Module 11 - CALCULATOR USING TKINTER

In this module, we'll apply all the Tkinter concepts learned in Module 10 to build a complete and functional calculator application. We'll start with the user interface, then move on to implementing the core calculation logic.

Chapter 1: Calculator (1) - Setting up the GUI Structure

The first step in building any GUI application is to design and lay out its visual components. For a calculator, this involves the display screen and the number and operation buttons.

1.1 Review of Calculator GUI Components

A typical calculator GUI consists of:

- **Display Screen:** An `Entry` widget where numbers are input and results are shown.
- **Buttons:**
  - **Digits:** 0-9
  - **Operators:** +, -, *, /
  - **Control Buttons:** '=', 'Clear', (sometimes decimal point '.', +/-)

1.2 Why `grid()` is Ideal for Calculators

`grid()` is generally preferred for calculator-like UIs because:

- **Organized Structure:** Calculators naturally arrange buttons in rows and columns, which is exactly what `grid()` is designed for.
- **Responsiveness:** With `grid()`, you can configure rows and columns to expand proportionally when the window is resized, making your calculator more adaptable.
- **Maintainability:** Adding or rearranging buttons is easier when you're thinking in terms of `row` and `column` numbers rather than absolute `x` and `y` coordinates.

1.3 Initial Setup and Display Entry

First, let's set up the main window and the `Entry` widget that will serve as our calculator's display.

Python
```
# calculator_part1.py
import tkinter as tk

# Create the main application window
root = tk.Tk()
root.title("TuteDude Python Calculator") # Set the window title

# Configure a slightly larger window size
# We'll let the grid manage resizing, but this gives a good starting point
root.geometry("300x400") # Width x Height
```

```
# Create the Entry widget for the calculator display
# We'll span it across multiple columns
e = tk.Entry(root, width=35, borderwidth=5, font=('Arial', 16),
justify='right')
# Place it at row 0, spanning all 4 columns, and make it stick to all sides
e.grid(row=0, column=0, columnspan=4, padx=10, pady=10, sticky="nsew")

# Configure columns to expand proportionally when the window is resized
# This is crucial for a responsive grid layout
for i in range(4):
    root.grid_columnconfigure(i, weight=1)

# Configure the display row (row 0) to expand slightly
root.grid_rowconfigure(0, weight=1)

# Start the Tkinter event loop (nothing will happen yet without buttons)
# root.mainloop() # We'll uncomment this when we add buttons
```

Explanation:

- `tk.Entry(...)`: Creates the input/display field.
  - `width=35`: Sets the width in characters.
  - `borderwidth=5`: Adds a border around the entry.
  - `font=('Arial', 16)`: Makes the text larger and easier to read.
  - `justify='right'`: Aligns the text to the right, like a real calculator.
- `e.grid(row=0, column=0, columnspan=4, padx=10, pady=10, sticky="nsew")`:
  - `row=0, column=0`: Places the entry at the top-left of the grid.
  - `columnspan=4`: Makes it stretch across 4 columns (our typical calculator will have 4 columns of buttons).
  - `padx=10, pady=10`: Adds external padding.
  - `sticky="nsew"`: Makes the widget expand to fill its entire cell in all directions (North, South, East, West).
- `root.grid_columnconfigure(i, weight=1)`: This is vital for `grid()`'s responsiveness. It tells Tkinter that all 4 columns (0 to 3) should equally share any extra space if the window is resized horizontally.
- `root.grid_rowconfigure(0, weight=1)`: This makes the display row expand vertically.

1.4 Laying Out Buttons with `grid()`

Now, let's add the number buttons and basic operation buttons, arranging them neatly in a grid.

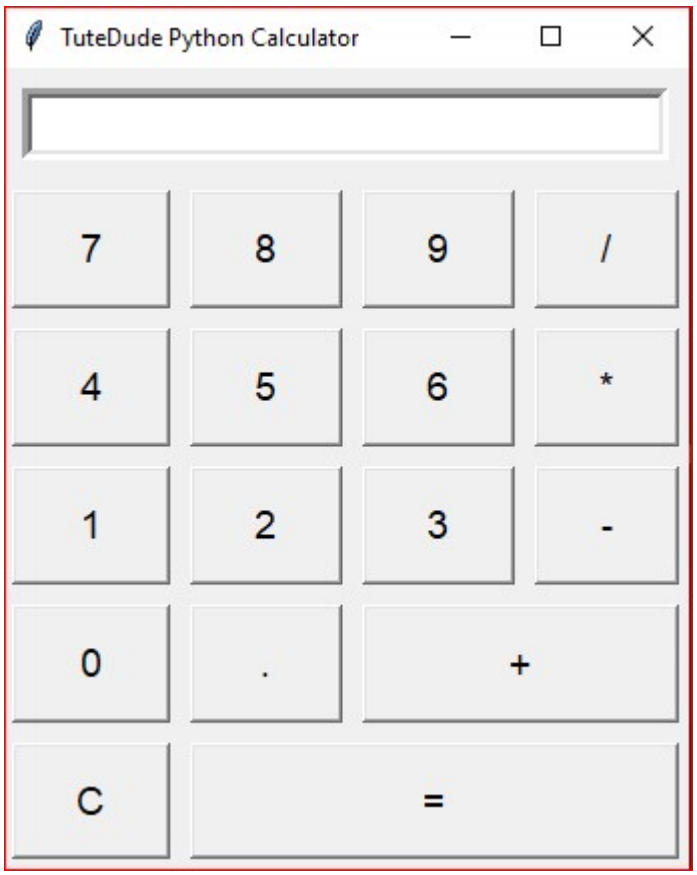
Python

```
# calculator_part1_full.py
import tkinter as tk

root = tk.Tk()
root.title("My Python Calculator")
root.geometry("300x400") # A reasonable starting size

# Configure rows and columns to be responsive
```

```
for i in range(4): # For 4 columns
    root.grid_columnconfigure(i, weight=1)
for i in range(1, 6): # For 5 rows of buttons (plus row 0 for display)
    root.grid_rowconfigure(i, weight=1)

# Entry for display
e = tk.Entry(root, width=35, borderwidth=5, font=('Arial', 16),
justify='right')
e.grid(row=0, column=0, columnspan=4, padx=10, pady=10, sticky="nsew")

# --- Define Buttons ---
# Button text and their corresponding grid positions (row, column)
buttons = [
    ('7', 1, 0), ('8', 1, 1), ('9', 1, 2), ('/', 1, 3),
    ('4', 2, 0), ('5', 2, 1), ('6', 2, 2), ('*', 2, 3),
    ('1', 3, 0), ('2', 3, 1), ('3', 3, 2), ('-', 3, 3),
    ('0', 4, 0), ('.', 4, 1), ('+', 4, 2), # '+' will span 2 columns
    ('C', 5, 0), ('=', 5, 1) # '=' will span 3 columns
]

# Create and place buttons dynamically
button_font = ('Arial', 14)

for (button_text, r, c) in buttons:
    # Create a button for each entry
    btn = tk.Button(root, text=button_text, font=button_font, padx=20,
pady=20) # Add some padding

    # Special handling for '+' and '=' buttons spanning multiple columns
    if button_text == '+':
        btn.grid(row=r, column=c, columnspan=2, padx=5, pady=5,
sticky="nsew") # Span 2 columns
    elif button_text == '=':
        btn.grid(row=r, column=c, columnspan=3, padx=5, pady=5,
sticky="nsew") # Span 3 columns
    else:
        btn.grid(row=r, column=c, padx=5, pady=5, sticky="nsew") # Regular
placement

root.mainloop()
```

What you will see (Calculator Part 1):

A window with a large input field at the top, and a well-organized grid of buttons for numbers and operators below it. The buttons will automatically adjust their size if you resize the window, demonstrating the power of `grid()` with `sticky="nsew"` and `weight` configurations.



At this stage, the calculator looks good, but the buttons don't do anything when clicked. That's the next step!

Chapter 2: Calculator (2) - Implementing Number and Clear Logic

Now, let's make the calculator interactive. We'll add functions to handle button clicks for numbers and the clear operation.

2.1 Managing Display Text with `StringVar`

Directly manipulating the `Entry` widget's content using `e.insert()` and `e.delete()` can work, but it's often cleaner and more robust to use a Tkinter **`StringVar`**.

- A `StringVar` is a special Tkinter variable that automatically synchronizes with the `Entry` widget it's linked to.
- When you change the `StringVar`, the `Entry` widget updates automatically, and vice-versa.

How to use `StringVar`:

1. Create an instance: `display_var = tk.StringVar()`

2. Link it to the `Entry` widget: `e = tk.Entry(..., textvariable=display_var)`
3. Set its value: `display_var.set("Some Text")`
4. Get its value: `current_text = display_var.get()`

2.2 `button_click` Function for Digits and Operators

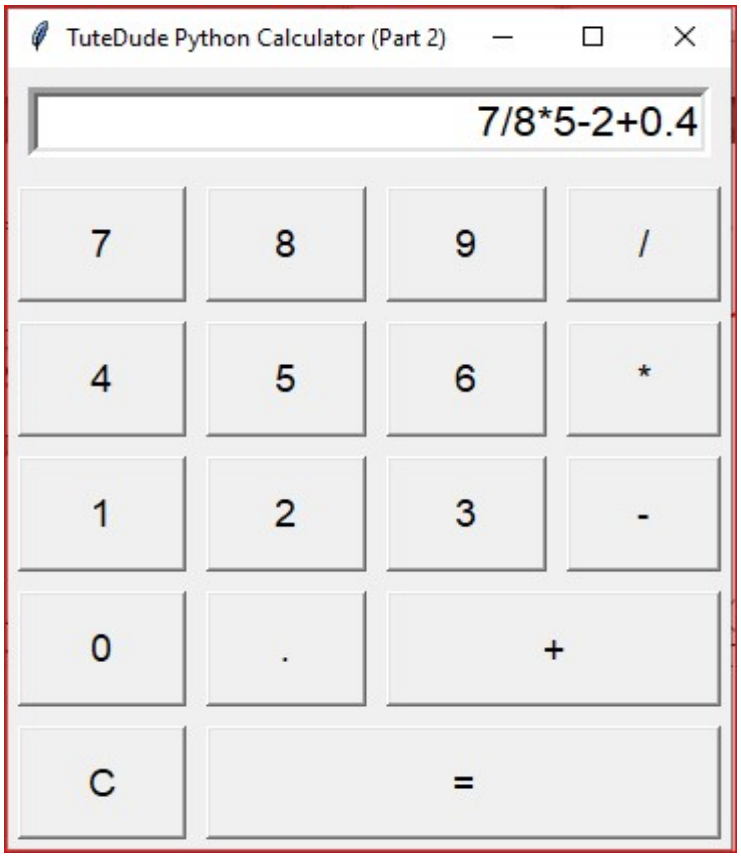
We need a function that will be called when a number or an operator button is pressed. This function will append the button's text to the display.

Python
```
# calculator_part2.py
import tkinter as tk

# Global variable to store the display content, managed by StringVar
display_var = None

def button_click(number):
    """Appends the clicked number/operator to the display."""
    current = display_var.get()
    display_var.set(current + str(number))

def button_clear():
    """Clears the calculator display."""
    display_var.set("")

root = tk.Tk()
root.title("TuteDude Python Calculator (Part 2)")
root.geometry("300x400")

# Configure rows and columns to be responsive
for i in range(4):
    root.grid_columnconfigure(i, weight=1)
for i in range(1, 6):
    root.grid_rowconfigure(i, weight=1)

# Create the StringVar and link it to the Entry widget
display_var = tk.StringVar()
e = tk.Entry(root, width=35, borderwidth=5, font=('Arial', 16),
justify='right', textvariable=display_var)
e.grid(row=0, column=0, columnspan=4, padx=10, pady=10, sticky="nsew")

# --- Define Buttons and place them dynamically ---
buttons_data = [
    ('7', 1, 0), ('8', 1, 1), ('9', 1, 2), ('/', 1, 3),
    ('4', 2, 0), ('5', 2, 1), ('6', 2, 2), ('*', 2, 3),
    ('1', 3, 0), ('2', 3, 1), ('3', 3, 2), ('-', 3, 3),
    ('0', 4, 0), ('.', 4, 1), ('+', 4, 2),
    ('C', 5, 0), ('=', 5, 1)
]

button_font = ('Arial', 14)

for (button_text, r, c) in buttons_data:
    btn = None # Initialize btn to ensure it's defined

    if button_text == 'C':
        btn = tk.Button(root, text=button_text, font=button_font, padx=20,
pady=20,
                        command=button_clear) # Link 'C' to clear function
```

```
        btn.grid(row=r, column=c, padx=5, pady=5, sticky="nsew")
    elif button_text == '=':
        # We will implement '=' logic in Part 3
        btn = tk.Button(root, text=button_text, font=button_font, padx=20,
pady=20)
        btn.grid(row=r, column=c, columnspan=3, padx=5, pady=5,
sticky="nsew")
    elif button_text == '+':
        # Special handling for '+' spanning 2 columns
        btn = tk.Button(root, text=button_text, font=button_font, padx=20,
pady=20,
                        command=lambda num=button_text: button_click(num))
        btn.grid(row=r, column=c, columnspan=2, padx=5, pady=5,
sticky="nsew")
    else:
        btn = tk.Button(root, text=button_text, font=button_font, padx=20,
pady=20,
                        command=lambda num=button_text: button_click(num))
# Link to button_click
        btn.grid(row=r, column=c, padx=5, pady=5, sticky="nsew")

root.mainloop()
```

What you will see (Calculator Part 2):

Now, when you click on any number button (0-9) or operator (+, -, *, /), its text will appear in the display entry. Clicking "C" (Clear) will clear the display. The "=" button still does nothing.



Chapter 3: Calculator (3) - Implementing Operation and Equals Logic

This is the most complex part, where we implement the core arithmetic. We'll need to store the first number, the operator, and then perform the calculation when the equals button is pressed.

3.1 Storing State for Calculations

To perform multi-step calculations, our calculator needs to remember:

- The **first number** entered by the user.
- The **operator** (+, -, *, /) selected.
- A flag or variable to indicate if an operator has been pressed and we're ready for the second number.

We'll introduce global variables to manage this state.

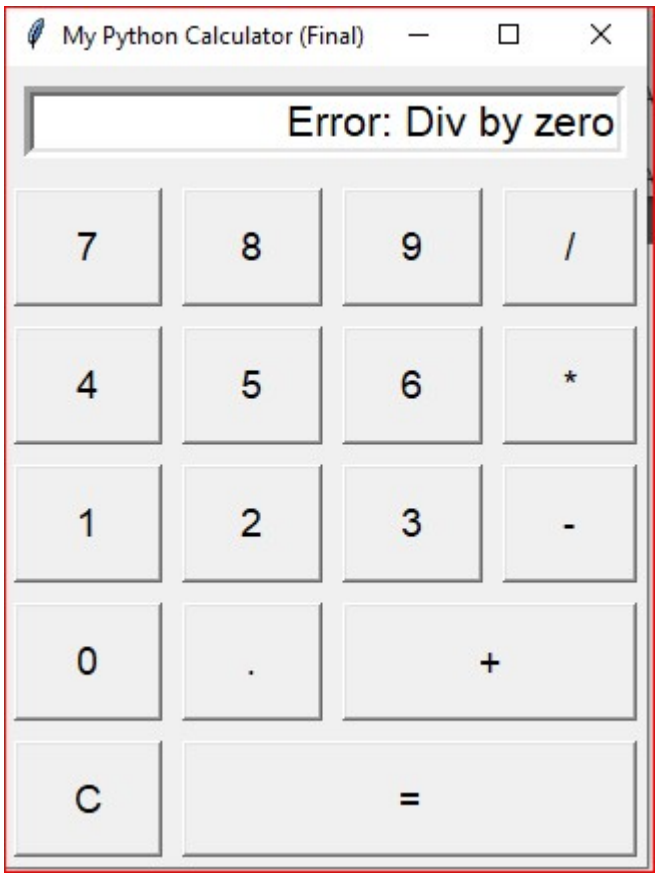
Python
```
# calculator_part3_final.py
import tkinter as tk

# Global variables to manage calculator state
display_var = None
first_num = None
operator = None

def button_click(number):
    """Appends the clicked number/operator to the display."""
    current = display_var.get()
    # Prevent multiple decimal points in one number
    if number == '.' and '.' in current:
        return
    display_var.set(current + str(number))

def button_clear():
    """Clears the calculator display and resets state."""
    display_var.set("")
    global first_num, operator
    first_num = None
    operator = None

def button_operation(op):
    """Handles operator (+, -, *, /) button clicks."""
    global first_num, operator
    try:
        first_num = float(display_var.get()) # Convert current display to
float
        operator = op                        # Store the operator
        display_var.set("")                  # Clear display for the next
number
    except ValueError:
        display_var.set("Error")
        first_num = None
        operator = None
```

```
def button_equals():
    """Performs the calculation when '=' is pressed."""
    global first_num, operator
    try:
        second_num = float(display_var.get()) # Get the second number
        display_var.set("") # Clear display for result

        if first_num is not None and operator is not None:
            result = 0
            if operator == '+':
                result = first_num + second_num
            elif operator == '-':
                result = first_num - second_num
            elif operator == '*':
                result = first_num * second_num
            elif operator == '/':
                if second_num == 0:
                    display_var.set("Error: Div by zero")
                    return
                result = first_num / second_num

            # Display result
            # Format to avoid too many decimal places for integers
            if result == int(result):
                display_var.set(str(int(result)))
            else:
                display_var.set(str(result))

            # Reset for next calculation
            first_num = None
            operator = None
        else:
            # If '=' pressed without valid prior operation
            display_var.set(second_num) # Just display the number entered

    except ValueError:
        display_var.set("Error")
    except Exception as e:
        display_var.set(f"Error: {e}")
    finally:
        first_num = None # Ensure state is reset
        operator = None

root = tk.Tk()
root.title("TuteDude Python Calculator (Final)")
root.geometry("300x400")

# Configure rows and columns to be responsive
for i in range(4):
    root.grid_columnconfigure(i, weight=1)
for i in range(1, 6):
    root.grid_rowconfigure(i, weight=1)

# Create the StringVar and link it to the Entry widget
display_var = tk.StringVar()
e = tk.Entry(root, width=35, borderwidth=5, font=('Arial', 16),
justify='right', textvariable=display_var)
e.grid(row=0, column=0, columnspan=4, padx=10, pady=10, sticky="nsew")

# --- Define Buttons and place them dynamically ---
```

```
buttons_data = [
    ('7', 1, 0), ('8', 1, 1), ('9', 1, 2), ('/', 1, 3, 'operator'),
    ('4', 2, 0), ('5', 2, 1), ('6', 2, 2), ('*', 2, 3, 'operator'),
    ('1', 3, 0), ('2', 3, 1), ('3', 3, 2), ('-', 3, 3, 'operator'),
    ('0', 4, 0), ('.', 4, 1), ('+', 4, 2, 'operator_long'), # '+' will span
2 columns
    ('C', 5, 0, 'clear'), ('=', 5, 1, 'equals') # '=' will span 3 columns
]

button_font = ('Arial', 14)

for item_data in buttons_data:
    button_text = item_data[0]
    r = item_data[1]
    c = item_data[2]
    button_type = item_data[3] if len(item_data) > 3 else 'number' #
Default to 'number'

    btn = None
    command_func = None
    column_span = 1

    if button_type == 'clear':
        command_func = button_clear
    elif button_type == 'equals':
        command_func = button_equals
        column_span = 3 # '=' spans 3 columns
    elif button_type == 'operator':
        command_func = lambda op=button_text: button_operation(op)
    elif button_type == 'operator_long': # For '+'
        command_func = lambda op=button_text: button_operation(op)
        column_span = 2 # '+' spans 2 columns
    else: # It's a number or decimal point
        command_func = lambda num=button_text: button_click(num)

    btn = tk.Button(root, text=button_text, font=button_font, padx=20,
pady=20,
                    command=command_func)
    btn.grid(row=r, column=c, columnspan=column_span, padx=5, pady=5,
sticky="nsew")

root.mainloop()
```

What you will see (Calculator Part 3 - Final):

A fully functional calculator! You can now type numbers, perform addition, subtraction, multiplication, and division, see results, and clear the display. Basic error handling for division by zero and invalid input is also included.


My Python Calculator (Final)
Error: Div by zero
7
8
9
/
4
5
6
*
1
2
3
-
0
.
+
C
=