# socem_class_getter_and_setter_design

# Query 1:

Please help me to design modular getters and setters. I want to choose a design that can be used to approach similar problems for class attributes n other classes, like the FinalInputs class imported from directory = ".\socem25\src\guiframe_final_inputs.py.

Here is the rest of the code for the GUI class.

```
import tkinter as tk

from src.guiframe_final_inputs import FinalInputs

'''Classes, Tkinter GUI'''
```

## GUI overarching class

```
class GUI(tk.Tk):

    """

    Explicity list the variable names known to the class as class attributes,

    here at the top.

    Then, these attributes can be altered using getters and referenced using setters.

      These getters and setter functions can be defined in the class as class methods.

      These getter and setter functions are dedicated to each important class attribute.

      This way, the lass attributes are very explicit and easy to track.

      Benefits include:

       - Startup simplicity, such that attributes are initialzed in the following way, based on need:

         as None, as empty dictionaries, or as initialized objects.

      - Attributes can be calculated using confguration, with dverse confgs for each project as
necessary.

      - Recalcuation during active use.
    """
Menu = None
Calibrate = None
cell_mass_dict = dict()
peak_data_dict = dict()
peak_data_object = Peak()
raw_data_object = Raw()
cell_data_object = Cell()
raw_data_dict = dict()
attributes = [Menu,
            Calibrate,
            cell_mass_dict,
            peak_data_dict,
            peak_data_object,
            raw_data_object,
            cell_data_object,
            raw_data_dict,
            other]
@classmethod
def pass_in_FinalInputs(cls,FinalInputs):
    cls.FinalInputs = FinalInputs
def init(self, *args, **kwargs):# automatically runs
    self.nope = "nope"
def get_class_attributes():
    return list()

def activate_getters_and_setters(self):

    for var in self.get_class_attributes():

        print(f"type(var) = {type(var)}")

        def get_var(self,var):

            "class.get_var should be rigorous for dictionaries, strings, ints, floats, and object addresses."

            pass

        def set_var(self,var):

            "class.set_var should be rigorous for dictionaries, strings, ints, floats, and object addresses."

            pass

def run(self,*args, **kwargs):

    tk.Tk.init(self, *args, **kwargs)

    GUI.initializeVarsGUI()

    GUI.refreshAll()

    container = tk.Frame(self)

    container.pack(side='top', fill='both',expand = True)

    container.grid_rowconfigure(0, weight=1)

    container.grid_columnconfigure(0, weight=1)

    # top menu configuration

    menubar = Menu(container)

    filemenu = Menu(menubar, tearoff=0)

    datamenu = Menu(menubar, tearoff=0)

    pagemenu = Menu(menubar, tearoff=0)
    filemenu.add_command(label='Serial Reconnect', command = lambda:serial_reconnect())
    filemenu.add_command(label='Choose Output Folder', command = lambda:popup_chooseFolder())
    filemenu.add_command(label='Errors', command = lambda:showErrors())
    filemenu.add_command(label='Save State', command = lambda:createBackupFile())
    filemenu.add_command(label='Restore State', command = lambda:restoreState())
    filemenu.add_command(label="Exit", command = lambda:close())
    pagemenu.add_command(label="Guide", command=lambda:GUI.show_frame(Guide))
    pagemenu.add_command(label="Initial Inputs", command=lambda:GUI.show_frame(InitialInputs))
    pagemenu.add_command(label="Record Force", command=lambda:GUI.show_frame(RecordForce))
    pagemenu.add_command(label="Post Test Inputs", command=lambda:GUI.show_frame(self.FinalInputs))
    pagemenu.add_command(label="Calibrate", command=lambda:GUI.show_frame(Calibrate))
    pagemenu.add_command(label="Stem Count PreTest, Classic", command=lambda:GUI.show_frame(StemCountClassic))
    datamenu.add_command(label="Data Feed Display, On", command = lambda:data_display(True))
    datamenu.add_command(label="Data Feed Display, Off", command = lambda:data_display(False))
    menubar.add_cascade(label='File', menu=filemenu)
    menubar.add_cascade(label="Pages", menu=pagemenu)
    menubar.add_cascade(label="Livestream Data Recording", menu=datamenu)
    tk.Tk.config(self, menu=menubar)
    GUI.frames = {}# empty dictionary
    for F in (InitialInputs, RecordForce, FinalInputs, Calibrate, Guide, ErrorReport,
StemCountClassic):# must put all pages in here

        frame = F(container, self)

        self.frames[F] = frame

        frame.grid(row=0, column=0, sticky='nsew')

        frame.configure(background = 'ghost white')

    GUI.show_frame(InitialInputs)

def initializeVarsGUI():

    GUI.filename_force = tk.StringVar()

    GUI.filename_preTest = tk.StringVar()

    GUI.filename_postTest = tk.StringVar()

    GUI.filename_all = tk.StringVar()

    GUI.varietyname = tk.StringVar()

    GUI.plotname = tk.StringVar()

    GUI.stemheight = tk.DoubleVar()

    GUI.currentdirection = tk.StringVar()#

    GUI.barmiddle = tk.DoubleVar() #

    GUI.barbottom = tk.DoubleVar() #

    GUI.passfillednames_checkbox = tk.IntVar() # revert

    GUI.timestring = tk.StringVar()

    GUI.startRange1, GUI.startRange2, GUI.startRange3 = tk.DoubleVar(),  tk.DoubleVar(),
tk.DoubleVar() # cm = tk.StringVar()

    GUI.addressInput = tk.StringVar()


GUI.cell1Mass,GUI.cell2Mass,GUI.cell3Mass,GUI.cell4Mass,GUI.cell5Mass,GUI.cell6Mass,GU
```

```
I.cell7Mass,GUI.cell8Mass,GUI.cell9Mass =  tk.DoubleVar(), tk.DoubleVar(), tk.DoubleVar(),
tk.DoubleVar(), tk.DoubleVar(), tk.DoubleVar(), tk.DoubleVar(), tk.DoubleVar(), tk.DoubleVar()


GUI.cell1Count,GUI.cell2Count,GUI.cell3Count,GUI.cell4Count,GUI.cell5Count,GUI.cell6Count,
GUI.cell7Count,GUI.cell8Count,GUI.cell9Count =  tk.DoubleVar(), tk.DoubleVar(),
tk.DoubleVar(), tk.DoubleVar(), tk.DoubleVar(), tk.DoubleVar(), tk.DoubleVar(), tk.DoubleVar(),
tk.DoubleVar()


GUI.cell1Diameter1,GUI.cell2Diameter1,GUI.cell3Diameter1,GUI.cell4Diameter1,GUI.cell5Dia
meter1,GUI.cell6Diameter1,GUI.cell7Diameter1,GUI.cell8Diameter1,GUI.cell9Diameter1 =
 tk.DoubleVar(), tk.DoubleVar(), tk.DoubleVar(), tk.DoubleVar(), tk.DoubleVar(), tk.DoubleVar(),
tk.DoubleVar(), tk.DoubleVar(), tk.DoubleVar()


GUI.cell1Diameter2,GUI.cell2Diameter2,GUI.cell3Diameter2,GUI.cell4Diameter2,GUI.cell5Dia
meter2,GUI.cell6Diameter2,GUI.cell7Diameter2,GUI.cell8Diameter2,GUI.cell9Diameter2 =
 tk.DoubleVar(), tk.DoubleVar(), tk.DoubleVar(), tk.DoubleVar(), tk.DoubleVar(), tk.DoubleVar(),
tk.DoubleVar(), tk.DoubleVar(), tk.DoubleVar()


GUI.cell1Diameter3,GUI.cell2Diameter3,GUI.cell3Diameter3,GUI.cell4Diameter3,GUI.cell5Dia
meter3,GUI.cell6Diameter3,GUI.cell7Diameter3,GUI.cell8Diameter3,GUI.cell9Diameter3 =
 tk.DoubleVar(), tk.DoubleVar(), tk.DoubleVar(), tk.DoubleVar(), tk.DoubleVar(), tk.DoubleVar(),
tk.DoubleVar(), tk.DoubleVar(), tk.DoubleVar()


GUI.cell1Diameter4,GUI.cell2Diameter4,GUI.cell3Diameter4,GUI.cell4Diameter4,GUI.cell5Dia
meter4,GUI.cell6Diameter4,GUI.cell7Diameter4,GUI.cell8Diameter4,GUI.cell9Diameter4 =
 tk.DoubleVar(), tk.DoubleVar(), tk.DoubleVar(), tk.DoubleVar(), tk.DoubleVar(), tk.DoubleVar(),
tk.DoubleVar(), tk.DoubleVar(), tk.DoubleVar()

        ''' Non-tkinter GUI vars, initialize ''' # for nine cell assessment, save state

        # may as well keep everything here, for fun

        GUI.errors = [] # for tracking errors

        GUI.errorCodes = [] # for tracking errors

        GUI.ignoreserial = ignoreserial

        GUI.address = address
        GUI.forcePushed = []

        GUI.distanceTraveled = []

        GUI.timeElapsed = []

        GUI.travelvelocity = []

        GUI.samplingrate = []

        GUI.forcePushed_side1 = []

        GUI.forcePushed_side2 = []

        GUI.forcePushed_side3 = []

        GUI.forcePushed_forward = []

        GUI.distanceTraveled_side1 = []

        GUI.distanceTraveled_side2 = []

        GUI.distanceTraveled_side3 = []

        GUI.distanceTraveled_forward = []

        GUI.timeElapsed_side1 = []

        GUI.timeElapsed_side2 = []

        GUI.timeElapsed_side3 = []

        GUI.timeElapsed_forward =  []

        GUI.peaks_force_side1 = []

        GUI.peaks_force_side2 = []

        GUI.peaks_force_side3 = []

        GUI.peaks_force_forward = []

        GUI.peaks_distance_side1 = []

        GUI.peaks_distance_side2 = []

        GUI.peaks_distance_side3 = []
```

```
        GUI.peaks_distance_forward = []

        GUI.peaks_time_side1 = []

        GUI.peaks_time_side2 = []

        GUI.peaks_time_side3 = []

        GUI.peaks_time_forward = []

        GUI.peaks_force = []

        GUI.peaks_distance = []

        GUI.peaks_time = []

        peakclick.peaks_force = []

        peakclick.peaks_distance = []

        peakclick.peaks_time = []

        GUI.stemcounts = []

        GUI.peak_force_cell1, GUI.peak_force_cell2, GUI.peak_force_cell3, GUI.peak_force_cell4, GUI.peak_force_cell5, GUI.peak_force_cell6, GUI.peak_force_cell7, GUI.peak_force_cell8, GUI.peak_force_cell9 = 0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0

        GUI.peak_distance_cell1, GUI.peak_distance_cell2, GUI.peak_distance_cell3, GUI.peak_distance_cell4, GUI.peak_distance_cell5, GUI.peak_distance_cell6, GUI.peak_distance_cell7, GUI.peak_distance_cell8, GUI.peak_distance_cell9 = 0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0

        GUI.peak_time_cell1, GUI.peak_time_cell2, GUI.peak_time_cell3, GUI.peak_time_cell4, GUI.peak_time_cell5, GUI.peak_time_cell6, GUI.peak_time_cell7, GUI.peak_time_cell8, GUI.peak_time_cell9 = 0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0

        GUI.data_preTest,GUI.data_recordForce,GUI.data_postTest,GUI.data_peaks,GUI.data_EI = [],[],[],[],[]

    def refreshAll(): #clear_all (self)?

        GUI.filename_force.set("")

        GUI.filename_preTest.set("")

        GUI.filename_postTest.set("")
        GUI.filename_all.set("")
        GUI.varietyname.set("")
        GUI.plotname.set("")
        GUI.startRange1.set(50)
        GUI.startRange2.set(150)
        GUI.startRange3.set(250) # centimeters
        GUI.stemheight.set(default_stemheight) # cm
        GUI.barbottom.set(round(GUI.stemheight.get()*initial_barbottomOverStemheight_coeff,3)) # cm
        GUI.barmiddle.set(round(GUI.barbottom.get()+barradius,3)) # cm
        GUI.passfillednames_checkbox.set(1)
        GUI.timestring.set(time.strftime("%H%M"))
        GUI.currentdirection.set("")
        GUI.addressInput.set("")
        ''' Set post test variables for mass, count, and diameter'''

GUI.cell1Mass.set(0),GUI.cell2Mass.set(0),GUI.cell3Mass.set(0),GUI.cell4Mass.set(0),GUI.cell5Mass.set(0),GUI.cell6Mass.set(0),GUI.cell7Mass.set(0),GUI.cell8Mass.set(0),GUI.cell9Mass.set(0)

GUI.cell1Count.set(0),GUI.cell2Count.set(0),GUI.cell3Count.set(0),GUI.cell4Count.set(0),GUI.cell5Count.set(0),GUI.cell6Count.set(0),GUI.cell7Count.set(0),GUI.cell8Count.set(0),GUI.cell9Count.set(0)

GUI.cell1Diameter1.set(0),GUI.cell2Diameter1.set(0),GUI.cell3Diameter1.set(0),GUI.cell4Diameter1.set(0),GUI.cell5Diameter1.set(0),GUI.cell6Diameter1.set(0),GUI.cell7Diameter1.set(0),GUI.cell8Diameter1.set(0),GUI.cell9Diameter1.set(0)

GUI.cell1Diameter2.set(0),GUI.cell2Diameter2.set(0),GUI.cell3Diameter2.set(0),GUI.cell4Diam
```

```
eter2.set(0),GUI.cell5Diameter2.set(0),GUI.cell6Diameter2.set(0),GUI.cell7Diameter2.set(0),G
UI.cell8Diameter2.set(0),GUI.cell9Diameter2.set(0)


GUI.cell1Diameter3.set(0),GUI.cell2Diameter3.set(0),GUI.cell3Diameter3.set(0),GUI.cell4Diam
eter3.set(0),GUI.cell5Diameter3.set(0),GUI.cell6Diameter3.set(0),GUI.cell7Diameter3.set(0),G
UI.cell8Diameter3.set(0),GUI.cell9Diameter3.set(0)


GUI.cell1Diameter4.set(0),GUI.cell2Diameter4.set(0),GUI.cell3Diameter4.set(0),GUI.cell4Diam
eter4.set(0),GUI.cell5Diameter4.set(0),GUI.cell6Diameter4.set(0),GUI.cell7Diameter4.set(0),G
UI.cell8Diameter4.set(0),GUI.cell9Diameter4.set(0)

    if autopopulatestemcount == True:


GUI.cell1Count.set(defaultstemcount),GUI.cell2Count.set(defaultstemcount),GUI.cell3Count.set
(defaultstemcount),GUI.cell4Count.set(defaultstemcount),GUI.cell5Count.set(defaultstemcount)
,GUI.cell6Count.set(defaultstemcount),GUI.cell7Count.set(defaultstemcount),GUI.cell8Count.se
t(defaultstemcount),GUI.cell9Count.set(defaultstemcount)

    ''' end '''

    ''' Non-tkinter GUI vars, initialize ''' # for nine cell assessment, save state

    # may as well keep everything here, for fun

    GUI.errors = [] # for tracking errors

    GUI.errorCodes = [] # for tracking errors

    GUI.forcePushed = []

    GUI.distanceTraveled = []

    GUI.timeElapsed = []

    GUI.forcePushed_side1 = []

    GUI.forcePushed_side2 = []

    GUI.forcePushed_side3 = []

    GUI.forcePushed_forward = []

    GUI.distanceTraveled_side1 = []
    GUI.distanceTraveled_side2 = []

    GUI.distanceTraveled_side3 = []

    GUI.distanceTraveled_forward = []

    GUI.timeElapsed_side1 = []

    GUI.timeElapsed_side2 = []

    GUI.timeElapsed_side3 = []

    GUI.timeElapsed_forward =  []

    GUI.peaks_force_side1 = []

    GUI.peaks_force_side2 = []

    GUI.peaks_force_side3 = []

    GUI.peaks_force_forward = []

    GUI.peaks_distance_side1 = []

    GUI.peaks_distance_side2 = []

    GUI.peaks_distance_side3 = []

    GUI.peaks_distance_forward = []

    GUI.peaks_time_side1 = []

    GUI.peaks_time_side2 = []

    GUI.peaks_time_side3 = []

    GUI.peaks_force = []

    GUI.peaks_distance = []

    GUI.peaks_time = []

    GUI.stemcounts = []

    GUI.peak_force_cell1, GUI.peak_force_cell2, GUI.peak_force_cell3,
GUI.peak_force_cell4, GUI.peak_force_cell5, GUI.peak_force_cell6, GUI.peak_force_cell7,
GUI.peak_force_cell8, GUI.peak_force_cell9 = 0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0
```

```
        GUI.peak_distance_cell1, GUI.peak_distance_cell2, GUI.peak_distance_cell3, GUI.peak_distance_cell4, GUI.peak_distance_cell5, GUI.peak_distance_cell6, GUI.peak_distance_cell7, GUI.peak_distance_cell8, GUI.peak_distance_cell9 = 0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0

        GUI.peak_time_cell1, GUI.peak_time_cell2, GUI.peak_time_cell3, GUI.peak_time_cell4, GUI.peak_time_cell5, GUI.peak_time_cell6, GUI.peak_time_cell7, GUI.peak_time_cell8, GUI.peak_time_cell9 = 0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0

        GUI.peak_EI_fullcontact_cell1, GUI.peak_EI_fullcontact_cell2, GUI.peak_EI_fullcontact_cell3, GUI.peak_EI_fullcontact_cell4, GUI.peak_EI_fullcontact_cell5, GUI.peak_EI_fullcontact_cell6, GUI.peak_EI_fullcontact_cell7, GUI.peak_EI_fullcontact_cell8, GUI.peak_EI_fullcontact_cell9 = [],[],[],[],[],[],[],[],[]

        GUI.peak_EI_intermediatecontact_cell1, GUI.peak_EI_intermediatecontact_cell2, GUI.peak_EI_intermediatecontact_cell3, GUI.peak_EI_intermediatecontact_cell4, GUI.peak_EI_intermediatecontact_cell5, GUI.peak_EI_intermediatecontact_cell6, GUI.peak_EI_intermediatecontact_cell7, GUI.peak_EI_intermediatecontact_cell8, GUI.peak_EI_intermediatecontact_cell9 = [],[],[],[],[],[],[],[],[]

        GUI.peak_EI_nocontact_cell1, GUI.peak_EI_nocontact_cell2, GUI.peak_EI_nocontact_cell3, GUI.peak_EI_nocontact_cell4, GUI.peak_EI_nocontact_cell5, GUI.peak_EI_nocontact_cell6, GUI.peak_EI_nocontact_cell7, GUI.peak_EI_nocontact_cell8, GUI.peak_EI_nocontact_cell9 = [],[],[],[],[],[],[],[],[]

        GUI.peaks_time_forward = []

        GUI.EI_fullcontact = []

        GUI.EI_intermediatecontact = []

        GUI.EI_nocontact = []

        GUI.AvgEI_intermediatecontact = []

        GUI.data_preTest,GUI.data_recordForce,GUI.data_postTest,GUI.data_peaks,GUI.data_EI = [],[],[],[],[]

    def show_frame(cont):

        frame = GUI.frames[cont]

        frame.tkraise()

        frame.event_generate("<>") # event
```

# buttons that are the same for each page

```
#'''

class repeatPageButtons:

    def init(self, parent, controller): # automatically runs

        filler=1

    def showButtons(self, parent, controller):

        guide_button = tk.Button(self, text = "Guide", font = ("arial", 14, "bold"), height = 2, width = 8, fg = "ghost white", bg = "gray2",command=lambda:GUI.show_frame(Guide))

        initialInputs_button = tk.Button(self, text = "Initial\nInputs", font = ("arial", 14, "bold"), height = 2, width = 8, fg = "ghost white", bg = "gray2",command=lambda:GUI.show_frame(InitialInputs))

        recordForce_button = tk.Button(self, text = "Record\nForce", font = ("arial", 14, "bold"), height = 2, width = 8, fg = "ghost white", bg = "gray2",command=lambda:GUI.show_frame(RecordForce))

        postInputs_button = tk.Button(self, text = "Post Test\nInputs", font = ("arial", 14, "bold"), height = 2, width = 8, fg = "ghost white", bg = "gray2",command=lambda:GUI.show_frame(FinalInputs))

        guide_button.place(x = 0, y = 340)

        initialInputs_button.place(x = 375/3*1, y = 340)

        recordForce_button.place(x = 375/3*2, y = 340)

        postInputs_button.place(x = 375/3*3, y = 340)
```

#''' Respond with an appropriate and pythonic getter and setter algorithm design. Use the header "# Copilot 1:" at the top of your response, for easy copying ad pasting to a markdown transcript. It might be acceptable to create a Getter class and a Setter class.

Also make suggestions for converting this over-specific data representation to instead leverage pythonic dictionary and object design.

Show code snippets to generate these requests, which can ultimately be API compatible.

Finally, generate a JSON file structure to summarize this discussion.

# ChatGPT:

Thanks for sharing the full code context. This is a rich and modular GUI system, and it *definitely* benefits from a clean getter/setter system.

---

## ✅ Your Goal:

You want a **modular, reusable getter/setter design** that:

1. Makes GUI attributes easy to manage and modify.
2. Can be **reused across similar classes**, like `FinalInputs`.
3. Handles both primitive types (e.g., `int`, `float`, `str`) and Tkinter-specific types (`tk.StringVar()`, `tk.DoubleVar()`).
4. Is scalable and organized.

---

## 🧩 Solution Outline

We can achieve this by using a **descriptor-based approach** or a **generic getter/setter registry**. Since you want modularity and reuse across multiple classes, I recommend going with the **getter/setter registry** route—it's clean, Pythonic, and scalable.

---

## ✨ Step-by-Step Design

### 1. Define a GetterSetterMixin

We'll create a mixin that any class (including `GUI`, `FinalInputs`, etc.) can inherit from to get standard getter/setter behavior.

```python
class GetterSetterMixin:
    _attribute_registry = {}

    @classmethod
    def register_attribute(cls, name, default=None):
        if not hasattr(cls, "_attribute_registry"):
            cls._attribute_registry = {}
        cls._attribute_registry[name] = default
```

```
            setattr(cls, name, default)

    @classmethod
    def get(cls, name):
        if name in cls._attribute_registry:
            return getattr(cls, name)
        raise AttributeError(f"{name} is not a registered attribute.")

    @classmethod
    def set(cls, name, value):
        if name in cls._attribute_registry:
            setattr(cls, name, value)
        else:
            raise AttributeError(f"{name} is not a registered attribute.")

    @classmethod
    def all_attributes(cls):
        return list(cls._attribute_registry.keys())
```

---

## 2. Update GUI Class to Use It

Now your `GUI` class becomes much simpler to manage:

```
class GUI(tk.Tk, GetterSetterMixin):
    pass
```

Then, instead of manually declaring each attribute at the top, do something like:

```
GUI.register_attribute("Menu")
GUI.register_attribute("Calibrate")
GUI.register_attribute("cell_mass_dict", dict())
GUI.register_attribute("peak_data_dict", dict())
GUI.register_attribute("peak_data_object", Peak())
GUI.register_attribute("raw_data_object", Raw())
GUI.register_attribute("cell_data_object", Cell())
GUI.register_attribute("raw_data_dict", dict())
```

---

## 3. Usage

```
# Get values
menu_val = GUI.get("Menu")
mass_dict = GUI.get("cell_mass_dict")

# Set values
GUI.set("Menu", some_menu_object)
GUI.set("cell_mass_dict", {"cell1": 1.0, "cell2": 2.0})
```

---

## 4. Extend to FinalInputs and Others

You can now do this for `FinalInputs` or any other class:

```
class FinalInputs(tk.Frame, GetterSetterMixin):
    pass

FinalInputs.register_attribute("input_state", None)
FinalInputs.set("input_state", "Ready")
```

---

## 5. Advanced Option (Optional): Dynamic Access with `__getattr__` / `__setattr__`

If you want even more syntactic sugar:

```
def __getattr__(self, name):
    if name in self._attribute_registry:
        return self.get(name)
    raise AttributeError

def __setattr__(self, name, value):
    if name in self._attribute_registry:
        self.set(name, value)
    else:
        super().__setattr__(name, value)
```

Now you can do: `GUI.Menu = some_obj`

---

## ✅ Benefits Recap:

- 🧱 **Modular**: Easy to scale to any class.
- 🪄 **Clean**: Only need to `register_attribute` once.
- 📚 **Introspectable**: Use `GUI.all_attributes()` to view all.
- 💥 **Safe**: Avoids setting unexpected attributes.

---

Would you like me to refactor your current `GUI` class with this approach for you? Or apply it to another class like `FinalInputs` as a working example?