Simple GUI Calculator

Using OOP

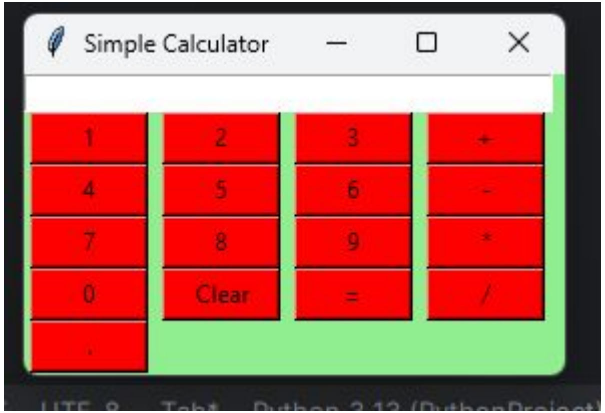
To Create a Calc

Importing the module – tkinter

Create the main window (container)

Add any number of widgets to the main window

Apply the event Trigger on the widgets.



Getting started

Step 1: Importing the Required Module

First, import everything from the Tkinter module, which will help create the GUI interface.

Step 2: Declaring Global Variables

Define a global variable `expression` to store the user input.

Step 3: Creating Functions

1. Function to Update the Expression

This function updates the expression in the text entry box when a button is clicked.

2. Function to Evaluate the Expression

This function evaluates the arithmetic expression entered by the user.

3. Function to Clear the Input

This function clears the entry box.

4: Creating the GUI Window

1. Initialize the Tkinter Window

2. Creating the Entry Field

3: Adding Buttons to the Calculator

4. Creating Operator and Special Buttons

5. Creating the Equal Button

Running the Tkinter Event Loop

```
gui.mainloop()
```

File Location

practice/gui/calc.py

Why Convert to OOP?

Encapsulation: Groups related functions and variables into a single class.

Modularity: Code is organized into separate methods for better readability.

Reusability: The class can be easily reused and extended without modifying existing code.

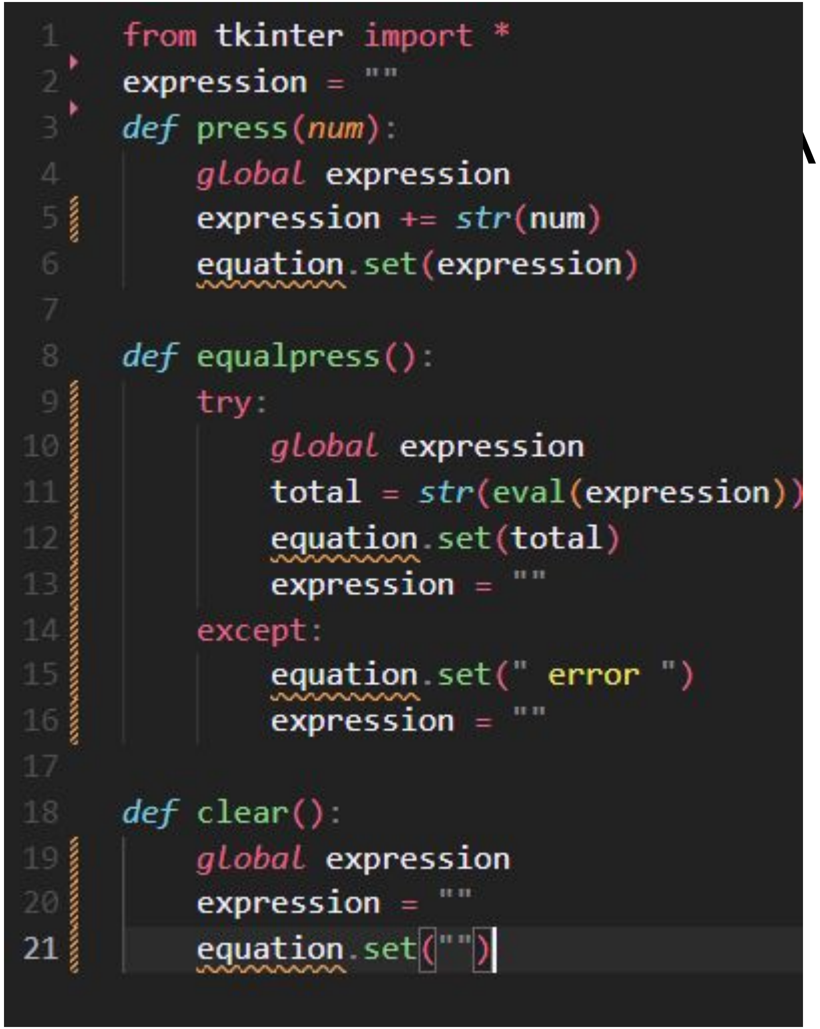
Scalability: Makes it easier to add new features without breaking existing functionality.

Maintainability: Reduces redundancy and makes debugging simpler.

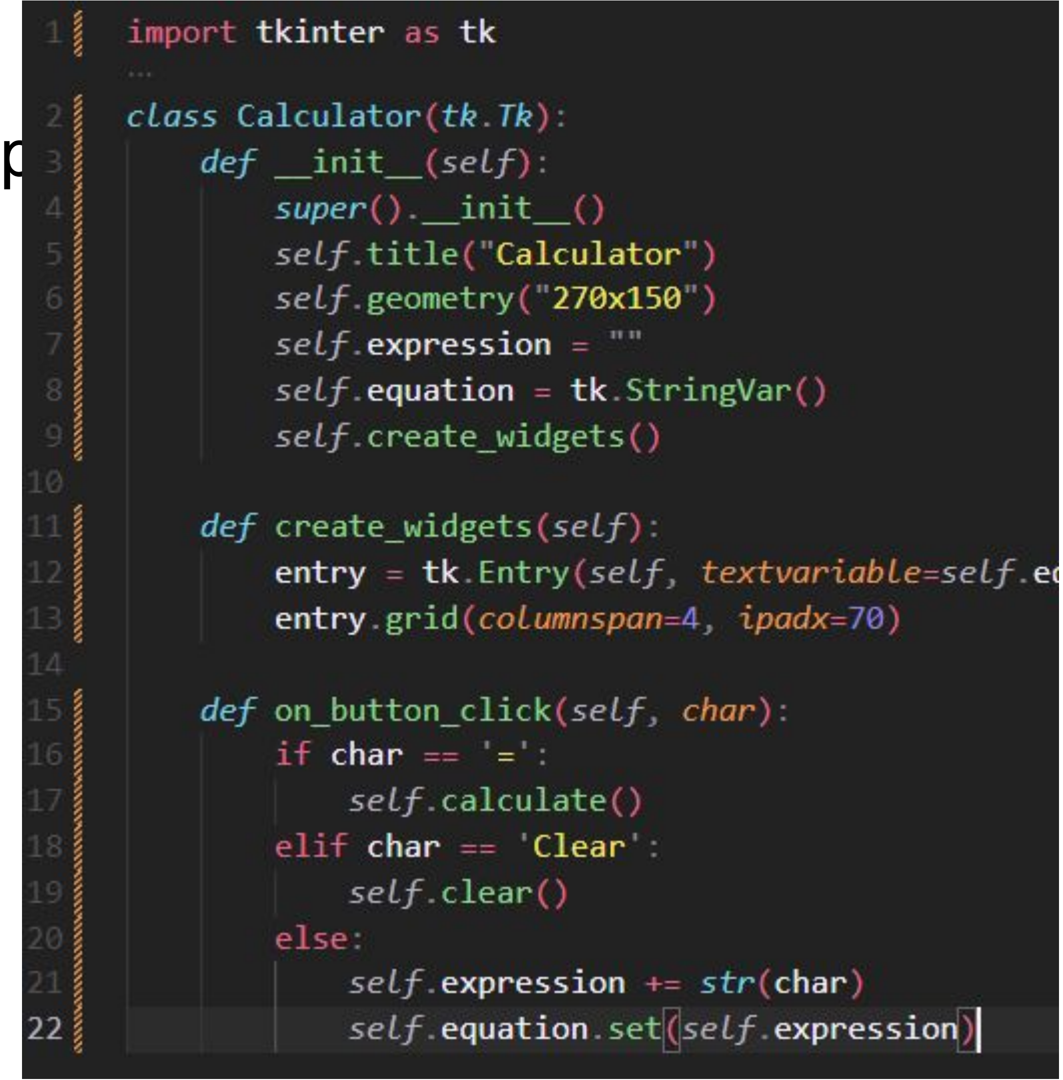
Steps to Convert Tkinter to OOP

1. **Create a Class**
   - Define a class that inherits from `tk.Tk`.
   - Initialize the `Tk` window inside the constructor.
2. **Move Global Variables into the Class**
   - Convert global variables (like `expression` and `equation`) into instance attributes.
3. **Encapsulate Functions into Methods**
   - Convert procedural functions (`press`, `equalpress`, `clear`) into instance methods.
4. **Create a Method for UI Setup**
   - Move all widget creation code into a method (`create_widgets`).
5. **Use `self` for Accessing Instance Attributes and Methods**
   - Replace global function calls with `self.method_name`.

Ap



```python
from tkinter import *
expression = ""
def press(num):
    global expression
    expression += str(num)
    equation.set(expression)

def equalpress():
    try:
        global expression
        total = str(eval(expression))
        equation.set(total)
        expression = ""
    except:
        equation.set(" error ")
        expression = ""

def clear():
    global expression
    expression = ""
    equation.set("")
```



```python
import tkinter as tk

class Calculator(tk.Tk):
    def __init__(self):
        super().__init__()
        self.title("Calculator")
        self.geometry("270x150")
        self.expression = ""
        self.equation = tk.StringVar()
        self.create_widgets()

    def create_widgets(self):
        entry = tk.Entry(self, textvariable=self.ec
        entry.grid(columnspan=4, ipadx=70)

    def on_button_click(self, char):
        if char == '=':
            self.calculate()
        elif char == 'Clear':
            self.clear()
        else:
            self.expression += str(char)
            self.equation.set(self.expression)
```

Key Takeaways

- OOP structures code better by **encapsulating logic into classes.**
- Instance attributes **replace global variables, reducing errors**.
- Methods replace standalone **functions for better organization**.
- Improves **maintainability, scalability, and code reusability**.
- Recommended for any **medium-to-large** Tkinter projects!

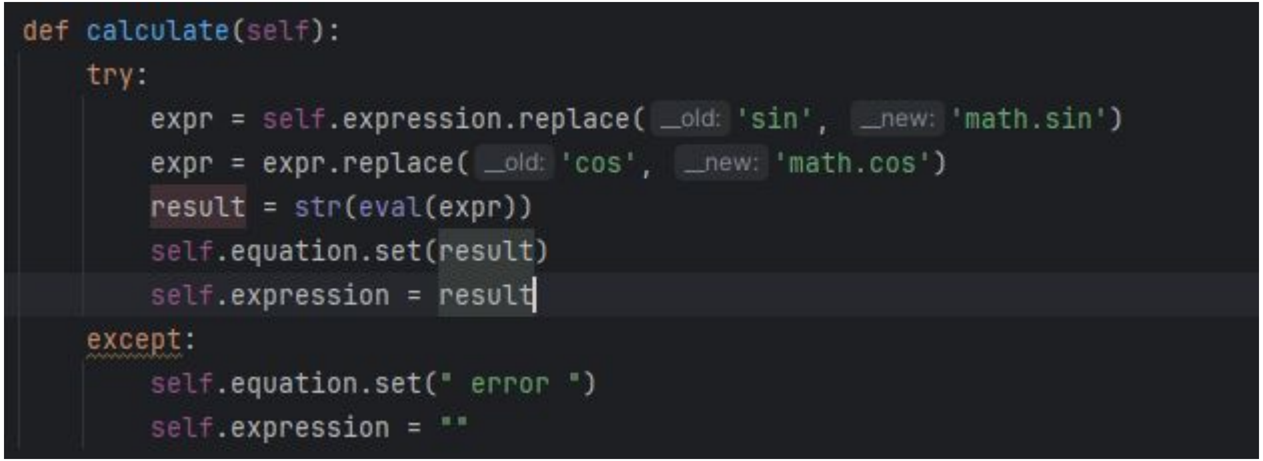
Scientific calculator

```
class BaseCalculator(tk.Tk):
```

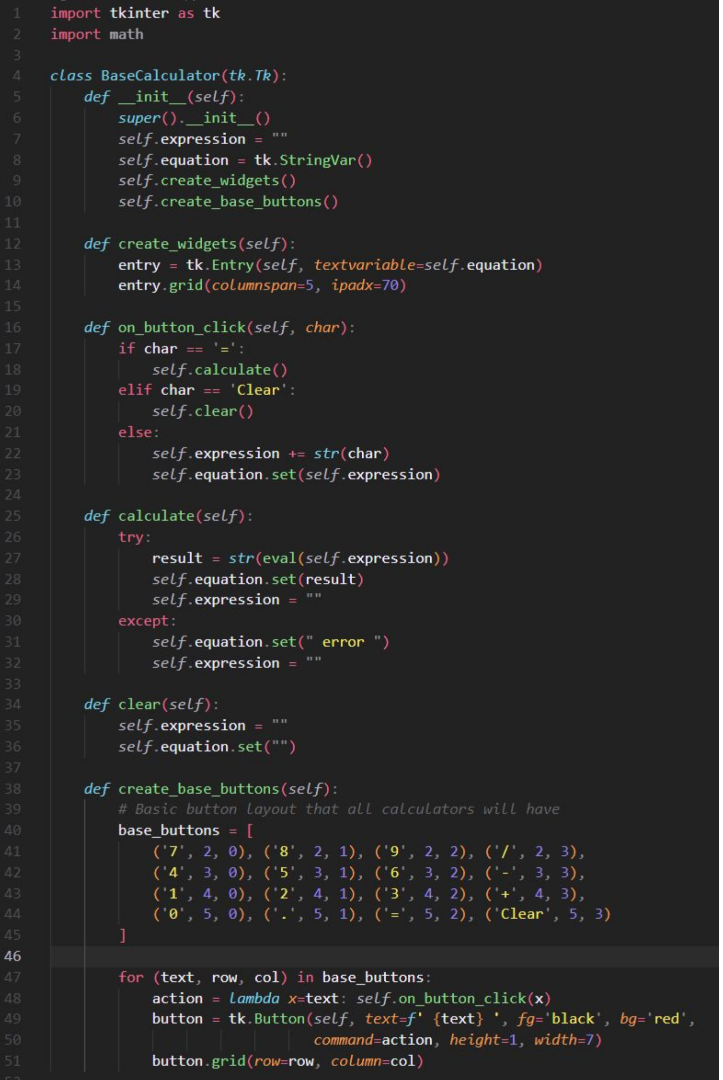
Calc code

```
Class StandardCalculator(BaseCalculator):

    That is the child of main class


Class ScientificCalculator(BaseCalculator):

    That is the child of main class
```

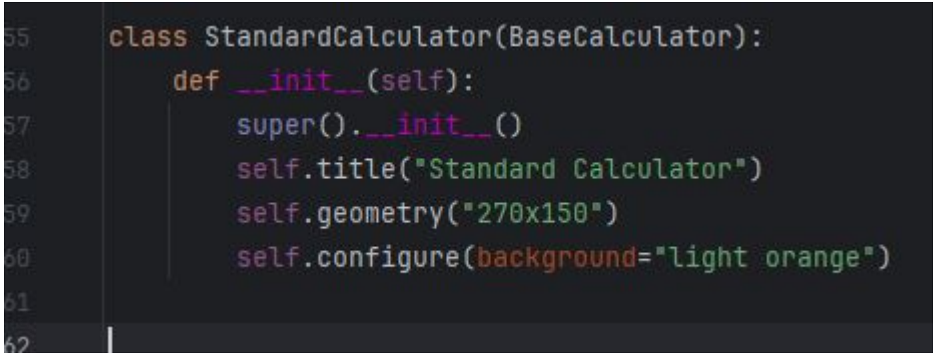
New Features, cos and sin



```
def calculate(self):
    try:
        expr = self.expression.replace( __old: 'sin',  __new: 'math.sin')
        expr = expr.replace( __old: 'cos',  __new: 'math.cos')
        result = str(eval(expr))
        self.equation.set(result)
        self.expression = result
    except:
        self.equation.set(" error ")
        self.expression = ""
```

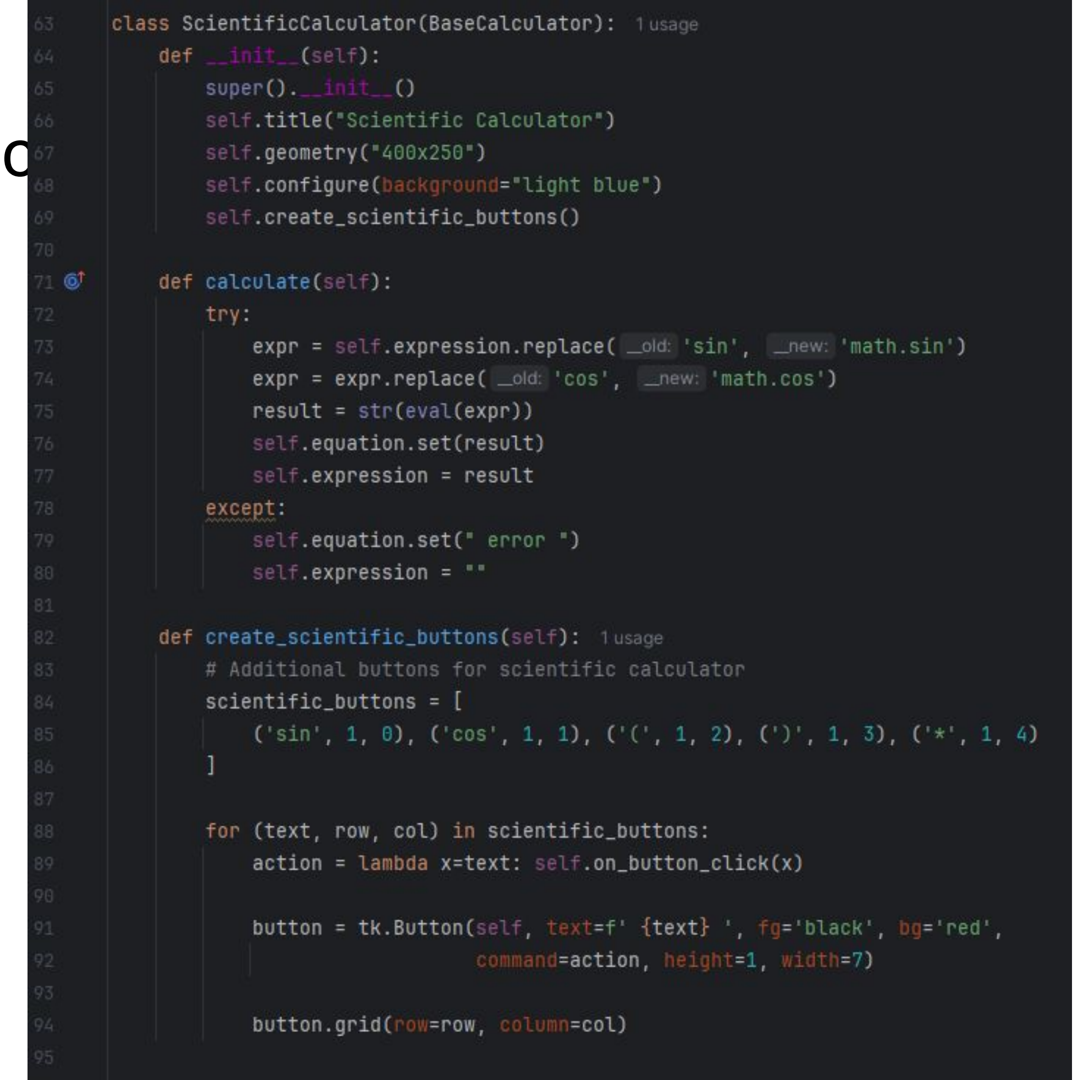


```python
import tkinter as tk
import math

class BaseCalculator(tk.Tk):
    def __init__(self):
        super().__init__()
        self.expression = ""
        self.equation = tk.StringVar()
        self.create_widgets()
        self.create_base_buttons()

    def create_widgets(self):
        entry = tk.Entry(self, textvariable=self.equation)
        entry.grid(columnspan=5, ipadx=70)

    def on_button_click(self, char):
        if char == '=':
            self.calculate()
        elif char == 'Clear':
            self.clear()
        else:
            self.expression += str(char)
            self.equation.set(self.expression)

    def calculate(self):
        try:
            result = str(eval(self.expression))
            self.equation.set(result)
            self.expression = ""
        except:
            self.equation.set(" error ")
            self.expression = ""

    def clear(self):
        self.expression = ""
        self.equation.set("")

    def create_base_buttons(self):
        # Basic button layout that all calculators will have
        base_buttons = [
            ('7', 2, 0), ('8', 2, 1), ('9', 2, 2), ('/', 2, 3),
            ('4', 3, 0), ('5', 3, 1), ('6', 3, 2), ('-', 3, 3),
            ('1', 4, 0), ('2', 4, 1), ('3', 4, 2), ('+', 4, 3),
            ('0', 5, 0), ('.', 5, 1), ('=', 5, 2), ('Clear', 5, 3)
        ]

        for (text, row, col) in base_buttons:
            action = lambda x=text: self.on_button_click(x)
            button = tk.Button(self, text=f' {text} ', fg='black', bg='red',
                               command=action, height=1, width=7)
            button.grid(row=row, column=col)
```

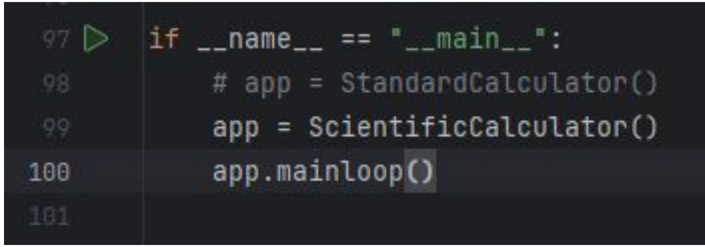


```python
class StandardCalculator(BaseCalculator):
    def __init__(self):
        super().__init__()
        self.title("Standard Calculator")
        self.geometry("270x150")
        self.configure(background="light orange")

```

Sc



```python
class ScientificCalculator(BaseCalculator):
    def __init__(self):
        super().__init__()
        self.title("Scientific Calculator")
        self.geometry("400x250")
        self.configure(background="light blue")
        self.create_scientific_buttons()

    def calculate(self):
        try:
            expr = self.expression.replace('sin', 'math.sin')
            expr = expr.replace('cos', 'math.cos')
            result = str(eval(expr))
            self.equation.set(result)
            self.expression = result
        except:
            self.equation.set(" error ")
            self.expression = ""

    def create_scientific_buttons(self):
        # Additional buttons for scientific calculator
        scientific_buttons = [
            ('sin', 1, 0), ('cos', 1, 1), ('(', 1, 2), (')', 1, 3), ('*', 1, 4)
        ]

        for (text, row, col) in scientific_buttons:
            action = lambda x=text: self.on_button_click(x)

            button = tk.Button(self, text=f' {text} ', fg='black', bg='red',
                               command=action, height=1, width=7)

            button.grid(row=row, column=col)

```



```python
if __name__ == "__main__":
    # app = StandardCalculator()
    app = ScientificCalculator()
    app.mainloop()

```

Styling

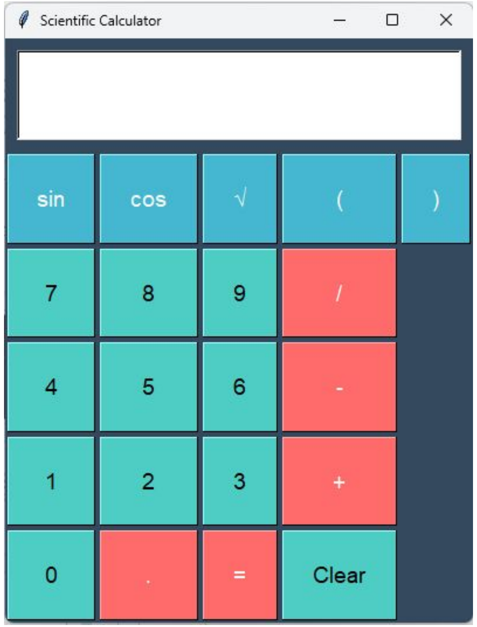
relief='raised' : for styling border

pad**x**=2: Horizontal padding

pad**y**=2: Vertically padding

sticky='nsew' : How the button "sticks" to its grid cell

'nsew' = North, South, East, West



Class work: Add the following features

-

Square root √

exponential function a^2

exponential function a^{10}

Percentage %