CMP-7009A Advanced Programming Concepts and Techniques

Project Report - 12th January, 2023

Interpreter for Maths Visualisation Software - MathChamp

Group Members:
Christopher Gavey, Ebin Paul, Max James, Aswin Sasi, Soniya Avarachan

https://github.com/paul3bin/maths-interpreter

School of Computing Sciences, University of East Anglia

Version 1.0

Contents

Abstract

This project aimed to develop an intuitive Python-focused Mathematical Interpreter, *MathChamp*, with an interactive Graphical User Interface (GUI) utilising *PyQt5*. The first step was researching similar interpreters, such as *MATLAB*. Following the analysis of approaches and completing the design process, lexical analysis was planned in detail and developed. Upon completion of the lexer, an abstract syntax tree (AST) was generated for syntactic analysis and integrated. The GUI was designed, developed, and integrated in the next stage with the core *MathChamp* program. The final step of implementation was visualising expressions with settings-rich plots and implementing additional features. Throughout the project, *MathChamp* followed the agile scrum methodology for consistent iterative and incremental development (IID) across the project life cycle. The project was split into development sprints, and roles were allocated for these sprints across the development team, with set and measured goals and deadlines. The final *MathChamp* solution delivers a modern and powerful user interface where users can enter expressions, statements, and commands with function calls using custom syntax. Variables can be defined, manipulated, and visualised, and additional features include zero crossings, factorials, and Boolean operations.

Chapter 1

Introduction

1.1 Project statement

This project will utilise the *Python* programming language to develop a maths interpreter solution. The interpreter, *MathChamp*, will evaluate expressions, support functionality for defining variables, calling predefined functions, and visualising functions of two or more variables on a plot in the form of *y=f(x)*. Users will be able to enter expressions, statements and commands with function calls using a custom syntax, and make changes to plotting utilising helpful graphic settings. The modern and intuitive graphical user interface (GUI) will be developed using *PyQt5*.

At a minimum, weekly meetings will be set to discuss the project's progress, issues and bugs, testing, delegated tasks, and, finally, to evenly delegate new tasks (upon completion of those previously assigned). The final deliverable will support additional functionality, as detailed in the MoSCoW below.

1.2 MoSCoW

MUST HAVE:

- Well tested, well documented software
- Clean code
- Intuitive GUI
- Variable assignment (and visualisation)
- Accurate BODMAS Order of Operations/Precedence
- Visualisation of Mathematical Functions

Firstly, the maths interpreter's core functionality and user experience will be the focus; with any additional features outlined in the 'should' or 'could' sections being addressed in future sprints depending on progress. Following an iterative and incremental Development (IID) cycle, the code will be tested regularly as functionality and features are added over time through unit testing and manual test where necessary (mostly for the GUI). Before implementing additional features, the code will be improved to increase efficiency, check for logic errors in calculations, and to ensure good coding practice is used.

SHOULD HAVE:

- All basic arithmetic expressions
- Excellent Graphical User Interface (GUI)
- Professionally presented plots with configuration settings

The interface and visualisation of functions will follow great design principles, plotting will provide multiple intuitive settings including zoom, subplot configuration, editable axis and curve parameters, view reset, and the ability to save the view.

COULD HAVE:

- Logical/Boolean Expressions
- Factorials
- Zero Crossings
- Polar Curve Spirograph
- Operations on Matrices

For additional functionality, first boolean expressions will be prioritised for integration into the interpreter, followed by zero crossings. If within the time-frame of the project delivery date, operations on matrices and a Spirograph in polar coordinates will too be explored. However, additional unconsidered features may also be implemented if they provide useful functionality to the interpreter.

WONT HAVE:

- Complex Numbers
- Differentiation

Due to group changes, three weeks of combined development time was absent. Therefore, complex numbers and differentiation will not be considered for implementation as additional features because of the considerable overhead, and therefore risk of not meeting the project due date to a high standard.

1.3 Report structure

The remainder of this report will be structured as follows: Background information on the MathChamp interpreter will be discussed, followed by lo-fi and medium-fi designs. The methodologies utilised in development will then be introduced and explained in detail, followed by the stages of implementation sprint-by-sprint and how each sprint correlates to the specific methodologies (where necessary). Next, the testing stages of the interpreter will be explored and discussed in detail. Finally, a project conclusion and discussion will be covered, along with individual contributions to the project. Appendices such as UML diagrams and test plans will appear at the end of the report.

Chapter 2

Background

2.1 Background

Various background resources have been used to understand the theoretical underpinnings of the techniques applied, namely, lexical and syntactic analysis. All resource references can be found in in the Bibliography following Chapter 6. The references have also been included in the code files they're applicable to; appearing at the top of the program as a comment. Of all resources, the most helpful in the development of MathChamp was the presentation slides from Lecture 2: Programming Languages, especially those covering interpreters and compilers, slides 26 - 82. The knowledge learnt from the slides and lectures was combined with further research on the design and development of lexical and syntactic analysis. The conducted research included blogs on lexing and parsing processes to provide some real-world insight into how the development of MathChamp could be tackled and planned for appropriately. MATLAB was too closely studied to assist in aligning the expectations and goals of an intuitive mathematical interpreter.

Chapter 3

Methodology

3.1 Method 1 - Agile Scrum

The development of MathChamp was split over four major sprints, between one or two weeks, as discussed in chapter 4. The Agile Scrum methodology was closely followed. Once the goals of MathChamp were determined as a team and clear sprint goals defined, work began. No specific team member was allocated the role of scrum master. However, each member regularly checked on the progress of other members as relevant to them. For example, the team member responsible for the GUI discussing progress with the members responsible for the core program and vice versa, to aid in integrating the systems and completion in a similar time-frame.

Weekly group meetings were held each Saturday and Wednesday for two hours at noon, and attended consistently by most members. To begin the meetings, progress since the last meeting was first discussed, along with any issues faced in completing the allocated tasks member by member. Issues faced, primarily technical, were often resolved in these meetings, therefore a critical tool in our progress. Once tasks had been completed and new functionality developed, this was tested before and after implementation into the core program by the team member responsible for rigorous testing. Upon completion of all allocated tasks for a sprint, a sprint retrospective was conducted to discuss the following:

1. What went well?
2. What can we improve?
3. How can we improve it?

The next sprint would then be planned with a set of appropriate goals, a deadline, and evenly distributed tasks amongst the team. Before closing each meeting, the location and time would be confirmed for the next, and an email reminder/calendar item would be sent out to all team members using the Booker System. For general communication, a WhatsApp group was utilised from the beginning to address any questions or issues in between meetings.

3.2 Method 2 - GitHub

GitHub was, arguably, the most critical collaboration tool employed. At the beginning of development, a repository was setup, and each group member was added. In the beginning, three branches were setup; main; development; and testing. The team members also setup GitHub Desktop and connected the repository on each of their computers to further aid collaboration. Once new code was developed, it would first be uploaded to development and then be tested by team member responsible for all the testing. Upon confirmation that the code was functional, reliable, and bug-free, and the test plan updated, it would then be uploaded to the main branch. Following sprint one, branches were created for the GUI, functions, and plotting. Bugs, enhancements, important goals,

and documentation were listed in the Issues Module of the repository, regularly kept up to date, and closed once completed. This was especially important in identifying, logging, and tracking bugs, which were easily accessed by utilising GitHub desktop and the Visual Studio Code GitHub Plugin. As the project progressed and dependencies grew, a helpful README file was uploaded with instructions for downloading the MathChamp program, creating a virtual environment (venv), and installing dependencies.

3.3 Method 3 - Abstract Syntax Tree (AST)

Figure 3.1: Expression Processing
ruslanspivak.com

The Abstract Syntax Tree data structure was utilised for the interpreter to generate the machine code from the list of tokens. The tokens are parsed, and the root node of the syntax tree is returned, following the BODMAS order of operations to ensure correct precedence. An input string is collected from the user and calls the Interpreter Class, which is instantiated with the input string. The Lexer first creates a list of characters from the user input string. The input string is looped through to identify each token using the Generate_Tokens method in the Lexer Class; the Get_Tokens method then returns the list of tokens. The Interpreter Class gets the list of tokens and creates an object of the Parser Class (which passes the list of tokens). The AST is created from this list and returns the root node of the tree, as shown in figure 3.2.

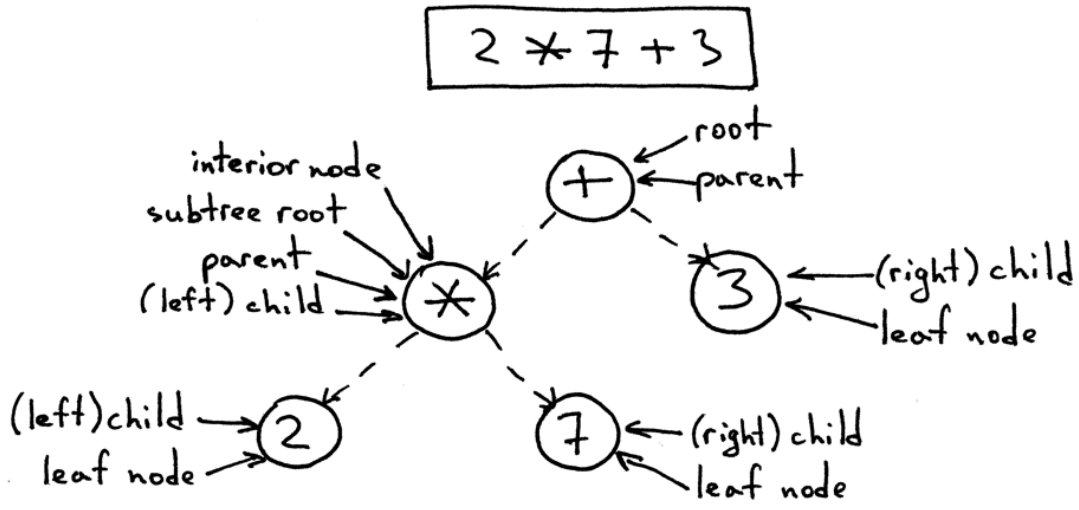

Figure 3.2: Abstract Syntax Tree
ruslanspivak.com

3.3.1 Polymorphism

Each class in the Nodes.py file has its own Get_Node_Value method and a Magic/Dunder method. This approach was selected so that rather than checking the token type of each node (such as Integer or Float), calling the method Get_Node_Value will return the node's value regardless of its class. By using our OperandNode class, this bypasses the requirement to create a different node class for every operator. In turn, this increased the code readability and maintainability, but more importantly, reduced execution time. Get_Node_Value is shown below in figure 3.3, in the OperandNode class.

```
class OperandNode:
    """
    Blueprint of a leaf node
    """

    def __init__(self, token: Token):
        self.token: Token = token
        self.__value = self.token.value

    def get_node_value(self):
        return self.__value

    def __str__(self):
        return f"{self.__value}"

    def __repr__(self):
        return self.__str__()
```

Figure 3.3: OperandNode Class
nodes.py

3.4 Method 4 - PyQt5

The critical tool employed for the development of the intuitive Graphical User Interface is the PyQt Python binding (PySide6) of the multi-platform Qt framework. PyQt5 is by far the most prevalent library to employ in a Python-based application requiring a modern GUI that functions well on a large assortment of devices [1]. Whilst the Tkinter Python binding is easier to use, it lacks much of the advanced functionality available in PyQt5. The desired level of advanced user interface design with in-depth tooling and overall functionality was not possible with Tkinter. However, besides more complex implementation, a disadvantage of PyQt5 is that Tkinter is built into Python, whereas PyQt5 is not and therefore requires a virtual environment (venv). Nevertheless, the additional time invested in implementing PyQt5 has paid off in delivering a modern, intuitive, and functionally scalable solution that despite being developed on a Mac, has been portable to Windows machines.

Matplotlib

Whist PyQtGraph is a popular choice for plotting functionality in Python, it does not deliver even somewhat close to the graphic quality made possible with Matplotlib. The main benefit of PyQtGraph is the fast execution speed; however, it is more focused on data acquisition and analysis in engineering applications, often for graphs that require constant updating [2]. For our mathematical interpreter, MathChamp, Matplotlib was the better option as it is geared towards MatLab programming, focusing on features, settings, and one-off plots. Paired with the advanced functionality of PyQt5 system support, it was the obvious choice. Our plotting function buttons give users the options to: pan in and out, alter subplot configurations, edit axis and curve parameters, reset their plot view following alterations, style graphs with a variety of line styles and colours, alter titles and labels, and save their custom plot. In addition, they can hover over graph points to display the values of the X and Y in real-time.

3.5 Method 5 - RegEx

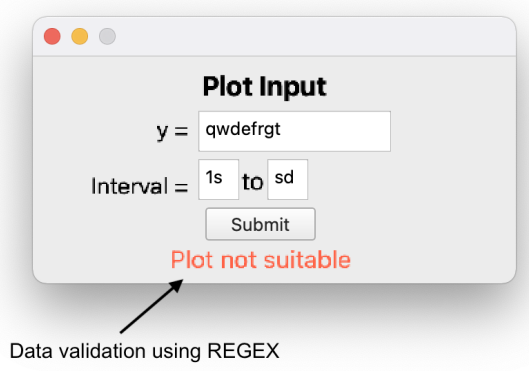

Figure 3.4: Showing REGEX being used to data validation

Regular expressions (RegEx) are employed to perform data validation on the pop-up windows, such as for Variable Assignment and Plotting. Therefore, users who are not familiar with the syntax of our custom language can use still use and interact with MathChamp thanks to increased accessibility. The data validation will prevent them from inputting invalid data into fields, such as entering the value of a variable as a String rather than an Integer or Float. In addition, error messages were included in MathChamp to guide users when they make input errors for performing the expressions themselves, such as entering an & or }.

3.6 Method 6 - Zero Crossing

It was decided to use the bisection method of finding zero crossings over the Newton-Raphson due to time constraints. This gives us an approximation of the crossing by diving the interval multiple times until a value close enough is reached. Please refer to figure 3.5, on the following page.

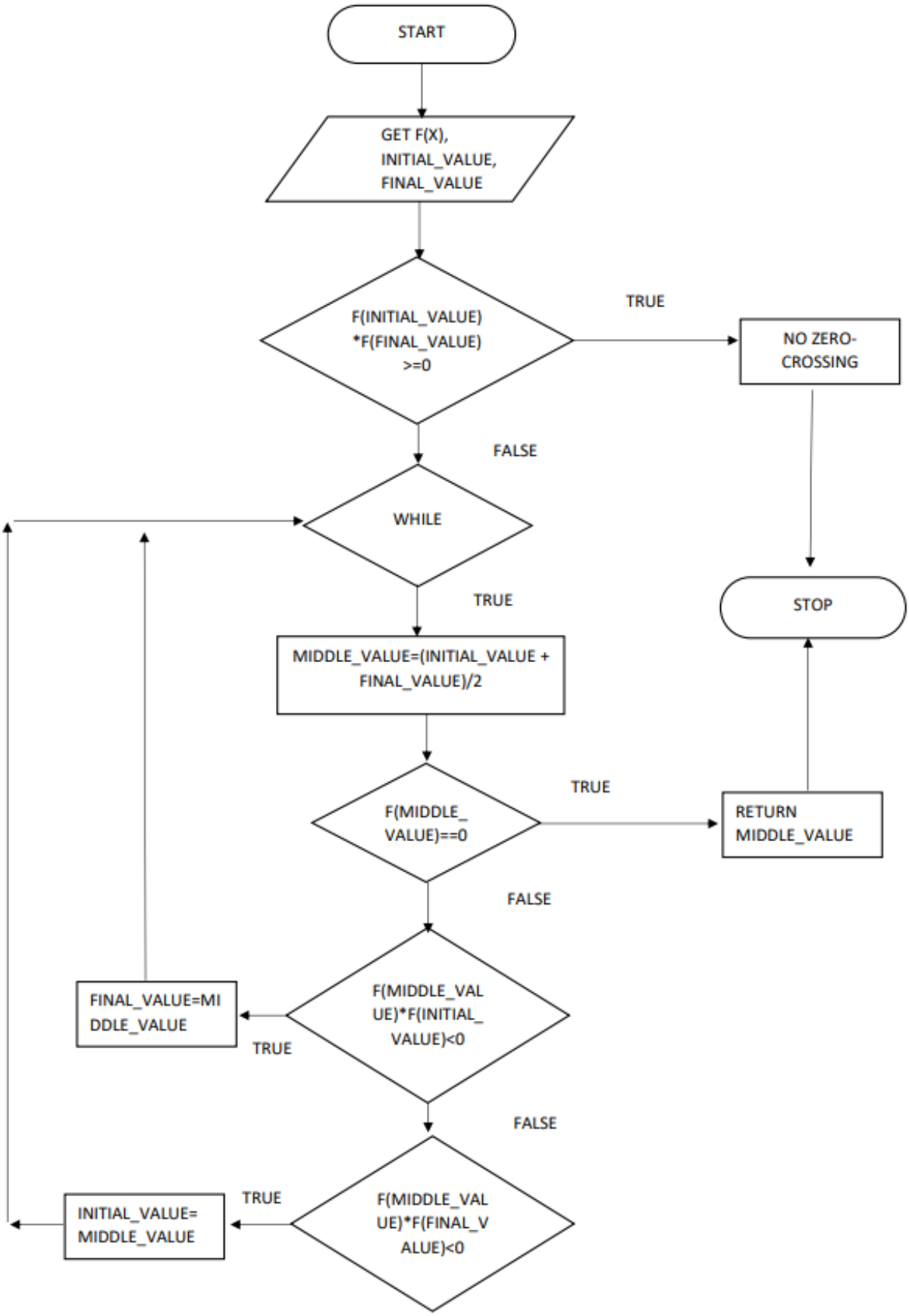

Figure 3.5: Bisection Method for Zero Crossings

3.7 Method 7 - Variable dependency

During development, it was brought to our attention that there was a way variables could be implemented that would mean that they have dependencies on one another. For example, if *'variable2'* is assigned the value of *'variable1'* the value of *'variable2'* should change as well. During the final sprint, one of the group members completed all their allocated tasks so they decided to spend their time implementing this functionality.

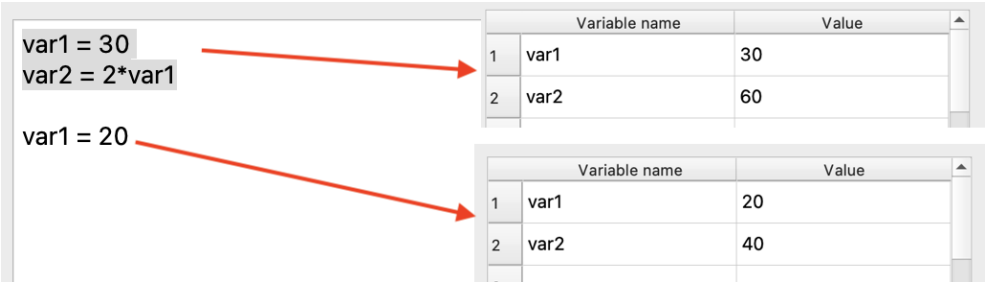

Figure 3.6: Showing variable dependency in our software

The way this was achieved was by expanding what information is stored within the program once a variable is declared. Previously, only the variable name and is value were stored within a dictionary; now there's two other dictionaries; one storing the variable name and the statement used to define it, and another with the variable name and a list which holds the names of the variables that have a dependency to the variable within the dictionary key.

Chapter 4

Implementation

4.1 Early sprints

For all sprints, a Gantt chart was used to keep track of what tasks were remaining and within what sprint, as shown in Appendix 6.3.

4.1.1 Sprint 1 (17/10/2022)

The first sprint of the MathChamp interpreter began at the start of the fourth week of the project timeline, on the 17th of October, 2022, a new group was formed following the withdrawal of our previous members. In turn, it was decided that due to most of the group's familiarity with Python, this would be the ideal language to develop in and ensure a robust and feature-rich interpreter solution.

The initial focus for this sprint is the lexical analyser/tokeniser. Our goal for completion of this sprint is the 24th of October, 2022 (1 week). Only the foundational tokens for MathChamp will be focused on at this stage. Therefore, the following tokens for identification and categorisation will be defined:

- INTEGER = *"INTEGER"*
- PLUS = *"PLUS"*
- MINUS = *"MINUS"*
- MULTIPLY = *"MULTIPLY"*
- DIVIDE = *"DIVIDE"*
- LEFT_PARENTHESIS = *"LEFT_PARENTHESIS"*
- RIGHT_PARENTHESIS = *"RIGHT_PARENTHESIS"*
- FLOAT = *"FLOAT"*
- END = *"END"*

```
class TokenType(Enum):
    """
    TokenType class declaration
    """

    INTEGER = "INTEGER"
    PLUS = "PLUS"
    MINUS = "MINUS"
    MULTIPLY = "MULTIPLY"
    DIVIDE = "DIVIDE"
    MODULO = "MODULO"
    LEFT_PARENTHESIS = "LEFT_PARENTHESIS"
    RIGHT_PARENTHESIS = "RIGHT_PARENTHESIS"
    FLOAT = "FLOAT"
    CARET = "CARET"
    ASSIGN = "ASSIGN"
    GT="GREATER_THAN"
    LT="LESS_THAN"
    IDENTIFIER = "IDENTIFIER"
    END = "END"
```

Figure 4.1: Completed Lexical Analyser - TokenType (Token.py)

```
class Token:
    """
    Token class declaration
    """

    def __init__(self, type, value=None):
        self.type: TokenType = type
        self.value: any = value

    # string representation of a token.
    def __str__(self):
        return f"Token({self.type.name}: {self.value})"

    def __repr__(self):
        return self.__str__()
```

Figure 4.2: Lexical Analyser - Token (Token.py)

As this project is of a large sizethe group was behind due to external factors such as group members changing modules and degrees, so tasks were divided so while one team member was working on the lexer and parser, another was developing UML class diagrams so that the flow of the system at a high-level could be understood to make development run smoother. This can be evidenced in Appendix 6.2.

4.1.2 Sprint 2 (24/10/2022)

Lexical analysis of the above foundational tokens was successful by the deadline. The second sprint begins on the 24th of October, 2022. It will first primarily focus on the expansion of the lexer tokens. Following this, the Abstract Syntax Tree (AST) will be defined from the identified tokens, and the syntactic analysis/parser will be developed. The deadline for completion will be set as the 30th October, 2022.

The following tokens will be added:

- CARET = *"CARET"*
- IDENTIFIER = *"IDENTIFIER"*
- ASSIGN = *"ASSIGN"*
- MODULO = *"MODULO"*

```
<assignment>    ->    <variable>         =          <expression>
<relational>    ->    <expression>   [(< | >)]     <expression>
<expression>    ->    <term>         [(+ | -)      <term>]*
<term>          ->    <term>         [(* | / | %)  <power>]*
<power>         ->    <factor>           ^         <power> | <factor>
<factor>        ->    <number>           |         (expression)
<number>        ->    <int>          <float>   |   <digit>
```

Figure 4.3: Syntax Analyser Backus–Naur Form

```
if self.operator.type == TokenType.PLUS:
    if self.left_node:
        return (
            self.left_node.get_node_value() + self.right_node.get_node_value()
        )

    return self.right_node.get_node_value()

elif self.operator.type == TokenType.MINUS:

    if self.left_node:
        return (
            self.left_node.get_node_value() - self.right_node.get_node_value()
        )

    return -self.right_node.get_node_value()

elif self.operator.type == TokenType.MULTIPLY:
    return self.left_node.get_node_value() * self.right_node.get_node_value()

elif self.operator.type == TokenType.DIVIDE:
    return self.left_node.get_node_value() / self.right_node.get_node_value()

elif self.operator.type == TokenType.CARET:
    return self.left_node.get_node_value() ** self.right_node.get_node_value()

elif self.operator.type == TokenType.MODULO:
    return self.left_node.get_node_value() % self.right_node.get_node_value()

elif self.operator.type == TokenType.LT:
    return self.left_node.get_node_value() < self.right_node.get_node_value()

elif self.operator.type == TokenType.GT:
    return self.left_node.get_node_value() > self.right_node.get_node_value()

elif self.operator.type == TokenType.ASSIGN:
    SYMBOL_TABLE[str(self.left_node)] = self.right_node.get_node_value()

    return SYMBOL_TABLE[str(self.left_node)]
```

Figure 4.4: Syntax Analyser - Operator Node Class (Nodes.py)

31/10/2022:

The deadline for completing the above tasks was met, the remainder of the lexical analyser was completed, and the parser was implemented successfully. The functionality of expressions was thoroughly tested using the manual test plan. A few precedence issues

were discovered which caused logic errors, such as the CARET^ operator; the issues were actioned and tested again to ensure correct, accurate functionality.

Next sprint task: As the basic functionality of MathChamp is now in place, developing the Graphical User Interface (GUI) utilising PyQt5 is the next focus. The first Lo-Fi design will be drawn up, and feedback will be collected from the group members to make changes to the design of the main window, variable assignment window, script window, and plotting window. Following this, a Medium-Fi design will be created in Microsoft Visio. Development of the interface will then begin following the agreed design specifications. The deadline for the GUI to be completed will be set as the 6th November, 2022.

4.2 Midway sprints

4.2.1 Sprint 3 (07/11/2022)

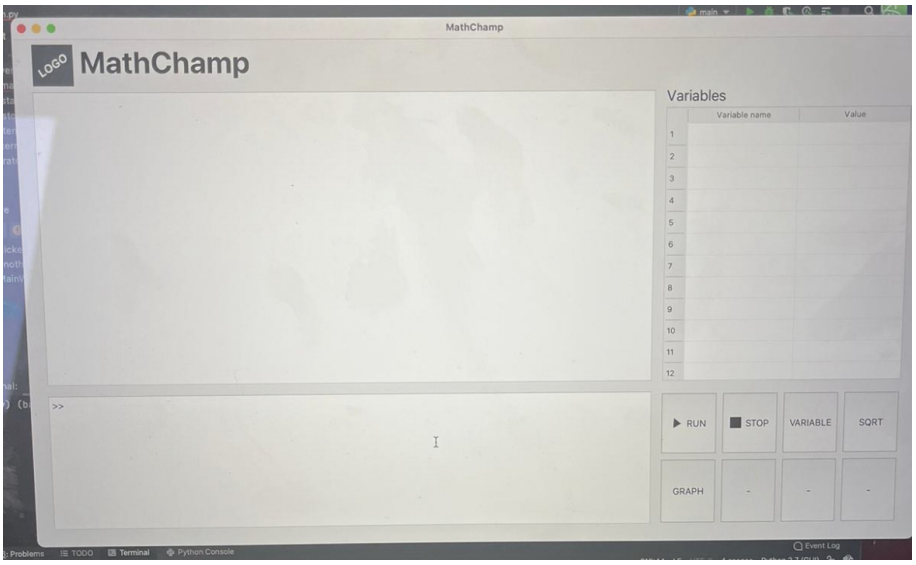

Figure 4.5: 1st iteration of GUI from Sprint 3

MathChamp is now at a stage where it has a working lexer, parser, and GUI (ready to be integrated with the main program). Using PyQt5 all the elements are present and designed in a way that implementation with the interpreter code should be possible next week. The interpreter should be able to be output via the output box command line. Sprint 3 can now begin and will focus on integrating the GUI with the main program and, importantly, implementing variable assignment functionality into the application. The deadline for this first stage of sprint three will be set as the 13th of November 2022.

13/11/2022:

The Variable assignment functionality has been successfully implemented into MathChamp and tested thoroughly. Users can define variables, manipulate the assigned values, and use them in expressions paired with Floats, Integers, and other Variables in complex calculations. In addition, the GUI has too been successfully integrated.

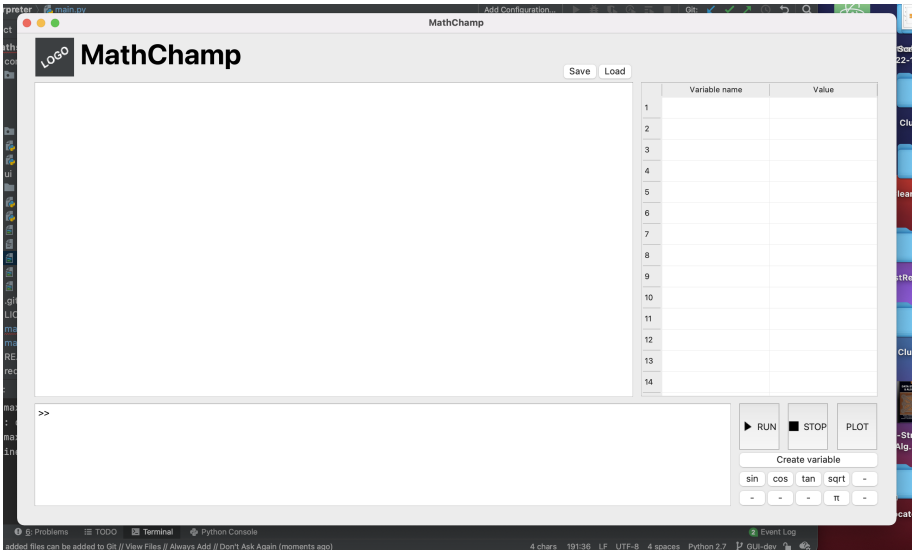

Figure 4.6: 2nd iteration of GUI from Sprint 3 (13/11/2022)

Users can now enter expressions, statements, and commands from the interpreter directly from the intuitive GUI. The GUI was able to be developed quickly as meetings were held amongst the team to collaboratively create a lo-fi and medium-fi prototype of the GUI as evidence in Appendix 6.1. Since last week, there has been the addition of being able to save, and load scripts at the top corner. Once the user presses 'run' add the commands within the script will run in succession; alternatively you could just run a section of the script by highlighting those commands. Using the 'Create Variable' button, users are prompted with the Create Variable window where they can enter the name and value; which will later be constrained with an error message using ReGEX to do data validation. Created variables will then be shown within a symbols table on the right side of the screen (above the function buttons).

```
def assignment(self):

    result = self.comparison()
    while (
        self.current_token.type != TokenType.END
        and self.current_token
        and self.current_token.type == TokenType.ASSIGN
    ):
        operator = self.current_token
        self.next_token()
        result = OperatorNode(result, operator, self.expression())

    return result
```

Figure 4.7: Syntax Analyser - Variable Assignment (syntaxAnalyzer.py)

4.2.2 Sprint 4 (14/11/2022)

In Sprint 4, the focus is on implementing plotting into MathChamp to visualise functions of variables and expressions. The plots will allow users to zoom in and out, alter the subplot configurations, edit axis and curve parameters, reset their view following alterations (with a home button), and to save the plot or custom plot view. Once plotting is functional, Zero Crossings functionality will be looked into. For further additional functionality, development on including Boolean Operators such as Less Than and Greater Than will be started, utilising the following new tokens:

- GT = *"GREATER_THAN"*
- LT = *"LESS_THAN"*

Further functions (which are yet to be confirmed) will also explored at this stage of development; there is a prospect for operations on matrices and factorials. The deadline for completion of plotting and Boolean Operators will be set as the 21st of November, 2022.

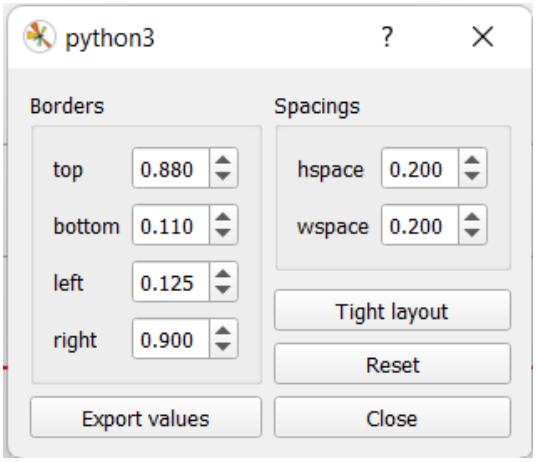

Figure 4.8: Plotting - General Configuration Options

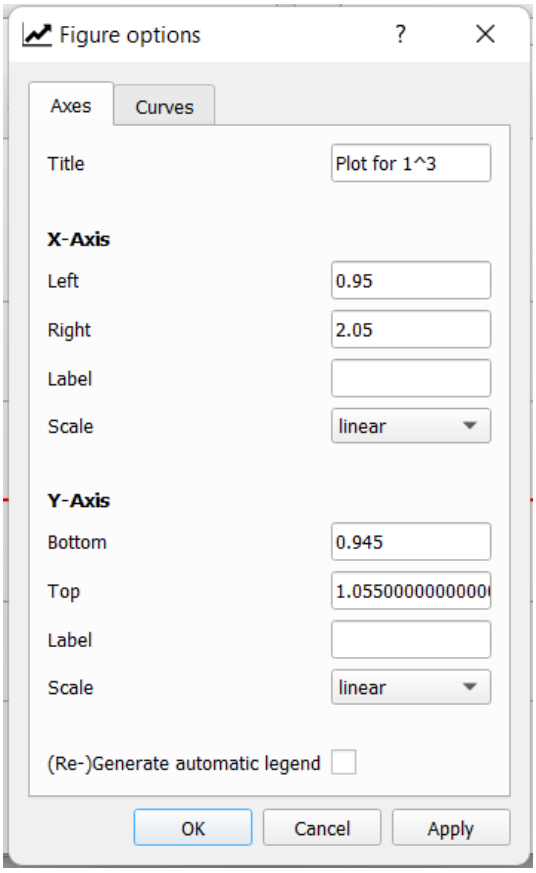

Figure 4.9: Plotting - Subplot Configuration Options (Axes)

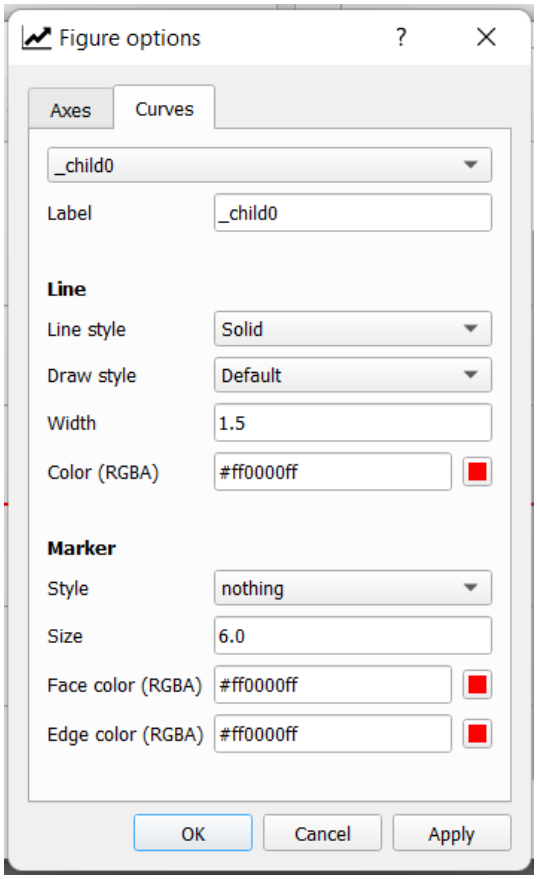

Figure 4.10: Plotting - Subplot Configuration Options (Curves)

The plotting was done using our interpreter, and using the output of the interpreter to popular Python lists which then correspond to the x and y axis of the plot. The interval the user inputs populates a list corresponding to the x axis, and then the 'x' within the function is substituted by the values in the aforementioned x-axis list to be executed through the interpreter and populate y-axis list. This is also the sprint where variable dependency functionality was successfully added to our software. This is further explained and explored within 3.7.

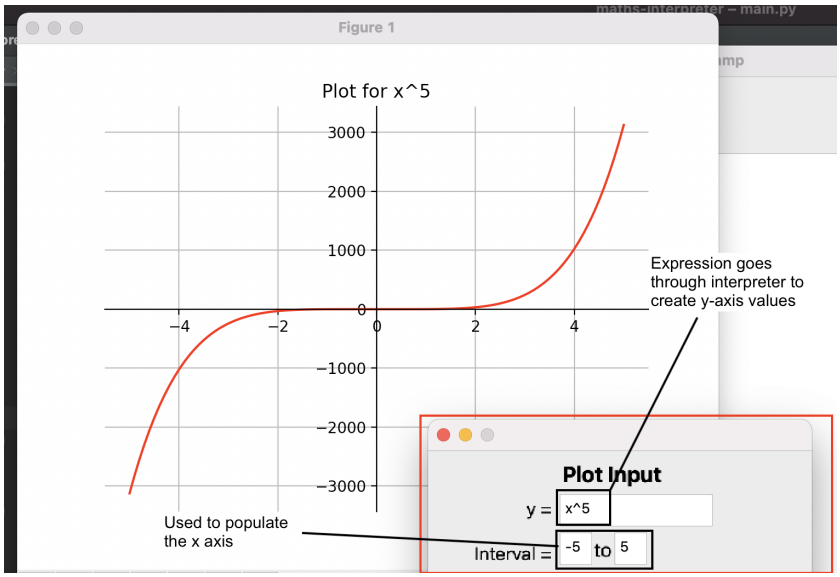

Figure 4.11: Example plot

4.3 Final implementation

In terms of functionality and appearance, the GUI is completely finished. The only changes that will need to be changed will be ones in response to changes to the interpreter (such changing the function buttons in response to what functions are included and omitted due to changing scope).

Figure 4.12: 3rd iteration of GUI

29/11/2022

Factorials were successfully implemented into the code, through iteration during development. Originally, factorial was implemented through regular mathematical notation (such as *10!*) and was working correctly. However, as the implementation of functions was added to the software it made more sense to implement factorials as a function such as "fact(10)". Therefore, this is how factorials are done in the final program.

While the GUI was finished from a functionality view point, there was one problem that was left unaddressed. The GUI was developed using a Mac, which meant that which the application worked functionally on all machines; some of the spacing between elements wasn't correct (demonstrated in 4.13) and the text size was too large (4.14). However, this was easily fixed by simply setting the style sheet within the program so there's uniformity between all operating systems.

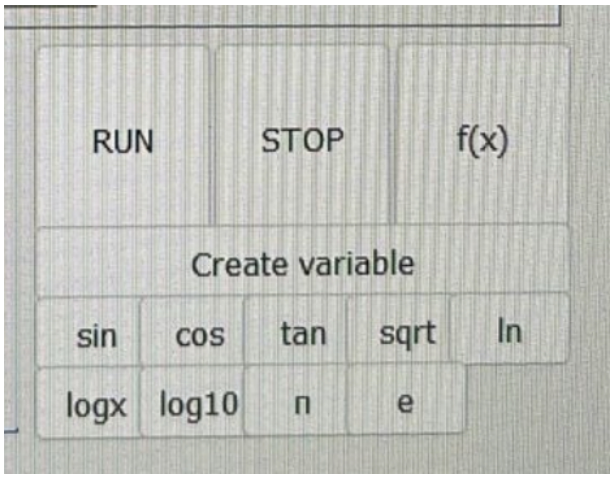

Figure 4.13: Overlapping elements on Windows

Figure 4.14: Incorrect text size on Windows

Additionally, zero crossings were implemented into our system. This is done by having a parameter on the plot input window which determines the interval for zero crossing which then displays as text in the top-corner of the graph. This is further explored in section 3.6.

Furthermore, functions such as SIN, COS, TAN, and FACT were also prioritised during our final sprint. Initially issues were faced with the order of precedence, but the issue was identified and resolved the bug. Following the implementation and testing of these functions, Zero Crossings was refined and tested against the new functionalities. The final actions in this implementation stage included increasing the code efficiency, improving and adding further comments, and final testing.

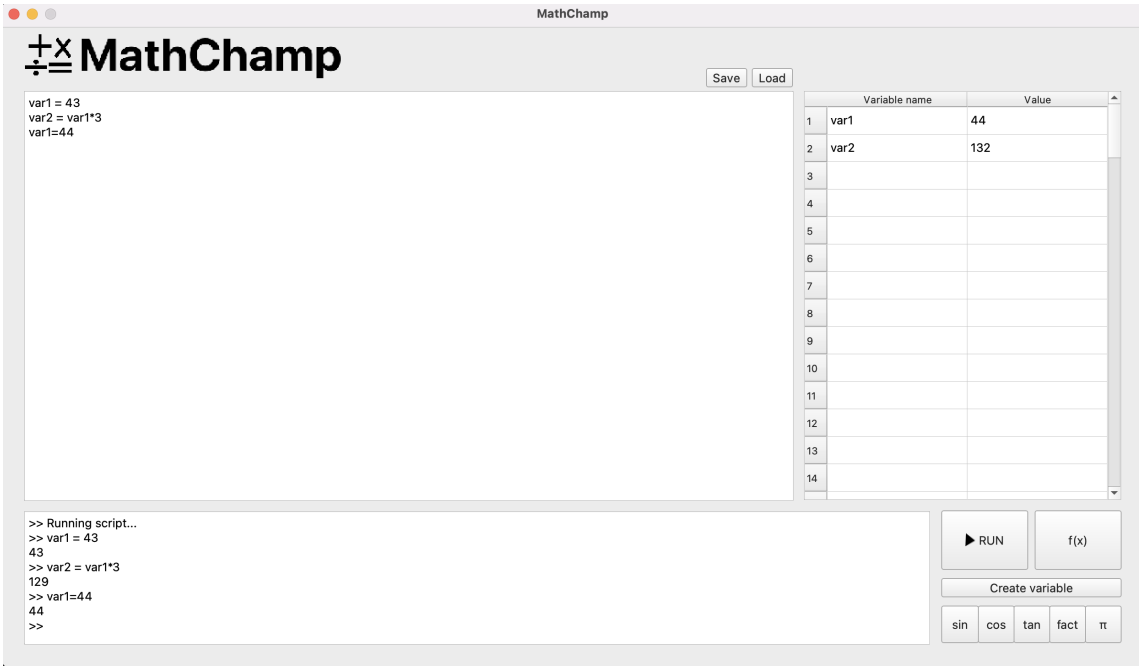

Figure 4.15: Final main window

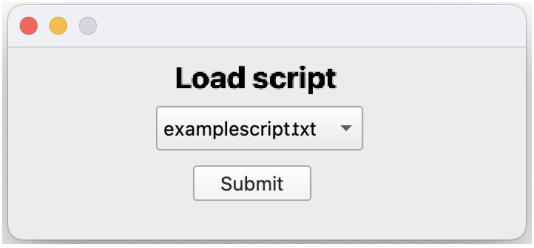

Figure 4.16: Final load window

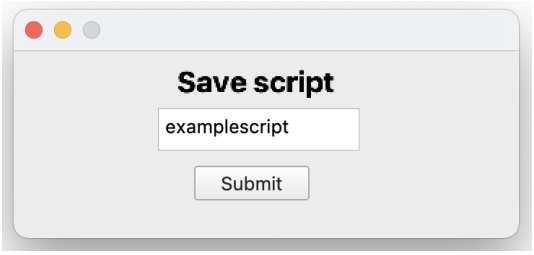

Figure 4.17: Final save window

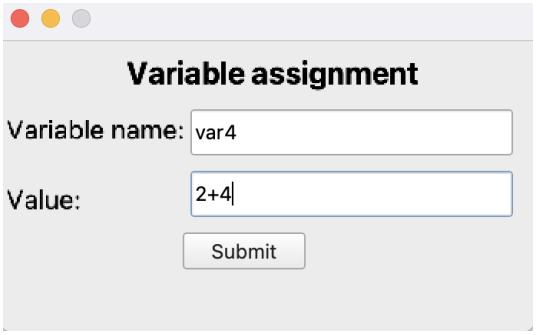

Figure 4.18: Final variable window

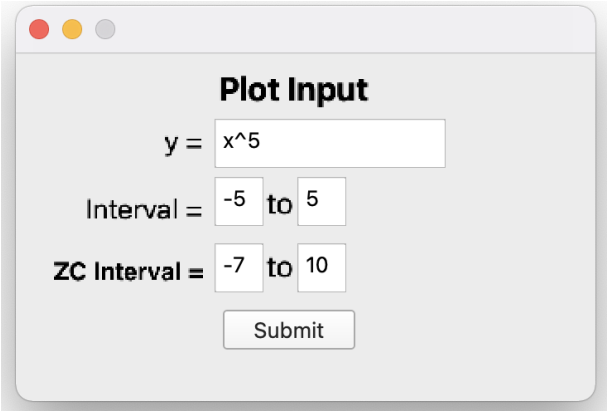

Figure 4.19: Final plot input window

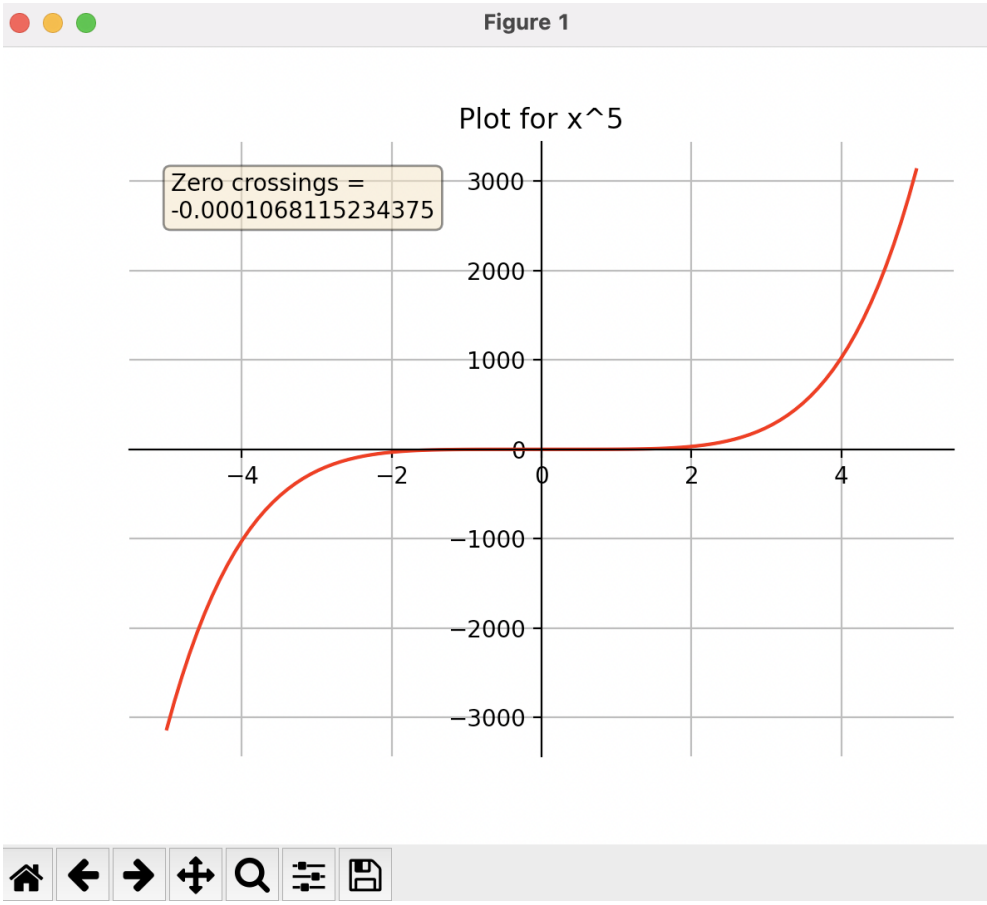

Figure 4.20: Final plot window using Matplotlib

Chapter 5

Testing

5.1 Manual Test Plan

The manual test plan are is shown in the appendices at the end of this report; figures 6.13, 6.14, 6.15, 6.17. Manual testing was performed at every stage of the development, before the implementation of new code and after. There was a critical focus on ensuring the correct order of precedence to ensure accurate program logic. All aspects of GUI functionality were too tested, including all buttons, plotting, and general system compatibility such as operating system support, window resizing, and input validation.

Note: MathChamp was tested on Windows 8.1, 10, 11, and MacOS. The code was written using Windows, and the GUI developed using MacOS.

5.2 Unit Testing

The Pytest Framework was incorporated for Unit Testing to ensure complex and efficient functionality testing of the MathChamp Interpreter.

- **Pytest Version:** 7.2.0
- **Platform:** Linux-5.15.0-52-generic-x86_64-with-glibc2.35
- **Python version:** 3.10.6
- **Packages:** "pluggy": "1.0.0", "pytest": "7.2.0"
- **Plugins:** "html": "3.2.0", "metadata": "2.0.4"

Please refer to figures 5.1, 5.2, and 5.3 on the pages which follow.

Environment

Packages	{"pluggy": "1.0.0", "pytest": "7.2.0"}
Platform	Linux-5.15.0-52-generic-x86_64-with-glibc2.35
Plugins	{"html": "3.2.0", "metadata": "2.0.4"}
Python	3.10.6

Summary

44 tests ran in 0.16 seconds.

(Un)check the boxes to filter the results.

☑ 44 passed, ☑ 0 skipped, ☑ 0 failed, ☑ 0 errors, ☑ 0 expected failures, ☑ 0 unexpected passes

Results

Show all details / Hide all details

Result	Test	Duration	Links
Passed	test_lexicalAnalyzer.py::test_get_tokens[1+2-3]	0.00	
No log output captured.			
Passed	test_lexicalAnalyzer.py::test_get_tokens[5*10-50]	0.00	
No log output captured.			
Passed	test_lexicalAnalyzer.py::test_get_tokens[1 - 5 - 10--14]	0.00	
No log output captured.			
Passed	test_lexicalAnalyzer.py::test_get_tokens[-0.02--0.02]	0.00	
No log output captured.			
Passed	test_lexicalAnalyzer.py::test_get_tokens[12+(5*3.3)-28.5]	0.00	
No log output captured.			
Passed	test_lexicalAnalyzer.py::test_get_tokens[500 + 2 * 5 / 4 - 8-494.5]	0.00	
No log output captured.			
Passed	test_lexicalAnalyzer.py::test_get_tokens[950 / (5*2) - 10-85]	0.00	
No log output captured.			
Passed	test_lexicalAnalyzer.py::test_get_tokens[10 + (5+2) * (5*3) - (9-2) / (1/2)-101]	0.00	
No log output captured.			
Passed	test_lexicalAnalyzer.py::test_get_tokens[(90 + 10) + (10 * 2) * (6 - 5) - (4 / 2)-118]	0.00	

Figure 5.1: Test Page 1

Passed test_lexicalAnalyzer.py::test_get_tokens[(5 / 2) * 10-25] 0.00
No log output captured.
Passed test_lexicalAnalyzer.py::test_get_tokens[2 + -1-1] 0.00
No log output captured.
Passed test_lexicalAnalyzer.py::test_get_tokens[20 * -2--40] 0.00
No log output captured.
Passed test_lexicalAnalyzer.py::test_get_tokens[40 + 6 / (4 - 5) - -10-44] 0.00
No log output captured.
Passed test_lexicalAnalyzer.py::test_get_tokens[2 ^ 4-16] 0.00
No log output captured.
Passed test_lexicalAnalyzer.py::test_get_tokens[2 + 3 ^ 4 -83] 0.00
No log output captured.
Passed test_lexicalAnalyzer.py::test_get_tokens[3 - 4 + 9^2 * (10 * 2)-1619] 0.00
No log output captured.
Passed test_lexicalAnalyzer.py::test_get_tokens[14 * 2 + 9 - 4 / 1 ^3-33] 0.00
No log output captured.
Passed test_lexicalAnalyzer.py::test_get_tokens[x=5-5] 0.00
No log output captured.
Passed test_lexicalAnalyzer.py::test_get_tokens[x^2-1-24] 0.00
No log output captured.
Passed test_lexicalAnalyzer.py::test_get_tokens[x = 2-2] 0.00
No log output captured.
Passed test_lexicalAnalyzer.py::test_get_tokens[x + 4-6] 0.00
No log output captured.
Passed test_lexicalAnalyzer.py::test_get_tokens[x = 4 *2-8] 0.00
No log output captured.
Passed test_lexicalAnalyzer.py::test_get_tokens[x = 140-140] 0.00
No log output captured.
Passed test_lexicalAnalyzer.py::test_get_tokens[x - 100-40] 0.00
No log output captured.
Passed test_lexicalAnalyzer.py::test_get_tokens[x>10-True] 0.00

Figure 5.2: Test Page 2

Passed test_lexicalAnalyzer.py::test_get_tokens[x % 16-12] 0.00
No log output captured.
Passed test_lexicalAnalyzer.py::test_get_tokens[500 % 42 + 5-43] 0.00
No log output captured.
Passed test_lexicalAnalyzer.py::test_get_tokens[50.12 * -8--400.96] 0.00
No log output captured.
Passed test_lexicalAnalyzer.py::test_get_tokens[-50 - -60-10] 0.00
No log output captured.
Passed test_lexicalAnalyzer.py::test_get_tokens[(-6 + 2 /4) * (-3^2)-49.5] 0.00
No log output captured.
Passed test_lexicalAnalyzer.py::test_get_tokens[2 > 4 -False] 0.00
No log output captured.
Passed test_lexicalAnalyzer.py::test_get_tokens[4 + 2 < 10-True] 0.00
No log output captured.
Passed test_lexicalAnalyzer.py::test_get_tokens[400 % 3 > 1000-False] 0.00
No log output captured.
Passed test_lexicalAnalyzer.py::test_get_tokens[5 ^ 2 > 1-True] 0.00
No log output captured.
Passed test_lexicalAnalyzer.py::test_get_tokens[10+6/2^1-8/4*2 > 2^1+6/2 -1*1-True] 0.00
No log output captured.
Passed test_lexicalAnalyzer.py::test_get_tokens[C=15>5-True] 0.00
No log output captured.
Passed test_lexicalAnalyzer.py::test_get_tokens[x=20-20] 0.00
No log output captured.
Passed test_lexicalAnalyzer.py::test_get_tokens[y=50-50] 0.00
No log output captured.
Passed test_lexicalAnalyzer.py::test_get_tokens[x+y-70] 0.00
No log output captured.
Passed test_lexicalAnalyzer.py::test_get_tokens[sin(90)-1] 0.00
No log output captured.
Passed test_lexicalAnalyzer.py::test_get_tokens[cos(180)--1] 0.00
No log output captured.
Passed test_lexicalAnalyzer.py::test_get_tokens[tan(0)-0] 0.00
No log output captured.
Passed test_lexicalAnalyzer.py::test_get_tokens[fact(5)-120] 0.00
No log output captured.
Passed test_lexicalAnalyzer.py::test_get_tokens[root(25)-5] 0.00
No log output captured.

Figure 5.3: Test Page 3

Chapter 6

Discussion, conclusion and future work

The final MathChamp product meets all of the 'Must Have' and 'Should Have' MoSCoW requirements. It is well-tested, well-documented software with clean code, an intuitive GUI, variable assignment and visualisation, accurate BODMAS order of operations, and visualisation of Mathematical Functions. All basic arithmetic expressions, professional style plotting functions, and configuration settings are supported. Furthermore, additional functionality has been achieved, such as boolean expressions, zero crossings, an excellent user interface, and broad functionality.

Due to time constraints, a polar curve spirograph was not implemented into the plotting. Complex numbers and differentiation are also unsupported as they did not make it to the final deliverable. The team decided it was better to focus on refining the existing functionality rather than rush extra functionality and risk a reduction in the robustness of the MathChamp product. However there was elements that weren't in our original MoSCoW such as variable dependency, in addition to meeting two of our 'could' requirements being zero crossings and factorials (as a function). The MathChamp interpreter could be improved by incorporating function differentiation and complex numbers. The interpreter could be made more efficient by improving all code efficiency (such as reducing the number of loops).

To conclude, despite a group restructuring one month into the project causing a considerable delay and some internal group difficulties, the development of our mathematical interpreter, MathChamp, has been a resounding success.

Bibliography

[1] QtOfficial. Using pyinstaller. https://doc.qt.io/qtforpython.

[2] PyQtGraphOfficial. Pyqt documentation. https://pyqtgraph.readthedocs.io/latest.html.

[3] Rudy J. Lapeer. *Lecture 2: Programming Languages*. Rudy J. Lapeer, 2022.

[4] PyInstaller. Qt for python documentation. https://pyinstaller.org/en/stable/usage.html.

[5] Ruslan Spivak. Ruslan's blog. https://ruslanspivak.com.

[6] Oxford. Shunting yard. http://math.oxford.emory.edu/site/cs171/shuntingYardAlgorithm.

[7] Datagy. How to make a list of the alphabet in python. https://datagy.io/python-list-alphabet/: :text=The%20easiest%20way%20to%20load,ascii$_u$*ppercase*%20*instances*.

[8] Umang Shrestha. Let's build an interpreter. https://python.plainenglish.io/writing-an-interpreter-in-python-from-scratch-part-1-af7698cff0d9.

[9] Evan Fosmark. Lexing with python. http://www.evanfosmark.com/2009/02/sexy-lexing-with-python.

[10] John Nyingi. Making a math interpreter. https://dev.to/j0nimost/implementing-a-math-interpreter-using-c-part1-2mf.

[11] Aqua Architect. Shunting yard algorithm. https://aquarchitect.github.io/swift-algorithm-club/Shunting%20Yard/.

Contributions

Descriptions and percentage of contribution by team-member:

- Ebin Paul: %
- Christopher Gavey: %
- Max James: %
- Aswin Sasi: %
- Soniya Avarachan: %

Appendix A

6.1 Design

6.1.1 Lo-Fi Design

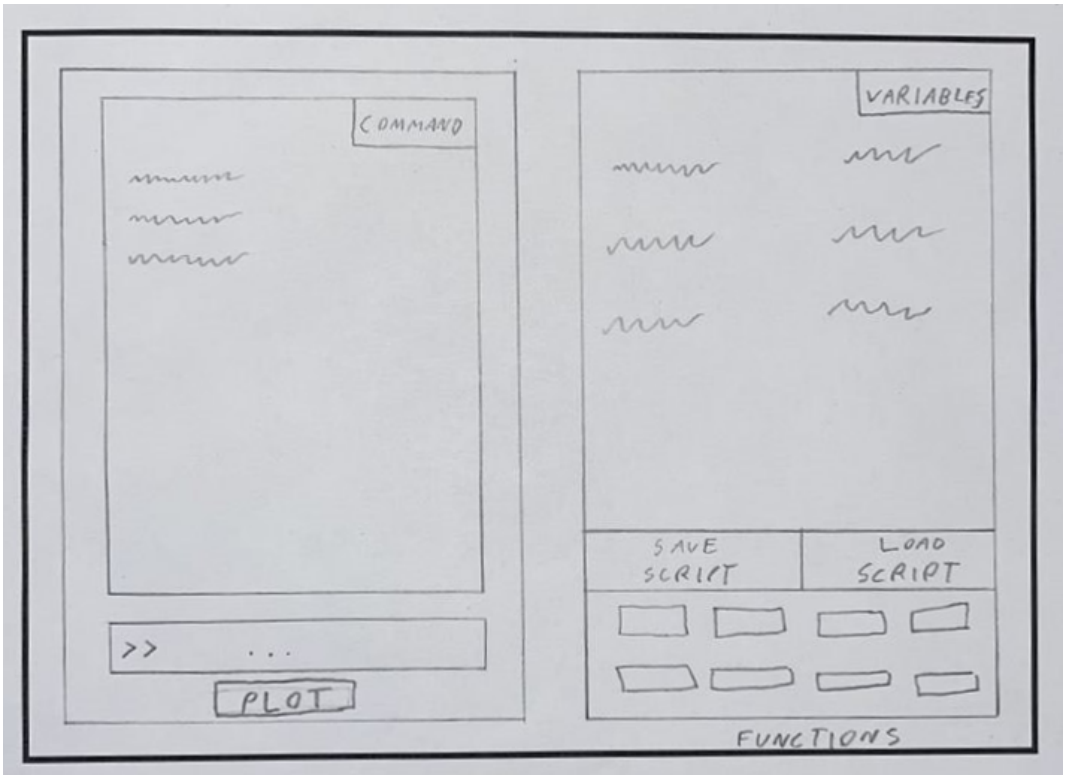

Figure 6.1: Main MathChamp window

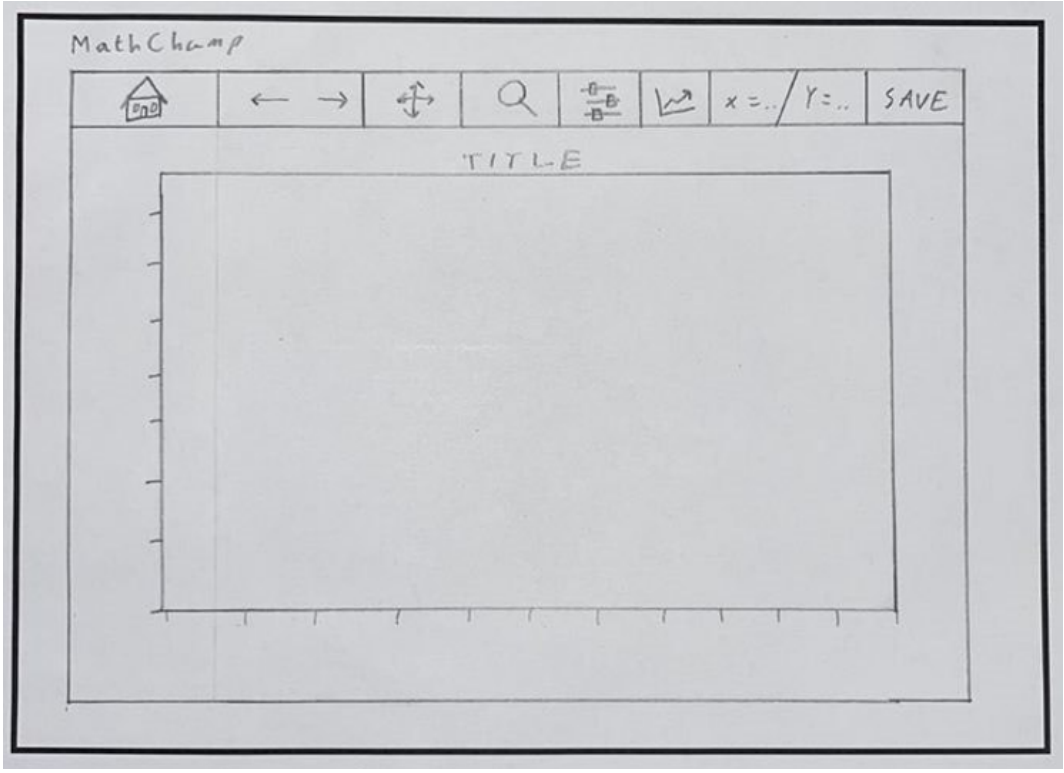

Figure 6.2: MathChamp plotting window

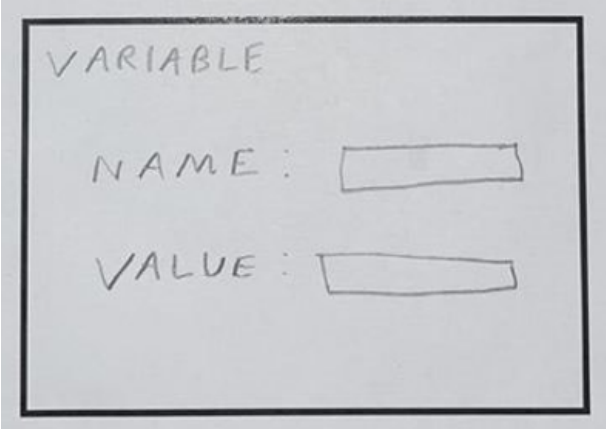

Figure 6.3: MathChamp variable assignment window

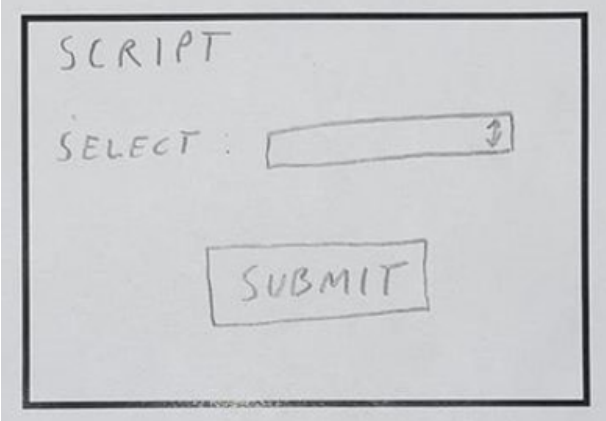

Figure 6.4: MathChamp scripts window

6.1.2 Medium-Fi Design

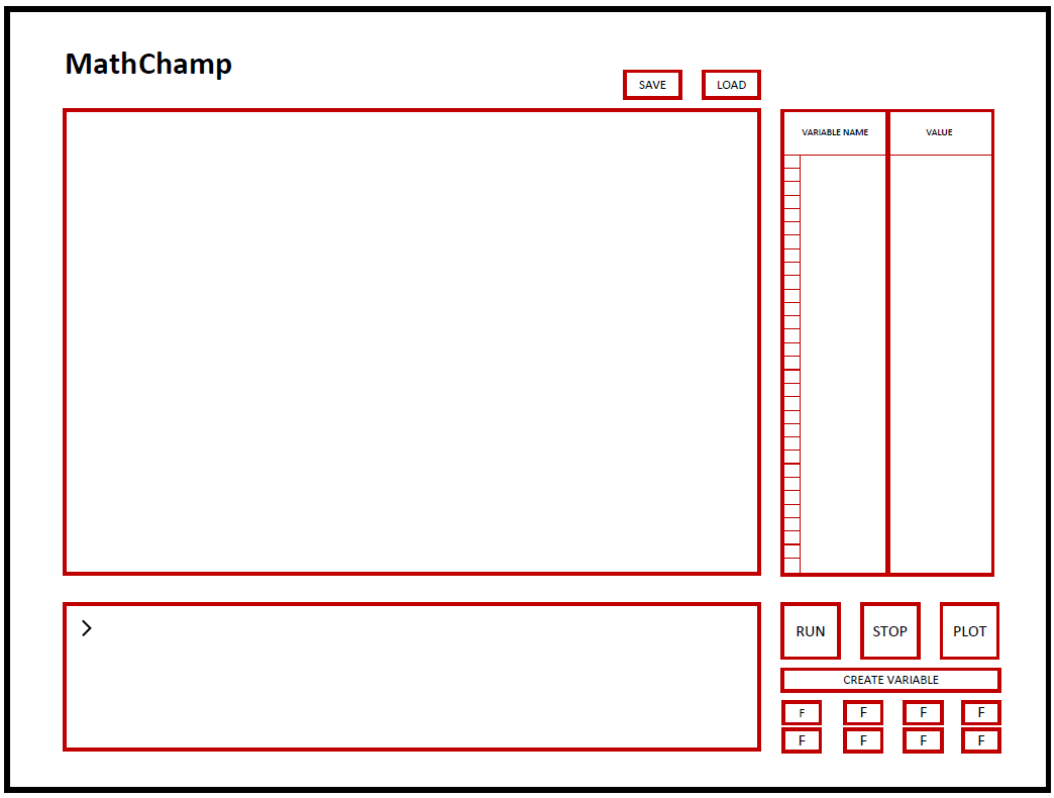

Figure 6.5: MathChamp main window

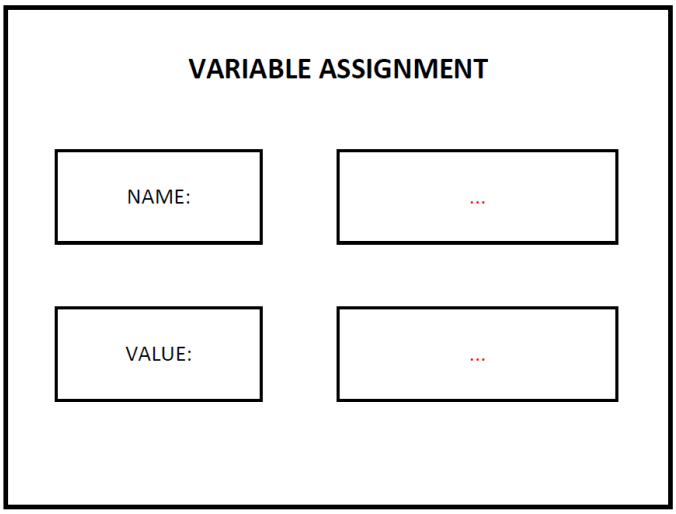

Figure 6.6: MathChamp variable assignment window

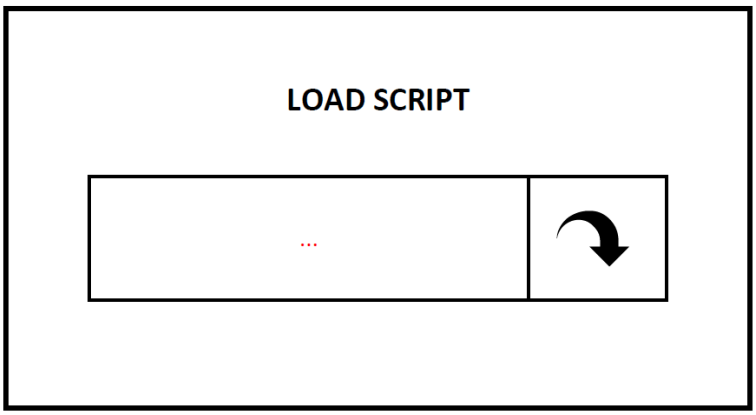

Figure 6.7: MathChamp scripts window

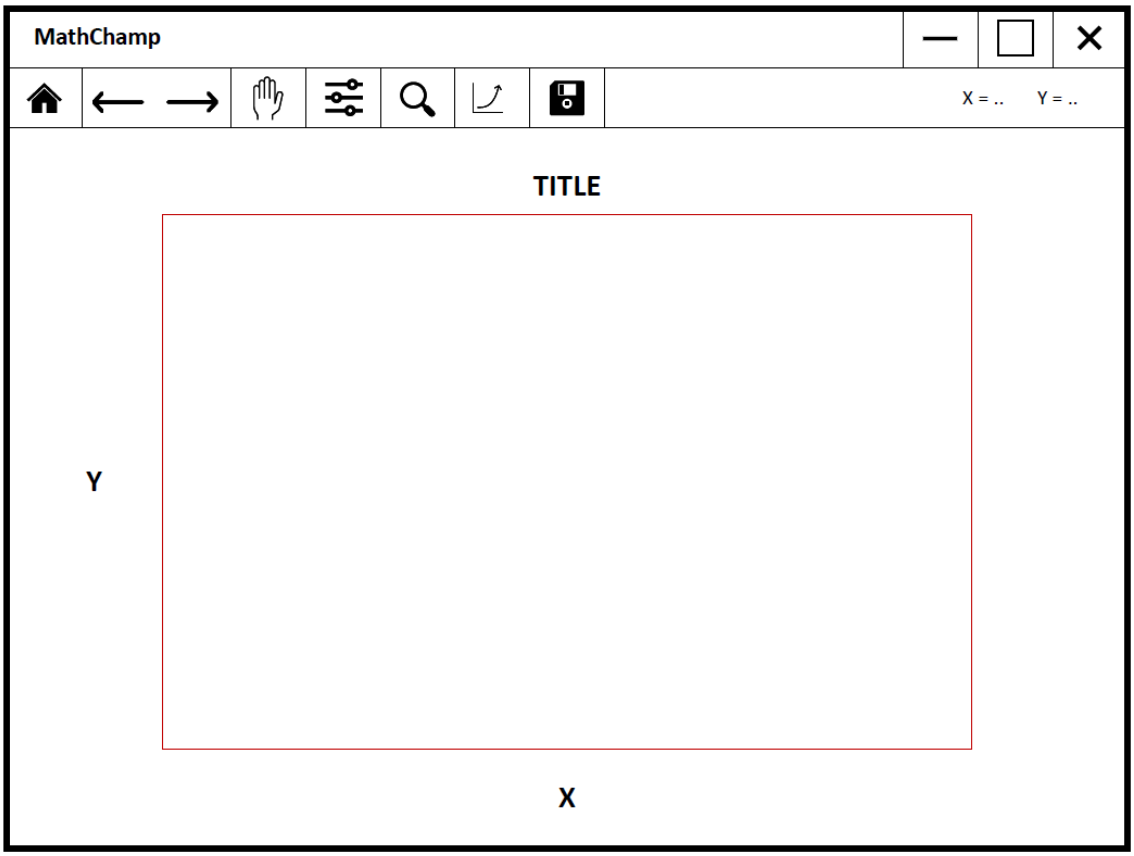

Figure 6.8: MathChamp plotting window

6.2 UML Diagrams

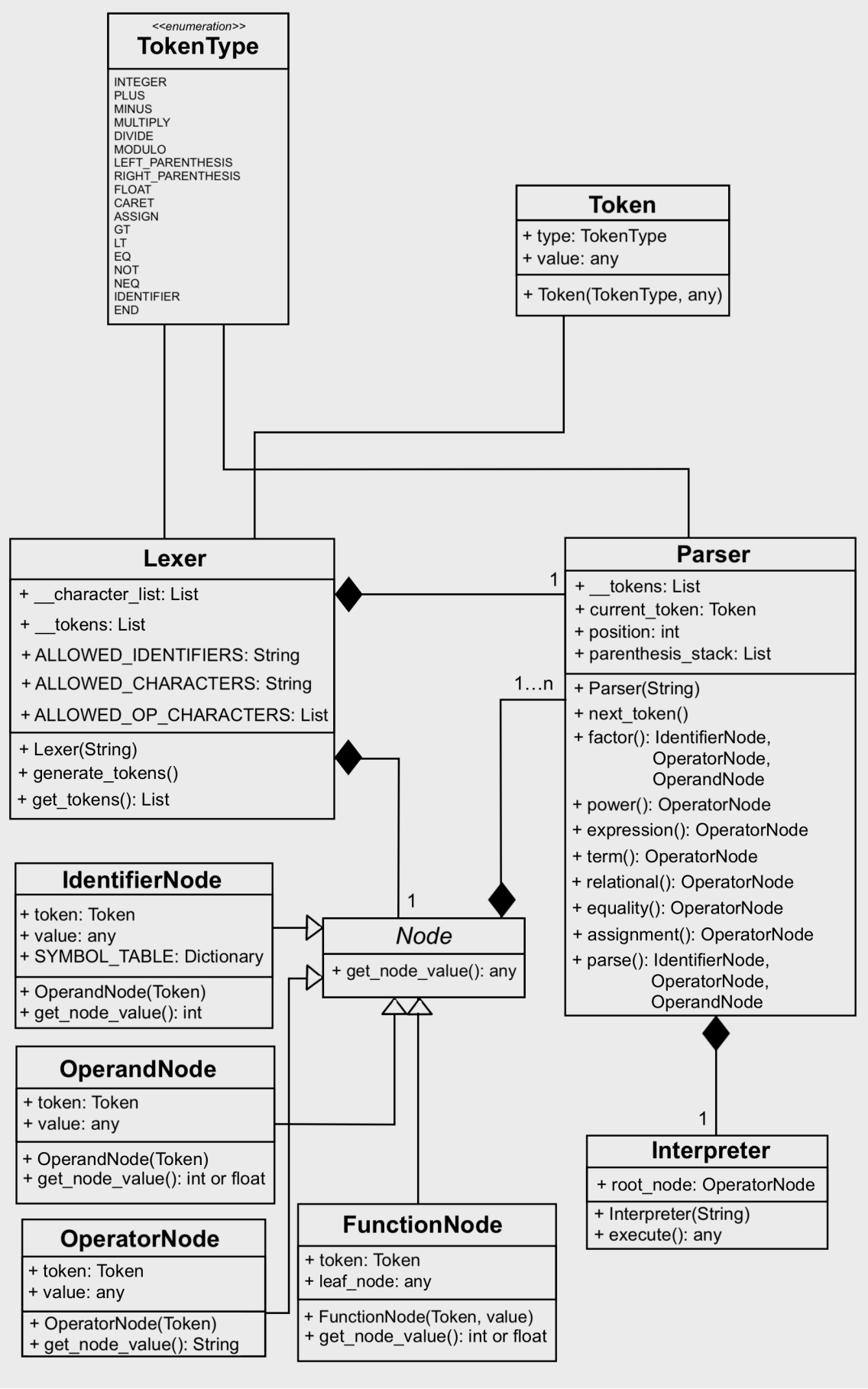

Figure 6.9: UML Class diagram for the Interpreter

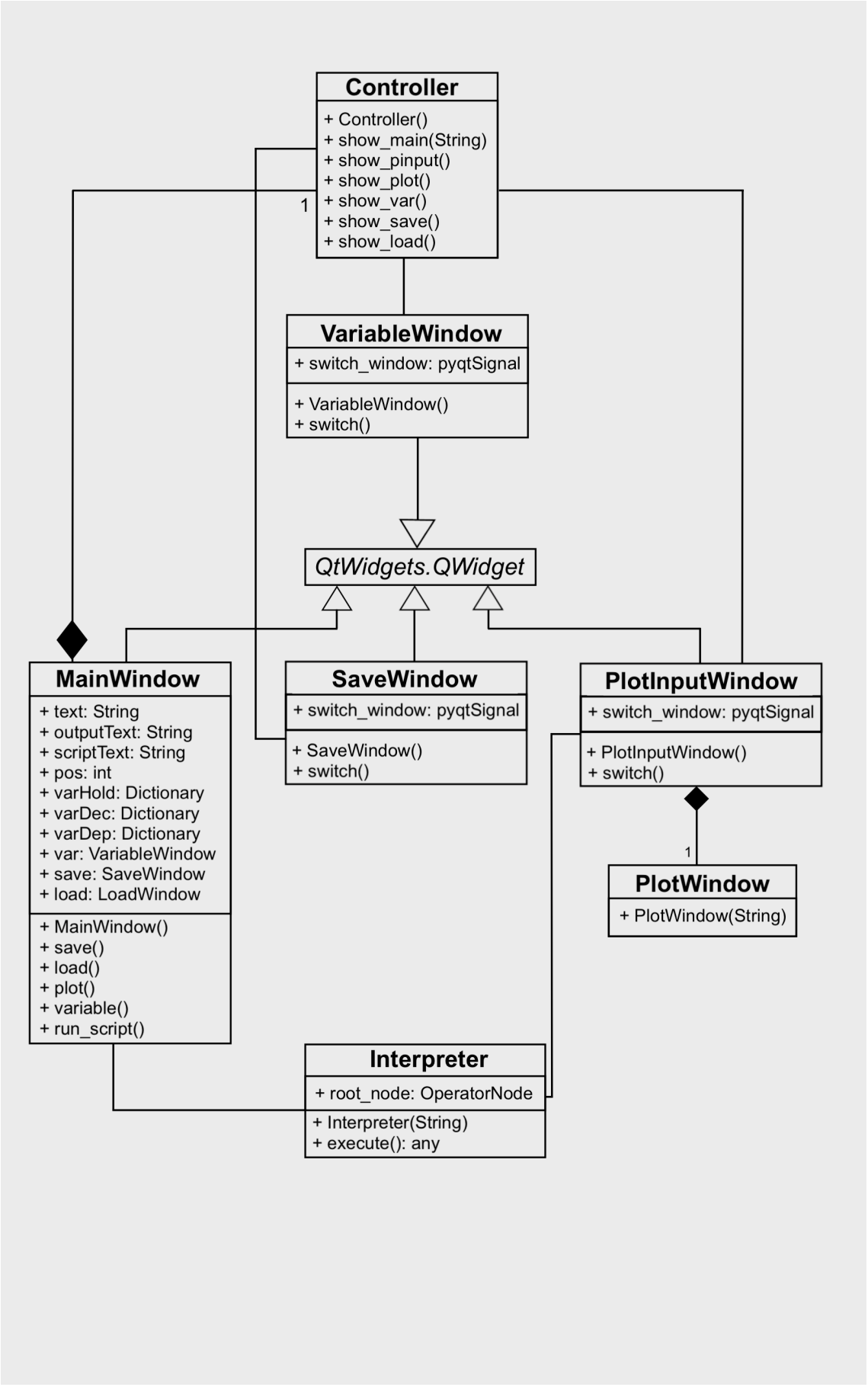

Figure 6.10: UML Class diagram for the GUI

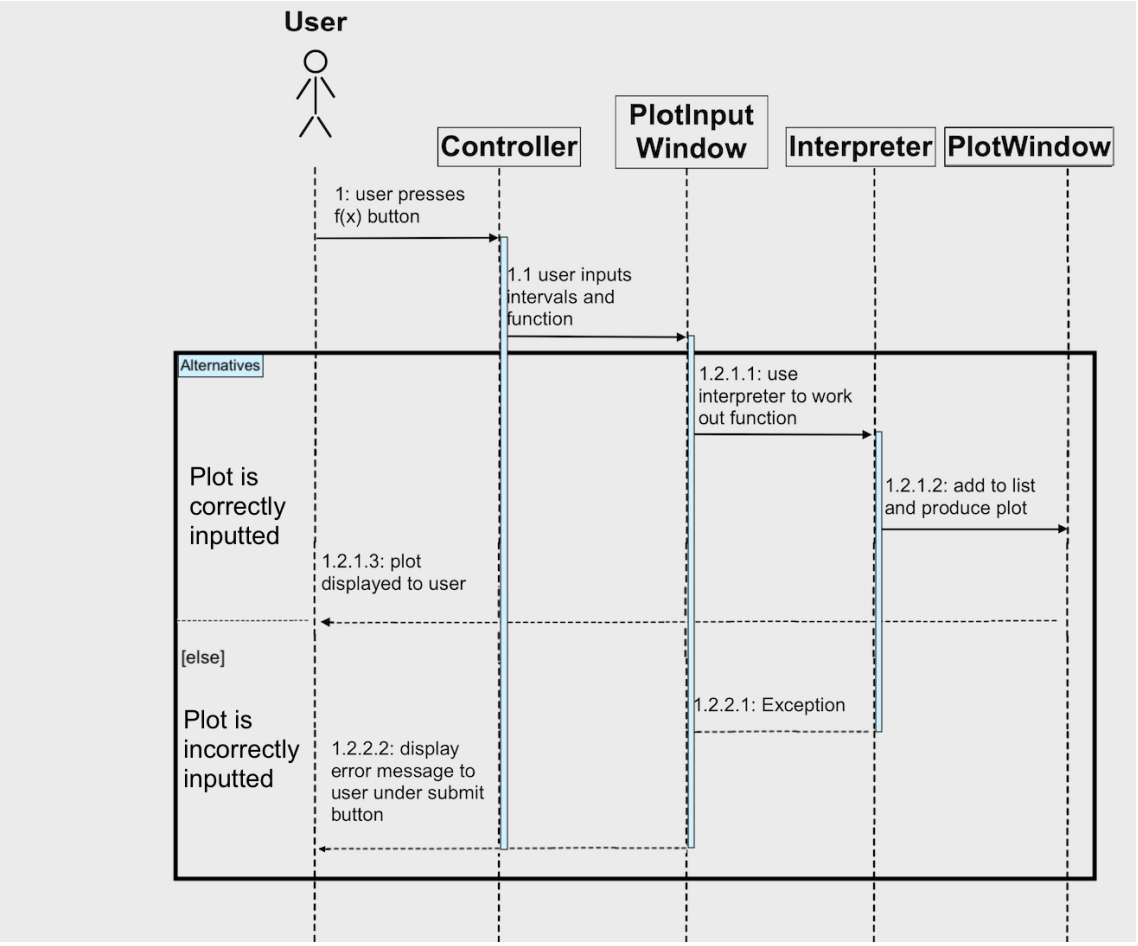

Figure 6.11: UML Class diagram for the GUI

6.3 Gantt Chart

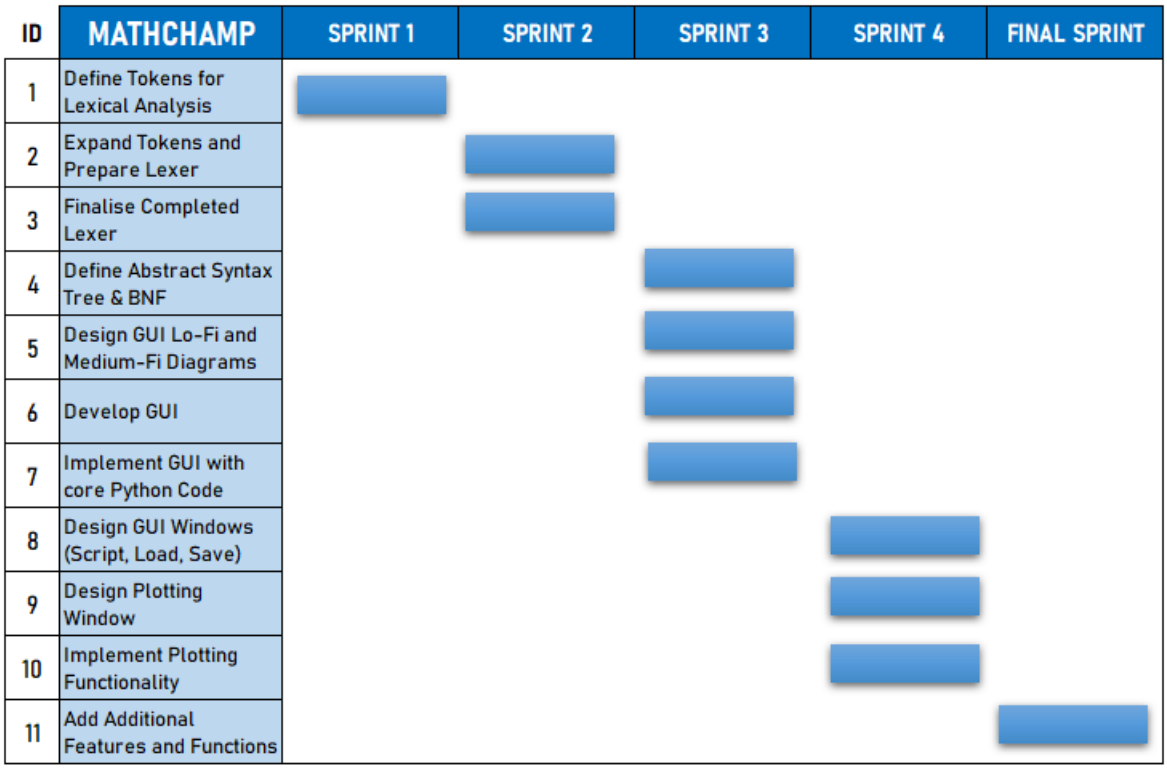

Figure 6.12: MathChamp Development Gantt

6.4 Manual Test Plans

T	OBJECTIVE	INPUT	EXEPECTED OUTPUT	OUTPU
tc1	Tokens allocated	1 + 2	INTEGER(1,2) and PLUS+	PASS
tc2	Tokens allocated	1 - 3	INTEGER(1,3) and MINUS-	PASS
tc3	Tokens allocated	3 * 4	INTEGER(3,4) and MULTIPLY*	PASS
tc4	Tokens allocated	5 * 6	INTEGER(5,6) and MULTIPILY*	PASS
tc5	Tokens allocated	7 - 8	INTEGER(7,8) and MINUS*	PASS
tc6	Tokens allocated	9 + 10	INTEGER(9,10) and PLUS+	PASS
tc7	Tokens allocated	12345 + 678910	INTEGER(1,2,3,4,5,6,7,8,9,10) and PLUS+	PASS
tc8	Tokens allocated	10 + (2*3)	INTEGER(10,2,3) and ( ) and *	PASS
tc9	Tokens allocated	12 + (5*3.2)	INTEGER(12,5,3,2) and ( ) and * FLOAT.	PASS
tc10	Tokens allocated	-1 + -2	INTEGER(-1,-2) and PLUS+	PASS
tc11	Tokens allocated	-3 + -4	INTEGER(-3,-4) and PLUS+	PASS
tc12	Tokens allocated	-5 + -6	INTEGER(-5,-6) and PLUS+	PASS
tc13	Tokens allocated	-7 * -8	INTEGER(-7,-8) and MULTIPLY*	PASS
tc14	Tokens allocated	-9 - -10	INTEGER(-9,-10) and MINUS-	PASS
tc15	Tokens allocated	12 / -2	INTEGER(12,-2) and DIVIDE/	PASS
tc16	Tokens allocated	14 - (-2 * 3)	INTEGER(14,-2,3) and ( ) and MINUS- & *	PASS
tc17	Successful parse	1 + 2 + 3 + 4 + 5	15	PASS
tc18	Successful parse	1 * 10 * 100	1000	PASS
tc19	Successful parse	1 - 5 - 10	-14	PASS
tc20	Successful parse	-0.02	0.02	PASS
tc21	Successful parse	500 + 2 * 5 / 4 - 8	494.5	PASS
tc22	Successful parse	950 / (5*2) - 10	85	PASS
tc23	Successful parse	10 + (5+2) * (5*3) - (9-2) / (1/2)	101	PASS
tc24	Successful parse	(90 + 10) + (10 * 2) * (6 - 5) - (4 / 2)	118	FAIL
tc25	Successful parse	10 * (5 / 2)	25	PASS

Figure 6.13: Manual Tests (1 - 25)

tc26	Successful parse	(5 / 2) * 10	25	**FAIL**
tc27	Successful parse	2 + -1	1	**PASS**
tc28	Successful parse	20 * -2	-40	**PASS**
tc29	Successful parse	40 / -10	-4	**PASS**
tc30	Successful parse	40 + 6 / (4 - 5) - -10	44	**PASS**
tc31	Successful parse	2 ^ 4	16	**PASS**
tc32	Successful parse	2 + 3 ^ 4	83	**PASS**
tc33	Successful parse	2 * 1 + 3 ^ 3	29	**PASS**
tc34	Successful parse	6 / 2 + 4 ^ 6	4099	**PASS**
tc35	Successful parse	8 ^ 2 - 4	60	**PASS**
tc36	Successful parse	3 - 4 + 9^2 * (10 * 2)	1619	**PASS**
tc37	Successful parse	14 * 2 + 9 - 4 / 1 ^3	33	**FAIL**
tc38	Successful parse	14 * 2 + 9 - 4 /(1 ^3)	33	**PASS**
tc39	Variable assignment	x = 2	2	**PASS**
tc40	Variable assignment	x = 2, x + 4	6	**PASS**
tc41	Variable assignment	x = 4 *2	8	**PASS**
tc42	Variable assignment	x = 10, x / 2	5	**PASS**
tc43	Variable assignment	x = 140, x - 100	40	**PASS**
tc44	Variable assignment	x = 1000, x % 16	8	**PASS**
tc45	Variable assignment	x = 10, y = 30, x + y	40	**PASS**
tc46	Variable assignment	x = 100, y = 2, x / y	50	**PASS**
tc47	Variable assignment	x = 1000, y = 6, x % y	4	**PASS**
tc48	Variable assignment	x = 200, y = 400, y - x	200	**PASS**
tc49	Variable assignment	x = 500, x / (1 + 2)	166.66	**PASS**
tc50	Successful parse	100 % 7.5	2.5	**PASS**

Figure 6.14: Manual Tests (26 - 50)

tc51	Successful parse %	500 % 42 + 5	43	**PASS**
tc52	Successful parse %	4000 % 67 * 2	94	**PASS**
tc53	Successful parse %	59 % 5 - (3 + 4)	-3	**PASS**
tc54	Successful parse %	6004.25 % 46 / (16 - 2)	1.73	**PASS**
tc55	Successful parse .	1.2 + 5.6	6.8	**PASS**
tc56	Successful parse .	50.3 * 5.9	296.77	**PASS**
tc57	Successful parse .	604.35 / 8.2	73.7	**PASS**
tc58	Successful parse .	30.4 ^ 70.1	8.9	**PASS**
tc59	Successful parse .	400.53 - 43.2	357.33	**PASS**
tc60	Successful parse .	600.57 % 8.2	1.97	**PASS**
tc61	Successful parse .	(67.2 + 4) * (1.09/2) % 2 - 1^2	-0.195	**PASS**
tc62	Successful parse -	-104 + -106.2	-210.2	**PASS**
tc63	Successful parse -	600.5 / -50	-12.01	**PASS**
tc64	Successful parse -	50.12 * -8	-400.96	**PASS**
tc65	Successful parse -	-89 ^ -2.5	-1.33	**PASS**
tc66	Successful parse -	-50 - -60.2	10.2	**PASS**
tc67	Successful parse -	(-600 + 2 /4.1) * (-32.5^1)	19484	**PASS**
tc68	Successful T/F parse	2 > 4	FALSE	**PASS**
tc69	Successful T/F parse	4 > 2	TRUE	**PASS**
tc70	Successful T/F parse	4 + 2 > 10	FALSE	**PASS**
tc71	Successful T/F parse	4 + 2 < 10	TRUE	**PASS**
tc72	Successful T/F parse	10 * 2 > 8	TRUE	**PASS**
tc73	Successful T/F parse	10 * 2 < 8	FALSE	**PASS**
tc74	Successful T/F parse	35 - 6 > 100	FALSE	**PASS**
tc75	Successful T/F parse	35 - 6 < 100	TRUE	**PASS**

Figure 6.15: Manual Tests (51 - 75)

tc76	Successful T/F parse	400 % 3 > 1000	FALSE	**PASS**
tc77	Successful T/F parse	400 % 3 < 1000	TRUE	**PASS**
tc78	Successful T/F parse	350 / 4 > 4000	FALSE	**PASS**
tc79	Successful T/F parse	350 / 4 < 4000	TRUE	**PASS**
tc80	Successful T/F parse	5 ^ 2 > 1	TRUE	**PASS**
tc81	Successful T/F parse	5 ^ 2 < 1	FALSE	**PASS**
tc82	Successful T/F parse	10+6/2^1-8/4*2 > 2^1+6/2 -1*1	TRUE	**PASS**
tc83	Successful T/F parse	10+6/2^1-8/4*2 < 2^1+6/2 -1*1	FALSE	**PASS**
tc84	Successful T/F parse	(15+2)/4^1-5*6 > (3^2)+3/1-5*4	FALSE	**PASS**
tc85	Successful T/F parse	(15+2)/4^1-5*6 < (3^2)+3/1-5*4	TRUE	**PASS**
tc86	Successful T/F parse	15.7+(3.54 % 2) - 4^2 > 10000.451	FALSE	**PASS**
tc87	Successful T/F parse	15.7+(3.54 % 2) - 4^2 < 10000.451	TRUE	**PASS**
tc88	GUI: Enter expressions	Input Field: 10 - 5	Output in GUI Result Field: 5	**PASS**
tc89	GUI: Enter expressions	Input Field: 100 * 6	Output in GUI Result Field: 600	**PASS**
tc90	GUI: Enter expressions	Input Field: 1000 / 2	Output in GUI Result Field: 500	**PASS**
tc91	GUI: Enter expressions	Input Field: 4 % 2	Output in GUI Result Field: 0	**PASS**
tc92	GUI: Enter expressions	Input Field: 4 ^ 6	Output in GUI Result Field: 4096	**PASS**
tc93	GUI: Enter expressions	Input Field: 10 + (4 * 6 - 2 / 3)	Output in GUI Result Field: 33	**PASS**
tc94	GUI: Use button	Click 'Save' script button	Save script window opens	**PASS**
tc95	GUI: Use button	> Click 'Save Script' in window	Script saves successfully	**PASS**
tc96	GUI: Use button	Click 'Load' script button	Load script window opens	**PASS**
tc97	GUI: Use button	> Select a script and select 'Load'	Scripts loads into GUI successfully	**PASS**
tc98	GUI: Use button	> Type into script field, press run	Manual script runs successfully	**PASS**
tc99	GUI: Use button	> Click 'Stop'	Running script stops successfully	**PASS**
tc100	GUI: Use button	Click 'Create Variable'	Create Variable' window opens	**PASS**

Figure 6.16: Manual Tests (76 - 100)

tc101	GUI: Use button	> Input variable name and value	Variable saves successfully	**PASS**
tc102	GUI: Use button	> Save a variable	Saved variable shown in Variables Table	**PASS**
tc103	GUI: Use button	Click 'Plot'	Plot input window is shown	**PASS**
tc104	GUI: Use button	> Input plot values (if required)	Plotting is shown in new window	**PASS**
tc105	GUI: Use button	Pre-enter expression, click 'Plot'	Plotting is shown in new window	**PASS**
tc106	GUI: Comaptibility	Resize MathChamp window	Window is responive, auto-scaling	**PASS**
tc107	GUI: Comaptibility	**Run MathChamp on macOS**	App performs all above processes	**PASS**
tc108	GUI: Comaptibility	**Run MathChamp on Window 11**	App performs all above processes	**PASS**
tc109	GUI: Comaptibility	**Run MathChamp on Windows 10**	App performs all above processes	**PASS**
tc110	GUI: Comaptibility	**Run MathChamp on Windows 8.1**	App performs all above processes	**PASS**
tc111	Variable Dependecy	x = 5 (return), x = 10	5 updated to 10, shown in variables	**PASS**
tc112	Variable Dependecy	x = 8 + 4 (return), x = 1	12 updated to 1, shown in variables	**PASS**
tc113	Variable Dependecy	q = 12 - 2 (return), q = 100	10 updated to 100, shown in variables	**PASS**
tc114	Variable Dependecy	U = 2*2 (return), U = 40	4 updated to 40, shown in variables	**PASS**
tc115	Variable Dependecy	h = 5^5 (return), h = 9	3125 updated to 9, shown in variables	**PASS**
tc116	Variable Dependecy	c = 6%3 (return), c = -10	0 updated to -10, shown in variables	**PASS**
tc117	Variable Dependecy	C = 15 > 5, (return), C = 15 < 5	True updated to false, shown in variables	**PASS**
tc118	Variable Dependecy	X = 15 < 5, (return), X > 15 > 5	False updated to true, shown in variable	**PASS**
tc119	Variable Dependecy	B = (15+2), (return), B = (16/3)	17 updated to 5.33, shown in variables	**PASS**
tc120	Input Validation	Main field: "string input"	"ERROR: String Input Not Defined"	**PASS**
tc121	Input Validation	Main field: 3 + &	"ERROR: Invalid character(s)"	**PASS**
tc122	Input Validation	Variable Name: 12	"Variable Not Suitable"	**PASS**
tc123	Input Validation	Variable Value: String	"Variable Not Suitable"	**PASS**
tc124	Input Validation	Script Name: *&^	"Script Name Not Suitable"	**PASS**
tc125	Input Validation	Plot Value: &^4	"Plot Not Suitable"	**PASS**

Figure 6.17: Manual Tests (101 - 125)