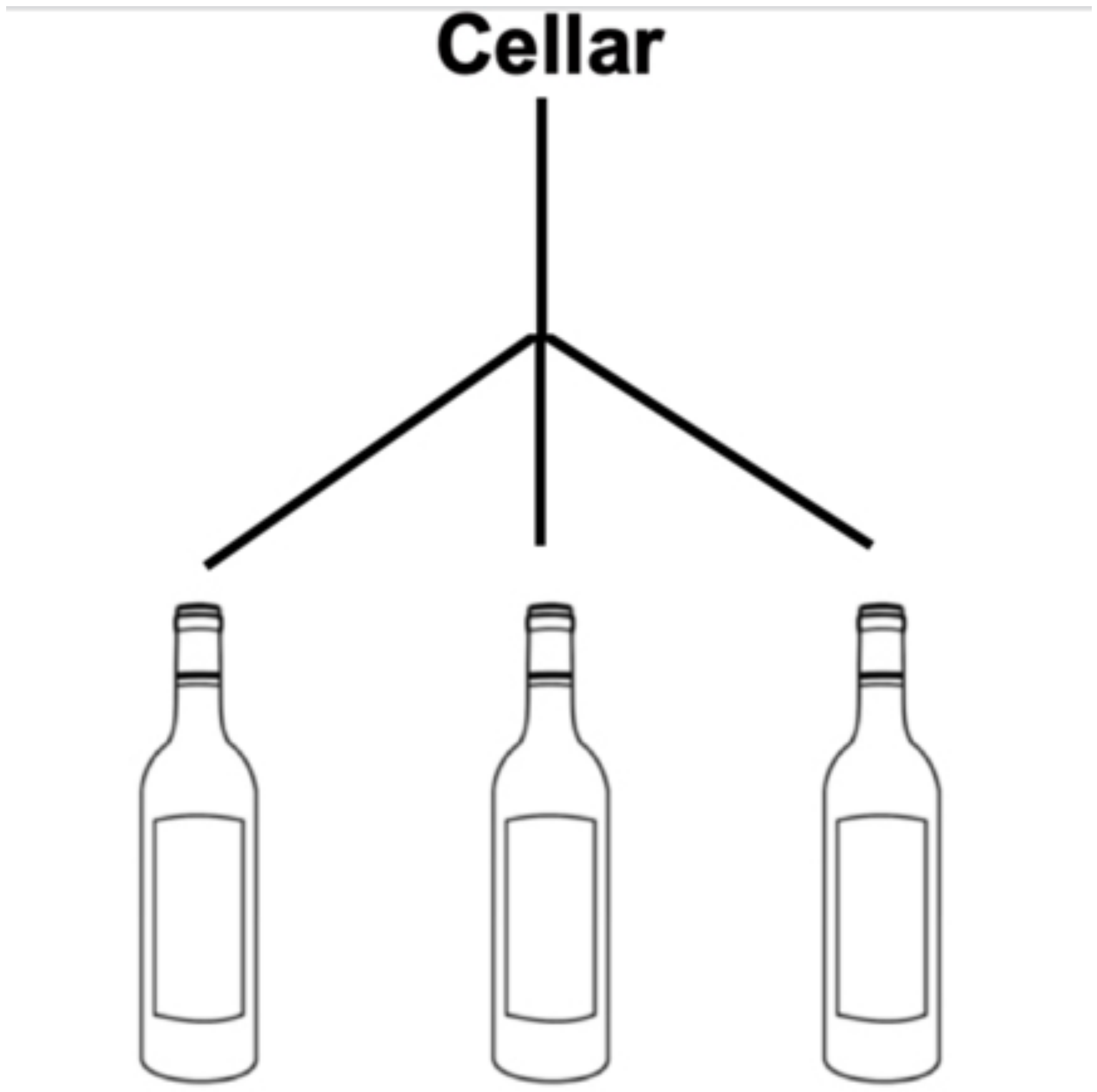
The Wine Library

Our task is to implement a management system for a wine library.

We will consider a physical structure consisting of many racks, and each rack contains different bottles of wine.



We begin with a single bottle of wine. This is a physical object which has a number of *properties, e.g. winery, varietal, vintage, etc. A rack is a storage element which contains a number of slots in which to store bottles of wine.*

Abstracting the Physical Wine Cellar

Clearly, each bottle of wine is a physical object belonging to the class of objects we might define as the bottle class. Each bottle (object) has a set of attributes:

Attribures: Bottle Object

bottle_id: Each bottle we purchase and add to our cellar has a unique designation. Even after we consume the contents, we will not reassign that bottle_id to a new bottle. Rather, we will keep a record of it in a list of bottles we have consumed.

category: red, white, rose

winery: The winemaker is an important attribute of a bottle object.

varietal: Chardonnay, Pinot Noir, etc.

vintage: The year the wine was bottled

category: This attribute is a designation that differentiates varietals from a winery: examples: Clone 5, Reserve, Dominique, etc.

Of course, there are any number of other attributes we could add including: purchase date, purchase price, etc. It is reasonably easy to add these to our class Bottle.

At this point we need to make a decision about how we model the wine racks. Are they objects, or are they attributes of a bottle (where the bottle is stored)?. We could choose either approach. In a first approach to modeling the cellar, I choose to add the following attributes to the bottle object, making the rack id and location slot properties of the bottle class.

Additional Attributes: Bottle Object

bin_no: The location of the bottle the rack in which it is stored.

irow: The location of the bottle in irow of this rack

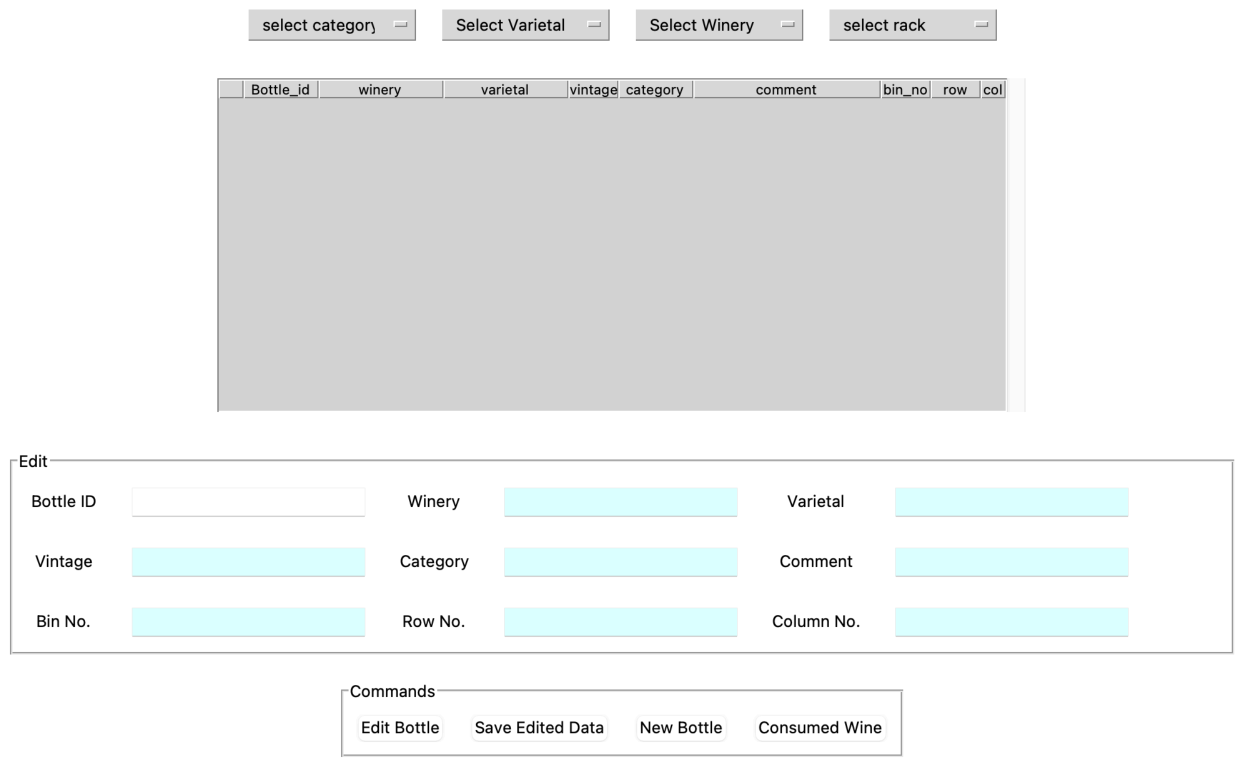
icol: The location of the bottle in icolumn of this rack

Program Vision

This program uses object oriented programming to manage a wine library. The primary object is a bottle of wine. Data is stored in a *comma separated file (csv)* with each record containing the attributes of a bottle (as summarized above). A Graphical User Interface (GUI) provides the user with an ability to view the library by selecting bottles by:

- CATEGORY (red, white, rose)
- VARIETAL (Chardonny, Pinot Noir, Zinfandel, etc.)
- WINERY
- STORAGE RACK

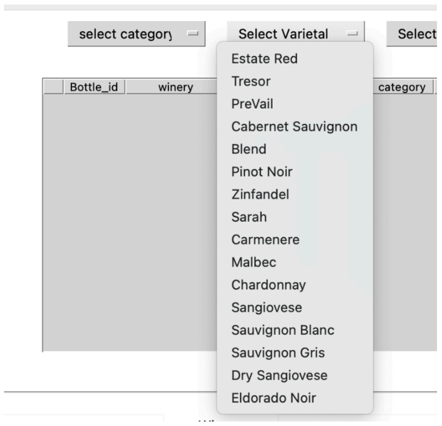
The Graphical User Interface is pictured below.

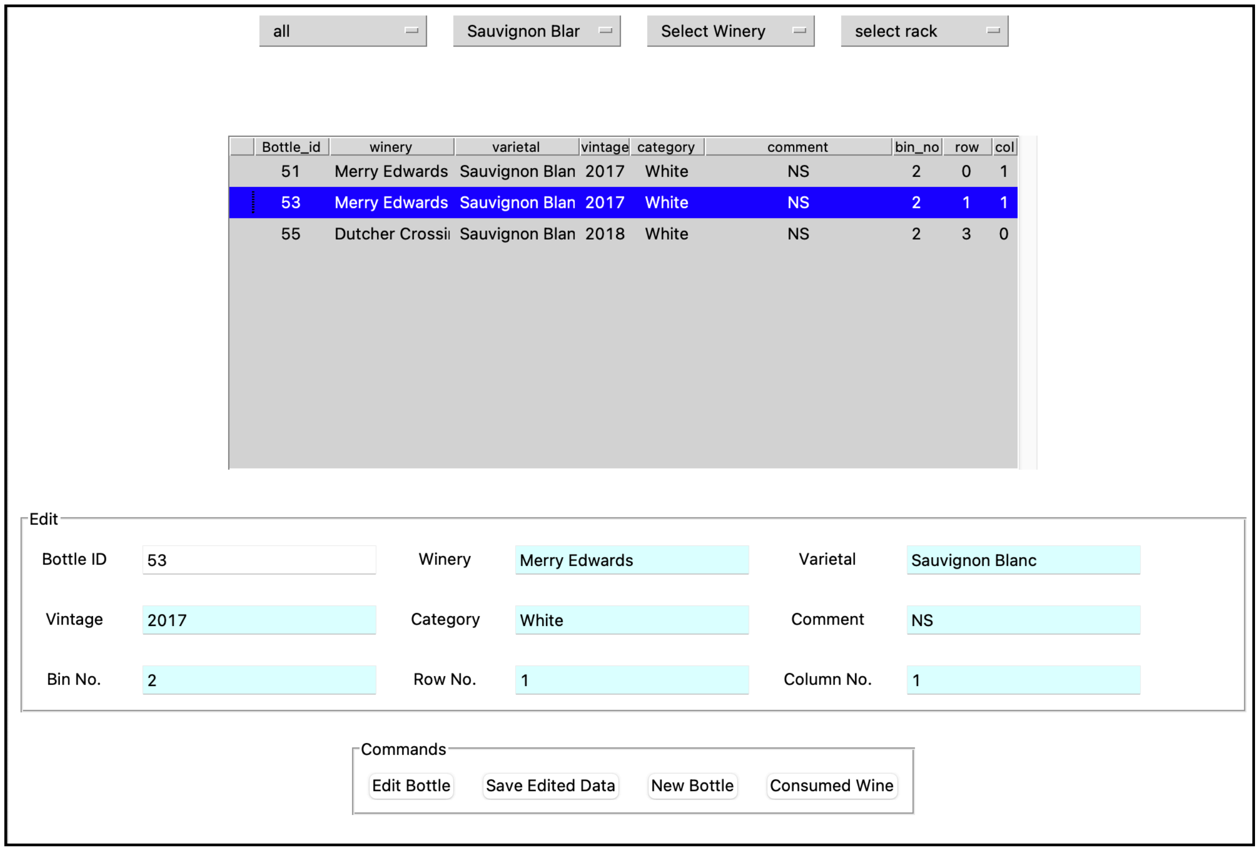


The top row of widgets are drop-down menus that provide a list of values for the attributes of all bottles in the wine library. These lists are constructed when the wine data is read into to the program.

The second element of the GUI is the larger grey block. Once an attribute value is selected, the list of all bottles in the library (with that attribute value) appear here

Below, the user has selected the varietal Sauvignon Blanc. The three bottles in the library are displayed. Here we use the mouse to select bottle 53 a 2017 Merry Edward Sauv. Blanc. Once we select a bottle from this inventory, the attributes are displayed below in Entry boxes in the EDIT FRAME. At this point we can make any changes to the data[1]. In the final frame below (the COMMAND FRAME) we can use the UPDATE button on the left to change the data held by bottle object 53. If we withdraw the bottle from the library (consume it, for example) we press the CONSUMED button. That does not delete the bottle, but it stores it in a special rack that we can peruse at a later time to recall a wine that we have consumed.





Bottle_id	winery	varietal	vintage	category	comment	bin_no	row	col
51	Merry Edwards	Sauvignon Blan	2017	White	NS	2	0	1
53	Merry Edwards	Sauvignon Blan	2017	White	NS	2	1	1
55	Dutcher Crossi	Sauvignon Blan	2018	White	NS	2	3	0

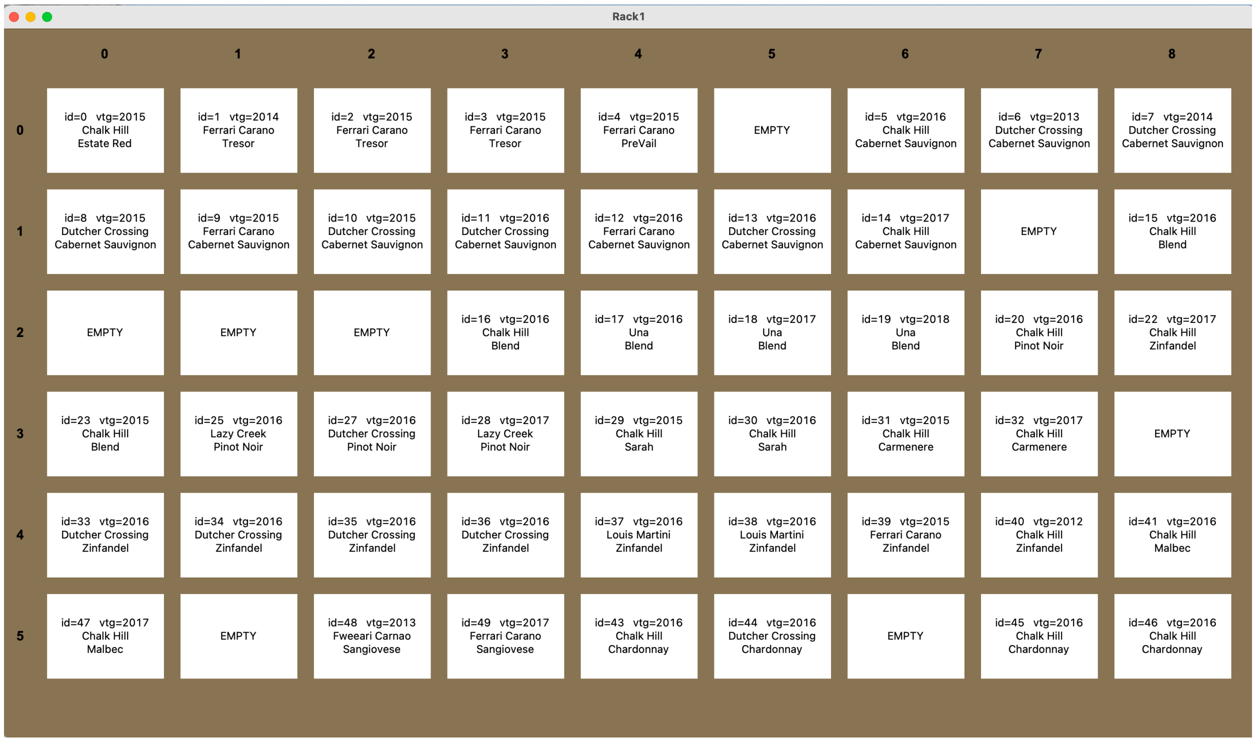
Edit

Bottle ID	53	Winery	Merry Edwards	Varietal	Sauvignon Blanc
Vintage	2017	Category	White	Comment	NS
Bin No.	2	Row No.	1	Column No.	1

[1] The bottle number cannot be edited. Even if the user were to edit Bottle ID here, it will not be changed in the bottle object.

If we don't select a bottle from the BOTTLE FRAME, the entry boxes below remain blank. Here we can add a bottle to our library by typing the details into the EDIT box. Once we have entered the data, the NEW BOTTLE button will enter the new wine data. The program will look up the BOTTLE ID of the last bottle in the library and assign the next bottle_id to tag the entry.

Notice that the last pull down menu in the SELECT frame enables a view of one of the wine racks in the library. Rack 0 is the list of all bottles withdrawn from the library (consumed), while racks 1,2, and 3 are physical racks. If we select rack 1 we get a display of the bottles in rack 1 which is a 9 x 6 matrix of physical spaces. The GUI displays the bottle number, vintage, winery, and varietal stored in each slot of the physical rack.



	0	1	2	3	4	5	6	7	8
0	id=0 vtg=2015 Chalk Hill Estate Red	id=1 vtg=2014 Ferrari Carano Tresor	id=2 vtg=2015 Ferrari Carano Tresor	id=3 vtg=2015 Ferrari Carano Tresor	id=4 vtg=2015 Ferrari Carano PreVail	EMPTY	id=5 vtg=2016 Chalk Hill Cabernet Sauvignon	id=6 vtg=2013 Dutcher Crossing Cabernet Sauvignon	id=7 vtg=2014 Dutcher Crossing Cabernet Sauvignon
1	id=8 vtg=2015 Dutcher Crossing Cabernet Sauvignon	id=9 vtg=2015 Ferrari Carano Cabernet Sauvignon	id=10 vtg=2015 Dutcher Crossing Cabernet Sauvignon	id=11 vtg=2016 Dutcher Crossing Cabernet Sauvignon	id=12 vtg=2016 Ferrari Carano Cabernet Sauvignon	id=13 vtg=2016 Dutcher Crossing Cabernet Sauvignon	id=14 vtg=2017 Chalk Hill Cabernet Sauvignon	EMPTY	id=15 vtg=2016 Chalk Hill Blend
2	EMPTY	EMPTY	EMPTY	id=16 vtg=2016 Chalk Hill Blend	id=17 vtg=2016 Una Blend	id=18 vtg=2017 Una Blend	id=19 vtg=2018 Una Blend	id=20 vtg=2016 Chalk Hill Pinot Noir	id=22 vtg=2017 Chalk Hill Zinfandel
3	id=23 vtg=2015 Chalk Hill Blend	id=25 vtg=2016 Lazy Creek Pinot Noir	id=27 vtg=2016 Dutcher Crossing Pinot Noir	id=28 vtg=2017 Lazy Creek Pinot Noir	id=29 vtg=2015 Chalk Hill Sarah	id=30 vtg=2016 Chalk Hill Sarah	id=31 vtg=2015 Chalk Hill Carmenere	id=32 vtg=2017 Chalk Hill Carmenere	EMPTY
4	id=33 vtg=2016 Dutcher Crossing Zinfandel	id=34 vtg=2016 Dutcher Crossing Zinfandel	id=35 vtg=2016 Dutcher Crossing Zinfandel	id=36 vtg=2016 Dutcher Crossing Zinfandel	id=37 vtg=2016 Louis Martini Zinfandel	id=38 vtg=2016 Louis Martini Zinfandel	id=39 vtg=2015 Ferrari Carano Zinfandel	id=40 vtg=2012 Chalk Hill Zinfandel	id=41 vtg=2016 Chalk Hill Malbec
5	id=47 vtg=2017 Chalk Hill Malbec	EMPTY	id=48 vtg=2013 Fweeari Carnao Sangiovese	id=49 vtg=2017 Ferrari Carano Sangiovese	id=43 vtg=2016 Chalk Hill Chardonnay	id=44 vtg=2016 Dutcher Crossing Chardonnay	EMPTY	id=45 vtg=2016 Chalk Hill Chardonnay	id=46 vtg=2016 Chalk Hill Chardonnay

Implementation

There are 9 major divisions in the program. These are identified in the code by the following:

- IMPORT PACKAGES TO SUPPORT THE PROGRAM
- DEFINE THE BOTTLE CLASS
- INSTANTIATE BOTTLE OBJECTS FROM Wine-data.csv

- CREATE THE GUI INTERFACE
    - SELECT FRAME
    - BOTTLE FRAME
    - EDIT FRAME
    - COMMAND FRAME AND COMMAND BUTTONS
- START TKINTER EVENT MONITORING

We will summarize each of these in the remainder of this documentation. I'll begin, however, with the last one first. It is the simplest to write, but is the key to understanding the GUI interface and the functioning of the program.

```
##################################################
# START TKINTER EVENT MONITORING
##################################################
root.mainloop()
```

The method mainloop() is the last statement in the program. It operates on the root window, in which all of the interactive tkinter widgets are placed. This function detects events associated with tkinter widgets. These are reflected in changes in the display, such as mouse clicks, button presses, key strikes, etc. After detecting an event, an entry is placed in the event queue. This launches a call which is routed to an appropriate function (defined in the tkinter widget) to implement an action in response to the event. The event loop functions as an infinite loop, continually looking for tkinter events.

```
##################################################
# IMPORT PACKAGES TO SUPPORT THE PROGRAM
##################################################
```

Lines 7-9

There are two packages of modules which we need to import to support the program: csv, and tkinter. The first of these enables us to read and write to a comma separated data file. The second (tkinter) enables the GUI

```
##################################################
# DEFINE THE BOTTLE CLASS
##################################################
```

Lines 15-44

The Bottle Class has nine attributes. (0) bottle_id, (1) winery, (2) varietal, (3) vintage, (4) category (cat) specifying red, white or rose wines, (5) comment specifying the differentiation of varietals from this winery, (6) bin_no (or storage rack), (7) irow: row address in the bin, (8) icol: the column address in the bin. The variable **bottles** is a list of bottle objects. The third object[2] in this list bottles[2] will be a bottle object which has a representation with it's nine attributes (separated by commas) as:

3, Ferrari Carano, Tresor, 2015, Red, NS, 1, 0, 2

In addition, we define two methods which can be applied to any bottle object. The first is **update_bottle**. The argument is passed to this function from the command frame (which we define later in this discussion). It identifies a specific bottle by its id, passed to the function as bid. There are a set of statements like:

```
bottles[bid].winery=bot_winery_entry.get()
```

.get() is the method to read information from an entry box (we define this in in the program) and `bottles[bid].winery` *is the* winery attribute of the bid element in the list bottles.

The method **display_bottles** writes the attributes of a subgroup of bottle objects into a tabular display called **bottle_tree**. Again, this is described below.

The important thing to understand at this point is that the bottle object can have its attributes updated (**update_bottle(...)** and it can be displayed in tablular form in the the GUI.

```
###################################################
# INSTANTIATE BOTTLE OBJECTS FROM Wine-data.csv
###################################################
```

Lines 50-79

The attributes of the wine library are stored in a csv file named:

```
Data/Wine-data.csv
```

This is a *comma separated values* file in which the first record is a header which lists the attributes, by name. Subsequent records list the values of the attributes of each bottle in the library. Before reading data into the program, two variables must be initialized: nbottles=0 (the number of records in the file) and the variable bottles=[ ]

[2] Recall that indexing in Python begins at 0

designating an empty list. A for loop iterates over the elements of rows of the csv records. After each record is read, a new_bottle object is created using the attribute data found in the file, and new_bottle is appended to the list bottles objects, which was initially a null list. After finishing reading the data, nbottles will be the number of objects, each representing a bottle in the library. The lists: varietal_list, cat_list, and winery_list are constructed to contain these attributes which appear in the data and will later be used to sort the data.

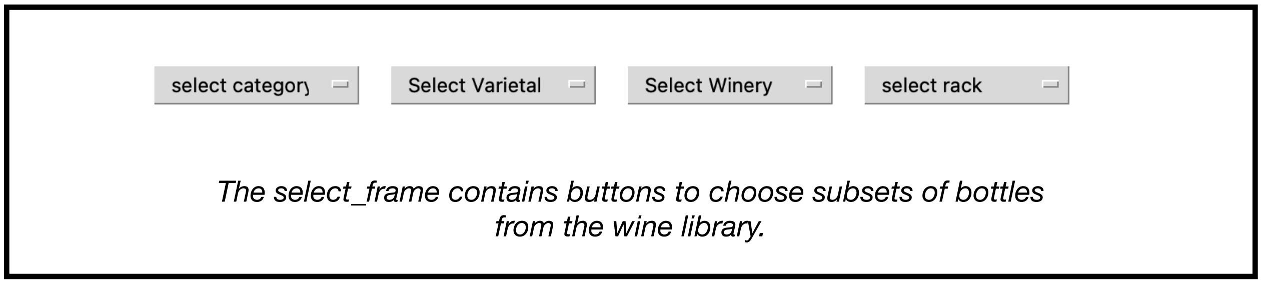
```
##########################################################
# CREATE THE GUI INTERFACE
##########################################################
```

Lines 84-90

Using tkinter we create a window named **root** which will contain the elements of the GUI interface. Within root we define tkinter StringVars which will be used to link user events to widgets. Specifically, these variables are **select_cat**, **select_var**, **select_winery**, and **select_rack**, We will see shortly that these variables indicate that the user wants to display a wine category (red, white, rose), varietal, winery, or storage rack. These variables will be used to displays the selected group of bottle objects in tabular form.

```
##########################################################
# SELECT FRAME
##########################################################
```

Lines 97-195

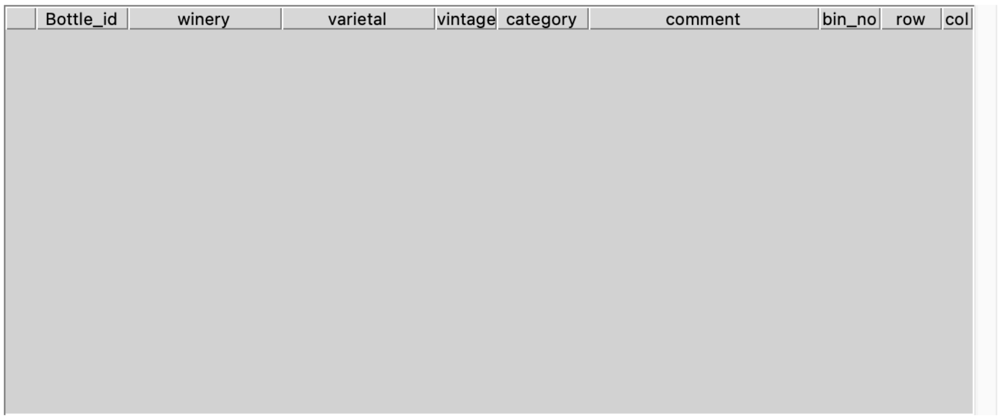


The select_frame contains buttons to choose subsets of bottles from the wine library.

The function of the **select_frame** is display the option menus so that the user can display bottles in the library by category, varietal, winery, or storage rack. We must create the select_frame and then pack or put it in root. (lines 173-174)

The select_frame contains four option menus. The following code creates the menu for the categories of wine (red, white, rose).

```
#-----Create option menu (cat) and grid in select frame
cat_menu=ttk.OptionMenu(select_frame,select_cat,*cat_list,command=update_cat)
cat_menu.config(width=10)
cat_menu.grid(column=0, row=0,padx=10,pady=10)
```



The bottle_frame contains a tabular display of bottles chosen from the select menus.

The name of the menu is **cat_menu** and it is created using the constructor **ttk.OptionMenu**(,,,) . The arguments are the parent in which the menu will be placed (**select_frame**), next we specify the tkinter StrVariable associated with this menu **select_cat,** and finally the list of categories **cat_list** collected from the csv file. The OptionMenu object assigns the second argument, select_cat, to the value in the list that the user selects[3]. The last argument of cat_menu is `command=update_cat`; this encapsulates the ACTION. When you select a particular option (e.g. 'red') the function update_cat will be executed. The code for this function is at the beginning of the program segment denoted by SELECT FRAME.

This section of the program begins with four functions named:

1. update_cat(event)
2. update_var(event)
3. update_winery(event)
4. update_rack(event)

[3] This event and its handling is built into the class of objects OptionMenu and is inherited by any object created using the constructor.

Each of these functions updates a display of a selected subgroup (using the menus just defined) in the BOTTLE FRAME. They are triggered by an event which is an argument of each function. We will defer discussion of the details of these functions until the next section.

Up to this point we have embedded widgets within frames using the method .pack() . This simply adds the widget to the next available space within the frame in which it is defined. In line 179 we use an alternate approach using the method .grid() . This method uses a grid of rows and columns in a frame. It is convenient for placing a group of objects. The argument row=0, column=0 designates the position in a matrix. Again, padx and pady are blank space paddings.

```
##################################################
# BOTTLE FRAME
##################################################
```

The Bottle Frame is the central feature of the program. We begin with the creation of the frame in root (Line 200), and the following statement packs this frame in the root below the **select_frame**. I'll skip down to describe the widget which is key to the program. The object **bottle_tree** is a ttk *treeview* object which is designed to display tabular data. While it has the capacity to display hierarchical[4] elements, we are not using that capability here.

Before packing **bottle_tree** into the **bottle_frame**, there is one more step we must take. We do not know the size of the tabular display created by bottle_tree. When we select category='all', for example, we will have a large group of bottles to display. On the other hand, selecting category='rose' we will display a much smaller number of bottles. Therefore, we want to make **bottle_frame** contain a scroll bar.

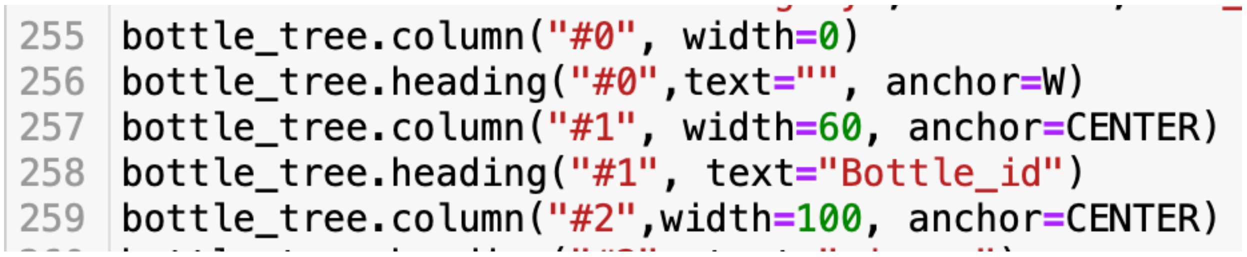
```
bottle_scroll=Scrollbar(bottle_frame)
bottle_scroll.config(command=bottle_tree.yview)
bottle_scroll.pack(side=RIGHT,fill=Y)
```

This code creates a new object bottle_scroll using the object constructor: `Scrollbar(bottle_frame)`. The new object is embedded in **bottle_frame**. In the second statement, we describe the functionality of bottle_scroll. It is to control the vertical scrolling of **bottle_tree**. We pack **bottle_scroll** on the right hand side of the **bottle_frame** and have it fill the height of the field.

[4] Think of an entry in a table which has sub-entries. The children of this entry can be displayed by clicking on a link attached to the parent.

We are now ready to define **bottle_tree**.

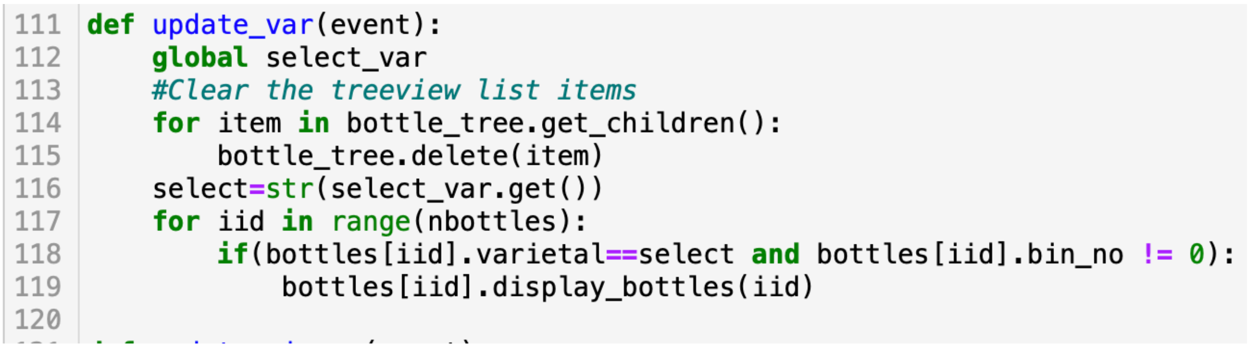
Lines 253-274 define the columns of the tabular data:



```
255 bottle_tree.column("#0", width=0)
256 bottle_tree.heading("#0",text="", anchor=W)
257 bottle_tree.column("#1", width=60, anchor=CENTER)
258 bottle_tree.heading("#1", text="Bottle_id")
259 bottle_tree.column("#2",width=100, anchor=CENTER)
```

Recall that in Python all indexing begins with 0. We are not using column 0 as this would be used for an hierarchal tree structure. Column 1 has a width of 60 characters and the text that will be placed here will be anchored in the CENTER of the column. The next line describes the heading for column 1.

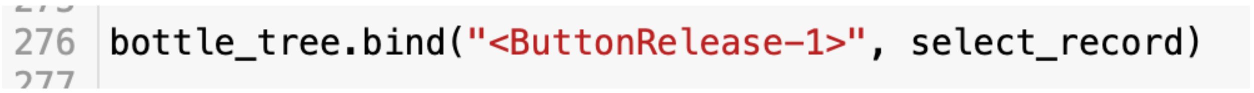
Now we can return to the functions that head this section. There are four functions that are associated with selecting a category, varietal, winery, or rack. I'll describe the varietal function **update_var(event)**.



```
111 def update_var(event):
112     global select_var
113     #Clear the treeview list items
114     for item in bottle_tree.get_children():
115         bottle_tree.delete(item)
116     select=str(select_var.get())
117     for iid in range(nbottles):
118         if(bottles[iid].varietal==select and bottles[iid].bin_no != 0):
119             bottles[iid].display_bottles(iid)
120
```

I'll start in the middle of the code. `select=str(select_var.get())` `Recall that` the variable select_var is a tkinter string variable which has definition within the root window. The method .get() retrieves that value of that variable. The for loop 117-120 sorts thru all the bottles to find the ones which satisfy the boolean expression varietal==select and bin_no !=0. The second component is there to exclude from our list any bottle in bin0, which are bottles we consumed. When we find a bottle matching the requirement we apply the display_bottles method defined for the class Bottles. This appears in the code (lines 35-45).

Now the really critical issues— what is the argument event in line 111? An event can be a key press, a mouse event, a button press, etc. The solution is to bind the mouse action of selecting to a particular record in bottle_tree.



```
276 bottle_tree.bind("<ButtonRelease-1>", select_record)
```

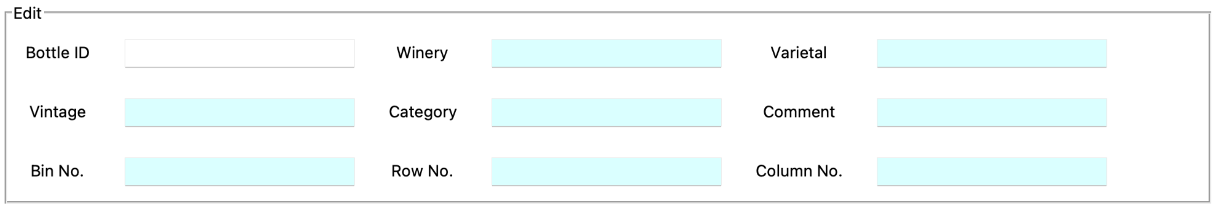
Here, “<ButtonRelease-1>"defines a right mouse click, activation when the mouse button is released. This issues a command to execute the function select_record(event). This function is defined (lines 204-228). When you select a record in bottle_tree the attributes of the bottle appear in entry boxes in the next frame—the edit frame. The first thing that the program does is to delete anything that is currently in the nine entry boxes (each box associated with an attribute of the bottle object.

```
217     r_select=bottle_tree.focus()
218     r_vals=bottle_tree.item(r_select,'values')
219     # Enter r_vals into entry boxes
220     bot_id_entry.insert(0,r_vals[0])
221     bot_winery_entry.insert(0,r_vals[1])
```

There is a method named .focus() defined for treeview objects which retrieves an item in bottle_tree (the row selected). Line 218 assigns the column values to the list r_vals. Finally, we can put these values into the appropriate entry box (starting at position 0) by referring to an element, for example r_vals[0]; this is the position that holds the value of bottle_id, which in the final frame we define as bot_id for that entry box.

```
276 ###########################################################
277 # EDIT FRAME
278 ###########################################################
```

Lines 276-325



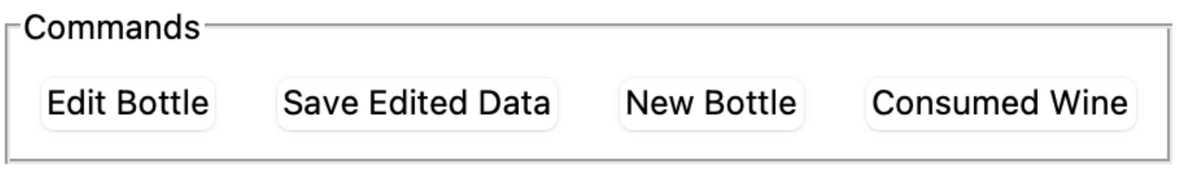
The edit frame is used to either edit attributes of bottles, or to enter new bottles to the wine library.

The edit frame is shown above. It displays the nine attributes of a bottle object. The name of the attribute is written in a tkinter object called a label, and the area to the right of each label (filled with light blue) is an entry box. When a record is selected from bottle_tree the values a copied here. The user will also use these boxes to change values (except Bottle_id, which cannot be changed).

After creating and packing **edit_frame**, we define the labels and entry boxes. An example follows. The code is self explanitory.

```python
bot_id_label=Label(edit_frame, text="Bottle ID")
bot_id_label.grid(row=0, column=0, padx=10, pady=10)
bot_id_entry=Entry(edit_frame)
bot_id_entry.grid(row=0, column=1, padx=10, pady=10)
```

```python
####################################################
# COMMAND FRAME AND COMMAND BUTTONS
####################################################
```



The command frame is used to update the data displayed in bottle_tree, the bottle attributes, and the csv file.

The final major division of the program is the (command) **com_frame**. In this frame we define four button objects to accomplish the following actions:

- Update the bottle object (not just the display)
- Save the updates to the bottle attributes to the csv file.
- Enter a new bottle of wine
- Mark a bottle as consumed

As an example of one of the four command frame buittons consider the second operation *Save the updates to the bottle*.

```python
update_button=Button(com_frame,text="Edit Bottle", command=edit_bottle)
update_button.grid(row=0,column=0, padx=10, pady=10)
```

First, we create **update_button** in this new frame **com_frame** and give it text = ‘Edit Bottle’. When the button is clicked, we issue the command edit_bottle. The function (lines 335-349) retrieves the edited values in the entry boxes and assigns them to new variables, using the method .get().

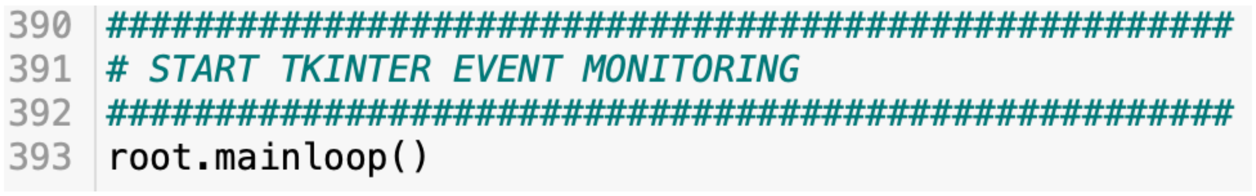
```
bot_id_entry.get()
bot_winery_entry.get()
…
…
```

Finally, a single command

```
bottles[bid].update_bottle(bid)
```

applies the update_bottle method we defined in the definition of the class Bottles, to update all bottle object attributes.

The program ends with launching the method .mainloop applied to root.



```
####################################################
# START TKINTER EVENT MONITORING
####################################################
root.mainloop()
```