Final Project (Extra Credit)

The Recipe Application (GUI)

In this iteration of the Recipe Application, we'll be replacing the presentation layer that displayed to a console to use the Tkinter module to display a GUI interface.

Requirements

1. Update the presentation layer of The Recipe Application to have a GUI interface.
    a. Main window for the application
        i. Including an icon in the upper left corner
    b. Dropdown to display the categories
    c. Listbox (with a vertical scroll bar) to display the meals for a category
    d. Textbox (readonly and with vertical scrollbar) to display the menu directions
    e. Exit button
    f. Labels to describe the parts

Assets

1. The Recipe Application (Data and Business layers)
2. recipe_icon.ico

Updating the Application

We will need to update the Recipe Application.

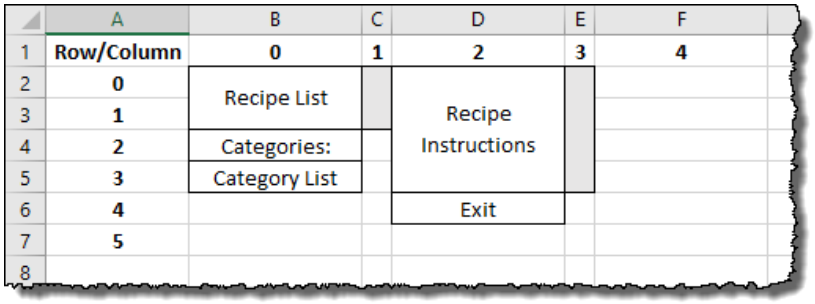
1. If it is not already opened, open the **myRecipeApplication**.

The GUI

We can now create the GUI. See Requirements above to understand what we need to create.

Designing the Screen

2. We'll be using the tkinter grid method so let's first map out how we want the screen to look. Remember, tkinter's grid uses rows and columns.
    a. Note: The user interface I am going to create is just a suggestion, please feel free to be creative and design your own if you want.
3. I am going to do this in Excel and create the interface.
    a. Note: The scrollbars (defined in grey) are separate widgets in tkinter and must be placed in separate columns from the listbox and textbox.

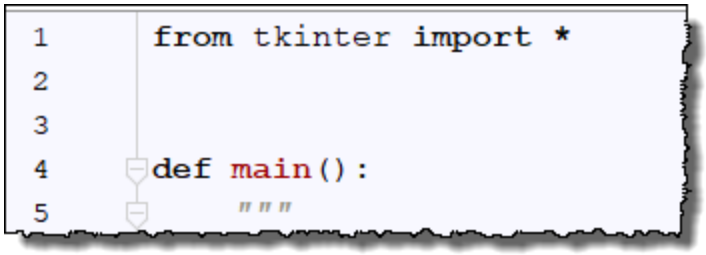


Creating the Main Screen

4. Now we'll create the GUI in python using the image above
5. Create a file named **recipes_ui.py**.
6. We'll start off by creating the main module so we can start testing right away

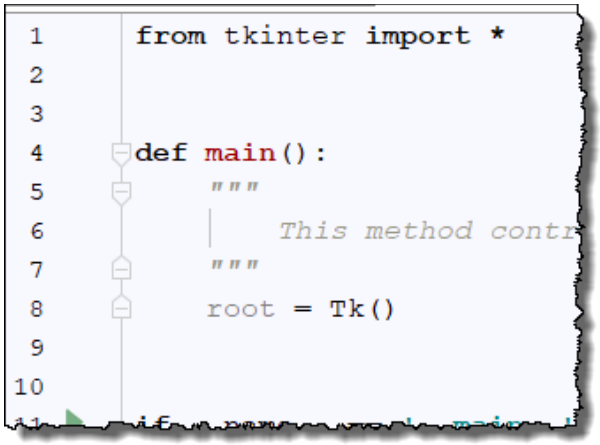
```
def main():
    """
        This method controls the main flow of the program
    """


if __name__ == '__main__':
    main()

```

7. For our interface, we'll start by adding the tkinter import.



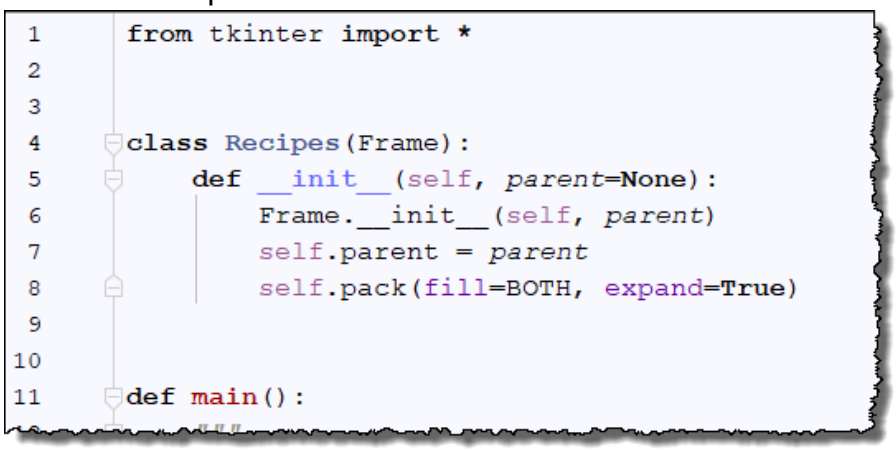
```
1   from tkinter import *
2
3
4   def main():
5       """
```

8. Now we'll go ahead and create the main (root) screen for our interface.



```
1   from tkinter import *
2
3
4   def main():
5       """
6           This method contr
7       """
8       root = Tk()
9
10
```

9. We'll create a class (Recipes) to build our layout. The layout will start off with a frame that will go into the root (pack) and we will end up putting all the components in the frame.
    a. Note, the root we created in step 8 above is the parent of this frame.
10. Code the Recipe class with the Frame as follows:



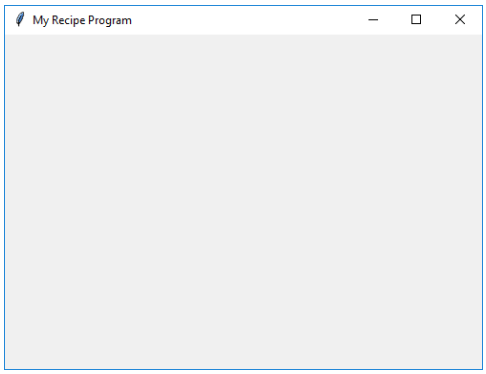
```
1   from tkinter import *
2
3
4   class Recipes(Frame):
5       def __init__(self, parent=None):
6           Frame.__init__(self, parent)
7           self.parent = parent
8           self.pack(fill=BOTH, expand=True)
9
10
11  def main():
```

11. We're almost ready to test but we first need to call the Recipes, passing the root. Additionally, we'll set the size and title while we're here as well as send the app into a loop of displaying the screen.

```
11  def main():
12      """
13          This method controls the main flow of the program
14      """
15      root = Tk()
16      app = Recipes(root)
17      app.parent.geometry("500x350")
18      app.parent.title("My Recipe Program")
19      root.mainloop()
20
```

Test the Main Screen

12. While not very exciting, let's test!



Adding the Components

13. We'll create a method (setup_ui) for adding all the components to the frame. Then we'll add the three labels which will be a 2-step process for each label, first step is to create the label and then add them to the form using the grid method. Review the layout in step 3 above to determine the row and column.

```
3
4   class Recipes(Frame):
5       def __init__(self, parent=None):
6           Frame.__init__(self, parent)
7           self.parent = parent
8           self.pack(fill=BOTH, expand=True)
9
10          # setup the screen
11          self.setup_ui()
12
13      def setup_ui(self):
14          """
15              This method will setup the UI:
16                  Add the widgets to the screen
17                  Configuring each widget as necessary
18          """
19
20          # Create and add the label widgets to the screen
21          recipe_lbl = Label(self, text="Recipes:")
22          recipe_lbl.grid(sticky=W, pady=10, padx=(10, 0))
23
24          categories_lbl = Label(self, text="Categories:")
25          categories_lbl.grid(row=2, sticky=W, pady=(10,0), padx=(10, 0))
26
27          directions_lbl = Label(self, text="Directions:")
28          directions_lbl.grid(row=0, column=2, sticky=W, padx=10, pady=10)
29
30
31  def main():
```

Test the Labels

14. Once again, let's test!

Add the OptionsMenu (drop down) for the Categories

15. The OptionMenu for the categories will require a few parameters
    a. A list to display
    b. The default item to display
16. Since we'll be getting the list from the request layer, we'll need to import the request file.

```
1   from tkinter import *
2
3   import requests
4
5
6   class Recipes(Frame):
7       def __init__(self, paren
```

17. Now we can get the list of categories from the request data layer.

```
29          directions_lbl = Label(self, text="Directions
30          directions_lbl.grid(row=0, column=2, sticky=W
31
32          # Get the categories from the site
33          category_list = requests.get_categories()
34
35
36  def main():
```

18. The OptionMenu displays a list of string elements and since the category list is a list of Category objects, we'll need to create a list to display that will just include the category name. We'll also add a default "Select:" to the list.

```
31
32          # Get the categories from the site
33          category_list = requests.get_categories()
34
35          # Create the display category list
36          display_category_list = ["Select:"]
37          for item in category_list:
38              display_category_list.append(item.get_category())
39
40
41  def main():
```

19. We can now create the default selection which will be the “Select:” element which was the first item we added to the display list.

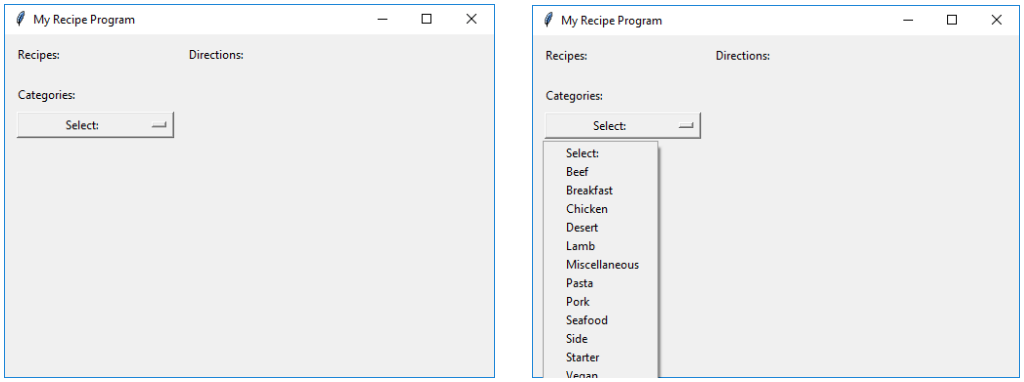
```
36          display_category_list = ["Select:"]
37          for item in category_list:
38              display_category_list.append(item.get_category())
39
40          # Create the default value to be displayed for the category list
41          default_category = StringVar(self)
42          default_category.set(display_category_list[0])
43
44
45      def main():
```

20. Now we’re ready to create the OptionMenu
    c. Note: The pady=5 and padx=(10,0). When the pad is set with 1 number, the padding will be applied to both side (right and left or top and bottom). To pad just one side, use two numbers in parenthesis, i.e. (10,0) where the first number will pad the top or left side and the second number will pad the bottom or right side.

```
41          default_category = StringVar(self)
42          default_category.set(display_category_list[0])
43
44          # Create the Options Menu widget for displaying the categories
45          categories_dropdown = OptionMenu(self, default_category,
46                                           *display_category_list)
47          categories_dropdown.grid(row=3, pady=5, padx=(10, 0))
48          categories_dropdown.config(width=20)
49
50
51      def main():
52          """
```

Test the Categories OptionMenu

21. Test again:



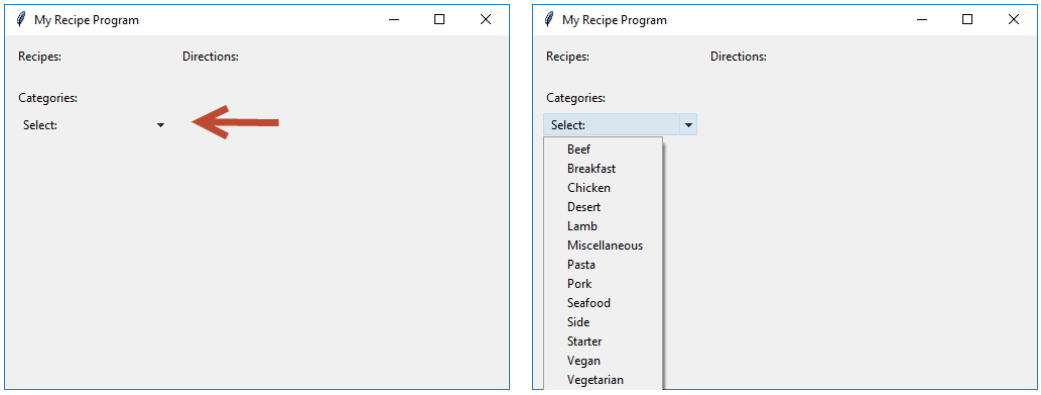
22. While this is functional, it's not really the look and feel we want so let's try the OptionMenu from the ttk import.
23. Start by adding the ttk to the imports.

```
1   from tkinter import *
2   from tkinter import ttk
3
4   import requests
5
6
```

24. Now reference the ttk OptionMenu when creating the widget.

```
43          default_category.set(display_category_list[0])
44
45          # Create the OptionMenu widget for displaying the categories
46          categories_dropdown = ttk.OptionMenu(self, default_category,
47                                               *display_category_list)
48          categories_dropdown.grid(row=3, pady=5, padx=(10, 0))
49          categories_dropdown.config(width=20)
50
```

25. And test again…



26. This is better but as you can see, is difficult to see when not selected (left image) so let's add a style to the widget to fix this issue.

Setting a Style

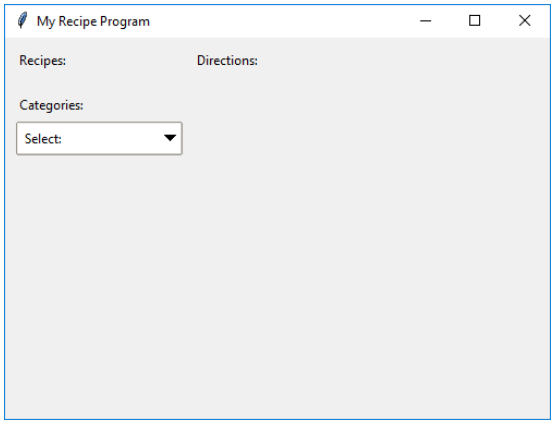
27. First, we'll need to create a style and we'll create this in a separate method.
28. In this style, we'll give the style a name, set a border width and set the background color.

```
47                                              *display_category_list)
48              categories_dropdown.grid(row=3, pady=5, padx=(10, 0))
49              categories_dropdown.config(width=20)
50
51          def set_style(self):
52              """
53                  This method creates a style to be used by the ttk widgets
54              """
55
56              style = ttk.Style(self.parent)
57              style.theme_use('clam')
58              style.configure('raised.TMenubutton', borderwidth=1, background="WHITE")
59
60
61      def main():
62          """
```

29. Next, we'll need to call the method to setup the style and then attach the style to the OptionMenu

```
42              default_category = StringVar(self)
43              default_category.set(display_category_list[0])
44
45              # Create style to update the OptionsMenu's look
46              self.set_style()
47
48              # Create the Options Menu widget for displaying the categories
49              categories_dropdown = ttk.OptionMenu(self, default_category,
50                                                   *display_category_list,
51                                                   style='raised.TMenubutton')
52              categories_dropdown.grid(row=3, pady=5, padx=(10, 0))
53              categories_dropdown.config(width=20)
```

30. And finally, test again.



31. This looks better. We will have to code for when the user selects a category from the list, but we're done with the OptionMenu for now.

Add the Listbox to Display the List of Meals

This widget will be a little different. So far, all the widgets have been static, ie once they are created, they don't change so we've just been able to create them in the setup_ui method. But the list of meals widget is different in that every time a user selects a different category, the displayed list in the widget will change. So, for the meal list listbox widget, we're going to create it in the "init" as a property of the Recipes class and then we can just refer to it anywhere else in the class.

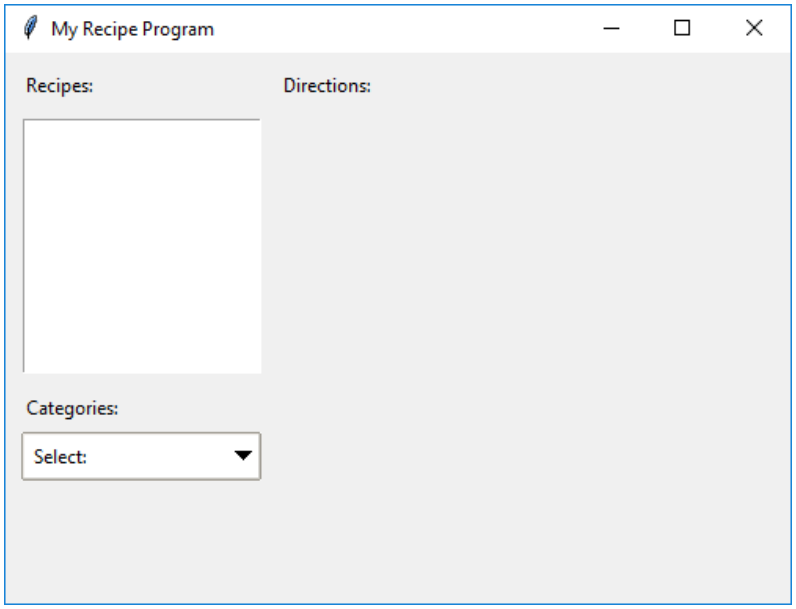
32. Create the meal_list_listbox as shown:

```
7   class Recipes(Frame):
8       def __init__(self, parent=None):
9           Frame.__init__(self, parent)
10          self.parent = parent
11          self.pack(fill=BOTH, expand=True)
12
13          # Create the screen widgets
14          self.meal_list_listbox = Listbox(self, width=25, height=10, selectmode=SINGLE)
15
16          # setup the screen
17          self.setup_ui()
18
19      def setup_ui(self):
20          """
```

33. Configure and add the listbox to the frame as follows:

```
54                                                  style='raised.TMenubutton')
55              categories_dropdown.grid(row=3, pady=5, padx=(10, 0))
56              categories_dropdown.config(width=20)
57
58              # Configure and add the listbox widget to display the recipe names
59              self.meal_list_listbox.configure(exportselection=False)
60              self.meal_list_listbox.grid(row=1, padx=(10,0), sticky='nw')
61
62      def set_style(self):
63          """
64              This method creates a style to be used by the tkk widgets
```

34. While the list box will be empty, let's test to make sure it displays and is in the correct location.



Populating the Meal List Listbox

Again, the meal list will be dynamic based on what category the user selects from the categories optionmenu.

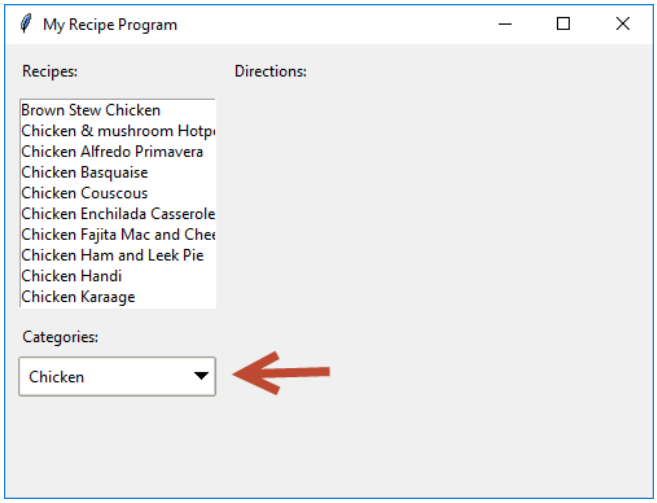
35. For reusability, we'll create a method for populating the meal list listbox based on a category that is passed in.

```
58          # Configure and add the listbox widget to display the recipe names
59          self.meal_list_listbox.configure(exportselection=False)
60          self.meal_list_listbox.grid(row=1, padx=(10,0), sticky='nw')
61
62      def load_meals(self, selected_category):
63          """
64              This method will populate the recipes names into a listbox.
65              The recipe list filtered by category
66          """
67
68          # Clear all information from the listbox
69          self.meal_list_listbox.selection_clear(0, END)
70          self.meal_list_listbox.delete(0, END)
71
72          # Get the list of meals from the site
73          meals_list = requests.get_meals_by_category(selected_category)
74
75          # Add the meals to the listbox
76          for item in meals_list:
77              self.meal_list_listbox.insert(END, item.get_meal())
78
79      def set_style(self):
80          """
81              This method creates a style to be used by the ttk widgets
82          """
```

36. We also need to add the actual call to load the meals when a category is selected.

```
48          # Create style to update the OptionsMenu's look
49          self.set_style()
50
51          # Create the Options Menu widget for displaying the categories
52          categories_dropdown = ttk.OptionMenu(self, default_category,
53                                               *display_category_list,
54                                               command=self.load_meals,
55                                               style='raised.TMenubutton')
56          categories_dropdown.grid(row=3, pady=5, padx=(10, 0))
57          categories_dropdown.config(width=20)
```

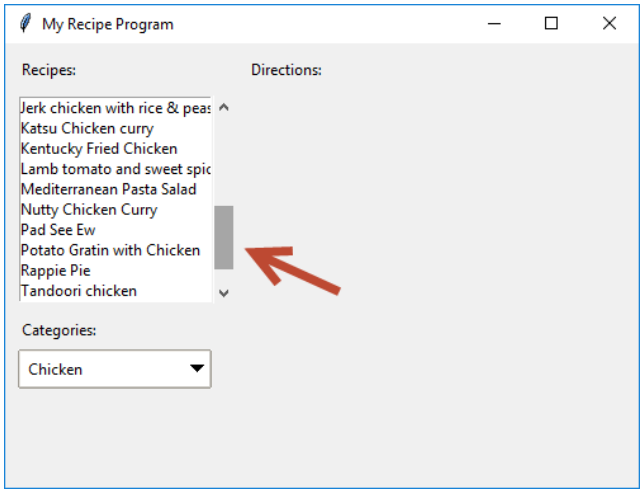
37. Now that it's all hooked up, we'll do a final test to verify the meal list is populating.
    d. Select a category from the optionsmenu.



38. The meal list is populated but there is currently no way to scroll through the list so we'll add a vertical scroll bar next.
    e. With the scrollbar, we first need to create it.
    f. Then we need to place it on the grid.
    g. And finally attach it to the meal_list_listbox as a vertical ("y") scrollbar.

```
61          self.meal_list_listbox.grid(row=1, padx=(10,0), sticky='nw')
62
63          # Create and attach a vertical scrollbar to the recipe listbox widget
64          recipe_list_scrollbar = Scrollbar(self, command=self.meal_list_listbox.yview)
65          recipe_list_scrollbar.grid(row=1, column=1, sticky='nsew')
66          self.meal_list_listbox['yscrollcommand'] = recipe_list_scrollbar.set
67
68      def load_meals(self, selected_category):
69          """
70              This method will populate the recipes names into a listbox.
```

39. Test the vertical scrollbar



Adding the Meal Instructions Textbox

It's time to add the meal preparation instructions to the UI. For this we'll use a text box and since the textbox is typically used to get text from the user and we only want to display the information, we'll set the textbox to be "readonly" meaning the user will not be able to enter data in.

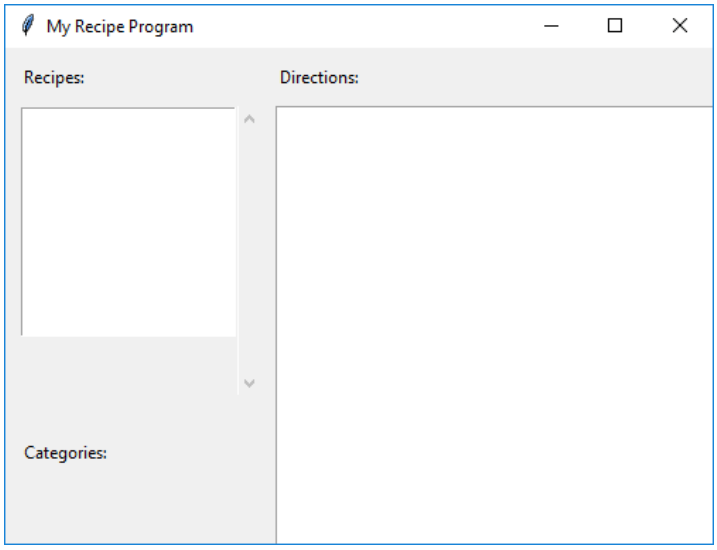
40. We first need to create the textbox and like the previous listbox, the contents will be dynamic so we'll create it in the init so it will be a property of the class.

```
9           Frame.__init__(self, parent)
10          self.parent = parent
11          self.pack(fill=BOTH, expand=True)
12
13          # Create the screen widgets
14          self.meal_list_listbox = Listbox(self, width=25, height=10, selectmode=SINGLE)
15          self.directions_text = Text(self, borderwidth=1)
16
17          # setup the screen
18          self.setup_ui()
19
20      def setup_ui(self):
```

41. Next, we'll add it to our form grid:

```
68
69          # Add and configure the textbox to display the recipe directions
70          self.directions_text.grid(row=1, column=2, columnspan=2, rowspan=4,
71                                    padx=(10,0), pady=(0,10), sticky=(N, S, E, W))
72
73      def load_meals(self, selected_category):
74          """
75              This method will populate the meal names into a listbox
```

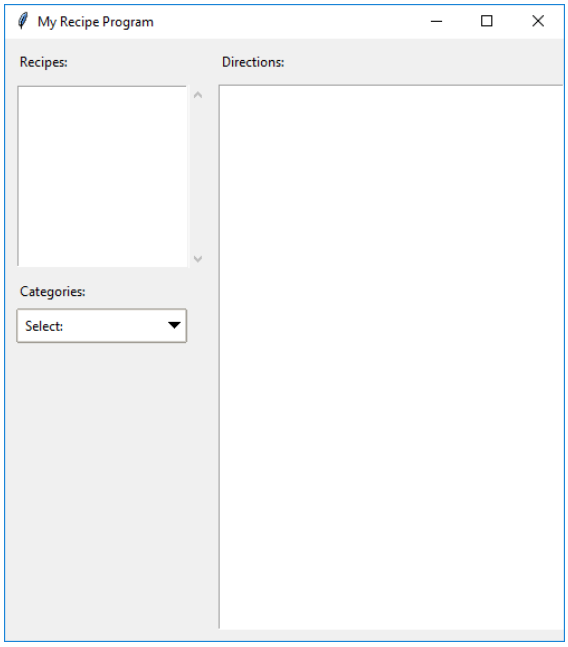
42. Let's test the screen again:
    h. You'll notice things got way out of sorts.



43. Now we'll start to pull this back together.
    i. If the user expands the main window vertically, horizontally or both, we want the direction text box to expand with it. To do that, add the following lines of code:

```
                                padx=(10,0), pady=(0,10), sticky=(N, S, E, W))

        # Set the row and column to expand when user adjusts the screen size
        self.columnconfigure(2, weight=1)
        self.rowconfigure(4, weight=1)

    def load_meals(self, selected_category):
        """
            This method will populate the recipes names into a listbox.
            The recipe list filtered by category
```

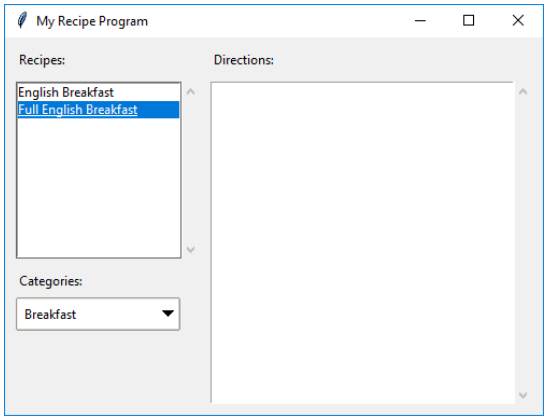
44. Now if we test again, we'll notice the screen looks good vertically. Additionally, if we expand the screen vertically, you should notice the recipe text box expands with the screen.



45. We're going to potentially have a lot of data in the recipe instruction text box so let's add a vertical scrollbar to the text box. Code as follows:

```
69          # Add and configure the textbox to display the recipe directions
70          self.directions_text.grid(row=1, column=2, columnspan=2, rowspan=4,
71                                    padx=(10,0), pady=(0,10), sticky=(N, S, E, W))
72
73          # Create and attach a vertical scrollbar to the recipe directions widget
74          directions_scrollbar = Scrollbar(self, command=self.directions_text.yview)
75          directions_scrollbar.grid(row=1, column=4, sticky='nsew', rowspan=4,
76                                    padx=(0,10), pady=(0,10))
77          self.directions_text['yscrollcommand'] = directions_scrollbar.set
78
79          # Set the row and column to expand when user adjusts the screen size
80          self.columnconfigure(2, weight=1)
81          self.rowconfigure(4, weight=1)
```

46. Test…



Populating the Recipe Instructions textbox

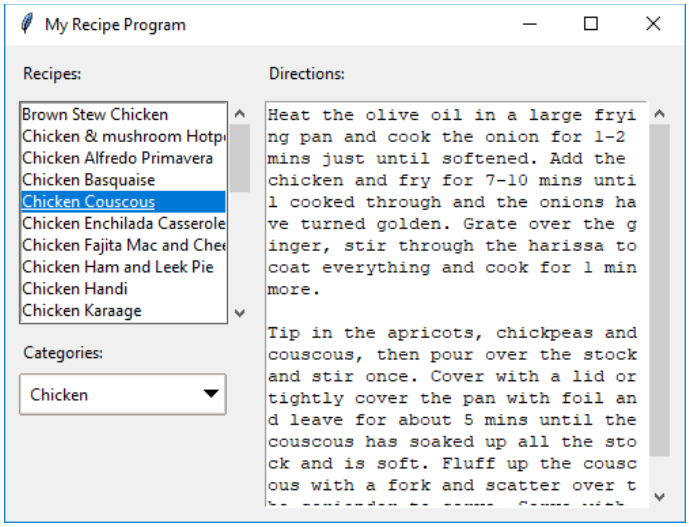
47. The screen looks good, but we still need to populate the recipe instruction text box.

```
81          self.rowconfigure(4, weight=1)
82
83      def load_meal(self, evt):
84          """
85              This method will populate the recipes listbox with the meals from the meals
86              by category. Note the Try Except... Selecting text in the meal directions
87              text box triggered this method and caused a 'Index out of Range Error' crash
88              when getting the index.  This try Except handles the crash.
89          """
90
91          try:
92              # Get the selected recipe from the recipe listbox
93              index = int(self.meal_list_listbox.curselection()[0])
94
95              # Get the recipe details
96              meal_name = self.meal_list_listbox.get(index)
97              meal = requests.get_meal_by_name(meal_name)
98
99              # Add the directions for the meal
100             self.directions_text.config(state="normal")
101             self.directions_text.insert(END, meal.get_instructions())
102             self.directions_text.config(state="disabled")
103         except IndexError:
104             pass
105
106     def load_meals(self, selected_category):
107         """
108             This method will populate the recipes names into a listbox.
```

48. The recipe direction should get added when the user selects on a meal in the list box so let's add the code to call the new load_meal method from there.

```
59
60              # Configure and add the listbox widget to display the recipe names
61              self.meal_list_listbox.configure(exportselection=False)
62              self.meal_list_listbox.bind('<<ListboxSelect>>', self.load_meal)
63              self.meal_list_listbox.grid(row=1, padx=(10,0), sticky='nw')
64
65              # Create and attach a vertical scrollbar to the recipe listbox widget
66              recipe_list_scrollbar = Scrollbar(self, command=self.meal_list_listbox.yv
```

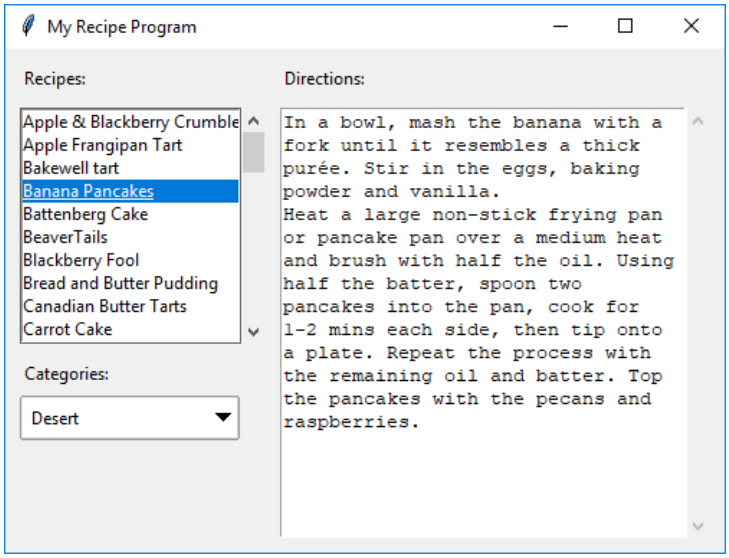
49. Test…



50. We're getting close, but you'll notice the text wrap is not correct, it's wrapping in the middle of words, we can easily fix that with the following line.

```
69
70          # Add and configure the textbox to display the recipe directions
71          self.directions_text.grid(row=1, column=2, columnspan=2, rowspan=4,
72                                    padx=(10,0), pady=(0,10), sticky=(N, S, E, W))
73          self.directions_text.config(wrap='word')
74
75          # Create and attach a vertical scrollbar to the recipe directions wid
```

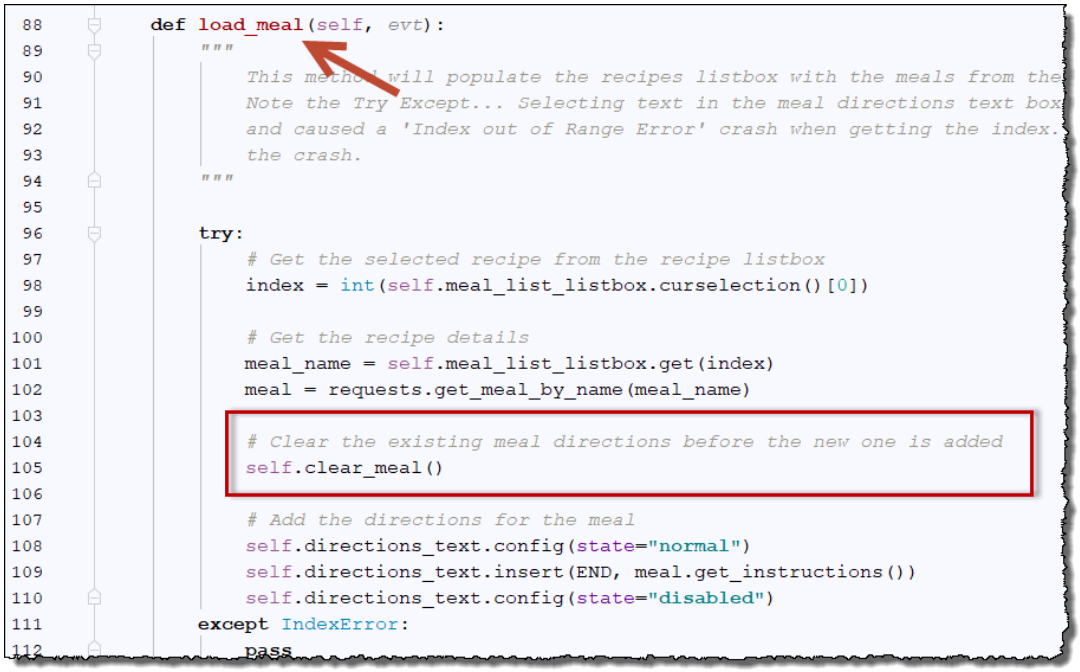
51. Again, we'll test.



52. The word wrapping issues is fixed but if you select another recipe, you'll notice that the recipe for the newly selected item gets added to the end of the directions and that's because when we add text to the directions text box, we're appended. So, what we need to do is clear the text box before adding the new recipe. We'll also clean the directions when the category is changed some at that time, there will not be a meal selected in the listbox.

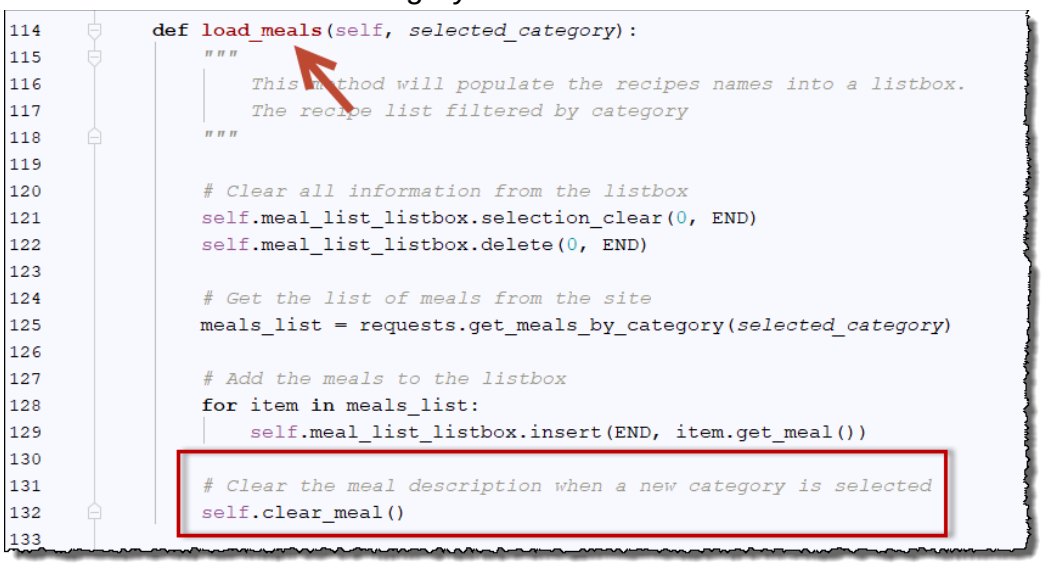
```
82          self.columnconfigure(2, weight=1)
83          self.rowconfigure(4, weight=1)
84
85      def clear_meal(self):
86          self.directions_text.config(state="normal")
87          self.directions_text.delete(1.0, END)
88          self.directions_text.config(state="disabled")
89
90      def load_meal(self, evt):
91          """
```

53. Now we need to call this in the two locations:
    j. When a meal is selected from the list…



```
88      def load_meal(self, evt):
89          """
90              This method will populate the recipes listbox with the meals from the
91              Note the Try Except... Selecting text in the meal directions text box
92              and caused a 'Index out of Range Error' crash when getting the index.
93              the crash.
94          """
95
96          try:
97              # Get the selected recipe from the recipe listbox
98              index = int(self.meal_list_listbox.curselection()[0])
99
100             # Get the recipe details
101             meal_name = self.meal_list_listbox.get(index)
102             meal = requests.get_meal_by_name(meal_name)
103
104             # Clear the existing meal directions before the new one is added
105             self.clear_meal()
106
107             # Add the directions for the meal
108             self.directions_text.config(state="normal")
109             self.directions_text.insert(END, meal.get_instructions())
110             self.directions_text.config(state="disabled")
111         except IndexError:
112             pass
```

    k. And when a new category is selected:



```
114     def load_meals(self, selected_category):
115         """
116             This method will populate the recipes names into a listbox.
117             The recipe list filtered by category
118         """
119
120         # Clear all information from the listbox
121         self.meal_list_listbox.selection_clear(0, END)
122         self.meal_list_listbox.delete(0, END)
123
124         # Get the list of meals from the site
125         meals_list = requests.get_meals_by_category(selected_category)
126
127         # Add the meals to the listbox
128         for item in meals_list:
129             self.meal_list_listbox.insert(END, item.get_meal())
130
131         # Clear the meal description when a new category is selected
132         self.clear_meal()
133
```

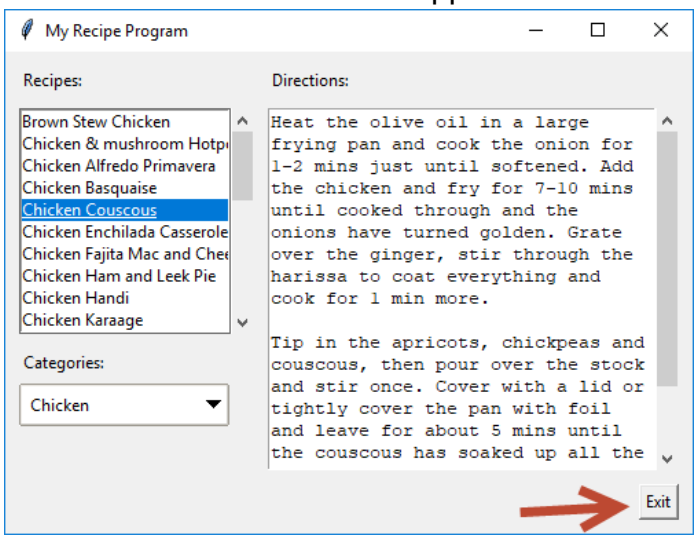
54. Test by selecting a category then a meal. Select another meal and make sure the previous directions were removed. Additionally, try selecting a different category and verifying the directions from the previous selected meal has been removed.

Adding an Exit Button

55. We need to add an exit button, so we'll do that next.

```
78                                  padx=(0,10), pady=(0,10))
79          self.directions_text['yscrollcommand'] = directions_scrollbar.set
80
81          # Create and add an exit button
82          exit_button = Button(self, text="Exit", command=self.parent.destroy)
83          exit_button.grid(row=5, column=3, pady=(0,10), padx=10, columnspan=2)
84
85          # Set the row and column to expand when user adjusts the screen size
86          self.columnconfigure(2, weight=1)
87          self.rowconfigure(4, weight=1)
88
```

56. Test to validate the exit button is displayed. Additionally, clicking on the exit button should shut down the application.

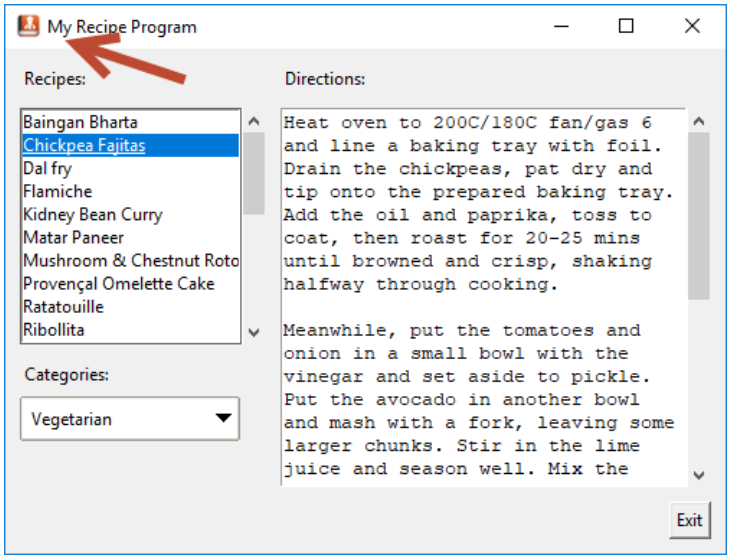


Adding an Application Icon

57. Finally, one last thing we need to do to our application, and that is update the icon in the upper left-hand corner of the window. You can download the icon I am using from Canvas or use your own.
58. Copy the icon into the root folder for your application
59. Add the following line to attach the icon:

```
144 def main():
145     """
146         This method controls the main flow of the program
147     """
148     root = Tk()
149     app = Recipes(root)
150     app.parent.geometry("500x350")
151     app.parent.title("My Recipe Program")
152     app.parent.iconbitmap('recipe_icon.ico')
153     root.mainloop()
154
155
156 if __name__ == '__main__':
157     main()
```

60. And finally, test!



That should wrap up the extra credit part of the final project, just test, test, test.