**AS 91637 Develop a complex computer program for a specified task**

*To get to excellence you must efficiently develop a complex computer program for a specified task*:

* ensuring that the overall modular and procedural design, graphical user interface, and event handling design, are a well-structured, logical decomposition of the task, and that the program is flexible and robust
* setting out the program code concisely and documenting the program with comments that explain and justify decisions
* Comprehensively testing and debugging the program in an organised and time effective way to ensure the program is correct on expected, boundary and invalid input cases.

*This is a really interesting process. To get to excellence you really must manage your project from the beginning with a pre-planned programme. You must plan to test before your start so your initial design process will be very full and thorough. If you are aiming for excellence you really will only have two steps. The initial design process planning for everything, amid project review and very quickly and efficiently reach your final project overall test. Evidence is crucial, you cannot leave gaps.*

The minute I see you muddle through or trial and error you lock into Achieved. You can do all your muddling out of my sight but you must come to class and work efficiently, methodically and effectively to get excellence.

THIS IS THE STANDARD THE UNIVERSITIES EXPECT YOU TO COME IN AT THEY ARE THE GOAL POST!

# Initial Design Process

## Planned development:

I will be developing a program that will provide a simple mathematical game to audiences around ages 2 to 6 years old specifically. The program I will be using is Python 3.3 where the aim to make a graphical interface (GUI) based around my mathematical idea is important.

Graphical elements that I plan to use will include Labels, Buttons and Entry widgets, with the use of shapes, such as creating rectangles to represent the user input for the maths game. While I will plan to implement event based inputs from the user, including clicking events (canvas.bind “button-1” for subfunction), input text for users name and age, and creating fixed generated shapes for the users to click on using clicking events.

## Initial Variable Table

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Variable Name | Variable Val. | Defined | Created By | Comments (describe code function and behaviour M) (why and what for E) |
| n\_number | Integer | Global | Fixed | Global Value for user input by pressing number. This is so I can make a value that is fixed but is determined by the user in order to enter an answer. |
| n\_answer | Integer | Global | Calculated | Global Value for random generated answer in link with equation. This is so I can store randomized answer values that are determined by both variables n and m. From this the n\_answer can be defined for the user to get correct logically and mathematically. |
| n | Integer | Global | Randomly Generated | Number randomly generated for equation. I’m using variable n as a randomized number in my randomized equation, instead of fixed equations for diversity. |
| m | Integer | Global | Randomly Generated | Number randomly generated for equation. I’m using variable m as a randomized number in my randomized equation, instead of fixed equations for diversity. |
| name | String | Local | Input Box | Defines the user’s valid name. The variable also can identify the user if the login is successful. I specifically used name variable to provide a boundary for validity of the users input in the login menu. |
| age | Integer | Global | Input Box | Defines the user’s valid age. I used age to define both the users age and also to commit to my level system, where if age increases then level increases. |
| question\_list | List | Global | Fixed | Displays approachable question introduction. I left this as a global variable so it can firstly provide a cover for the welcome label when reset, but also to be a definite “what to do” for the user. It is also a list due to how I can easily change up the question\_list introduction. |
| pressed | Integer | Local | Calculated | If user presses certain number define pressed = x. I used this variable so if the user presses the selected “squares” then the variable pressed will equal that certain number. This also defines if the user presses the number, then the answer will be defined if it is correct. |

## Initial Testing Table

|  |  |
| --- | --- |
| Input: name | Expected Result |
| Name\_Input = “Hello” | Display\_message = “Welcome “Hello” to Math Game” |
| Name\_Input = “Carlos” | Display\_message = “Welcome “Carlos” to Math Game” |
| Name\_Input = “carlos” | Display\_message = “Welcome “carlos” to Math Game” |
| Name\_Input = “192873192837” | Display\_message = “Please input valid name” |
| Name\_Input = “#$\*&#Hi” | Display\_message = “Please input valid name” |

|  |  |
| --- | --- |
| Input: age | Expected Result |
| Age\_Input = “2” or “3” | Display\_message = “You will be doing level 1” |
| Age\_Input = “4” or “5” | Display\_message = “You will be doing level 2” |
| Age\_Input = “6” | Display\_message = “You will be doing level 3” |
| Age\_Input = “1” | Display\_message = “Please input valid age” |
| Age\_Input = “-5” | Display\_message = “Please input valid age” |
| Age\_Input = “100” | Display\_message = “Please input valid age” |
| Age\_Input = “carlos” | Display\_message = “Please input valid age” |
| Age\_Input = “#$\*&#3242” | Display\_message = “Please input valid age” |

## Initial Flow Chart

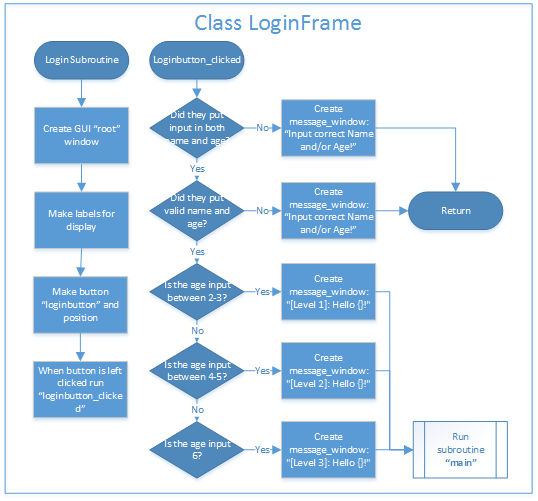


Initial Flow Chart draft includes base game and includes the generating of random numbers with GUI labels and certain times when they will be used. Main functions like Quit, Start, and Reset are within the flow chart.

Quit – Exit Game

Start – Conducts questions for user to do

Reset – Resets all number values and placement of labels such as the equation\_function1

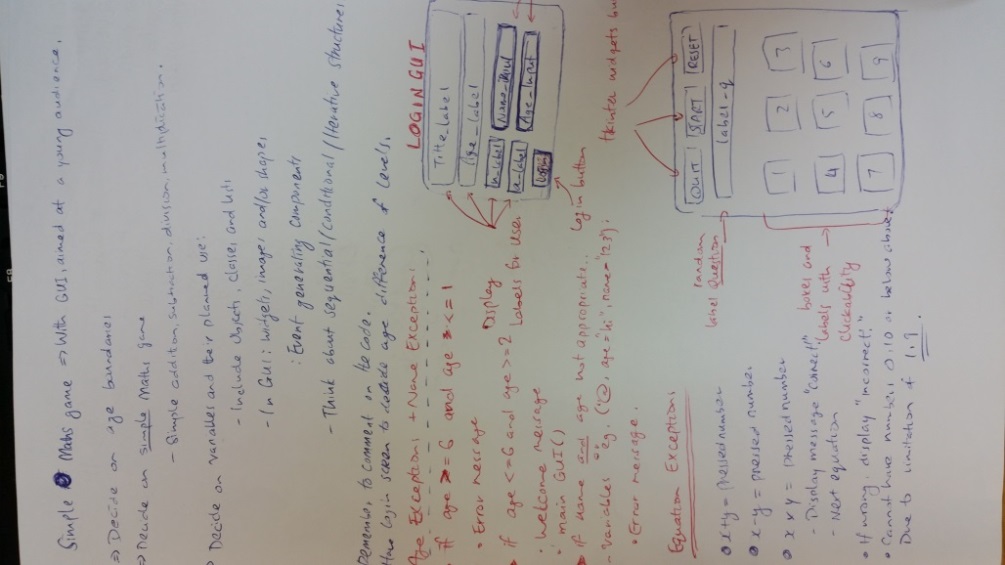


Initial Flow Chart draft only includes the base math game, where the start runs class LoginFrame and only includes the initial level of questions, where if age is in-between 2-3 equation\_function1 will run.

Loginbutton\_clicked – is the function when the user is within the login window and clicks the “start” button. From this there are exceptions to how the user entered properly and restrictively. So ages between 2-6 are allowed, and values of name only having string, while age can only have integers

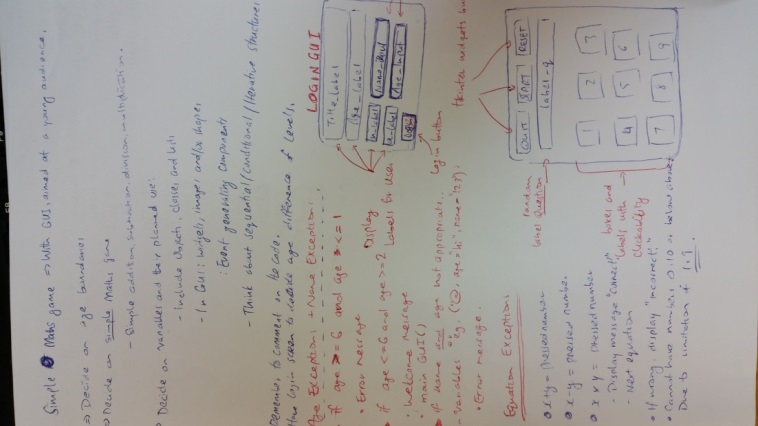
LoginSubroutine – is the function to create the initial login menu window and display its GUI labels, input boxes and buttons.

## Initial Graphical User Interface Design



Class Loginframe

Final planning product of my python program, which includes exception to input of age and so the equation difficulty, will vary in terms of age > 2. I decided on making a login GUI screen, which if user successfully inputs both age and name, they will be brought to the main GUI window. I also decided to make square shapes to represent the “buttons” of inputting an answer. Most importantly, the equations are so that the randomized x-value interacting with the random y-value must equal to the “pressed number” in order to progress.



Initial planning of design, ideas and what is needed within my computer program using Python. This includes the certain age boundaries, concept of game (simple mathematics application), deciding of certain classes and lists to use, use of GUI concepts (widgets, shapes and input text), and to apply easy access and readability to my program.

## Initial Functions/Subroutines

|  |  |  |
| --- | --- | --- |
| Function/Subroutine | Purpose and use | Comments |
| Subroutine - “start” | Purpose is to initially start the game when user does so (button widget) and thereby an equation is generated (specified by difficulty from user’s age input) and also disables start button widget to eliminate start subroutine spam. | “When start button widget is pressed/activated, run “start” subroutine”  “Generate labels and equations”  “If age is \*\*\* run certain equation function” |
| Subroutine – “reset” | Purpose is once user activates reset button widget, the whole game will “reset” in terms of label placement and activation of the start button widget | “When reset function is running make message labels visible and activate start button widget” |
| Subroutine – “quit” | Purpose is to provide a quit function within the game, instead of exiting through the window aspect | “When quit function is running, destroy window of game” |
| Subroutine – “main” | This is the main foundations to the game, where the window is specified and named and initial generating shapes and label are made from this function. Most importantly generates buttons | “When main function is running, generate rectangle shapes and labels and buttons”  “Make clicking interactive with mouse-1 for user to click on boxes” |

## Array List + Append List

I will make it so the List including in my code will only interfere with the generating of text after the user has clicked start button widget. This list would include various titles/starting positions that just notify the user to answer these specified questions.

Where: text\_list = [“Question 1:”, “Question:”, “Solve:”]

And include an append list to add onto those question examples:

text\_list.append(“Answer: “)

## Procedural structures:

|  |  |  |
| --- | --- | --- |
| Sequential | Conditional (A) | Iterative structure (A) |
|  | IF | Loop |
| I am planning to make my equation handled as a String where x = a + b, but also handle as integer using StringVar(). | If user input name or age doesn’t meet the requirements, then login screen will continue until everything is valid. Therefore display message\_window saying “Input correct Name and/or age” | If user inputs name and age correctly, then loop a “loading” type screen in the background for when the subroutine “main” is beginning. |

I’ve used some base code from my draft program and initial planning towards my final program. With exceptions of “If” statements from user entering age and name, and use of Loops, such as using “While” to loop a conceptual “loading” screen.

# Final Design Process to Production

**Final Product: Coding will be complete and commented | the GUI will be complete | Testing will be complete, errors and changes will be complete | the final version of this report will reflect your final game**

In the Final Design Process, I constructed my GUI as what was planned. So for instance, I planned for a login screen that would involve User Inputs, Labels and Widgets that are generated automatically from start-up of the Class LoginFrame.

The GUI in the “Main” subroutine, I would have added some small changes planned, such as if user gets the answer correct or incorrect, then message will display telling them.

Other than that, the main subroutine window is expected within my planning, where I included the buttons “Start”, “Quit” and “Reset” which all function properly and made concise to suit the GUI. Also include some labels and shapes, such as the rectangles generated as the “buttons” that the user can click on, while the labels are the titles and lists that show what the user has to do.

## Final Variable Table

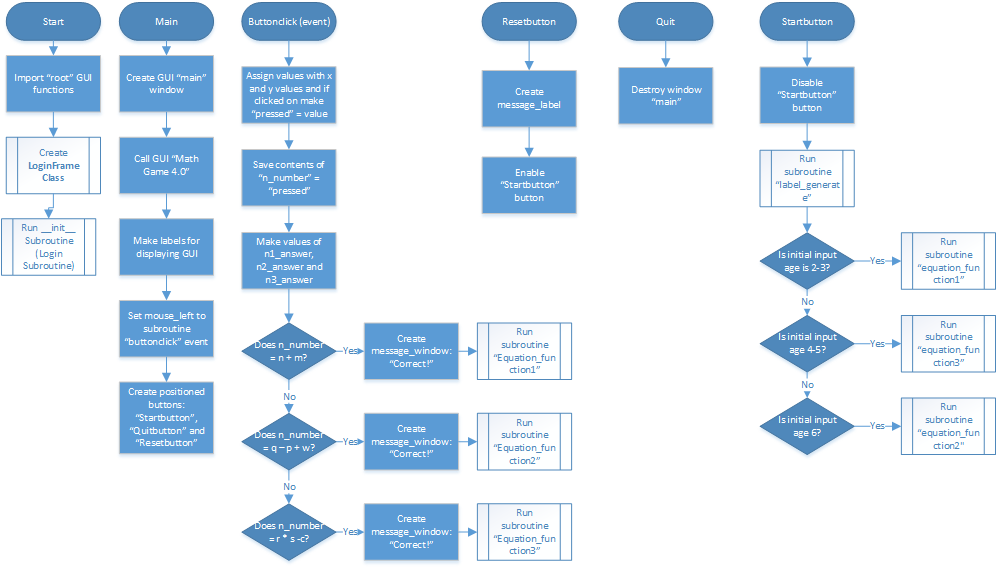
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Variable Name | Variable Val. | Defined | Created By | Comments (describe code function and behaviour M) (why and what for E) |
| n\_number = 0 | Integer | Global | Fixed | Global Value for user input by pressing number. This is so I can make a value that is fixed but is determined by the user in order to enter an answer. |
| n\_answer = 0 | Integer | Global | Fixed | Global Value for random generated answer in link with equation. This is so I can store randomized answer values that are determined by both variables n and m. From this the n\_answer can be defined for the user to get correct logically and mathematically. |
| n = 0 | Integer | Global | Fixed | n Number randomly generated for equation. I’m using variable n as a randomized number in my randomized equation, instead of fixed equations for randomization. This is for subfunction equation\_function1 |
| m = 0 | Integer | Global | Fixed | m Number randomly generated for equation. I’m using variable m as a randomized number in my randomized equation, instead of fixed equations for randomization. This is for subfunction equation\_function1 |
| p = 0 | Integer | Global | Fixed | p Number randomly generated for equation. I’m using variable p as a randomized number in my randomized equation, instead of fixed equations for randomization. This is for subfunction equation\_function2 |
| q = 0 | Integer | Global | Fixed | q Number randomly generated for equation. I’m using variable q as a randomized number in my randomized equation, instead of fixed equations for randomization. This is for subfunction equation\_function2 |
| r = 0 | Integer | Global | Fixed | r Number randomly generated for equation. I’m using variable r as a randomized number in my randomized equation, instead of fixed equations for randomization. This is for subfunction equation\_function2 |
| s = 0 | Integer | Global |  | s Number randomly generated for equation. I’m using variable s as a randomized number in my randomized equation, instead of fixed equations for randomization. This is for subfunction equation\_function3 |
| W = 0 | Integer | Global |  | w Number randomly generated for equation. I’m using variable w as a randomized number in my randomized equation, instead of fixed equations for randomization. This is for subfunction equation\_function3 |
| C = 0 | Integer | Global |  | c Number randomly generated for equation. I’m using variable c as a randomized number in my randomized equation, instead of fixed equations for randomization. This is for subfunction equation\_function3 |
| X | Integer | Global | Fixed | X is defined in subroutines “new\_start” and “start” where the positions of the generated rectangles are assigned a certain X position amount |
| Y | Integer | Global | Fixed | Y is defined in subroutines “new\_start” and “start” where the positions of the generated rectangles are assigned a certain Y position amount |
| Count = 0 | Integer | Local | Fixed | This is the feature to provide a “loading” screen, which helps the user indicate that the “main” subroutine is starting. This is used within the “while” code in : |

## Planned Input Testing

|  |  |
| --- | --- |
| Input: **name** | Expected Result |
| Name\_Input = “Hello” | Display\_message = “Welcome “Hello” to Math Game” |
| Name\_Input = “Carlos” | Display\_message = “Welcome “Carlos” to Math Game” |
| Name\_Input = “carlos” | Display\_message = “Welcome “carlos” to Math Game” |
| Name\_Input = “192873192837” | Display\_message = “Please input valid name” |
| Name\_Input = “#$\*&#Hi” | Display\_message = “Please input valid name” |

|  |  |
| --- | --- |
| Input: **age** | Expected Result |
| Age\_Input = “2” or “3” | Display\_message = “You will be doing level 1” |
| Age\_Input = “4” or “5” | Display\_message = “You will be doing level 2” |
| Age\_Input = “6” | Display\_message = “You will be doing level 3” |
| Age\_Input = “1” | Display\_message = “Please input valid age” |
| Age\_Input = “-5” | Display\_message = “Please input valid age” |
| Age\_Input = “100” | Display\_message = “Please input valid age” |
| Age\_Input = “carlos” | Display\_message = “Please input valid age” |
| Age\_Input = “#$\*&#3242” | Display\_message = “Please input valid age” |

## Final Flow Chart



The “main” subroutine, which is linked “reset”, “quit” and “start” as well as “buttonclick (event)



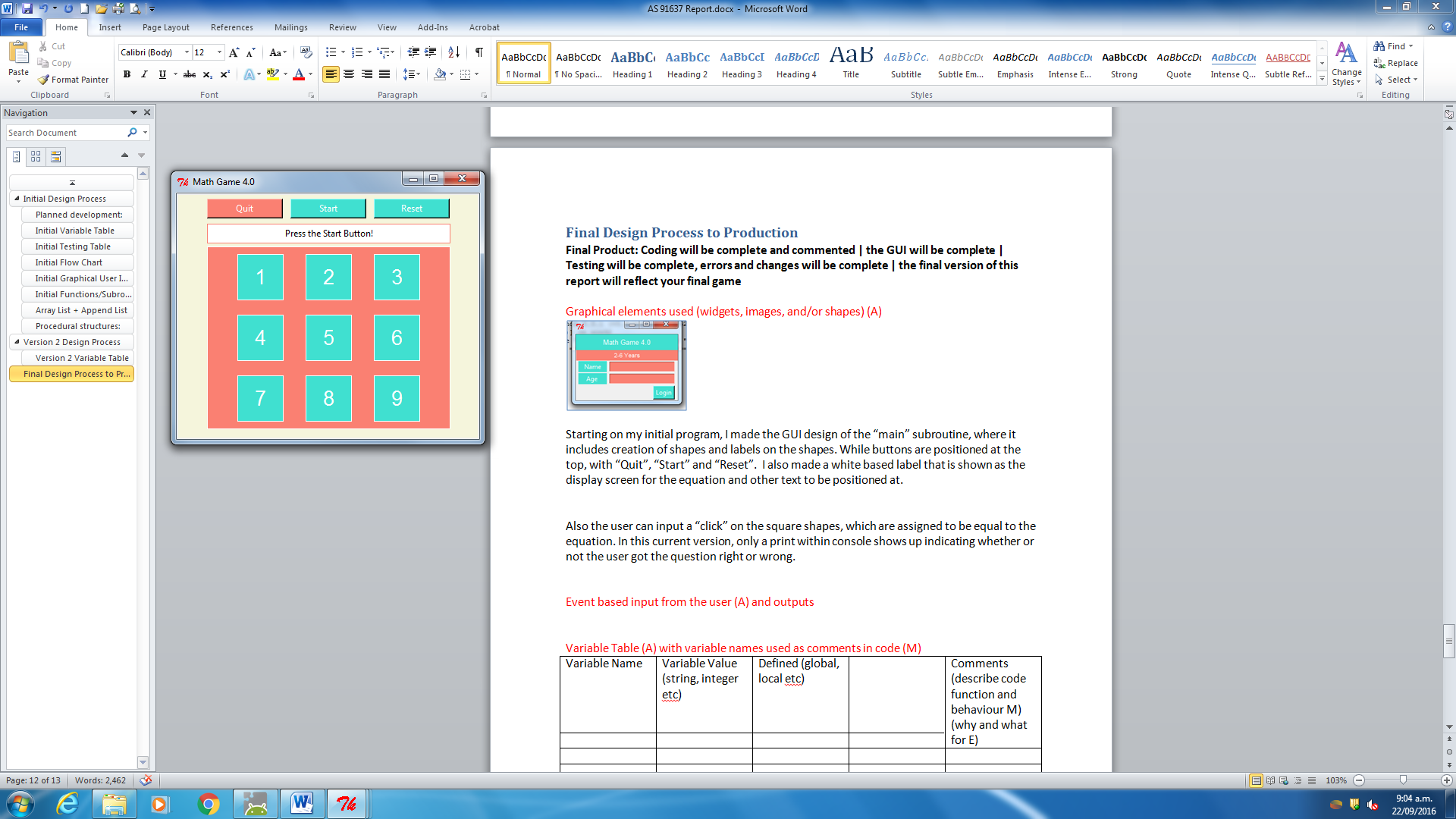
Where Start also determines equation\_function 1, equation\_function 2, equation\_function 3 and Label\_generate

Class loginframe creates the GUI of the login screen, and also determines whether “main” subroutine will be run

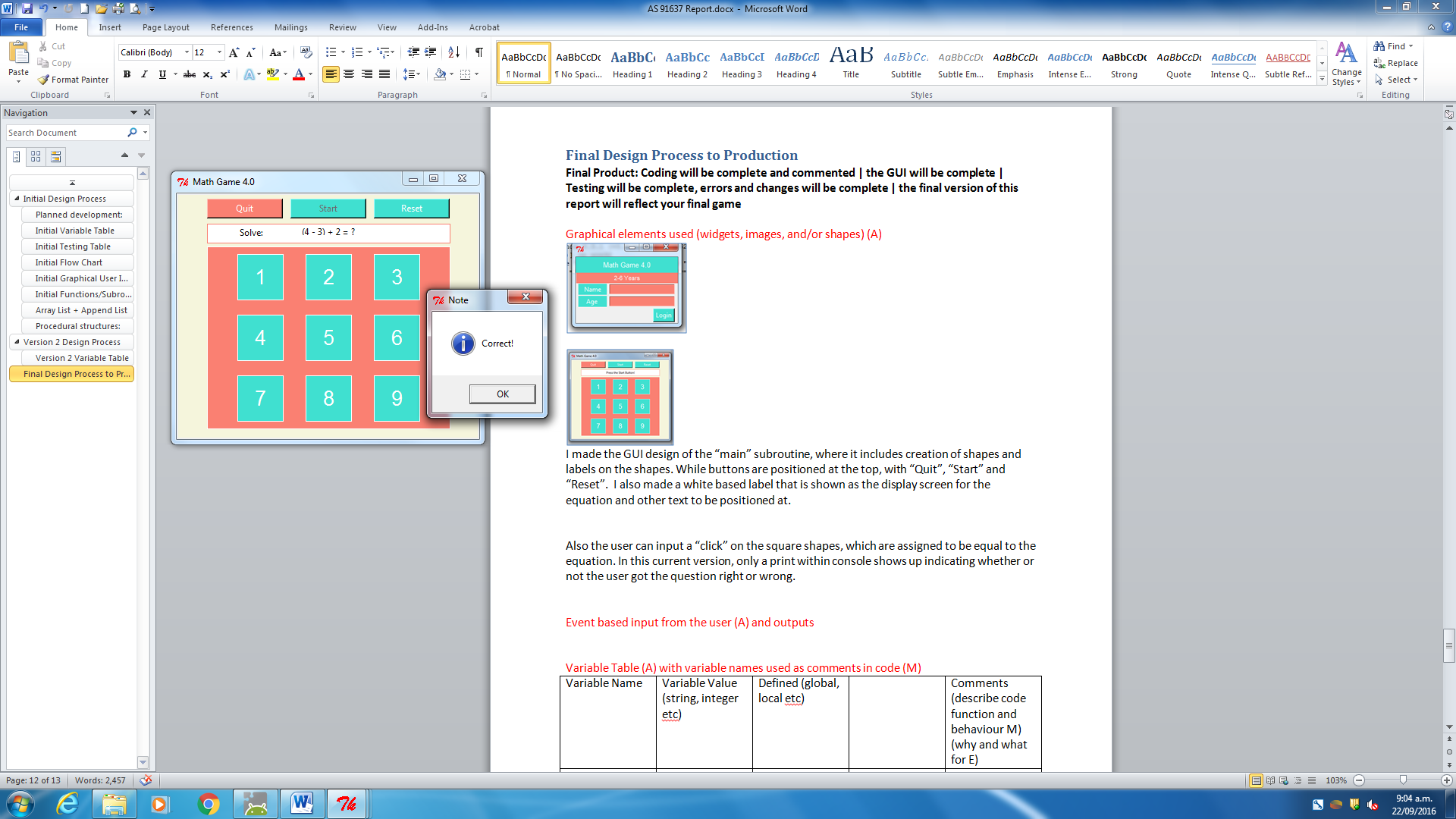
Graphical User Interface Design



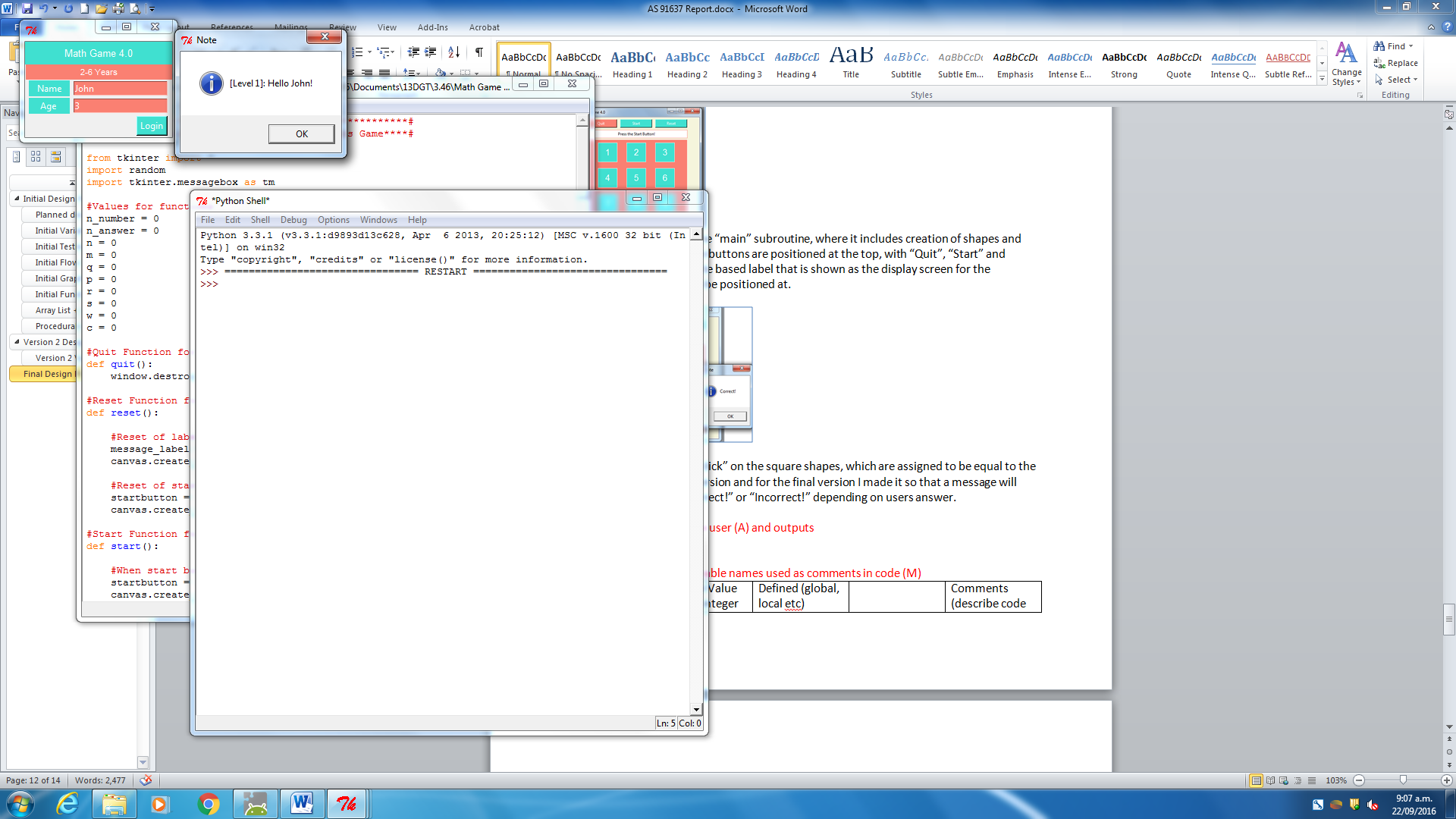
I made the Login GUI screen, using grid placement with labels, inputs and button widgets. Where the class “LoginFrame” is used in order to generate the labels, buttons and user input text.



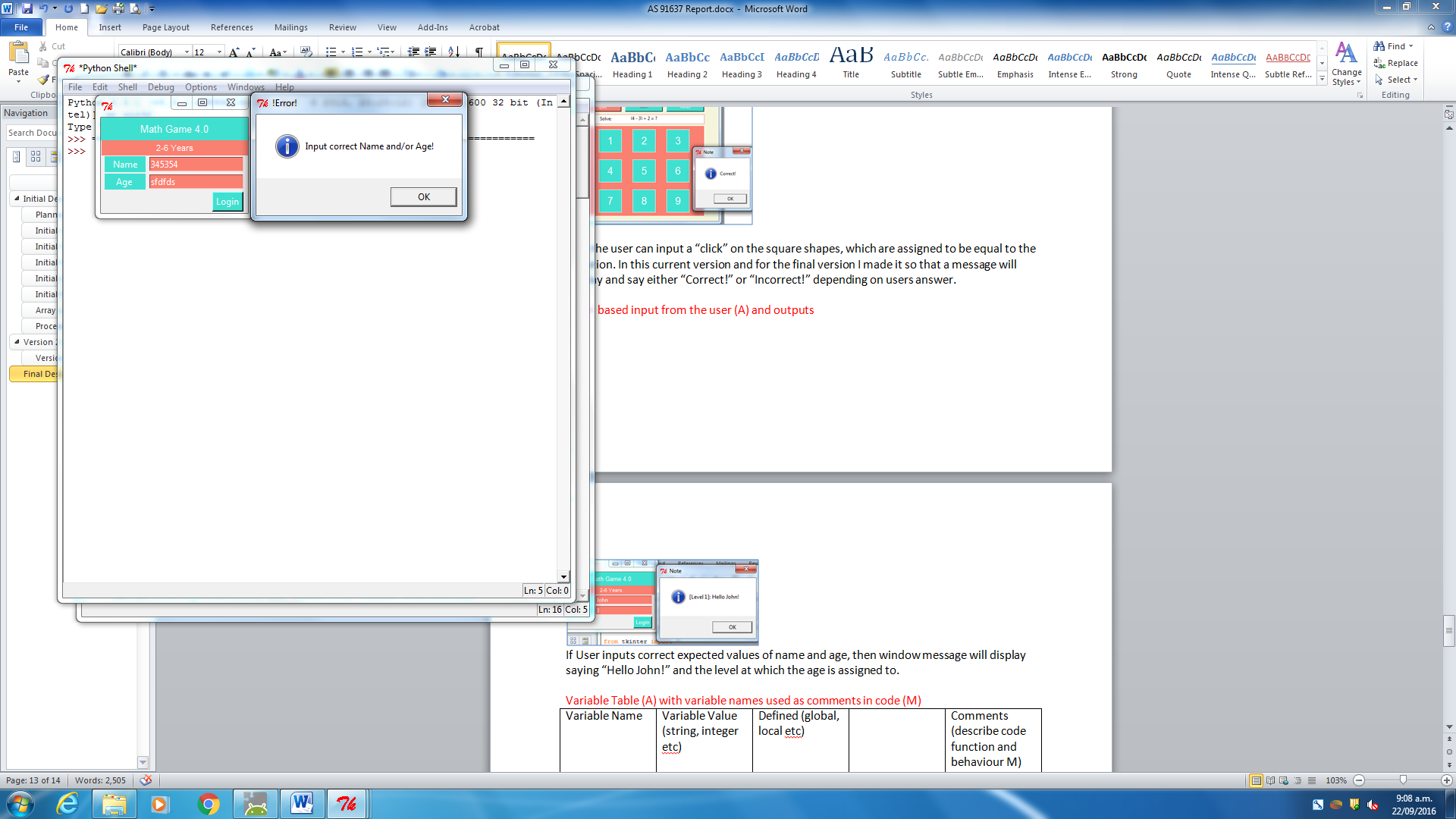
I made the GUI design of the “main” subroutine, where it includes creation of shapes and labels on the shapes. While buttons are positioned at the top, with “Quit”, “Start” and “Reset”. I also made a white based label that is shown as the display screen for the equation and other text to be positioned at.



Also the user can input a “click” on the square shapes, which are assigned to be equal to the equation. In this current version and for the final version I made it so that a message will display and say either “Correct!” or “Incorrect!” depending on users answer.

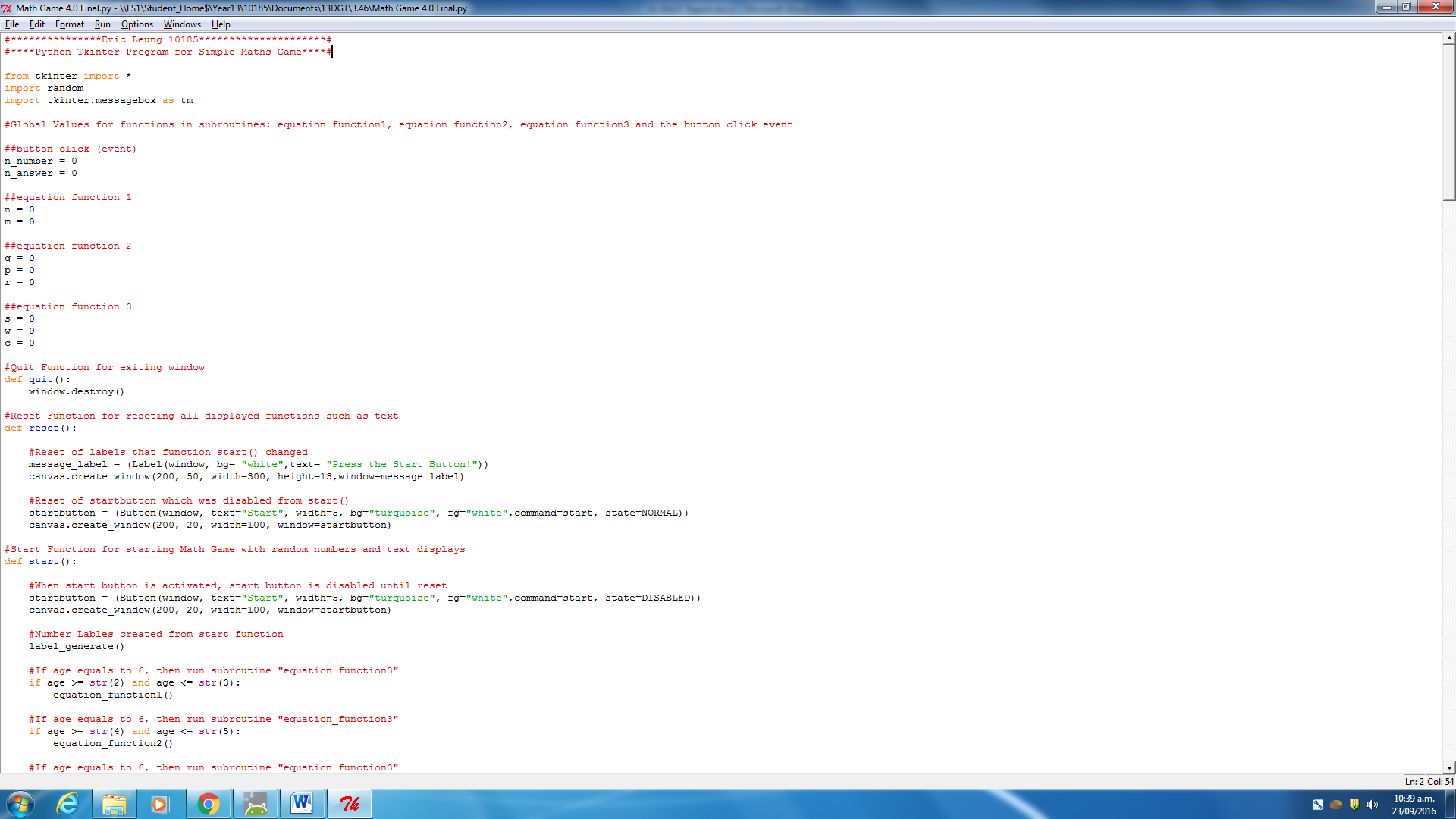


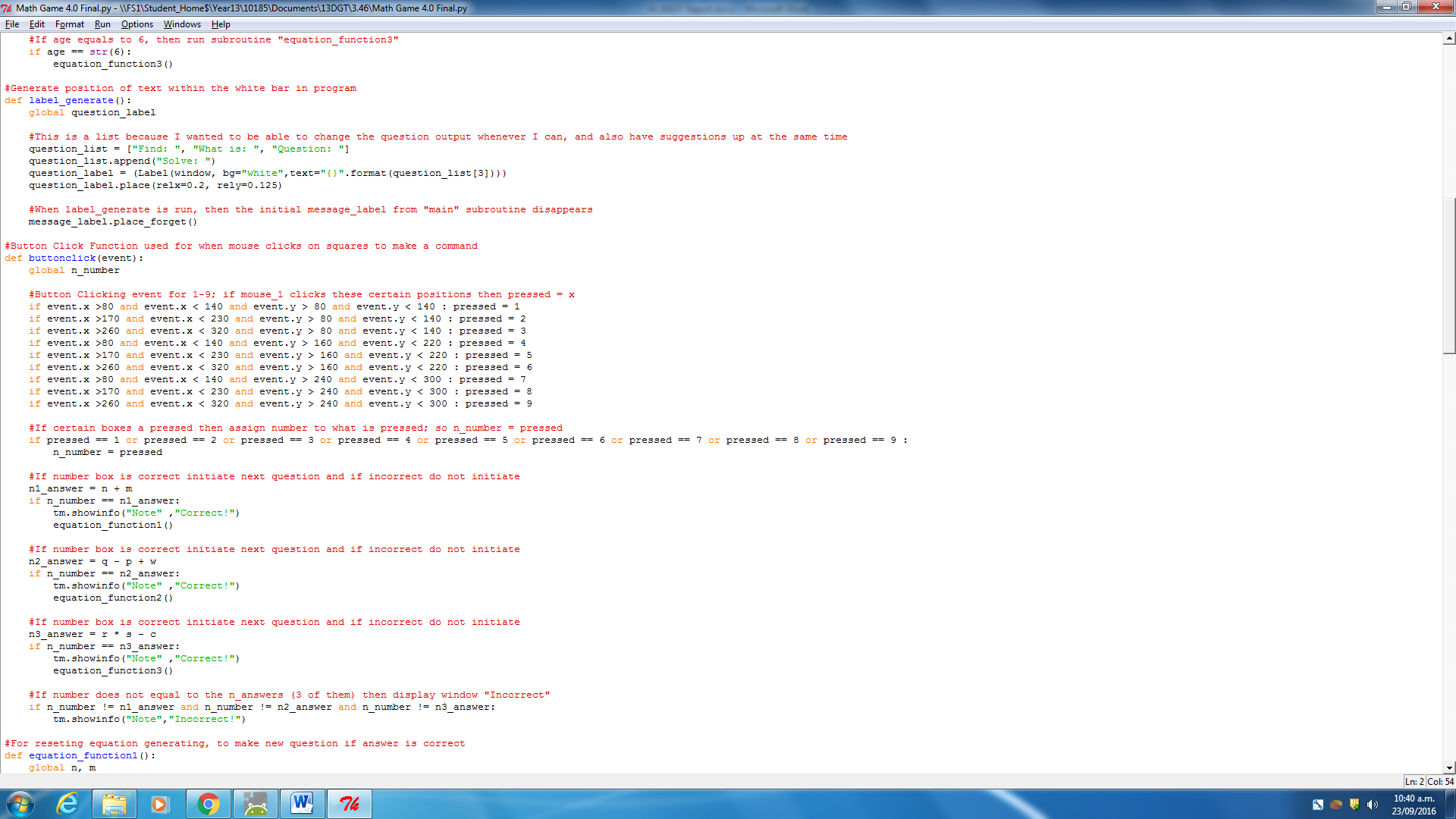
If User inputs correct expected values of name and age, then window message will display saying “Hello John!” and the level at which the age is assigned to.

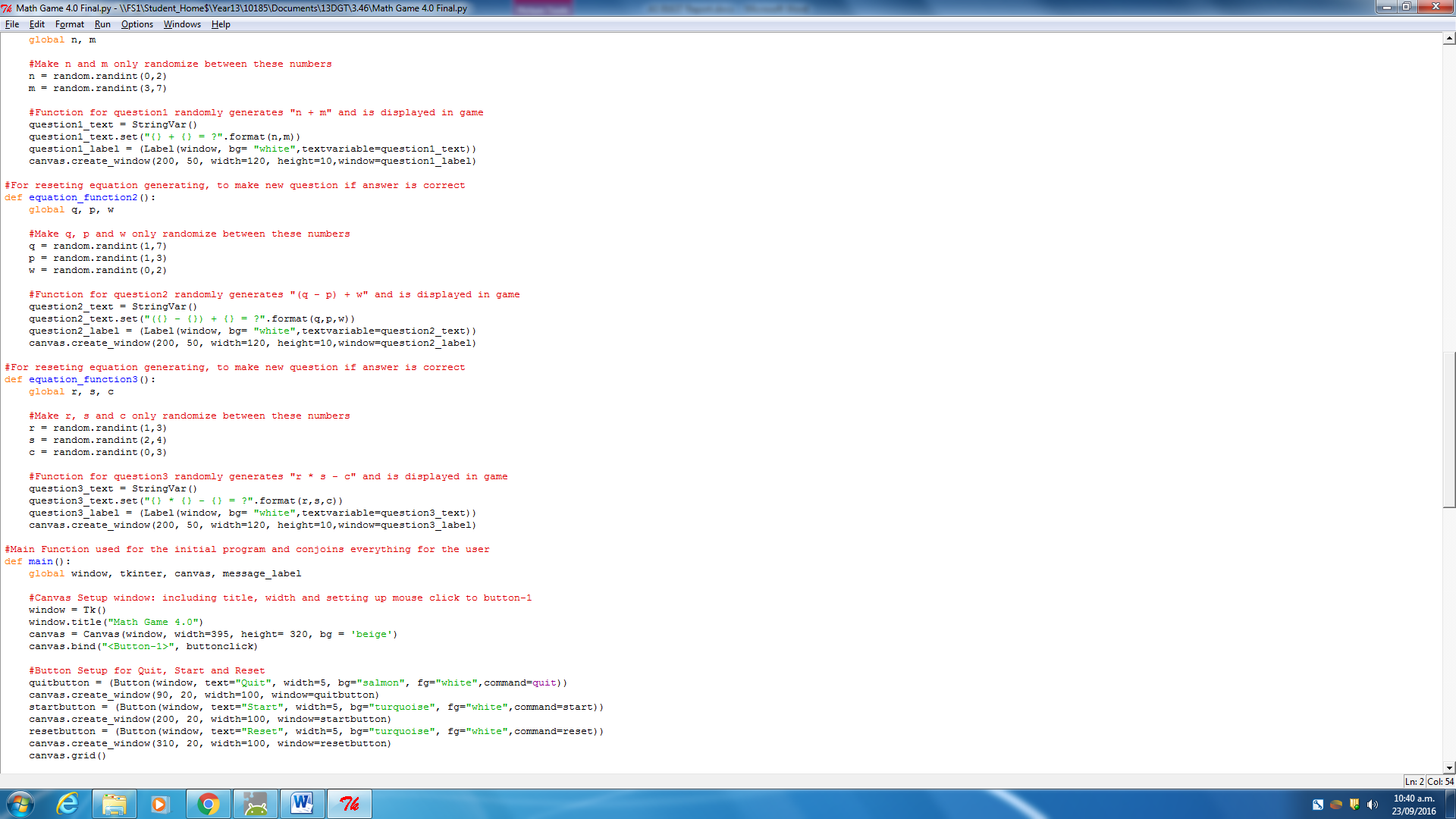


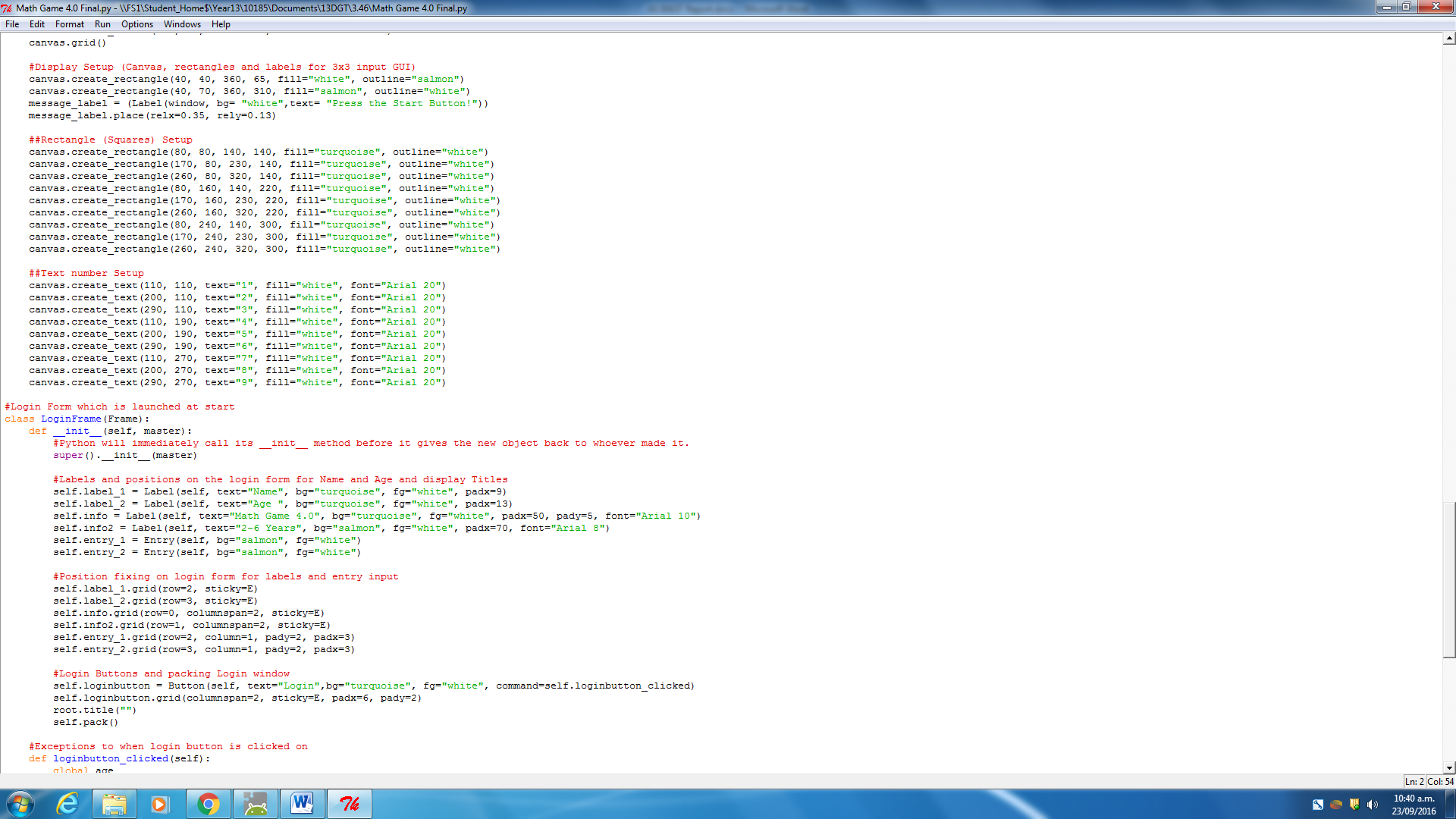
If user inputs incorrect expected values of name and age, then window message will display saying “Input correct Name and/or Age!”

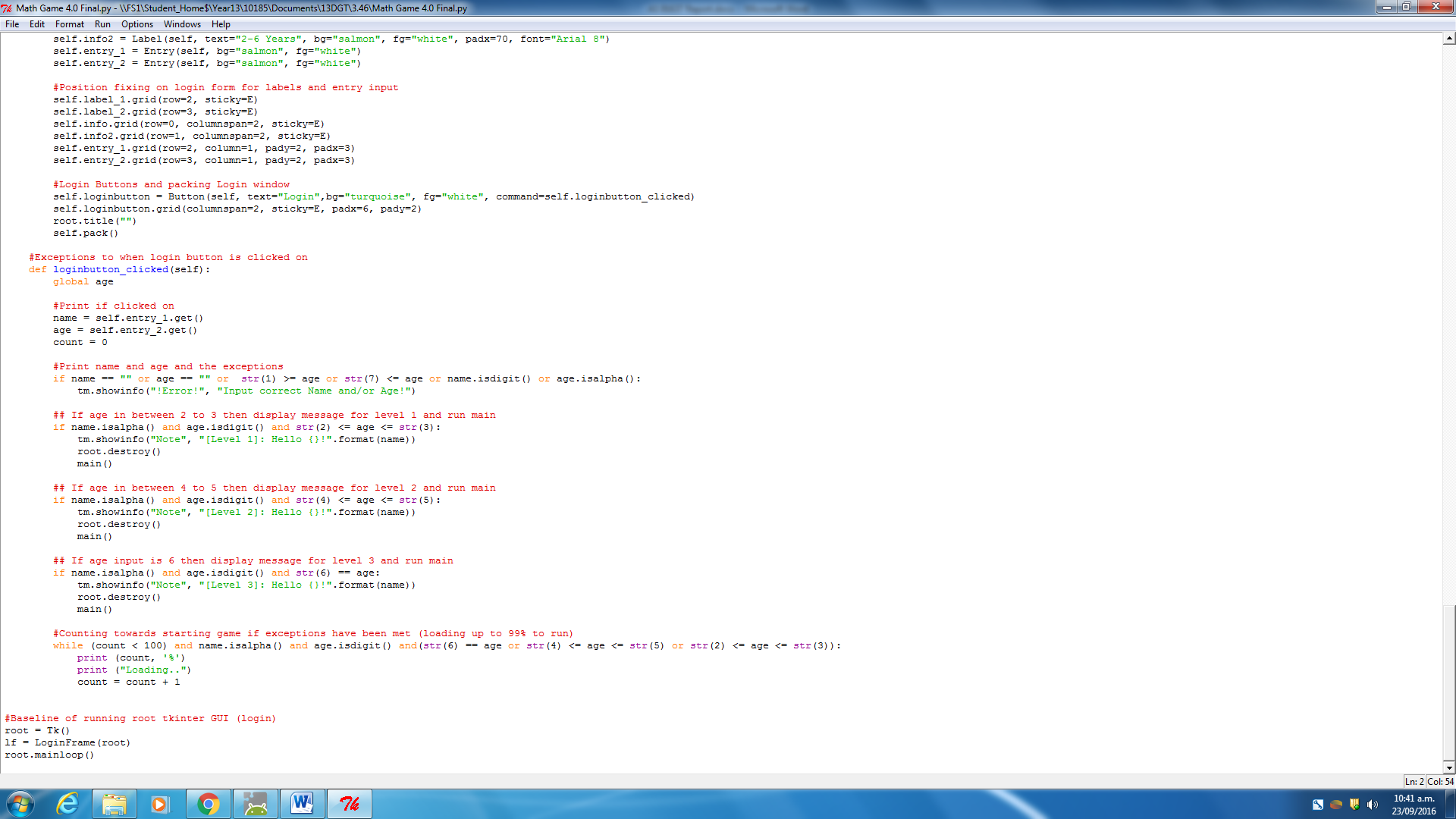
## Final Python code



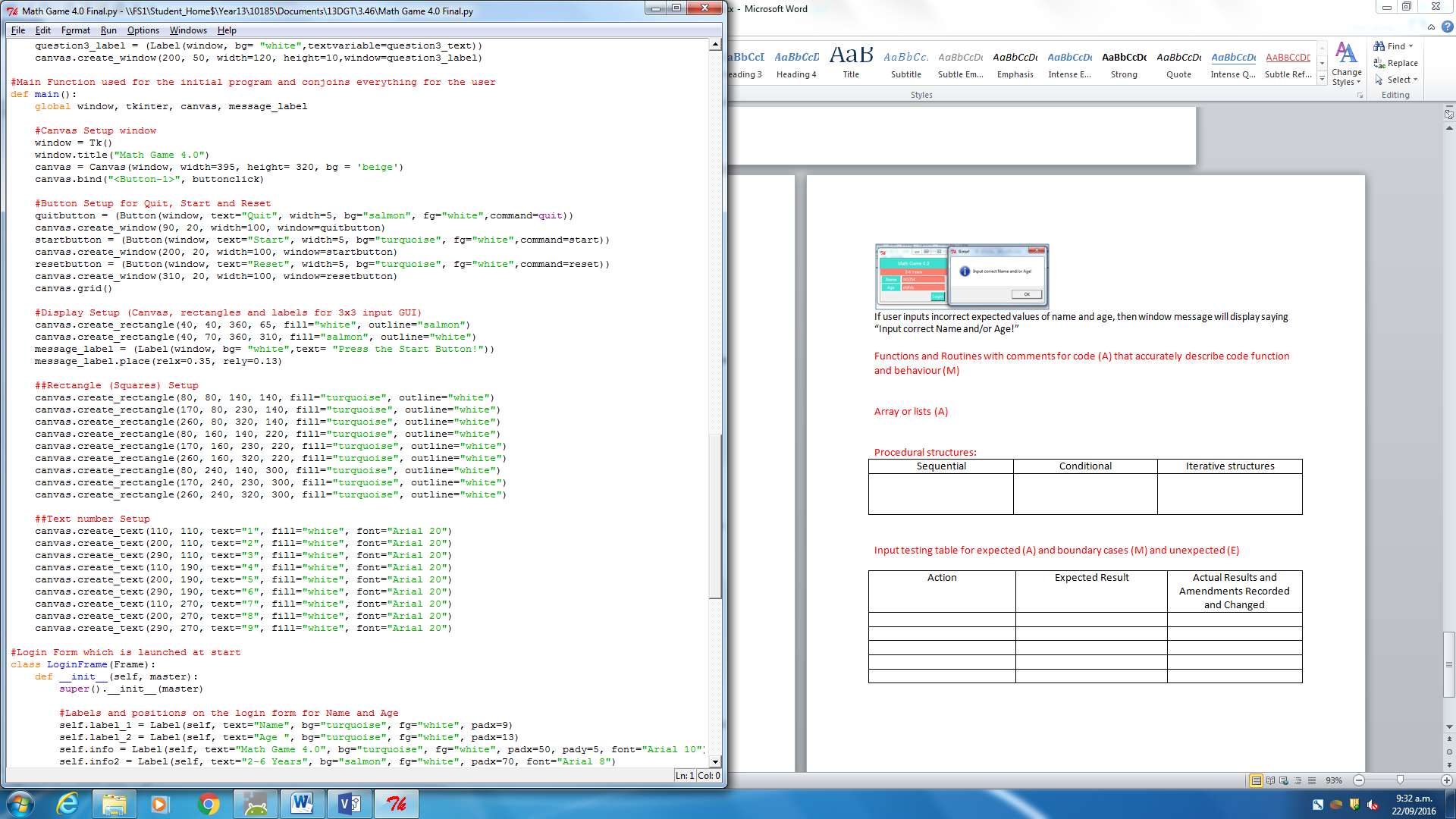








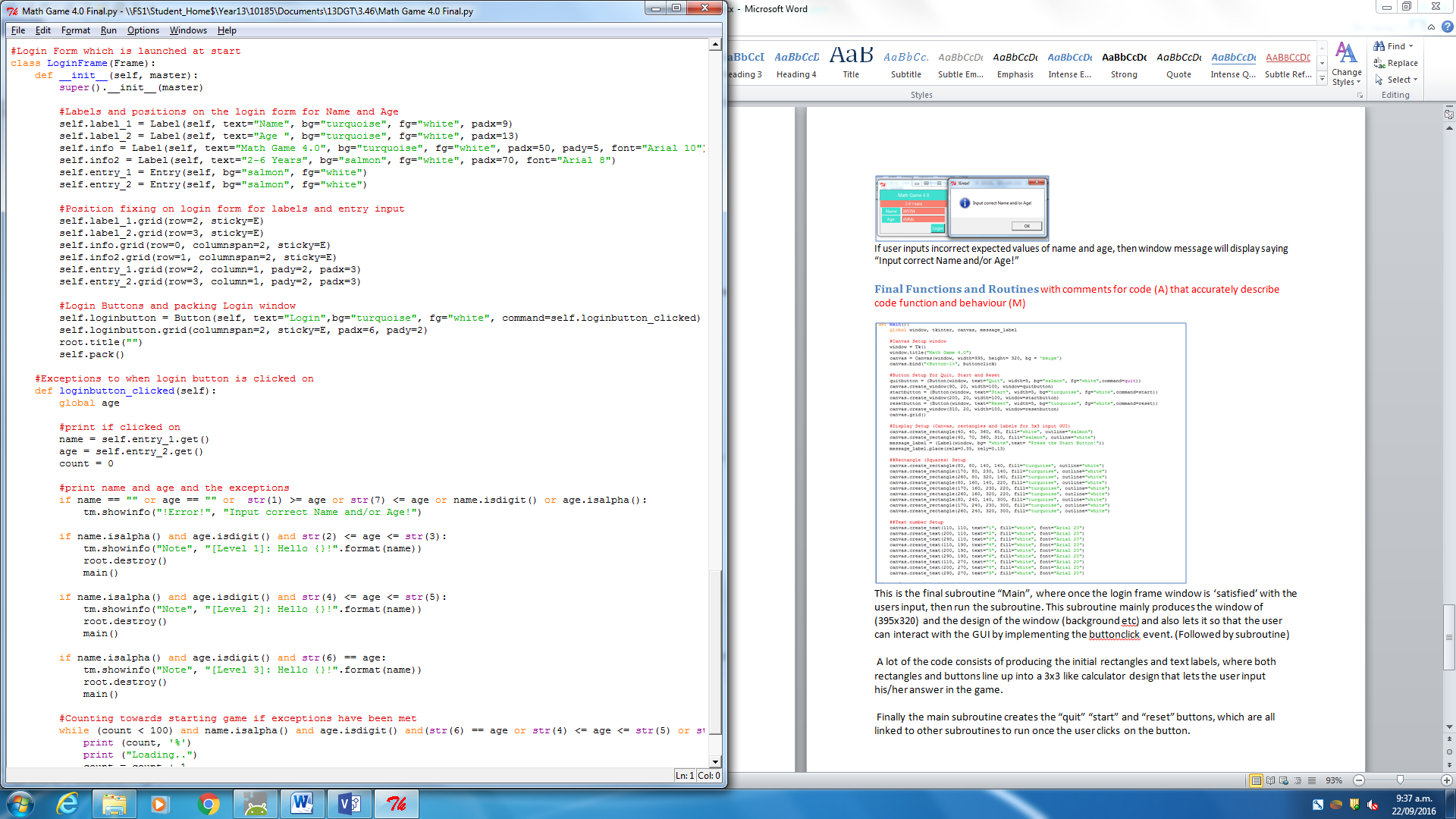
Final Functions and Routines



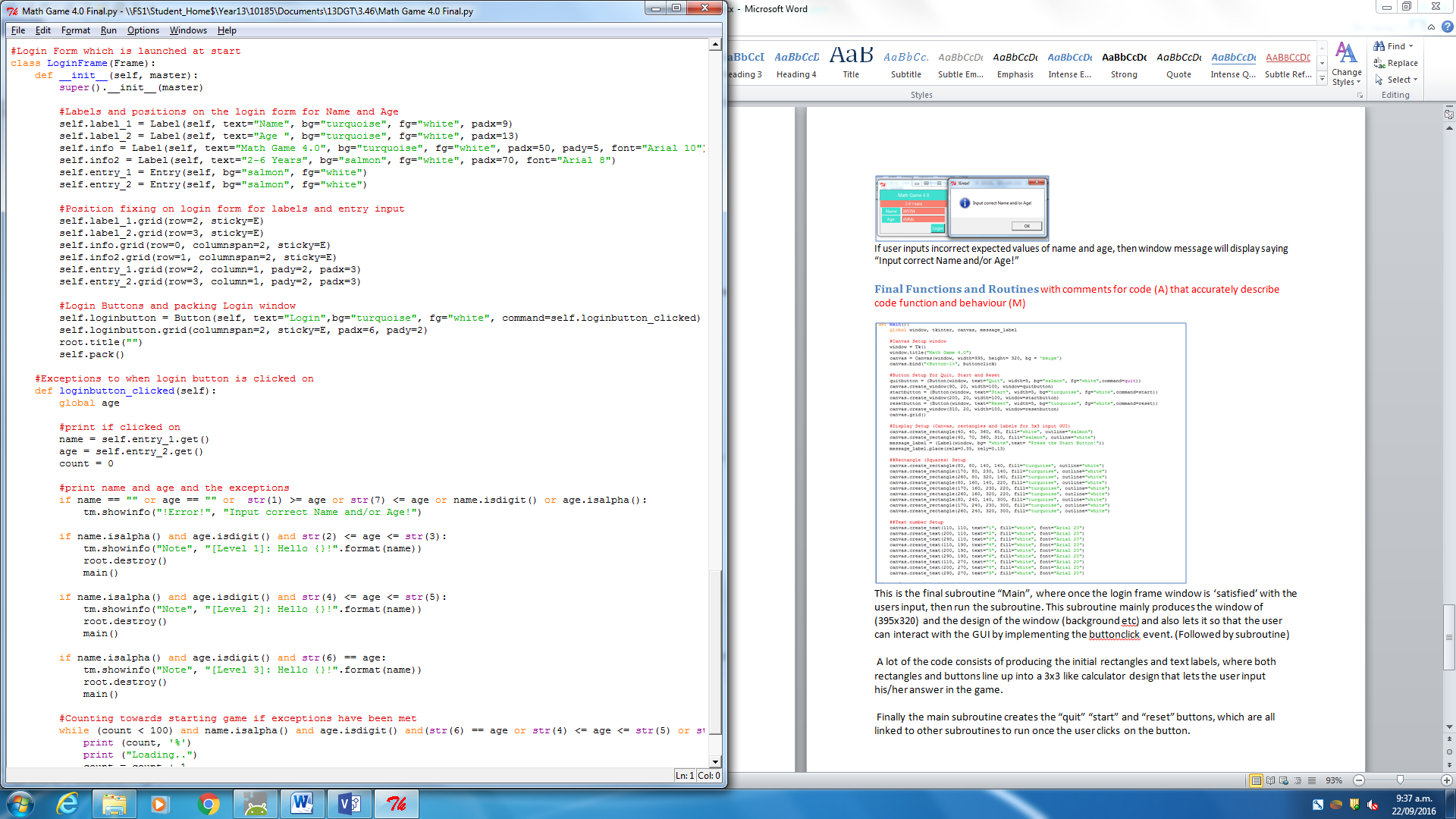
This is the final subroutine “Main”, where once the login frame window is ‘satisfied’ with the users input, then run the subroutine. This subroutine mainly produces the window of (395x320) and the design of the window (background etc) and also lets it so that the user can interact with the GUI by implementing the buttonclick event. (Followed by subroutine)

A lot of the code consists of producing the initial rectangles and text labels, where both rectangles and buttons line up into a 3x3 like calculator design that lets the user input his/her answer in the game.

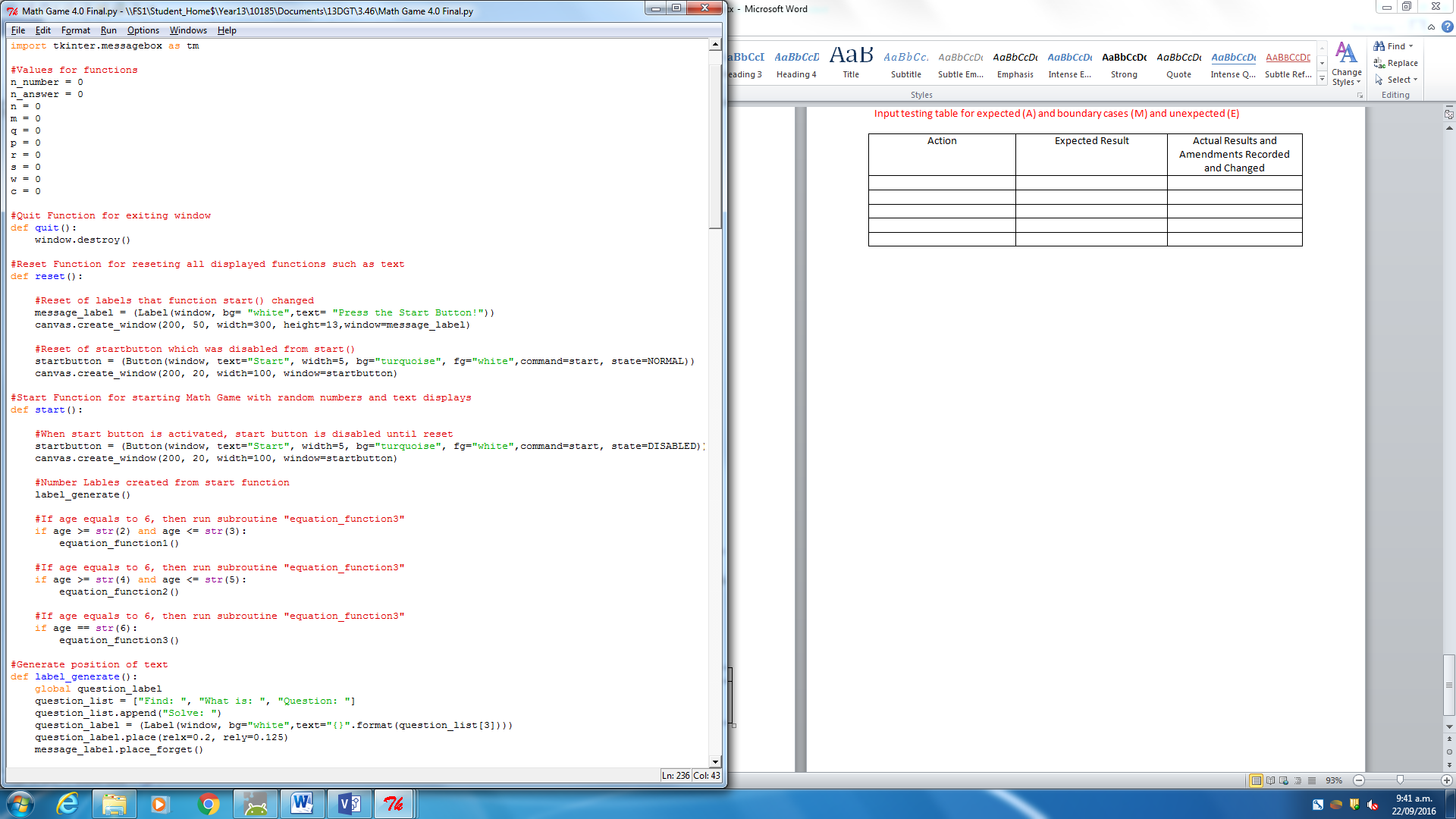
Finally the main subroutine creates the “quit” “start” and “reset” buttons, which are all linked to other subroutines to run once the user clicks on the button.



The class LoginFrame consists of two subroutines, “\_\_init\_\_” and “loginbutton\_clicked”, where the first subroutine is basically similar to the “main” subroutine, where it makes a login window with labels, user inputs and a Login button.

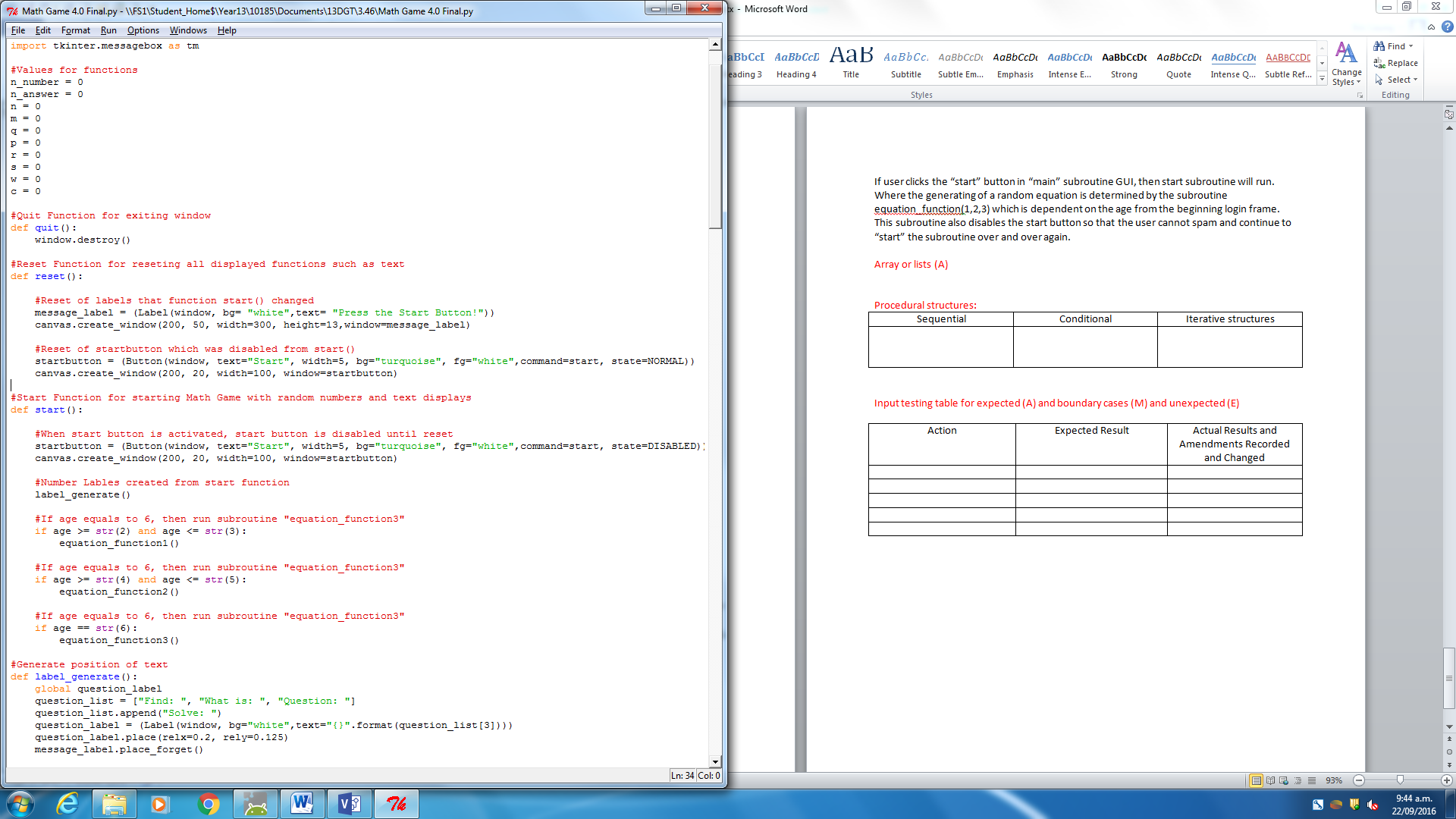


If the login button is interacted with (clicked on) then the “Loginbutton\_clicked” subroutine runs, where it basically tests whether or not the user has correctly or incorrectly entered his/her name and age in right. This also determines the level at which the user will be doing, and so it will run the “main” subroutine if correct.

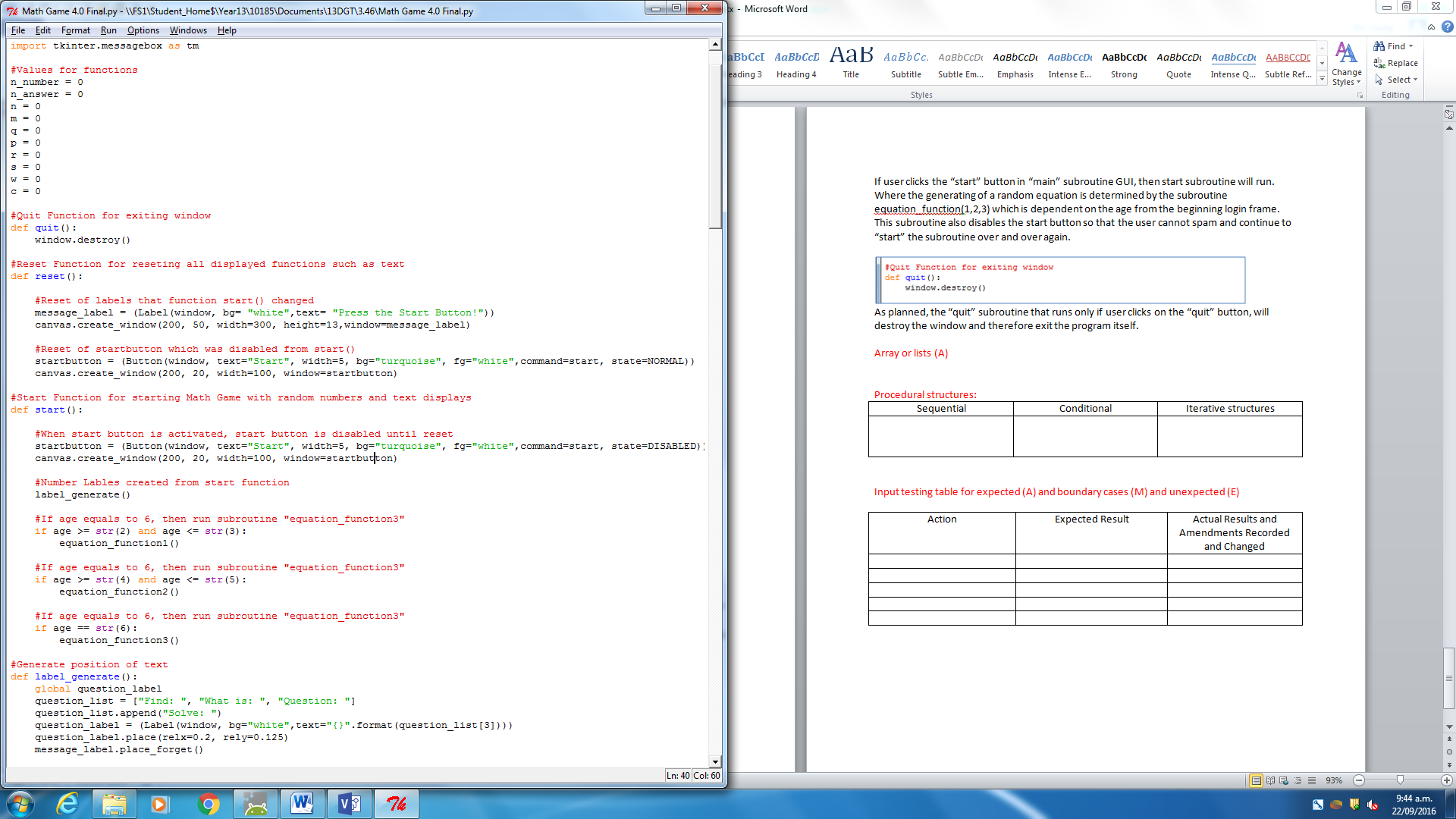


If user clicks the “start” button in “main” subroutine GUI, then start subroutine will run. Where the generating of a random equation is determined by the subroutine equation\_function(1,2,3) which is dependent on the age from the beginning login frame.

This subroutine also disables the start button so that the user cannot spam and continue to “start” the subroutine over and over again.



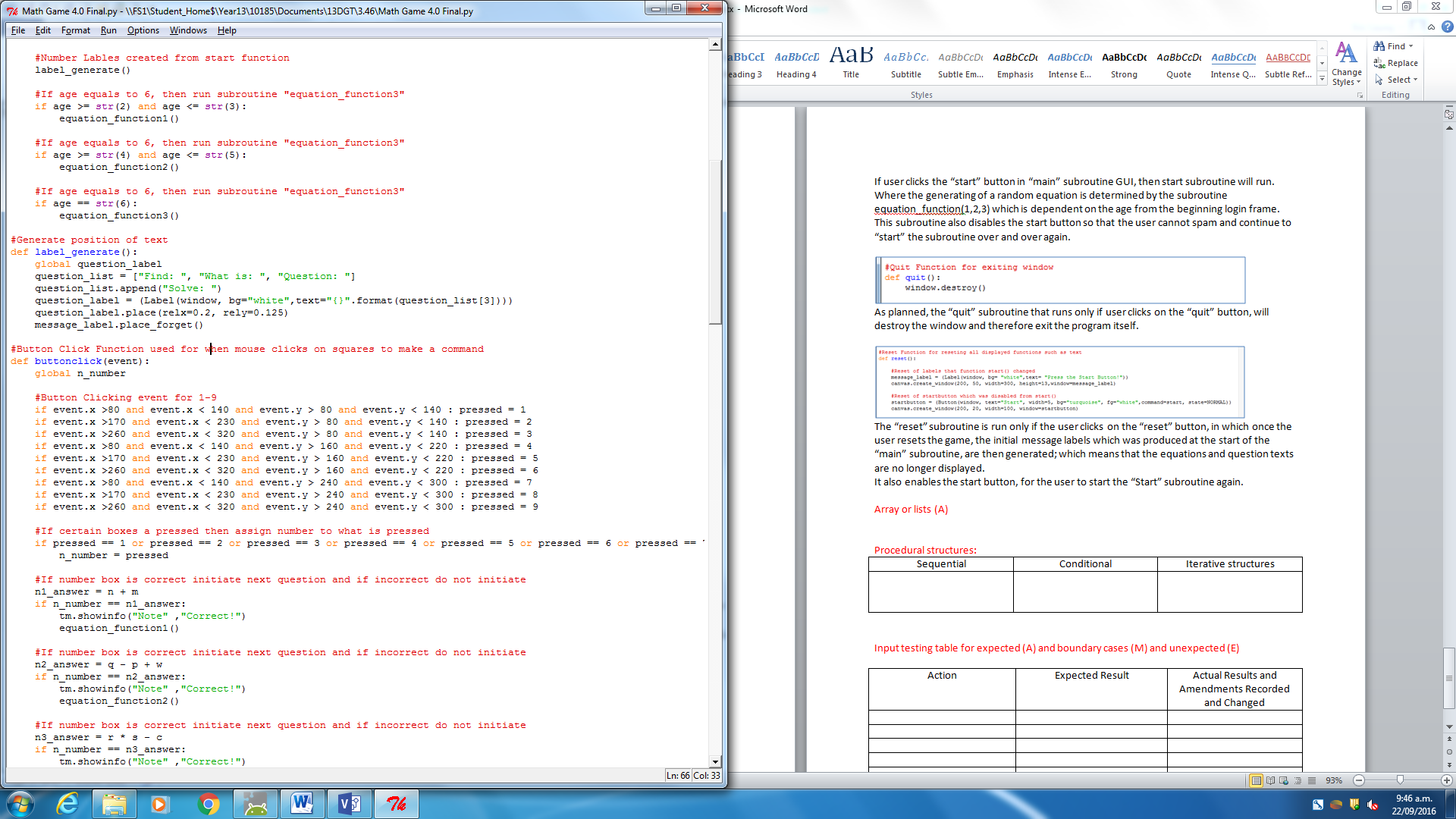
As planned, the “quit” subroutine that runs only if user clicks on the “quit” button, will destroy the window and therefore exit the program itself.



The “reset” subroutine is run only if the user clicks on the “reset” button, in which once the user resets the game, the initial message labels which was produced at the start of the “main” subroutine, are then generated; which means that the equations and question texts are no longer displayed.

It also enables the start button, for the user to start the “Start” subroutine again.

## Final Array/ Lists



My final planned List, includes having a question label display in the “main” subroutine GUI, where once the user runs the “start” subroutine, the question\_label will display and is dependent on what question text it is assigned as; where I can further change and/or add the question text whenever I like.

## Procedural structures:

|  |  |  |
| --- | --- | --- |
| Sequential | Conditional (A) | Iterative structure (A) |
|  | IF | Loop |
| I have made my equation handled as a String where x = a + b, but also handle as integer using StringVar(). | If user input name or age doesn’t meet the requirements, then login screen will continue until everything is valid. Therefore display message\_window saying “Input correct Name and/or age” | If user inputs name and age correctly, then loop a “loading” type screen in the background for when the subroutine “main” is beginning. |

From the initial design process, I had some program code that was assigned to conditional and iterative functions, and from this I further used them throughout my code.

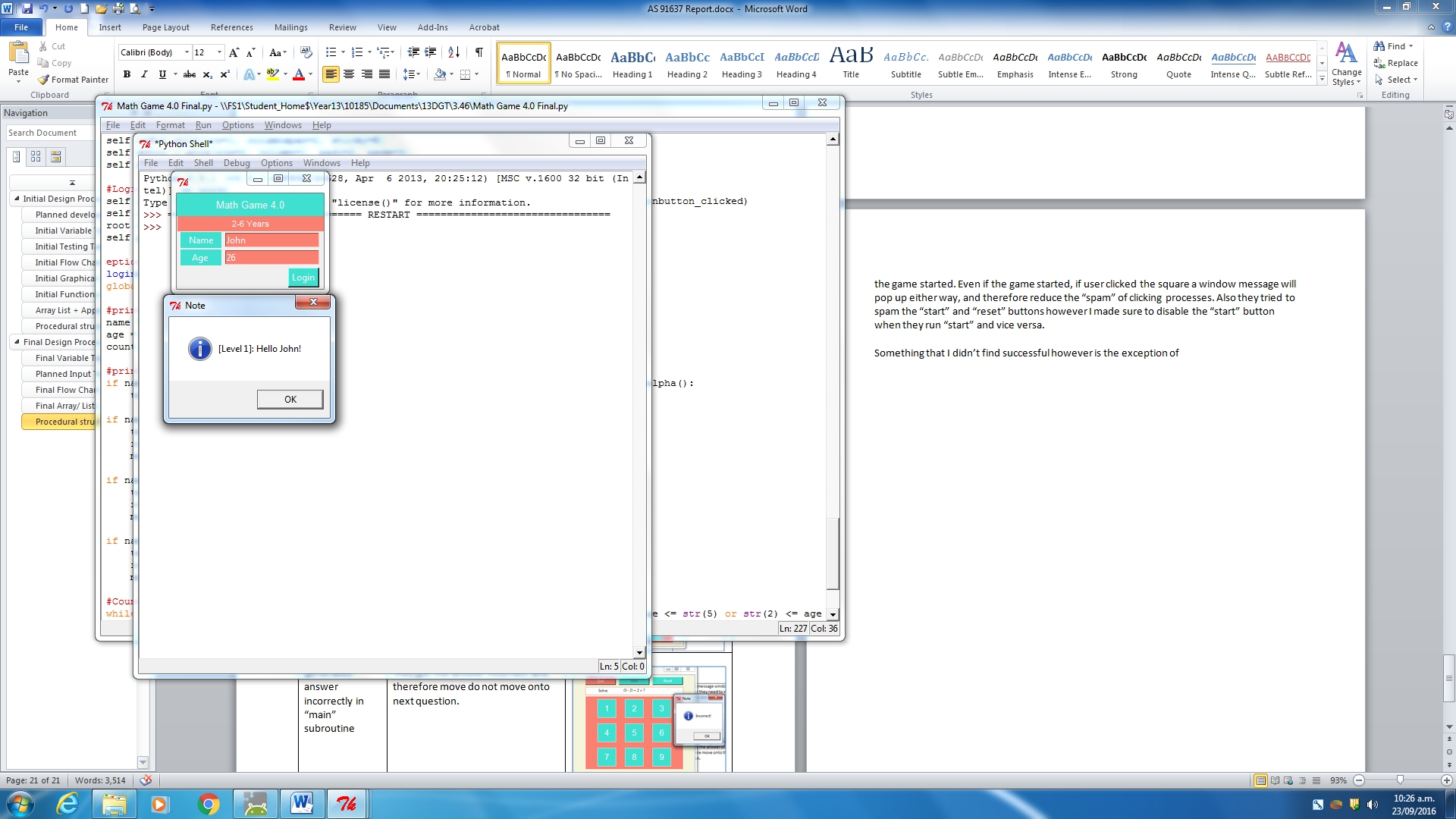
Final Input testing table

|  |  |  |
| --- | --- | --- |
| Action | Expected Result | Actual Results and Amendments Recorded and Changed |
| Name = “John”  Age = “2“ or “3” | Display message window saying the users name and greeting them with level 1 |  |
| Name = “John”  Age = “4“ or “5” | Display message window saying the users name and greeting them with level 2 |  |
| Name = “John”  Age = “6” | Display message window saying the users name and greeting them with level 3 |  |
| Name = “John”  Age = “7“ | Display message window saying to the user they need to input age correctly and validly |  |
| Name = “324”  Age = “2” | Display message window saying to the user that they input name incorrectly |  |
| Name = “John”  Age = “1” | Display message window saying to the user they need to input age correctly and validly |  |
| Name = “ ”  Age = “ ” | Display message window saying to the user that they input name and age incorrectly |  |
| Name = “John”  Age = “John” | Display message window saying to the user they need to input age correctly and validly |  |
| If user clicks generated answer correctly in “main” subroutine | Display message window saying that they got the answer correct, and therefore move onto the next question. |  |
| If user clicks generated answer incorrectly in “main” subroutine | Display message window saying that they got the answer incorrect and therefore move do not move onto next question. |  |

I asked a student to test my program and told them to test as much random inputs as they can. From this, they entered things like Name = “akdjshdkjahskjdahds” and Age = “1234567890”, which although bound to fail due to age, the name is still accepted.

They also entered Name = “” and Age = “” which was unexpected error, and so I made sure to include an exception to when they do not enter either name and/or age.

When they were testing out my “main” game within the subroutine after a successful login, they tried to click everywhere and to click on as many buttons (committing to spam), however I made sure that the squares can only be clickable and would only bring effect if the game started. Even if the game started, if user clicked the square a window message will pop up either way, and therefore reduce the “spam” of clicking processes. Also they tried to spam the “start” and “reset” buttons however I made sure to disable the “start” button when they run “start” and vice versa.



Something that I didn’t find successful however is the exception in the login screen, where if the user entered a number that included numbers from 2 to 6 even if it was higher (such as “26” then the program will still except the input and run the “main” subroutine.