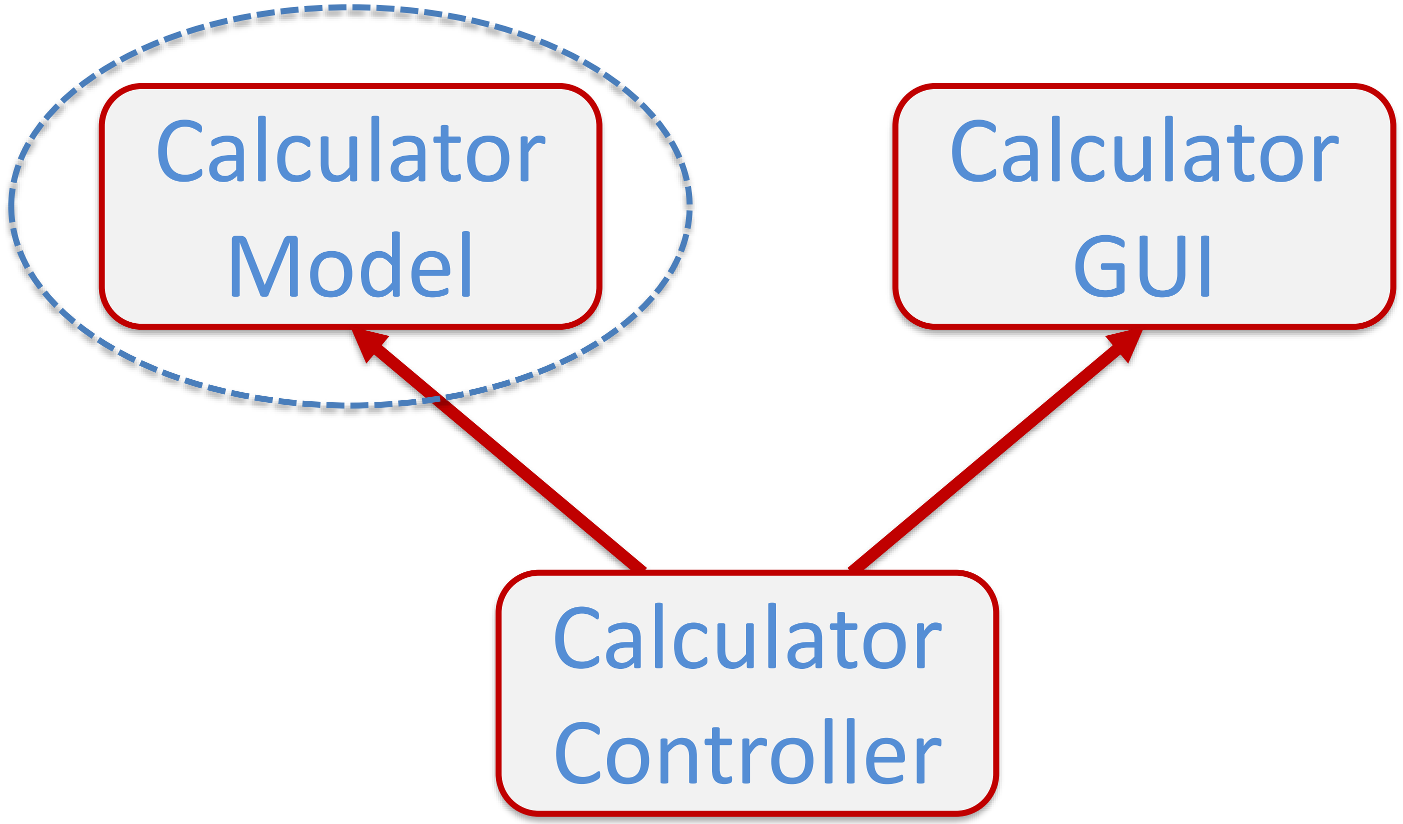
Introduction to Computer Science (intro2cs)

GUI – Calculator: A Complete Example

Design of the program

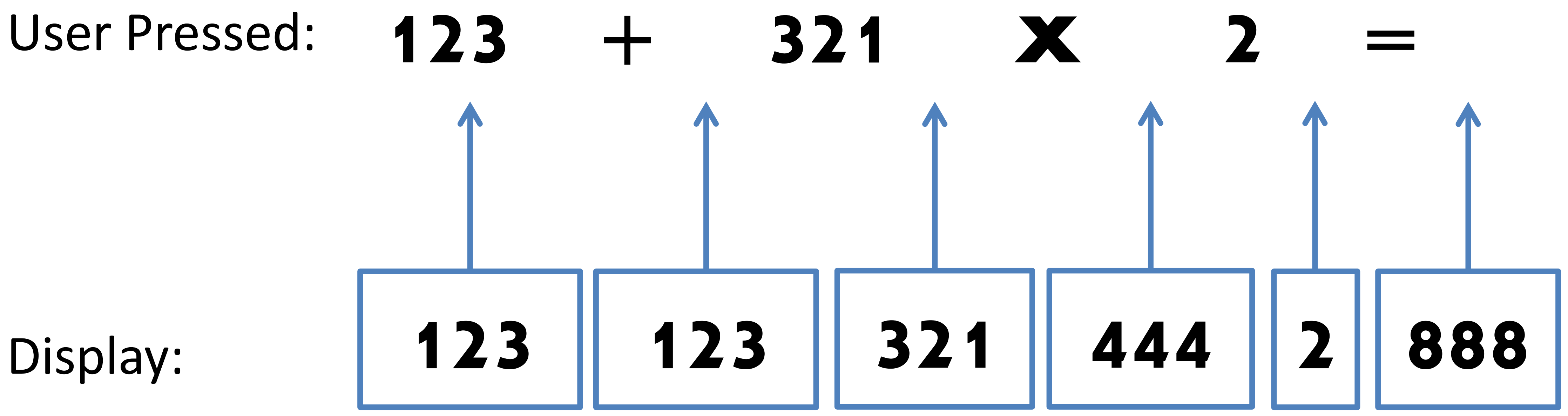


- The model
  - contains the complex logic
- The GUI
  - contains all the visual design

They are completely independent of one another

- The controller
  - is lightweight
  - connects them together

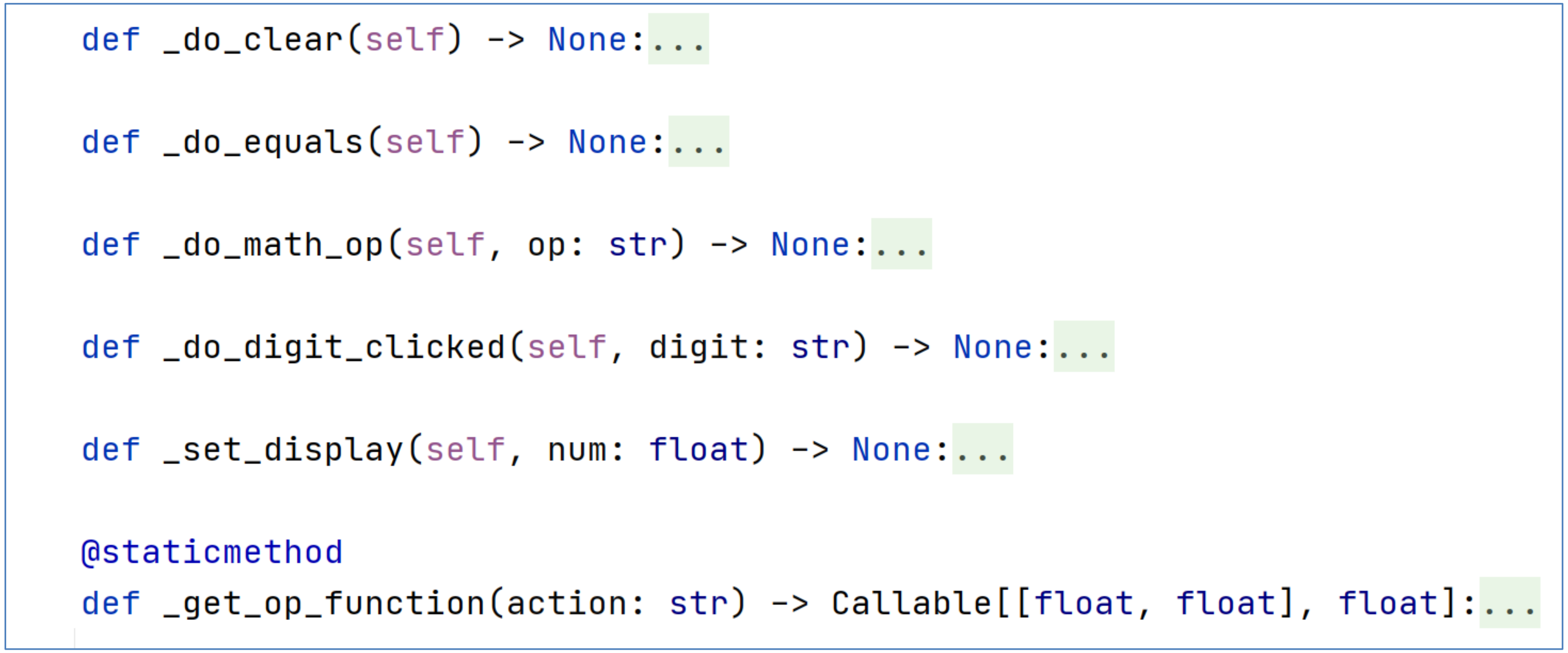
What to do when a user presses an operator?



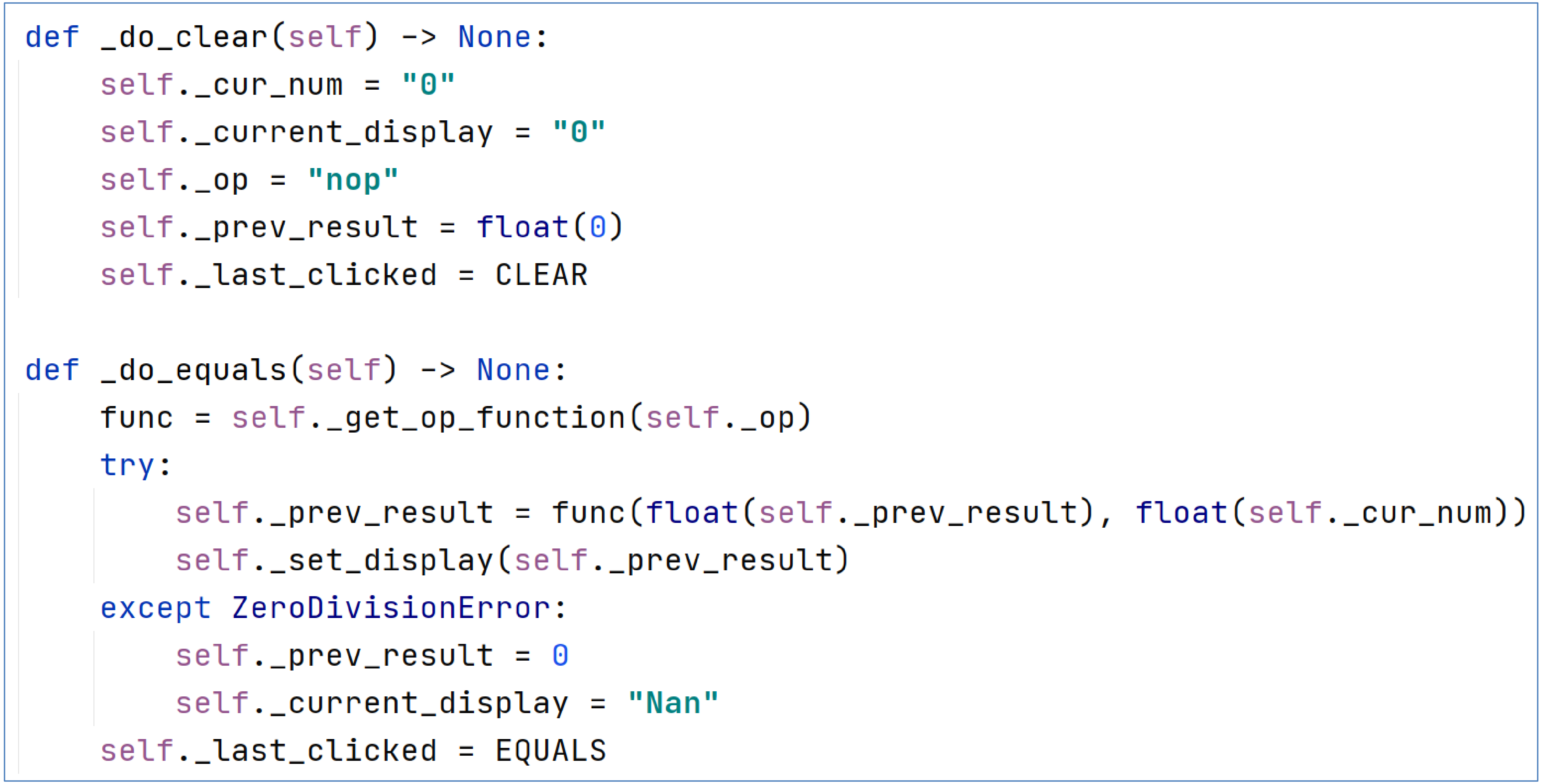
When an operator is pressed we apply the **previous** operator to old + new number and store the result

```python
class CalculatorModel:

    def __init__(self) -> None:...

    def get_display(self) -> str:...

    def type_in(self, c: str) -> None:...
```

```python
class CalculatorModel:
    _cur_num: str
    _prev_result: float
    _current_display: str
    _op: str
    _last_clicked: str

    def __init__(self) -> None:
        self._do_clear()

    def get_display(self) -> str:
        return self._current_display

    def type_in(self, c: str) -> None:
        if c.isdigit() or c == ".":
            self._do_digit_clicked(c)
        elif c == "=":
            self._do_equals()
        elif c == "C":
            self._do_clear()
        elif c in {"*", "+", "-", "/"}:
            self._do_math_op(c)
        else:
            raise ValueError("Unknown key")
```

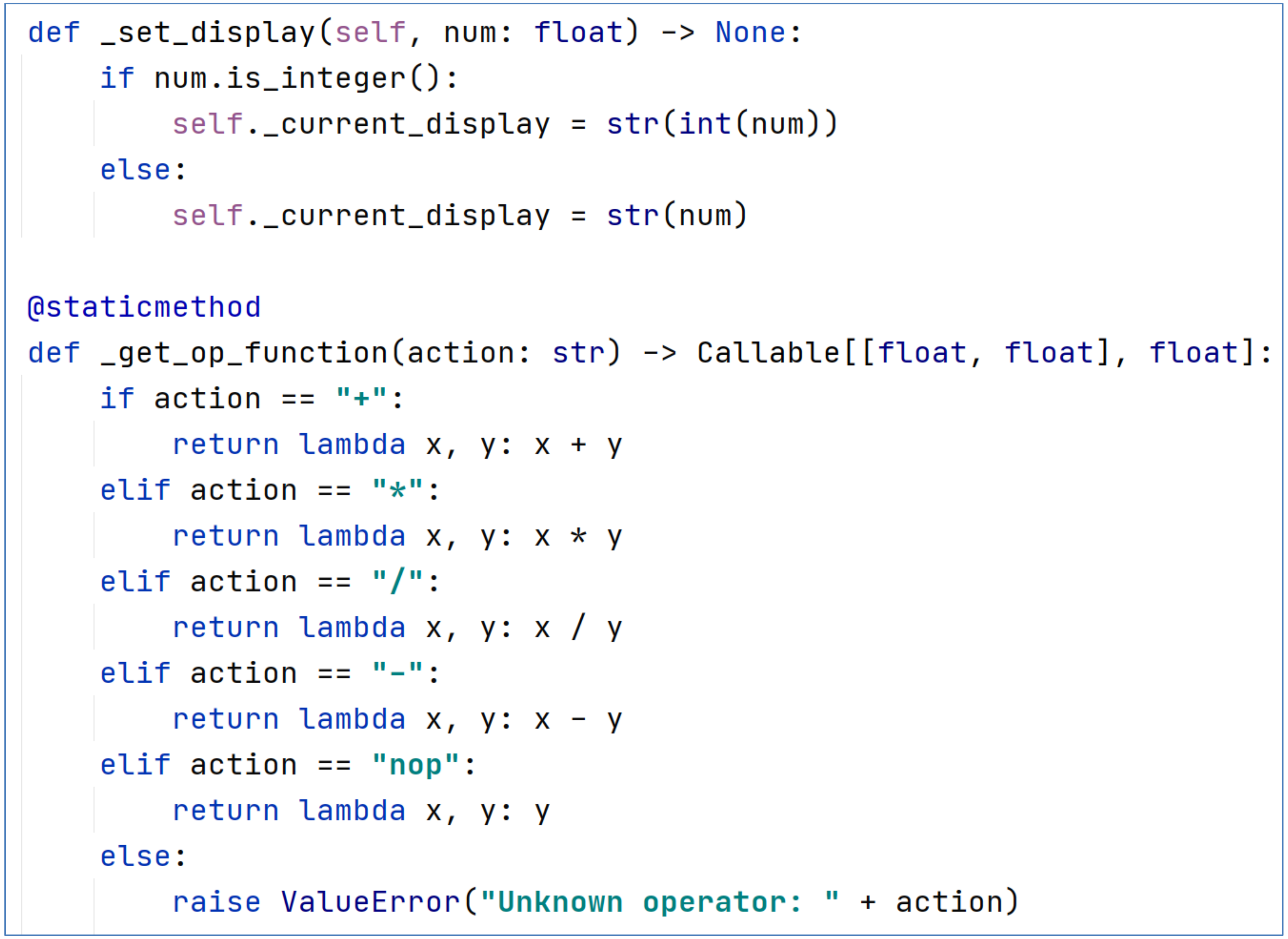


```
    def _do_clear(self) -> None:...

    def _do_equals(self) -> None:...

    def _do_math_op(self, op: str) -> None:...

    def _do_digit_clicked(self, digit: str) -> None:...

    def _set_display(self, num: float) -> None:...

    @staticmethod
    def _get_op_function(action: str) -> Callable[[float, float], float]:...
```



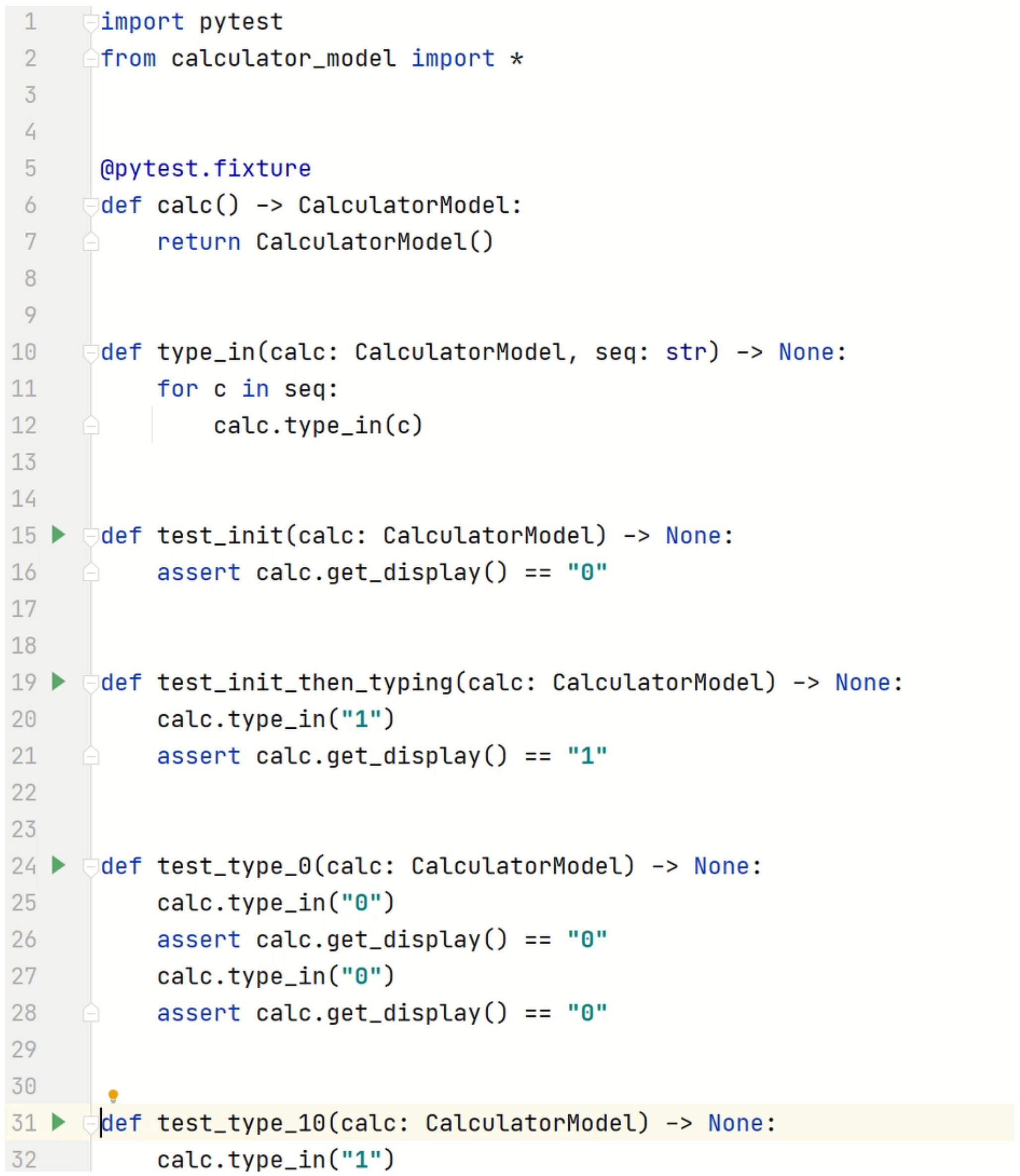
```python
def _do_clear(self) -> None:
    self._cur_num = "0"
    self._current_display = "0"
    self._op = "nop"
    self._prev_result = float(0)
    self._last_clicked = CLEAR

def _do_equals(self) -> None:
    func = self._get_op_function(self._op)
    try:
        self._prev_result = func(float(self._prev_result), float(self._cur_num))
        self._set_display(self._prev_result)
    except ZeroDivisionError:
        self._prev_result = 0
        self._current_display = "Nan"
    self._last_clicked = EQUALS
```

```python
def _do_math_op(self, op: str) -> None:
    if self._last_clicked != OP and self._last_clicked != EQUALS:
        self._do_equals()
    self._op = op
    self._last_clicked = OP

def _do_digit_clicked(self, digit: str) -> None:
    if self._last_clicked is not DIGIT:
        self._cur_num = digit
    else:
        if digit != "." or "." not in self._cur_num:
            self._cur_num += digit
    self._set_display(float("0" + self._cur_num))
    if self._last_clicked == EQUALS:
        self._op = "nop"
    self._last_clicked = DIGIT
```



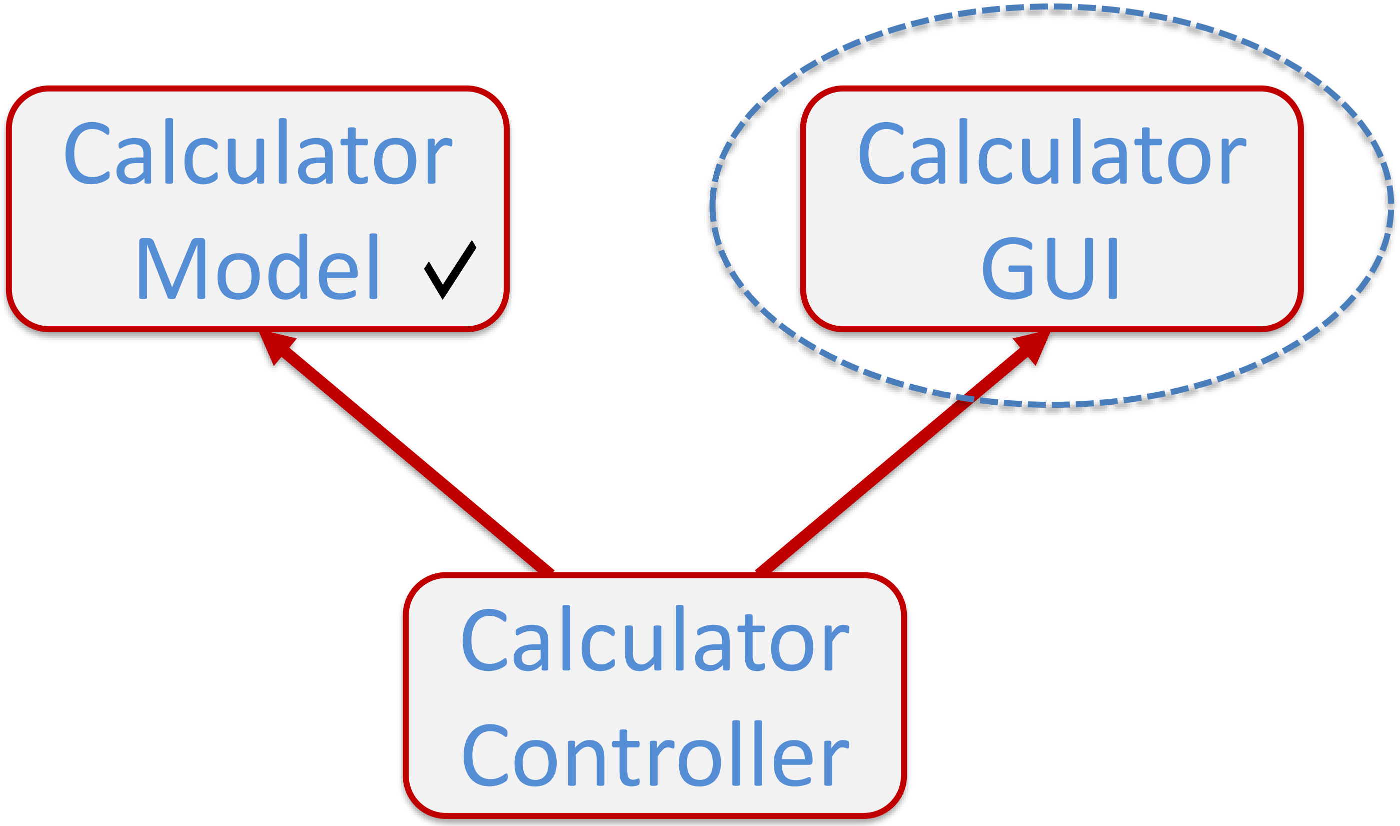
```python
def _set_display(self, num: float) -> None:
    if num.is_integer():
        self._current_display = str(int(num))
    else:
        self._current_display = str(num)

@staticmethod
def _get_op_function(action: str) -> Callable[[float, float], float]:
    if action == "+":
        return lambda x, y: x + y
    elif action == "*":
        return lambda x, y: x * y
    elif action == "/":
        return lambda x, y: x / y
    elif action == "-":
        return lambda x, y: x - y
    elif action == "nop":
        return lambda x, y: y
    else:
        raise ValueError("Unknown operator: " + action)
```

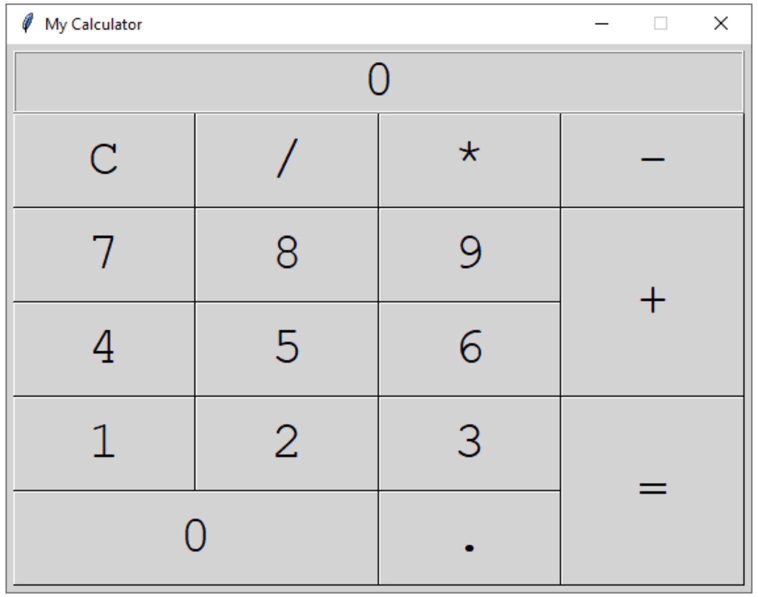
- The CalculatorModel class contains all the complex logic
- It does not do anything GUI related
- We can and should write tests for it



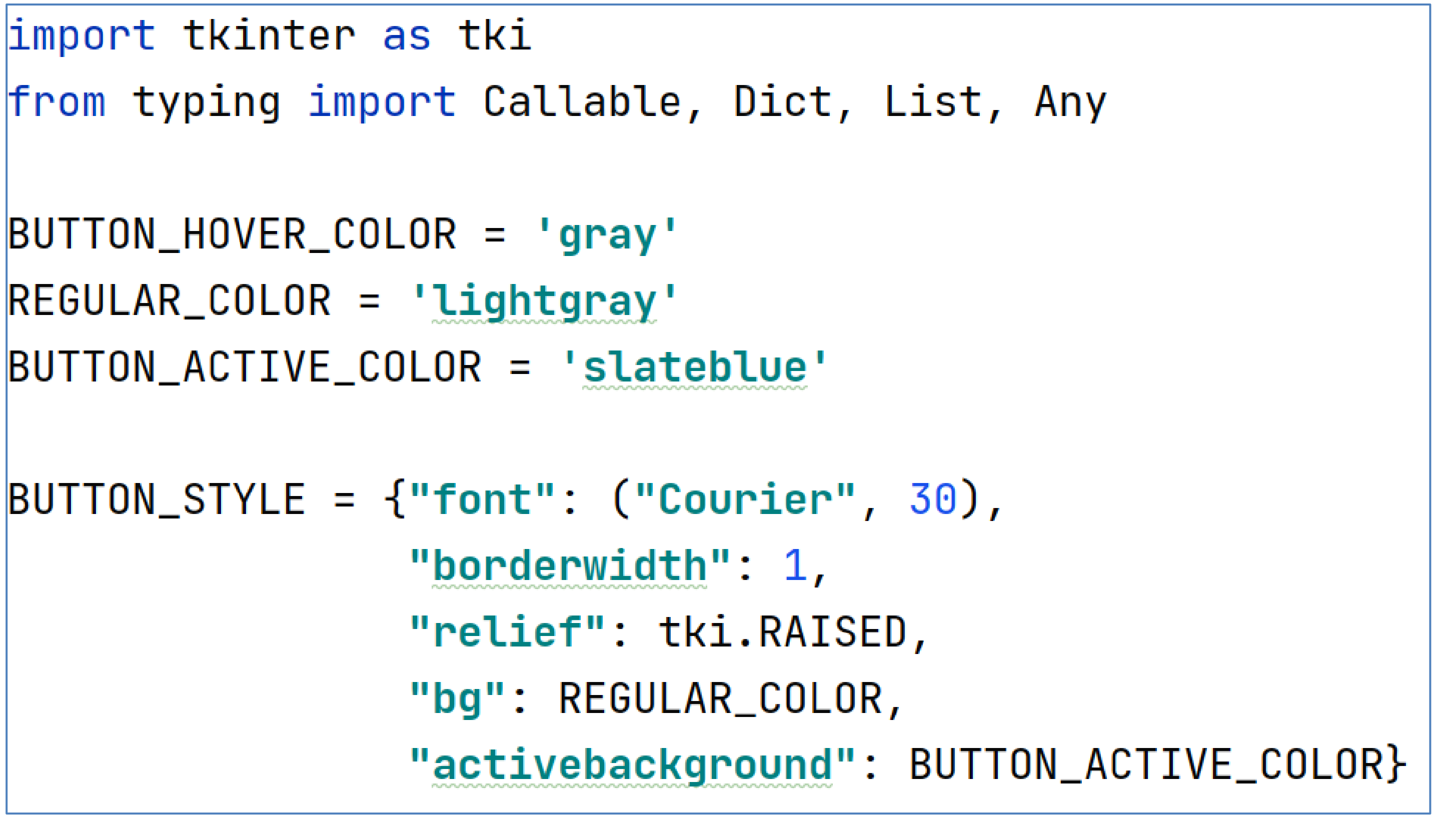
```python
import pytest
from calculator_model import *


@pytest.fixture
def calc() -> CalculatorModel:
    return CalculatorModel()


def type_in(calc: CalculatorModel, seq: str) -> None:
    for c in seq:
        calc.type_in(c)


def test_init(calc: CalculatorModel) -> None:
    assert calc.get_display() == "0"


def test_init_then_typing(calc: CalculatorModel) -> None:
    calc.type_in("1")
    assert calc.get_display() == "1"


def test_type_0(calc: CalculatorModel) -> None:
    calc.type_in("0")
    assert calc.get_display() == "0"
    calc.type_in("0")
    assert calc.get_display() == "0"


def test_type_10(calc: CalculatorModel) -> None:
    calc.type_in("1")
```

Design of the program



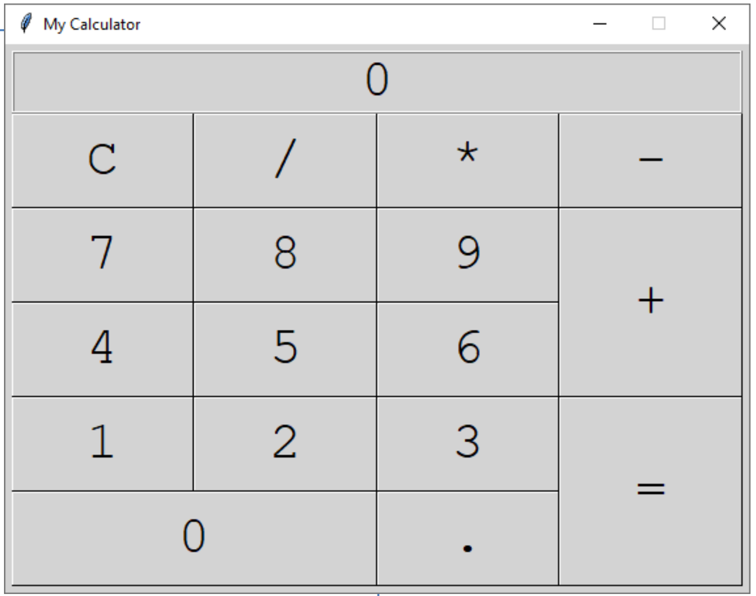


```python
class CalculatorGUI:

    def __init__(self) -> None:...

    def run(self) -> None:...

    def set_display(self, display_text: str) -> None:...

    def set_button_command(self, button_name: str, cmd: Callable[[], None]) -> None:

    def get_button_chars(self) -> List[str]:...
```



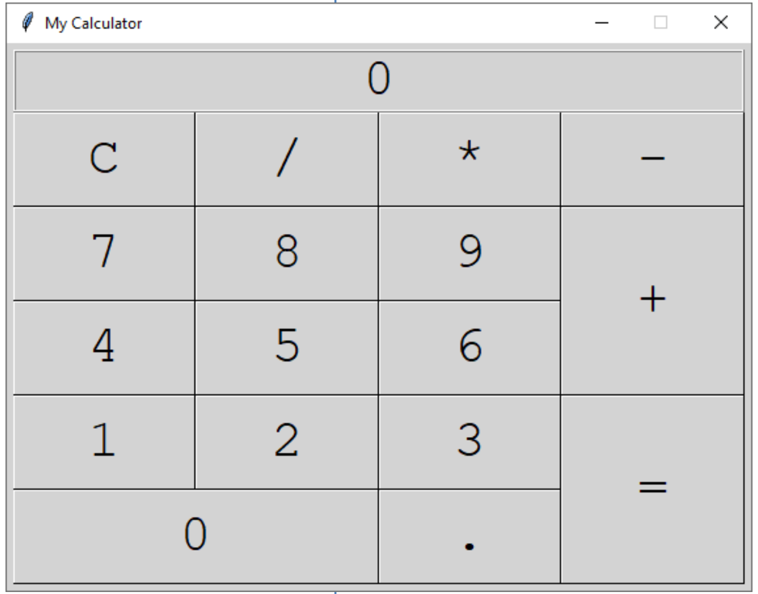
```python
import tkinter as tki
from typing import Callable, Dict, List, Any

BUTTON_HOVER_COLOR = 'gray'
REGULAR_COLOR = 'lightgray'
BUTTON_ACTIVE_COLOR = 'slateblue'

BUTTON_STYLE = {"font": ("Courier", 30),
                "borderwidth": 1,
                "relief": tki.RAISED,
                "bg": REGULAR_COLOR,
                "activebackground": BUTTON_ACTIVE_COLOR}
```

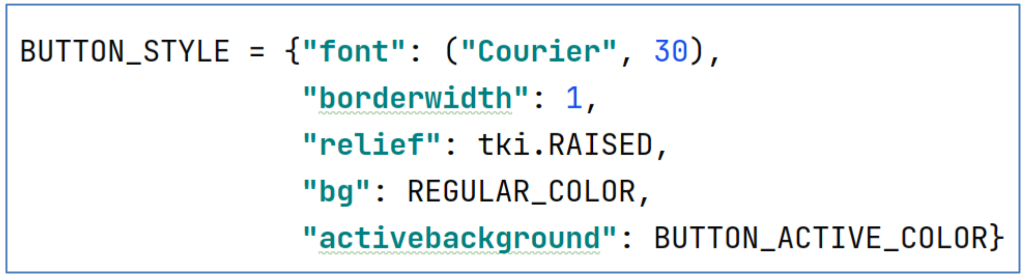
```python
class CalculatorGUI:
    _buttons: Dict[str, tki.Button] = {}

    def __init__(self) -> None:
        root = tki.Tk()
        root.title("My Calculator")
        root.resizable(False, False)
        self._main_window = root

        self._outer_frame = tki.Frame(root, bg=REGULAR_COLOR,
                                      highlightbackground=REGULAR_COLOR,
                                      highlightthickness=5)
        self._outer_frame.pack(side=tki.TOP, fill=tki.BOTH, expand=True)

        self._display_label = tki.Label(self._outer_frame, font=("Courier", 30),
                                        bg=REGULAR_COLOR, width=23, relief="ridge")
        self._display_label.pack(side=tki.TOP, fill=tki.BOTH)

        self._lower_frame = tki.Frame(self._outer_frame)
        self._lower_frame.pack(side=tki.TOP, fill=tki.BOTH, expand=True)

        self._create_buttons_in_lower_frame()
        self._main_window.bind("<Key>", self._key_pressed)
```



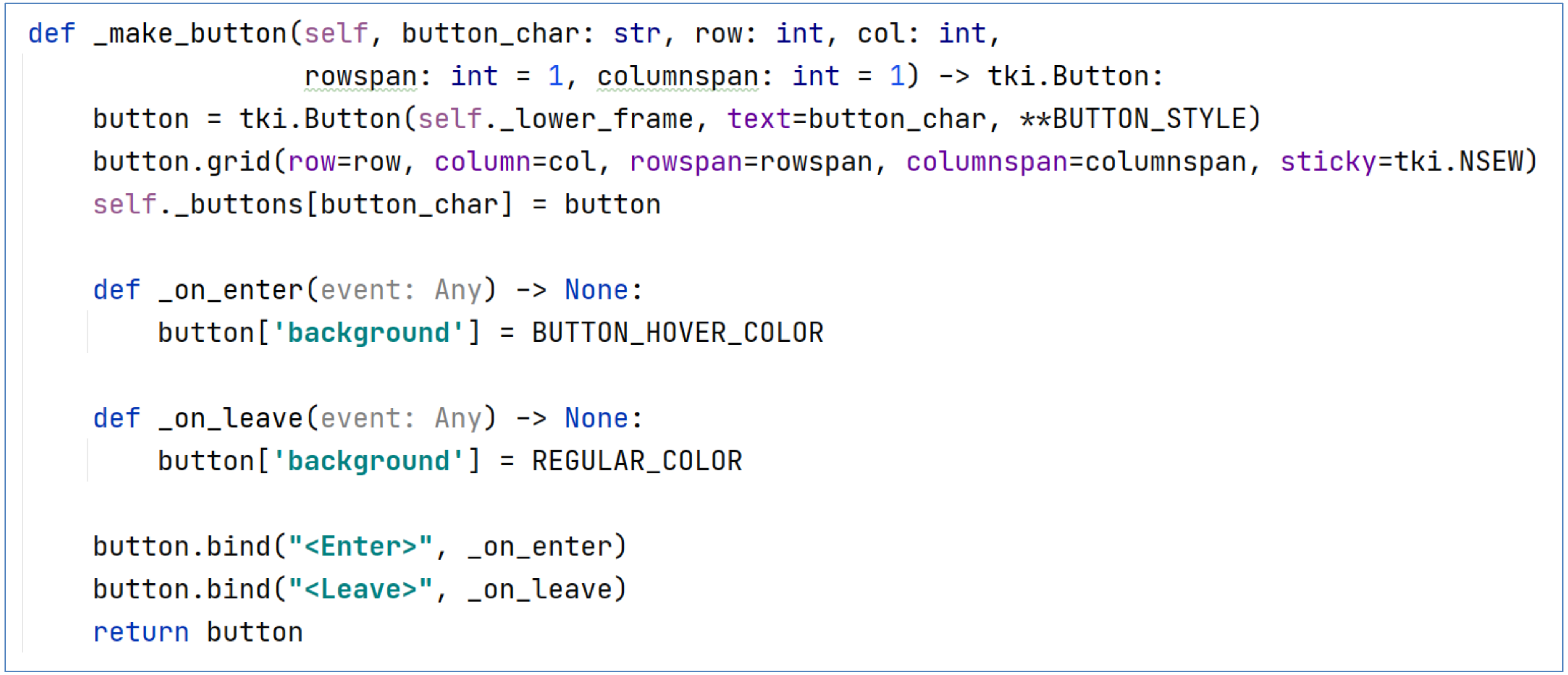
```python
def run(self) -> None:
    self._main_window.mainloop()

def set_display(self, display_text: str) -> None:
    self._display_label["text"] = display_text

def set_button_command(self, button_name: str, cmd: Callable[[], None]) -> None:
    self._buttons[button_name].configure(command=cmd)

def get_button_chars(self) -> List[str]:
    return list(self._buttons.keys())
```

```python
def _create_buttons_in_lower_frame(self) -> None:
    for i in range(4):
        tki.Grid.columnconfigure(self._lower_frame, i, weight=1)  # type: ignore

    for i in range(5):
        tki.Grid.rowconfigure(self._lower_frame, i, weight=1)  # type: ignore

    self._make_button("C", 0, 0)
    self._make_button("/", 0, 1)
    self._make_button("*", 0, 2)
    self._make_button("-", 0, 3)
    self._make_button("7", 1, 0)
    self._make_button("8", 1, 1)
    self._make_button("9", 1, 2)
    self._make_button("+", 1, 3, rowspan=2)
    self._make_button("4", 2, 0)
    self._make_button("5", 2, 1)
    self._make_button("6", 2, 2)
    self._make_button("1", 3, 0)
    self._make_button("2", 3, 1)
    self._make_button("3", 3, 2)
    self._make_button("=", 3, 3, rowspan=2)
    self._make_button("0", 4, 0, columnspan=2)
    self._make_button(".", 4, 2)
```





```python
BUTTON_STYLE = {"font": ("Courier", 30),
                "borderwidth": 1,
                "relief": tki.RAISED,
                "bg": REGULAR_COLOR,
                "activebackground": BUTTON_ACTIVE_COLOR}
```



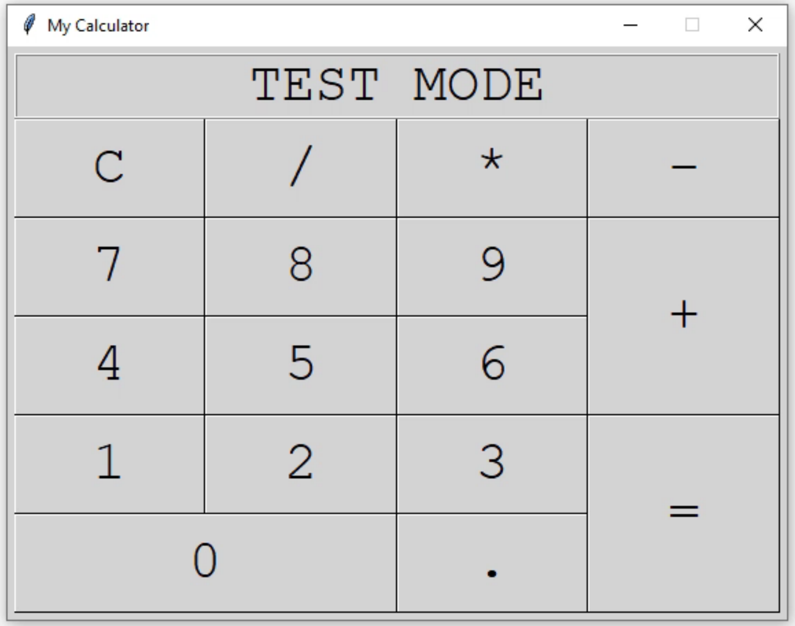
```python
def _make_button(self, button_char: str, row: int, col: int,
                 rowspan: int = 1, columnspan: int = 1) -> tki.Button:
    button = tki.Button(self._lower_frame, text=button_char, **BUTTON_STYLE)
    button.grid(row=row, column=col, rowspan=rowspan, columnspan=columnspan, sticky=tki.NSEW)
    self._buttons[button_char] = button

    def _on_enter(event: Any) -> None:
        button['background'] = BUTTON_HOVER_COLOR

    def _on_leave(event: Any) -> None:
        button['background'] = REGULAR_COLOR

    button.bind("<Enter>", _on_enter)
    button.bind("<Leave>", _on_leave)
    return button
```

```python
def _key_pressed(self, event: Any) -> None:
    """the callback method for when a key is pressed.
    It'll simulate a button press on the right button."""
    if event.char in self._buttons:
        self._simulate_button_press(event.char)
    elif event.keysym == "Return":
        self._simulate_button_press("=")


def _simulate_button_press(self, button_char: str) -> None:
    """make a button light up as if it is pressed,
    and then return to normal"""
    button = self._buttons[button_char]
    button["bg"] = BUTTON_ACTIVE_COLOR

    def return_button_to_normal() -> None:
        # find which widget the mouse is pointing at:
        x, y = self._main_window.winfo_pointerxy()
        widget_under_mouse = self._main_window.winfo_containing(x, y)
        # change color accordingly:
        if widget_under_mouse is button:
            button["bg"] = BUTTON_HOVER_COLOR
        else:
            button["bg"] = REGULAR_COLOR

    button.invoke()  # type: ignore
    button.after(100, func=return_button_to_normal)
```

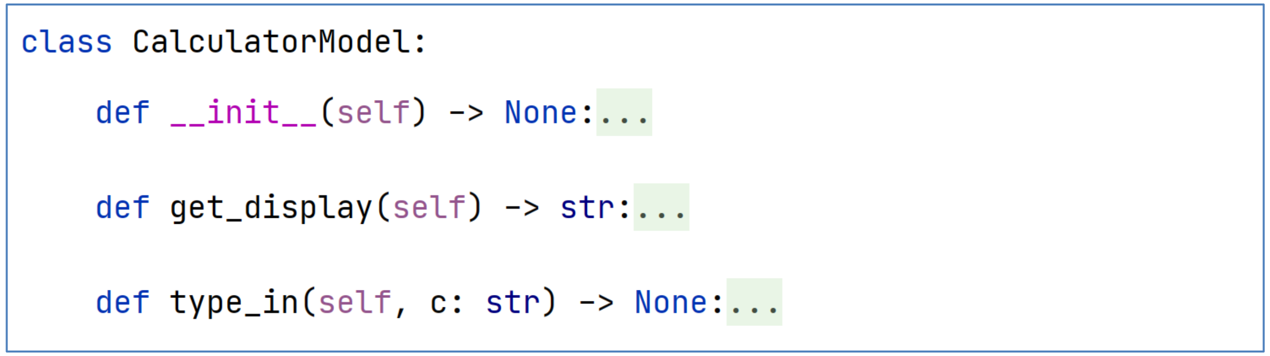
In `__init__()` we had:

```python
self._main_window.bind("<Key>", self._key_pressed)
```

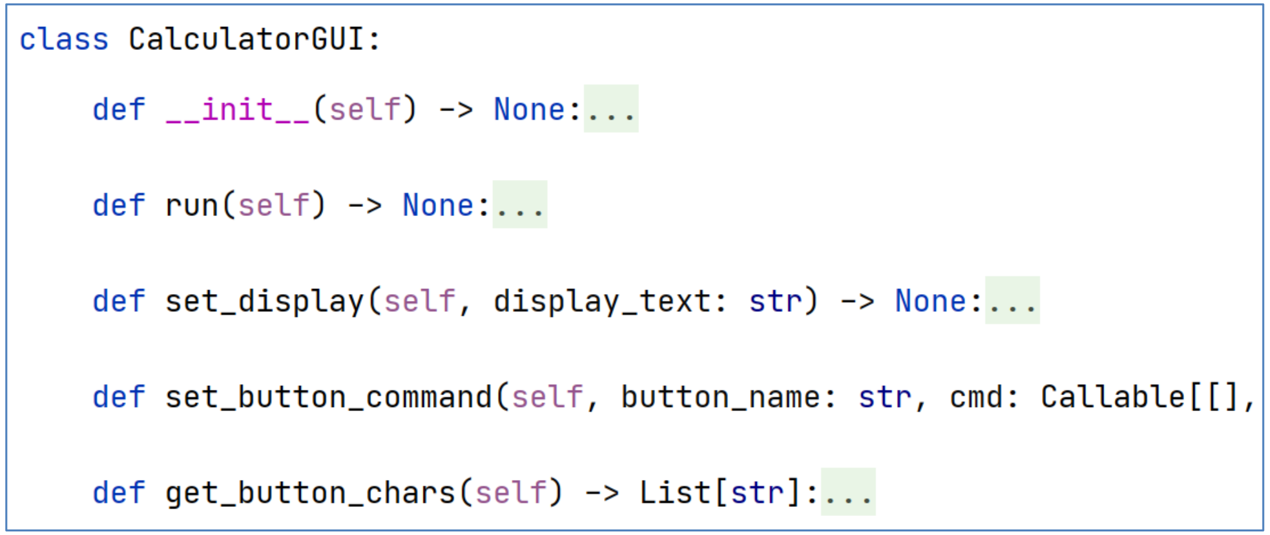
```python
if __name__ == "__main__":
    cg = CalculatorGUI()
    cg.set_display("TEST MODE")
    cg.run()
```



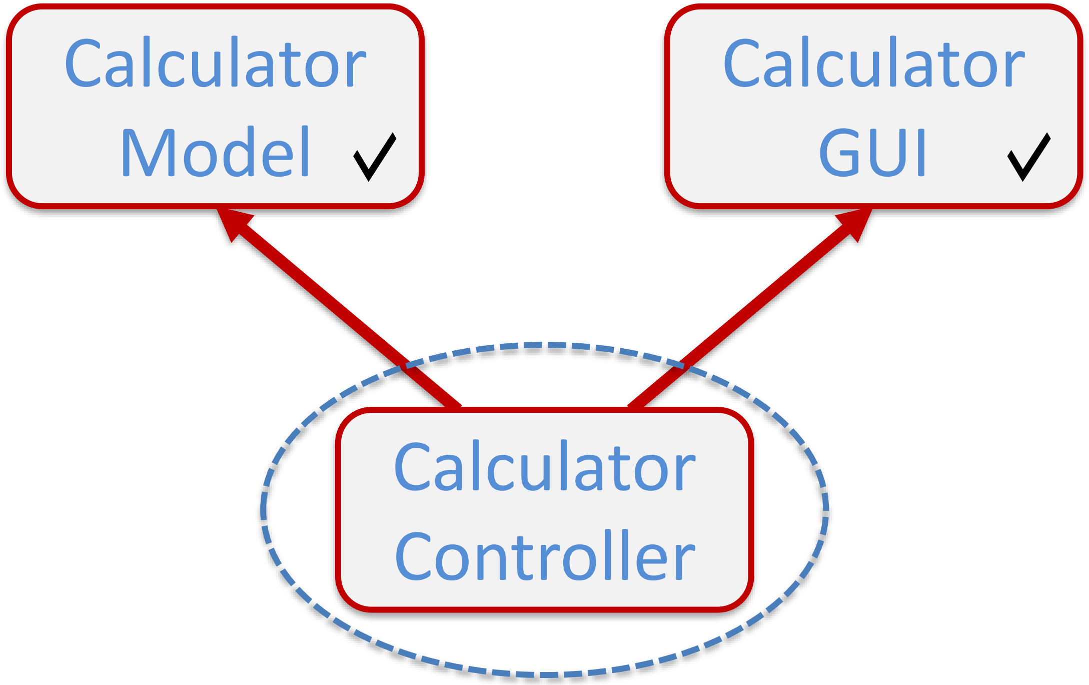
Design of the program



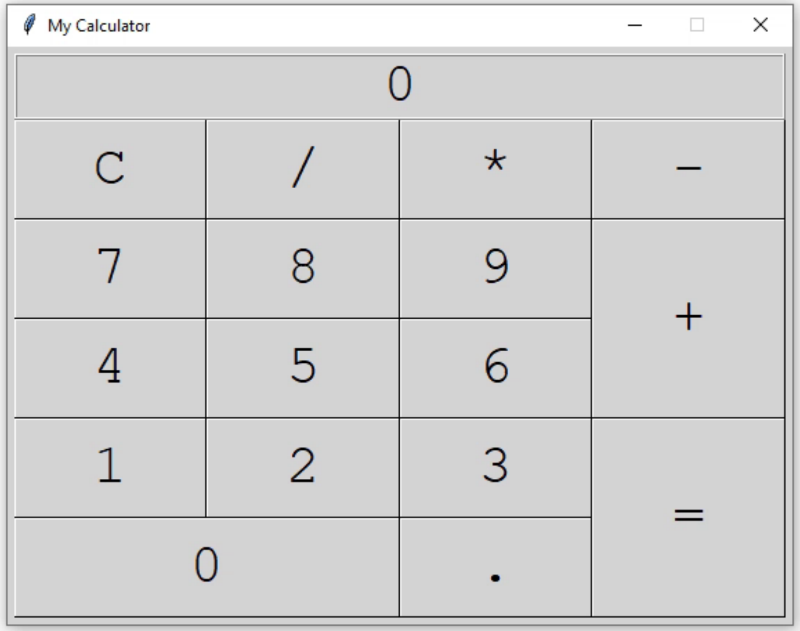
```python
class CalculatorModel:

    def __init__(self) -> None:...

    def get_display(self) -> str:...

    def type_in(self, c: str) -> None:...
```



```python
class CalculatorGUI:

    def __init__(self) -> None:...

    def run(self) -> None:...

    def set_display(self, display_text: str) -> None:...

    def set_button_command(self, button_name: str, cmd: Callable[[],

    def get_button_chars(self) -> List[str]:...
```



```python
from typing import Callable
from calculator_model import CalculatorModel
from calculator_gui import CalculatorGUI


class CalculatorController:
    def __init__(self) -> None:
        self._gui = CalculatorGUI()
        self._model = CalculatorModel()

        for button_text in self._gui.get_button_chars():
            action = self.create_button_action(button_text)
            self._gui.set_button_command(button_text, action)
        self._gui.set_display("0")

    def create_button_action(self, button_text: str) -> Callable[[], None]:
        def fun() -> None:
            self._model.type_in(button_text)
            self._gui.set_display(self._model.get_display())

        return fun

    def run(self) -> None:
        self._gui.run()


if __name__ == "__main__":
    CalculatorController().run()
```


My Calculator
0
C
/
*
−
7
8
9
+
4
5
6
1
2
3
=
0
.