

Instructions and Specifications

For Exam 2 over Tkinter, you will write a short program to create a form where you will read and write data from a file. For this assignment, we will simulate the data to be read by writing a function to return a list (simulated data from a file read). Using the form, you will edit (i.e., correct) the data in the GUI Tkinter application (that you will write). Once the data has been edited to the users satisfaction, they will click on the write button to simulate the writing of the edited data to a file; rather, than writing the data to a file, you will print the data to the terminal.

See **Figure 1** for a diagram of the GUI that you need to write. For this programming project you can use some of *your* previously written Tkinter code as the basis for this project. All of the code that is contained in your program must be of *your own making*.

Criteria 1: Layout Use Figure 1 as the basis for how the form should be organized in your window. Take note of the padding between cells. Use your own default settings in terms of what is aesthetically pleasing to you.

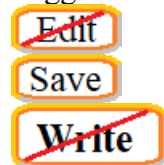
Criteria 2: Buttons and Entry Widgets

You need three buttons. See Figure 1.

Default State



Toggled State



Entry widgets containing the data should be disabled when the form is initially populated.

All images will be provided to you. Remember, we used images as buttons in Hybrid Assignment 3.

Criteria 3: Form Population

When you simulate the file read via a function call, you will need to begin the process of dynamically creating entry widgets. You may or may not attach these widgets (as data members) to the class that you will create. You will need to create an empty list as a data member. After you create each row of data in the form (two elements), you need to store the created widgets in a list. This should be a list of lists. So, create a list of the items that you create and then append that list to your data member. Please create a list of 4 elements, something like `[entryName, entry1, entryBday, entry2]` where `entryName` is the control variable for the `entry1` widget and `entryBday` is the control variable for the `entry2` widget.

Instructions and Specifications

Criteria 4: Functions that you need to write

As you are writing a window class for this assignment, it goes without saying that you will need to write an `__init__` function.

In addition, you will need to write the following functions (including appropriate docstrings).

Use your own meaningful function names for each function. The functions that need to be written are (in no particular order) as follows (all functions should be contained in your class).

Function1: `__init__`

For your initializing dunder method, I recommend that you set default values for formatting similar to what we did for the other Tkinter coding problems.

I suggest the following data members be implemented in your `__init__` function.

- list for storing the dynamically created widgets.
- rowCount
- Images (Create data attributes to store the images that we will need for our buttons)
 - editData
 - editDataSlash
 - saveData
 - saveDataSlash
 - writeData
 - writeDataSlash
- list for storing the data from the (simulated) file read

At the end of the `__init__` function (before the call to `mainloop()`), I would call the create form function (**Function7**).

Function2: Static Method to simulate the reading of the data file.

Use a static method (with the `@` decorator) to simulate the file read. This function when called will simply return the hard coded list of data.

```
@staticmethod
def readRecords():
    database = [
        ["Richy Blackmoore", "04/14/1945"],
        ["Tom Pettie", "10/20/1950"],
        ["Rosa Parks", "02/04/1913"],
        ["J.J. Rowling", "07/31/1965"],
```

Instructions and Specifications

```
["Carry Fisher","10/21/1956"],  
["Peter Seller","10/08/1925"],  
["Claudea Cardinale","04/15/1938"],  
["Dannie Kays","01/18/1911"],  
["Fran Jeffries","05/18/1930"],  
["Gioachino Rossetta","02/29/1792"],  
["Ada Lovelace","12/10/1815"],  
["Alan Roaming","06/23/1912"],  
["Michael Cane","03/14/1933"],  
["Sallie Ryde","05/26/1951"],  
]  
return database
```

Use your own function name. As you can observe, this function just returns the data as a two-dimensional list. For each person, it contains a name (with or without errors) and a birthdate. The correct information is presented as a list in Appendix 2. You will receive a data file where you can copy and paste the list (above) in your function. Do not use the data presented in the appendix, you (or me when I run and grade your code) will correct the typos using your GUI.

Function3: A function to populate your initial form.

After you read in the data (simulated by the function above), you will dynamically create a sequence of pairs of entry widgets (using a for loop as you loop through the two dimensional list of data). You will have a control variable for the name and an entry widget for the name. You will have a control variable for the birthdate and an entry widget for the birthdate. Inside your loop, you will:

- create the control variable, create the widget for the name
- create the control variable, create the widget for the birthday
- use the grid geometry manager to place them
- you will use increment the self.counter variable (or whatever name you give it) to determine the new row number and use this value in the grid manager

An example of the creation of an entry widget (inside your loop) is provided below:

```
entry1 = Entry(  
    self.root,  
    state=DISABLED,  
    textvariable = entryName,
```

Instructions and Specifications

```

        foreground='#000000',
        background='#FFFFFF',
        # background='#A9A9A9',
        width=20
    )

```

You will notice that the default state of the widgets is **DISABLED**. Also, as you iterate through your list of data, you will need to set the values of each of the entry widgets using the data and the set method. For example, if **entryName** is the control variable for the **entry1** widget and **name** points to the **name** from the data, then **entryName.set(name)** would correctly set the value for the entry widget.

In your loop (near the end), after your widgets have been created, create a temporary list

```
tlist = [entryName, entry1, entryBday, entry2]
```

and append this to the list attribute you created in your **__init__** method.

Nota Bene: You will note in this example, the entry widgets and control variables are **not** attached to the object (via **self**). You may follow this suggestion or not. You may infer that you do not have to do that.

Finally, this function should (after the loop executes), place the Write button at the end of the form in an enabled state. Since the default state of the form is closed, the write button should be enabled for (simulated) writing to a file. The image you will use for this button is



Function4: A function to open the form elements for editing.

Write a loop to step through your list and change the status of the widget elements to NORMAL. An example of the command for an entry widget (whose reference variable is **entry1**) would be: **entry1.config(state=NORMAL)**

(Remember, there are two entry widgets per row.) This function will have **three** other features. When the **Edit** button is clicked this function will be called; that is, this is the command callback function for your edit button. This function will change the state of the **Edit** button to disabled; if button is the name of the button widget is **self.button**, then

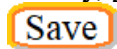
```
self.button.config(state=DISABLED)
```

will disable the button.

You will also toggle the image of the Edit button to



Finally, you will change the state of the **Save** button to normal and toggle the image of the Save button to



Instructions and Specifications

Function5: A function to close the form elements for editing.

Write a loop to step through your list and change the status of the widget elements to NORMAL. An example of the command for an entry widget (whose reference variable is **entry1**) would be: **entry1.config(state=DISABLED)**

(Remember, there are two entry widgets per row.) This function will have *three* other features. When the **Save** button is clicked this function will be called; that is, this is the command callback function for your Save button. This function will change the state of the **Save** button to normal (i.e., enabled); if button is the name of the button widget is **self.button**, then

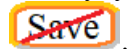
self.button.config(state=NORMAL)

will enable the button.

You will also toggle the image of the **Edit** button to



Finally, you will change the state of the **Save** button to normal and toggle the image of the Save button to



Nota Bene: Clearly, you will not name all of your buttons “**button**” nor will you name both of your entry widgets “**entry1**”.

Function6: Day of the Week Calculator

Use your function from the previous assignment.

Function7: Create the Form

This function will not actually write the elements to a file. Just write a loop that prints the contents of the data to the terminal in the following format:

Name: Richy Blackmoore	Birthday: 04/14/1945	The day of the week was Saturday.
Name: Tom Pettie	Birthday: 10/20/1950	The day of the week was Friday.
Name: Rosa Parks	Birthday: 02/04/1913	The day of the week was Tuesday.
Name: J.J. Rowling	Birthday: 07/31/1965	The day of the week was Saturday
...		

Notice the formatting of the data use the same values of 20 and 12 for the string formatted output.

Also, notice that for each birthdate, you will need to call your day-of-the week function which was previously written.

Instructions and Specifications

Criteria 4: Functionality

When you initially populate the form, the default state for the entry widgets should be disabled. That is the user needs to click the Edit button to open the form for editing. When the user clicks the **Edit** button,

- a) the Edit button and Write buttons should be disabled (since the form is not ready to be written to a file as editing is or will take place). Images of buttons should change to match disabled state.
- b) The **Save** button should be enabled (with appropriate image)
- c) All of the entry fields need to be enabled for editing.

Once editing is completed, the user should click the **Save** button. When the user clicks the **Save** button,

- a) the Edit button and Write buttons should be enabled (since the form is not ready to be written to a file as editing is or will take place).
- b) The **Edit** button should be enabled (with appropriate image)
- c) All of the entry fields need to be disabled for editing.

After all edits have been completed to the user's satisfaction, click the Write button to write the data to a file (really, print to the terminal).

Criteria 5: Day of the Week Computation

Use your previously written function, to compute the day of the week that corresponds to the birthday.

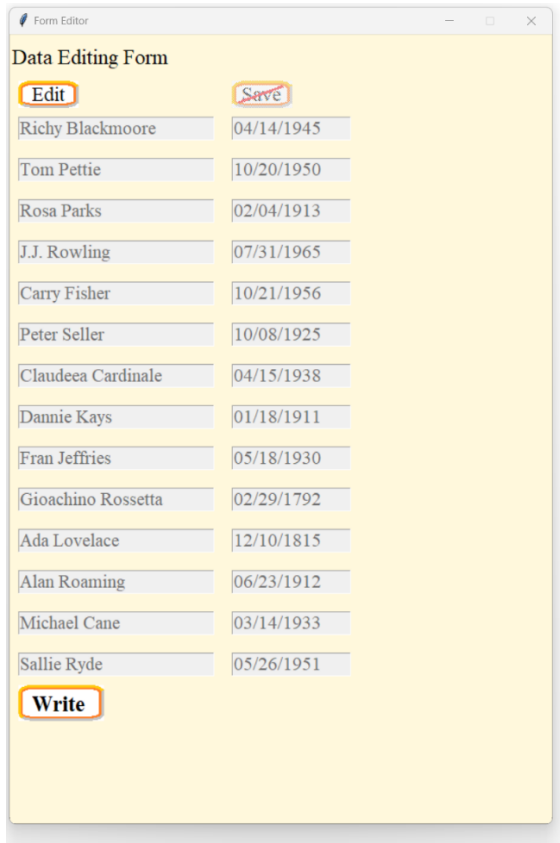
Criteria 6: Formatted Output for the Simulated File Write

In your simulated file write, which will simply be printed to the terminal. Use the following as a guide for your output formatting.

Name: Richy Blackmoore	Birthday: 04/14/1945	The day of the week was Saturday.
Name: Tom Pettie	Birthday: 10/20/1950	The day of the week was Friday.
Name: Rosa Parks	Birthday: 02/04/1913	The day of the week was Tuesday.
Name: J.J. Rowling	Birthday: 07/31/1965	The day of the week was Saturday

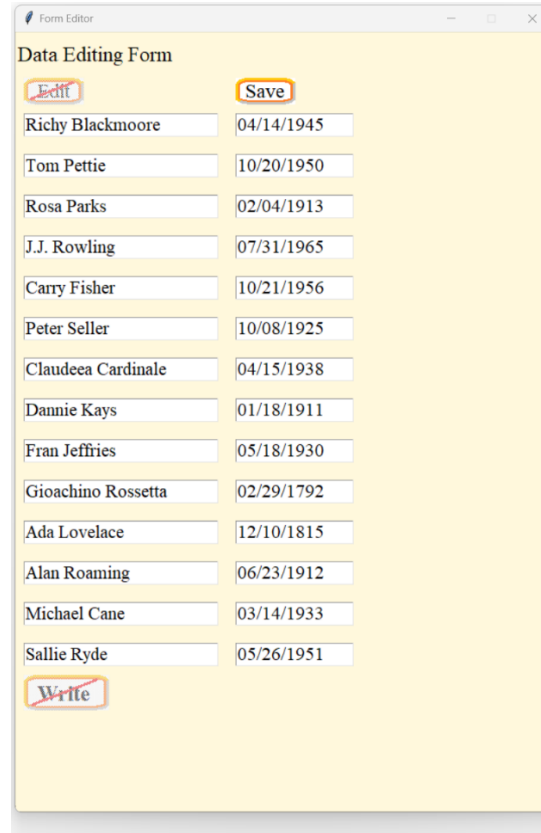
Instructions and Specifications

Appendix 1



Name	Date
Richy Blackmoore	04/14/1945
Tom Pettie	10/20/1950
Rosa Parks	02/04/1913
J.J. Rowling	07/31/1965
Carry Fisher	10/21/1956
Peter Seller	10/08/1925
Claudea Cardinale	04/15/1938
Dannie Kays	01/18/1911
Fran Jeffries	05/18/1930
Gioachino Rossetta	02/29/1792
Ada Lovelace	12/10/1815
Alan Roaming	06/23/1912
Michael Cane	03/14/1933
Sallie Ryde	05/26/1951

Figure 1



Name	Date
Richy Blackmoore	04/14/1945
Tom Pettie	10/20/1950
Rosa Parks	02/04/1913
J.J. Rowling	07/31/1965
Carry Fisher	10/21/1956
Peter Seller	10/08/1925
Claudea Cardinale	04/15/1938
Dannie Kays	01/18/1911
Fran Jeffries	05/18/1930
Gioachino Rossetta	02/29/1792
Ada Lovelace	12/10/1815
Alan Roaming	06/23/1912
Michael Cane	03/14/1933
Sallie Ryde	05/26/1951

Figure 2

Figure 1 depicts how your form should look in its initial state (i.e., when the window first opens). Notice the state of the buttons. Deactivated buttons have the “*slashed*” version of the image. Notice that entry widgets are grayed out because they are disabled in this state.

Figure 2 depicts the state after the Edit button has been clicked. Notice the state of the buttons. Deactivated buttons have the “*slashed*” version of the image. Notice that entry widgets are not grayed out because they are enabled for editing in this state.

Instructions and Specifications

Appendix 2

```
database = [  
    ["Ritchie Blackmore","04/14/1945"],  
    ["Tom Petty","10/20/1950"],  
    ["Rosa Parks","02/04/1913"],  
    ["J.K. Rowling","07/31/1965"],  
    ["Carrie Fisher","10/21/1956"],  
    ["Peter Sellers","10/08/1925"],  
    ["Claudia Cardinale","04/15/1938"],  
    ["Danny Kaye","01/18/1911"],  
    ["Fran Jeffries","05/18/1930"],  
    ["Gioachino Rossini","02/29/1792"],  
    ["Ada Lovelace","12/10/1815"],  
    ["Alan Turing","06/23/1912"],  
    ["Michael Caine","03/14/1933"],  
    ["Sally Ride","05/26/1951"],  
]
```