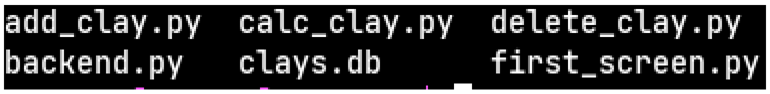
Criterion C: Development

The final product is a Python program split into multiple files. It calculates the shrinkage of a pot by taking in a dimension and the type of clay, and then outputting the size of the un-fired pot to the user. There is a database filled with common clays that the client uses, containing the shrinkage rate and absorption rate. The shrinkage rate was taken from the vendor's website at the temperature my client typically uses to fire her clays, and combined with the drying shrinkage to create a unified shrinkage value. This is in percentage form, so the program converts it to a decimal by dividing it by 100. It then gets the reciprocal of the decimal by subtracting it from one, and finally divides the dimension provided by that number. This gets the original dimension, which the potter would then use to throw a pot that shrinks to the desired size.

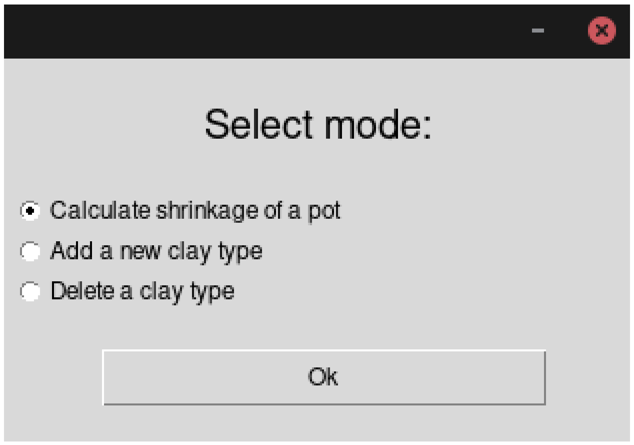
Files



```
add_clay.py   calc_clay.py   delete_clay.py
backend.py    clays.db       first_screen.py
```

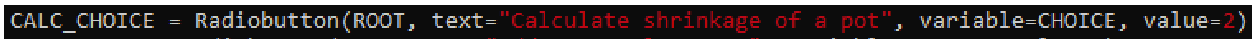
Each one of these files corresponds to a window, excluding the .db file and the backend. I split up the files to help keep the code as clean as possible.

Main window

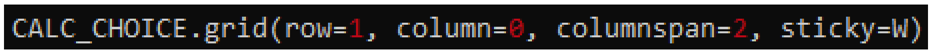


All three of these functions are implemented, and more could be added in the future due to the modularity of Tkinter's in-build selection button called 'Radiobuttons' shown below.

Creating and positioning a Tkinter "Radiobutton"



```python
CALC_CHOICE = Radiobutton(ROOT, text="Calculate shrinkage of a pot", variable=CHOICE, value=2)
```

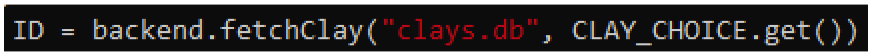


```python
CALC_CHOICE.grid(row=1, column=0, columnspan=2, sticky=W)
```

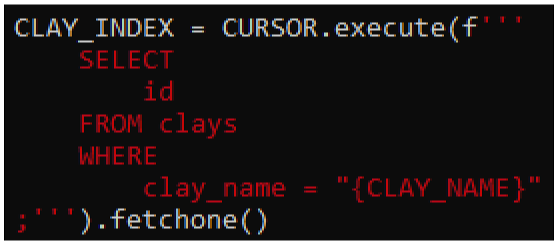
Here, I'm creating a Radiobutton and assigning it to a variable. Then, I modify its parameters to get it to where I want it to be.

Techniques used (with examples):

- Searching for specified data in a file
    - I search for the ID of a row of clay information in SQLite

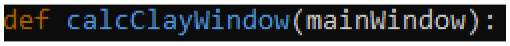


```python
ID = backend.fetchClay("clays.db", CLAY_CHOICE.get())
```

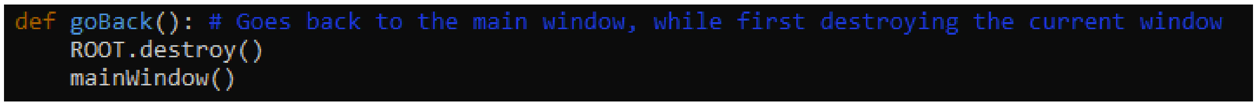


```
CLAY_INDEX = CURSOR.execute(f'''
    SELECT
        id
    FROM clays
    WHERE
        clay_name = "{CLAY_NAME}"
;''').fetchone()
```

- Recursion
    - When I call another window's function from the mainWindow function, I pass mainWindow into it so that it can call it again with the return button

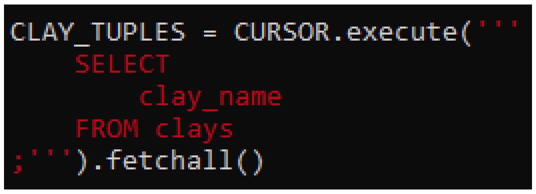


```
def calcClayWindow(mainWindow):
```



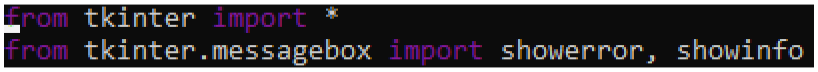
```
def goBack(): # Goes back to the main window, while first destroying the current window
    ROOT.destroy()
    mainWindow()
```

- Parsing a text file or other data stream
    - I input data and read from a database

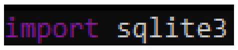


```
CLAY_TUPLES = CURSOR.execute('''
    SELECT
        clay_name
    FROM clays
;''').fetchall()
```

- Use of additional libraries
    - I made a GUI program with Tkinter, and I also used SQLite

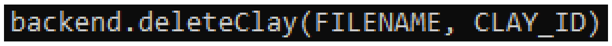


```
from tkinter import *
from tkinter.messagebox import showerror, showinfo
```

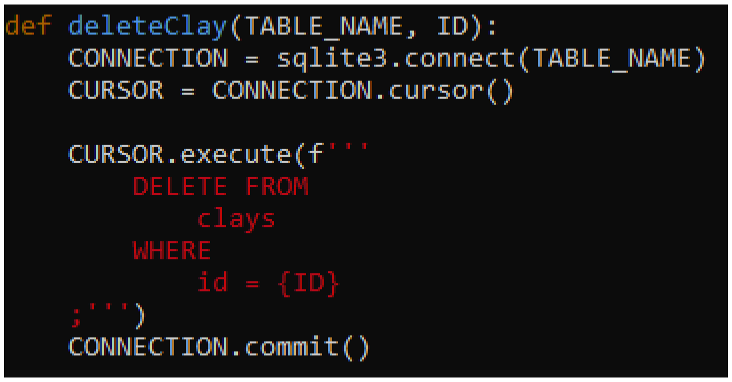


```
import sqlite3
```

- Deleting data from a sequential file without reading the entire file into RAM
    - I delete data from a database

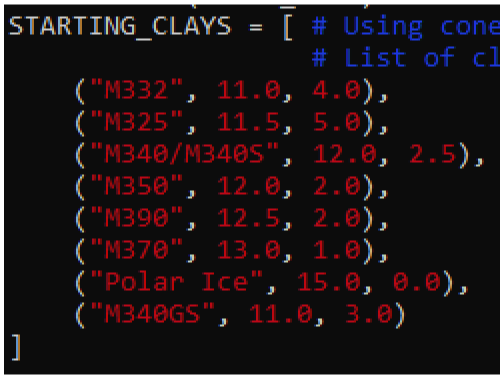


```
backend.deleteClay(FILENAME, CLAY_ID)
```



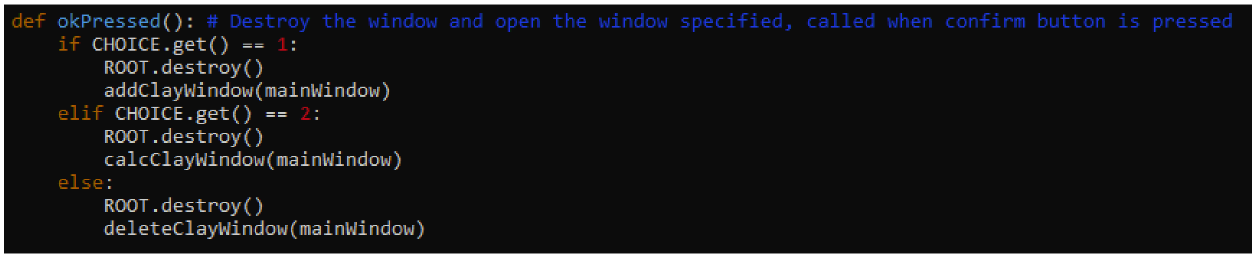
```
def deleteClay(TABLE_NAME, ID):
    CONNECTION = sqlite3.connect(TABLE_NAME)
    CURSOR = CONNECTION.cursor()

    CURSOR.execute(f'''
        DELETE FROM
            clays
        WHERE
            id = {ID}
    ;''')
    CONNECTION.commit()
```

- Arrays of two or more dimensions
    - This is how I store the clay that comes pre-installed with the program



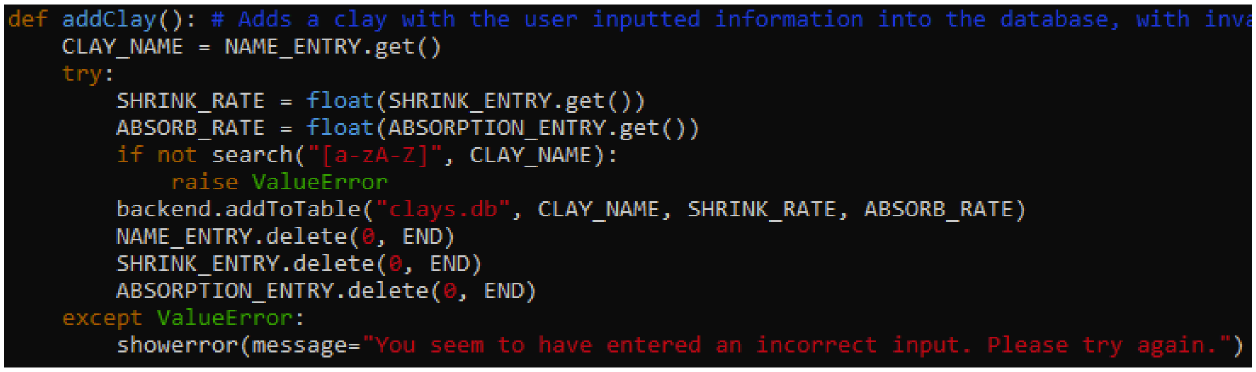
```
STARTING_CLAYS = [ # Using cone
                   # List of cl
    ("M332", 11.0, 4.0),
    ("M325", 11.5, 5.0),
    ("M340/M340S", 12.0, 2.5),
    ("M350", 12.0, 2.0),
    ("M390", 12.5, 2.0),
    ("M370", 13.0, 1.0),
    ("Polar Ice", 15.0, 0.0),
    ("M340GS", 11.0, 3.0)
]
```

- Simple selection
    - I use simple selection to determine which window to open in the first screen



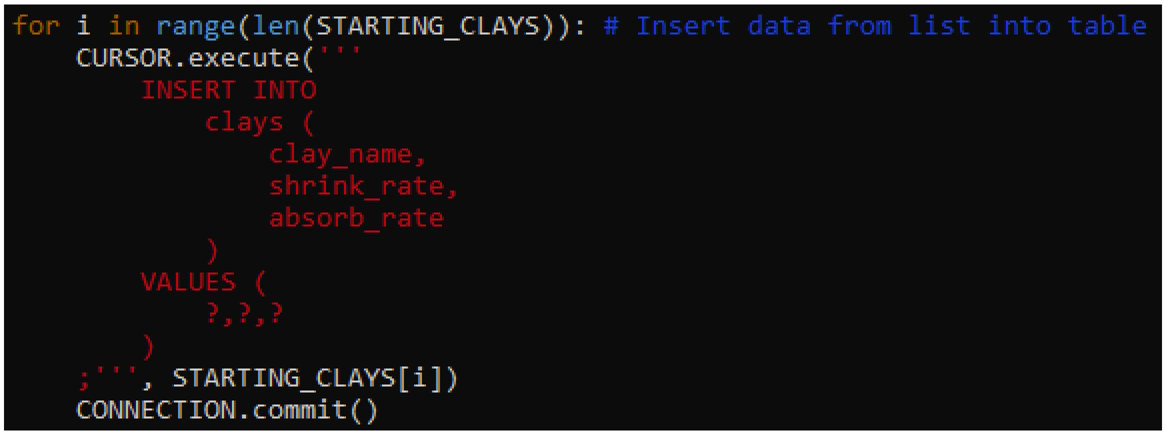
```
def okPressed(): # Destroy the window and open the window specified, called when confirm button is pressed
    if CHOICE.get() == 1:
        ROOT.destroy()
        addClayWindow(mainWindow)
    elif CHOICE.get() == 2:
        ROOT.destroy()
        calcClayWindow(mainWindow)
    else:
        ROOT.destroy()
        deleteClayWindow(mainWindow)
```

- Complex selection
    - I use complex selection with an if statement nested inside of a try except statement in the adding clay window. The try except checks if the value can be converted, and the if statement checks if the value is blank.



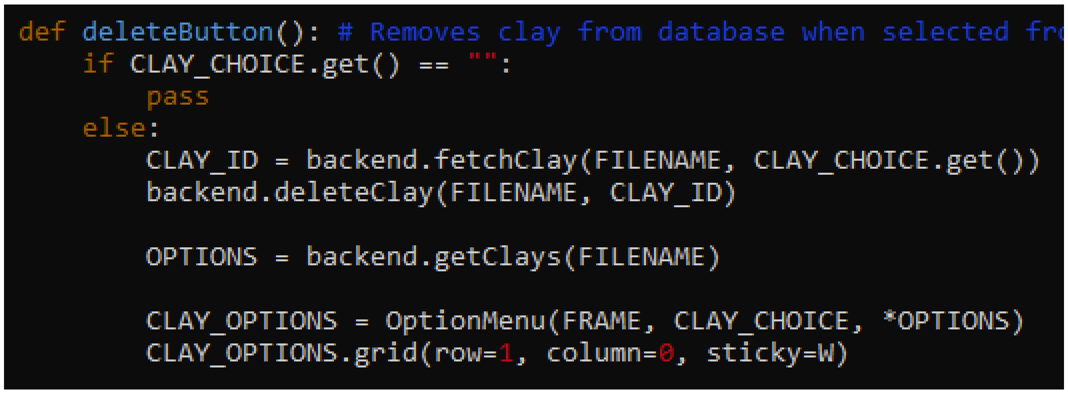
```
def addClay(): # Adds a clay with the user inputted information into the database, with inva
    CLAY_NAME = NAME_ENTRY.get()
    try:
        SHRINK_RATE = float(SHRINK_ENTRY.get())
        ABSORB_RATE = float(ABSORPTION_ENTRY.get())
        if not search("[a-zA-Z]", CLAY_NAME):
            raise ValueError
        backend.addToTable("clays.db", CLAY_NAME, SHRINK_RATE, ABSORB_RATE)
        NAME_ENTRY.delete(0, END)
        SHRINK_ENTRY.delete(0, END)
        ABSORPTION_ENTRY.delete(0, END)
    except ValueError:
        showerror(message="You seem to have entered an incorrect input. Please try again.")
```

- Loops
    - In addition to Tkinter having a mainloop() function for its windows, I use a for loop to put data into the table when the program starts for the first time



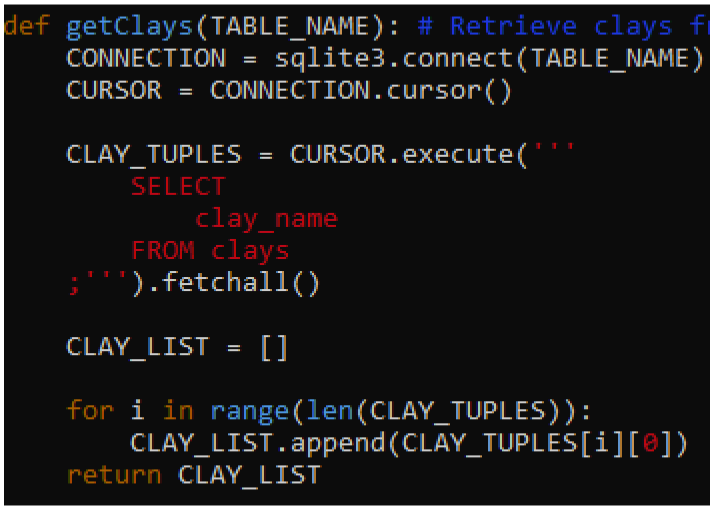
```
for i in range(len(STARTING_CLAYS)): # Insert data from list into table
    CURSOR.execute('''
        INSERT INTO
            clays (
                clay_name,
                shrink_rate,
                absorb_rate
            )
        VALUES (
            ?,?,?
        )
    ;''', STARTING_CLAYS[i])
    CONNECTION.commit()
```

- User-defined methods
  - I use many user defined methods for UI elements such as button presses



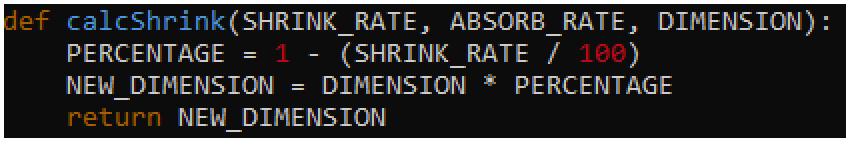
```
def deleteButton(): # Removes clay from database when selected fr
    if CLAY_CHOICE.get() == "":
        pass
    else:
        CLAY_ID = backend.fetchClay(FILENAME, CLAY_CHOICE.get())
        backend.deleteClay(FILENAME, CLAY_ID)

        OPTIONS = backend.getClays(FILENAME)

        CLAY_OPTIONS = OptionMenu(FRAME, CLAY_CHOICE, *OPTIONS)
        CLAY_OPTIONS.grid(row=1, column=0, sticky=W)
```

- User-defined methods with parameters
  - My SQLite methods have parameters for the table name so that it can create a connection.



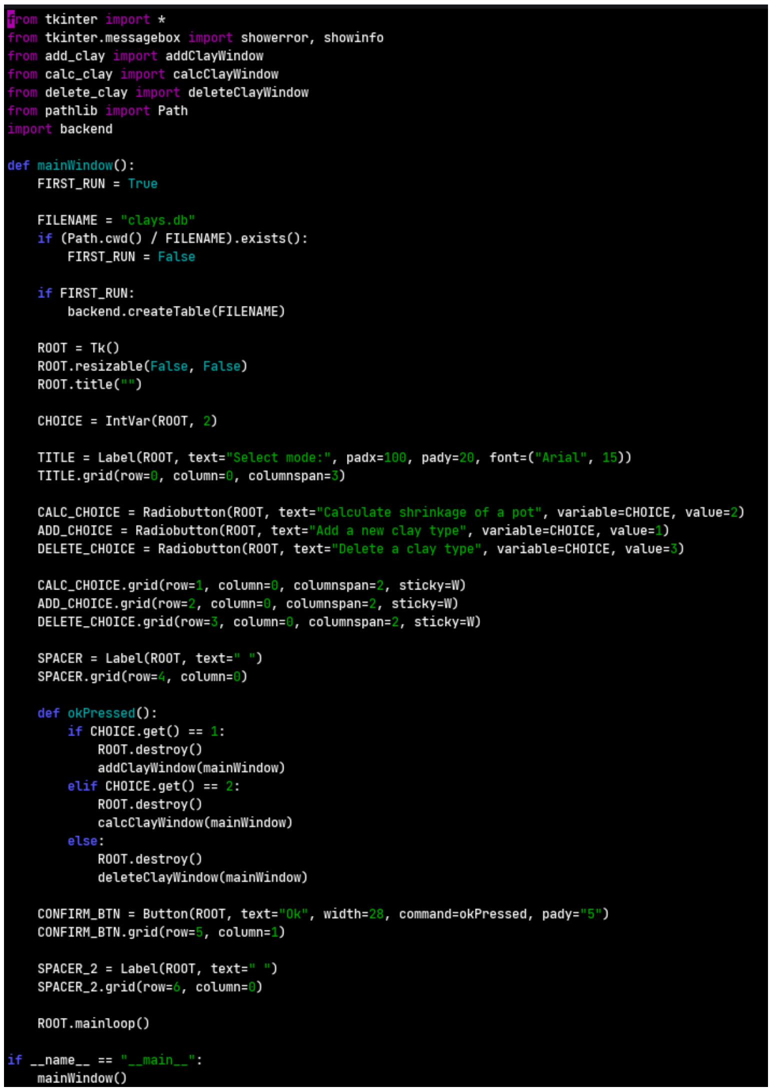
```
def getClays(TABLE_NAME): # Retrieve clays f
    CONNECTION = sqlite3.connect(TABLE_NAME)
    CURSOR = CONNECTION.cursor()

    CLAY_TUPLES = CURSOR.execute('''
        SELECT
            clay_name
        FROM clays
    ;''').fetchall()

    CLAY_LIST = []

    for i in range(len(CLAY_TUPLES)):
        CLAY_LIST.append(CLAY_TUPLES[i][0])
    return CLAY_LIST
```

- User-defined methods with appropriate return values
  - My mathematical methods return the appropriate value after the calculation is complete, such as calcShrink() which returns the dimensions before shrinkage since the user inputting dimensions after shrinkage.



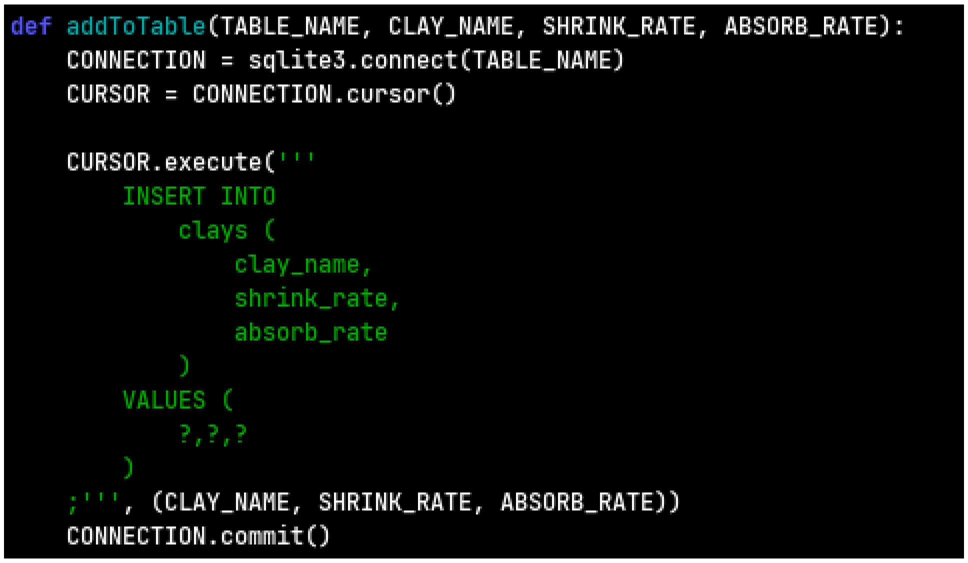
```python
def calcShrink(SHRINK_RATE, ABSORB_RATE, DIMENSION):
    PERCENTAGE = 1 - (SHRINK_RATE / 100)
    NEW_DIMENSION = DIMENSION * PERCENTAGE
    return NEW_DIMENSION
```

first_screen.py



```python
from tkinter import *
from tkinter.messagebox import showerror, showinfo
from add_clay import addClayWindow
from calc_clay import calcClayWindow
from delete_clay import deleteClayWindow
from pathlib import Path
import backend

def mainWindow():
    FIRST_RUN = True

    FILENAME = "clays.db"
    if (Path.cwd() / FILENAME).exists():
        FIRST_RUN = False

    if FIRST_RUN:
        backend.createTable(FILENAME)

    ROOT = Tk()
    ROOT.resizable(False, False)
    ROOT.title("")

    CHOICE = IntVar(ROOT, 2)

    TITLE = Label(ROOT, text="Select mode:", padx=100, pady=20, font=("Arial", 15))
    TITLE.grid(row=0, column=0, columnspan=3)

    CALC_CHOICE = Radiobutton(ROOT, text="Calculate shrinkage of a pot", variable=CHOICE, value=2)
    ADD_CHOICE = Radiobutton(ROOT, text="Add a new clay type", variable=CHOICE, value=1)
    DELETE_CHOICE = Radiobutton(ROOT, text="Delete a clay type", variable=CHOICE, value=3)

    CALC_CHOICE.grid(row=1, column=0, columnspan=2, sticky=W)
    ADD_CHOICE.grid(row=2, column=0, columnspan=2, sticky=W)
    DELETE_CHOICE.grid(row=3, column=0, columnspan=2, sticky=W)

    SPACER = Label(ROOT, text=" ")
    SPACER.grid(row=4, column=0)

    def okPressed():
        if CHOICE.get() == 1:
            ROOT.destroy()
            addClayWindow(mainWindow)
        elif CHOICE.get() == 2:
            ROOT.destroy()
            calcClayWindow(mainWindow)
        else:
            ROOT.destroy()
            deleteClayWindow(mainWindow)

    CONFIRM_BTN = Button(ROOT, text="Ok", width=28, command=okPressed, pady="5")
    CONFIRM_BTN.grid(row=5, column=1)

    SPACER_2 = Label(ROOT, text=" ")
    SPACER_2.grid(row=6, column=0)

    ROOT.mainloop()

if __name__ == "__main__":
    mainWindow()
```

Since Tkinter buttons can't execute functions with parameters (without using Lambdas), I created functions for each button in my program. This also helps to simplify things because there aren't two buttons using the same function. This window checks which radiobutton you pressed, and calls the correct window's function for your choice. It also passes itself into the window so that you can return to the main window after you're done in the other window.



```
def addToTable(TABLE_NAME, CLAY_NAME, SHRINK_RATE, ABSORB_RATE):
    CONNECTION = sqlite3.connect(TABLE_NAME)
    CURSOR = CONNECTION.cursor()

    CURSOR.execute('''
        INSERT INTO
            clays (
                clay_name,
                shrink_rate,
                absorb_rate
            )
        VALUES (
            ?,?,?
        )
    ;''', (CLAY_NAME, SHRINK_RATE, ABSORB_RATE))
    CONNECTION.commit()
```

This is one of my backend functions, and a great example of my program accessing the SQLite database.

Additionally, there are two other modes: Adding and deleting clay types. When you add a clay type, there are three text input boxes for the name of the clay, the combined shrinkage rate of the clay, and the absorption rate of the clay. The name and shrinkage rate of the program are used as covered above, and the absorption rate is for the future, in case I want to add a weight function (i.e. calculating how much the weight will increase when it absorbs water). This data is saved into the SQLite database when the “Add!” button is clicked. Similarly, the delete function shows the user a drop down menu of clay types and deletes the clay from the database when the button is clicked.

Changes in development:

While I mostly stuck to the flow-chart, visualization and plan, there were a few things I ended up changing. The most notable being the small change I made to the UI that differed from the plan. In the visualization, the main window stays there the whole time, and a popup window dictates what goes in said window. I have since realized that to be impractical. Now, the windows are all their own functions and (other than results or error messages) they are only on the screen one at a time.

Word count: 837