Criterion C: Development

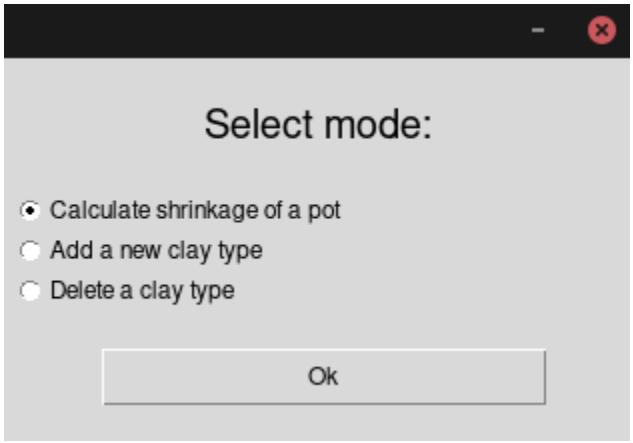
The final product is a Python program, split into multiple files. It calculates the shrinkage of a pot by taking in the diameter and the type of clay, and then outputting the size of the pot before firing to the user. There is a database filled with common clays that the client uses, containing the shrinkage 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. I split up the files to avoid a convoluted codebase.

Main window



All three of these functions are implemented, and more could be added in the future due to the modular nature of Tkinter's Radiobuttons, as seen in the next picture.

```
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.

Techniques used:

- Searching for specified data in a file
    - I search for the ID of a row of clay information in SQLite
- 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
- Parsing a text file or other data stream
    - I input data and read from a database
- Use of additional libraries
    - I made a GUI program with Tkinter, and I also used SQLite
- Deleting data from a sequential file without reading the entire file into RAM
    - I delete data from a database
- Arrays of two or more dimensions
    - This is how I store the clay that comes pre-installed with the program
- Simple selection
    - I use simple selection to determine which window to open in the first screen
- 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.
- 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
- User-defined methods
    - I use many user defined methods for UI elements such as button presses
- User-defined methods with parameters
    - My SQLite methods have parameters for the table name so that it can create a connection.
- 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.

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 addition is the deleting clay function. I had not originally planned for this but it was quite simple to add midway through development and I figured it would be quite helpful in case the user makes a mistake while adding clay or wants to delete a clay that came with the program. Additionally, I made a small change to the UI that differs from the visualization. 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: 634