Laboratory Activity No. 11	
The Grid Manager	
Course Code: CPE103	**Program:** BSCPE
Course Title: Object-Oriented Programming	**Date Performed:** 03/05/2025
Section: A1	**Date Submitted:** 03/05/2025
Name: Ampong, J-kevin L.	**Instructor:** Engr. Maria Rizette Sayo

1. Objective(s):

This activity aims to familiarize students on how to implement geometry manager

2. Intended Learning Outcomes (ILOs):

The students should be able to:
2.1 Identify the main components in a GUI Application
2.2 Create a simple GUI Application using Grid manager

3. Discussion:

A Graphical User Interface (GUI) application is a program that the user can interact with through graphics (windows, buttons, text fields, checkboxes, images, icons, etc..) such as the Desktop GUI of Windows OS by using a mouse and keyboard unlike with a Command-line program or Terminal program that support keyboard inputs only.

Geometry managers are tools used to place widgets on the screen. There are three geometry managers available in tkinter—grid, pack, and place. The place manager provides complete control in the positioning of widgets, but is complicated to program

Grids

- A grid is an imaginary rectangle containing horizontal and vertical lines that subdivide it into rectangles called cells. The first row of cells is referred to as row 0, the second row is referred to as row1, and so on. Similarly, the first column of cells is referred to as column 0, the second column of cells is referred to as column 1, and so on. Each cell is identified by its row and column numbers.

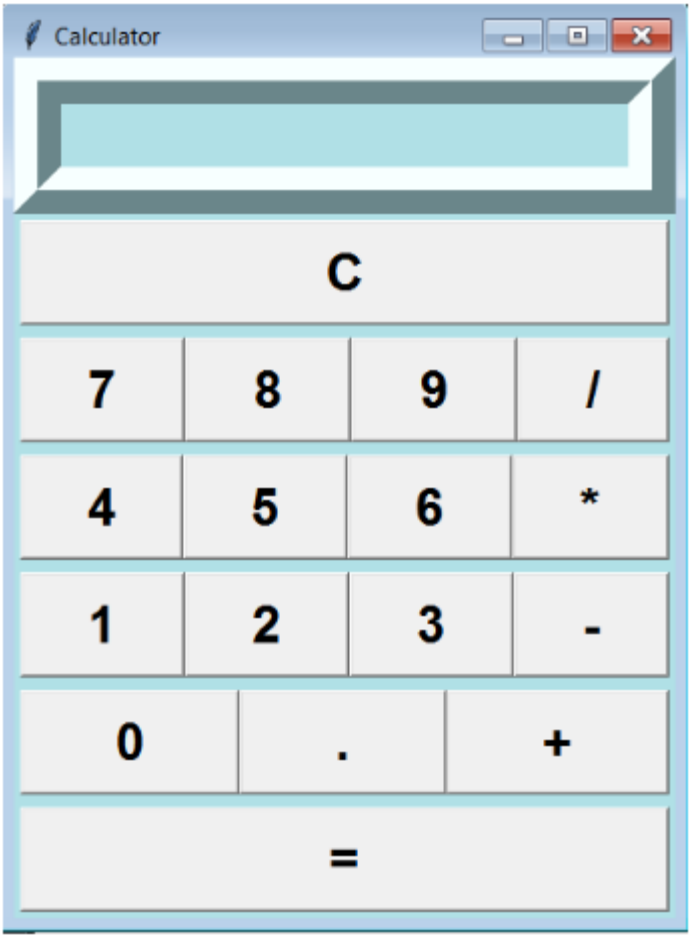
4. Materials and Equipment:

Desktop Computer with Pycharm
Windows Operating System

5. Procedure:

General Instruction:

1. Redesign the interface of the standard calculator using grid ( ) method:



2. Run the program and observe the output when the button is clicked.

Please refer to this link: https://github.com/AmpongJkevin2/CPE-103-OOP-1-A/blob/main/lab11/part1_redesign.py

6. Supplementary Activity:

1. Make a calculator program that can compute perform the Arithmetic operations as well as exponential operation, sin, cosine math functions as well clearing using the C button and/or clear from a menu bar.
2. Use Geometry manager grid()

3. Use bind () or command parameter in associating event to callback a function.

Please refer to this link:

https://github.com/AmpongJkevin2/CPE-103-OOP-1-A/blob/main/lab11/supplementary.py

Questions

1. How do you configure rows and columns in PyCharm when using Tkinter's grid() manager?
   You set up rows and columns for grid() right in your Python code. Just use .grid(row=..., column=...) when you place a widget. If you want rows or columns to stretch when you resize the window, you need to tell the *parent* window or frame using .rowconfigure(index, weight=1) or .columnconfigure(index, weight=1) – it's not something you click in PyCharm itself.

   _____

   _____

2. Why do widgets sometimes disappear when using grid() in PyCharm, and how can you fix it?
   Well, widgets vanishing with grid() usually happens because you accidentally mixed .grid() with .pack() or .place() in the same area – Tkinter gets totally confused then! Stick to just one method for widgets inside the same frame or window. Sometimes, you might just forget to actually *call* .grid() on a widget you made, or you left out the final window.mainloop() line.

   _____

   _____

3. How can message boxes be used to provide a better User Experience or how can message boxes be used to make a GUI Application more user-friendly?
   How can you align widgets across multiple frames using grid() in PyCharm? Message boxes are super handy for giving the user quick feedback, like showing an error message (showerror), asking "are you sure?" (askyesno), or just saying "Saved!" (showinfo), keeping your main window clean. For lining things up across different frames, the trick is to put related widgets (like a label and its input box) in the *same column number* inside their own frames, and then make sure those frames themselves line up nicely in the main window grid, maybe giving columns the same weight so they resize together.

7. Conclusion:

This activity gave us hands-on practice with Tkinter's grid() layout manager. We learned how to arrange buttons and displays in rows and columns, control their spacing with padding, make them span multiple cells, and even use weights to help the layout resize more predictably. It definitely shows how grid() offers more precise control for creating structured interfaces like a calculator compared to simpler methods.

Building on the layout, we then added the calculator's brain, connecting button clicks to Python functions using the command option. This involved handling user input, performing calculations with the math module, and using message boxes for important feedback like errors. We also explored how basic color and widget options in Tkinter can be used to apply different visual themes, like dark mode or a simpler flat style, improving the overall look and feel without needing extra libraries.

8. Assessment Rubric: