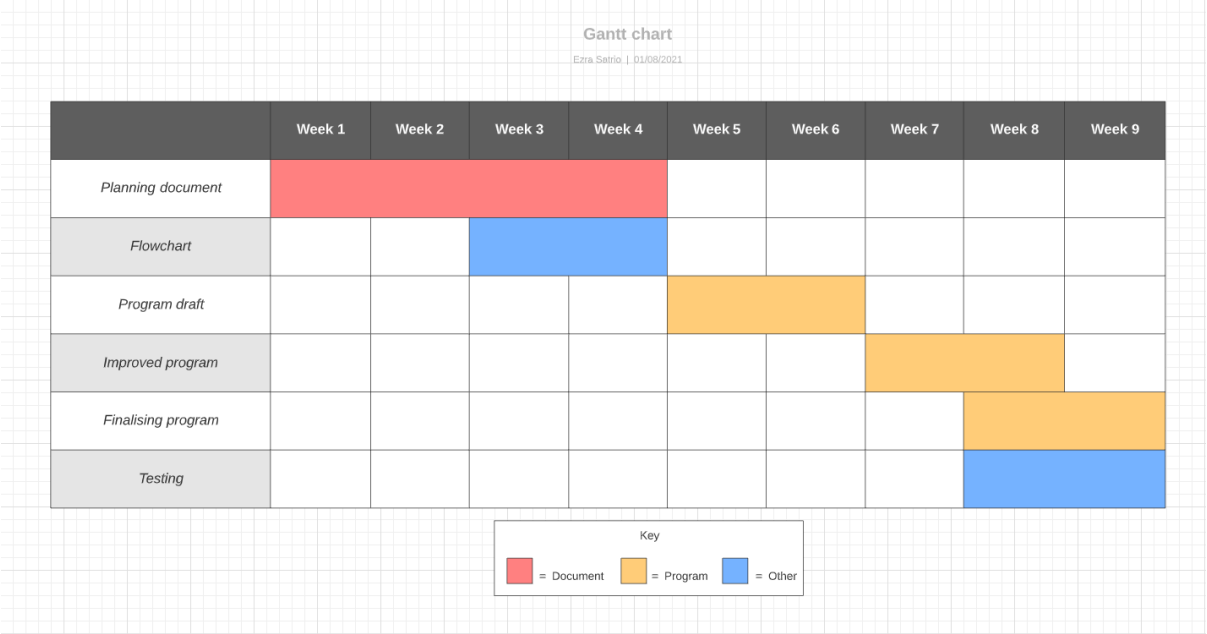
AS91907

Brief: Create a program that utilises a Graphical User Interface (GUI) to perform a task for the user. In this case, I will be creating the functions of a scientific/graphics calculator for desktop use. The program will be able to display a calculator interface, and lets the user interact to perform arithmetic equations as they please. This program is more directed towards high-school students, who will require scientific calculators throughout their years in high-school. Although, this calculator can be just as easily used for an adult seeking for a way to calculate complex arithmetics.

Timeline:



Planning: Due 15th September
Program: Due 22nd September
(Note: Dates vary because of lockdown)

The planning will be finished before the program is started. Due to the recent lockdown, the due date has been extended, and it is unclear whether the time restraints will affect the timeline and the end product. For now, the same schedule still applies.

Specifications:
For a calculator to be properly useful for complex arithmetics, it requires the ability to use complex arithmetic functions such as roots, powers, logs, etc. The calculator also needs an easy-to-use interface so users will be able to adapt to its UI quickly without difficulty.

Program specifications:

The program will be written in Python 3, and requires several external modules that will allow the calculator to use complex arithmetic functions. This means that the user will most likely require Python 3 installed on their computer,and the math module for Python. Since the math module already comes with the Python package by default, it is very unlikely that the user requires installation of the math module. However, in the rare case that the user is missing the math module, the command *'pip install python-math'* can be used to install the module, although if this occurs it's much better to re-install Python 3, as this indicates some missing in-built modules.

The Tkinter module is also required to properly provide the Graphical User Interface (GUI) for the program to provide a user interface so the user can interact with the program. This module also comes with Python, however in the rare case it is not, the command *'pip install tk'* can be used to install the module. A guide on the installation of both modules, and Python 3, will be provided to the user in the user manual.

If possible I will attempt to convert my finished program in an executable file using Pyinstaller, so that it can be used without requiring Python or any modules installed on the user's computer, making it much simpler to use and access (this depends on final product)

Calculator specifications:
A scientific calculator possesses many more complex arithmetics than a standard calculator. These functions will have to be planned beforehand to ensure that the program can blend all these functions together smoothly. To do this, I will model my program to follow the functions of a Casio graphics calculator.

The GUI will appear similar to a handheld calculator to provide the most similar UI that is comfortable to most users. The user will be able to interact with the GUI by clicking buttons, or typing directly into the calculator.

Planned functions:

Basic operators: ÷, ✖, +, -	Basic operators include multiplication, division, addition, subtraction, like any other calculator.
BEDMAS	The ability to chain several operations together and compute them according to the BEDMAS rule (e.g. *(5+2)*(4/2)*).
Screen clear	The ability to clear the screen for the next calculation.

Delete	The ability to delete single characters from the screen in the case of mis-inputs.
Multi-line outputs	The calculator will provide outputs in multiple lines, as opposed to a standard calculator that outputs in single lines, allowing the user to view past calculations.
Answer storage	The ability to store the previous answer from a calculation, and use that value in future calculations.
Variable storage	The ability to store values in variables, which can then be used in future equations. Variables use letters as symbols, and range from letters A to Z. Storing the values into a variable requires a specific function as well.
Decimal to fraction to percentage (and vice versa)	The ability to convert the output as both a decimal and fraction output.. Useful for students as they sometimes have to give their answer in a certain form.
Numerical constants (Pi, Euler)	The calculator will need arithmetic constants that are used in complex mathematics, such as Pi and Euler. If possible, I will try to add physics constants as well, although this is mostly unnecessary, time-consuming, and probably won't be used.
Powers and roots	The ability to perform powers and roots operations. This also includes negative, fractional, decimal indices.
Logarithms	The ability to perform logarithmic equations, to different bases (including natural) if possible.
Angular calculations	The ability to use sin, cos, tan to perform equations, in both radians and degrees. Can also do inverse.

Factorial	The ability to calculate factorials through a single digit input (e.g. ‘5 = 5!’)
Help menu	A menu that provides the user a guide on how to use the different functions. Very helpful to the user, so it is necessary.
Error (Number error, Limit error, Stack error)	An error message that pops up on the calculator display (not on the Python interpreter) if the user tries a mathematically impossible calculation, or uses an incorrect input.

The following functions are planned, but will take more time, and if I run out of time it is likely they won’t be implemented.

Quadratics and cubics	The ability to compute quadratics and solve equations with quadratics and cubics.
~~Grapher~~	~~The ability to draw a graph from a given equation.~~
~~Multi-modes~~	~~Switch between modes to do different functions (e.g. Math mode performs basic operations, Graph mode draws graphs)~~
~~Statistics~~	~~The ability to use a table and input statistical data, which can then be used for other functions such as graphing~~
Navigator	The ability to let the user manually edit the equation from specific places, making it easier to fix input errors for the user, instead of clearing the screen.
Keyboard input	User can also use the keyboard to input characters in the calculator, which improves functionality and ease of use, as this program runs on a computer so using the keyboard is a must. Instead of simply clicking, which gets tiring.

Program steps:

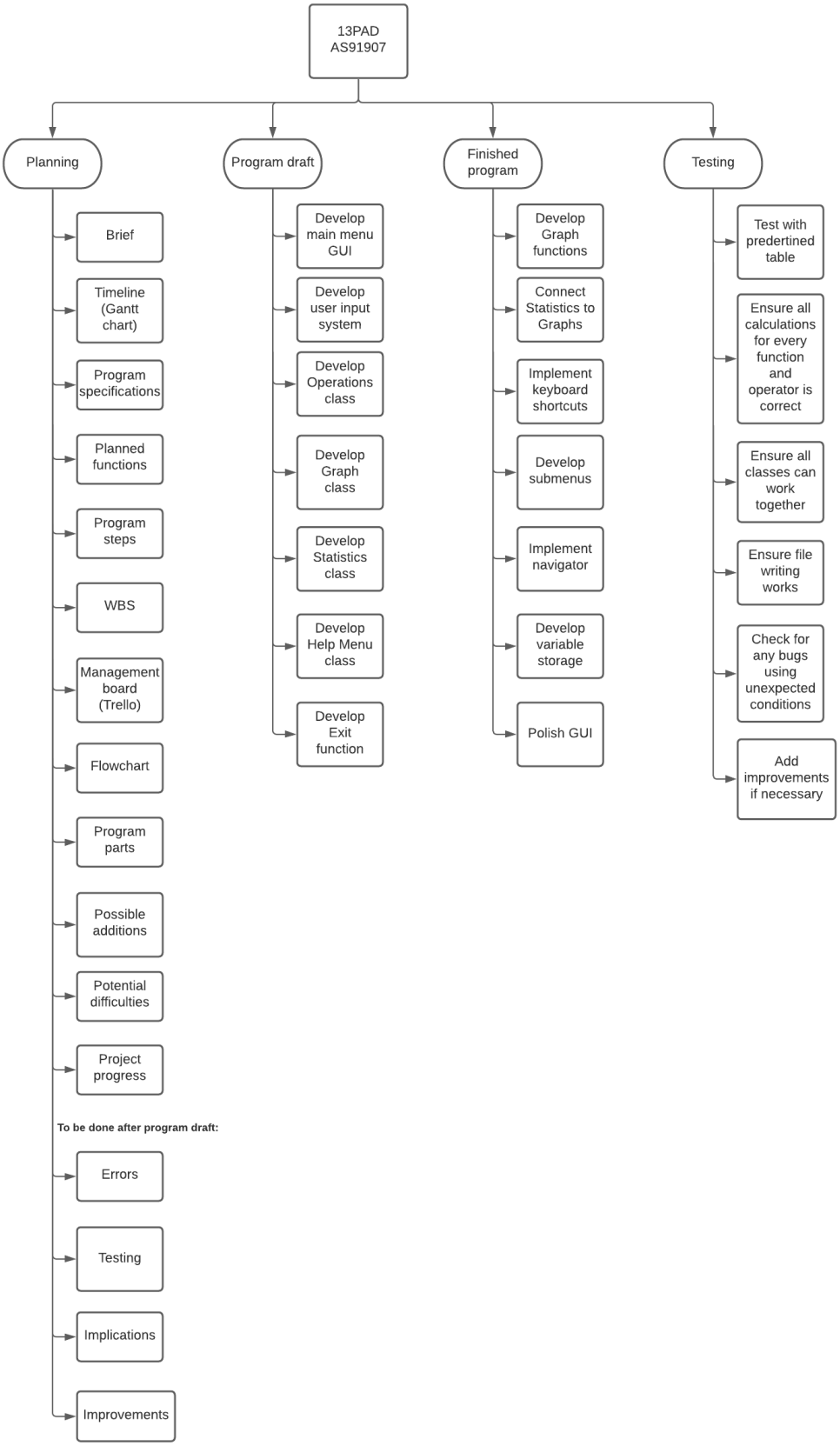
1. Welcome screen starts up
2. Option menu displaying the 4 options (Operations, graphs, Recall, Help) along with help menu
3. Wait for user to pick a menu option
4. Depending on the chosen option, the program will engage the corresponding class
    - **Operations:** Engage operations class, which provides a numerical interface along with operations, letting the user input number and operations, and get results from calculations
    - ~~**Graphs:** Engage the graphing class, which provides a numerical interface along with operations, and unknown variables (X, Y) which is designed to allow the user to enter different types of equations (such as quadratics), and draw a graphical representation of that equation. The calculator will also be able to calculate data using the graph, such as the x-intercept, y-intercept, etc.~~
    - ~~**Statistics**: Engage Statistics class, which will let the user input data into a table, which can then be graphed using the Graph class.~~
    - **Recall**: Open up a new window that shows the last 50 previous calculations: The user is able to review them and derive the answer of those calculations to use in the next calculation.
    - **Help menu:** Open up a new window that gives the user a detailed list on how to use the calculator and keyboard shortcuts.
    - **Exit:** A little button in the corner that the user can select to exit the program.
5. For each option, the program will behave in different ways:
    - **Operations**: Wait for user input. If the user enters digits, wait until they enter an operator, at which the program compiles the previous string of numbers as a multiple digit integer (if the user enters several digits in a row, e.g. 233, 10002, 23). If the user presses the equal button, the program reads the string of integers and operators, recognizes operators as functions, which will process the integers and provide the correct output. If multiple operators are used, the program will employ the BEDMAS procedure to determine which will determine which operators should be processed first. This option will mainly use functions, Python's Math module, and internal processing power. The user can open a submenu that reveals more functions that would typically not be used in everyday calculations, such as sin, log, etc.
    - ~~**Graphs:** The program provides a template for 'y = ' which is where the user will enter the input as an equation in terms of x (e.g. 'y = $3x^2$ + 4). The user can open a submenu which will allow the user to alter the~~

~~properties of the graphing process, such as making it in terms of y, or adding more equations. The program will apply the same process as Operators, isolating integers and operators until they are compiled together and produce an output. However, since the results of an equation are theoretically infinite, the program will automatically set limits to the equation so that the program doesn't break from the infinite results. If the set limit is undesirable to the user, the user can 'zoom out' the graph until they are satisfied with the shape of the graph. Once a graph is drawn, the user can access another submenu that gives several other functions, such as finding the x-intercept. This option requires functions, Math module, Tkinter canvas, and internal processing power.~~

- ~~**Statistics:** The program initially displays 2 columns and infinite rows, where x is one column and y is the other. Assuming the user has two sets of values, they can input these into the columns which generates a set of values for both x and y, which is used as coordinates for the graph. The submenu allows the user to pick connected or unconnected plotting points. Uses Tkinter entry/text, functions, Math module, internal processing power.~~
- **Recall:** The program displays a new window containing a list of the previous 50 calculations, which the user can then read over, and save any of the answers to use as an input, or save it to a variable. This will use functions and Tkinter list.
- **Help menu:** The program displays a new window that provides a readable information regarding the calculator, such as how to use the various functions and etc. Keyboard shortcuts are also shown, which will help the user in comfortability of using the calculator. Also gives the option of clearing the calculator cache, which clears out previous calculations.
- **Exit:** Program displays little goodbye message, then quits. Previous calculations are still stored.

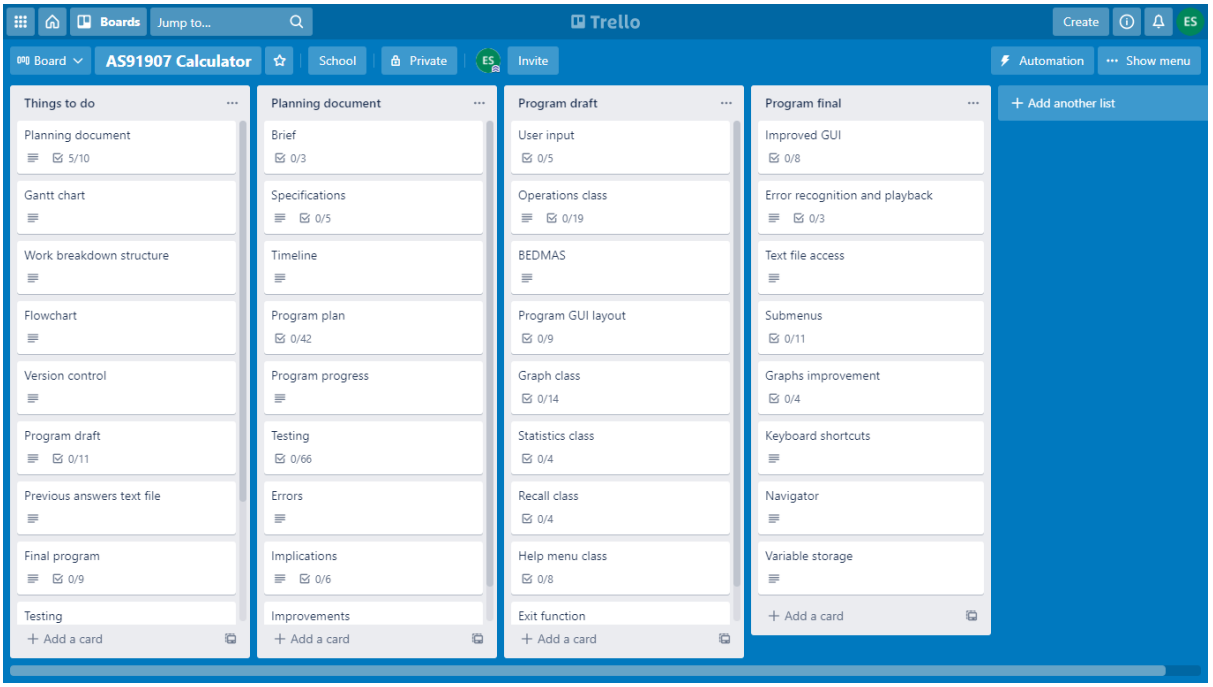
6. Program loops again once output is calculated. Output is saved to previous answer list in a separate text file. From there on, the program will continuously accept inputs from the user and produce outputs, until the user selects the exit button, or presses the exit tab of the GUI. This could be done via index analysis in the textbox, or storing inputs into a list and analysing the list.

The waterfall methodology will be used to manage the project
Work Breakdown Structure:

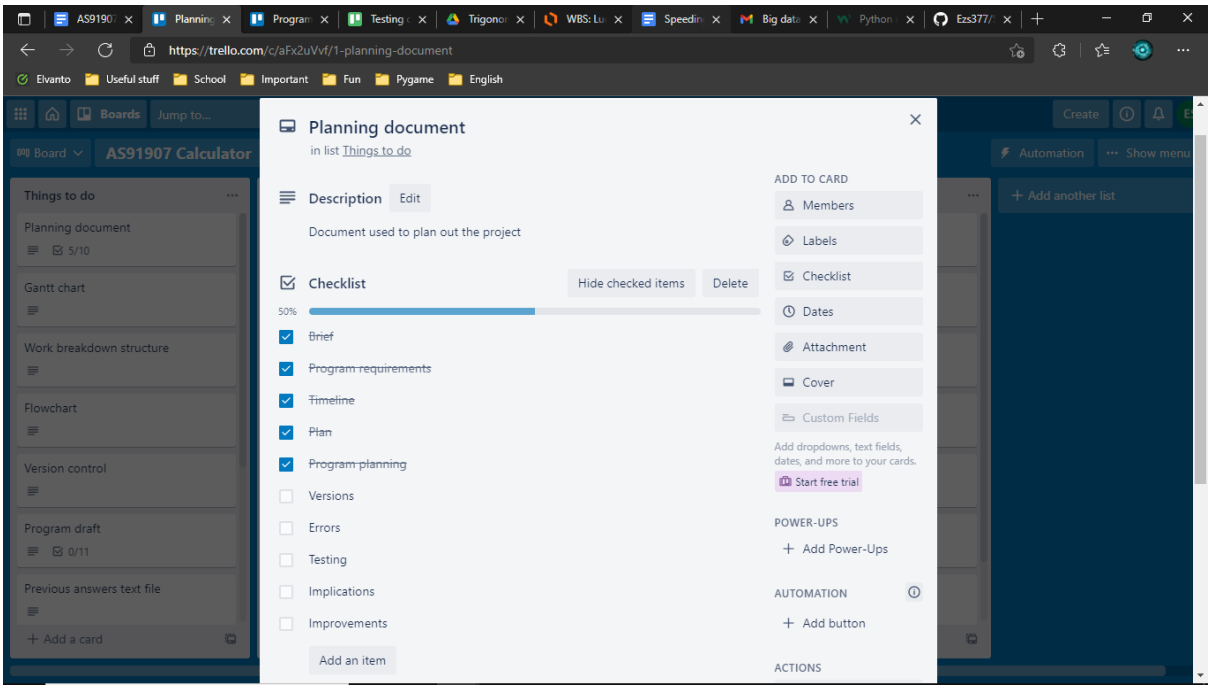


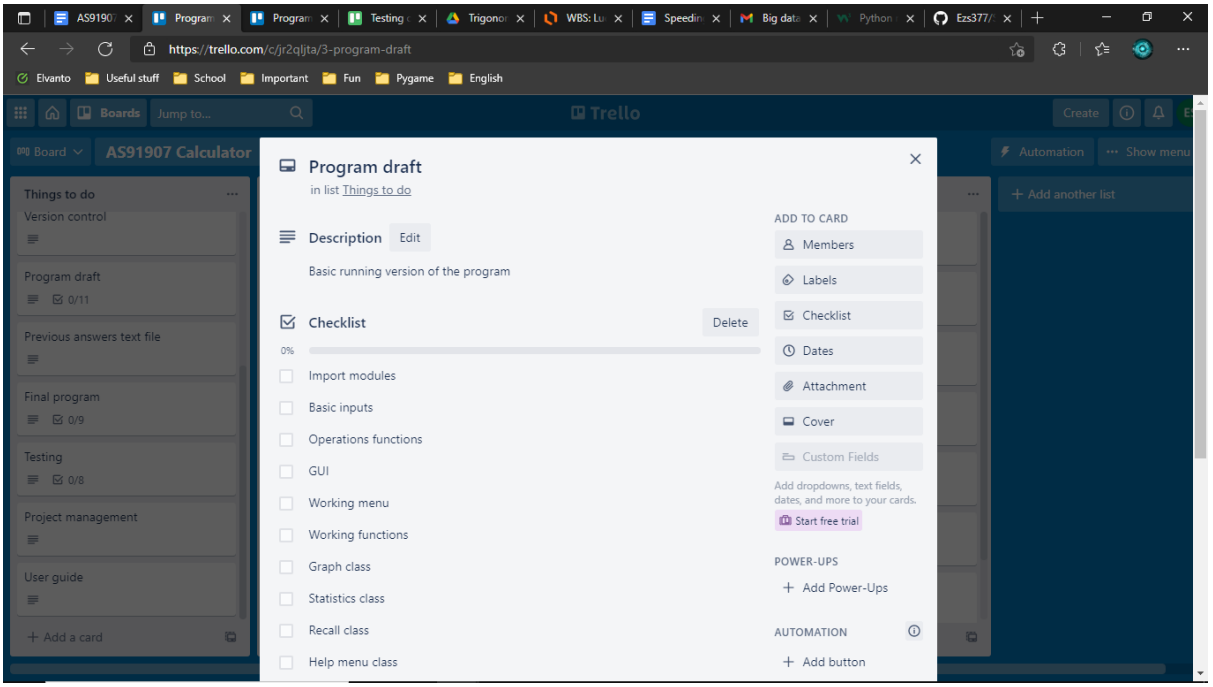
Trello board:

Finished lists are labelled green. Checklists are ticked off as they are done.



Examples of checklists:





Program parts:

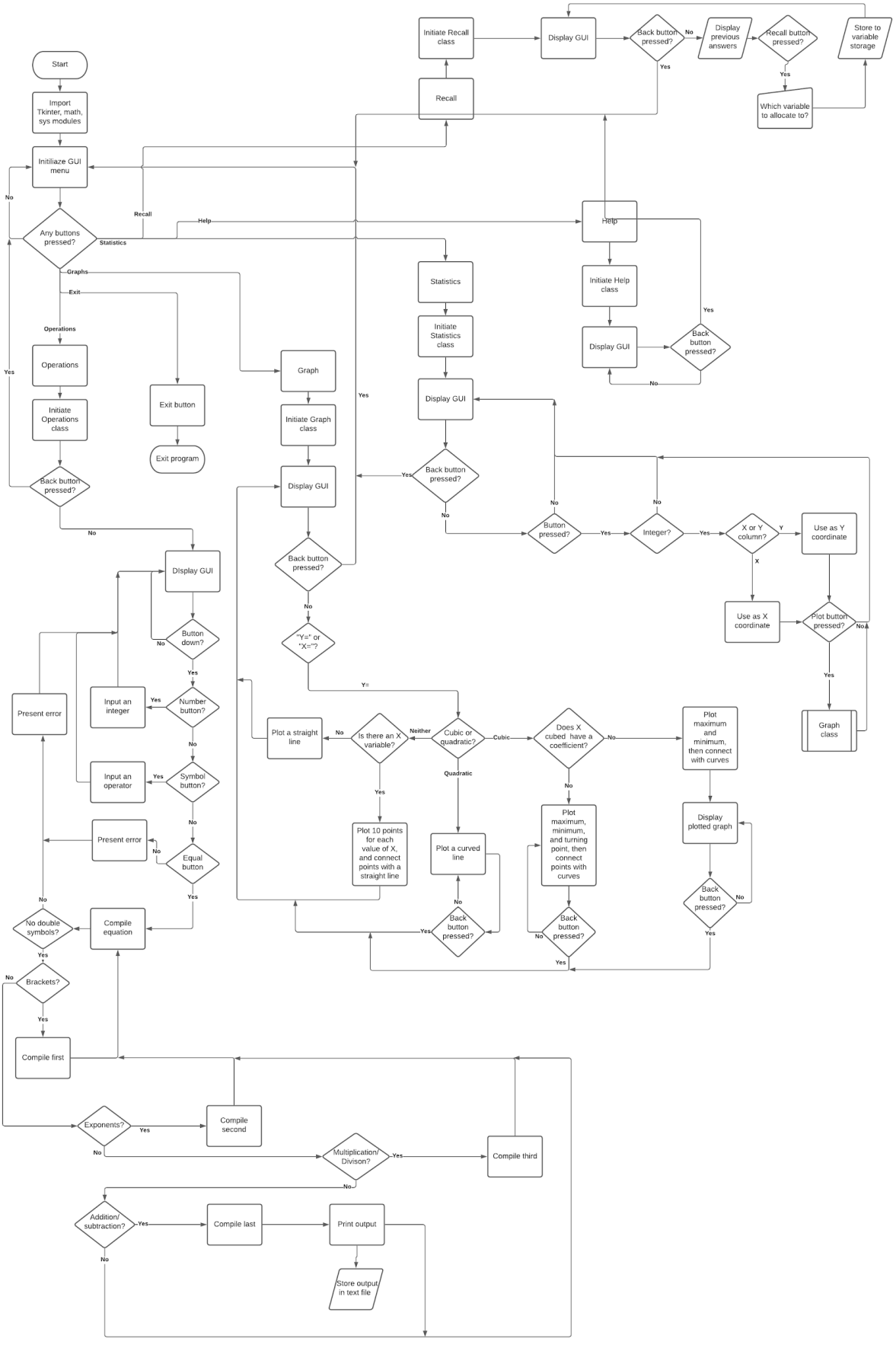
1. **GUI**
2. **Main menu**
3. **Operations:**
    - User input
    - Operators
    - BEDMAS
    - Submenu
    - Complex arithmetics
        - Constants (Pi, Euler)
        - Powers
        - Roots
        - Log
        - Natural log
        - Sin, cos, tan, and inverse counterparts
        - Factorials
    - Decimal to fraction to percentage (and vice versa)
    - Error recognition
    - Save to file
    - Variable storage
    - Screen clear/delete
    - Navigator
4. ~~**Graphs:**~~
    - ~~User input~~
    - ~~Equation formats~~
    - ~~Equation calculation~~

- ~~'Y='~~
- ~~'X='~~
- ~~Quadratics~~
- ~~Cubics~~

- ~~Submenu~~
  - ~~Plotting type~~
  - ~~Zoom~~
- ~~Tkinter Canvas graphing~~
- ~~Graph options~~
  - ~~Y-intercept~~
  - ~~X-intercept~~
  - ~~Point of intersection(s)~~
  - ~~Gradient(?)~~
  - ~~Gradient at point(?)~~

5. ~~**Statistics:**~~
   - ~~User input~~
   - ~~Tkinter list/entrybox for inputs~~
   - ~~Connect to **graph** class~~
   - ~~Coordinate substitution~~
   - ~~Submenu~~
     - ~~Plotting type~~
6. **Recall:**
   - Recall from save file
   - Tkinter list display
   - Bring back answer function
   - Allocate answer to variable function
7. **Help menu:**
   - Calculator information
   - Keyboard shortcut list
   - Clear cache function
8. **Exit:**
   - Implement sys.exit()
   - Goodbye message

Flowchart:


Start
Import Tkinter, math, sys modules
Initiliaze GUI menu
Any buttons pressed?
Recall
Help
Statistics
Graphs
Exit
Operations
Initiate Recall class
Display GUI
Back button pressed?
Display previous answers
Recall button pressed?
Store to variable storage
Which variable to allocate to?
Recall
Help
Initiate Help class
Display GUI
Back button pressed?
Statistics
Initiate Statistics class
Display GUI
Back button pressed?
Button pressed?
Integer?
X or Y column?
Use as Y coordinate
Use as X coordinate
Plot button pressed?
Graph class
Operations
Initiate Operations class
Back button pressed?
Exit button
Exit program
Graph
Initiate Graph class
Display GUI
Back button pressed?
"Y=" or "X="?
Y=
Display GUI
Button down?
Number button?
Input an integer
Present error
Symbol button?
Input an operator
Present error
Equal button
Compile equation
No double symbols?
Brackets?
Compile first
Exponents?
Compile second
Multiplication/ Divison?
Compile third
Addition/ subtraction?
Compile last
Print output
Store output in text file
Is there an X variable?
Plot a straight line
Plot 10 points for each value of X, and connect points with a straight line
Cubic or quadratic?
Plot a curved line
Back button pressed?
Does X cubed have a coefficient?
Plot maximum, minimum, and turning point, then connect points with curves
Back button pressed?
Plot maximum and minimum, then connect with curves
Display plotted graph
Back button pressed?

Potential difficulties:

- To display powers, I would need to use superscript text characters within Python. There are multiple ways: Import a module that provides more text options, the maketrans() method (which requires a set of superscript characters to be imported first as a string), and Unicode characters. The simplest and easiest way is to use Unicode, as this doesn't require any imports, and is built in Python itself.
- ~~Graphing using Tkinter Canvas would be very difficult. The matplotlib module is capable of drawing graphs, but this is an entirely new module that I am not familiar with, so I will need to find an alternative using Tkinter, math module, or inbuilt Python functions. However, as Canvas itself uses coordinates, it seems possible that I could simply translate plane coordinates to the Canvas so that Python 'sees' the Canvas as a cartesian plane. From there on, all I would need to do is code the functionality of plotting coordinates using a formula.~~
- For complicated arithmetics such as logs, graphing, plotting, etc. I would need to be able to recall the formulas used for such problems, which may take longer as I would need to study math formulas again, in particular graphs, and I need to be able to implement the steps taken in solving equations to a program.
- Since there will be a lot of buttons used for the GUI, it would be more efficient to create a function that generates a button in a specific location with its own command, instead of creating multiple buttons in one class.The problem with this method is that it will be difficult to keep track of the buttons, however, it will also take less time to build code for this method compared to simply creating each button one by one.

Other notes:

- All data is saved to an external, local text file, so the user can recall their calculations even if the program is closed.
- Some functions may be removed, or new functions may be added, depending on inspiration.
- When testing, all testing is done using an online calculator, and a physical graphics calculator. Desmos (a free online graphing tool) will be used to test the graphs generated by this program.
- Superscripts (for powers) and custom characters can be implemented via Unicode characters. This is the simplest and easiest way to display mathematical symbols such as root and powers.

NOTE: After considering the time constraints on this project, especially with the comeback of the NZ lockdown due to Covid, I have decided to cancel the Graphing and Statistics functions, as this will be quite complicated to create, and take considerably longer for the project.

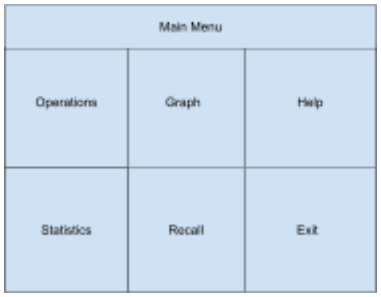
Program classes process:

- **Operations:** The calculator presents a grid of buttons, which resemble the operators, as well as digits 0-9, and the equal sign. The user can either click these buttons to input them into the calculator, or use the keyboard buttons instead. Inputted characters appear on the output box which is at the top of the window. The program will continue to accept inputs until the *equals* sign is pressed. At that moment, the program will firstly examine the input for errors. Such errors include double operators, spaced out characters, trailing operators, unpaired brackets. If there is any error detected, the program will throw an error message at the user (Tkinter message) without clearing the output box, so the user can fix their errors using the navigator. Once there are no more errors, the program will begin to separate operators and integers. Integers that are next to each other will be considered as a multiple digit integer. After all integers are compiled, the program will use the BEDMAS order to determine which operators to compile first. The operators separating the integers are used to calculate a value, which is then combined with the other compiled operators to produce a singular integer output, and is displayed in the output box in the GUI. This output is also stored in the text file, and the Answer variable. The user can then clear the screen, oor perform another calculation. The user can also go back to the main menu using a button.
- ~~**Graph:** The calculator presents a window that displays a 'Y=' sign, and the same set of buttons from the Operations class, along with the additional buttons X and Y, which represent the axis for the graphs respectively. The user can input integers and operators on this line to create an equation, which uses the same calculation procedure from the Operations class, and the errors checking system as well. If the user presses the *equal* button after inputting their equation the program will create another 'Y=' line, allowing the user to input another equation in terms of X. If the *equal* button is pressed while a 'Y=' line is blank, the program assumes the user is done inputting their desired equations, and will compute the inputted equations. Firstly, the Canvas is pre-formatted so that instead of using the default coordinate system of Tkinter, the Canvas will have the coordinates of a cartesian plane, thus allowing the program to calculate a point for Y for every value of X *(more to be elaborated on)*. Once the graph is drawn, the user can press the back button to return to the equation GUI.~~
- ~~**Statistics:** The program displays a GUI involving two columns, one for X and one for Y. Each column lines up with each other so that values for X correspond to values for Y. The user can enter integer values into both columns, and the program will reject symbols and string inputs. The button grid only consists of digits and no operators. Once the user is done inputting values, the user can press the *Graph* button which utilises the Graph class to create a graph using the coordinates inputted by the user. This will not work if the user hasn't inputted a corresponding value for X or Y, or has left some values blank in the columns.~~

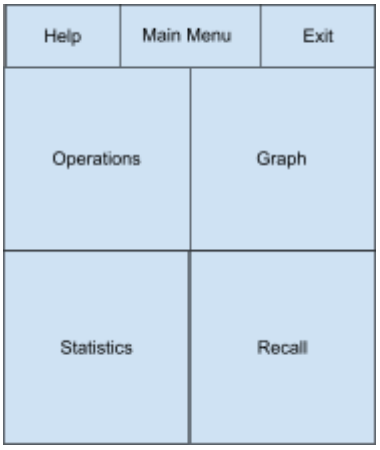
- **Recall:** The program displays a GUI consisting of a list of previous outputs from the Operations class, which has been read from the text file. The list is scrollable, and each item has a button beside it. The user can click the button to get the value and allocate it into a variable of their choosing, which could either be the string values A to Z, or the Answer variable. The user can go back to the main menu anytime using the back button.
- **Help:** The program opens a new window that has text which contains a simplified user guide, and a scrollable list of keyboard shortcuts that has been implemented into the program to allow easier use. There is also a *clear* button that wipes out the text file used to store previous outputs, which will warn the user if they click it and ask the user to confirm. The *close* button closes the window.
- **Complex functions:** Aside from the classes, there are other features available in the program. The Decimal to Fraction to Percentage (DFP) function converts any integer value into the desired format. This button is present in the Operations class, and pressing it will toggle the output through decimal, fraction, and percentage forms. Storing this output results in the form it was stored in. Numerical constants are also implemented, which will be Pi and Euler's number. The *shift* button also allows the user to toggle between functions for the calculator. Toggling Sin, Cos and Tan will invert them to their inverse counterparts. Toggling the constants lets the user pick between Pi or Euler. The *shift* button also toggles other complex arithmetic functions such as powers, roots, log. The variable storage button is present in the Operations class, which lets the user store an output to the variable they choose. This variable is labelled as a string letter, from A to Z. The user can either use the keyboard to pick the desired variable, or click a button on the GUI. When allocating an output, all buttons on the grid are temporarily swapped with the letters A to Z so the user can pick one to choose a variable.

Program GUI layout concepts:

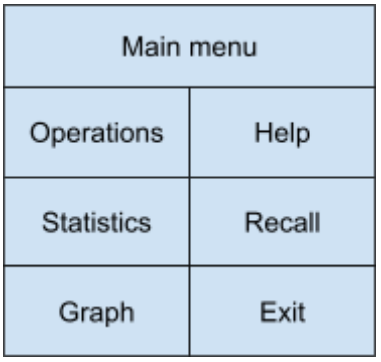
Main menu:



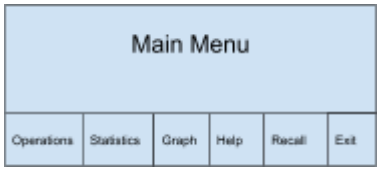
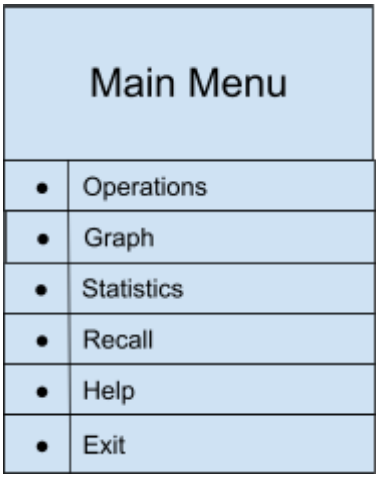
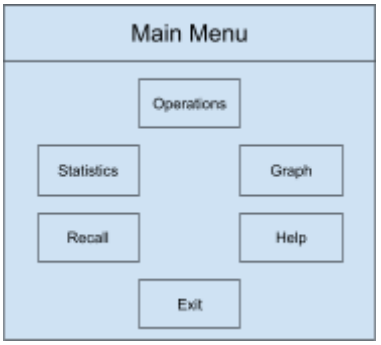
A basic layout, buttons are too big



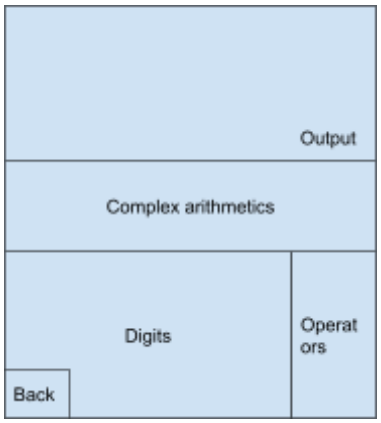
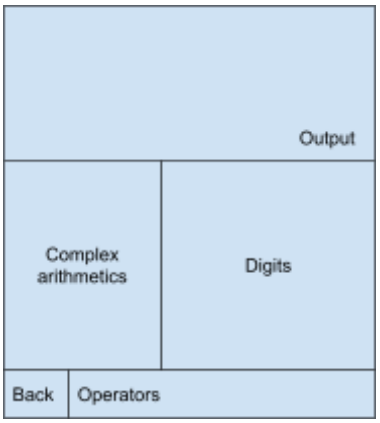
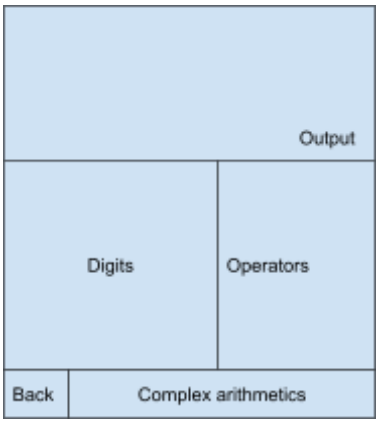
Too cluttered, buttons are



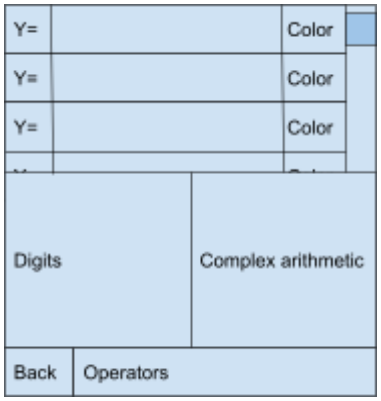
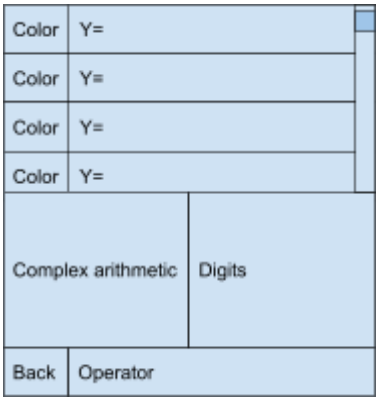
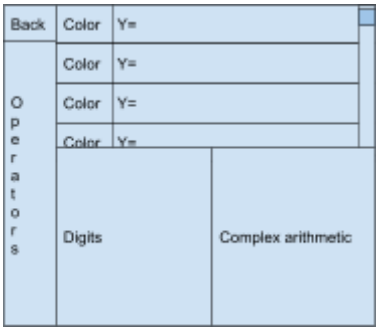
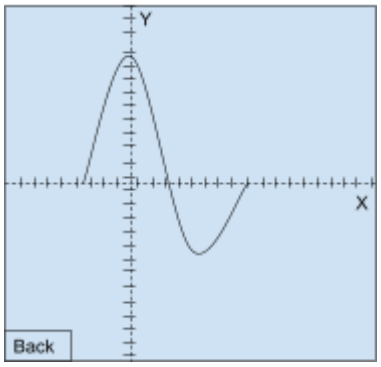
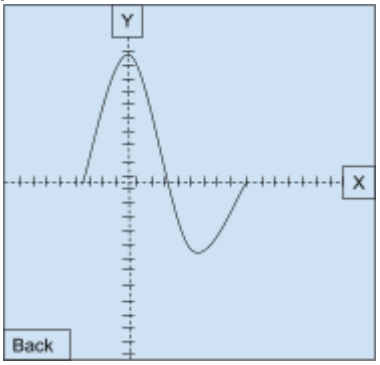
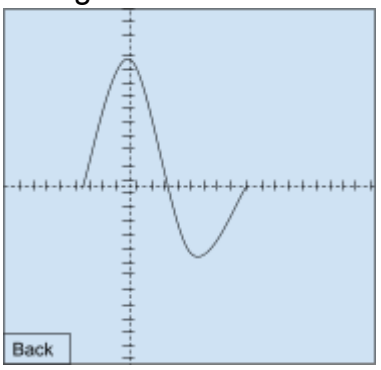
This one, it is simple and clean.

	out of place	
 Not bad, but too much blank space in the title.	 Looks good, but overly complicated to design aesthetically.	 Nice concept but buttons are too weirdly positioned for a rectangle shaped window.

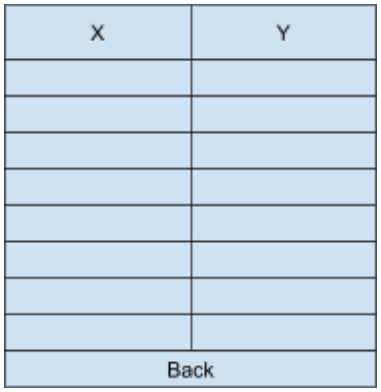
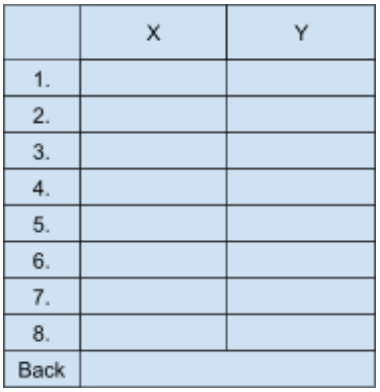
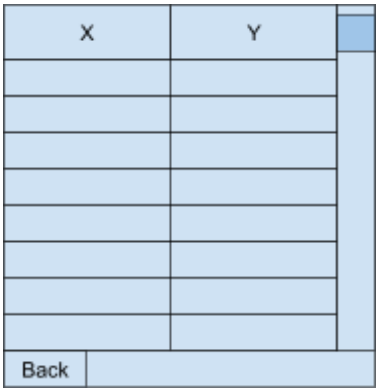
Operations:

 The traditional calculator design	 More neatly designed to utilise all space. This one will be picked, with minor modifications if applicable	 Same as previous but inverted positions

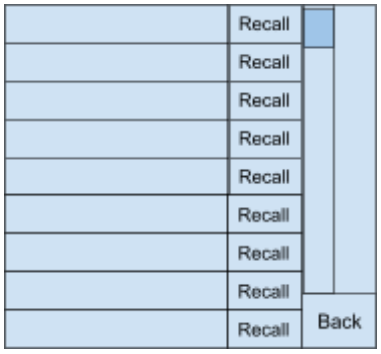
Graph:

		
~~Classic design. Equations on top, buttons on bottom.~~	~~Sleeker design. Inverted placement.~~	~~Upheaved design. Follows the classical calculator arrangement more closely, albeit inverted button arrangement.~~
		
~~Classic graph design. X and Y axis labelled.~~	~~Same as previous, except X and Y axis labels are more pronounced.~~	~~X and Y axis aren't labelled.~~

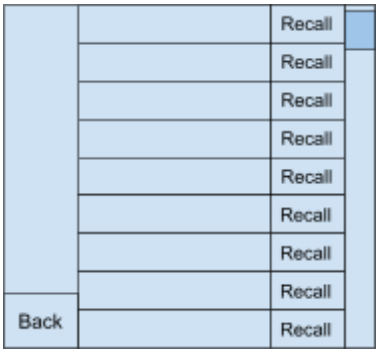
~~Statistics:~~

		
~~Basic design~~	~~Same design, but with list numbers.~~	~~Same design, but with a scrollbar implemented.~~

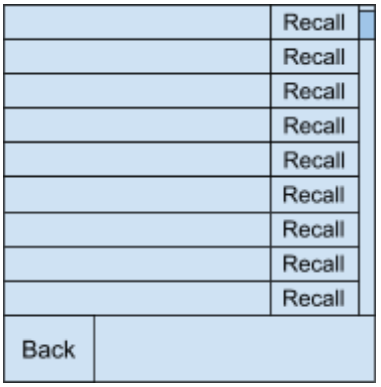
Recall:



Basic design

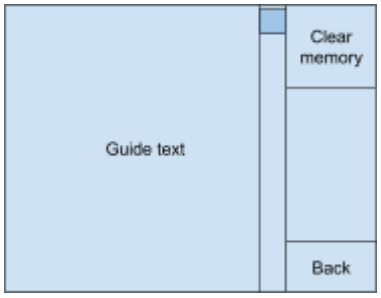


Tidied design, leftover space on the left side

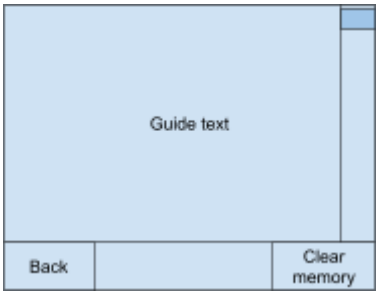


Sleeker design, still some leftover space on bottom

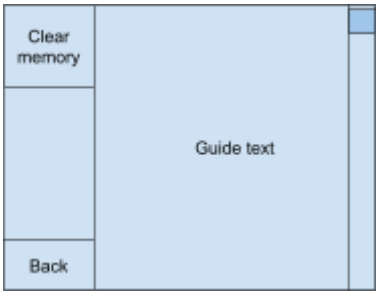
Help:



Basic design

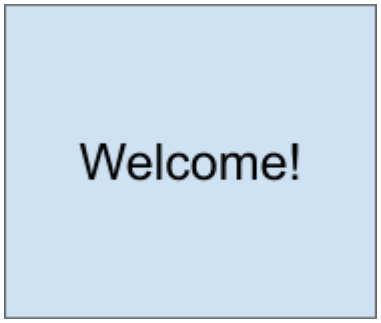


Buttons moved to bottom



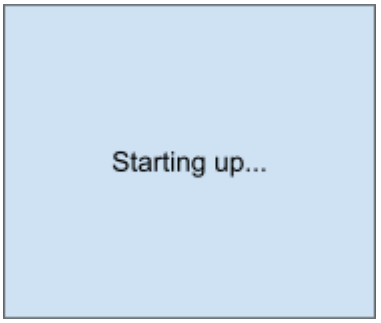
Buttons moved to left side

Hello:



Friendly/neutral

Friendly/basic



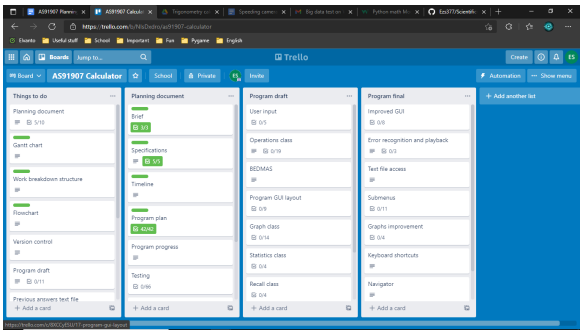
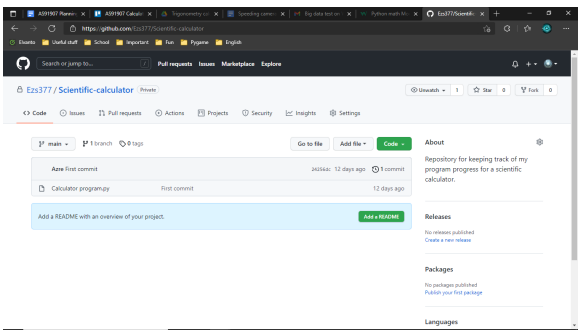
Neutral

Goodbye:

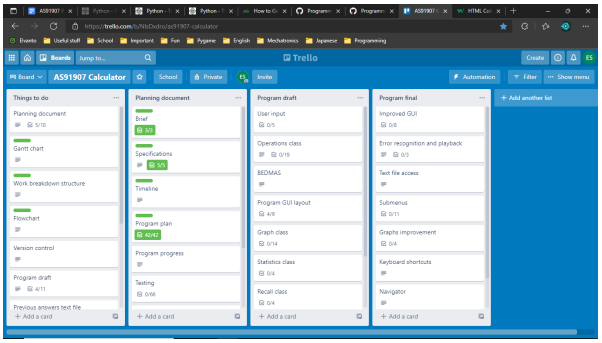
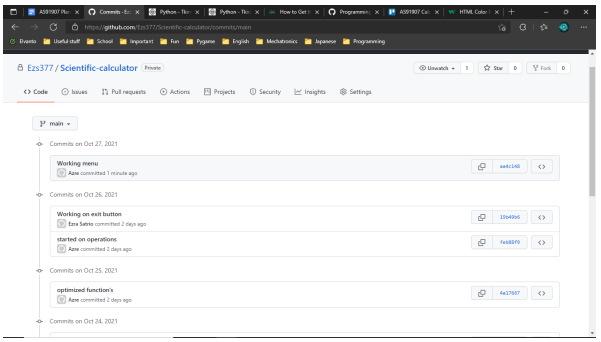
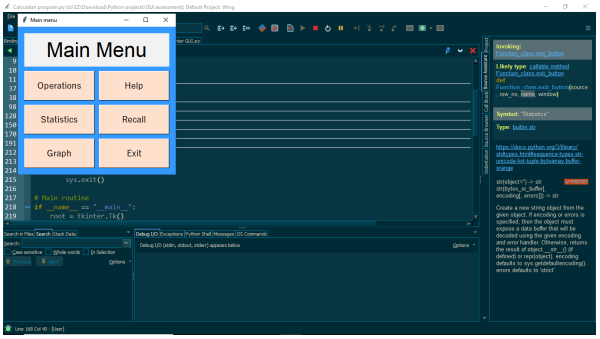
Shutting down...	Exiting program...	Thank you for using this program!
Neutral	Neutral/out of place	Friendly

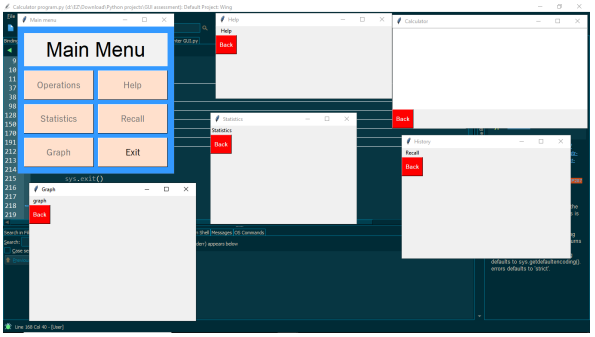
Project progress (updated periodically, commit and Trello):

1. Started on program

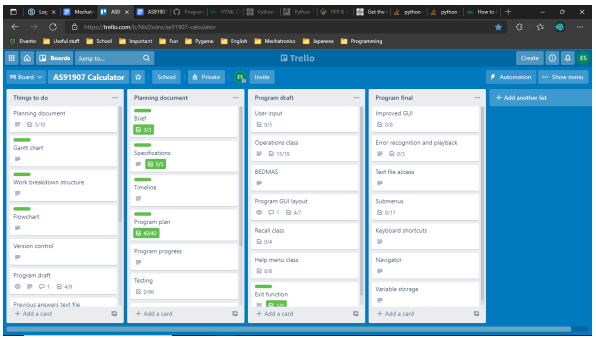



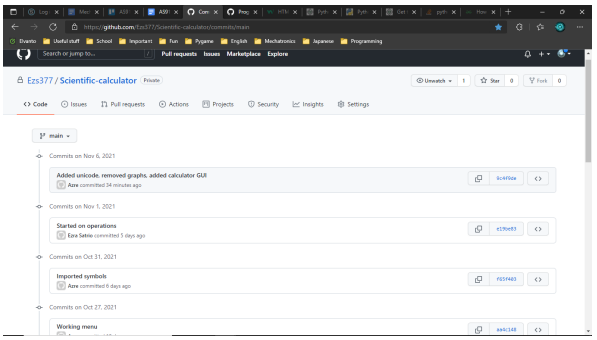
2. Main menu finished and working



3. Imported all special characters via unicode, and finished calculator buttons

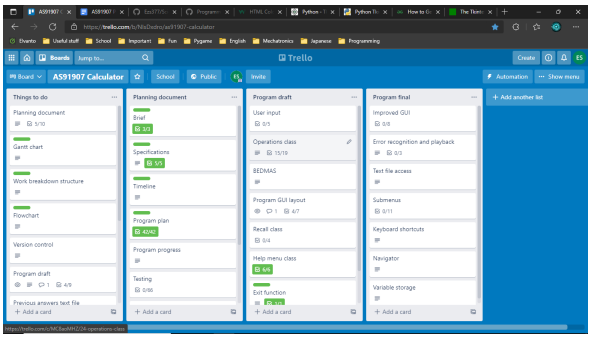


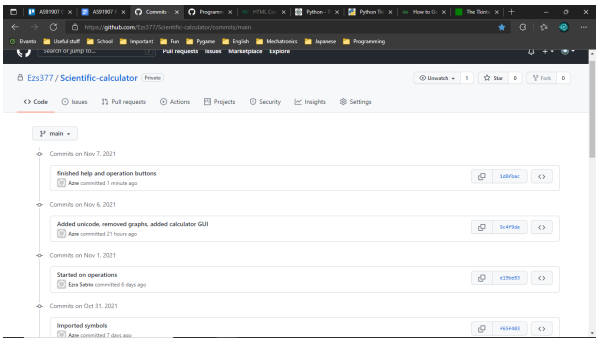


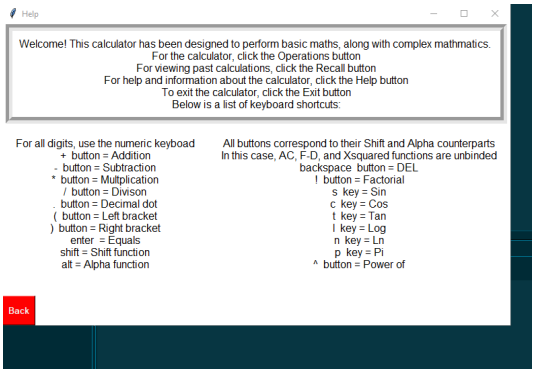
AC	DEL	!	+	F-D	Shift	Alpha
7	8	9	-	sin	cos	tan
4	5	6	x	log	ln	π
1	2	3	÷	X²	^	=
Back		0	.	(	)	

```
class Characters:
    character_dict = {'euler': "\u2107", 'pi': "\u03c0", 'squared': "\u00b2",
                      'lbracket': '(', 'rbracket': ')', 'log': 'log ',
                      'sin': 'sin ', 'cos': 'cos ', 'tan': 'tan ', 'ln': 'ln ',
                      'factorial': '!', 'inverse': '\u207b' + '\u00b9',
                      'inversesin': 'sin' + '\u207b' + '\u00b9 ',
                      'inversetan': 'tan' + '\u207b' + '\u00b9 ',
                      'inversecos': 'cos' + '\u207b' + '\u00b9 ',
                      'percent': '\u0025','clear': "AC",'delete': 'DEL',
                      "plusminus": "+/-",'add': '+', 'minus': '-',
                      'divide': '\u00f7','times': 'x', 'equals': '=', 'power': '^',
                      'squareroot': '\u221a', 'root': '\u02e3' + '\u221a',
                      'zero': '0', 'dot': '.', 1:'1', 2:'2', 3:'3',
                     4:'4', 5:'5', 6:'6', 7:'7', 8:'8', 9:'9',
                      'convert':'F-D', 'shift':'Shift', 'alpha':'Alpha',
                      'xsquared':'X'+"\u00b2"}
```

4. Finished Help class and info

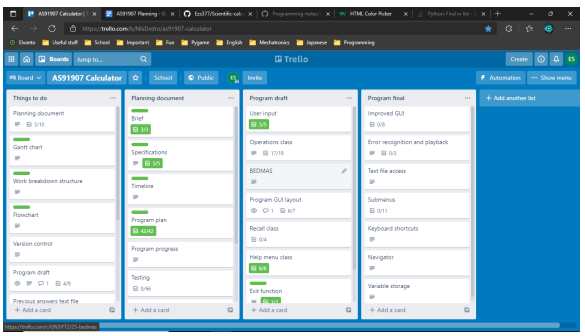


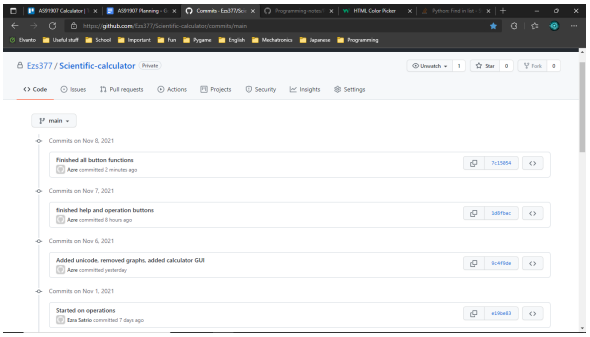


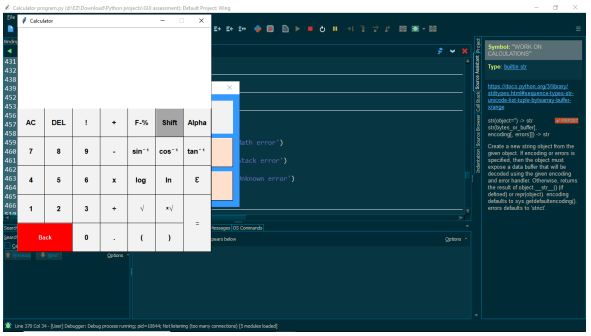



```
# User guide
class Help:
    def __init__(self):
        self.file = open('Help.txt', 'r')
        self.help_file = (self.file.read())

        self.file2 = open('Left help.txt', 'r')
        self.l_help = (self.file2.read())

        self.file3 = open('Right help.txt', 'r')
        self.r_help = (self.file3.read())

        # This disables the corresponding button in the main menu
        button_dict["Help"].config(state=tkinter.DISABLED)

        # Create window and frame
        self.help_window = tkinter.Toplevel()
```

5. Finished all button functions, now to work on calculations

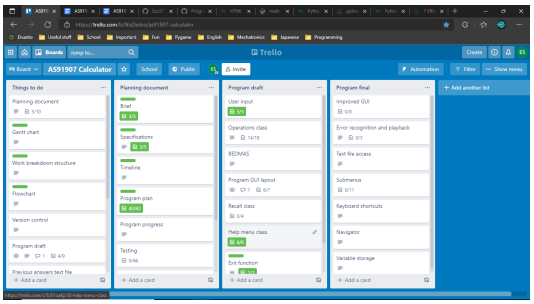


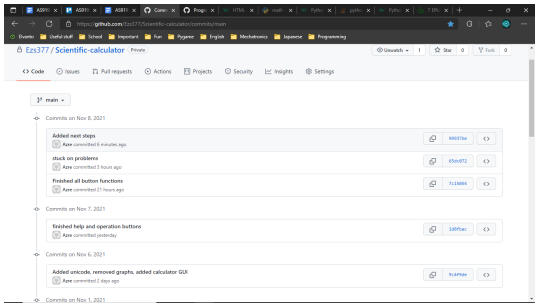


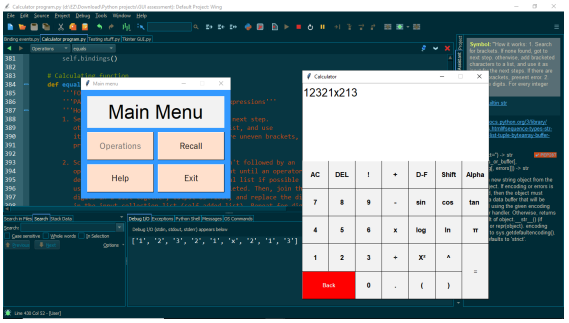


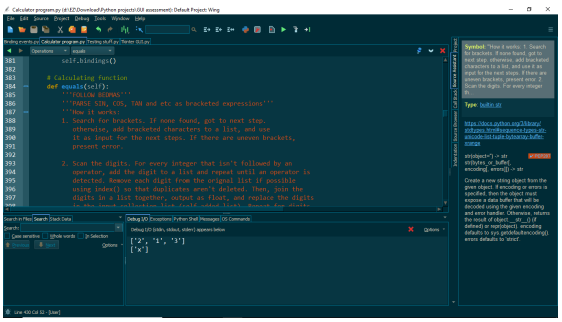
```
# For the shift button
def shift_convert(self):

    # If unshifted
    if self.toggle == 1 and self.a_toggle == 1:

        # Change color of SHIFT
        self.row1[5].config(bg='#a6a6a6')

        for a in self.allrows:
            # If the label of a button matches
            # a character in shift dictionary
            if a['text'] in Characters.shift_characters:
                self.convert = Characters.shift_characters[a['text']]
                a.config(text=self.convert,
                         command=lambda x=a:self.printout(x))
        else:
            pass
```

6. Hit a stepback, undid all the calculation coding due to a cluttered program. Added notes for the next steps

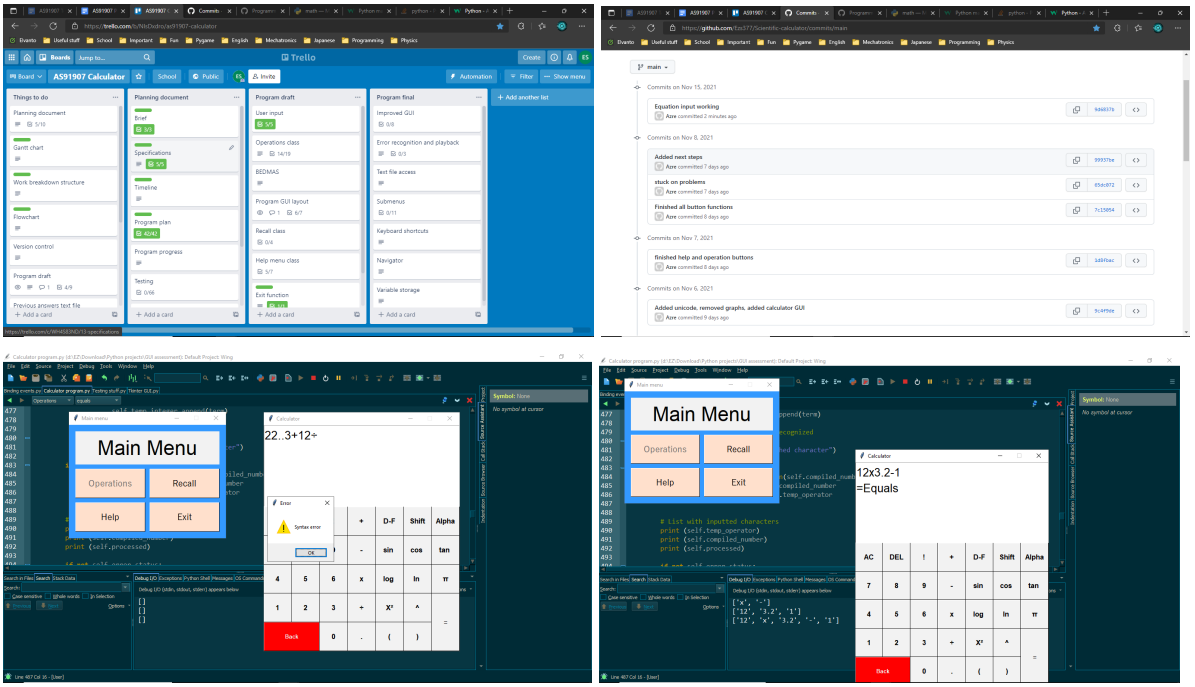




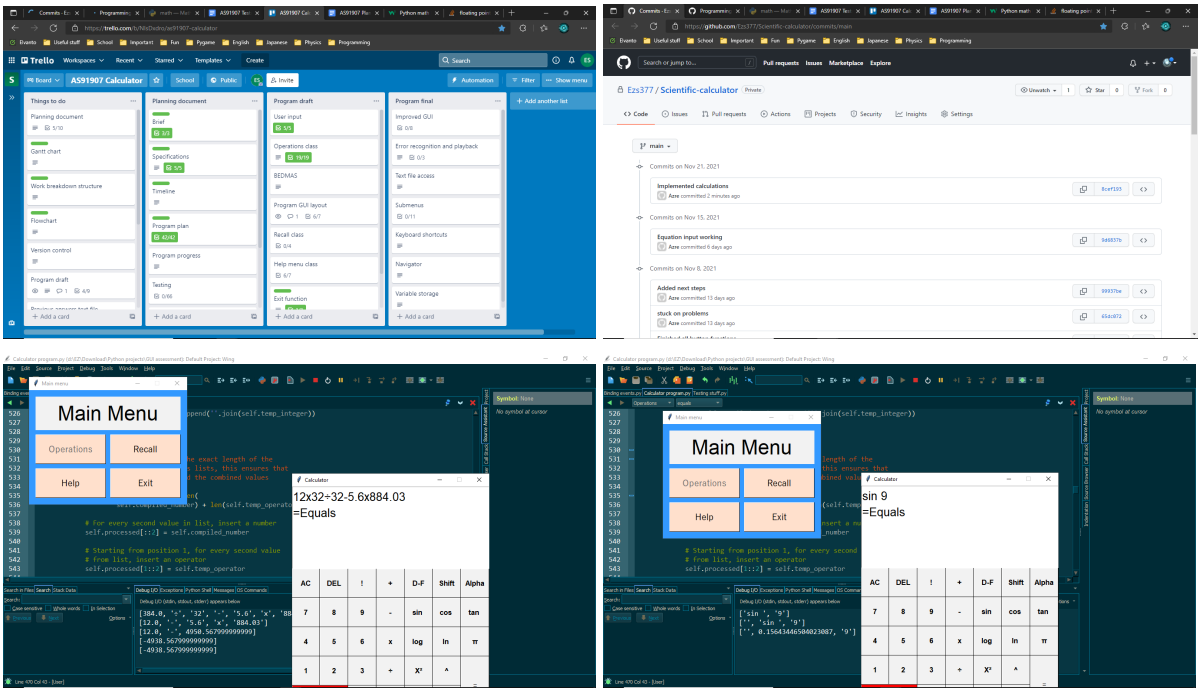




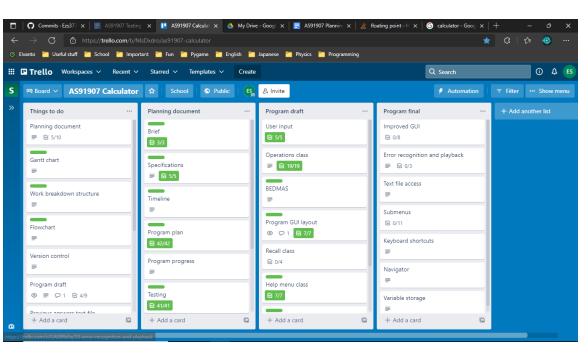
7. Error checking, input checking, number and operation processing re-done and working, the program can effectively derive an equation from a list of characters. Next step is to implement analysis of brackets, and BEDMAS calculations

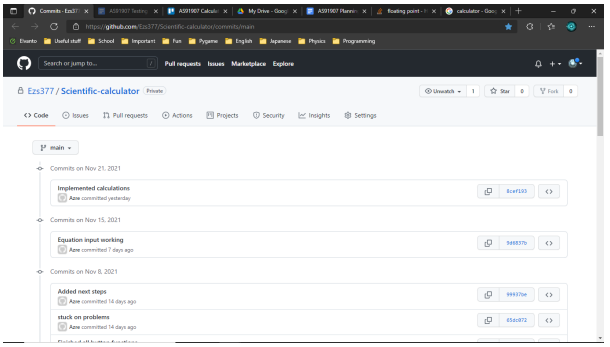


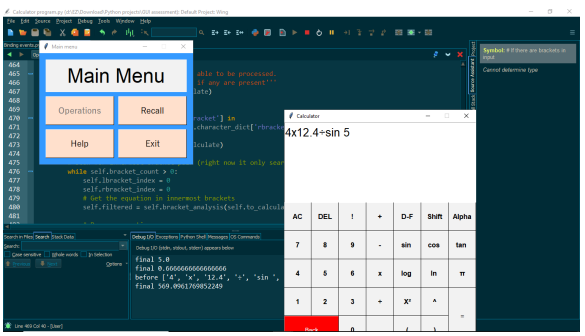
8. Implemented beta version of calculative process, still contains a few bugs.

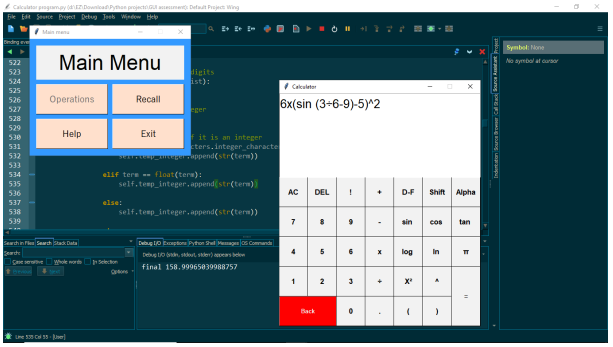


8. Calculative and BEDMAS formula finished, now all that's left is to provide and output, and Recall class.

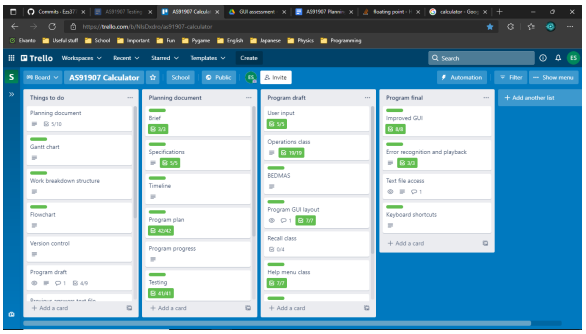


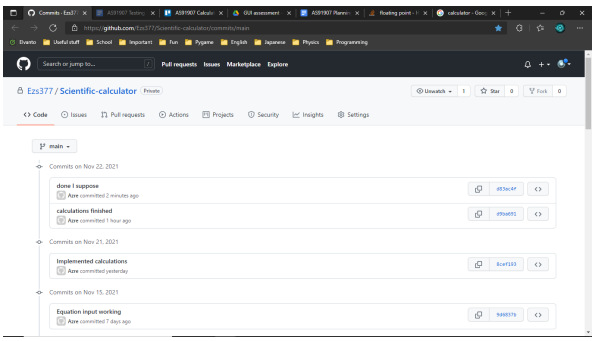


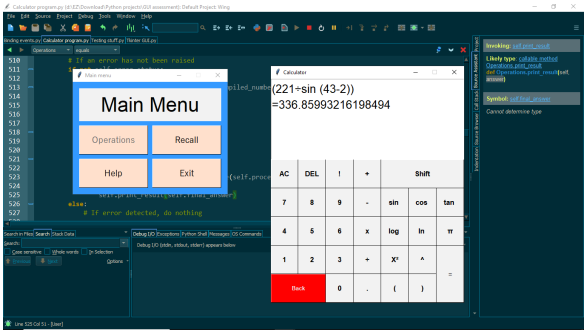


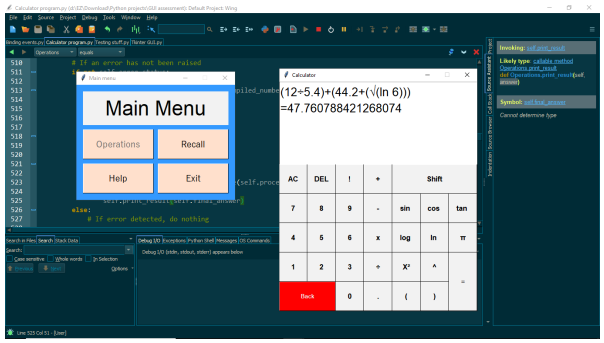


9. Program prototype DONE!

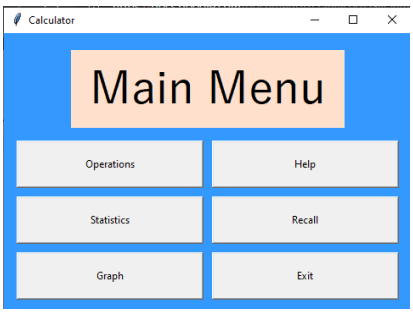


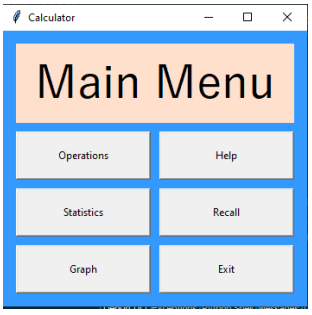


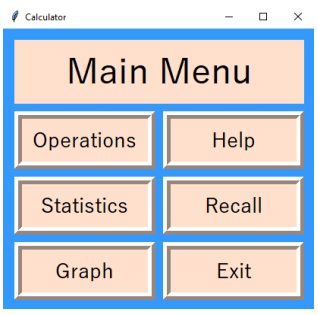


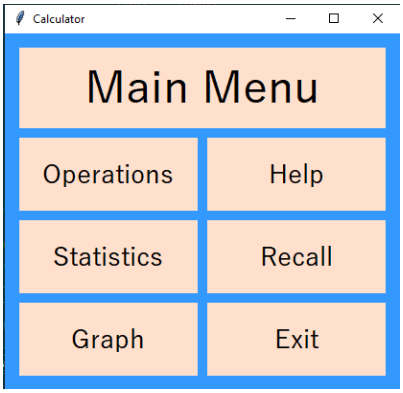


Trialling program:
Main menu:

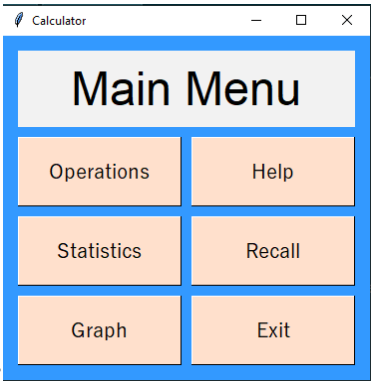




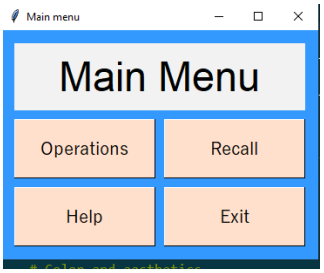




~~This one -->~~



After eliminating the Graph and Statistics, the remastered GUI:



Operations class:
When attempting to create the calculator buttons I realized that there were insufficient characters on my laptop keyboard, so I couldn't insert such characters (e.g. Pi, division symbol). And copying such characters from another source could lead to unicode issues which I've had before on previous projects, like when copy-pasting text from a Word document to Python. Therefore, I need to utilize unicode characters in my program, which raises the issue of unicode outputs and terminal compatibility, as not all computers operate the same way, thus the character encoding system could be different on some computers. However, Python runs UTF-8 encoding by default, so therefore it should be safe to print out UTF-8 encoded characters. To further safeguard against this, I'll compile the program as an exe file, which should solve all software/hardware related issues with the program.

A problem that occurred when making the calculator buttons is that I used one list to contain the data of all the buttons, and since I used various 'for' loops to generate

each ‘group’ of buttons, this resulted in an issue where I couldn’t locate the index of the buttons in the list, as a ‘for’ loop counts from the first item in the list. Obviously, if I was trying to generate the 2nd buttons and onwards, the ‘for’ loop would keep repeating the first value in the list, which meant that the first button would always be generated regardless of the amount of buttons generated. To fix this, instead of generating buttons in a ‘group’ manner (e.g. Doing all numbers, then operations, etc) I decided to generate my buttons in a row-by-row manner, which simplifies the button generation process, and I will store each row in a list, which should make it easier to navigate the lists of generated buttons, as I would just need the row number+column number to locate a button. It also means that I can create a function that would generate a row of buttons, instead of creating several ‘for’ loops with different parameters.

I also decided to collect all the characters of the calculator buttons into one dictionary, instead of splitting them up into their respective groups (e.g. all numbers belong to the ‘integer’ list, operations go into the ‘operations’ dictionary, etc).

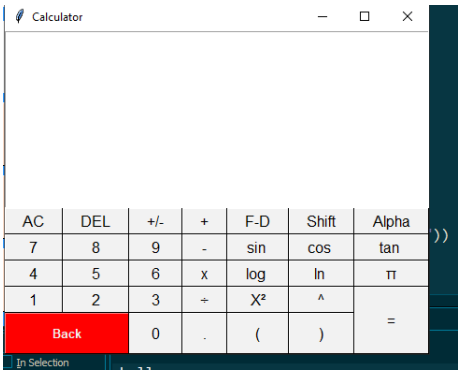
From:

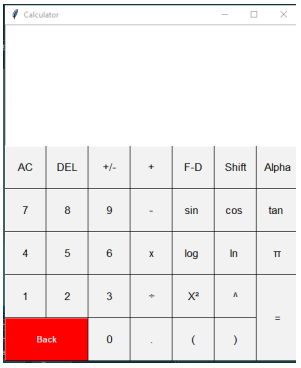
```
def __init__(self):
    self.button_list = []
    for no in range(1, 10):
        self.button_list.append(str(no))

    self.operation_dict = {'add': '+', 'minus': '-', 'divide': '\u00f7',
                           'times': 'x', 'equals': '=', 'power': '^',
                           'squareroot': '\u221a', 'root': '\u02e3' + '\u221a',
                           'zero': '0', 'dot': '.'}

    self.symbols_dict = {'euler': "\u2107", 'pi': "\u03c0", 'squared': "\u00b2",
                         'lbracket': '(', 'rbracket': ')', 'log': 'log ',
                         'sin': 'sin ', 'cos': 'cos ', 'tan': 'tan ', 'ln': 'ln ',
                         'factorial': '!', 'inverse': '\u207b' + '\u00b9',
                         'inversesin': 'sin' + '\u207b' + '\u00b9 ',
                         'inversetan': 'tan' + '\u207b' + '\u00b9 ',
                         'inversecos': 'cos' + '\u207b' + '\u00b9 ',
                         'percent': '\u0025',
                         'clear': "AC",
                         'delete': 'DEL',
                         "plusminus": "+/-"}
```

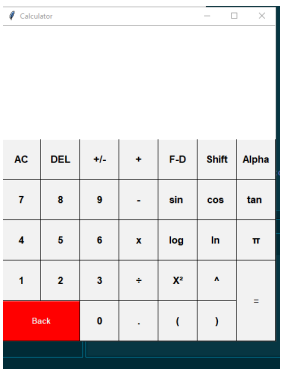
To:

```
.character_dict = {'euler': "\u2107", 'pi': "\u03c0", 'squared': "\u00b2",
                   'lbracket': '(', 'rbracket': ')', 'log': 'log ',
                   'sin': 'sin ', 'cos': 'cos ', 'tan': 'tan ', 'ln': 'ln ',
                   'factorial': '!', 'inverse': '\u207b' + '\u00b9',
                   'inversesin': 'sin' + '\u207b' + '\u00b9 ',
                   'inversetan': 'tan' + '\u207b' + '\u00b9 ',
                   'inversecos': 'cos' + '\u207b' + '\u00b9 ',
                   'percent': '\u0025','clear': "AC",'delete': 'DEL',
                   "plusminus": "+/-",'add': '+', 'minus': '-',
                   'divide': '\u00f7','times': 'x', 'equals': '=', 'power': '^',
                   'squareroot': '\u221a', 'root': '\u02e3' + '\u221a',
                   'zero': '0', 'dot': '.', 1:'1', 2:'2', 3:'3',
                   4:'4', 5:'5', 6:'6', 7:'7', 8:'8', 9:'9'}
```

Operation designs:

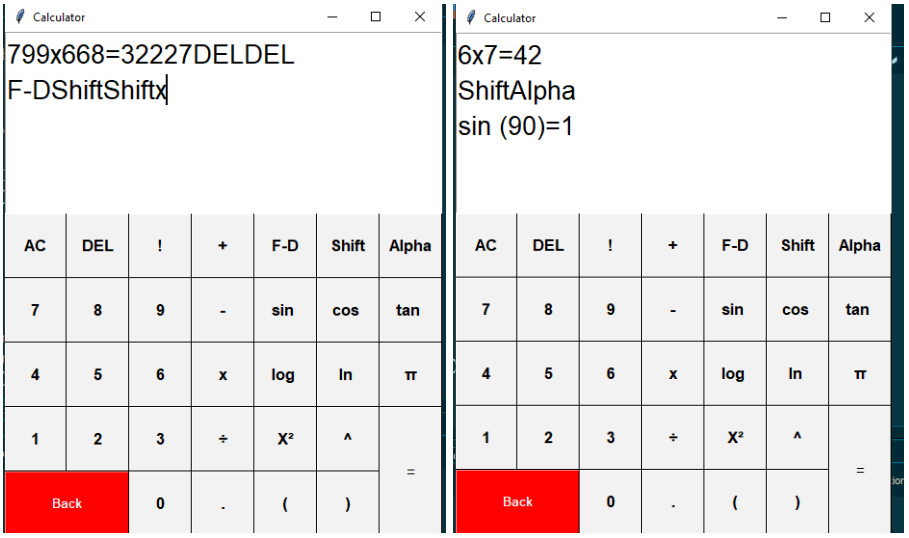






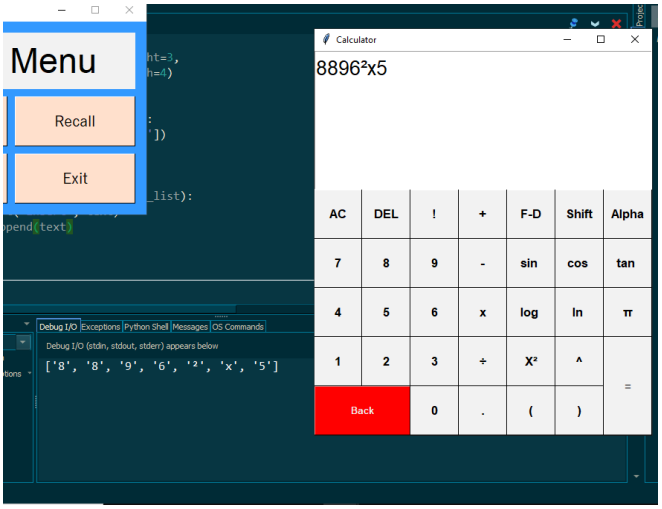
<---This one

Button inputs working:

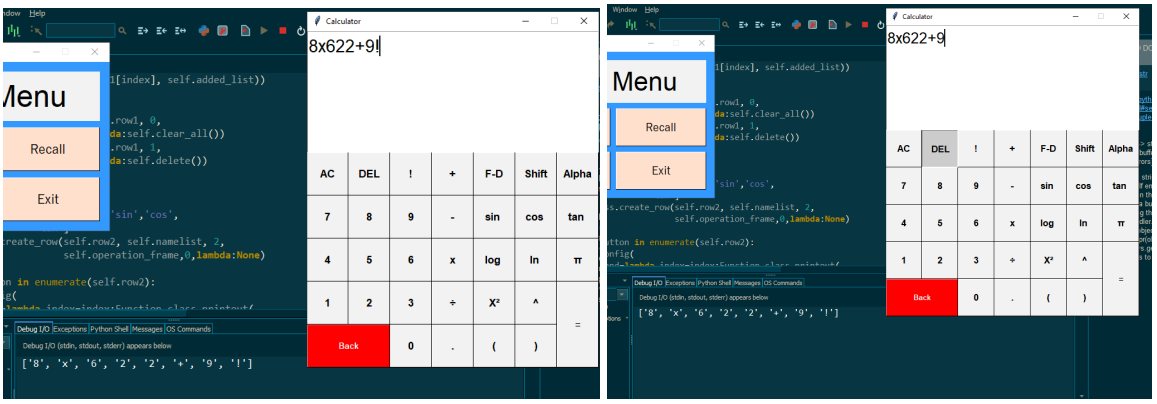


I could use the grabset() method to keep the focus on the toplevel window only (i.e. the calculator GUI), but this would disable the main menu. This means that the user couldn't access other menu items while using the calculator, which I thought would be detrimental and annoying as the user would have to close the calculator everytime to access the menu, such as Help and Recall.

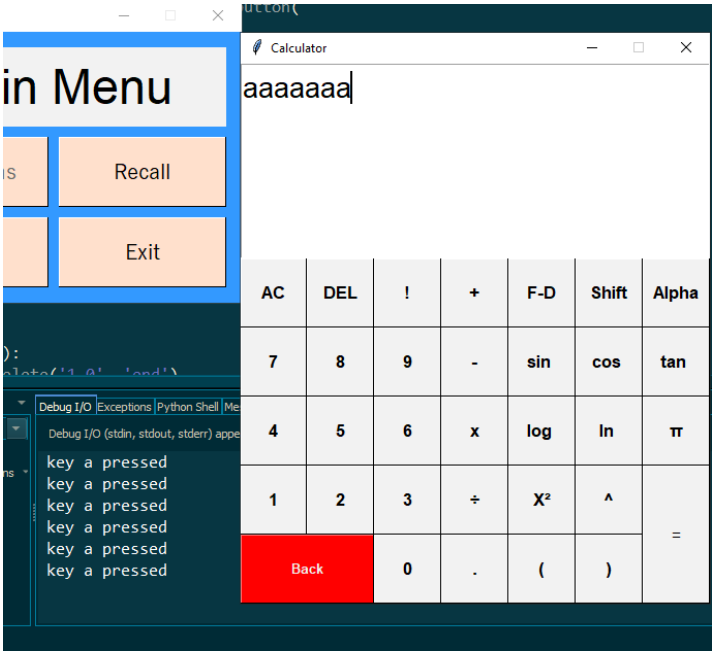
Equals button can derive inputs from the calculator. First step to calculation and error checking.



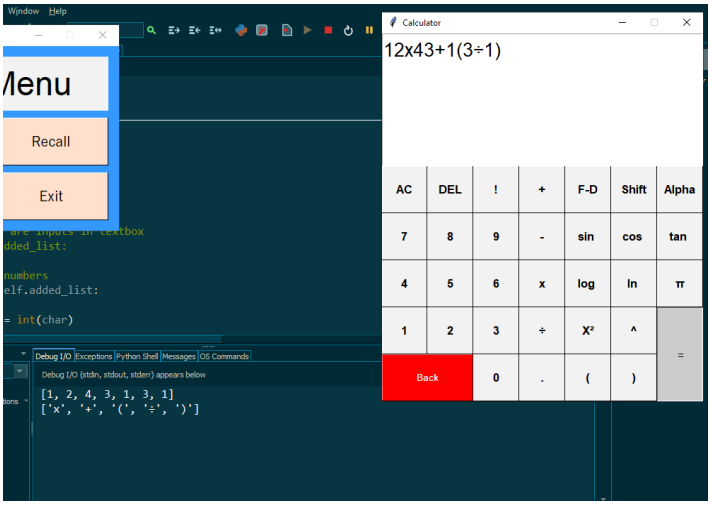
The DEL button now deletes one character off the end



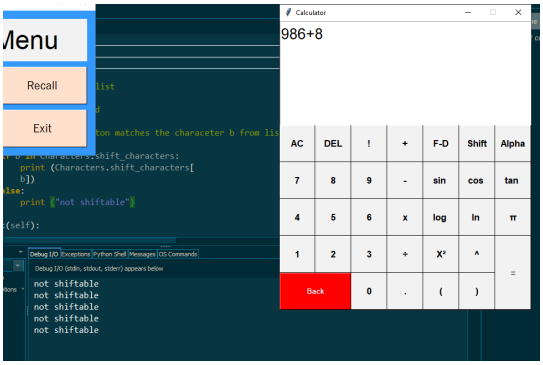
Keyboard inputs are recognised

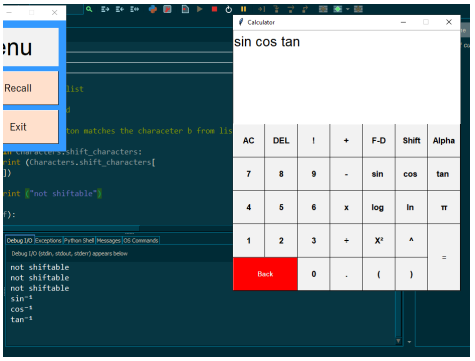


Program can distinguish between integers and operators

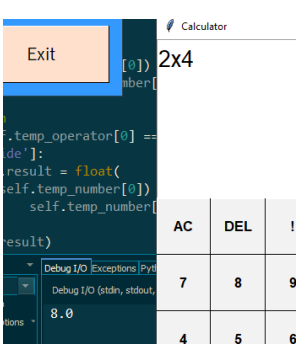


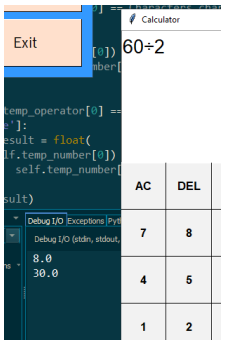
Program can detect shift function

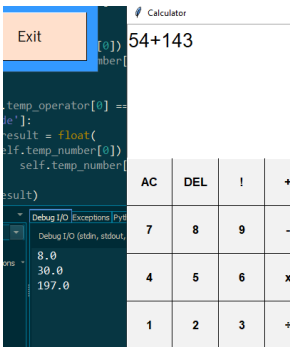


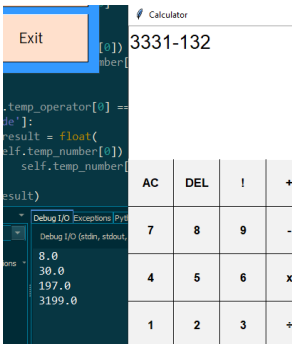


Program can do basic operations

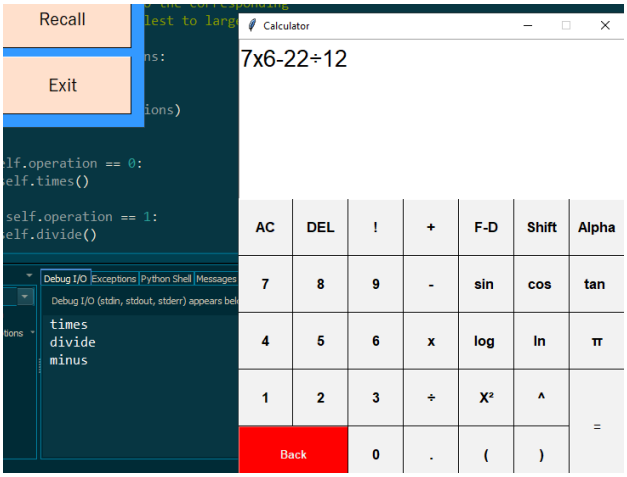




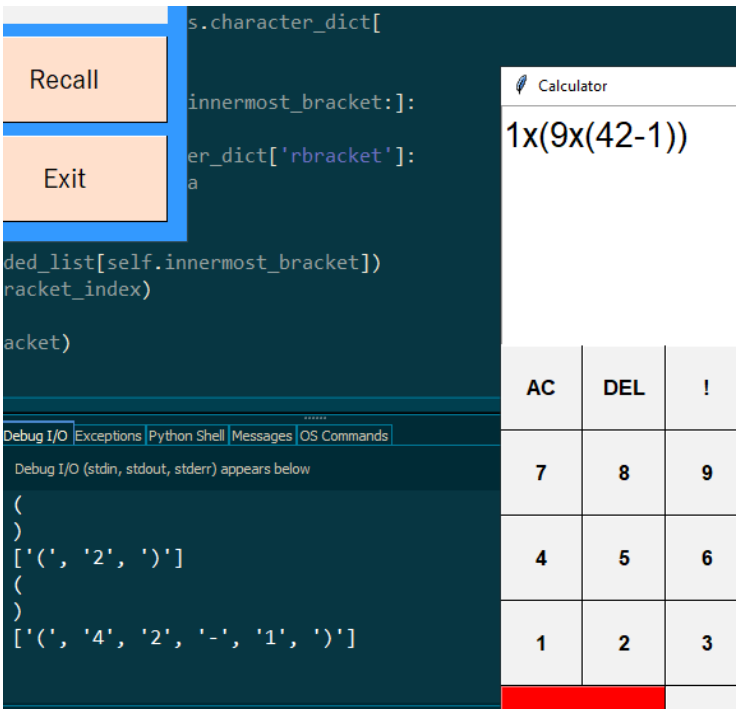




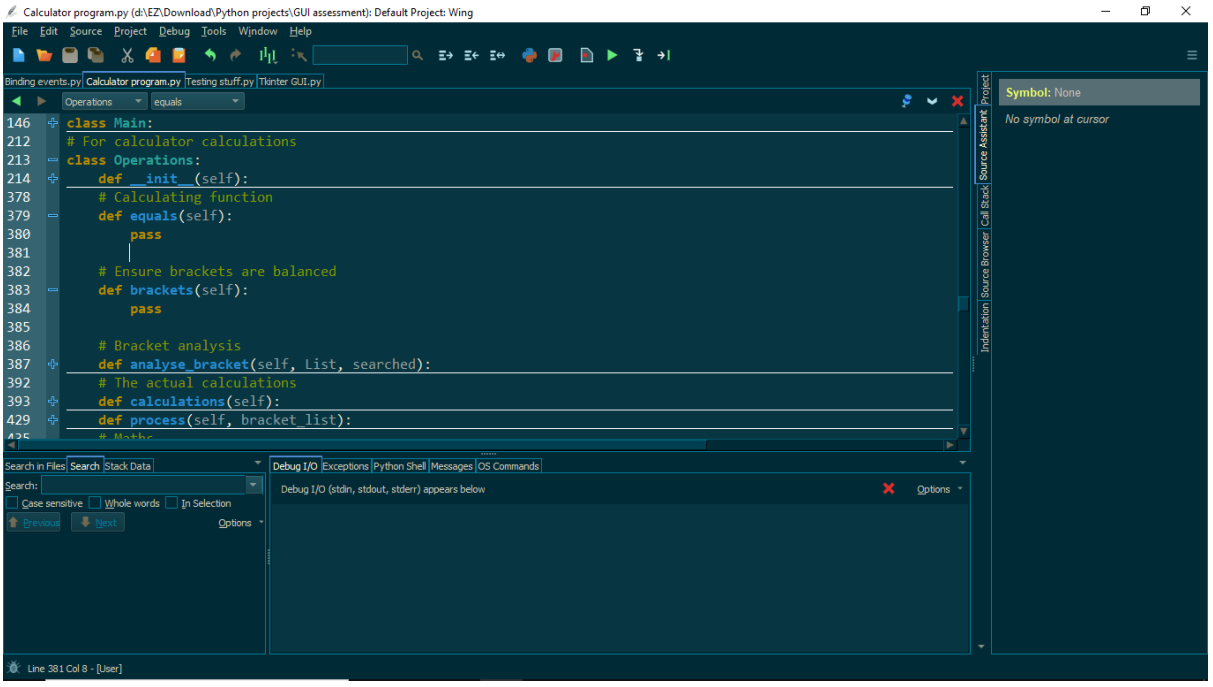
Program can follow BEDMAS



Program can derive inputs inside brackets

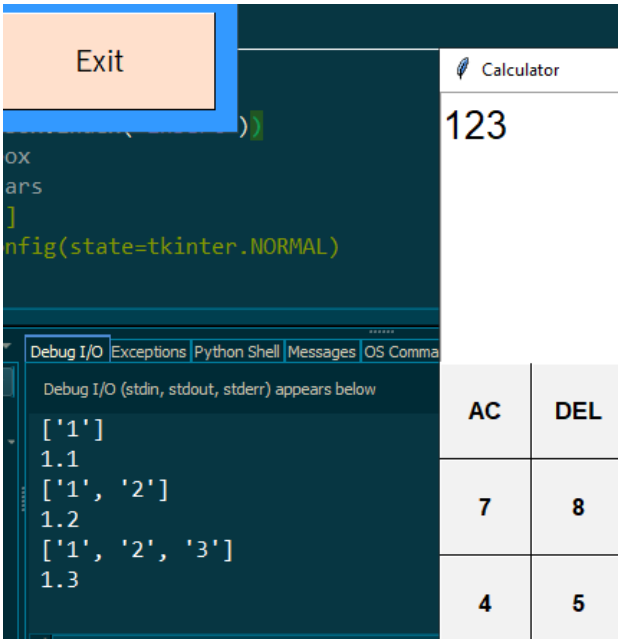


At this point, I had to redo some of the functions because it was getting cluttered and the program made no progress, undoing my previous work. The functions removed were: Calculations, Printout, Equals, Brackets. These functions were intended for the calculations of user input, and deriving equations from user input. However, I hit a dead end when I couldn't find the source of an error that occurred if the user entered several operations in one equation.

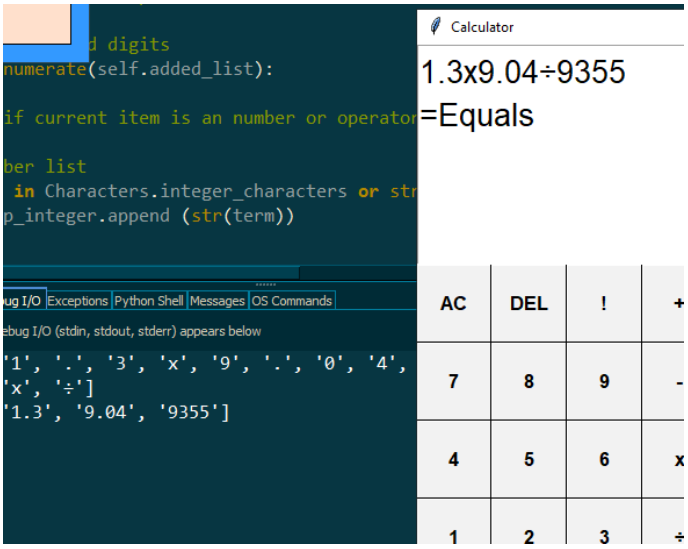


After this, I have to re-code the deleted functions, but this time, I have to ensure everything is neatly arranged and coded so that it is easy to follow for myself and others, and allowing me to continue coding and diagnosing errors.

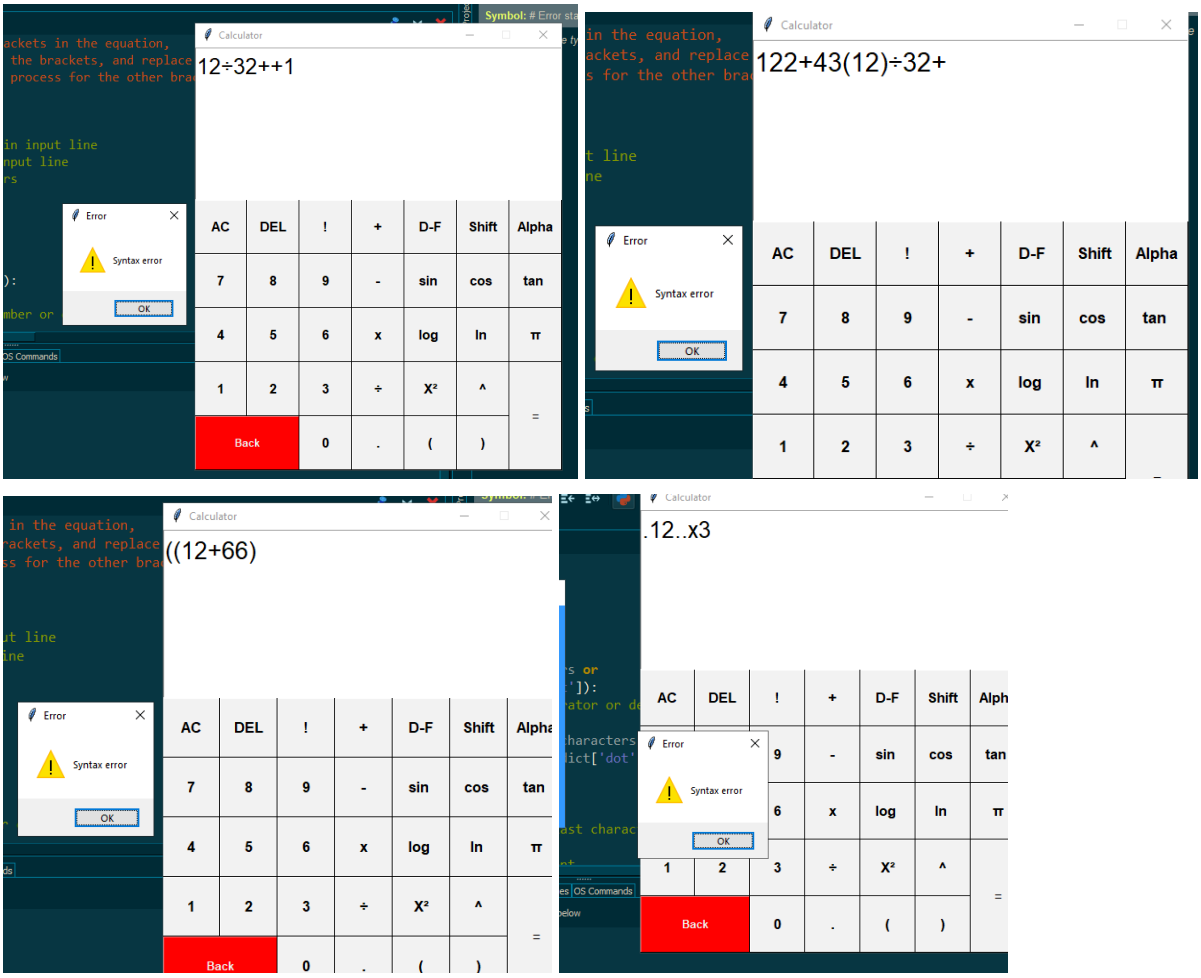
Getting index of text cursor



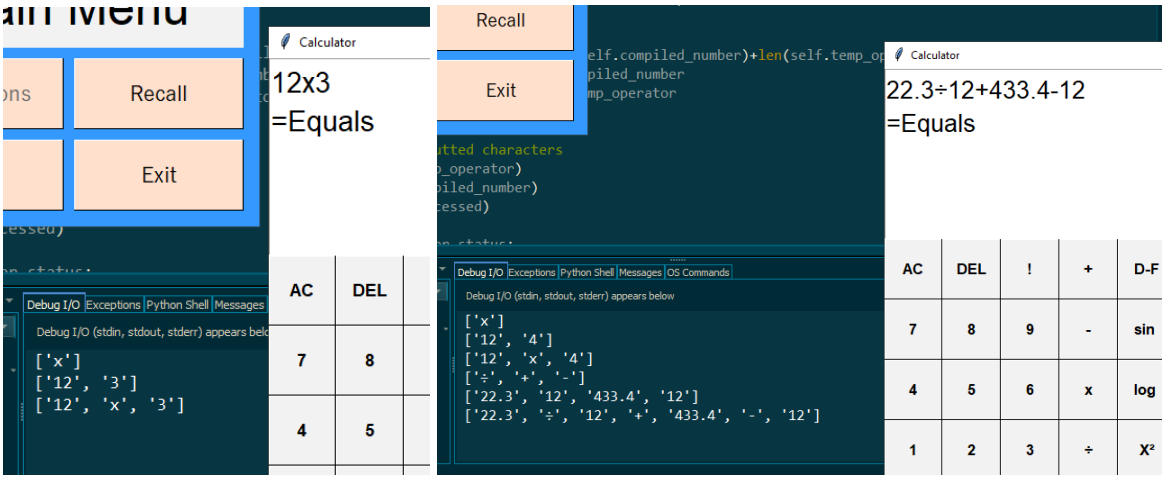
Program can differentiate between operation, integer, and decimal

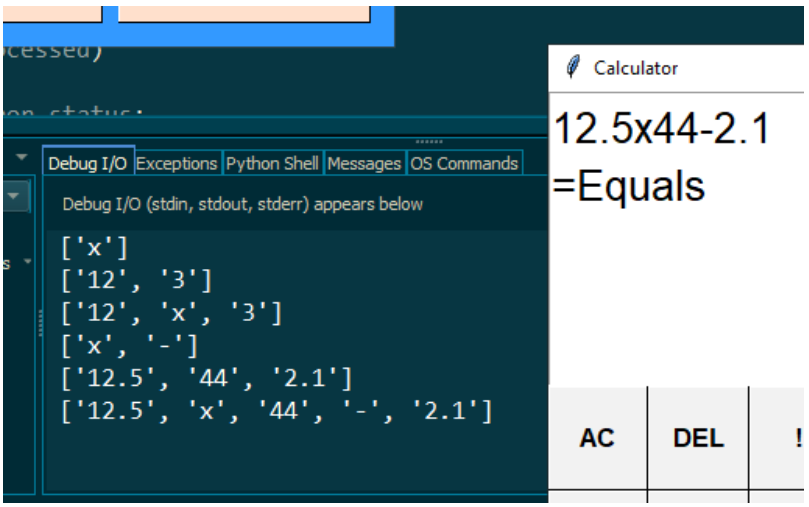


Program can detect syntax errors

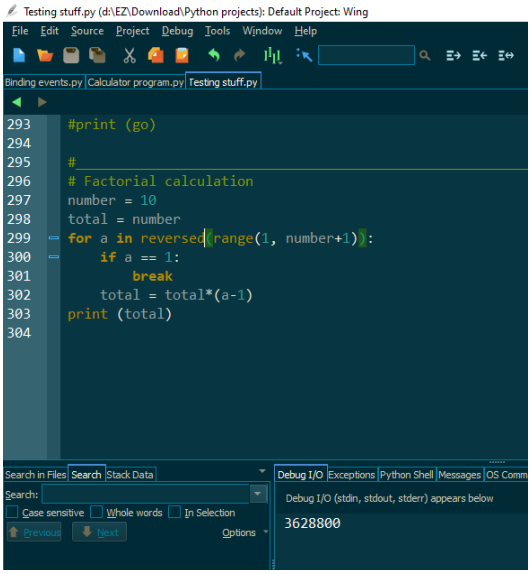


Program can compile multi digits, separate them from operators, and then bring them back together in original order (very important in recognizing equations!!!)

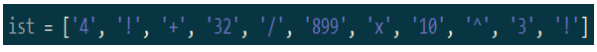


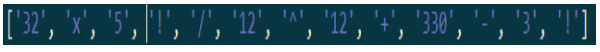


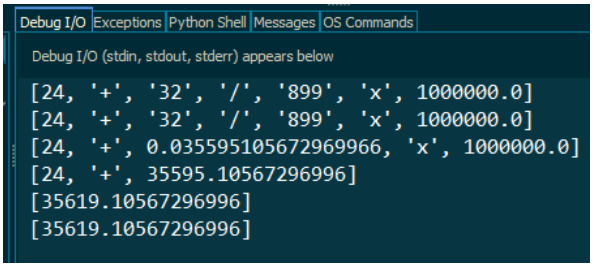
In order to be able to test the calculation process in my calculator, I decided to test them separately from my main program. This is due to the fact that my program only works properly when everything is implemented correctly, so without finishing the calculation function, I can't test the rest of the program. So I will test segments of the calculation process, and use a predefined input, instead of the calculator itself.

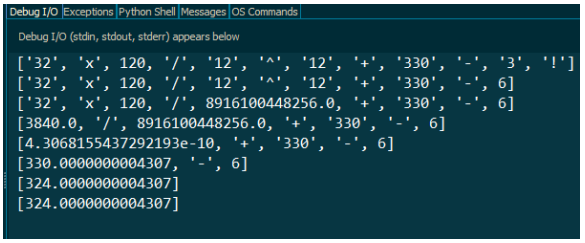


Testing results with Google calculator:







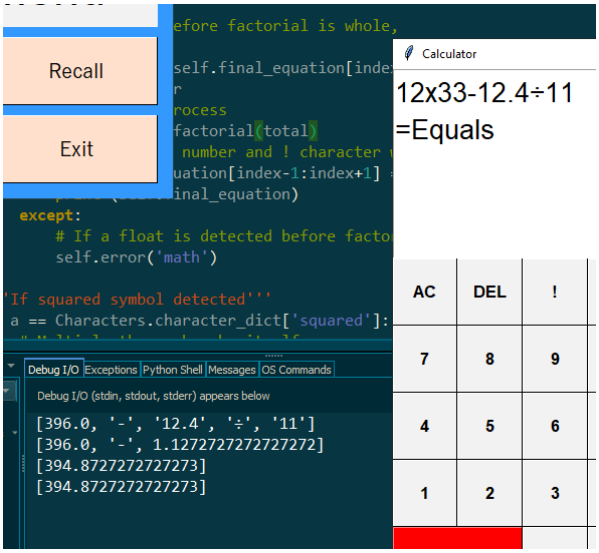


(Rounded value)

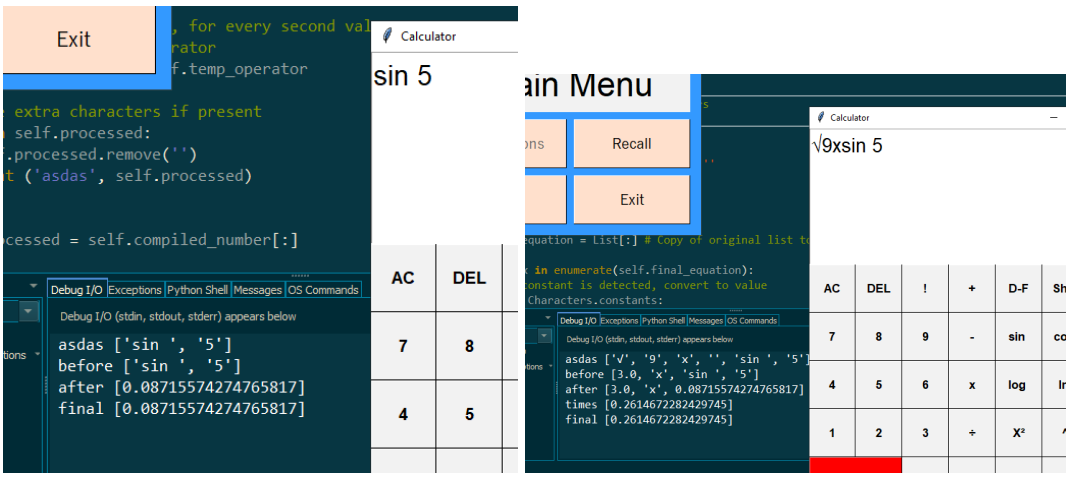
Sin, cos, tan have no limits, therefore, any value can be entered, whether negative or positive. No limits need to be placed on input. However, the inverse versions of sin and cos functions have a limit between -1 to 1, which needs to be set to prevent errors. Tan and inverse tan has no limit. Log and Ln also require positive values only, or else an error occurs (as nothing to the power of 10 or *e* equals a negative).

Roots and square roots should also only take positive values. Powers can take negative values, since this is basically the power of the inverse value. Factorials can only take positive, whole numbers.

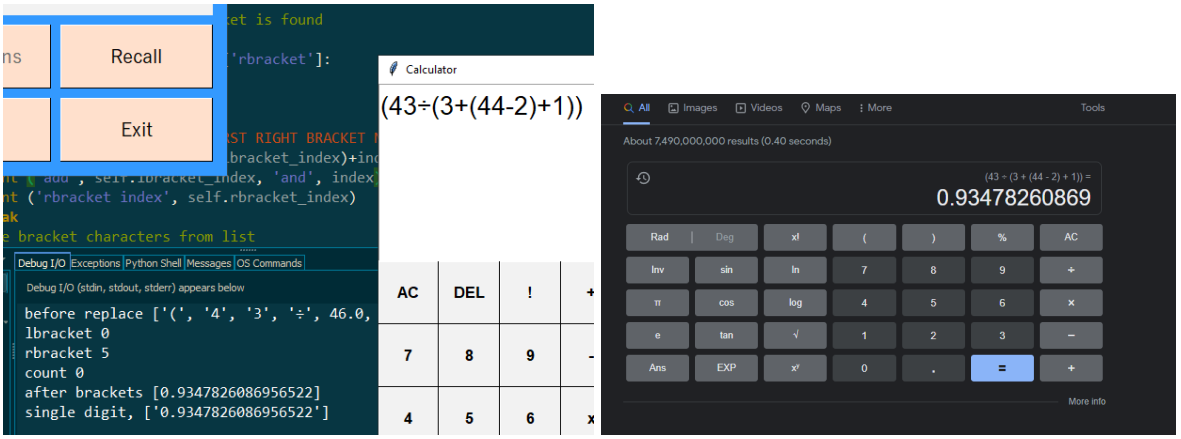
Program put together, with calculations. At this stage, there were still a few errors.



Program can successfully calculate complex arithmetics

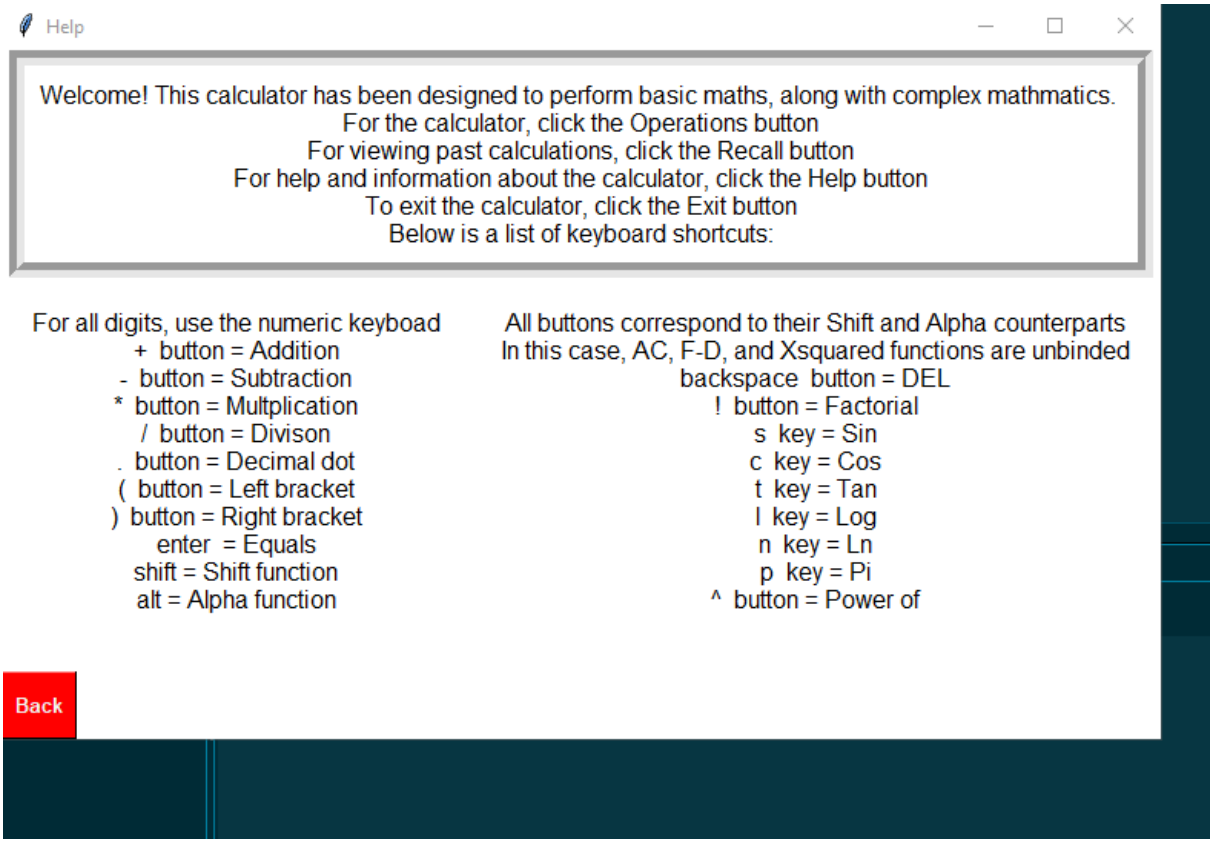


Program can succesfuly parse brackets and calculate according to BEDMAS



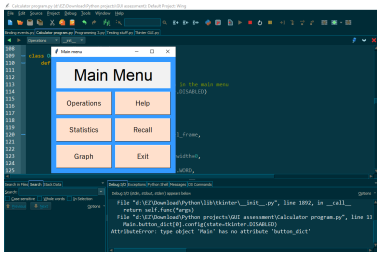
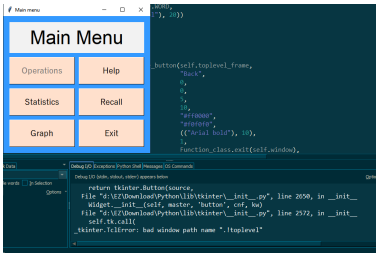
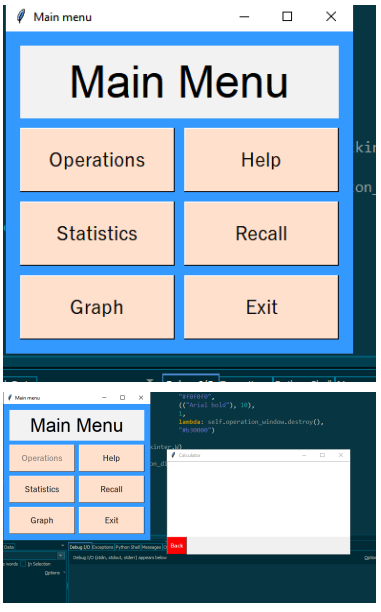
Help class:

Finished help guide and info

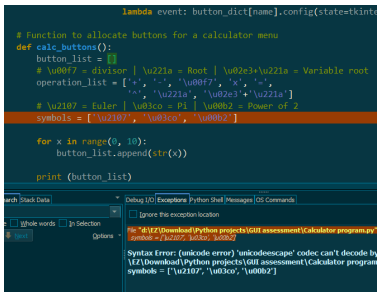
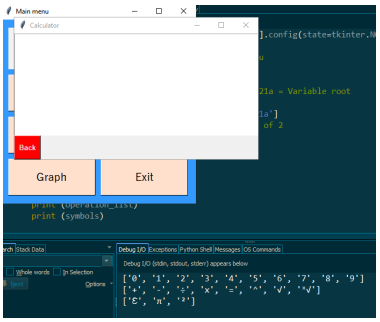
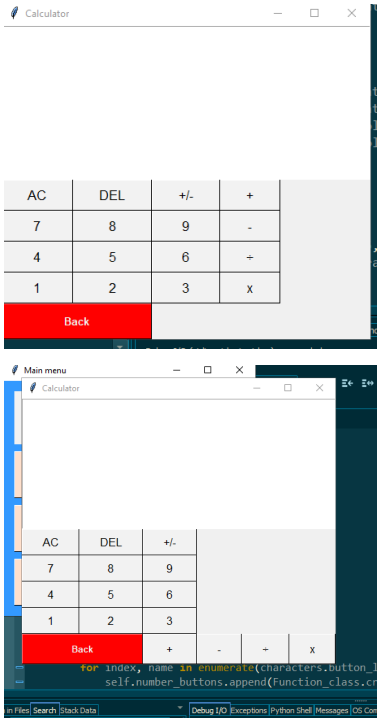
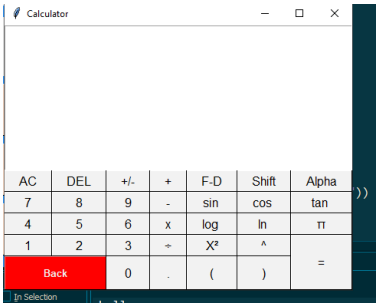
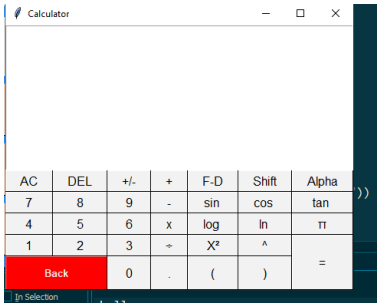


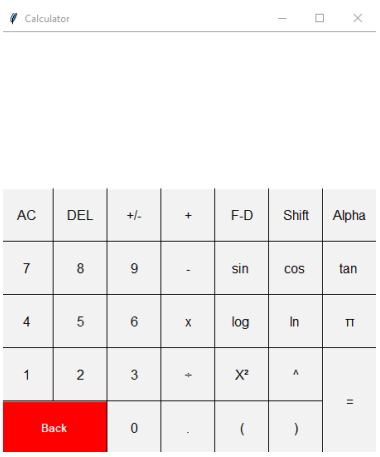
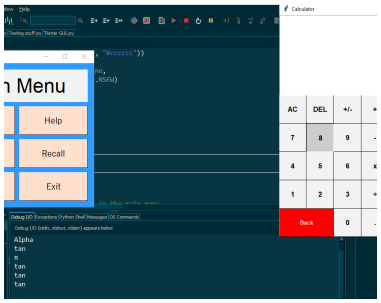
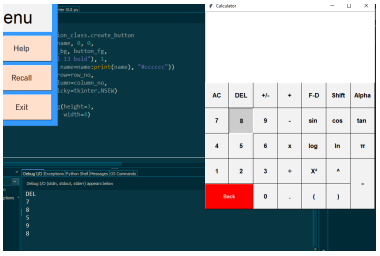
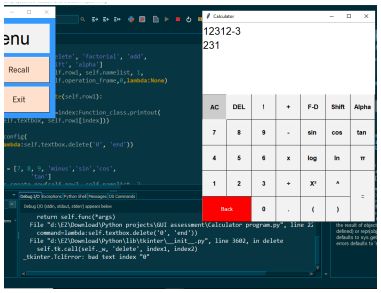
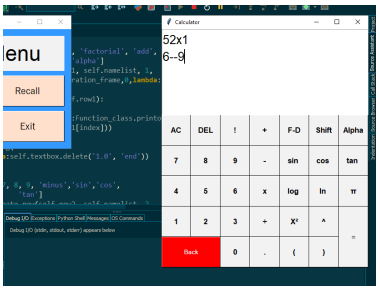
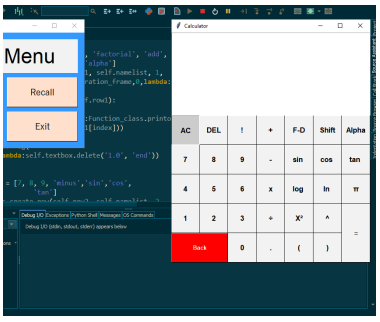
Unfortunately, the Recall class couldn't be implemented in time

Errors:

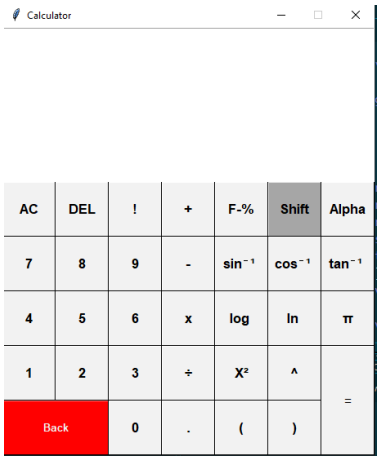
Program cannot detect button press with different classes	Fixed, but now the toplevel window won't open	Fully fixed:
		

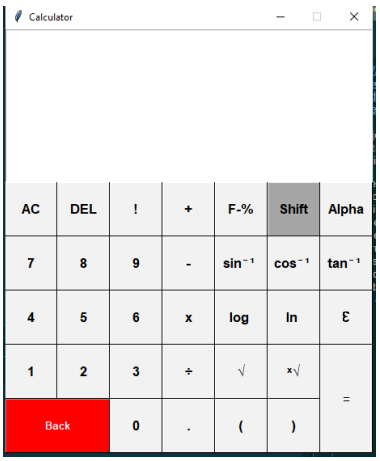
:

Python cannot decode unicode character 	Fixed, used the letter o instead of number zero for unicode 	
Buttons are repositioned in new positions whenever I try add new buttons 	Fixed, use a row-by-row button generation function. More details in program trialling. 	
Buttons are unevenly spread in GUI 	Fixed using rowconfigure and columnconfigure	

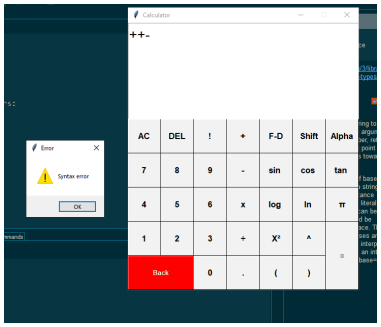
		
Buttons only print out the same value for each row 	Fixed 	
Clear button doesn't clear 	AC button clears  	
Program doesn't convert all shift buttons when	Fixed, now they all do	

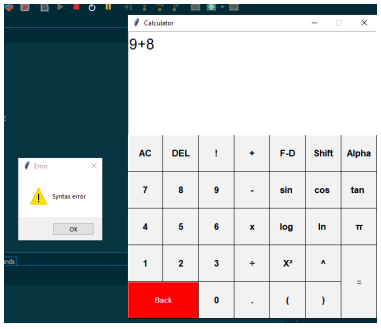
shifted



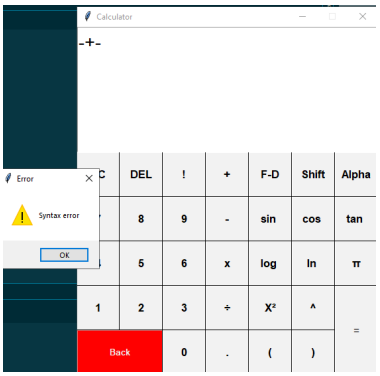


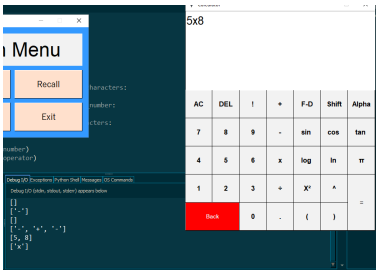
Program detects errors all the time



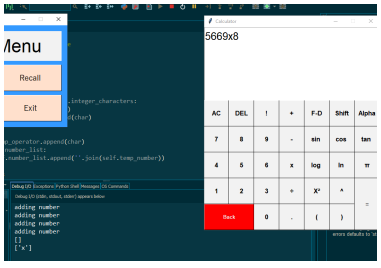


Fixed, syntax error works

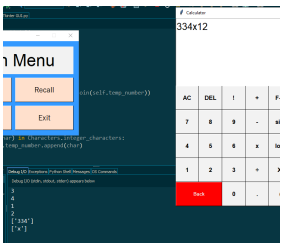




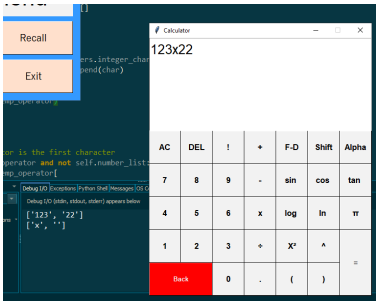
Program detects numbers but doesn't add them to the list



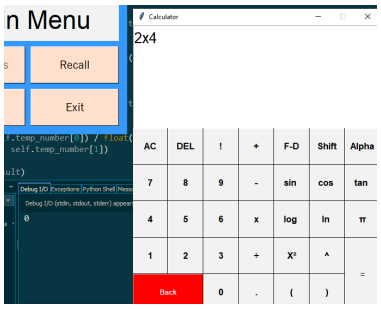
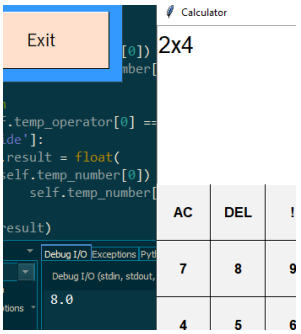
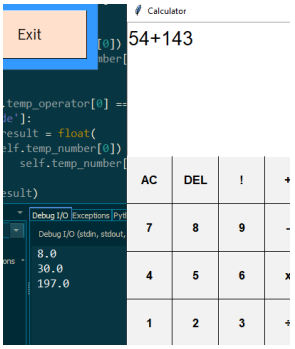
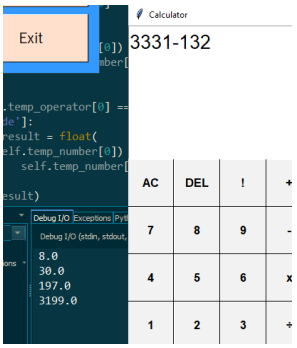
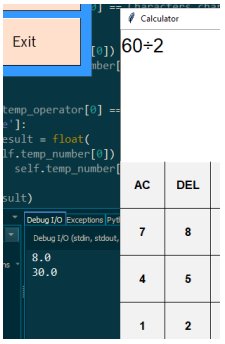
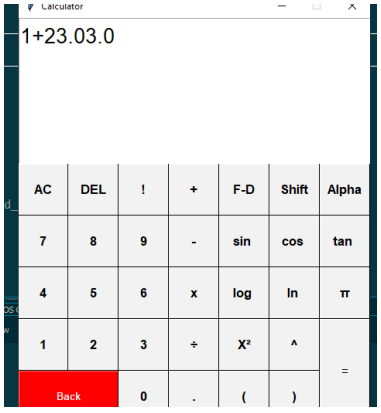
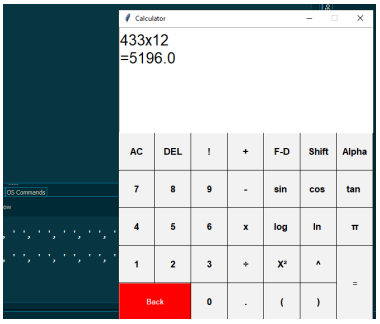
Fixed, now program can concatenate multiple digit numbers, but only the first number

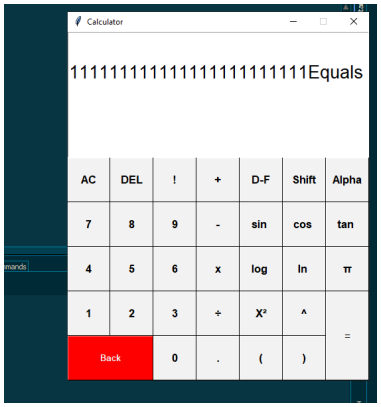


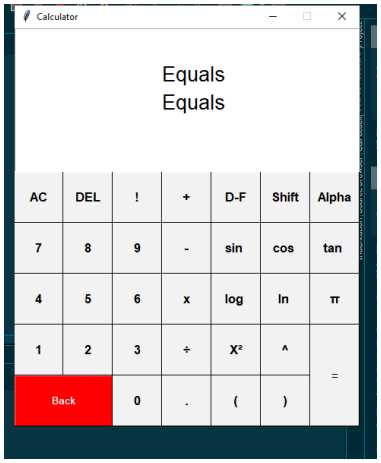
Fully fixed



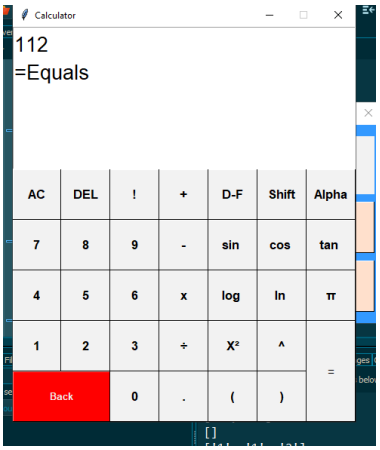
(the blank string is so the program knows it's the end of the line)

Program can’t perform basic operations 	Fixed, works for all basic operations  	(continued)  
Program outputs result in the same line 	Program can output in a new line 	
Program formats result properly, but not when using blank spaces	After considering the difficulty of Python’s default character sizes, it became too difficult to evenly space out the output result since Python’s blank characters are slightly uneven to	

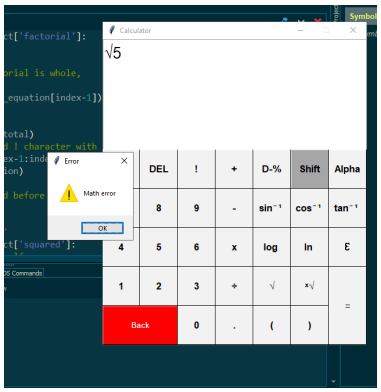




Tkinter's character widths.



Program can't calculate roots

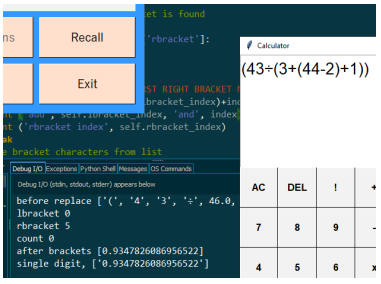


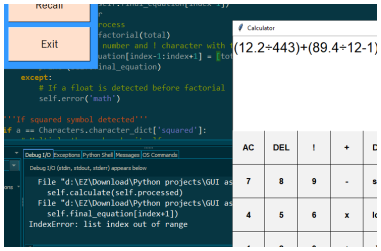
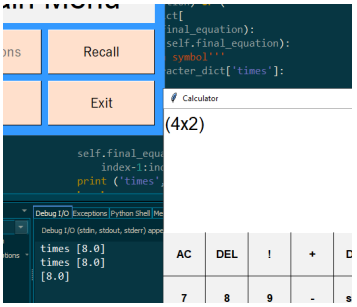
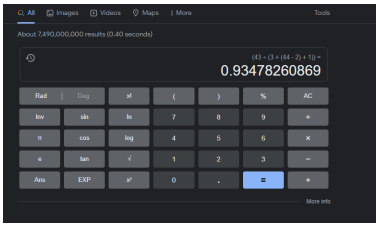
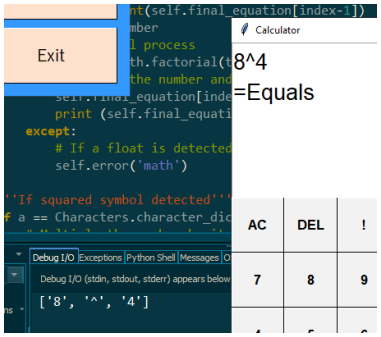
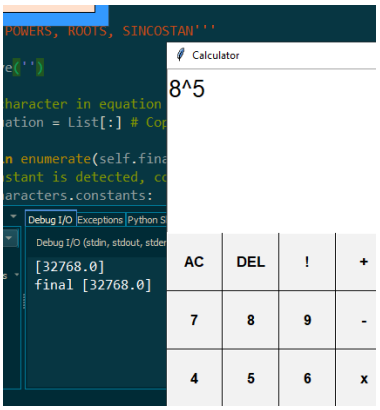
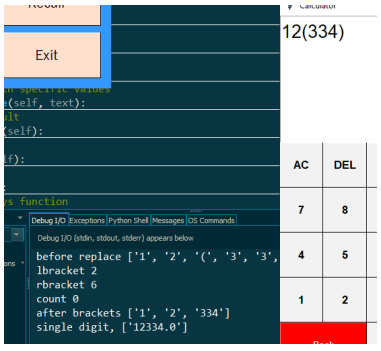
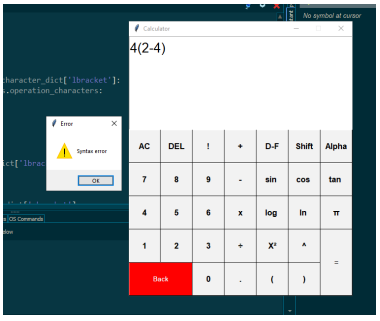
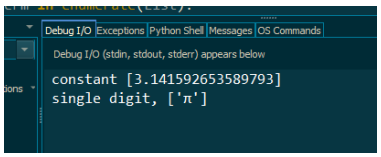
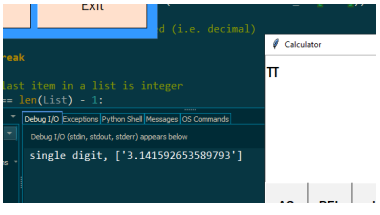
Fixed, an error where positive numbers are rejected instead of negative numbers

Program can't calculate bracketed values

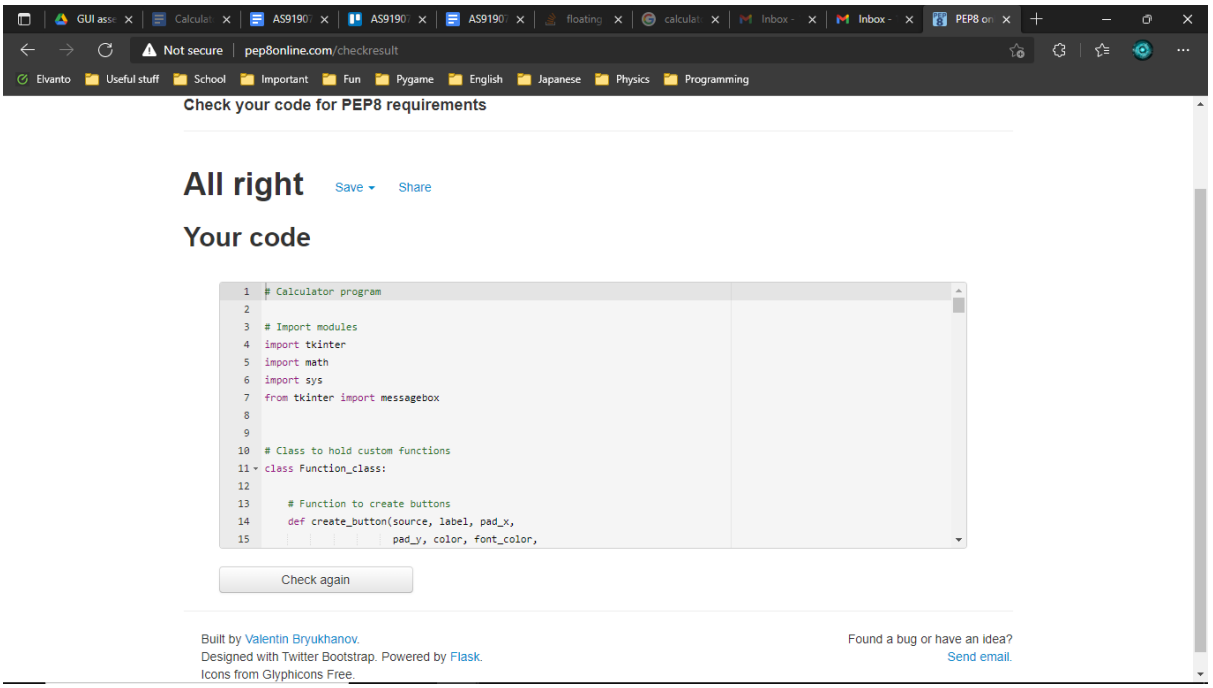
Fixed initial issue of not clearing a list

Fully fixed



		
Program can’t compute powers to a number 	Fixed the indexing, so program replaces the correct characters with answer 	
Program joins a number and bracketed number instead of multiplying 	Fixed, now an error appears instead (this issue is noted in the Help section) 	
Program can’t convert constant symbols into values 	Fixed, had to convert the input to string first 	

Pep8 check:



Testing is done in a separate document

Relevant implications:

Functionality:

The program works at its basic level. The main program is the calculator program, which can take an input as an equation, and convert that equation into a number output by performing maths based on the input. My program can accomplish this, including complex arithmetics such as trigonometry and powers. However, the program lacks the Recall and Answer functions, which were intended originally, but I lacked the time to do this. However, since the basic function of this program has been accomplished, the program can work for its main intended purpose, that is, to calculate an equation.

End-User requirements:

This program was designed for high-school/adult students that required a calculator that could also perform complex arithmetic operations. This program has accomplished that, being able to process a string of equations, and giving a correct output. The GUI is also simple to use and has information for help.

Usability:

- Match between system and real world:

I used typical conventions for character symbols, such as the √ symbol for the root button, which allows the user to instantly recognize it and know what the button does. Keyboard shortcuts have also been implemented to follow typical calculator buttons, such as the '-' key being a minus. The '+' input is accomplished by holding down the 'shift' and '=' buttons at the same time, which is also the typical method of writing '+".

- User control and freedom:
  The user is practically free to input anything they like, AS LONG as it exists in the program's database of characters. This allows the user to input any character that is shown in the menu, but they cannot enter random characters such as letters, as this would break the program since a calculator can't process letters, they can only process operational characters and numbers.

- Diagnose and recover from errors:
  The program is able to analyse the input from users, and if the syntax is incorrect (e.g. '(4+)-(((2)') the program can raise an error message that doesn't break the program. The program can also detect math errors, which would be impossible to calculate in reality (e.g. root of a negative number)

Improvements:
I wished that the Graph and Statistics classes would've been implemented, but given the current time schedule, this could not be accomplished. If it was, it would give the program a better reason to use, instead of a generic calculator.

The Recall function also couldn't be implemented in time, which sucks, but because of time constraints, I had to let it go.

Currently, the textbox only displays one answer, then the user has to wipe it out to do another equation. I would preferably like it to keep the answer and previous calculation on screen, like a scientific calculator, and scrollable to the user so they can see their past calculations on-screen.

Overall, the program had many features planned, which were scrapped due to time constraints. Next time, I would like to include all the promised features in a program.