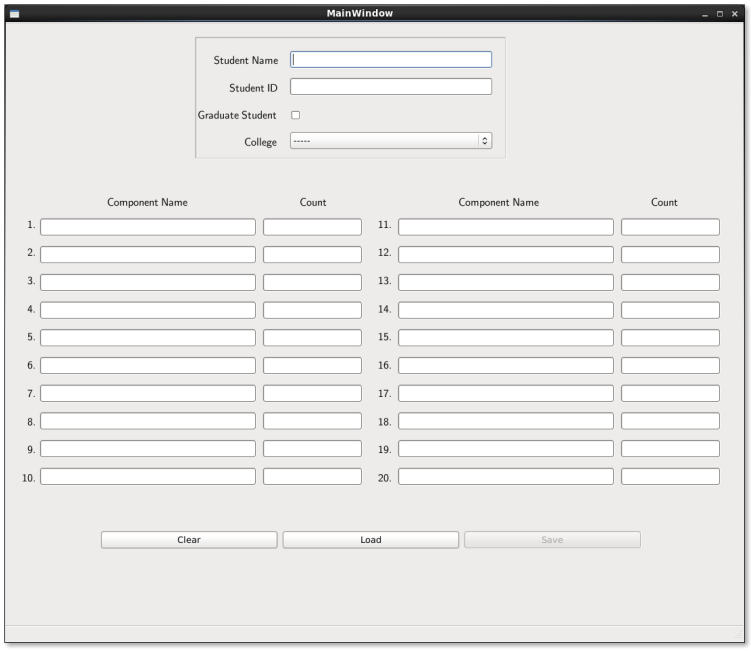

Figure 2: Initial Application State.

You application should adhere to the following behavior:

1. **"Clear" Operation:** At any point in the application lifetime, clicking on `Clear` must reset the form to the initial state.

2. **Data Entry:** You can start modifying the widgets freely. The moment you start modifying any data entry widget (the text boxes, the check box, or the combo box,) the `Save` button must be enabled, and the `Load` button disabled. The user may enter less than 20 components, and they may be entered in order, as in Figure 3, or out of order, as in Figure 4. Both of these are valid scenarios.

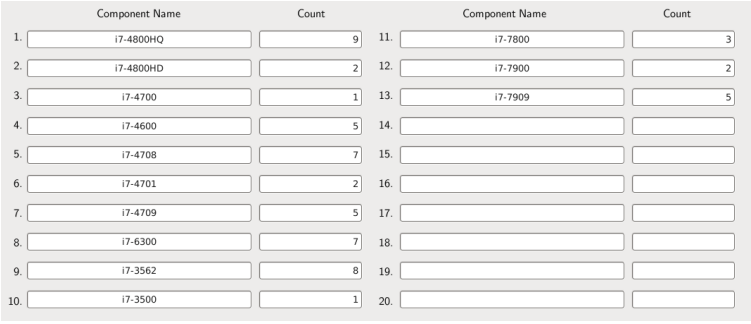

Figure 3: Data entered by the user in order.

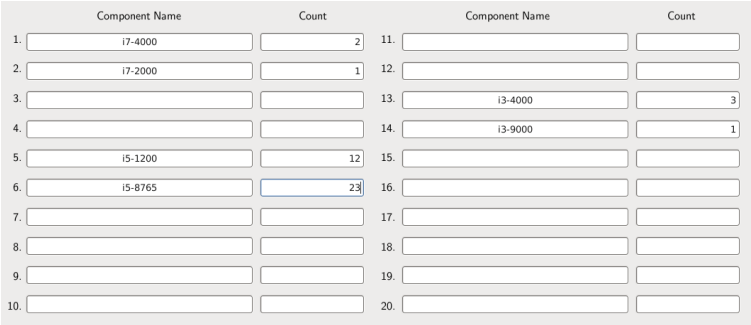

Figure 4: Data entered by the user out of order.

3. **Saving:** Clicking on `Save` will save an XML file, called `target.xml`, with the following format:

```
<?xml version="1.0" encoding="UTF-8"?>
<Content>
    <StudentName graduate="[true or false]">[Data]</StudentName>
    <StudentID>[Data]</StudentID>
    <College>[Data]</College>
    <Components>
        <Component name="[Data]" count="[Data]" />
        <Component name="[Data]" count="[Data]" />
        ...
    </Components>
</Content>
```

   The content of the XML should capture the data entered in all entry widgets. If the components are entered out of order, you will need to store them in the order of their presence in the components' section. Note that the XML file should not have empty entries.

4. **Loading:** If you are in the initial state, or if you just performed a form-reset via the `Clear` operation, (remember that this is the only time the `Load` button is enabled,) clicking on the `Load` will perform the following:

   - Show a file dialog box to "open" an XML file that conforms to the format mentioned above. This could be either a file that you have just saved, or a predefined file given to you.

     Note: You are given a method called `loadData(self)` that will show the dialog box and return the selected file path. Additionally, you are also given the method `loadDataFromFile(self, filePath)` that you need to populate with the file loading logic. If you fail to use these methods, the grading script will not function correctly.

   - Populate the form with the data obtained from file. Note that the number of available components may not be 20. If there are less components that 20, they must be populated in order, as shown in Figure 3. If the data file contains more than 20 components, you should display the first 20 present in the file and ignore the rest.
   - Once the data is loaded, disable the `Load` button and enable the `Save` button. This will put the application in the Ready-to-Save state, as if you have entered the data yourself.
   - You should be able to edit the data, and save it again.

Testing the GUI

Before you check in your code, make sure that the following tests produce the expected behavior:

1. **Initial State:** Start the application.
   Expected: Form will be in the initial state.

2. **Clearing:** Start the application → Modify widgets' content → Click on `Clear`.
   Expected: Form will reset to initial state.

3. **Entering Content:** Start the application → Modify the widgets → Click on `Save` → Save to `target.xml`.
   Expected: Content of the saved file must match the form.

4. **Modifying Content:** Start the application → Load the file `input.xml` → Modify widgets → Click on `Save` → Save to `target.xml`.
   Expected: Content of the saved file must match the form content.