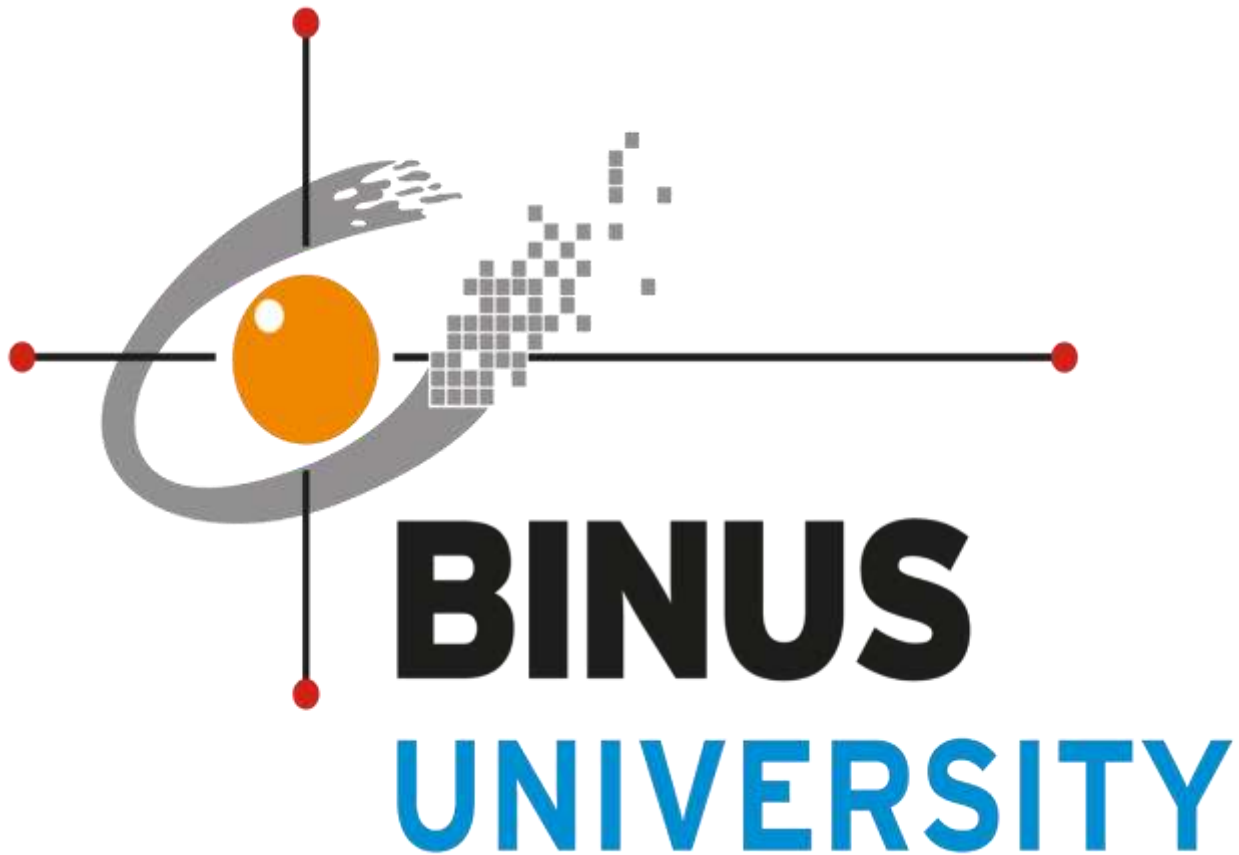


## Specifications For 1<sup>st</sup> Semester's Program Design Methods Final Project



<b>Lecturer</b>	<b>: Ida Bagus Kerthyayana Manuaba, S.T., Ph.D.</b>
<b>Lecturer-ID</b>	<b>: D5757</b>
<b>Class Code</b>	<b>: L1BC</b>
<b>Assignment Type</b>	<b>: 1<sup>st</sup> Semester's PDM Final Project</b>
<b>Student</b>	<b>: Edward Kevin Winata</b>
<b>Student-ID</b>	<b>: 2440081614</b>
<b>E-Mail</b>	<b>: <a href="mailto:edward.winata@binus.ac.id">edward.winata@binus.ac.id</a></b>
<b>Student Phone Number</b>	<b>: 087781080099</b>

## Report Contents

<b>Introductory</b>	.....	<b>3</b>
<b>Project Purpose</b>	.....	<b>4</b>
<b>Design</b>	.....	<b>5</b>
<b>Code Breakdown</b>	.....	<b>7</b>

## Introductory

During the course of my learning experience in Computer Science Major at Binus University International, me and my other classmates were introduced to a very popular programming language that has been around for quite some time, but due to its simplicity, it has been retaining its popularity and major developments by many developers out there in the world. This special and unique programming language is called “Python” and it was designed to be highly beginner friendly for starters, beginners or for some other people who would like to revise and sharpen their skills in programming. Python implements short and clear syntaxes which is very comfortable to learn and work with, for mostly others and myself as well. What I love most about python is that it has a large collection of packages used to manipulate and visualize any form of data given to it. I also have been learning some complex user defined data structures and tried to implement some of them in python lately, such as linked list and trees.

What I have learned throughout the first semester profoundly have assisted myself really well about the basics of the syntax in python programming language and allowed myself to develop a more complex pattern of logical thinking which will prove to be very useful in learning other languages which offers significantly more complicated syntax, and also to be able to develop a project made entirely from python programming language, along with the use of external packages to increase the ability of code to interpret more functionality.

For this final project which was intended to increase logical thinking ability and to understand more of python’s wide collection of features, I decided to develop a scientific mannered calculator intended to operate on some complex expressions which is not supported in most basic calculators. This calculator offers much more scientific operators and some of the most commonly used mathematical constants. In order to include more features and to increase the functionality of the scientific calculator, two other functionalities are also included which will be further explained in Project Purpose. As for the whole structure of completed project, it can be accessed from my GitHub repository with the link below :

<https://github.com/horbosis556/Scientific-Calculator->

## Project Purpose

This project was created and meant to be used for users who might have had some problems when dealing with complex mathematical operations or expressions that they cannot handle simply by relying on calculating manually in the mind. This is the main reason why calculators were invented. In my case, my scientific calculator is created to handle them to the point where the user will be able to achieve instant results directly after an expression is given to the calculator entry and evaluate button is clicked.

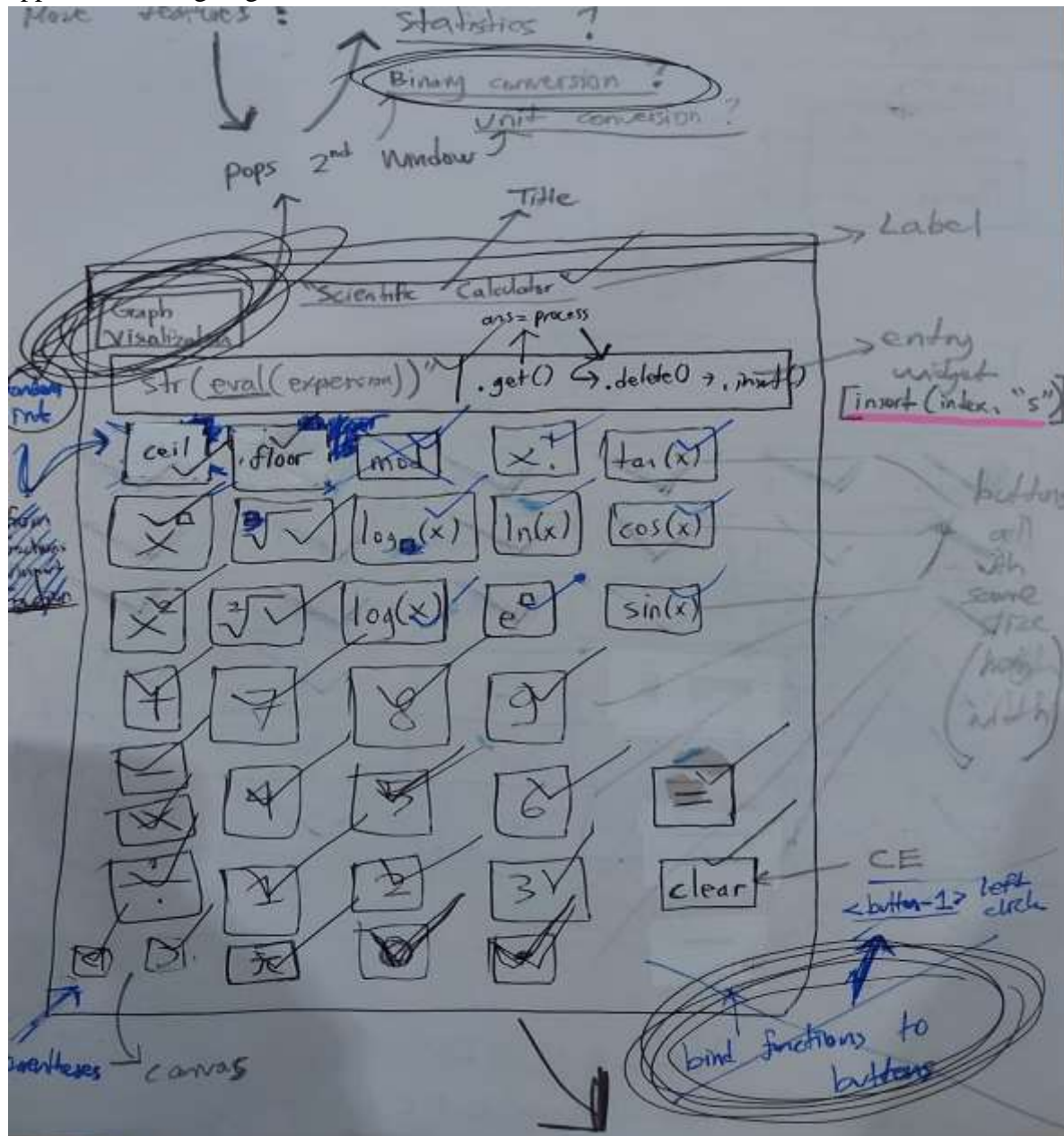
My Scientific Calculator offers these operators for the user to choose from :

1. Floor operator.
2. Ceiling operator.
3. Insert random integer.
4. Insert pi  $\rightarrow$  Constant.
5. Insert euler  $\rightarrow$  Constant.
6. Insert tau  $\rightarrow$  Constant.
7. Absolute Value operator.
8. Negation operator.
9. Square operator.
10. Powered to x operator.
11. Square and Cube Roots.
12. Modulus.
13. Logarithms with base-10, base-2 and base-euler.
14. Factorial.
15. Exp operator.
16. Sine, Cosine, Tangent, Cosecant, Secant, Cotangent.
17. ArcSine, ArcCosine, ArcTangent.
18. Parentheses to control operation order.
19. Decimal point to create floating point numbers.
20. Basic operators(+, -,  $\times$ ,  $\div$ ).

In addition to them, There are also two sub-menu widgets I created to act as access points towards two features which I included in my calculator app. The first one will allow users to visualize trigonometric graphs and locate a specific angle or a special one and map it to the value of a trigonometric operator being applied to it. Whereas the second sub-menu will allow users to convert an integer number up to the ranges in int64(unsigned) and the calculator will return its value in standard computer numeral system(BIN, OCT, HEX). The user could then copy these values from the entry by pressing CTRL + C, and use them somewhere else simply by pasting(CTRL + V).

## Design

I started from scratch by first creating some sketches regarding the visualization of my calculator app (e.g. how it is going to look like, positions of button and entries, where to place feature access point, and many others.). After I am done, this picture below is how my calculator application was going to be assembled :



This sketch was made a week after i finished researching Tkinter and PyQt5 Packages, which were used to create GUI's. The check marks were given in every sketched button above for me to mark when I have understood how to write a proper code to display them.

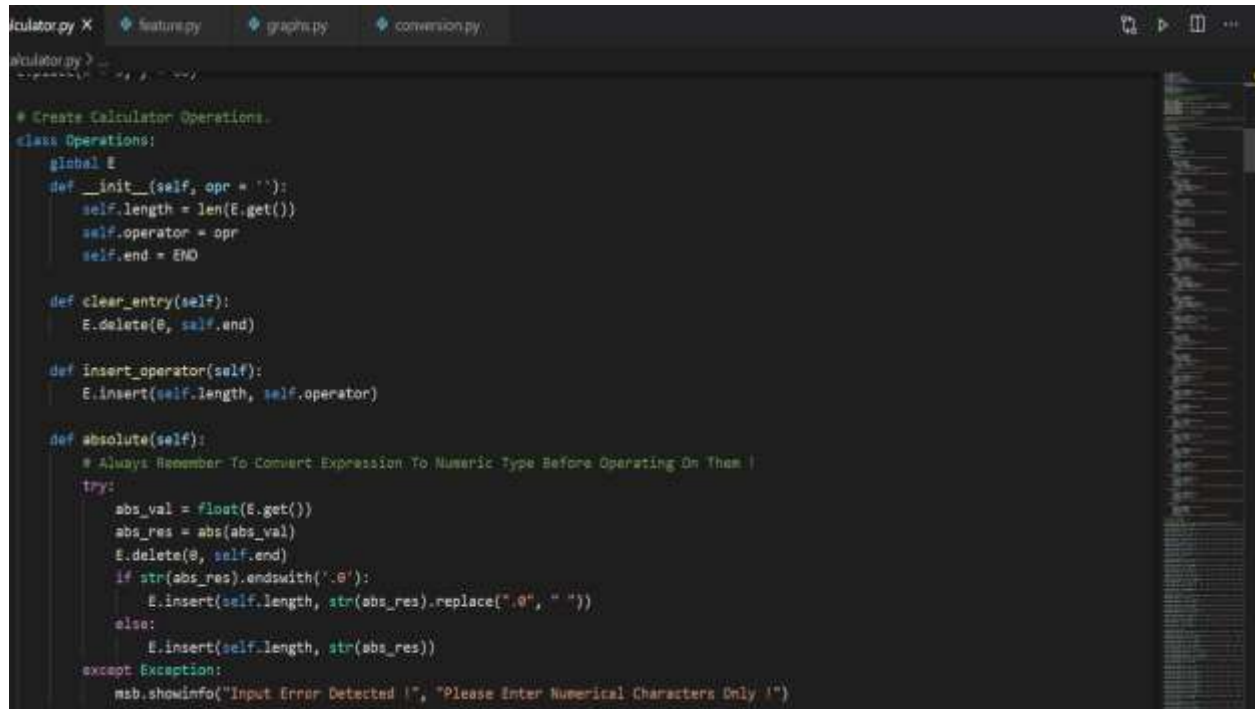
## Code Breakdown

The IDE that I used to develop this calculator project is Visual Studio Code and started off by creating and activating a virtual environment to contain the package used for the project. This is what my .Venv package contains :

Package	Version
-----	-----
appdirs	1.4.4
astroid	2.4.2
attrs	20.3.0
cffi	1.14.4
colorama	0.4.4
cycler	0.10.0
distlib	0.3.1
filelock	3.0.12
isort	5.6.4
kiwisolver	1.3.1
lazy-object-proxy	1.4.3
matplotlib	3.3.3
mccabe	0.6.1
mpmath	1.1.0
numpy	1.19.3
pandas	1.1.5
Pillow	8.0.1
pip	20.3.3
pycparser	2.20
pylint	2.6.0

- I was going to use SymPy and Mpmath to plot inverse trigonometric functions(Csc, Sec, Cot) but was not able to, since these functions contain invalid trigonometric values, ZeroDivisionError was raised. I had to use  $1 / \text{np.sin}()$ ,  $1 / \text{np.cos}()$  and  $1 / \text{np.tan}()$  instead, strangely this solves the issue and the graph was created.

For the main calculator app, I created a class called Operations and initialized entry length, operator or constant to be inserted to the entry and TkInter END position to tell Tkinter to clear entry widget all the way to the last character when the user clears the entry. After that, what the rest of the other functions do is to insert operators / constants into the entry :

A screenshot of a code editor with a dark theme. The editor has three tabs at the top: 'calculator.py X', 'feature.py', and 'graph.py'. The 'calculator.py' tab is active. The code is written in Python and defines a class named 'Operations'. The class has an attribute 'E' and methods: '\_\_init\_\_', 'clear\_entry', 'insert\_operator', and 'absolute'. The 'absolute' method includes a try-except block to handle non-numeric input. The code is as follows:

```
calculator.py X feature.py graph.py conversion.py
calculator.py > ...
# Create Calculator Operations.
class Operations:
    global E
    def __init__(self, opr = ''):
        self.length = len(E.get())
        self.operator = opr
        self.end = END

    def clear_entry(self):
        E.delete(0, self.end)

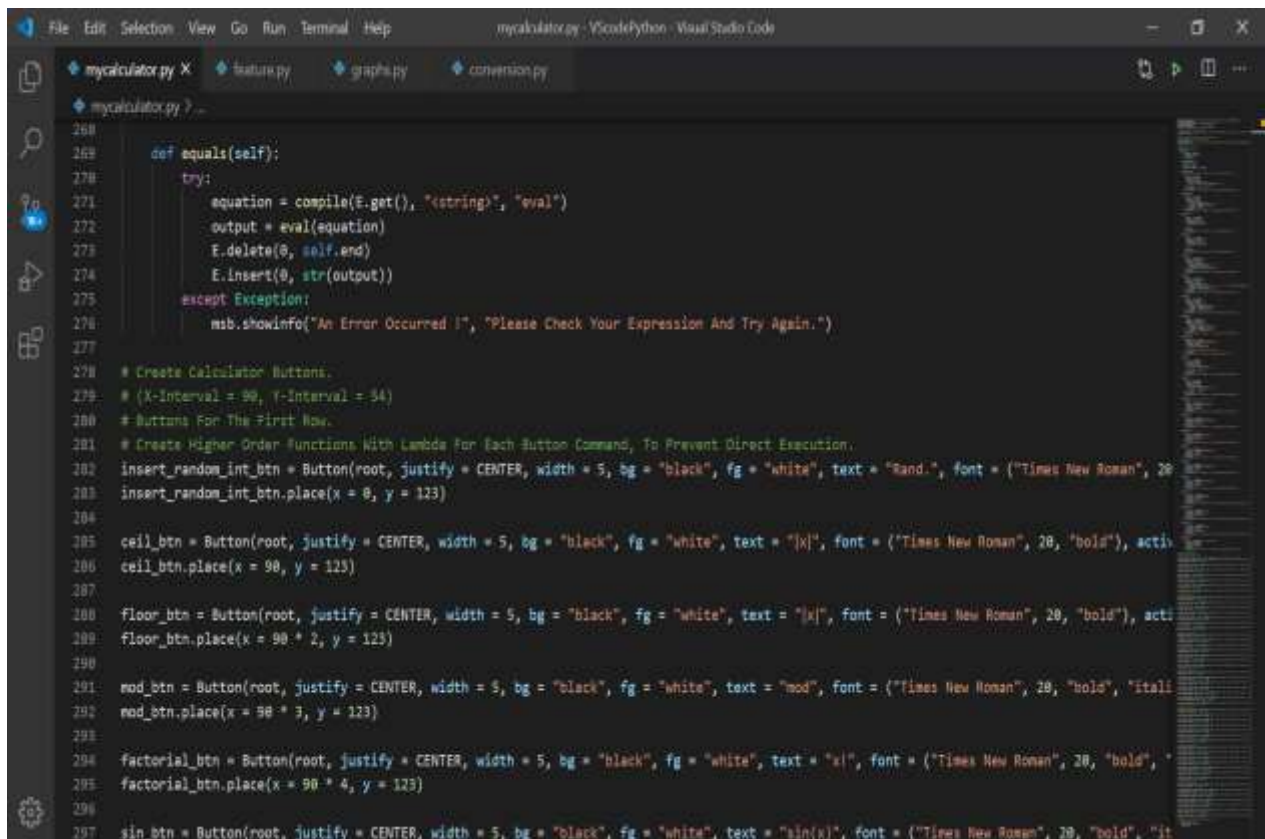
    def insert_operator(self):
        E.insert(self.length, self.operator)

    def absolute(self):
        # Always Remember To Convert Expression To Numeric Type Before Operating On Them !
        try:
            abs_val = float(E.get())
            abs_res = abs(abs_val)
            E.delete(0, self.end)
            if str(abs_res).endswith('.0'):
                E.insert(self.length, str(abs_res).replace(".0", " "))
            else:
                E.insert(self.length, str(abs_res))
        except Exception:
            msb.showinfo("Input Error Detected !", "Please Enter Numerical Characters Only !")
```

The reason why i casted data type “float” to every expression is because originally, it is a string based on TkInter’s StringVar(), and if I casted int() instead, the calculator will work on only integers and not floats.

I came out with a logic to replace floats ending with a zero after the decimal point with a whitespace, so that if a float ends with “.0” then it is replaced with a whitespace, essentially turning every float ending with a ‘.0’ to an integer.





```
268
269     def equals(self):
270         try:
271             equation = compile(E.get(), "<string>", "eval")
272             output = eval(equation)
273             E.delete(0, self.end)
274             E.insert(0, str(output))
275         except Exception:
276             msb.showinfo("An Error Occurred !", "Please Check Your Expression And Try Again.")
277
278     # Create Calculator Buttons.
279     # (X-Interval = 90, Y-Interval = 34)
280     # Buttons For The First Row.
281     # Create Higher Order Functions With Lambda For Each Button Command, To Prevent Direct Execution.
282     insert_random_int_btn = Button(root, justify = CENTER, width = 5, bg = "black", fg = "white", text = "Rand.", font = ("Times New Roman", 20
283     insert_random_int_btn.place(x = 0, y = 123)
284
285     ceil_btn = Button(root, justify = CENTER, width = 5, bg = "black", fg = "white", text = "⌈x⌉", font = ("Times New Roman", 20, "bold"), acti
286     ceil_btn.place(x = 90, y = 123)
287
288     floor_btn = Button(root, justify = CENTER, width = 5, bg = "black", fg = "white", text = "⌊x⌋", font = ("Times New Roman", 20, "bold"), acti
289     floor_btn.place(x = 90 * 2, y = 123)
290
291     mod_btn = Button(root, justify = CENTER, width = 5, bg = "black", fg = "white", text = "mod", font = ("Times New Roman", 20, "bold", "Itali
292     mod_btn.place(x = 90 * 3, y = 123)
293
294     factorial_btn = Button(root, justify = CENTER, width = 5, bg = "black", fg = "white", text = "x!", font = ("Times New Roman", 20, "bold", "
295     factorial_btn.place(x = 90 * 4, y = 123)
296
297     sin_btn = Button(root, justify = CENTER, width = 5, bg = "black", fg = "white", text = "sin(x)", font = ("Times New Roman", 20, "bold", "it
```

The method equals() compiles the expression based of a string variable to be evaluated. It is then passed to the eval() function. I did this to increase eval function's capability so that it does not throw Exception when it evaluates an expression containing math functions.

I was going to use the grid method to place buttons in a more simple manner, the output is just not like what it needs to be. Only the place() method can place a widget right in the exact coordinates which can be controlled by the coder, unlike where only sides(TOP, LEFT, RIGHT, BOTTOM, CENTER) can be modified and not exact coordinates.

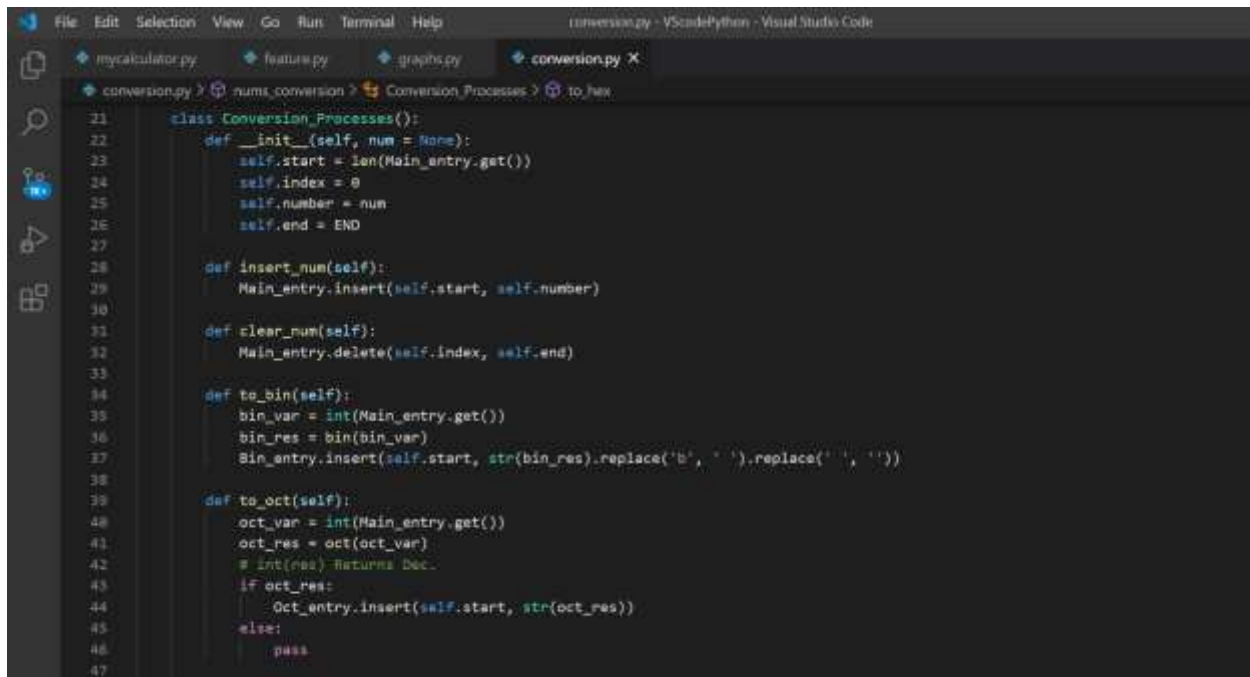
## Feature 1 Trigonometric Graphs

```
mycalculator.py  feature.py  graphs.py X  conversion.py
graphs.py > plot_sin
1  from tkinter import *
2  import math
3  import numpy as np
4  import matplotlib.pyplot as plt
5  from matplotlib.figure import Figure
6  # Matplotlib With Tkinter Backend.
7  from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg, NavigationToolbar2Tk
8
9  def plot_sin():
10     # Tkinter Window For The Graph.
11     win = Tk()
12     win.title('Graph')
13     win.geometry("500x500")
14
15     # Figsize = (5x5)inches, rads -> (0 <= rads <= 2π), vals -> (-1 <= vals <= 1)
16     # Use Default Dots Per Inch(100).
17     x = np.arange(0, math.pi*2, 0.05)
18     fig = plt.figure()
19     axe = fig.add_axes([0.1, 0.1, 0.8, 0.8])
20     y = np.sin(x)
21     axe.plot(x, y, 'r')
22     axe.grid(True) # I Added Grid And Color To Each Graph, For The Eyes To Read Them Easier.
23     axe.set_xlabel('Angles')
24     axe.set_title('Y = Sin(x)')
25     # axe.set_xticks() Provides Accurate Distance Measurement Between Each Quadrant.
26     axe.set_xticks([0, 1.5, 3.1, 4.7])
27     axe.set_xticklabels(['(0° To 90°)', '(90° To 180°)', '(180° To 270°)', '(270° To 360°)'])
28     axe.spines['left'].set_color('red')
```

I wanted each graph to be displayed in a TkInter Canvas inside a Tkinter window, so I set the backend of matplotlib = "tkAgg", inserting the figure to tk.Tk() using FigureCanvasTkAgg and added a toolbar from NavigationToolbar2Tk which allows users to copy / save the graph. Colors for y-axis and the plot line in each graph were also added, randomized for each type of graph.

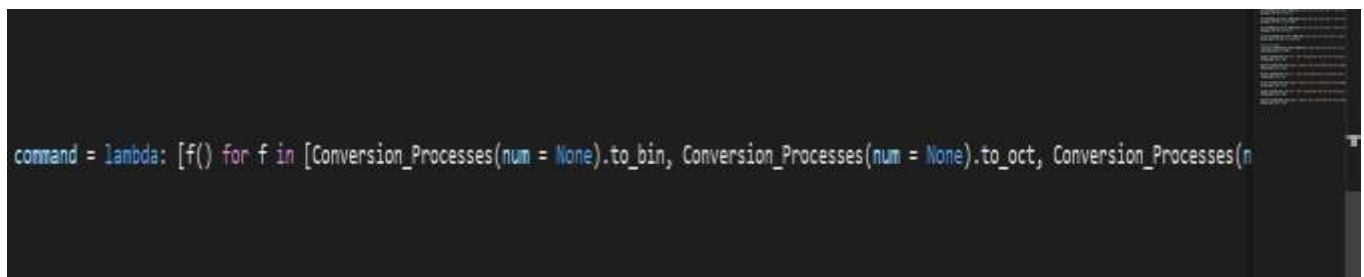
## Feature 2

### Number System Conversion



```
21 class Conversion_Processes():
22     def __init__(self, num = None):
23         self.start = len(Main_entry.get())
24         self.index = 0
25         self.number = num
26         self.end = END
27
28     def insert_num(self):
29         Main_entry.insert(self.start, self.number)
30
31     def clear_num(self):
32         Main_entry.delete(self.index, self.end)
33
34     def to_bin(self):
35         bin_var = int(Main_entry.get())
36         bin_res = bin(bin_var)
37         Bin_entry.insert(self.start, str(bin_res).replace('b', ' ').replace(' ', ''))
38
39     def to_oct(self):
40         oct_var = int(Main_entry.get())
41         oct_res = oct(oct_var)
42         # int(res) Returns Dec.
43         if oct_res:
44             Oct_entry.insert(self.start, str(oct_res))
45         else:
46             pass
47
```

I also created a class for this feature in order to ease the process of converting the number into BIN, OCT and HEX systems. 1 button click will execute 3 functions, first one converts the integer to Binary value, removes the char 'b' and any spaces. The second converts the integer to Octal, the third one to Hexadecimal.



```
command = lambda: [f() for f in [Conversion_Processes(num = None).to_bin, Conversion_Processes(num = None).to_oct, Conversion_Processes(n
```

## Screenshot Of Visualization

