Computer Science NEA

# Section 1 and 2:

## Assess:

* Get name and age
* Split string to first 3 characters and add age to the end of the string
* Get user password

## Design:

name = INPUT What is your name?

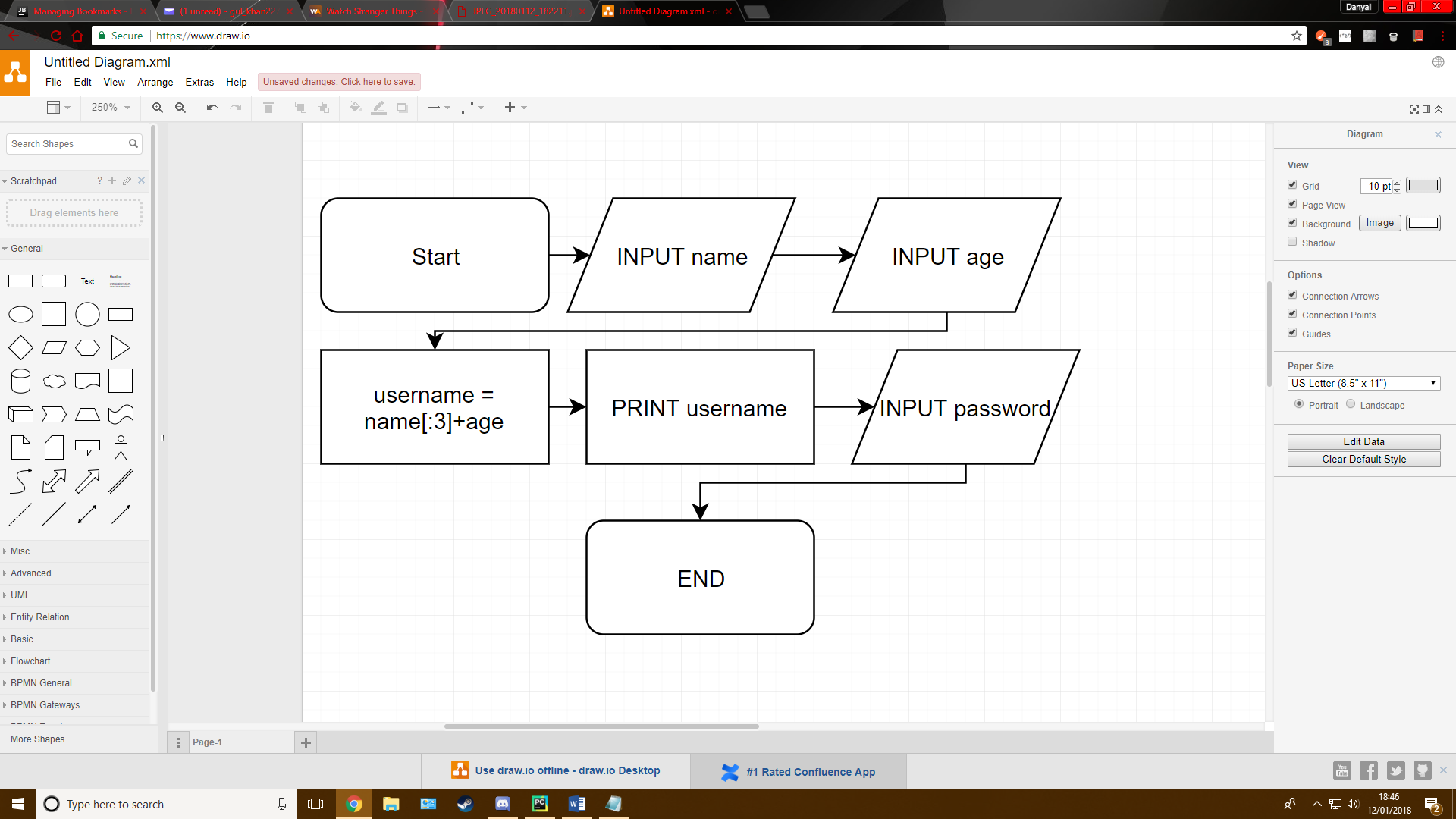
age = INPUT What is your age?

username = name[:3]+age

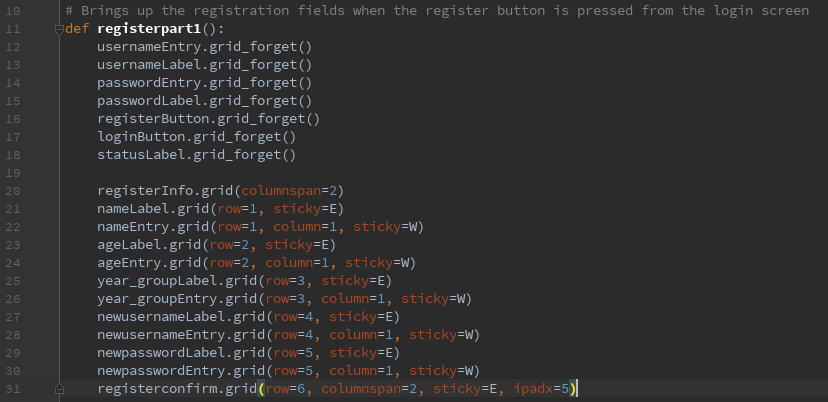
PRINT username

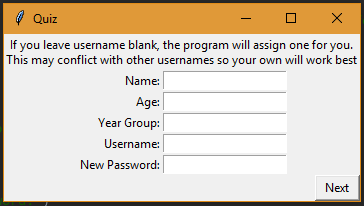
password = INPUT Enter a password for your account

Add details to file “User Logins.txt”



## Develop:

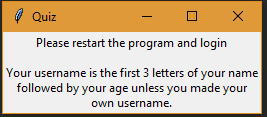
I have given the user the choice to create a username but if they want the program to assign one then the format given in the brief is used.

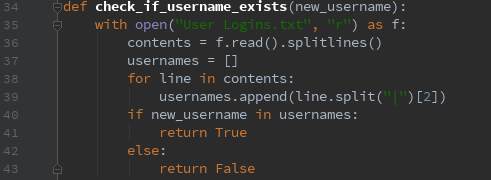
This is the function executed when the user presses the register button on the login screen. It brings up the necessary entry fields detailed in the brief as well as the aforementioned option of creating a unique username.

This is the register screen where the user will enter the details. It also tells the user that the program will assign a username if the username field is left empty. This is a good addition to the brief as people with the same first 3 letters and age would be stuck with the same username. This is something that should not have to be a problem and gives a more tailored experience for the user.

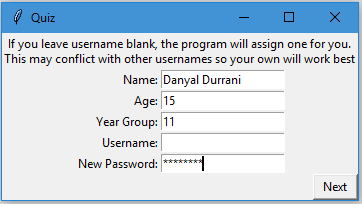


*Note: In the root.after line, 2000 has been changed to 4000 to allow for easier reading*

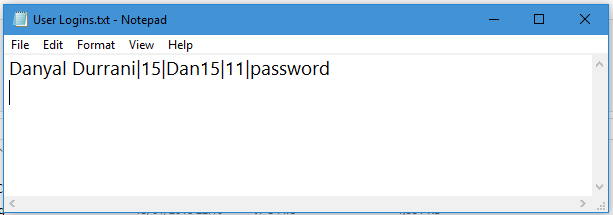
This is the function executed when the user presses the ‘Next’ button. It takes the values that the user has entered in each field and then stores it an appropriately named variable. This also checks whether the user has entered something in the username field and generates a username based on the outcome of the if statement. Next, it checks if the username already exists by using another function that I will show below. If the username does already exist, then the username and password fields are cleared, and it tells the user for 4 seconds that the username already is taken and that they must enter a new username and the function is stopped. If the username doesn’t exist, the program writes the information provided by the user in to a txt file called User Logins. Finally, it removes the widgets and shows a label that says:

This is the function that checks each username and sees if the username exists. It looks at each line and splits it, so it only deals with the usernames. It then adds the username to the list and checks whether the username the user has entered is from already in use. This also works for program generated usernames.

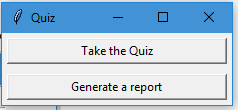
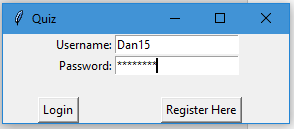
## Testing:



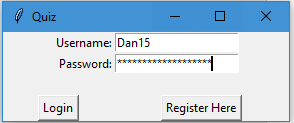
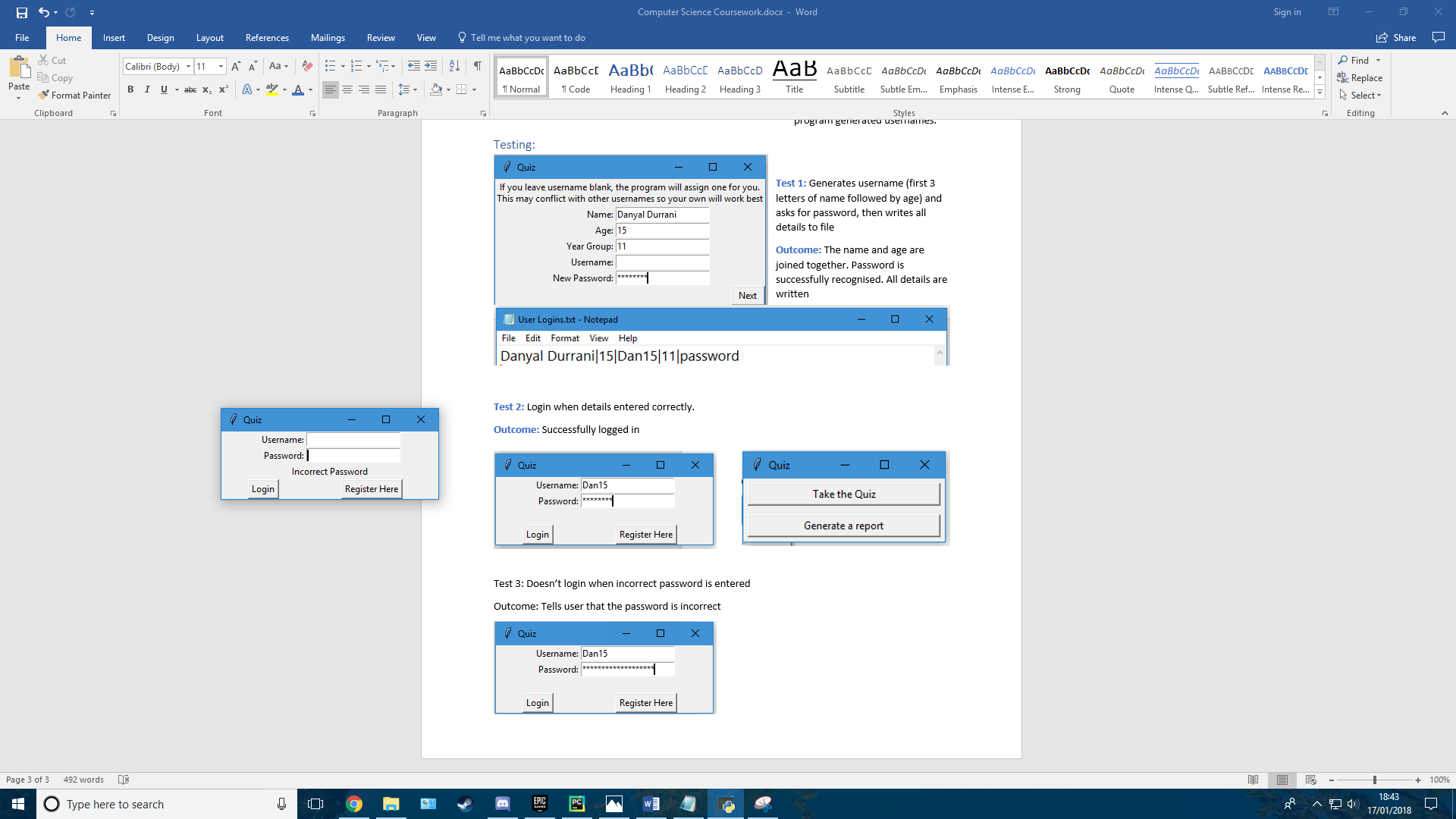
**Test 1:** Generates username (first 3 letters of name followed by age) and asks for password, then writes all details to file

**Outcome:** The name and age are joined together. Password is successfully recognised. All details are written

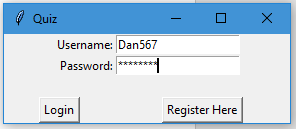
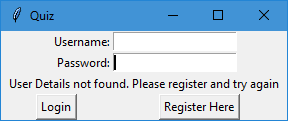
**Test 2:** Login when details entered correctly.

**Outcome:** Successfully logged in

**Test 3:** Doesn’t login when incorrect password is entered

**Outcome:** Tells user that the password is incorrect

**Test 4:** Doesn’t login when unregistered username entered

**Outcome**: Tells user to register

# Section 3 (and code for Section 4):

## Assess:

* Choose difficulty
* Choose subject
* Get the questions from the question bank and store them in a variable

## Design:

difficulty = INPUT Easy, Medium or Hard?

subject = INPUT subject

IF subject == physics THEN

DO Load physics questions with correct difficulty

ELIF subject == maths THEN

DO Load maths questions with correct difficulty

ELIF subject == general knowledge THEN

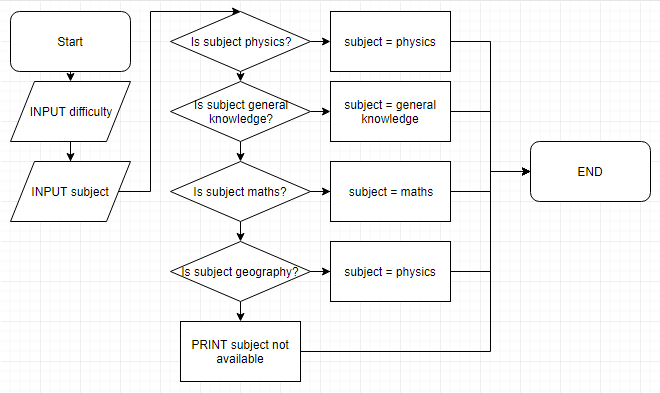
DO Load general knowledge questions with correct difficulty

ELIF subject == geography THEN

DO Load geography questions with correct difficulty

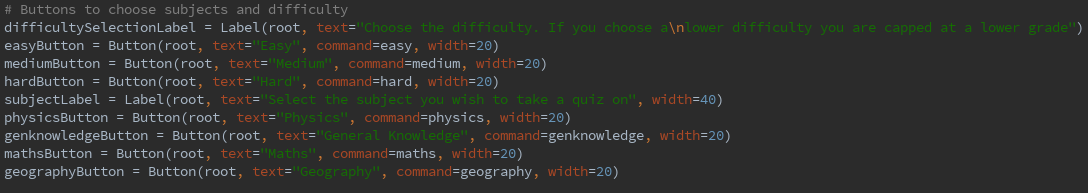
ELSE

DO PRINT subject not available

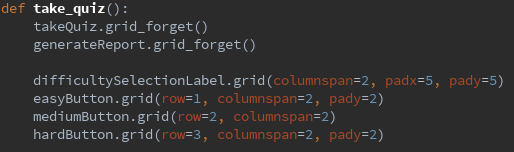
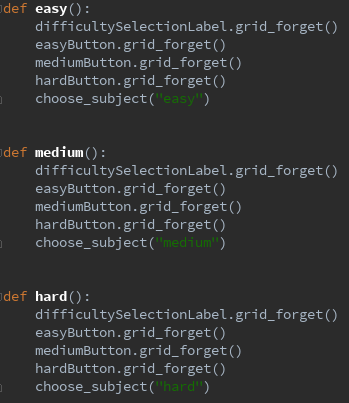


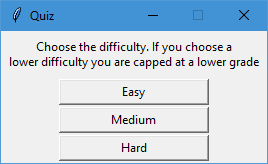
*Note: In the final selection in the flow chart, it should lead to subject = geography, not physics (was a typo and not intentional)*

## Develop:

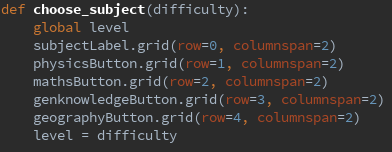
These are the widgets that are involved in the choosing of subject and difficulty. The width of each widget is set manually as TKinter defaults to resize the buttons to the amount of text units it contains. This makes the program look unpolished and not uniform, so manual widths have been set to counter this.

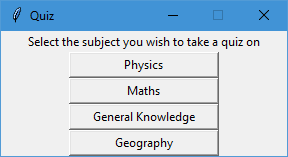
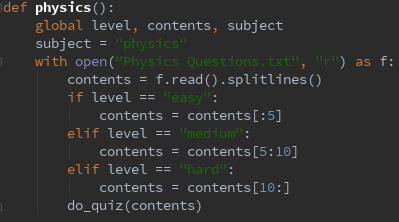
If you look at test 2 in section 1 and 2, you can see that the user is presented with two options, take the quiz or generate a report. The function below shows what happens when the user selects “Take the Quiz”.



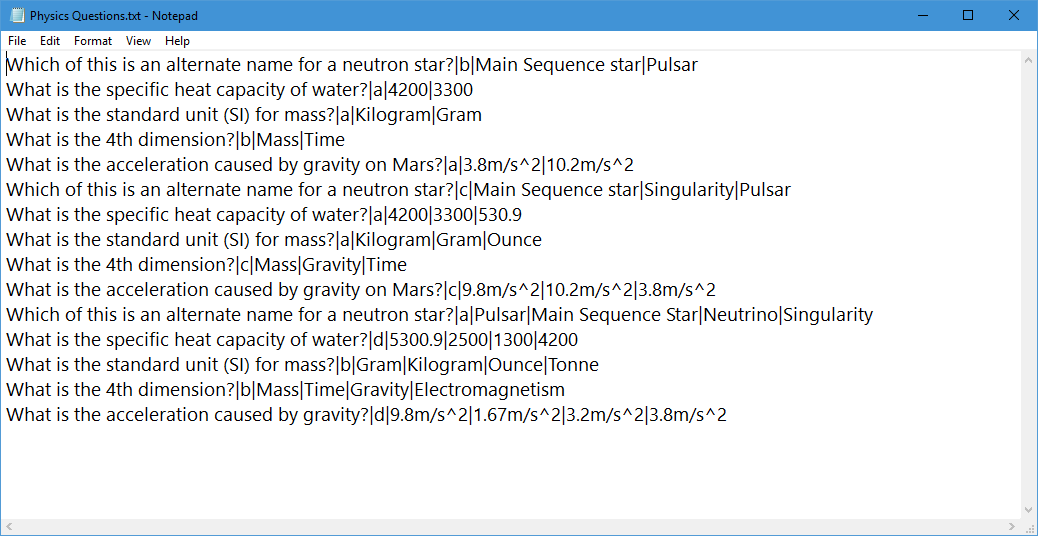
This removes the 2 previous widgets and replaces them with the difficulty selection menu, which looks like the window on the right.

When the user clicks one of these buttons, one of the functions on the left is executed. In each of these functions, it removes the current widgets, and calls the same function “choose\_subject”, however the parameters for each case is different, where the difficulty selected is written in the brackets.

The first thing that happens is that we enable this function to change the value if the global variable “level”. Then the subject selection menu is brought up, and finally the value of the parameter difficulty is assigned to level. The menu looks like this:

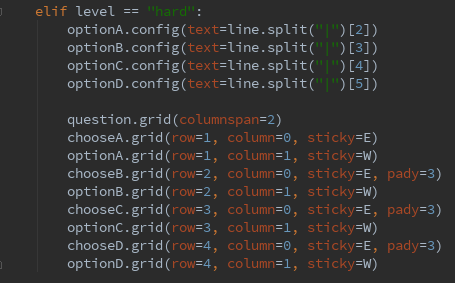
This is like the difficulty selection menu, as the same basic concepts are applied here. Each of the buttons has its own function. All these functions are identical apart from the file that these questions read. The physics function reads the physics file, the maths function reads the maths file etc. Here is an example of one of these functions:

The function uses the global variables; level, contents and subject. Level is shown above, contents is the file contents and the subject variable is used when writing the user results to the results file. All the difficulties are stored in the same file, to mean less clutter and less space used on the drive (optimised for space). Here is the file for physics:

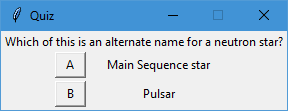
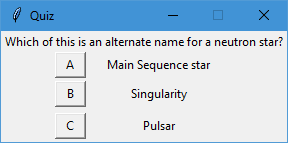


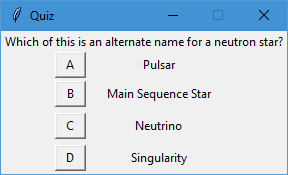
The program reads this, and selects the correct lines depending on difficulty and then calls the “do quiz” function with the parameter contents, which are the questions. This is the do quiz function:

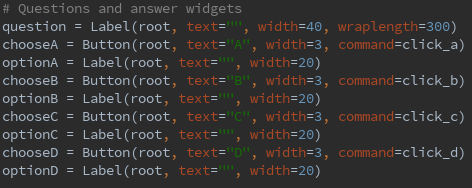
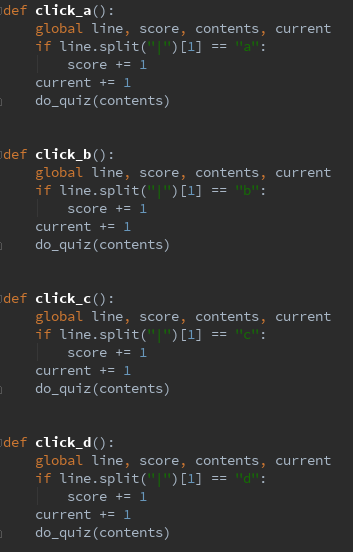
This is the longest function in the program so far, but only because of the repeated replacement of the widgets which is necessary to move to the next question. The widgets will be shown under this.



First, it declares the global variable line, current and level. Line is the line in the text file that is used, current is the index of that line and level is the difficulty.

First, it tries to assign the item at in current position in question (the parameter). If an Index Error is returned, it means that there is no item at current position, meaning that all the questions have been asked and so it does the final function, which is section 5 of the brief. If the line assignment is successful, then it removes the widgets from the subject menu and starts the quiz. It checks the value of level, then it assigns the options that are in the text file and assigns it to optionA, optionB etc. Then it places these on the screen. The option widgets are labels that have the options and the choose widgets are buttons, which the user presses.

The first window is easy, the second is medium and the third is hard.

Here are the widgets for the questions and answers:

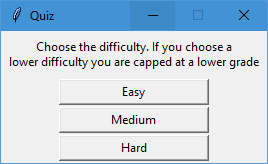
Here are the functions that are executed when a button is clicked:

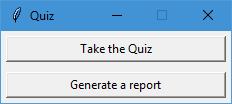
First, it declares line, score, contents and current as global variables, then checks if for that question if the answer is correct. If this returns true, then it increases the score by one and increases current by one, meaning that the next time do\_quiz is performed, the next question is asked.

If current is greater than or equal to six, it will trigger an index error, which is why I used the try and except statement above.

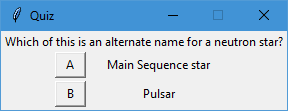
## Testing:

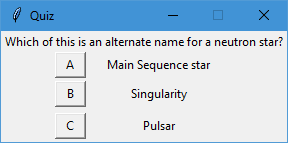
**Test 1:** Pressing Take Quiz takes you to the difficulty selection menu

**Outcome:** The difficulty selection menu is brought up

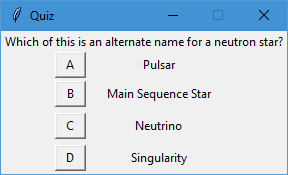


**Test 2:** Selecting Easy then Physics will bring up a physics question with 2 options

**Outcome:** User is presented with 2 options

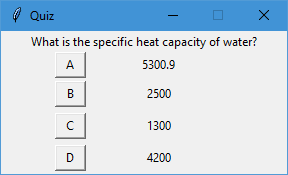
**Test 3:** Selecting Medium then Physics will bring up a physics question with 3 options

**Outcome:** 3 options



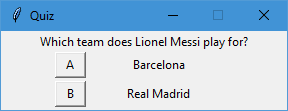
**Test 4:** Selecting Hard then Physics will bring up a question with 4 options

**Outcome:** 4 options



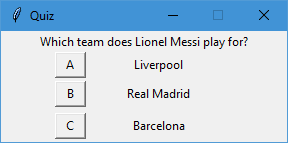
**Test 5:** Selecting an answer will bring the next question

**Outcome:** New question is displayed



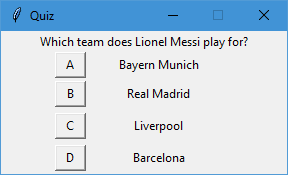
**Test 6:** Selecting Easy, then General Knowledge will bring up a General Knowledge question with 2 options

**Outcome:** User is presented with 2 options



**Test 6:** Selecting Medium, then General Knowledge will bring up a General Knowledge question with 3 options

**Outcome:** User is presented with 3 options



**Test 7:** Selecting Hard, then General Knowledge will bring up a General Knowledge question with 4 options

**Outcome:** User is presented with 4 options

# Section 4 (code is in section 3):

## Assess:

* Opens the file
* Assigns questions to a variable
* Closes file

## Design:

IF subject == physics THEN

DO f = OPEN Physics Questions.txt in read mode

ELIF subject == general knowledge THEN

DO f = OPEN General Knowledge Questions.txt in read mode

ELIF subject == maths THEN

DO f = OPEN Maths Questions.txt in read mode

ELIF subject == geography THEN

DO f = OPEN Geography Questions.txt in read mode

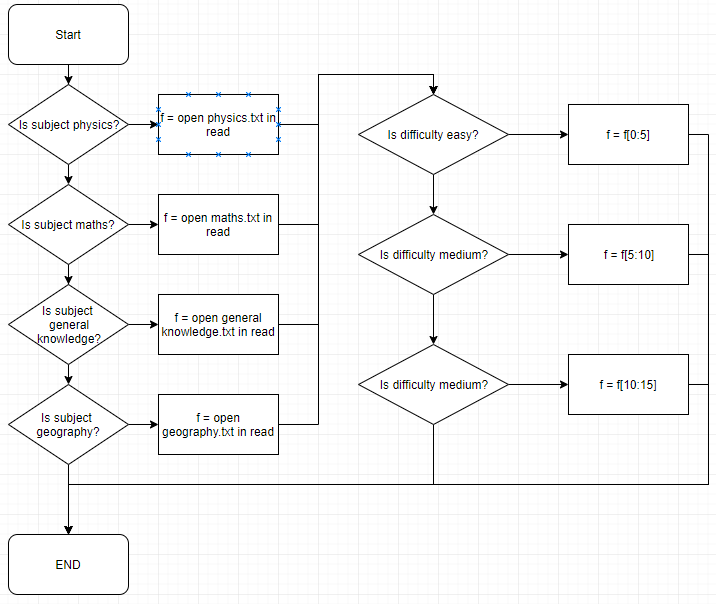
IF difficulty == easy THEN

DO f = f[0:5]

ELIF difficulty == medium THEN

DO f = f[5:10]

ELSE

 DO f = f[10:15]

## Develop:

Done in step 3

## Testing:

Questions are shown to be accessed in step 3 testing

# Section 5:

## Assess:

* Calculate score, percentage and grade
* Output to user
* Write these to the file “Global Statistics.txt”

## Design:

Percentage = score/5 \* 100

IF difficulty == easy:

IF percentage < 50%:

Grade = fail

ELSE:

Grade = pass

ELIF difficult == medium:

IF percentage < 40%:

Grade = fail

ELIF percentage <= 80%:

Grade = Pass

ELSE:

Grade = Merit

ELSE:

IF percentage == 0%:

Grade = fail

ELIF percentage == 20%:

Grade = Pass

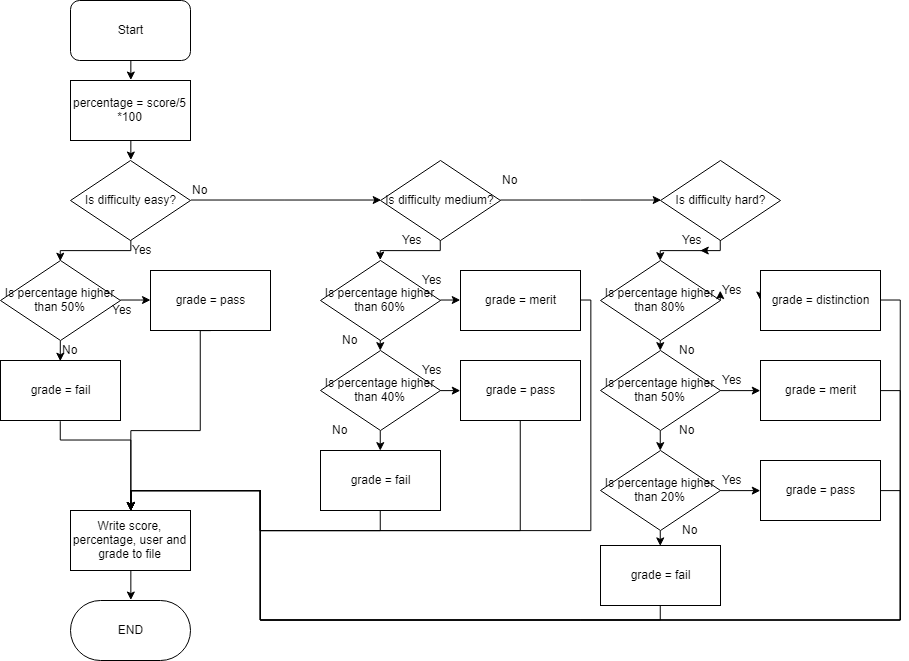
ELIF percentage < 80%:

Grade == Merit

ELSE:

Grade = Distinction

OUTPUT score, grade, percentage to user and file



# Section 6:

## Assess:

* Open Global Statistics file.
* Find user and return the results for all quizzes they have taken
* Find average and highest scores

## Develop:

Choice = INPUT Report for individual or everyone?

IF choice == individual THEN

DO Open Global Statistics.txt in read mode

Name = INPUT Whose results do you want?

Find name in Global Statistics

Print results

ELIF choice == everyone THEN

Do Open Global Statistics.txt in read mode

Subject = INPUT which subject?

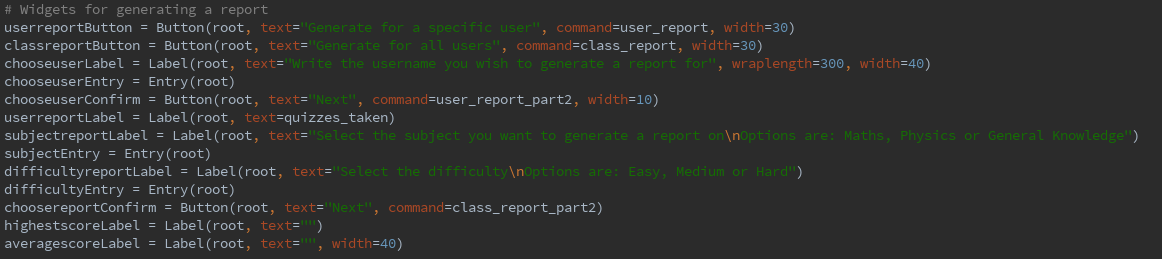
Level = INPUT Which difficulty?

Find scores for subject and difficulty

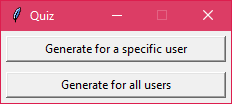
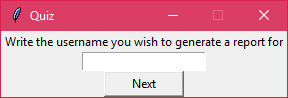
Print max(scores)

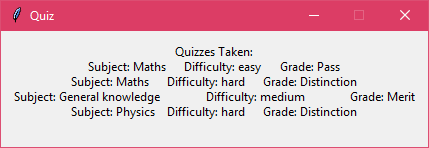
Print average(scores)

## Develop:





The “Generate Report” function removes the current widgets and displays the next options. If the user presses the option for 6a on the brief, it displays the entry field where the person enters the username of the person they want to generate a report for.

After the person enters the username and presses the next button, the final function is executed. It checks if the user has taken a quiz, if not then the entry is cleared, and the user is told that the user has not been found in the file. If the user is found, then it goes through each line and sees if it matches the user who the user has entered. It then is put into a string that looks tidy and is finally shown the user.

If the user wants to generate a report for the whole class, then the following function is executed:

