Calculator_Project

September 29, 2025

1 Mini Calculator Project

Course: Python Final Project
Group: Group 3
Date: 28 September 2025
Team Members: - Reihaneh Niknahad - Thi Ngoc Hanh Nguyen - Crishan Tharaka - Harisha Wishmini - Seyeddanial Kazemi

Introduction:
This project implements a simple calculator with a Gradio user interface.
It supports basic arithmetic operations (addition, subtraction, multiplication, division, square, and square root) and includes error handling for invalid inputs.

1.1 GitHub repository: https://github.com/moonwindy1981-hub/Python-calculator-project

1.2 Calculator_Project HTML link: https://moonwindy1981-hub.github.io/Python-calculator-project/?

1.3 Project Description

Our project is a **Mini Calculator with a simple UI built using Gradio**.

It supports basic arithmetic operations (`+`, `-`, `*`, and `/`), and also advanced ones (x^2, $\sqrt{x}$).

Error handling is included for division by zero and invalid square root inputs.

The program is divided into multiple files for better structure and reusability:

- `simple_calculator.py` – first console-based version
- `math_core.py` – core calculation functions
- `calculator.py` – UI with Gradio

1.4 Test Cases

#	Operation	Inputs	Expected Output	Actual Output	Status
1	Addition	5 + 7	12	12	

#	Operation	Inputs	Expected Output	Actual Output	Status
2	Subtraction	10 - 4	6	6	
3	Multiplication	3×8	24	24	
4	Division	$20 \div 5$	4	4	
5	Division by 0	$9 \div 0$	Error / Message	Error handled	
6	Square	5^2	25	25	
7	Square root	√9	3	3	
8	Square root	√-9	Error / Message	Error handled	

1.5 Peer Evaluation (teammates)

Reviewer	**What's Good**	**Suggestions**
Harisha Wishmini	- Works well with main operations: `+`, `-`, `*`, `/`, `^2`, `√`. - Good error checks for division by zero and square root of negatives. - Code is clear and reusable for other projects.	- Add a general power function (e.g., `a^b`) for more flexibility. - Lowercase conversion is not needed now, but could help if you allow keyboard input later. - Use a dictionary to map operators to functions instead of many `if/else` statements.
Reihaneh Niknahad	- Several interesting ideas were proposed. - All the key aspects of this project were thoroughly addressed.	- I wish there was a shorthand symbol we could use instead of writing so many if/else statements.
Crishan Tharaka	- Highly efficient event handling. - Clear layout. - Clean token mapping and clarity of codes.	- Make button sizing more flexible. - Improve display usability. - Implement a more flexible layout.

Reviewer	**What's Good**	**Suggestions**
Seyeddanial Kazemi	- All the main operations worked correctly. - Errors like division by zero and invalid inputs were handled properly. - The interface was simple and easy to use. - Code is straightforward and readable. - Covers all the basic arithmetic operations. - Handles common errors like division by zero. - Easy-to-understand design and clear output. - Good separation of logic and UI.	- Limited to single operations at a time (cannot handle expressions like 2 + 3 * 4). - Negative numbers are not fully supported (cannot start directly with a minus). - Only basic functionality, no advanced features like percentage, memory functions, or history. - Add support for negative numbers: Allow input such as -5 + 3. - Support multiple operations with correct precedence. - Improve formatting to avoid floating point issues. - Add features like percentage, memory buttons, or history. - Allow keyboard input, not only button clicks.
Nguyen Thi Ngoc Hanh	- Meets the basic requirements of the project. - Handles some exceptional cases properly. - Good team member contributions (Test & Code reviews).	- Did not manage group code work on GitHub like in a professional project. - Add more features such as exponentiation `x^y`. - Allow users to input numbers directly from the keyboard.

---

2 UI Event Handling and Core Logic Writing

The Gradio UI is connected to the core logic using a **single event handler**, `on_click`.

Concept	Description
Event Binding	Every `gr.Button`'s `.click()` method is linked to the `on_click` function.
Call Signature	The binding uses the following structure: `btn.click(on_click, inputs=[token, state, stopped], outputs=[state, display, stopped])`
State Management	The `on_click` function receives the current state (expression string) and the clicked button's token.

---

3 on_click Function Logic

The `on_click` function handles different input types:

3.0.1 Numbers/Operators

- **Action:** Appends the new token directly to the current state string.
- **Example:** State `"12"` + button `"+"` → New State `"12+"`.

3.0.2 Clear (C)

- **Action:** Resets the internal state to an empty string (`""`).

3.0.3 Calculation (=)

1. **Parsing:** When the `=` button is clicked, `on_click` parses the current state string to identify the operator and the operands (a and b).
2. **Core Call:** It calls the pure Python function `calculate(a, b, op)` to execute the mathematical operation and retrieve the result.

3.0.4 Error Handling

- The function uses standard **`try...except`** blocks to catch potential errors (e.g., `ZeroDivisionError`, `ValueError` from core functions).
- If an error occurs, the error message is prepared and displayed in the result box.

3.0.5 Display Update

- The final output of `on_click` (which includes the new state string and the resulting value/error) simultaneously updates:
  - The internal **`state`** variable (for continued input).
  - The visible **`display`** textbox (to show the result or error).

---

4 Python Code Implementation

[1]:
```python
"""
====================================================
 Mini Calculator Project - math_core.py
----------------------------------------------------
 Purpose:
   - Contains all core math functions (add, subtract, multiply, divide, sqrt,␣
 ↪power).
   - Designed to be reusable for different UIs.
==================================
"""
import math
```

```
def calculate(a, b, op):
    if op == '+':
        return a + b
    if op == '-':
        return a - b
    if op == '*':
        return a * b
    if op == '/':
        if b == 0:
            raise ZeroDivisionError("Cannot divide by 0")
        return a / b
    if op == '^2':
        return a ** 2
    if op == 'sqrt':
        if a < 0:
            raise ValueError("Cannot take sqrt of negative number")
        return math.sqrt(a)
    raise ValueError("Does not support this operator")
```

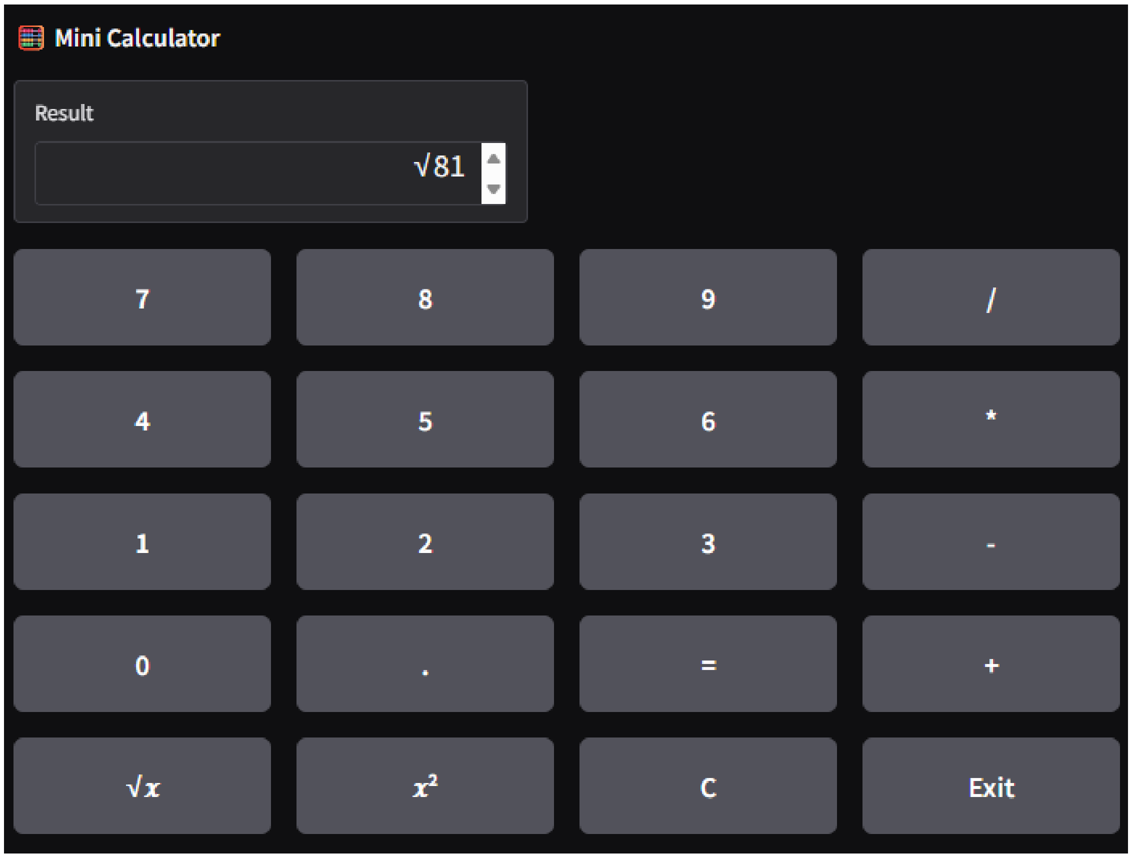
[2]:

```
"""
=====================================================
 Mini Calculator Project - calculator.py
-----------------------------------------------------
 Purpose:
   - Provides the User Interface using Gradio.
   - Connects UI buttons to functions in math_core.py.
=====================================================
"""

import gradio as gr
from math_core import calculate  # core calculation logic

# Map pretty labels -> internal tokens
PRETTY_TO_TOKEN = {
    "√ ": "√",
    " ²": "^2",
}

# ------------------------------
# 1. Validate input
# ------------------------------
def validate_input(state, btn):
    """Check invalid input cases before processing"""
    if state.endswith("√") and btn == "-":
        return "Err: Cannot take sqrt of negative number"
    if state == "" and btn == "-":
        return "Err: Cannot start with minus"
    if "√-" in state + btn or "√(-" in state + btn:
        return "Err: Cannot take sqrt of negative number"
    return None


# -------------------------------
# 2. Do the calculation
# -------------------------------
def do_calculate(state):
    """Extract operator, operands and compute result"""
    try:
        op = None
        for o in ["+", "-", "*", "/", "^2", "√"]:
            if o in state:
                op = o
                break

        if op is None:
            return "Err: Invalid expression"

        if op == "√":
            a = float(state.replace("√", ""))
            if a < 0:
                return "Err: Negative number under sqrt"
            return str(calculate(a, 0, "sqrt"))

        elif op == "^2":
            a = float(state.replace("^2", ""))
            return str(calculate(a, 0, "^2"))

        else:  # binary ops
            parts = state.split(op)
            if len(parts) != 2:
                return "Err: Invalid expression"
            a = float(parts[0].strip())
            b = float(parts[1].strip())
            return str(calculate(a, b, op))

    except Exception as e:
        return f"Err: {e}"


# -------------------------------
# 3. Handle special buttons
# -------------------------------
def handle_special_buttons(btn, state):
    """Handle C, Exit, ="""
    if btn == "C":
        return "", "", False     # reset state, reset display, keep running
    elif btn == "Exit":
        return state, "The application was stopped", True
    elif btn == "=":
        result = do_calculate(state)
        return state, result, False
    return None


# --------------------------------
# 4. Main click handler
# --------------------------------
def on_click(btn, state, stopped):
    # Map pretty labels to tokens
    btn = PRETTY_TO_TOKEN.get(btn, btn)

    if stopped:
        return state, gr.update(value="Stop the application",␣
 ↪elem_classes="display-err"), True

    # Validate input first
    err = validate_input(state, btn)
    if err:
        return state, gr.update(value=err, elem_classes="display-err"), False

    # Handle special buttons
    special = handle_special_buttons(btn, state)
    if special:
        s, val, stop = special
        elem = "display-err" if "Err" in val or "stopped" in val else␣
 ↪"display-num"
        return s, gr.update(value=val, elem_classes=elem), stop

    # Otherwise append new char
    new_state = state + btn
    return new_state, gr.update(value=new_state, elem_classes="display-num"),␣
 ↪False


# --------------------------------
# 5. Build UI
# --------------------------------
def launch_ui():
    with gr.Blocks(css="""
        .btn { width:60px !important; height:60px !important; font-size:18px;␣
 ↪flex:none !important; }
```

```
        .display-num textarea { width:400px !important; height:40px !important;␣
↪font-size:20px; text-align:right; }
        .display-err textarea { width:400px !important; height:40px !important;␣
↪font-size:18px; text-align:left; color:#ff5555; }
    """) as demo:
        gr.Markdown("###   Mini Calculator")

        state = gr.State("")
        stopped = gr.State(False)

        with gr.Row():
            with gr.Column(scale=0):
                display = gr.Textbox(
                    label="Result",
                    value="",
                    interactive=False,
                    lines=2,
                    max_lines=2,
                    elem_classes="display-num"
                )

        buttons = [
            ["7", "8", "9", "/"],
            ["4", "5", "6", "*"],
            ["1", "2", "3", "-"],
            ["0", ".", "=", "+"],
            ["√", "²", "C", "Exit"]
        ]

        for row in buttons:
            with gr.Row():
                for label in row:
                    btn = gr.Button(label, elem_classes="btn")
                    internal = PRETTY_TO_TOKEN.get(label, label)
                    btn.click(
                        on_click,
                        inputs=[gr.Textbox(value=internal, visible=False),␣
↪state, stopped],
                        outputs=[state, display, stopped],
                        show_progress=False
                    )

    demo.launch()


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

```
* Running on local URL:  http://127.0.0.1:7863
* To create a public link, set `share=True` in `launch()`.
```

```
<IPython.core.display.HTML object>
```



```
[ ]:
```