Computing Coursework

Analysis

1. Introduction

1.1 Client Identification

My Client is Adam McNicol, who is in charge of A-Level Computing at Long Road Sixth Form College. He has a very adept knowledge of computers and uses it for a variety of things ranging from basic to advanced. He performs most of these on an Apple MacBook but also uses a Dell Windows XP laptop when Windows applications on the network are necessary, such as registration.

The client finds the current system to be very manual and tedious in return for very little information, besides a way to store data. As Adam's status as a teacher and head of Computing, he will be in the main role of the program, as the software's administrator. Adam would also like the ability to be able to make his data secure easily, but still be able to access it, and more functions to be automated, such as being alerted about a particular student.

1.2 Define the current system

The current system is a semi-computerised system, which is a combination of Moodle, Excel and the client manually joining the two.

Currently, the client must first design their own spread sheet. This can vary from a quick few minute basic design to a several hour job, including the look and the inner workings of the sheet, such as formulas. Then they must copy all the names from the registration to the excel spread sheets, a process that may come up with many mistakes and takes a while in itself. Then each teacher must enter the names of every assignment into the spread sheet.

Every student will either complete and upload their homework to Moodle or do it directly on Moodle. They will then receive a mark for this homework on Moodle. This is then either automatically or manually compiled into a list of students and scores.

The teacher must then manually copy across all results from Moodle to the spread sheet each time a student does their homework to the excel spread sheet, where he matches each student with their scores.

The system is designed to calculate whether or not a pupil is at risk of failing the course. It is also designed to average the scores from each class and draw graphs.

1.3 Describe the problems
Most of this system is done manually. The teachers must first design their spread sheets.

This causes a discrepancy among different teachers' designs as it's down to personal choices. This may suit each teacher for their own sheet however when teachers want to look at each other's spread sheets, they must first ask the other teachers for the document and when they receive it, they may not know how it works exactly. For example, if a teacher has conditional formatting in use where it is colour coded. The creator knows what each colour means however unless they provide a key, the other teachers do not know and must either ask the creator or look up exactly what they have set it to be in the settings. Due to this, it means that both looking at others specifically and having an overall look at how the department is doing.

Different teachers also may have different thoughts about what they consider to be borderline between being successful and failing.

It's also difficult to monitor the progress of students. For example, It's very difficult to get excel to immediately tell you who is not currently on track without going to look for it, even if it is colour coded. It is also very difficult to make the software detect if a student is going wrong where it must detect two consecutive failings.

The client has also raised security concerns about the data that is stored. Currently, to secure the data the client must use a single password to prevent opening of the workbook. To share the data with others, he must share the password he has set, this requires him to choose a password that he can both remember easily and is willing to share with others. He has also expressed wishes to secure select parts of data. The only plausible way he could do this currently is to hide the sheet and password protect the editing. This is far from ideal as it requires the client to enter two passwords whenever he wishes to edit his sheet. It also means that he must put back in place the protection upon each view or edit of the restricted material if he saves the changes.

1.4 Section appendix
The client interview was conducted by emailing questions to the client who responded with his answers.

Overall
The proposed system is to monitor students' homework history. Can you go into further detail about this?

The system should be able to store the results of all students homework and tests (and comments if required) that are undertaken during AS and A2 Computing. It should be possible to see clearly how students are progressing – are they getting better or worse. The system should automatically tell me which students are at risk of failing the course and automatically place them on a list of students who should attend workshop time.

What is your current system and why is not preferable?
(If computer system, is it possible to have a copy or to try it?)

The current system is a mix of Excel spread sheets and Moodle logging of results. Neither is ideal as each teacher keeps their own records so the formatting is slightly different and it is not possible to check the progress of other groups without first getting the spread sheet from the teacher of that group. It is difficult to track progress as again the systems used vary from teacher to teacher. The Moodle logging is confusing as it is not possible to filter the results to only show those that are of interest at any particular time and in addition it does not give a clear overview of progress – rather it is by individual assignment.

What do you like about your current system?

Not a lot really. It does the job but it could be much better.

What would you like the program to do that you can't easily do in other programs such as Excel?

As above, it needs to be possible to keep track of all groups at all times.

Are there any constraints on hardware?

The college computers, plus it must work on my Mac.

Data and Processing

What data is currently being stored?

We store details on the students – GCSE average scores, learning needs etc. plus their results from each homework/test. We store brief details about each assignment as well.

Do you need to migrate any data from the current system? If so, what data and how much?

No migration is needed.

Beside the students' names and their progress, is there any other data that needs to be stored?

As above – details relating to learning needs, assignment details.

How much data will there be in total?

In general there are roughly 50 AS students and 30 A2 students each year. It should store data for at least 3 years before giving the option to detail. However, it may prove useful in terms of historical analysis to see how the department is improving if we can easily see the results from last year.

How often will this data need to be updated?

In terms of the basic details of students/assignments this will probably be once a year but data about homework etc. will be added every day.

Would you need or like the option to add or remove students?

Yes.

Will you need to make changes in batches or once or twice at a time?

Changes will happen as you make them.

How frequently will this need to be done?

As above really

What processes or functions are performed by the current system?

Averaging scores for class, cohort. Working out at risk students. Graphing results of assignment/group of assignments

What processes or functions are to be performed by the new system?

As above plus ability to see how individual students are performing in relation to other students of similar ability, match again previous years etc.

What inputs are currently used by the system?

We use data from Unit e to get student names etc. and we complete the assignment names from our scheme of work

Are any more inputs for the new system required?

I can't think of any at this time.

What outputs are currently used?

Generally it is numbers relating to average scores in the group, estimated grades based on current progress etc.

Any new outputs for the new system?

As above.

How often will outputs be required?

All outputs should be available on demand

Are any hard copies required?

It would be good to be able to print out reports based on the data – progress report on individual student for example.

How often will hard copies be required?

On demand.

Is the new software to run from the shared server on the network or from the local computer?

This is a difficult one, best case scenario would be a central database that all teachers can access via a front end that runs on their machines.

Should the data be stored alongside the software or in your documents? Anywhere else?

Don't understand the question.

<u>Features</u>
"Finished" and "Waiting" are two options. Would you like another option for mid-way?

Again, don't understand the question?

Would you like to call those options anything else?

?

Would you like to be able to add Notes to each student and/or each of their homework assignments?

Yes.

Would you like E-mail to be a part of the program? If so, how?

Yes, to e-mail teachers lists of students at risk, to e-mail students grades/marks etc.

__Security__
Is security of the data an issue?

Yes, it should only be accessible by the computing department staff and Sue Bridgeland (head of department)

Is the logon just for ease of use, as well as editing, or is it to prevent your data being read?

Editing of some data should require an additional password – student details for instance

Would you like your password stored as plain text or hashed?
If they are stored plain text they are not very secure

Would you like the ability to reset your password?

Yes.

If so, how would you like to reset your passwords? For example, would you like to write your own question and answer it? Email reminders?

Either is fine.

__Errors__
Would you like backups to be made automatically? If so, how often and where to?

Yes, a backup every week to another location would be a good idea.

How are errors and exceptions currently handled?

Not much all for that in a spread sheet

How would you like them to be handled in the new system?

Clearly the system should deal with them gracefully.

__Follow up questions__
The follow up questions do not cover the questions that the client could not answer because the client answered them elsewhere.

How do you transfer the students' names from Unit E to the spread sheet?

Manually

Do you get all your student information, e.g. Special needs, from Unit E or does it come from alternative sources?

From unit E

How do you calculate predicted grades?

The original predicted grades are based on average GCSE scores and other factors but they are generated by Unit E. The predicted grade is altered based on how well students perform in homework and mock exams over the course of the year.

Is it possible to export data from both Unit E and Moodle? Or maybe even copy and paste in a useable fashion?

Possibly from Moodle

Should the data be stored in one large file or split into multiple files?

Don’t really care as long as it works

Should there be a separate data file for each tutor or should it be all in one?

The course team leader should be able to see all of the data

When marking off a piece of homework, do you apply a percentage or a mark out of a total?

Both

Should a Tutor have write access to other Tutors?

As above, CTL should have access to everything, others to their own.

Should there be one overall "Administrator" user who has the ability to edit everything (such as resetting passwords as a last resort) or should all teachers be equal?

yes there should be an administrator

Do different teachers have different assignments or do all teachers follow the same assignments?

generally the same, if there is time build the ability to have different assignments

Will they have different start and end dates for assignments?

As above really

Would you like to export the data from Moodle more automatically and thus import it into the program more automatically?

if possible but it is not the priority

What's the threshold of the warning? How exactly is it calculated (i.e. an overall average)?

If the student has missed two assignments in a row or that they performed below TMG two assignments in a row.

What graphs does it currently draw and should it draw?

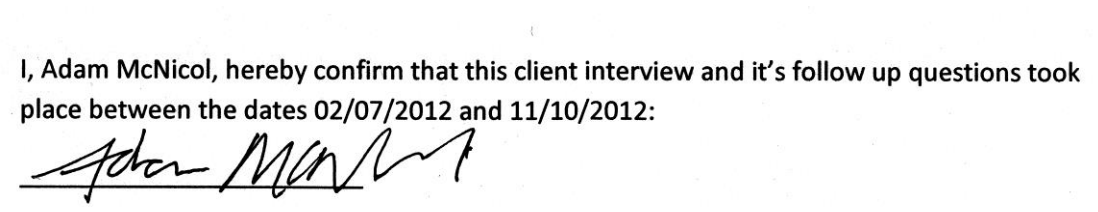
Graphs are drawn on an ad-hoc basis currently. All data reports should have visualisations

What problems arise from the current system?

We talked about this - difficult to monitor student progress, hard to determine who is working below target. Difficult to get an overall picture of the department

Could you please provide me with any forms or reports that are generated by the current process? Such as an assessment sheet.

There are no forms that I can provide.


I, Adam McNicol, hereby confirm that this client interview and it's follow up questions took place between the dates 02/07/2012 and 11/10/2012:

[illegible]

2. Investigation

2.1 The Current System

2.1.1 Data Sources and destinations

In the current system, there are 4 data sources used.

The Scheme of Work is the first data source used, which is where the client gets the assignments that need to be added to the workbook. This is a one-off usage and will not need to be used after initial setup. It is possible, however, that if the course changes, it may need to be altered after a year.

Unit E is another "initial setup" source. This is where the names of the students and additional information, such as GCSE results and learning needs, come from. The client then transfers these to the workbook, like the assignment names.

Moodle is one of the on-going sources of the data. Moodle handles the homework that gets submitted and provides details about this. This includes the student names, their scores, and their time on an assignment by assignment basis upon request. The client will then insert the data in the correct place.

The student could also be considered a source as they give their homework to either the client or Moodle, where it is then marked.

The client themself is the final data source. This is because while he collects all the data, the program collects all its data from him. This also means that the client is also a data destination for the other data sources. He also adds any further details about the students that may not be on the register.

The workbook is also a source as it gives statistics such as Predicted Grades.

Source	*Data*	*Example Data*	*Destination*
Scheme of Work	# of Task, Name of task, short description	1, "Read Section 1", "Vocab – Pages 3-6"	Client
Client	# of Task, Name of task, short description	1, "Read Section 1", "Vocab – Pages 3-6"	Workbook
Unit E	Student Names, GCSE scores, learning needs	Greg Davis,"1 A 3 Bs, 2 Cs", "Dyslexia"	Client
Client	Student details: GCSE scores, learning needs	"1 A 3 Bs, 2 Cs", "Dyslexia"	Workbook
Student	Homework	Answers to be	Client or Moodle

		marked.	
Client OR Student	Scores	10	Moodle
Moodle	Student Names, Scores	Greg, 10	Client
Client	Scores	10	Workbook
Workbook	Statistics	Predicted a "C"	Client

2.1.2 Algorithms

Currently, the system has very few algorithms as it is generally used as a data store as opposed to a program that processes data a lot. One of these a list of figures to provide an average. An example of this could be in the form of:

```
FUNCTION Average(list: Array)
        Average: Float
        Average ← CALL SUM(list) / CALL LEN(list)
        RETURN Average
END FUNCTION
```

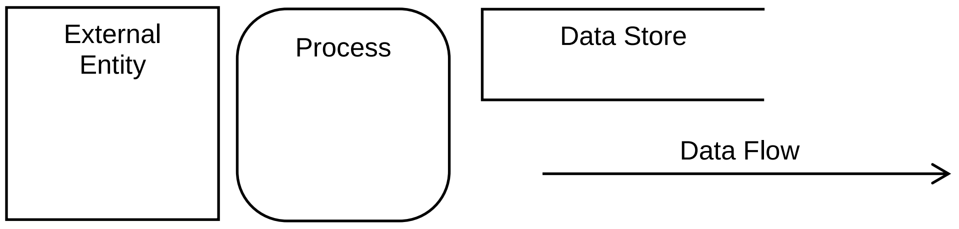
Another algorithm is taking a table of results from Moodle and adding it to the workbook. Currently, this is not computerised however below is what it may be in Pseudo Code.

The "list" variable is the output that Moodle gives. The below example is my presumption of what it looks like. I am using it as a 2D array.

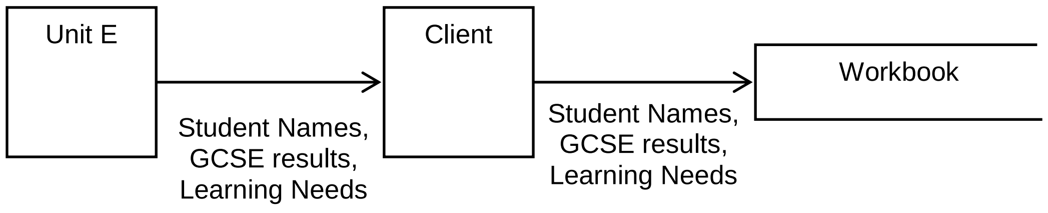
Student 1	0.23
Student 2	0.33
Student 3	0.67
Student 4	0.84
Student 5	0.65

```
FUNCTION UpdateAssignment(students: Array, list: Array, assignmentNumber: Integer)
        FOR EachSource ←1 1 TO (CALL LEN(students)) DO
                For EachDestination ← 1 TO (CALL LEN(students)) DO:
                        IF list[EachSource][0] == students[EachDestination].Name() THEN

        Students[EachDestination].Assignments[assignmentNumber] , ←list[EachSource][1]
```

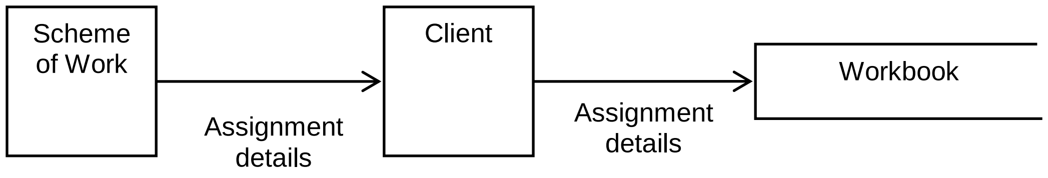
2.1.3 Data Flow Diagram
Key:



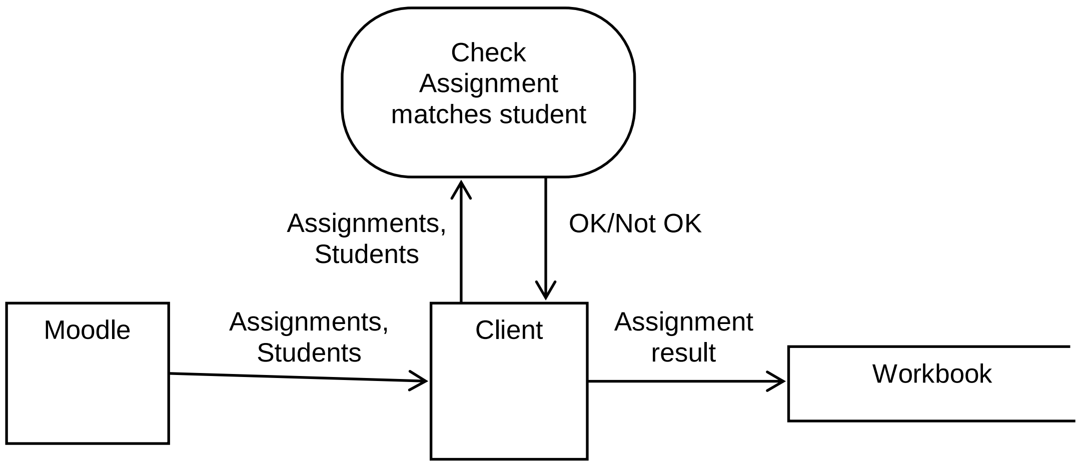
Adding a new student:


Unit E
Student Names, GCSE results, Learning Needs
Client
Student Names, GCSE results, Learning Needs
Workbook

Adding the assignments



Updating the assignments

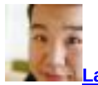


2.1.4 Input Forms, Output Forms, Report Formats
Currently, there are only a number of input forms. One for the student to either upload their file or complete a quiz and another for the teacher to be able to edit the changes. This is because all information is currently given by other sources that are not gained through forms.

Reports are currently given in the format of averages and graphs. As this is in Excel, any of these can be printed off immediately when required

Moodle gives a report of the students “grades” in a table format. This can then be used to copy across details to another place without manually looking things up. An example of this can be seen below.

Grader report

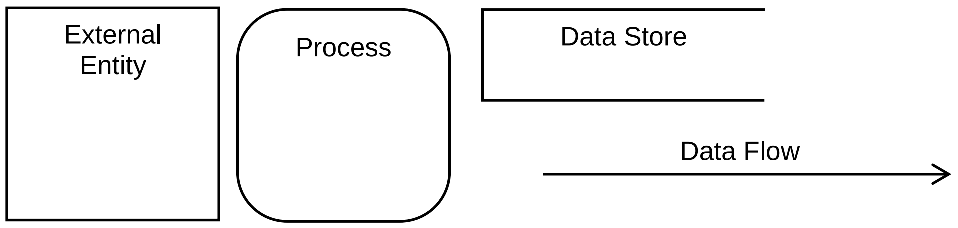
			World of Water												
Surname ↑ First name		Email address	30 Days Challenge	Find and rate the most ..	Aqua Quiz	Create a Junior School ...	Frequency	Create a Junior School ...	30 Days Challenge	Bottled Water? Thoughts?	Content	Presentation	Research skills	Write a Water Poem	Course total
 Lao Cai	▦	laocai154@example.com	-	-	-	-	-	-	-	-	-	-	-	-	-
Barbara Gardner	▦	barbaragardner249@example.com	-	-	7.25	59.00	-	20.00	-	7.00	Often	Sometimes	Sometimes	-	77.71
Charles Gardner	▦	charlesgardner223@example.com	-	-	4.75	-	-	-	-	-	-	-	-	-	47.50
Overall average			-	-	6.69	45.44	-	20.00	-	7.00	Often	Sometimes	Sometimes	-	63.33

2.2 The Proposed system

2.2.1 Data Sources and Destinations

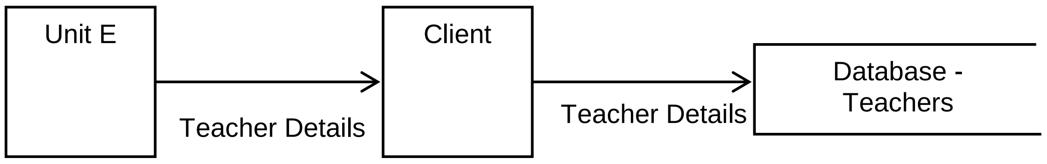
Source	Data	Data Type	Destination
Students Database			
Unit E via Client	Surname	String	Database – Students
Unit E via Client	Forename	String	Database – Students
Unit E via Client	DOB	Date	Database – Students
Unit E via Client	E-mail	String	Database – Students
Unit E via Client	Scribe	Boolean	Database – Students
Unit E via Client	Computer	Boolean	Database – Students
Unit E via Client	25Extra	Boolean	Database – Students
Unit E via Client	50Extra	Boolean	Database – Students
Unit E via Client	GCSEResults	Float	Database - Students
Unit E via Client	AssignmentsResults	Array	Database – Students
Teachers Database			
Client	Name	String	Database – Teachers
Client	Username	String	Database – Teachers
Client	Password	String	Database – Teachers
Client	Email	String	Database – Teachers
Client	ResetQuestion	String	Database – Teachers
Client	ResetAnswer	String	Database – Teachers
Assignments Database			
Scheme of work via Client	Assignment.Name	String	Database – Assignments
Scheme of work via Client	Assignment.Description	String	Database – Assignments
Scheme of work via Client	Assignment.Deadline	Date	Database - Assignments

2.2.2 Data Flow Diagram



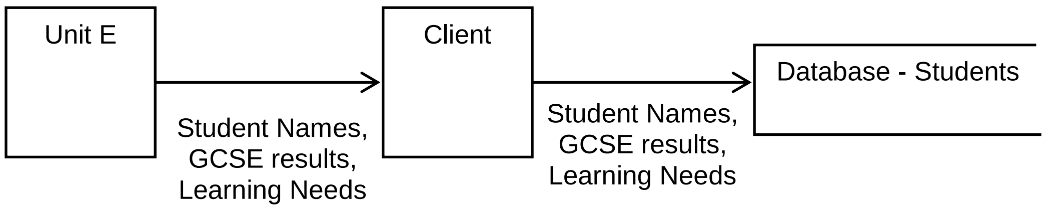
The data flow diagrams remain simple as the new proposed system is not mainly designed to process more data, but rather to process the current data in more ways.

Adding a member of staff

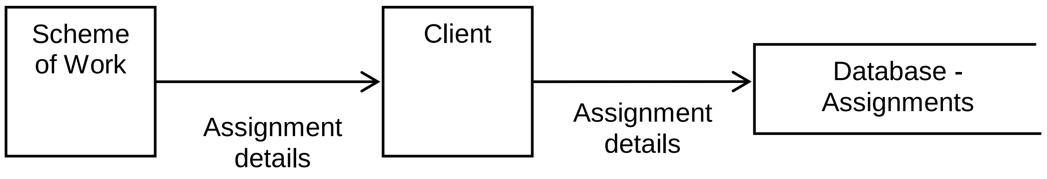


These next 3 tasks are equivalent to the tasks on the current system, except these now go into dedicated places.

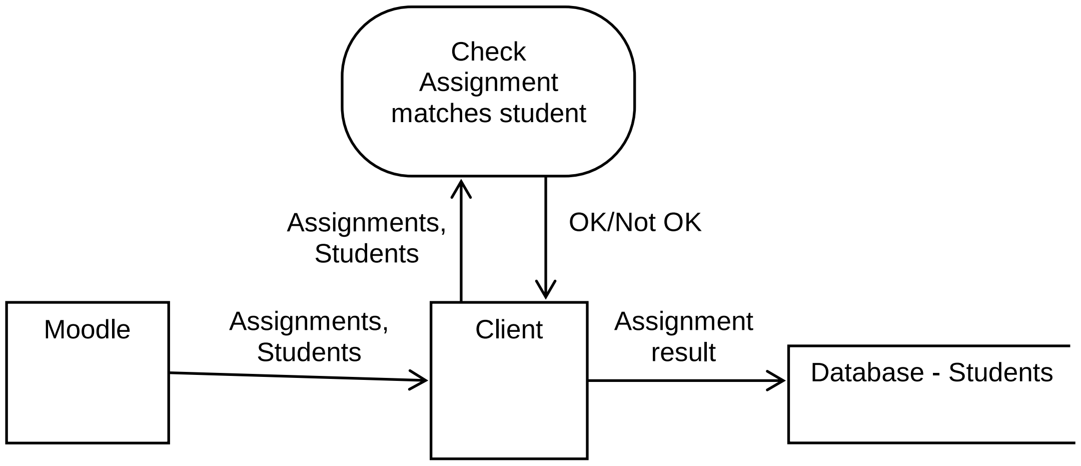
Adding a new student:


Unit E
Student Names, GCSE results, Learning Needs
Client
Student Names, GCSE results, Learning Needs
Database - Students

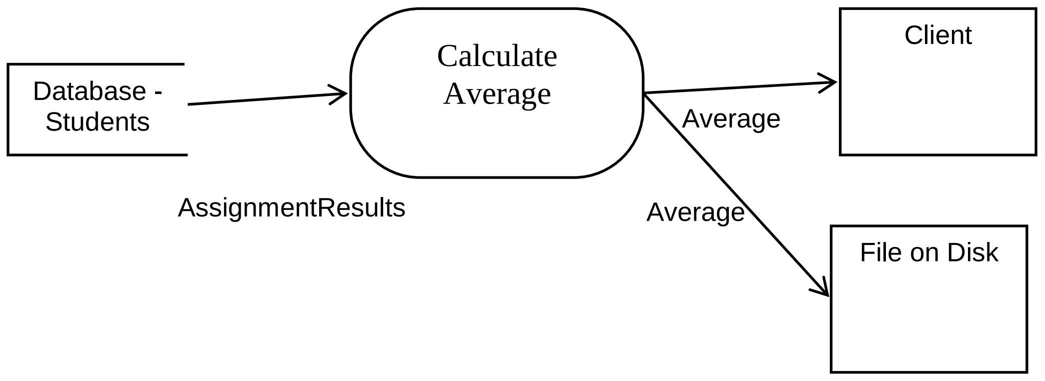
Adding the assignments



Updating the assignments

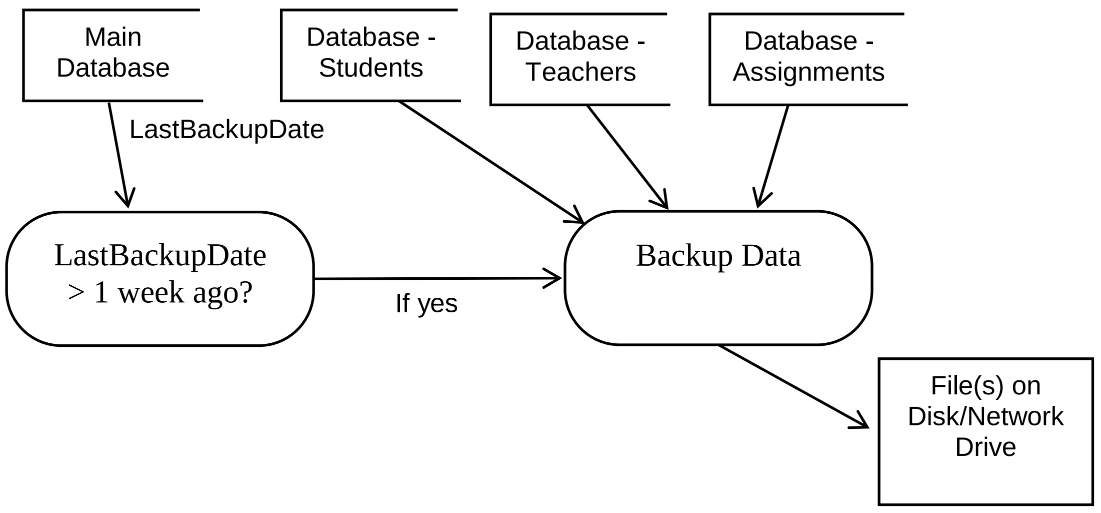


Statistics play a major role in the proposed system. The majority of these are similar to the one for averaging below.



Statistics are currently not planned to be exported to disk, as they are given to the client as and when, who can save them if he so wishes. This may change in future so I have added that option. It is an external entity because it a one way process and therefore not considered to be a data store.

Backups are another option that the client requested.



2.2.3 Data Dictionary

Name	Data Type	Length	Validation	Example Data	Comment
TeacherID	Integer	0-255		42	Automatically created so no validation
TeacherUserName	String	Up to 20	Length, More than 3 characters	“Bmanger”	
TeacherPassword	String	32	Length	“098f6bcd4621d373cade4e832627b4f6”	Encypted using MD5. Checked before encrypting and storing. Actual password may be up to 64 characters.
TeacherAdmin	Boolean	1	Presence check	True	Defines if the teacher has admin privs.
TeacherAdditionalPassword	String	32	Length	“098f6bcd4621d373cade4e832627b4f6”	Encrypted using MD5. Checked before encrypting and storing. Actual password may be up to 64 characters. For very restricted areas.
TeacherName	String	Up to 32	Length/Presence check	“Billis Manger”	Presence check includes space in

					the middle.
TeacherEmail	String	Up to 128	Length/Presence check	Bmanger@longroad.ac.uk	Include check for @ and domain. My limit down to one domain.
TeacherQuestion	String	Up to 512	Length	“Where did you buy your first car from?”	
TeacherAnswer	String	Up to 32	Length	“Bucks and Dos”	Capitalisation will not matter when comparing.
TeacherLastEmailed	Date		Format	20/11/2013	To prevent teachers being over emailed about problems.
StudentID	Integer	0-4096		23	Automatically created so no validation.
StudentSurname	String	Up to 32	Length	“Davies”	
StudentForename	String	Up to 32	Length	“Greg”	
StudentDOB	Date		Format	22/04/1995	
StudentEMail	String	Up to 128	Length/Presence Check	64634@longroad.ac.uk	Include check for @ and domain. May limit down to one domain.
StudentScribe	Boolean		Presence Check	True	
Student25Extra	Boolean		Presence Check	True	

Student50Extra	Boolean		Presence Check	True	
StudentGCSEResults	Float	0-10	Range	7.66	
StudentLastEmailed	Date		Format	20/11/2013	To prevent students being over emailed about problems.
StudentNotes	String	Up to 1024			Not required for program to work and optional so no validation.
AssignmentID	Integer	0-255			Automatically created so no validation.
AssignmentName	String	Up to 32	Length	“Reading on Booleans”	
AssignmentDescription	String	Up to1024	Length	“AS Computing text book – Pages 2-7”	
AssignmentDeadline	Date		Format	01/11/2013	
AssignmentMaxMarks	Integer	Up to 256	Format/Length	20	Defines max marks for the assignment
AssignmentMark	Integer	Up to 256	Format/Length	50	
AssignmentNotes	String	Up to 1024	Length	“Notes”	
SMTPHost	String	Up to 32	Length/presence check	“smtp.gmail.com”	Check of valid domain.
SMTPUsername	String	Up to 32	Length/Presence check	“Bmanger’	Check of characters

SMTPPassword	String	Up to 32	Length	“cravat332”	Needs to be encrypted but it needs to also be easily decrypted by the program (and the program alone)
LastBackedUp	Date			14/03/2012	Automatically used by program, no validation required.

2.2.4 Volumetrics
It is assumed that the program will be around 2mb, dates take up around 80 bytes, all data is stored in Unicode UTF-8 and there are around 50 assignments in total. These are maximum estimates. Some rounded has occurred, these are rounded up.

Name	Data Type	Length	Max Space used
TeacherID	Integer	0-255	2 bytes
TeacherUserName	String	Up to 20	40 bytes
TeacherPassword	String	Up to 32	64 bytes
TeacherAdmin	Boolean	1	1 bit
TeacherAdditionalPassword	String	Up to 32	64 bytes
TeacherName	String	Up to 32	64 bytes
TeacherEmail	String	Up to 128	256 bytes
TeacherQuestion	String	Up to 512	1 megabyte
TeacherAnswer	String	Up to 32	64 bytes
TeacherLastEmailed	Date		80 bytes
Total for 1 Teacher			1658 bytes
Total for 5 Teachers			8295 bytes
StudentID	Integer	0-255	2 bytes
StudentSurname	String	Up to 32	64 bytes
StudentForename	String	Up to 32	64 bytes
StudentDOB	Date		80 bytes
StudentEMail	String	Up to 128	256 bytes
StudentScribe	Boolean		1 bit
Student25Extra	Boolean		1 bit
Student50Extra	Boolean		1 bit
StudentGCSEResults	Float	0-10	Assumed at 2 bytes
StudentLastEmailed	Date		80 bytes
StudentNotes	String	Up to 1024	2 mb
Total for 1 student			Without notes: 549 bytes With notes: 2596

Total for 80 students			Without: 48mb With: 123mb
AssignmentID	Integer	0-255	1 byte
AssignmentName	String	Up to 32	64 bytes
AssignmentDescription	String	Up to1024	2 mb
AssignmentDeadline	Date		80 bytes
AssignmentMaxMarks	Integer	Up to 256	8 bytes.
1 Assignment			2201
50			108 mb
AssignmentMark	Integer	Up to 256	8 bytes.
AssignmentNotes	String	Up to 1024	2 mb
For 50 assignments with 80 students			Min 31.25 mb Max: 8032 mb
SMTPHost	String	Up to 32	64 bytes
SMTPUsername	String	Up to 32	64 bytes
SMTPPassword	String	Up to 32	64 bytes
LastBackedUp	Date		80 bytes
Total Miscs			272 bytes

This table shows that for a year's worth of data, it should take up to around 195mb without notes. With notes would raise it to around 8163mb. It is worth noting that this is a worst possible scenario as the likelihood of all the notes being used is very small. It is also worth noting that if the characters are stored in ASCII, the space used could be reduced by almost half.

3. Objectives

3.1 General Objectives

- A secured environment for viewing and editing the data
- Easy to use interface that is uniform across teachers
- Be alerted when a student falls below acceptable parameters
- Display statistics

3.2 Specific Objectives

- The user should be able to create and delete teacher accounts from a main account
- The user should be able to add and delete students

- The user should be able to add, edit and delete assignments
- The user should be able to edit students details
- The user should be able to edit the assignment figures
- The user should be able to find details out about a specific student.
- The user should be able to secure all areas to prevent unauthorised access.
- The user should be able to input data via a GUI.
- The user should be able to view statistics on all the data.
- The user should be able to be alerted on pupils who are failing
- The system should be encrypted
- The system should back itself up.
- The user should be able to have a reset email
- The user should be able to reset others' passwords
- Ability to send emails to students
- A central database on a shared server with a GUI on each computer
- Statistics over the past few years showing how it has changed
- Printable reports on each student available on demand

3.3 Core Objectives

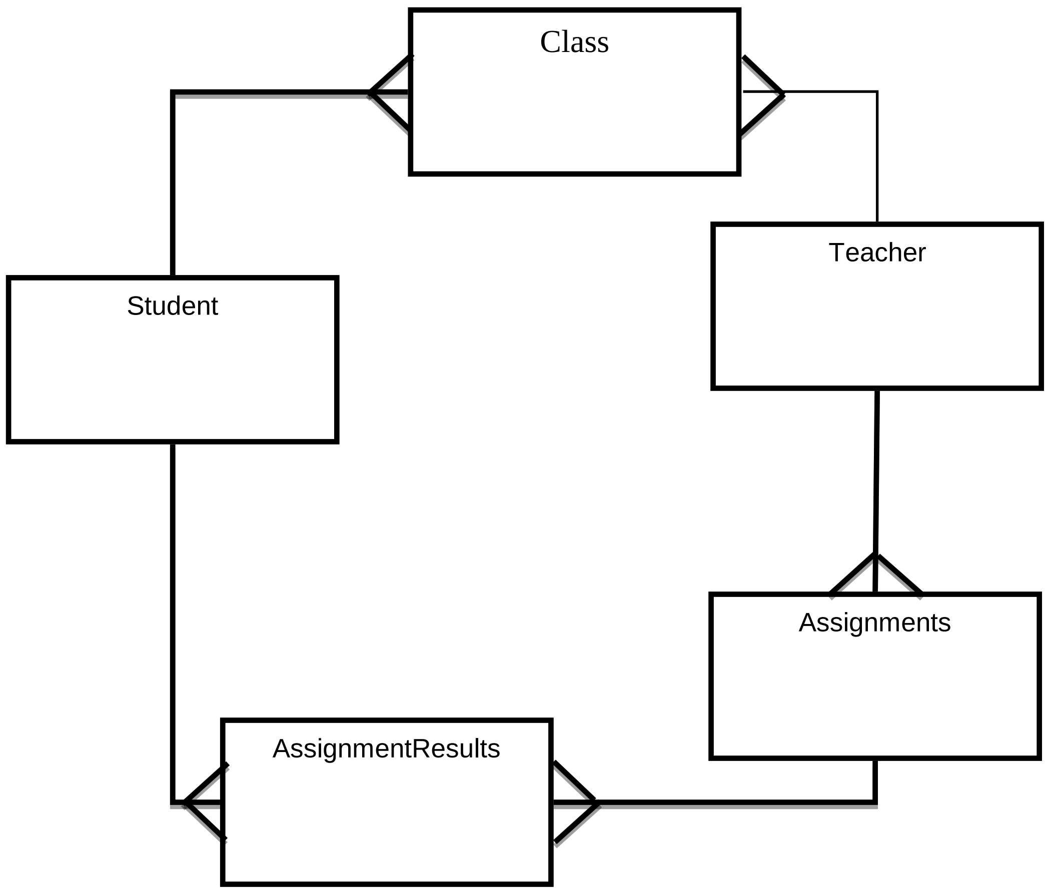
- The user should be able to create and delete teacher accounts from a main account
- The user should be able to add and delete students
- The user should be able to add, edit and delete assignments
- The user should be able to edit the assignment figures
- The user should be able to secure all areas to prevent unauthorised access.
- The user should be able to input data via a GUI.

3.4 Other Objectives

- The user should be able to edit students details
- The user should be able to find details out about a specific student.
- The user should be able to find details out about a specific student.
- The user should be able to view statistics on all the data.
- The user should be able to be alerted on pupils who are failing
- The system should be encrypted
- The system should back itself up.
- The user should be able to have a reset email
- The user should be able to reset others' passwords
- Ability to send emails to students
- A central database on a shared server with a GUI on each computer
- Statistics over the past few years showing how it has changed
- Printable reports on each student available on demand
-

4. E-R Diagrams and Descriptions

4.1 E-R Diagram



4.2 Entity Descriptions

Teacher(<u>TeacherID</u>, TeacherUserName, TeacherPassword, TeacherAdditionalPassword, TeacherFirstName, TeacherLastName, TeacherEmail, TeacherQuestion, TeacherAnswer, TeacherLastEmailed)

Student(<u>StudentID</u>,StudentSurname, StudentForename, StudentDOB, StudentEMail, StudentScribe, Student25Extra, Student50Extra, StudentGCSEResults, StudentLastEmailed, StudentNotes)

Class(<u>*TeacherID, StudentID*, Year</u>, YearStart)

Assignment(<u>AssignmentID</u>, AssignmentName, AssignmentDescription, AssignmentDeadline, MaxMarks)

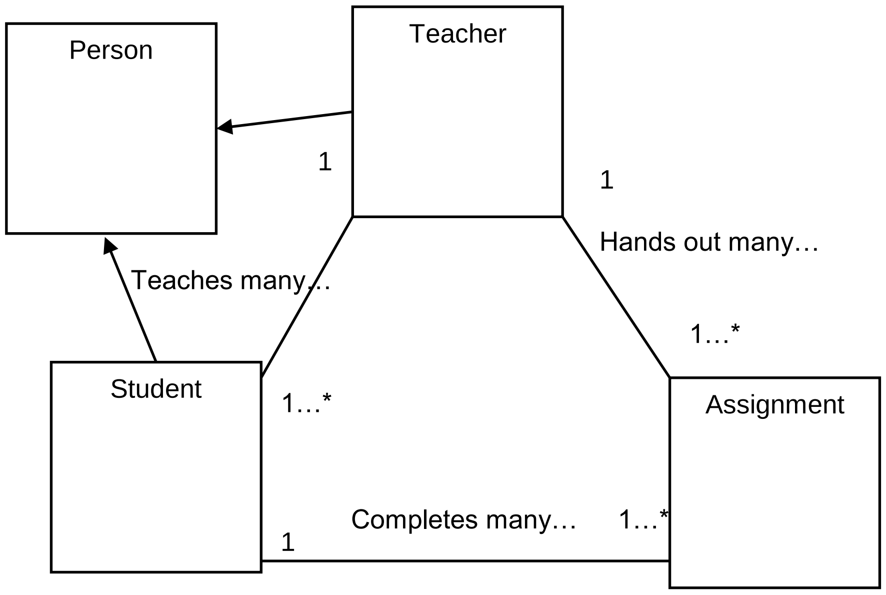
AssignmentResults(<u>*StudentID, AssignmentID*</u>, AssignmentMark, AssignmentNotes)

5. Object Analysis

5.1 Object Listing

- Person
    - Students
    - Teachers
- Assignments

5.2 Relationship Diagrams



5.3 Class Definitions

Person
Id FirstName LastName Email LastEmailed
getID getFirstName setFirstName getLastName setLastName getEmail setEmail getLastEmailed setLastEmailed

Teacher
Username Password

Admin
ResetQuestion
ResetAnswer
getAdmin
setAdmin
setUsername
getUsername
setPassword
checkPasswordMatch
SetResetQuestion
GetResetQuestion
SetResetAnswer
CheckResetAnswerMatch

Student
DOB
Scribe
25Extra
50Extra
GCSEResults
AssignmentResults
getDOB
setDOB
getScribe
setScribe
get25Extra
set25Extra
get50Extra
set50Extra
getGCSEResults
setGCSEResults
getAssignmentResults

setAssignmentResults

Assignment
ID Name Description Deadline
getID getName setName getDescription setDescription getDeadline setDeadline

6. Other Abstractions

6.1 Graphs

7. Constraints

7.1 Hardware

The system must work on the College's systems. These are all low-end Dells connected to the network; however this should not be of great impact. I would have to consider the screen resolution when designing the GUI however. This is mainly because my main client's MacBook is a laptop and as such, screen resolution isn't very high on it. I will also need to consider it because not all the computers have a high screen resolution where some have extremely low resolutions anyway.

7.2 Software

All the computers used at College are Windows XP and above and networked and have Python 3.2 already installed so this should be perfect.

The client has also specified that this should work on his Mac. While on the outside this may not cause a problem, this may cause a problem with directories. This is due to his Mac being a personal one and therefore he does not have instant easy access to the college network.

Possible solutions for this may include the client attempting to use Samba on his Mac to access the shared folder or synchronising his file manually with the network.

It also may be cause for concern if I choose to use absolute directories where the GUI is stored locally with the network files stored on the network. This is due to Windows computers using paths such as *C:/Users/Client/Documents* and Macs using paths such as */Mac HD/Documents/.* This should be solvable by detecting the platform and using alternate paths for each.

7.3 Time
There are no deadlines set by the client so I will work to the project deadlines, which is Tuesday, 18 February 2014.

7.4 User knowledge
While the client and the main target users are Computing teachers and therefore proficient in the use of computers, the client has expressed that the head of department, Sue Bridgeland, should also have access. She is not a Computing teacher and judging from personal experience, she is not adept at Computers. This means I must make the UI as clear and concise as possible to avoid easy confusion and as such, it will affect the Computing teachers UIs as they will use the same one.

I think some larger buttons and a bit of clear indication, possibly through the use of colours, should help here.

7.5 Access Restrictions
The client has expressed this as one of the top concerns. The program must only be accessible to the client and his immediate colleagues through the use of a username and password each. They must also enter a further password to enter further restricted zones.

8. Limitations

8.1 Areas which will not be included in computerisation
The program is unable to import data directly from any of its sources so the client must do this manually. As I have no control over these applications, I am unable to include any method in those to allow this to be done.

Marking would be a good area to be able to include in this program but Moodle already has it and is accessable.

8.2 Areas considered for future computerisation
- Additional statistics
- More tools to use
    - Such as batch editing

9. Solutions

9.1 Alternative Solutions
If the client had no solution from square one and did not express a wish to not use spread sheets, then I would think have given that some thought. Other solutions are included.

Solution	Advantages	Disadvantages
Shared Spread sheet	• Simple and clear to use • Almost everyone has software for it • All in one place • Ability to do a lot with it • Can be backed up easily	• Difficult to setup or modify without knowledge of program • Due to a lot of functions, it looks complex • Algorithms will look complex as they must all be in one cell, or spread across many • Without use of complex and tedious methods, security is impossible. • Only one can edit it at any one time • If copied, can bring up

		discrepancies.
Spread sheet on Google Docs	In addition to above • Ability to secure it via username and password each • Can be edited by more than one person at once. • Backups automatically	In addition to above • Many do not feel comfortable with information like this being "in the cloud". • Less functionality • Requires internet access
Web application	• Can be accessed anywhere with internet • No installation of software needed • Clear interface	• It can be accessed anywhere with internet – meaning anyone can attempt to gain access • Requires knowledge I do not have • Requires messing about with hostnames and ports – something that is almost impossible at a college • If it went down, it would not be accessible until it could be restored • Requires an internet connection • Not as easy to create statistics • Not as simple to backup
Command-line (with no GUI)	• More simple to make without needing to worry about GUIs	• Intense training and documentation will need to be produced • Impossible to view all the data clearly • Still requires the same software to be installed •

9.2 Justification of chosen solution
I have chosen to do a Python application with a PyQT GUI.

- Access restrictions are easier to control, it means physical access to the network is a must
- The application would be a bespoke application designed for the client
- Backs can be made easily via both copy and paste and automatically
- I am familiar with the chosen language

Design

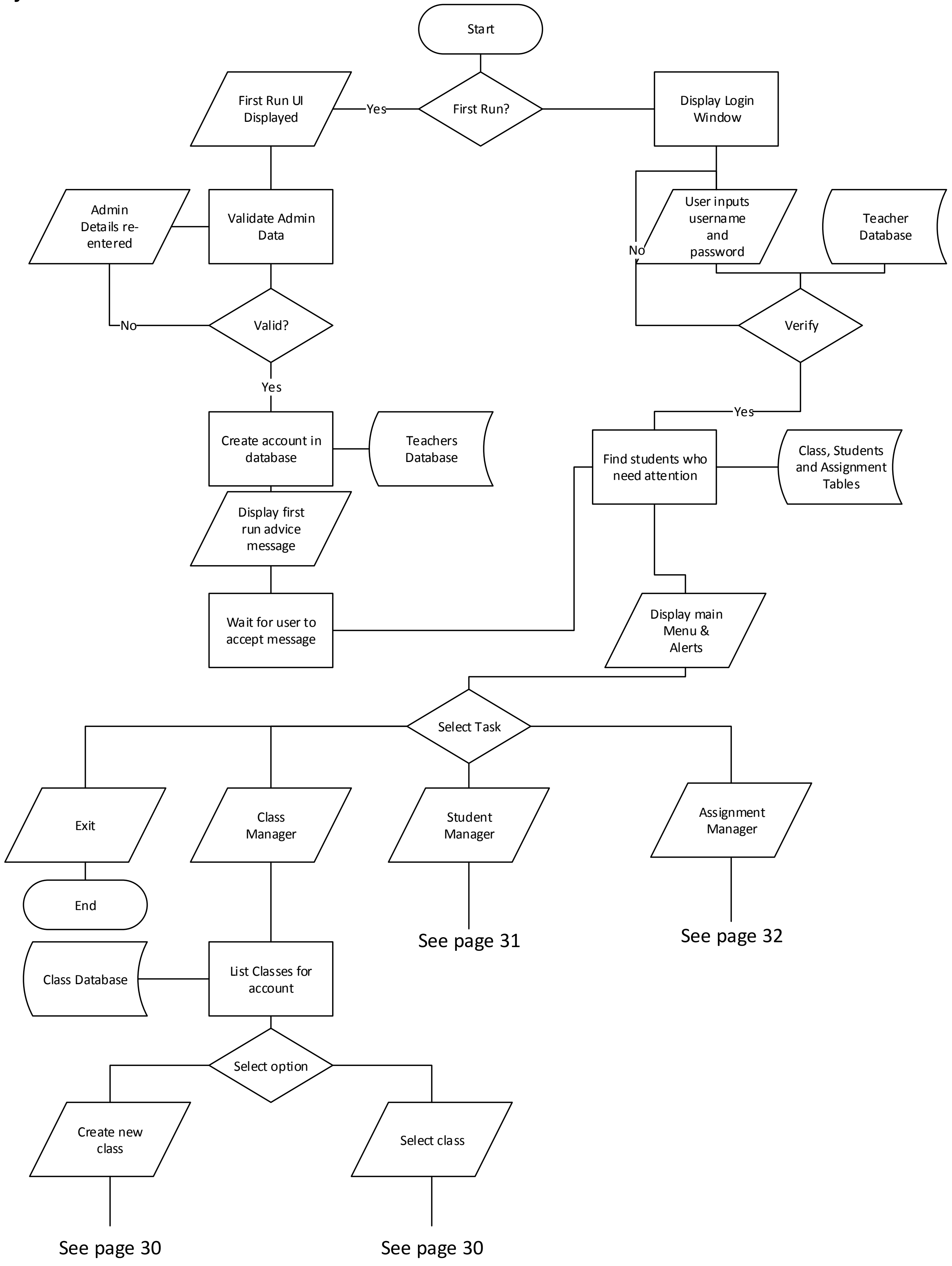
1. Overall System Design

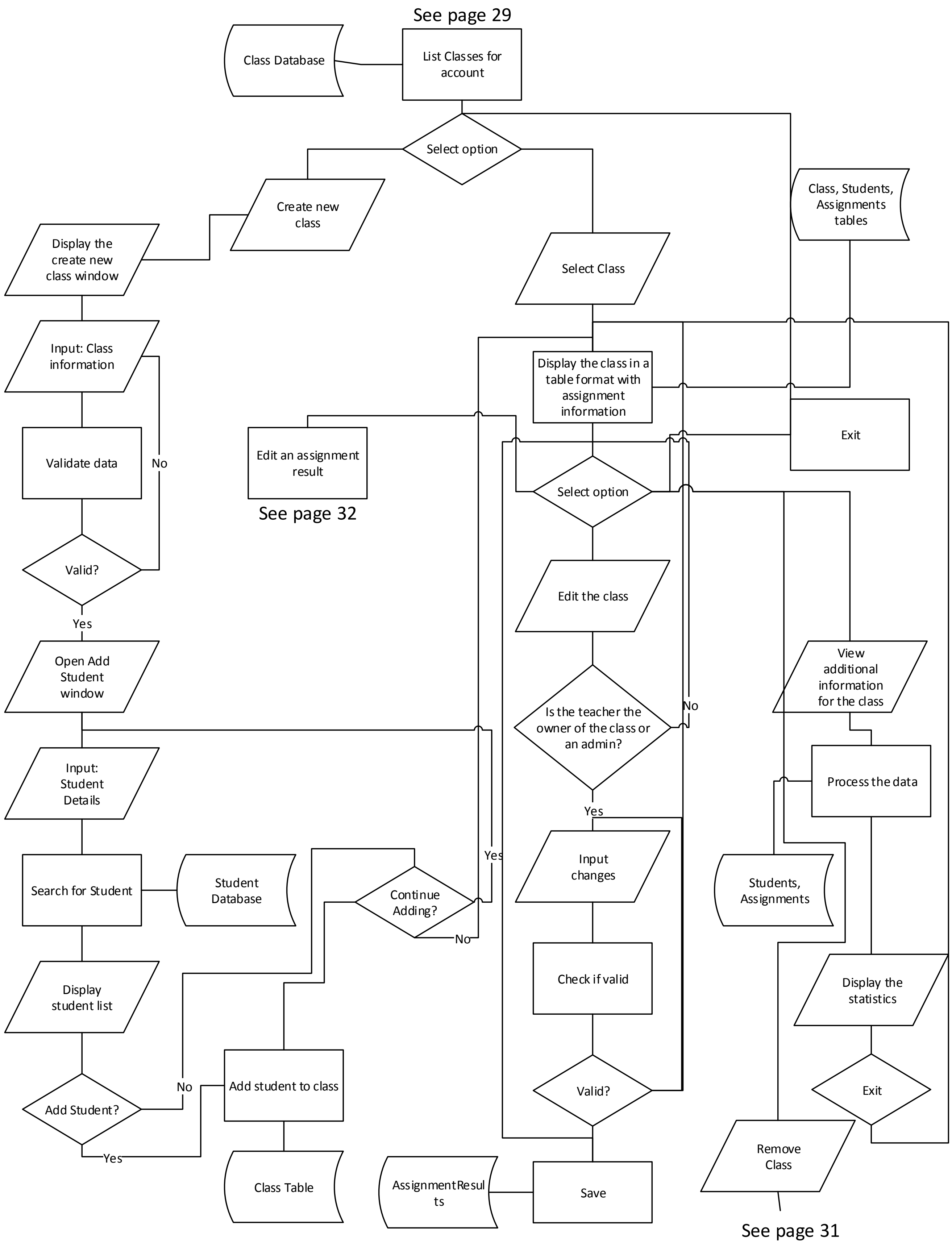
1.1 Short description of the main parts of the system

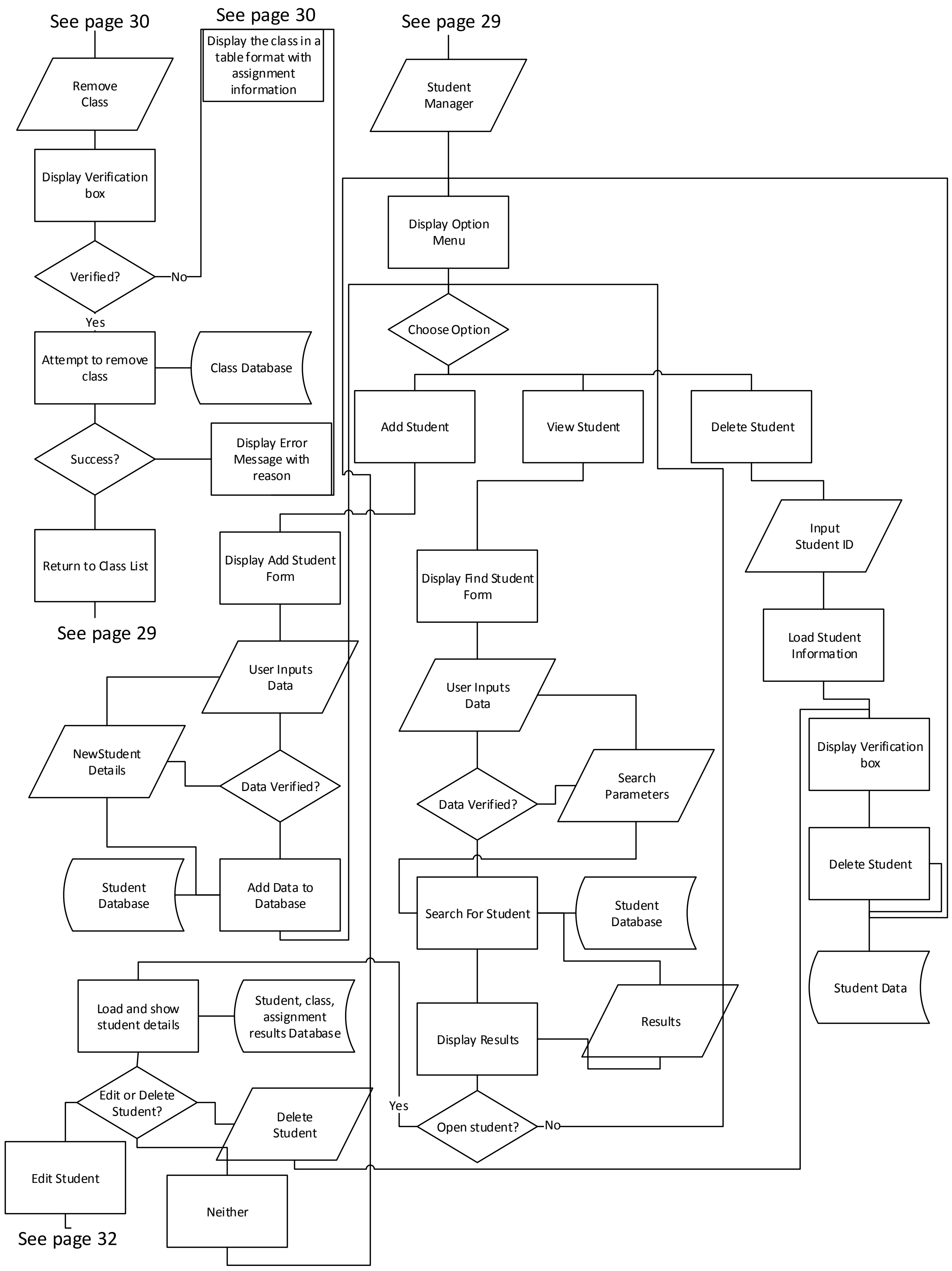
- Student Assignment Management system
    - First run
    - Administration
    - Managing students
    - Managing students’ assignments
    - Reporting details
    - Program maintenance
    - User interface

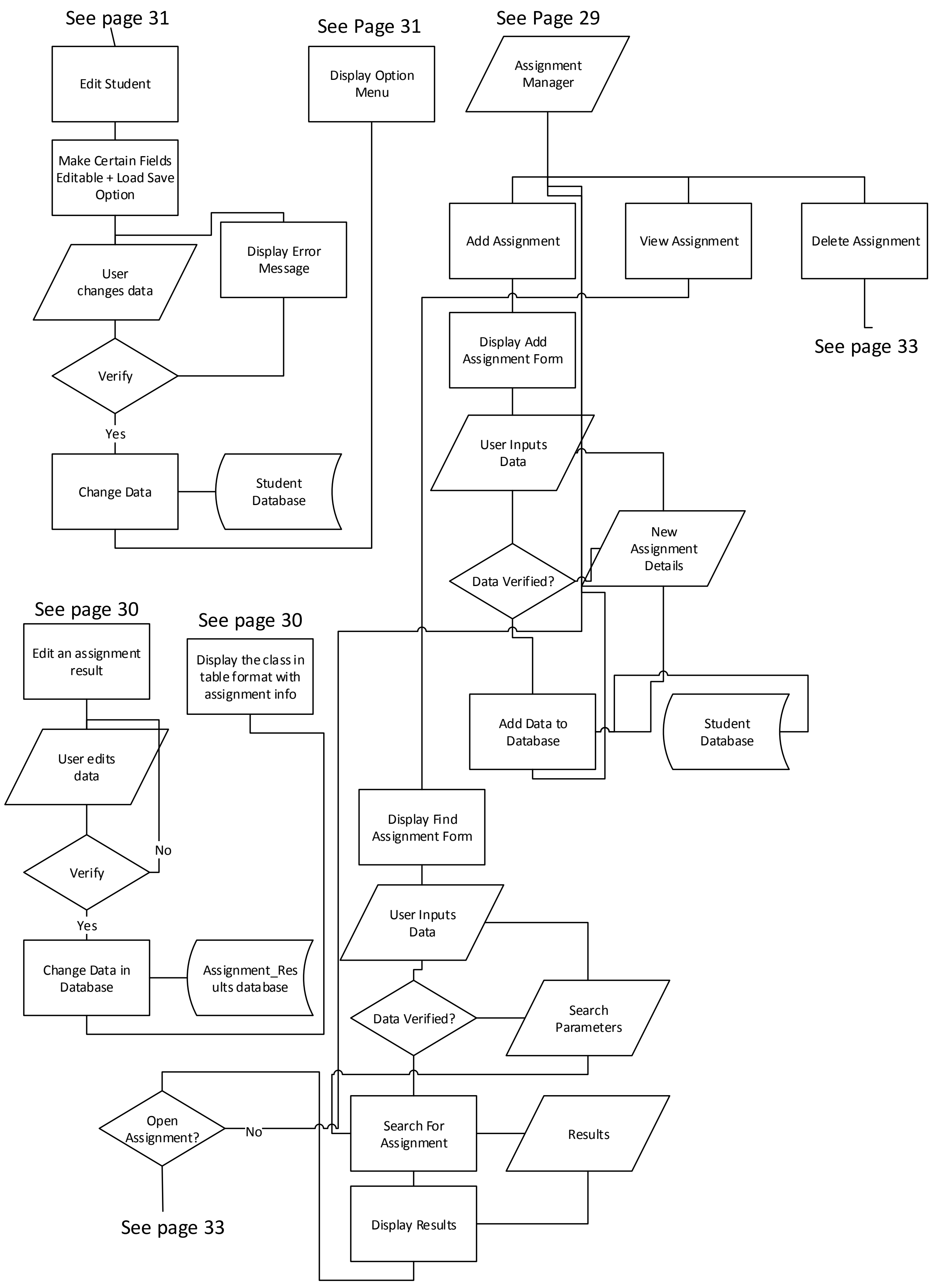
- First run
    - Creating details for the first user.
    - Adding the teachers to the database
    - Adding assignments to the database
- Administration
    - Editing the teachers – Eg. If they want to change their passwords or deactivating them.
    - Adding, editing assignments.
    - Removing assignments in case of error.
- Managing students
    - Adding a new student for when they enrol.
    - Editing a student. For example, if the teacher wants to apply a note to the student or the student has accepted to get extra time in the exam after being added.
    - The option to a delete a student should be there in case the student drops the course early on or mistakes are made during input. It could also help for testing purposes.
- Statistics
    - Display people who need help with assignments or are at risk of failing
    - Display graphs of stats of how well assignments have been done over time.

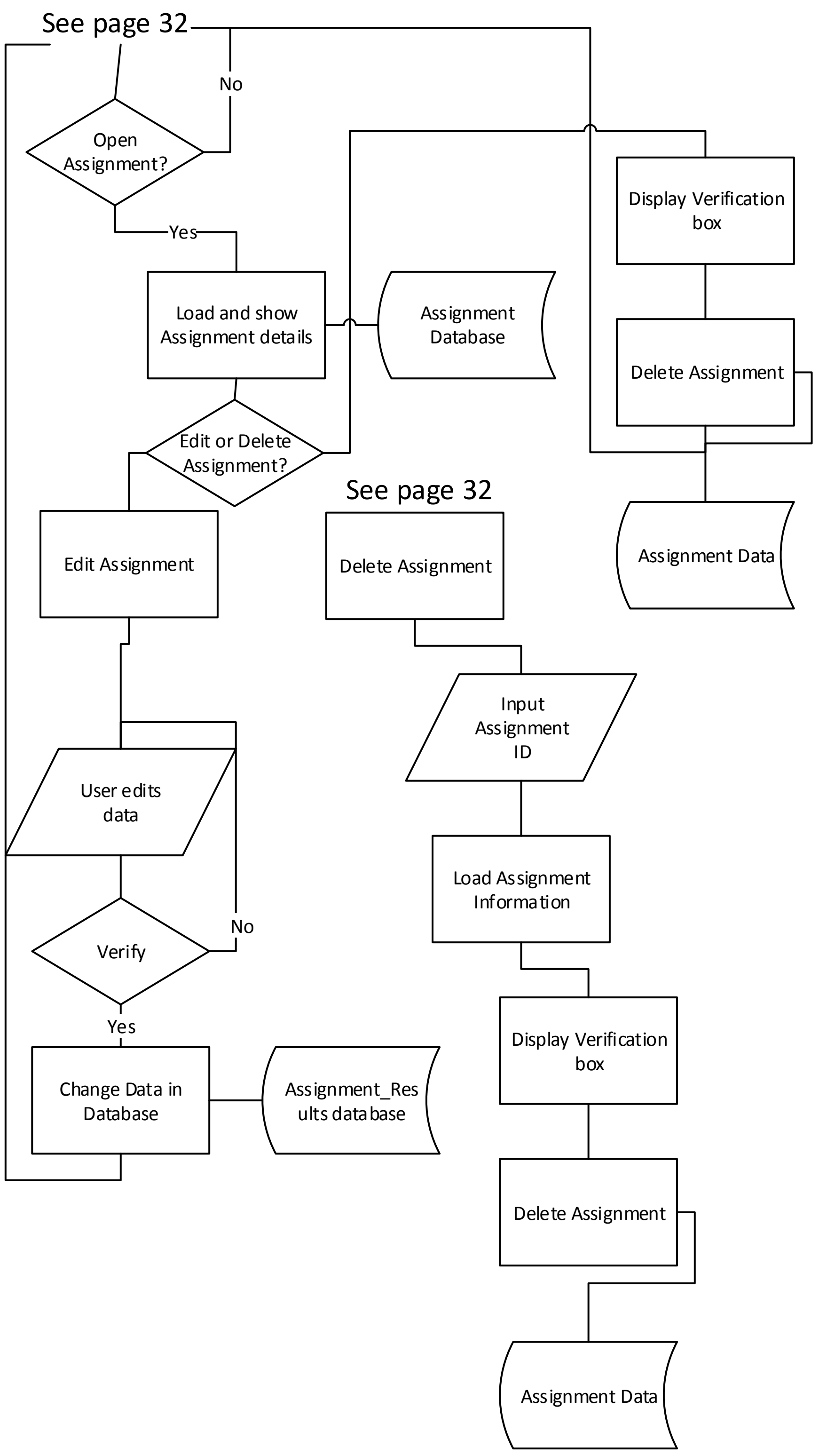
1.2 System flowcharts showing an overview of the complete system



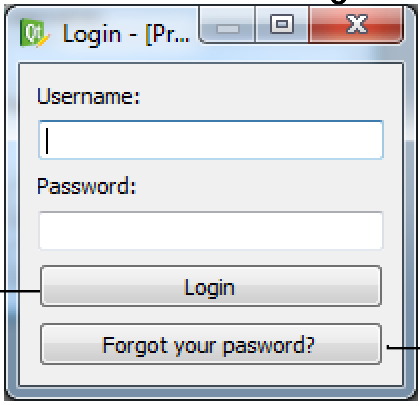

See page 29
Class Database
List Classes for account
Select option
Create new class
Display the create new class window
Input: Class information
Validate data
No
Valid?
Yes
Open Add Student window
Input: Student Details
Search for Student
Student Database
Display student list
Add Student?
No
Yes
Add student to class
Class Table
Continue Adding?
Yes
No
Edit an assignment result
See page 32
Select Class
Display the class in a table format with assignment information
Class, Students, Assignments tables
Exit
Select option
Edit the class
Is the teacher the owner of the class or an admin?
No
Yes
Input changes
Check if valid
Valid?
Save
AssignmentResults
View additional information for the class
Process the data
Students, Assignments
Display the statistics
Exit
Remove Class
See page 31


See page 30
Remove Class
Display Verification box
Verified?
No
Yes
Attempt to remove class
Class Database
Success?
Display Error Message with reason
Return to Class List
See page 29
See page 30
Display the class in a table format with assignment information
See page 29
Student Manager
Display Option Menu
Choose Option
Add Student
View Student
Delete Student
Display Add Student Form
User Inputs Data
NewStudent Details
Data Verified?
Student Database
Add Data to Database
Display Find Student Form
User Inputs Data
Search Parameters
Data Verified?
Search For Student
Student Database
Results
Display Results
Open student?
Yes
No
Input Student ID
Load Student Information
Display Verification box
Delete Student
Student Data
Load and show student details
Student, class, assignment results Database
Edit or Delete Student?
Delete Student
Edit Student
Neither
See page 32

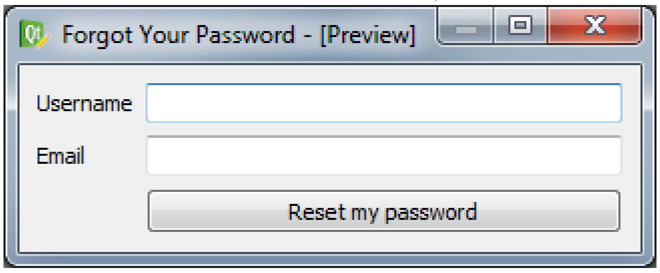

See page 31
Edit Student
Make Certain Fields Editable + Load Save Option
User changes data
Display Error Message
Verify
Yes
Change Data
Student Database
See Page 31
Display Option Menu
See Page 29
Assignment Manager
Add Assignment
View Assignment
Delete Assignment
See page 33
Display Add Assignment Form
User Inputs Data
Data Verified?
New Assignment Details
Add Data to Database
Student Database
See page 30
Edit an assignment result
User edits data
Verify
No
Yes
Change Data in Database
Assignment_Results database
See page 30
Display the class in table format with assignment info
Display Find Assignment Form
User Inputs Data
Data Verified?
Search Parameters
Open Assignment?
No
Search For Assignment
Results
Display Results
See page 33


See page 32
No
Open Assignment?
Yes
Display Verification box
Load and show Assignment details
Assignment Database
Delete Assignment
Edit or Delete Assignment?
See page 32
Edit Assignment
Delete Assignment
Assignment Data
Input Assignment ID
User edits data
Load Assignment Information
No
Verify
Yes
Display Verification box
Change Data in Database
Assignment_Results database
Delete Assignment
Assignment Data

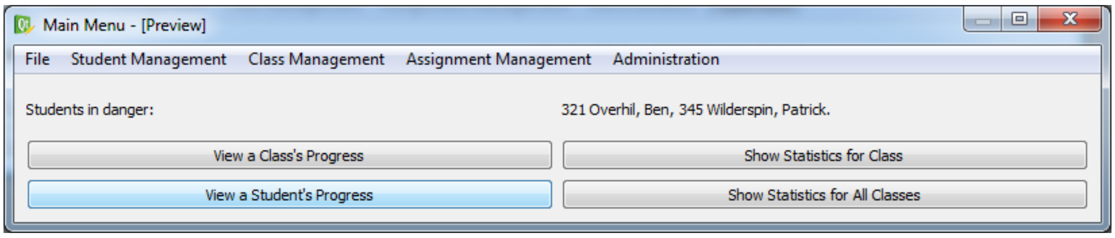
2. User Interface Designs



This is the first thing that a user will see upon loading, to keep the system secure.
If login has failed, a dialog box will alert the user.

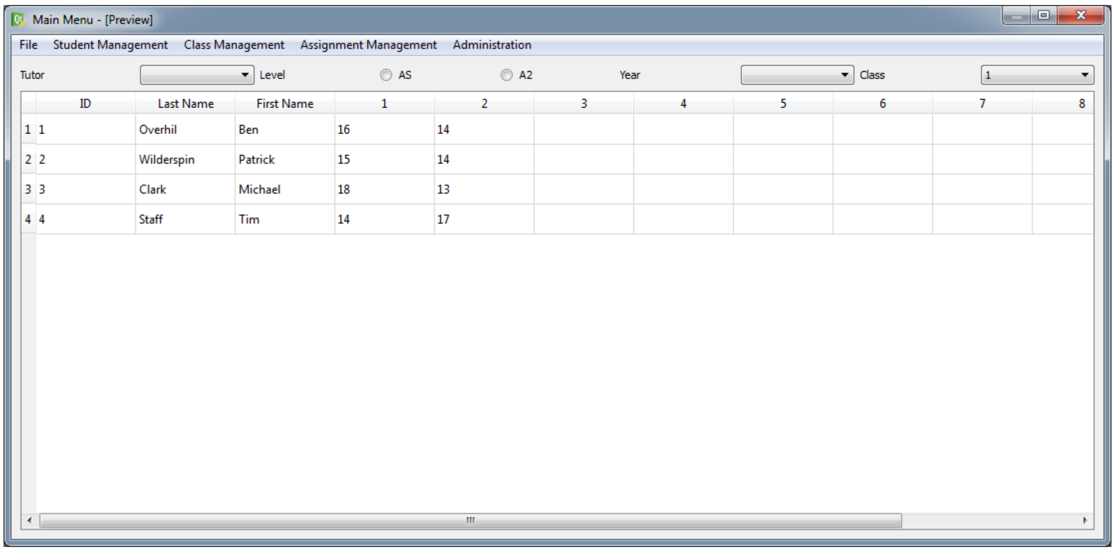


This simple box allows a user to put in a username and email, and the program will be reset their password and email the new password.
A dialog box will alert the user if it was successful, where it will then return to the login screen.


Main Menu - [Preview]
File Student Management Class Management Assignment Management Administration
Students in danger:
321 Overhil, Ben, 345 Wilderspin, Patrick.
View a Class's Progress
Show Statistics for Class
View a Student's Progress
Show Statistics for All Classes

This is the main menu. It allows the user to select the option they require.
It includes an alert to the tutor of the students that are in danger of failing.
The less used options are available in the menu bar above, while the most commonly used are in buttons.

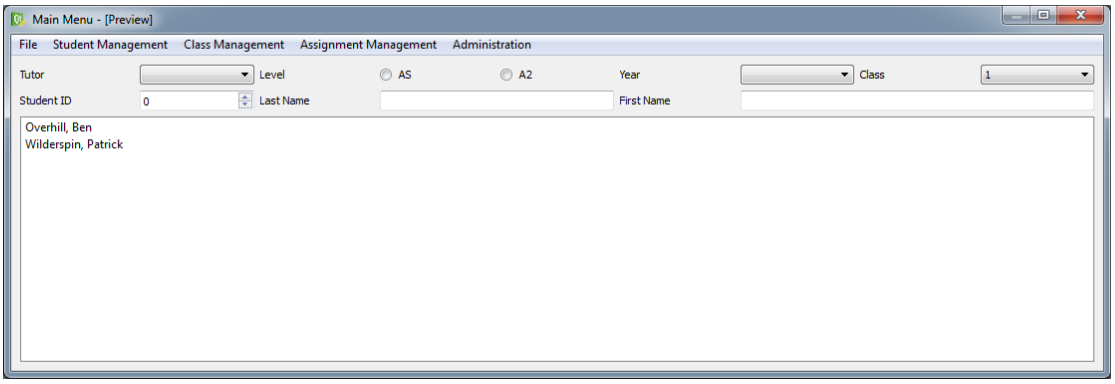
View a Class's Progress



	ID	Last Name	First Name	1	2	3	4	5	6	7	8
1	1	Overhil	Ben	16	14						
2	2	Wilderspin	Patrick	15	14						
3	3	Clark	Michael	18	13						
4	4	Staff	Tim	14	17						

This is where we look at a class's assignments progress. The user first sees a list of every student. They then use the options to pick the class they wish to view.
A user can edit a box by double clicking on it. If they have not already entered their additional password, they will be asked to do so.
Colours *may* be used to bring attention to the assignments that did not do well.

View a Student's Progress

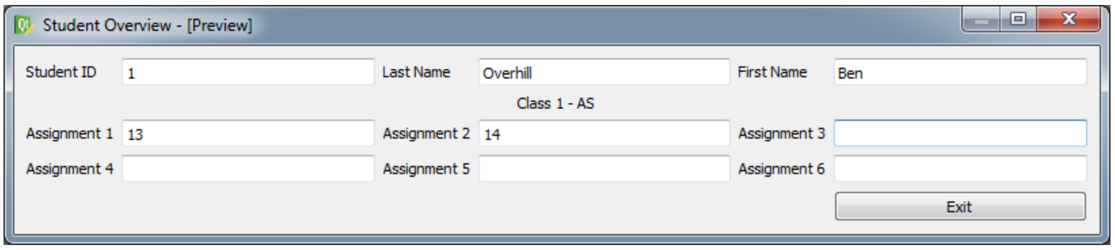


Class: 1
Student ID: 0

Overhill, Ben
Wilderspin, Patrick

This is where we find a student to open. The concept is the same as viewing a class's progress, but also able to narrow it down by Student's ID, Last name and first name. A user can open a student's profile by double-clicking them.

Viewing a Student's Progress - Student



Student ID: 1 Last Name: Overhill First Name: Ben

Class 1 - AS

Assignment 1: 13 Assignment 2: 14 Assignment 3:

Assignment 4: Assignment 5: Assignment 6:

This window shows a student’s overview of his assignment results. It allows the user to change their Assignments and exit. If the student does A2, it will display information for that too.

3. Hardware Specification

The following hardware specification is what is required to run the program

- Monitor at a minimum resolution of 800x600
- Keyboard and Mouse
- Storage for the Program & DB, Probably Network or USB stick.
- A minimum of 256MB of RAM **in addition** to any that the OS requires should be enough.

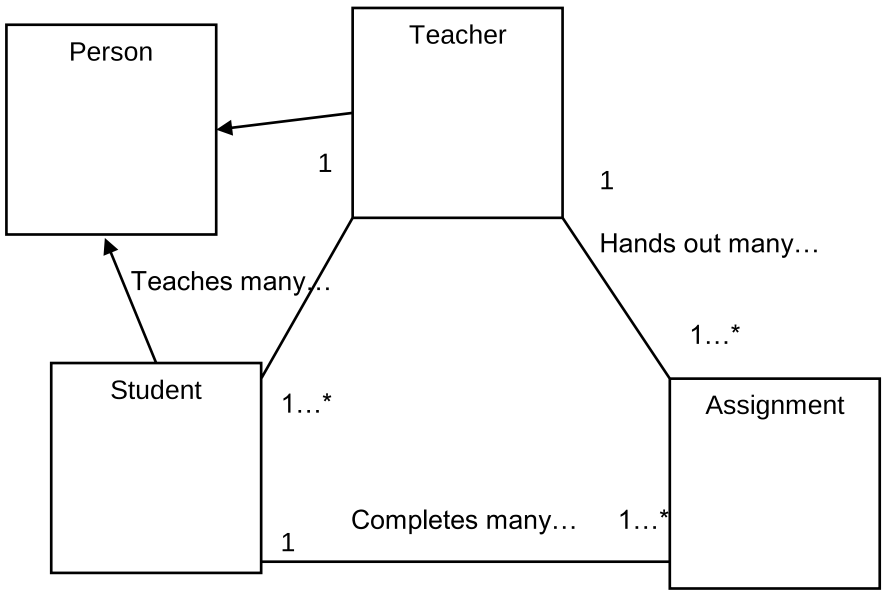
If the program will include the option to print reports, a printer will be required too.

4. Program Structure

4.1 Topdown design structure charts

4.2 Algorithms in pseudo-code for each data transformation process

4.3 Object diagrams



4.4 Class definitions

Person
Id FirstName LastName Email LastEmailed
getID getFirstName

setFirstName getLastName setLastName getEmail setEmail getLastEmailed setLastEmailed

Teacher
Username Password Admin ResetQuestion ResetAnswer
getAdmin setAdmin setUsername getUsername setPassword checkPasswordMatch SetResetQuestion GetResetQuestion SetResetAnswer CheckResetAnswerMatch

Student
DOB Scribe 25Extra 50Extra GCSEResults AssignmentResults

getDOB setDOB getScribe setScribe get25Extra set25Extra get50Extra set50Extra getGCSEResults setGCSEResults getAssignmentResults setAssignmentResults

Assignment
ID Name Description Deadline
getID getName setName getDescription setDescription getDeadline setDeadline

5. Prototyping

5.1 Consideration of impact on design and development

I plan to do the following prototypes:

- GUI – A table such as a spreadsheet
- SQL statements involving multiple tables
- Creating graphs from data using matplotlib
- Using slight amounts of colour in the program

6. Definition of Data Requirements

6.1 Identification of all data input items

The following variables must be input into the system. Due to the nature of the program, almost all of the items stored need to the input originally.

Name
TeacherUserName
TeacherPassword
TeacherAdmin
TeacherAdditionalPassword
TeacherName
TeacherEmail
TeacherQuestion
TeacherAnswer
StudentSurname
StudentForename
StudentDOB
StudentEMail
StudentScribe
Student25Extra
Student50Extra
StudentGCSEResults
StudentNotes
AssignmentName
AssignmentDescription
AssignmentDeadline
AssignmentMaxMarks
AssignmentMark
AssignmentNotes
SMTPHost
SMTPUsername
SMTPPassword

6.2 Identification of all data output items
As the system is mainly a database that stores and retrieves data, it will be required to output all items it inputs **except for the password variables.** It will also need to output:

- Students at risk of failing
- Graphs

However, it will also need to display a table about the students and their assignment progress differently to how it was input.

6.3 Explanation of how data output items are generated
Students at risk of failing

It will calculate this based on the average of the scores of the 3 last assignments the student has performed and match this to a percentage.

Graphs

These will be generated using the external library *matplotlib*.

6.4 Data Dictionary

Name	Data Type	Length	Validation	Example Data	Comment
TeacherID	Integer	0-255		42	Automatically created so no validation
TeacherUserName	String	Up to 20	Length, More than 3 characters	“Bmanger”	
TeacherPassword	String	32	Length	“098f6bcd4621d373cade4e832627b4f6”	Encypted using MD5. Checked before encrypting and storing. Actual password may be up to 64 characters.
TeacherAdmin	Boolean	1	Presence check	True	Defines if the teacher has admin privs.
TeacherAdditionalPassword	String	32	Length	“098f6bcd4621d373cade4e832627b4f6”	Encrypted using MD5. Checked before encrypting and storing. Actual password may be up to 64 characters. For very restricted areas.
TeacherName	String	Up to 32	Length/Presence check	“Billis Manger”	Presence check includes space in the middle.

TeacherEmail	String	Up to 128	Length/Presence check	Bmanger@longroad.ac.uk	Include check for @ and domain. My limit down to one domain.
TeacherQuestion	String	Up to 512	Length	"Where did you buy your first car from?"	
TeacherAnswer	String	Up to 32	Length	"Bucks and Dos"	Capitalisation will not matter when comparing.
TeacherLastEmailed	Date		Format	20/11/2013	To prevent teachers being over emailed about problems.
StudentID	Integer	0-4096		23	Automatically created so no validation.
StudentSurname	String	Up to 32	Length	"Davies"	
StudentForename	String	Up to 32	Length	"Greg"	
StudentDOB	Date		Format	22/04/1995	
StudentEMail	String	Up to 128	Length/Presence Check	64634@longroad.ac.uk	Include check for @ and domain. May limit down to one domain.
StudentScribe	Boolean		Presence Check	True	
Student25Extra	Boolean		Presence Check	True	
Student50Extra	Boolean		Presence Check	True	
StudentGCSEResults	Float	0-10	Range	7.66	
StudentLastEmailed	Date		Format	20/11/2013	To prevent students being over emailed about problems.
StudentNotes	String	Up to 1024			Not required for program to work and optional so no validation.
AssignmentID	Integer	0-255			Automatically created so no validation.
AssignmentName	String	Up to 32	Length	"Reading on Booleans"	

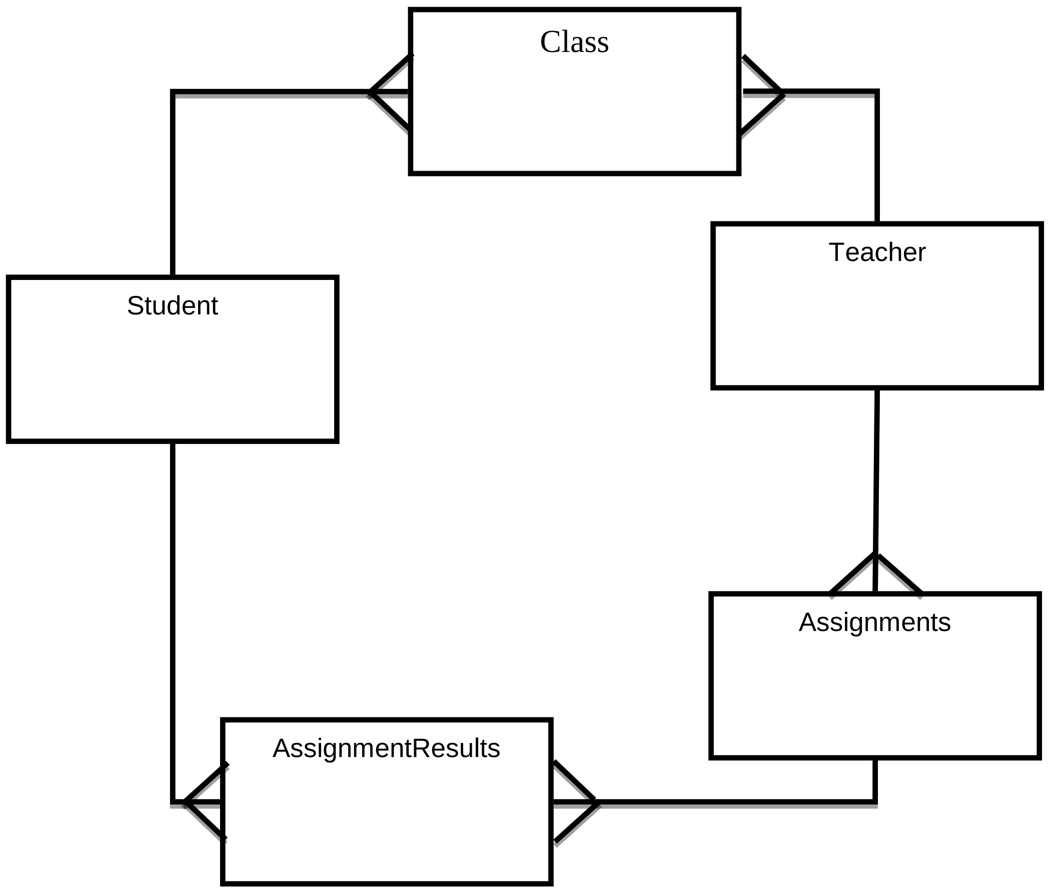
AssignmentDescription	String	Up to1024	Length	“AS Computing text book – Pages 2-7”	
AssignmentDeadline	Date		Format	01/11/2013	
AssignmentMaxMarks	Integer	Up to 256	Format/Length	20	Defines max marks for the assignment
AssignmentMark	Integer	Up to 256	Format/Length	50	
AssignmentNotes	String	Up to 1024	Length	“Notes”	
SMTPHost	String	Up to 32	Length/presence check	“smtp.gmail.com”	Check of valid domain.
SMTPUsername	String	Up to 32	Length/Presence check	“Bmanger’	Check of characters
SMTPPassword	String	Up to 32	Length	“cravat332”	Needs to be encrypted but it needs to also be easily decrypted by the program (and the program alone)
LastBackedUp	Date			14/03/2012	Automatically used by program, no validation required.

6.5 Identification of appropriate storage media
The most appropriate place to store the data would be on the college network as there it is both accessible by all and backed up. However, an alternate backup would be the good idea in case an error occurs that doesn't affect the whole network. A good choice would be a usb memory stick or rewritable optical media.

7. Database Design

7.1 Normalisation

7.1.1 ER Diagrams



7.1.2 UNF to 3NF

UNF	1NF
TeacherID TeacherUserName TeacherPassword TeacherAdmin TeacherAdditionalPassword TeacherName TeacherEmail TeacherQuestion TeacherAnswer TeacherLastEmailed StudentID StudentSurname StudentForename StudentDOB StudentEmail StudentScribe Student25Extra Student50Extra StudentGCSEResults StudentLastEmailed StudentNotes Year YearStart AssignmentID AssignmentName AssignmentDescription AssignmentDeadline AssignmentMaxMark AssignmentMark AssignmentNotes	**Repeating** ➢ **StudentID** ➢ MarkID TeacherUserName TeacherPassword TeacherAdmin TeacherAdditionalPassword TeacherName TeacherEmail TeacherQuestion TeacherAnswer TeacherLastEmailed StudentSurname StudentForename StudentDOB StudentEmail StudentScribe Student25Extra Student50Extra StudentGCSEResults StudentLastEmailed StudentNotes Year YearStart **Non-repeating** ➢ MarkID AssignmentID AssignmentName AssignmentDescription AssignmentDeadline AssignmentMaxMark AssignmentMark AssignmentNotes

2NF	3NF
➢ **StudentID** ➢ MarkID Year YearStart TeacherUserName TeacherPassword TeacherAdmin TeacherAdditionalPassword TeacherName TeacherEmail TeacherQuestion TeacherAnswer TeacherLastEmailed ➢ **StudentID** StudentSurname StudentForename StudentDOB StudentEmail StudentScribe Student25Extra Student50Extra StudentGCSEResults StudentLastEmailed StudentNotes ➢ MarkID AssignmentID AssignmentName AssignmentDescription AssignmentDeadline AssignmentMaxMark AssignmentMark AssignmentNotes	➢ **StudentID** ➢ AssignmentID ➢ TeacherID TeacherUserName TeacherPassword TeacherAdmin TeacherAdditionalPassword TeacherName TeacherEmail TeacherQuestion TeacherAnswer TeacherLastEmailed ➢ ClassID *StudentID* *TeacherID* Year YearStart ➢ **StudentID** StudentSurname StudentForename StudentDOB StudentEmail StudentScribe Student25Extra Student50Extra StudentGCSEResults StudentLastEmailed StudentNotes ➢ StudentID ➢ AssignmentID AssignmentMark AssignmentNotes ➢ AssignmentID AssignmentName AssignmentDescription AssignmentDeadline AssignmentMaxMark

7.2 SQL Queries
I will be using Python to program so the following code includes parts from it.

This query will add a new student to the database.

```
sql = """insert into Student(StudentLastName, StudentFirstName,
            StudentDOB, StudentEmail,StudentScribe,Student25Extra,
            Student50Extra, StudentGCSEResults, StudentLastEmailed, StudentNotes)
            values
            ('{0}','{1}','{2}','{3}','{4}','{5}','{6}','{7}','{8}','{9}')""".format(
               StudentLastName, StudentFirstName, StudentDOB, StudentEmail,
               StudentScribe, Student25Extra, Student50Extra, StudentGCSEResults,
               StudentLastEmailed, StudentNotes)
```

This is an example of a find student code. Due to the nature of it, most of it is in Python. There is an example SQL code that is generated at the end, however.

```
def find_student(self, StudentID=None, StudentLastName=None, StudentFirstName=None,
            StudentDOB=None, StudentEmail=None, StudentScribe=None, Student25Extra=None,
            Student50Extra=None, StudentGCSEResults=None, StudentLastEmailed=None):
    #This function is designed to find all the rows that match the following data.
    #It works in the same way as the update function.

    #Creates a new list
    parameters = []

    #Detects if Student the named parameters are used
    #if Studentso, it will append them to the list
    if StudentID != None:
        parameters.append(("StudentID",StudentID))
    if StudentLastName != None:
        parameters.append(("StudentLastName",StudentLastName))
    if StudentFirstName != None:
        parameters.append(("StudentFirstName",StudentFirstName))
    if StudentDOB != None:
        parameters.append(("StudentDOB",StudentDOB))
    if StudentEmail != None:
        parameters.append(("StudentEmail",StudentEmail))
    if StudentScribe != None:
        parameters.append(("StudentScribe",StudentScribe))
    if Student25Extra != None:
        parameters.append(("Student25Extra",Student25Extra))
    if Student50Extra != None:
        parameters.append(("Student50Extra",Student50Extra))
    if StudentGCSEResults != None:
        parameters.append(("StudentGCSEResults",StudentGCSEResults))
    if StudentLastEmailed != None:
        parameters.append(("StudentLastEmailed",StudentLastEmailed))

    #This begins the select command for the list
    #It's choosing only certain columns for the list, because of security.
    sql = """select *
        FROM student
        where """

    #This adds all the parameters to the sql statement
    for parameter in parameters:
        sql = sql + "{0}='{1}' and".format(parameter[0],parameter[1])

    #This removes the final " and" from the sql statement
    sql = sql[:-4]
    return self._select_query(sql)
```

Example:

```
select *
        FROM student
        where StudentScribe='1' andStudent25Extra='1'
```

Likewise, here's one for editing a student.

```
    def edit_student(self, StudentID, StudentLastName=None, StudentFirstName=None,
                StudentDOB=None, StudentEmail=None, StudentScribe=None, Student25Extra=None,
                Student50Extra=None, StudentGCSEResults=None, StudentLastEmailed=None,
StudentNotes=None):
        #This function allows me to edit all of a student's values in one go
        #It uses named parameters to allow me to have them optional

        #Starts the list of changes needed
        changes = []

        #Checks each value to see if Studentthey're used
        #if Studentused, it will append each change to the list as a list
        #Ie, a list of lists.
        if StudentLastName != None:
            changes.append(("StudentLastName",StudentLastName))
        if StudentFirstName != None:
            changes.append(("StudentFirstName",StudentFirstName))
        if StudentDOB != None:
            changes.append(("StudentDOB",StudentDOB))
        if StudentEmail != None:
            changes.append(("StudentEmail",StudentEmail))
        if StudentScribe != None:
            changes.append(("StudentScribe",StudentScribe))
        if Student25Extra != None:
            changes.append(("Student25Extra",Student25Extra))
        if Student50Extra != None:
            changes.append(("Student50Extra",Student50Extra))
        if StudentGCSEResults != None:
            changes.append(("StudentGCSEResults",StudentGCSEResults))
        if StudentLastEmailed != None:
            changes.append(("StudentLastEmailed",StudentLastEmailed))
        if StudentNotes != None:
            changes.append(("StudentNotes",Notes))
        #This is the start of the sql statement that will be added to
        sql = "update student set "
        #Iteration of each list within the changes list
        for update in changes:
            #This adds each update to the sql statement
            sql += "{0}='{1}', ".format(update[0],update[1])

        #Remove the last 2 characters ', '
        sql = sql[:-2]
        #Adds which ID to edit
        sql+= " where StudentID ='{0}'".format(StudentID)

        #Performs the query to the database
        self._query(sql)
```

```
            update student set StudentScribe='1', Student25Extra='1' where StudentID ='1'
```

Example:

8. Security and Integrity of the System and Data

8.1 Security and Integrity of Data

The client requires that the data is to be encrypted. This is because the data contains information about pupils and must comply with the Data Protection Act.

The actual location of the data should be on the network in a place that can only be accessed by teachers. Unfortunately, my client uses a Mac and may be unable to directly access it and may need to store it on his personal memory stick.

The passwords will additional security in the form of MD5 hashing.

Backups should occur as part of the College's policy but also should be done separately with the DB.

8.2 System Security

Access will be restricted to teachers relating to the department only, with an additional password for editing data. It is hoped that encryption and physical access will prevent unauthorised people from accessing it.

9. Validation

Most of my data will involve Booleans or integers. The Booleans should not need validation as these will be done via either radio buttons or dropdowns.

The integers will be mostly checked from range. However some will simply be checked on whether or not they are numbers.

The email addresses and dates will be checked using REGEX. This is to ensure a uniform format. It is also to ensure that numbers do not exceed what they are not allowed to exceed (i.e. Nothing over 12 in months).

All strings will be length checked, however, these will not require any other checking.

10. Testing

10.1 Outline Plan

10.1.1 Identification and explanation of suitable test strategies

Test Series	Purpose of Test Series	Testing Strategy	Strategy Rationale
1	Test inputting values	Bottom-up	Each will be tested when ready
2	Test transformation between program and database	System-testing	When each overall module is ready.
3	Test the alert system	System testing	When the system is more or less complete
4	Test the graphs	Bottom-up	As they become available.

10.2 Detailed Plan

Errors should be faced with a red mark and a tooltip

Test Series and number	Purpose	Description	Test Data	Test Data Type (Normal/ Erroneous /Extreme)	Expected Result	Actual Result
1.1	Validate email address	Ensure any email addresses are entered correctly	[Nothing] aaa@longroad.ac.ua aaa@longroad.ac.uk adasd.longroad.com	Error Error Normal Error	Error Error Valid Error	
1.2	Validate Dates	Ensure dates are correct for conversion	[Nothing] 21/33/1993 12/11/1994 12/11/1003	Error Error Normal Extreme	Error Error Valid Error	
1.3	Validate Integers	Ensure integers are correct	[Nothing] 10.2 30 -1	Error Error Normal Error	Error Error Valid Error	
2 This is simply adding each item to the program via the gui and then checking the database to ensure everything is correct.						
3.1	Validate Correct	Ensure that it works	Data that is below a certain percentage	Should be alerted		
3.2	Valid Incorrect	Ensure there are no false positives	Data that is above the percentage.	Should not be alerted		
3.3	Validate recheck function	Ensure it updates properly	Change the percentage up and down	Should change as required.		

Testing

1. **Test Plan**

1.1 Detailed Test Plan

Key for Changes		
✏ New Test	↻ Moved Test Number	✂ Deleted Test

Test Series and number	Purpose	Description	Test Data	Test Data Type (Normal/ Erroneous /Extreme)	Expected Result	Actual Result	Evidence in Appendix
1.1 ✏	Validate Option Choice	Ensures the user can choose option correctly	[Nothing]	Error	Error displayed and asked again	As expected	1.1.1 on Page 62
			0	Normal/Extreme	Exit program or go up a level	As expected	1.1.2 on Page 63
			1	Normal	Load a menu Option	As expected	1.1.3 on Page 64
			Ab	Error	Error displayed and asked again	As expected	1.1.4 on Page 65
1.2 ↻	Validate Integers	Ensure integers are correct	[Nothing]	Error	Error displayed and asked again	No Error displayed. Still looped	1.2.1.1 on Page 66
						Error and looped.	1.2.1.2 on Page 66
			10.2	Error	Error displayed and asked again	As expected	1.2.2 on Page 66
			30	Normal	Accepted and moved	As expected	1.2.3 on Page

					on		67
			-1	Error	Error displayed and asked again	Accepted	1.2.4.1 on Page 67
						As expected	1.2.4.2 on Page 67
1.3	Validate email address	Ensure any email addresses are entered correctly	[Nothing]	Error	Error displayed and asked again	As expected	1.3.1 on Page 67
			aaa@longroad.ac.ua	Error	Error displayed and asked again	As expected	1.3.2 on Page 67
			aaa@longroad.ac.uk	Normal	Normal	As expected	1.3.3 on Page 68
			adasd.longroad.com	Error	Error displayed and asked again	As expected	1.3.4 on Page 68
			@longroad.ac.uk ✎	Error	Error displayed and asked again	As expected	1.3.5 on Page 68
1.4	Validate Dates	Ensure dates are correct for conversion	[Nothing]	Error	Error displayed and asked again	As expected	1.4.1 on Page 68
			21/33/1993	Error	Error displayed and asked again	As expected	1.4.2 on Page 69
			1994-04-03	Normal	Valid	As expected	1.4.3 on Page 69
			1003-11-12	Extreme	Error displayed and	Accepted	1.4.4 on Page

					asked again		69
			12121993 ✏	Error	Error displayed and asked again	As expected	1.4.5 on Page 69
			“Date” ✏	Error	Error displayed and asked again	As expected	1.4.6 on Page 69
2 ✂ This is simply adding each item to the program via the gui and then checking the database to ensure everything is correct.							
3.1 ✂	Validate Correct	Ensure that it works	Data that is below a certain percentage	Should be alerted			
3.2 ✂	Valid Incorrect	Ensure there are no false positives	Data that is above the percentage.	Should not be alerted			
3.3 ✂	Validate recheck function	Ensure it updates properly	Change the percentage up and down	Should change as required.			
4. Statistics Checking							
4.1 ✏	Check function runs	Ensures the statistics functions runs correctly	Run the function before all are ready	Normal	Displays results with unfinished marked as 0	Crashes	
			Run after	Normal	Displays data correctly	Displays data, incorrectly.	

1.2 Changes from Original Plan
The main change is the removal of the GUI sections due to the lack of a GUI to test and therefore I am unable to test it.

I have added a few additional tests to the emails and dates to ensure the regular expressions are accurately preventing issues.

2. Test Data

2.1. Test Data

Test Series and number	Purpose	Description	Test Data	Test Data Type (Normal/ Erroneous /Extreme)	Expected Result	Actual Result	Evidence in Appendix
1.1 ✎	Validate Option Choice	Ensures the user can choose option correctly	[Nothing]	Error	Error displayed and asked again	As expected	1.1.1 on Page 62
			0	Normal/Extreme	Exit program or go up a level	As expected	1.1.2 on Page 63
			1	Normal	Load a menu Option	As expected	1.1.3 on Page 64
			Ab	Error	Error displayed and asked again	As expected	1.1.4 on Page 65
1.2 ()	Validate Integers	Ensure integers are correct	[Nothing]	Error	Error displayed and asked again	No Error displayed. Still looped	1.2.1.1 on Page 66
						Error and looped.	1.2.1.2 on Page 66
			10.2	Error	Error displayed and asked again	As expected	1.2.2 on Page 66
			30	Normal	Accepted and moved	As expected	1.2.3 on Page

					on		67
			-1	Error	Error displayed and asked again	Accepted	1.2.4.1 on Page 67
						As expected	1.2.4.2 on Page 67
1.3	Validate email address	Ensure any email addresses are entered correctly	[Nothing]	Error	Error displayed and asked again	As expected	1.3.1 on Page 67
			aaa@longroad.ac.ua	Error	Error displayed and asked again	As expected	1.3.2 on Page 67
			aaa@longroad.ac.uk	Normal	Normal	As expected	1.3.3 on Page 68
			adasd.longroad.com	Error	Error displayed and asked again	As expected	1.3.4 on Page 68
			@longroad.ac.uk	Error	Error displayed and asked again	As expected	1.3.5 on Page 68
1.4	Validate Dates	Ensure dates are correct for conversion	[Nothing]	Error	Error displayed and asked again	As expected	1.4.1 on Page 68
			21/33/1993	Error	Error displayed and asked again	As expected	1.4.2 on Page 69
			1994-04-03	Normal	Valid	As expected	1.4.3 on Page 69
			1003-11-12	Extreme	Error displayed and	Accepted	1.4.4 on Page

					asked again		69
			12121993 ✎	Error	Error displayed and asked again	As expected	1.4.5 on Page 69
			"Date" ✎	Error	Error displayed and asked again	As expected	1.4.6 on Page 69
2 ✂ This is simply adding each item to the program via the gui and then checking the database to ensure everything is correct.							
3.1 ✂	Validate Correct	Ensure that it works	Data that is below a certain percentage	Should be alerted			
3.2 ✂	Valid Incorrect	Ensure there are no false positives	Data that is above the percentage.	Should not be alerted			
3.3 ✂	Validate recheck function	Ensure it updates properly	Change the percentage up and down	Should change as required.			
4. Statistics Checking							
4.1 ✎	Check function runs	Ensures the statistics functions runs correctly	Run the function before all are ready	Normal	Displays results with unfinished marked as 0	Crashes	
			Run after	Normal	Displays data correctly	Displays data, incorrectly.	

2.2 Changes from Original Data
Same as the overall tests, I have added some additional test data in a few to ensure the program works against them, and added whole sets to other areas.

3. Annotated Samples

3.1. Actual Results

Test Series and number	Purpose	Description	Test Data	Test Data Type (Normal/ Erroneous /Extreme)	Expected Result	Actual Result	Evidence in Appendix
1.1 ✏	Validate Option Choice	Ensures the user can choose option correctly	[Nothing]	Error	Error displayed and asked again	As expected	1.1.1 on Page 62
			0	Normal/Extreme	Exit program or go up a level	As expected	1.1.2 on Page 63
			1	Normal	Load a menu Option	As expected	1.1.3 on Page 64
			Ab	Error	Error displayed and asked again	As expected	1.1.4 on Page 65
1.2 ()	Validate Integers	Ensure integers are correct	[Nothing]	Error	Error displayed and asked again	No Error displayed. Still looped	1.2.1.1 on Page 66
						Error and looped.	1.2.1.2 on Page 66
			10.2	Error	Error displayed and asked again	As expected	1.2.2 on Page 66
			30	Normal	Accepted and moved on	As expected	1.2.3 on Page 67
			-1	Error	Error displayed and asked again	Accepted	1.2.4.1 on Page 67
						As	1.2.4.2

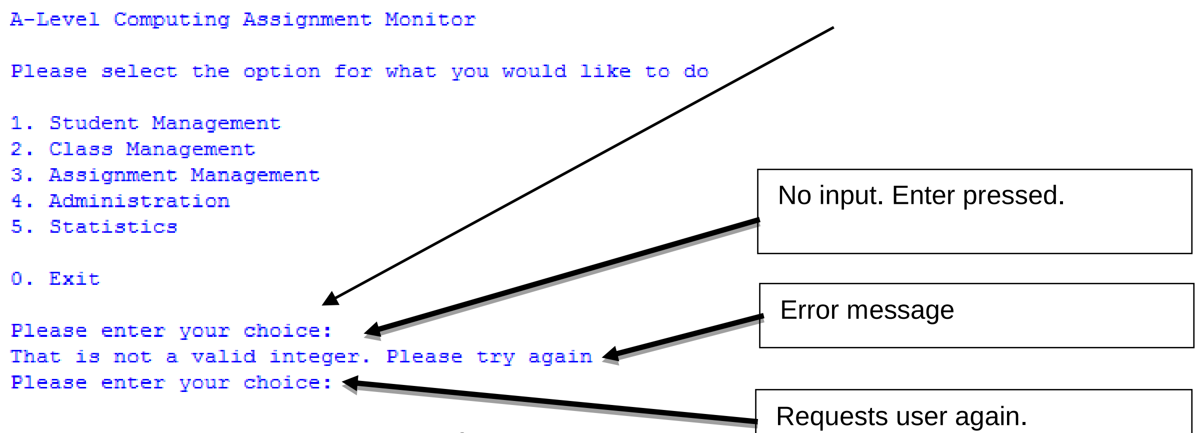
						expected	on Page 67
1.3 ⟳	Validate email address	Ensure any email addresses are entered correctly	[Nothing]	Error	Error displayed and asked again	As expected	1.3.1 on Page 67
			aaa@longroad.ac.ua	Error	Error displayed and asked again	As expected	1.3.2 on Page 67
			aaa@longroad.ac.uk	Normal	Normal	As expected	1.3.3 on Page 68
			adasd.longroad.com	Error	Error displayed and asked again	As expected	1.3.4 on Page 68
			@longroad.ac.uk ✎	Error	Error displayed and asked again	As expected	1.3.5 on Page 68
1.4 ⟳	Validate Dates	Ensure dates are correct for conversion	[Nothing]	Error	Error displayed and asked again	As expected	1.4.1 on Page 68
			21/33/1993	Error	Error displayed and asked again	As expected	1.4.2 on Page 69
			1994-04-03	Normal	Valid	As expected	1.4.3 on Page 69
			1003-11-12	Extreme	Error displayed and asked again	Accepted	1.4.4 on Page 69
			12121993 ✎	Error	Error displayed and asked	As expected	1.4.5 on Page 69

					again		
			“Date” ✏	Error	Error displayed and asked again	As expected	1.4.6 on Page 69
2 ✂ This is simply adding each item to the program via the gui and then checking the database to ensure everything is correct.							
3.1 ✂	Validate Correct	Ensure that it works	Data that is below a certain percentage	Should be alerted			
3.2 ✂	Valid Incorrect	Ensure there are no false positives	Data that is above the percentage.	Should not be alerted			
3.3 ✂	Validate recheck function	Ensure it updates properly	Change the percentage up and down	Should change as required.			
4. Statistics Checking							
4.1 ✏	Check function runs	Ensures the statistics functions runs correctly	Run the function before all are ready	Normal	Displays results with unfinished marked as 0	Crashes	
			Run after	Normal	Displays data correctly	Displays data, incorrectly.	

3.2. Evidence

1.1.1 Choosing an option – Empty

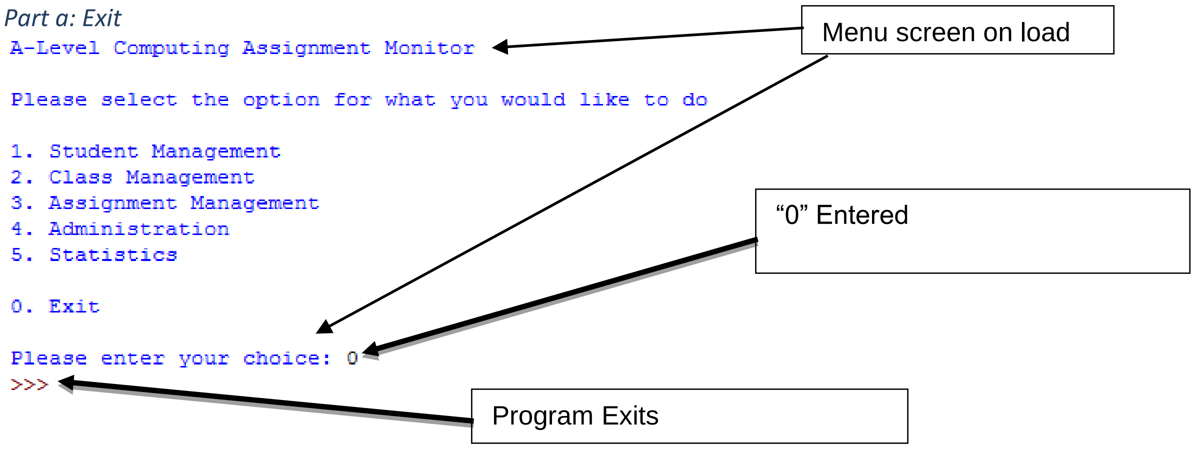
Menu screen on load



```
A-Level Computing Assignment Monitor

Please select the option for what you would like to do

1. Student Management
2. Class Management
3. Assignment Management
4. Administration
5. Statistics

0. Exit

Please enter your choice:
That is not a valid integer. Please try again
Please enter your choice:
```

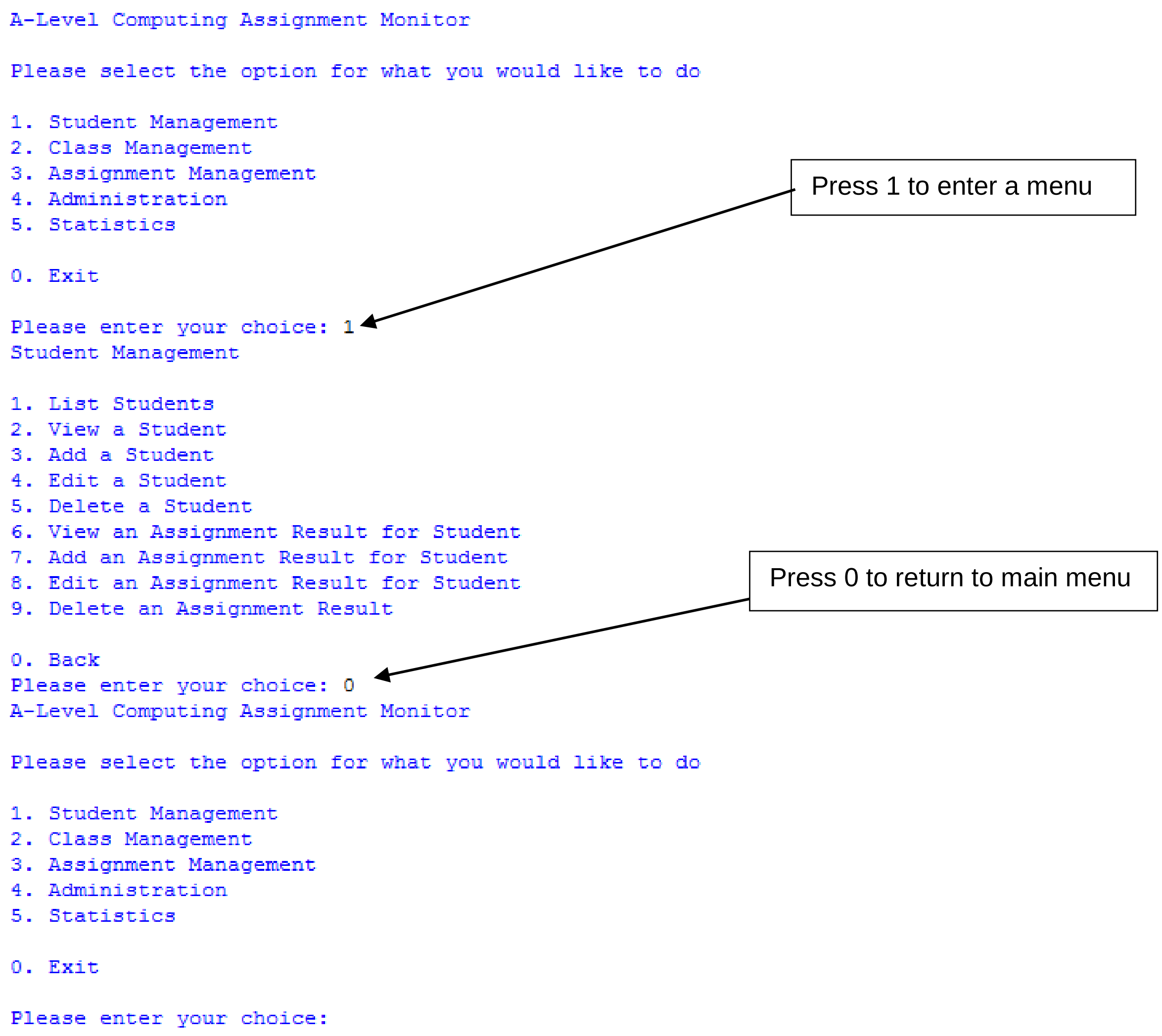
1.1.2 Choosing an option – Exit/Go Back

Part a: Exit



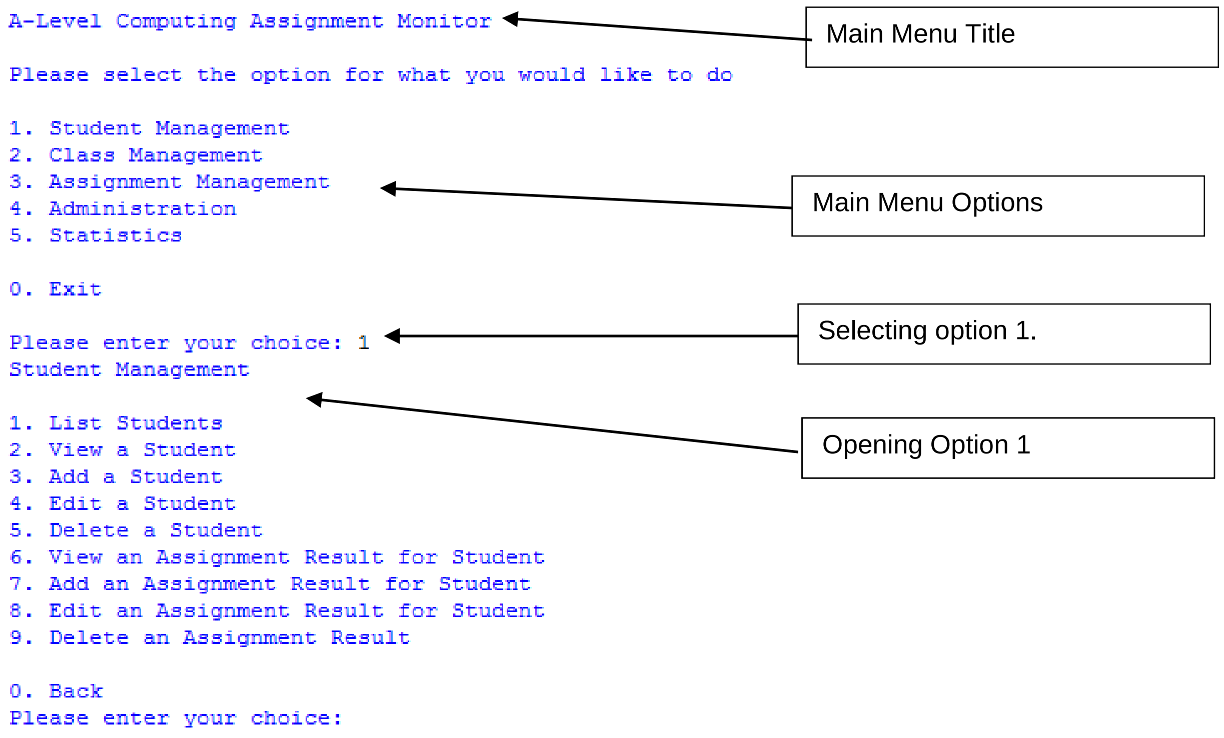
```
A-Level Computing Assignment Monitor

Please select the option for what you would like to do

1. Student Management
2. Class Management
3. Assignment Management
4. Administration
5. Statistics

0. Exit

Please enter your choice: 0
>>>
```

Part b: Go back



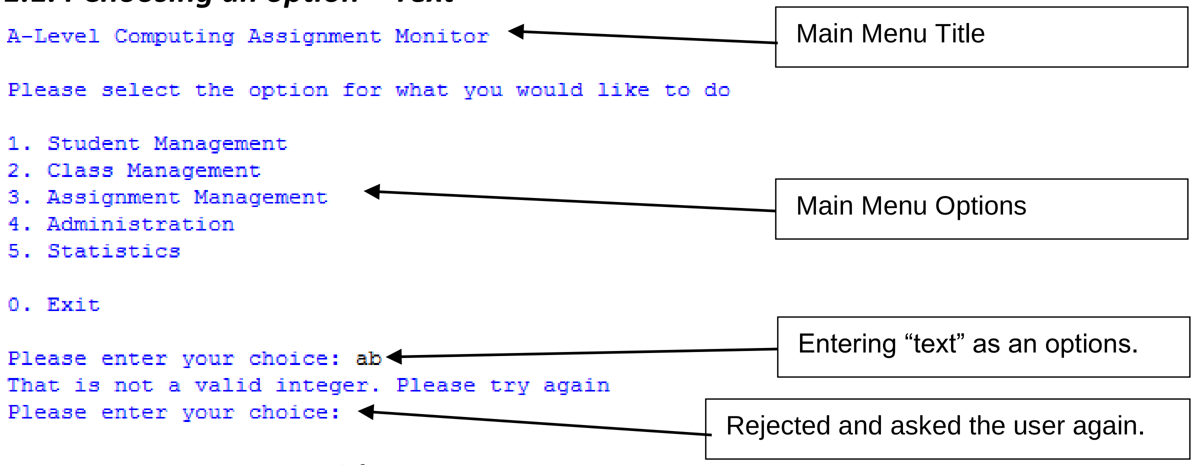
```
A-Level Computing Assignment Monitor

Please select the option for what you would like to do

1. Student Management
2. Class Management
3. Assignment Management
4. Administration
5. Statistics

0. Exit

Please enter your choice: 1
Student Management

1. List Students
2. View a Student
3. Add a Student
4. Edit a Student
5. Delete a Student
6. View an Assignment Result for Student
7. Add an Assignment Result for Student
8. Edit an Assignment Result for Student
9. Delete an Assignment Result

0. Back
Please enter your choice: 0
A-Level Computing Assignment Monitor

Please select the option for what you would like to do

1. Student Management
2. Class Management
3. Assignment Management
4. Administration
5. Statistics

0. Exit

Please enter your choice:
```

1.1.3 Choosing an option – Load Option



```
A-Level Computing Assignment Monitor

Please select the option for what you would like to do

1. Student Management
2. Class Management
3. Assignment Management
4. Administration
5. Statistics

0. Exit

Please enter your choice: 1
Student Management

1. List Students
2. View a Student
3. Add a Student
4. Edit a Student
5. Delete a Student
6. View an Assignment Result for Student
7. Add an Assignment Result for Student
8. Edit an Assignment Result for Student
9. Delete an Assignment Result

0. Back
Please enter your choice:
```

1.1.4 Choosing an option – Text

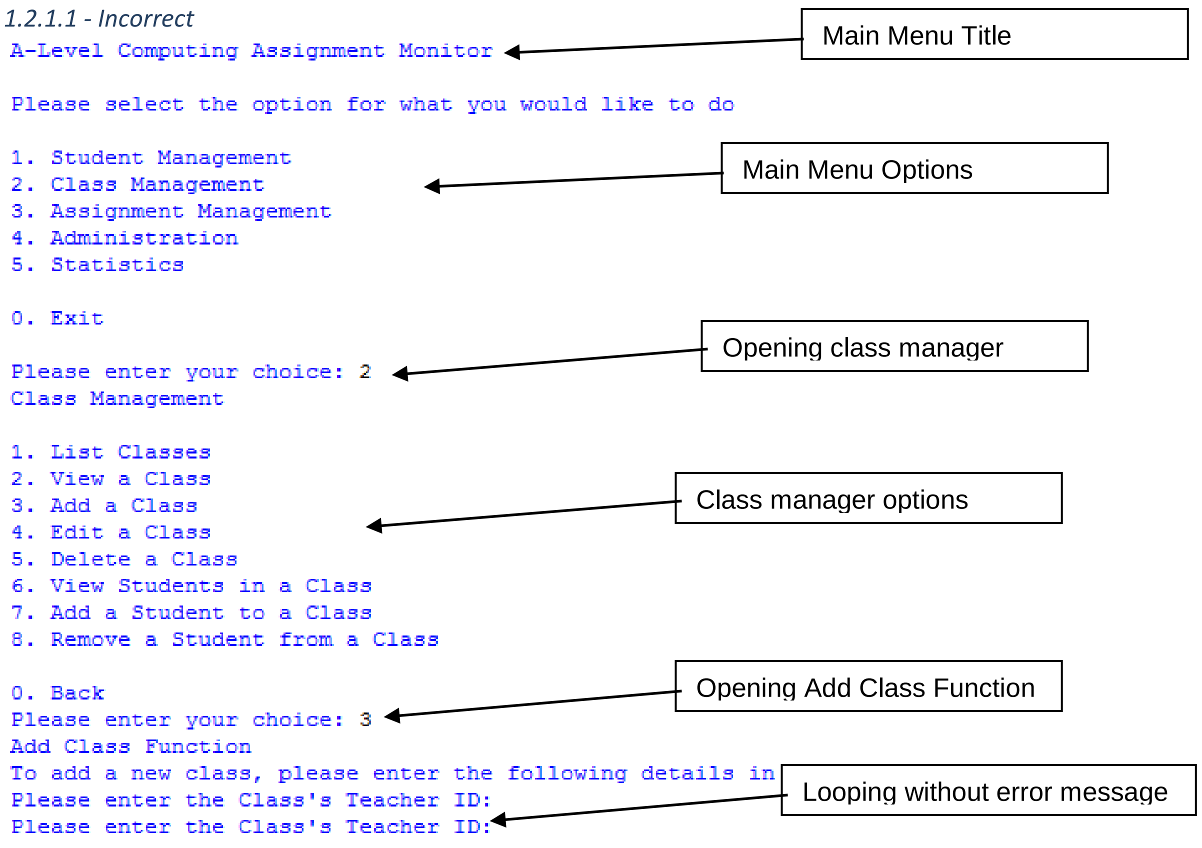


```
A-Level Computing Assignment Monitor

Please select the option for what you would like to do

1. Student Management
2. Class Management
3. Assignment Management
4. Administration
5. Statistics

0. Exit

Please enter your choice: ab
That is not a valid integer. Please try again
Please enter your choice:
```

Main Menu Title
Main Menu Options
Entering “text” as an options.
Rejected and asked the user again.

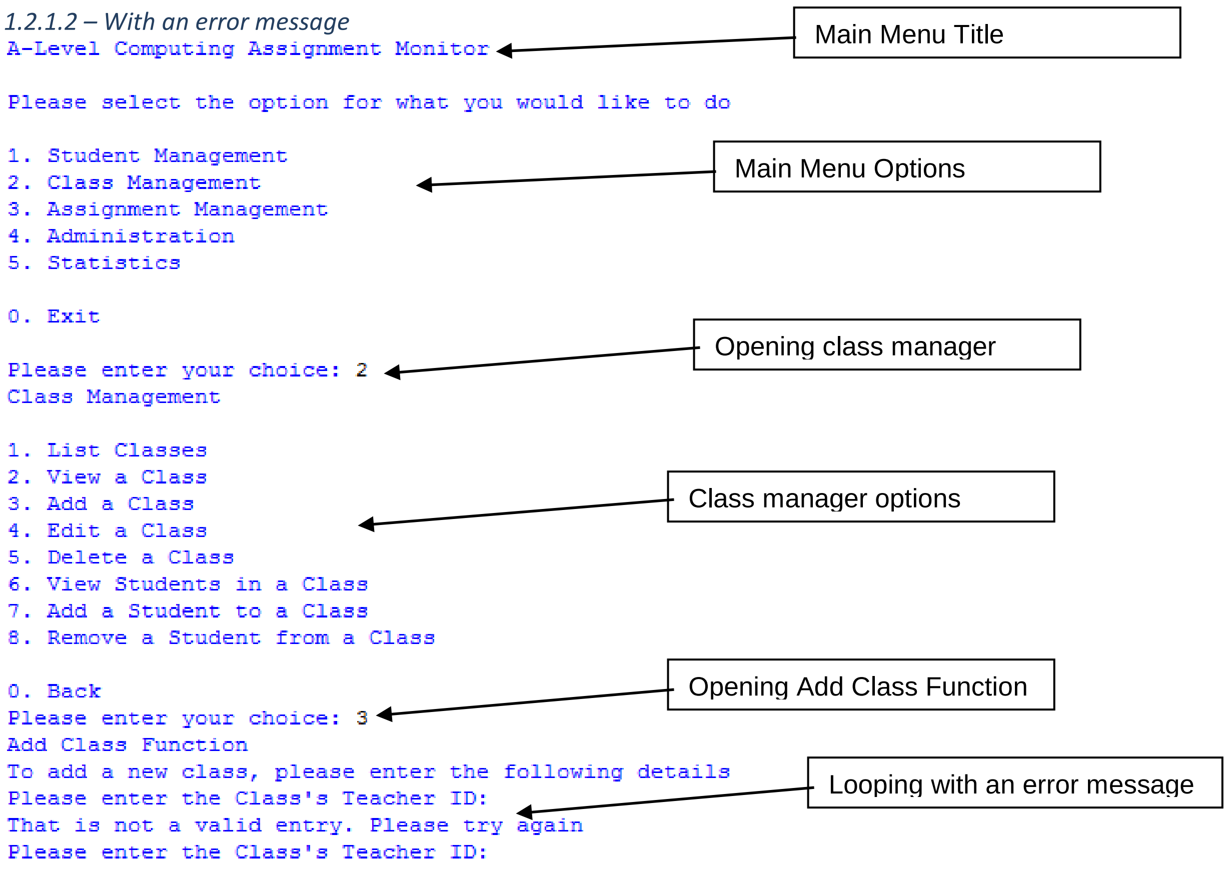
1.2.1 Integer entry – Without an error message

1.2.1.1 - Incorrect



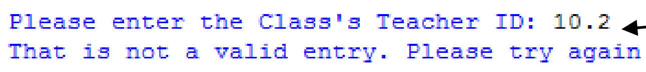
```
A-Level Computing Assignment Monitor

Please select the option for what you would like to do

1. Student Management
2. Class Management
3. Assignment Management
4. Administration
5. Statistics

0. Exit

Please enter your choice: 2
Class Management

1. List Classes
2. View a Class
3. Add a Class
4. Edit a Class
5. Delete a Class
6. View Students in a Class
7. Add a Student to a Class
8. Remove a Student from a Class

0. Back
Please enter your choice: 3
Add Class Function
To add a new class, please enter the following details in
Please enter the Class's Teacher ID:
Please enter the Class's Teacher ID:
```

1.2.1.2 – With an error message



```
A-Level Computing Assignment Monitor

Please select the option for what you would like to do

1. Student Management
2. Class Management
3. Assignment Management
4. Administration
5. Statistics

0. Exit

Please enter your choice: 2
Class Management

1. List Classes
2. View a Class
3. Add a Class
4. Edit a Class
5. Delete a Class
6. View Students in a Class
7. Add a Student to a Class
8. Remove a Student from a Class

0. Back
Please enter your choice: 3
Add Class Function
To add a new class, please enter the following details
Please enter the Class's Teacher ID:
That is not a valid entry. Please try again
Please enter the Class's Teacher ID:
```

1.2.2 Integer Entry – A Float Number

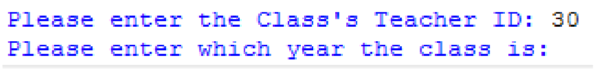


```
Please enter the Class's Teacher ID: 10.2
That is not a valid entry. Please try again
```

Entering an incorrect number

Error.

1.2.3 Integer Entry – An Integer



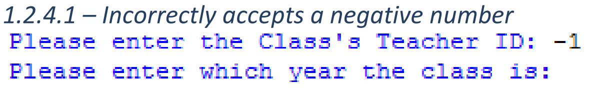
```
Please enter the Class's Teacher ID: 30
Please enter which year the class is:
```

Entering a correct number.

Correct entry, next question

1.2.4 – Negative number

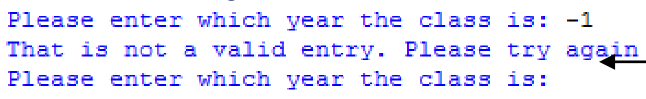
1.2.4.1 – Incorrectly accepts a negative number



```
Please enter the Class's Teacher ID: -1
Please enter which year the class is:
```

Incorrectly accepts a negative number.

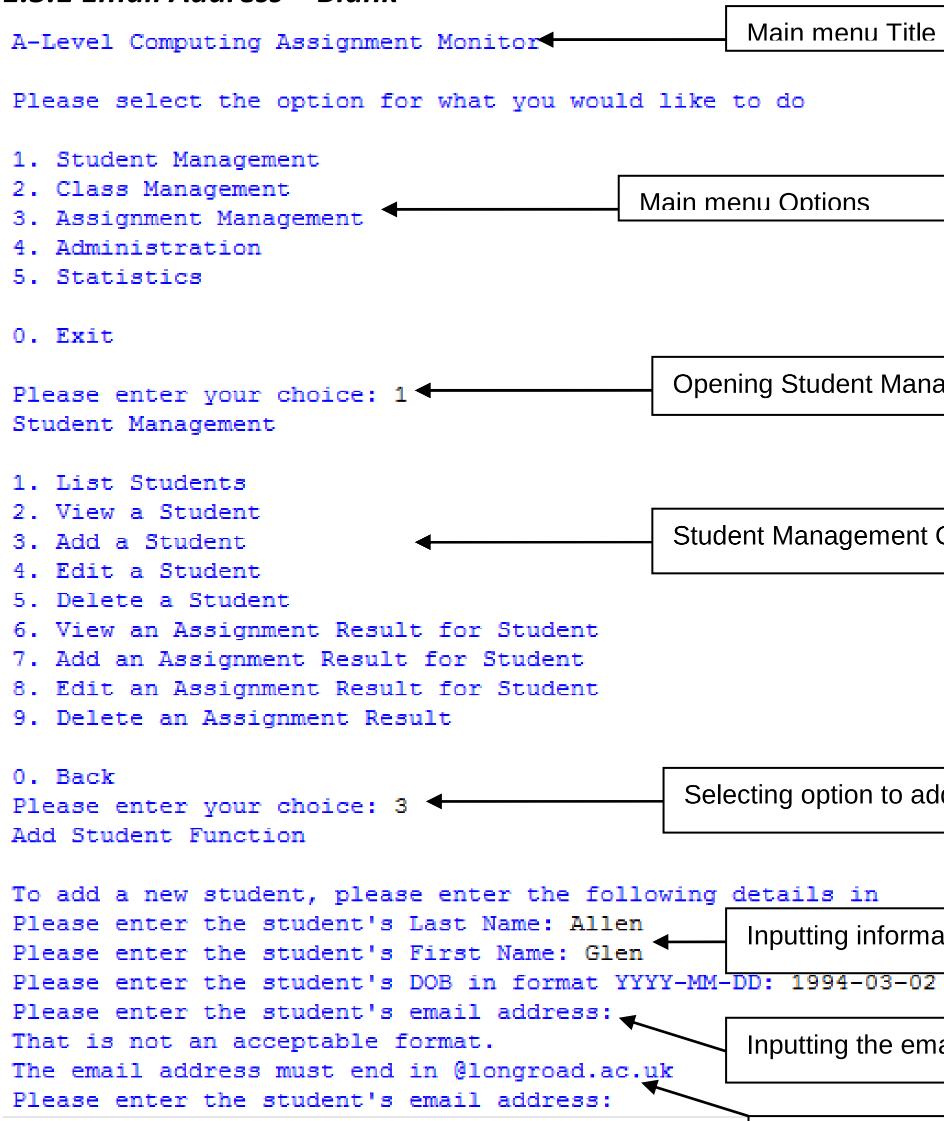
1.2.4.2 – Denies a negative number



```
Please enter which year the class is: -1
That is not a valid entry. Please try again
Please enter which year the class is:
```

Correctly denies negative numbers

1.3.1 Email Address – Blank



```
 A-Level Computing Assignment Monitor

 Please select the option for what you would like to do

 1. Student Management
 2. Class Management
 3. Assignment Management
 4. Administration
 5. Statistics

 0. Exit

 Please enter your choice: 1
 Student Management

 1. List Students
 2. View a Student
 3. Add a Student
 4. Edit a Student
 5. Delete a Student
 6. View an Assignment Result for Student
 7. Add an Assignment Result for Student
 8. Edit an Assignment Result for Student
 9. Delete an Assignment Result

 0. Back
 Please enter your choice: 3
 Add Student Function

 To add a new student, please enter the following details in
 Please enter the student's Last Name: Allen
 Please enter the student's First Name: Glen
 Please enter the student's DOB in format YYYY-MM-DD: 1994-03-02
 Please enter the student's email address:
 That is not an acceptable format.
 The email address must end in @longroad.ac.uk
 Please enter the student's email address:
```

Main menu Title

Main menu Options

Opening Student Management

Student Management Options

Selecting option to add a student.

Inputting information to add a student.

Inputting the email address

Error Message and asking the user again

1.3.2 Email Address – Incorrect ending

```
Please enter the student's email address: aaa@longroad.ac.ua
That is not an acceptable format.
The email address must end in @longroad.ac.uk
Please enter the student's email address:
```

Incorrect email

Error Message and asking the user again

1.3.3 Email Address – Correct

```
Please enter the student's email address: aaa@longroad.ac.uk
Does the student have a scribe? (Y/N)
```

Correct email

Accepted and moved on

1.3.4 Email Address – Subdomain, not email

```
Please enter the student's email address: adasd.longroad.com
That is not an acceptable format.
The email address must end in @longroad.ac.uk
Please enter the student's email address:
```

Incorrect email

Error Message and asking the user again

1.3.5 Email Address – Missing first part

```
Please enter the student's email address: @longroad.ac.uk
That is not an acceptable format.
The email address must end in @longroad.ac.uk
Please enter the student's email address:
```

Incorrect email

Error Message and asking the user again

1.4.1 Date – Empty

```
A-Level Computing Assignment Monitor

Please select the option for what you would like to do

1. Student Management
2. Class Management
3. Assignment Management
4. Administration
5. Statistics

0. Exit

Please enter your choice: 1
Student Management

1. List Students
2. View a Student
3. Add a Student
4. Edit a Student
5. Delete a Student
6. View an Assignment Result for Student
7. Add an Assignment Result for Student
8. Edit an Assignment Result for Student
9. Delete an Assignment Result

0. Back
Please enter your choice: 3
Add Student Function

To add a new student, please enter the following details in
Please enter the student's Last Name: Allen
Please enter the student's First Name: Glen
Please enter the student's DOB in format YYYY-MM-DD:
That is not an acceptable format for a date

Please enter the student's DOB in format YYYY-MM-DD:
```

Main Menu Title

Main Menu Options

Loading Student Manager

Loading Add Student Function

Added first few details in

Nothing entered. Rejected and asked again.

1.4.2 Date – Wrong format

```
Please enter the student's DOB in format YYYY-MM-DD: 21/33/1993
 That is not an acceptable format for a date
Please enter the student's DOB in format YYYY-MM-DD:
```

Wrong date format entered, rejected and asked again.

1.4.3 Date - Acceptable input

```
Please enter the student's DOB in format YYYY-MM-DD: 1994-04-03
Please enter the student's email address:
```

Correct date format, accepted and moved on

1.4.4 Date - Extreme Input

```
Please enter the student's DOB in format YYYY-MM-DD: 1003-11-12
Please enter the student's email address:
```

Correct format, but too far out. Should fail, but accepted.

1.4.5 Date - Error – Wrong format

```
Please enter the student's DOB in format YYYY-MM-DD: 12121993
That is not an acceptable format for a date
Please enter the student's DOB in format YYYY-MM-DD:
```

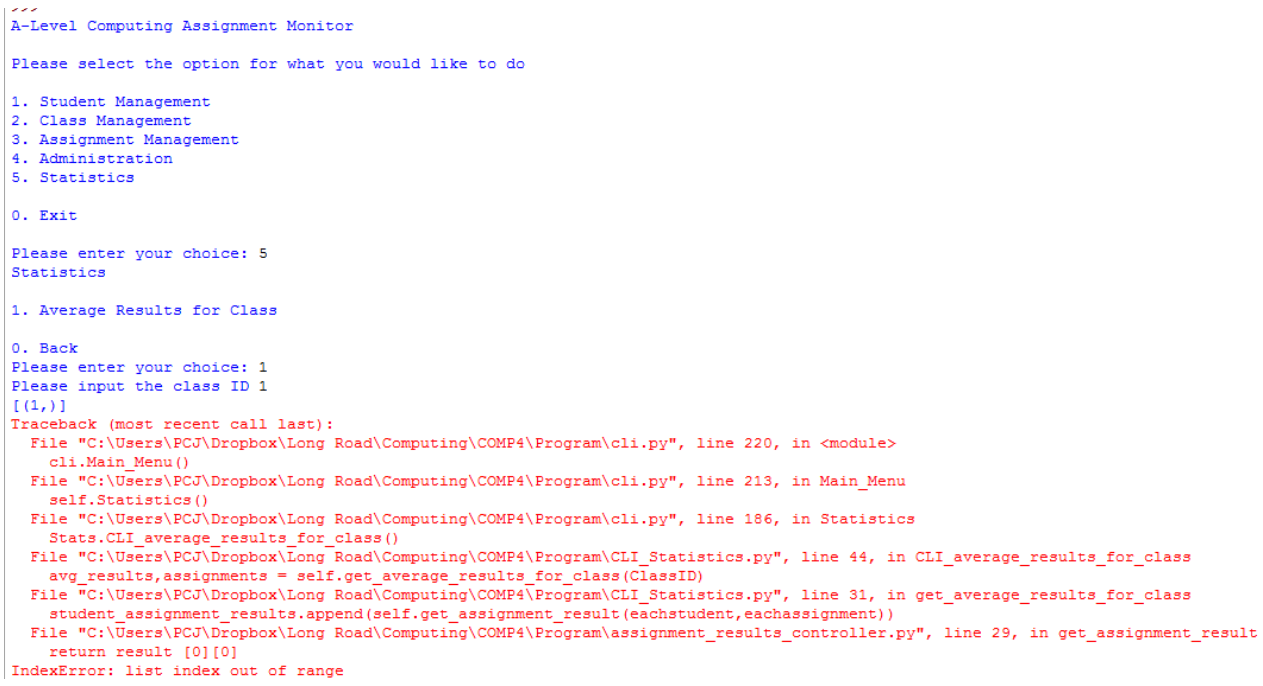
Wrong format, rejected and asked again.

1.4.6 Date - Error – Text

```
Please enter the student's DOB in format YYYY-MM-DD: Date
That is not an acceptable format for a date
Please enter the student's DOB in format YYYY-MM-DD:
```

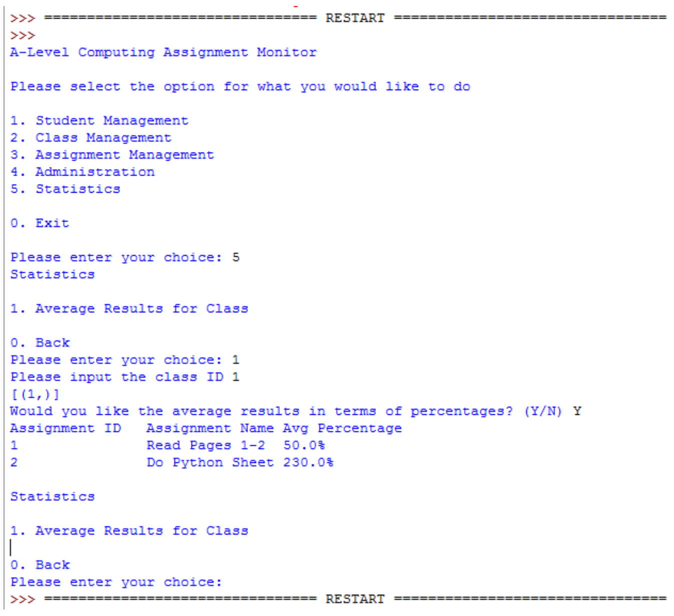
Wrong format, rejected and asked again.

Statistics – Before



```
A-Level Computing Assignment Monitor

Please select the option for what you would like to do

1. Student Management
2. Class Management
3. Assignment Management
4. Administration
5. Statistics

0. Exit

Please enter your choice: 5
Statistics

1. Average Results for Class

0. Back
Please enter your choice: 1
Please input the class ID 1
[(1,)]
Traceback (most recent call last):
  File "C:\Users\PCJ\Dropbox\Long Road\Computing\COMP4\Program\cli.py", line 220, in <module>
    cli.Main_Menu()
  File "C:\Users\PCJ\Dropbox\Long Road\Computing\COMP4\Program\cli.py", line 213, in Main_Menu
    self.Statistics()
  File "C:\Users\PCJ\Dropbox\Long Road\Computing\COMP4\Program\cli.py", line 186, in Statistics
    Stats.CLI_average_results_for_class()
  File "C:\Users\PCJ\Dropbox\Long Road\Computing\COMP4\Program\CLI_Statistics.py", line 44, in CLI_average_results_for_class
    avg_results,assignments = self.get_average_results_for_class(ClassID)
  File "C:\Users\PCJ\Dropbox\Long Road\Computing\COMP4\Program\CLI_Statistics.py", line 31, in get_average_results_for_class
    student_assignment_results.append(self.get_assignment_result(eachstudent,eachassignment))
  File "C:\Users\PCJ\Dropbox\Long Road\Computing\COMP4\Program\assignment_results_controller.py", line 29, in get_assignment_result
    return result [0][0]
IndexError: list index out of range
```

Statistics – After



```
>>> ================================ RESTART ================================
>>>
A-Level Computing Assignment Monitor

Please select the option for what you would like to do

1. Student Management
2. Class Management
3. Assignment Management
4. Administration
5. Statistics

0. Exit

Please enter your choice: 5
Statistics

1. Average Results for Class

0. Back
Please enter your choice: 1
Please input the class ID 1
[(1,)]
Would you like the average results in terms of percentages? (Y/N) Y
Assignment ID   Assignment Name Avg Percentage
1               Read Pages 1-2  50.0%
2               Do Python Sheet 230.0%

Statistics

1. Average Results for Class

0. Back
Please enter your choice:
>>> ================================ RESTART ================================
```

4. Evaluation

4.1. Approach to testing

My testing was split up slightly. I did basic testing of each of the functions as they were being written, i.e. Bottom –up, to ensure that they were working, I then applied white-box testing to the program using other small tests to ensure the program will not crash if the user accidently used the wrong formatting.

4.2. Problems

I found a few problems in my testing.

The main error, interface wise, I had with my program was the fact that it did not produce reasons for some of the errors although it did not crash. I simply added some print statements to rectify this.

I also found out that negative numbers were allowed, which were also rectified by adding more to the IF statement.

The last issue that I found was that the program accepted dates that were implausible. An example of this can be seen in 1.4.4 on page 69. This has been rectified by checking that the integer of the first 4 digits are in a selected range.

4.3. Strengths

I found that the testing strategy worked well with time constraints. This is because it allows me to make sure that the program functionality works as it should, and then use white-box testing to do all the testing at the same time, which felt to work much faster.

4.4. Weaknesses

I felt that rather than testing it from the actual program, it would have been quicker if I did much more of the testing from the individual modules as opposed to following the menu along.

4.5. Reliability

I found that when the users would follow the user manual, or the correct procedure, that the program will be fairly reliable, but if they stumble into doing something out of order, then the program will crash and require the user to fix the problem. The program should, instead, either fill in for the rest of the numbers (i.e. 0s for Assignments that are not filled in) or cleanly exit with an explanation.

4.6. Robustness
Data Entry for the program is extremely robust and the program will not fail at these stages, although there is no way to cancel data input.

If the system is used out of order, or if an assignment result has not yet been filled in for **all** of the assignments, then the program will crash.

System Maintenance

1. Environment

1.1 Software

- Python 3.2 (or 3.3)
  - IDLE for writing the code
  - RE module for Regular Expressions
  - DateTime module for processing dates
  - SQLite3 module for database interaction
  - SMTPLib for email
- PyQt for GUI
  - Designer
- SQLite Database Browser

1.2 Usage Explanation

- Python 3.2/3.3
  - Python Language
    - I used the Python language because it is both the only language that I know, but also because it is an extremely simple and easy language to use when compared to the likes of Java or C.
  - IDLE
    - Allows for easy coding in Python
    - Provides code highlighting for easier viewing
    - Provides a Class Browser to avoid me having to scroll
    - Provides limited syntax error checking to avoid problems when the code actually launches.
  - RE Module
    - Used to check strings of data to see if they match a pattern
  - Datetime Module
    - Used to convert between strings and datetime variables
    - Can get the current date and time
    - Can add and subtract date/times

- SQLite3 Module
  - Allows me to store and manipulate data in a database
  - Very light solution. Simpler than MySQL
  - Allows use of foreign keys and cross-referencing of tables.
- SMTPLib
  - Allows me to send emails to the students or classes via use of a SMTP server.
- PyQT
  - Popular library for GUI programs.
  - Some easy solutions to customising the widgets
  - Designer
    - Designer let me easily create mock-ups of what I wanted the GUI to look like
    - Used the correct names for the widgets.
- SQLite Database Browser
  - Allows me to easily view the tables
  - Allows me to view the information used to create the tables to ensure they are accurate.
  - Allows me to add, edit and delete tables in the database.

1.3 Features Used

- Python 3.2/3.3
  - IDLE
    - I used the syntax highlighting to help make sure I have typed keywords in correctly, because I find that I may often accidently do typos that I may miss.
    - I used the find and replace for when I decided to change a variables name because it is much easier than hunting for it. It takes less effort to do so and it's much less likely to miss one.
    - I used the class browser for easily moving to a class or function that I wanted to look at because it allows me to easily see what functions I have already made. Being able to scroll instantly to a function helps out a lot, especially in longer program files.
    - I used the automated error checking to help prevent errors earlier rather than having to look for them later, because it is very likely to miss a line that has a slight error in it that I wouldn't normally spot.
    - I used the console to test small lines from directly within python because it is much easier to type in the few lines directly into the console rather creating a new .py file and loading that.
  - RE Module

        - I used the patter match function to check a string against a set pattern because it is an extremely easy way to do advanced validation.
    - Datetime module
        - I used the strptime function to convert a string to a date variable to ensure it is a valid input because I know that if the program will accept a date, then it is much more failproof than a method via string testing.
        - I planned to use the datetime module to also take differences to ensure I do email a student too often because this is the easiest way to take differences between two dates, which could then be easily checked if it's less or more than.
    - SQLite3
        - I used the referential integrity ability to ensure no data can be deleted without having the data it is referenced by deleted first. This is because it can leave behind problematic records and cause the user problems with trying to handle data when part of it no longer exists, if they deleted it without knowing that they still had data referencing it.
        - I used the foreign keys ability to allow me to use the same variables between tables because it allows me to split the tables up into separate ones, thereby saving the need to repeat data, thereby saving disk space.
    - SMTPLib
        - I planned to use this module for emailing the students automatically in batches because it would enable the user to directly send the results of their assignment to the student(s) without needing much effort.
- PyQT
    - I used the included widgets and their settings to customise the widgets to how they fit in with my design, such as when creating the login window, I wanted to not have the password visible as it is being typed.
    - I used the included designer to use the correct widget names and create a clear design for me to follow. This made more sense to me because it allowed me to create printable diagrams that looked realistic for my design, as well as creating files that I can easily open that would produce a list of all the widgets that I need to create.
- SQLite Database Browser
    - I used the ability to view tables to see if my testing went through because it is much easier than creating SQL codes to manually show. It also is much easier to look at.
    - I used the ability to add rows and data to the tables to quickly add data that is needed for testing.

2. System Overview

The system is primarily a database, and is split into several sections, each handling their own section of storage. To monitor a Sixth Form College's department's assignment history, it needs to be split into several sections. It needs to keep a record of all the assignments for both the years, the students that do them, what results they get, which classes they're in and which teachers teach those classes and have access to the program.

2.1 Student Management

The student management class is responsible for viewing, adding, editing and deleting the students stored within the database. It is also responsible for managing the results that the student receives from their assignments.

2.2 Class Management

The class manager class is responsible for managing the classes, by viewing, adding, editing and deleting them. But since a class is useless without students, this section also covers adding and removing students from classes.

2.3 Assignment Management

The assignment management class is responsible for managing the assignments by adding, editing, deleting.

2.5 Statistics

This section is separate from the rest as it does not manage any data, but rather manipulate it to "tell a tale". Here, the user can find out the average results his/her class are getting on their assignments.

3. Code Structure

My code structure is mostly uniform throughout. I have used classes, and inheritance with them, and functions to make the program easier. Examples of this include:

3.1 Questioning and Validation

When I am asking my user to enter data, while I am writing out a line for each variable, I am using a function that only needs the variable name and why. This allows me to use the same function for adding, deleting and editing, where I am able to ensure that some inputs are required and others are not where applicable.
For example, this line (10.11 page 144 line 185) is much easier to use and repeat:

```
StudentLastName = self.student_variable("StudentLastName","add")
```

This line thereby calls the student_variable function, which in turn calls the `get_student_question` and `check_student_variable` functions.

3.2 Queries to the Database

Within my controllers, I used an inherited class with 2 private functions that allow me to easily and cleanly perform a query on the database, without typing out the code several times. The function is private because I do not want others to be able to do anything to the database except for what I have already set.
For example, rather than typing out the following:

```
self.db = sqlite3.connect(self.dbname)
self.cursor = self.db.cursor()
self.cursor.execute("PRAGMA foreign_keys = ON")
self.cursor.execute(sql)
self.db.commit()
self.cursor.close()
```

I only need to type out:

```
self._query(sql)
```

Due to inheriting the controller class, I only need to write self, rather than needing to instantiate another copy.

3.3 Printing Tables

In my controllers, I used a function to print my tables for each of my list or view functions. This is because it varies slightly between classes, due to lengths of names, but is mostly the same.

This is mostly because it simply clears up the view and list functions, where examples of which can be seen in the student manager starting on page 144 on lines 170 through 200.

```
    def print_table(self,headings,data):
        #This function prints a table for a list/view function
        #Sets up a blank array for the attributes
        table_attributes = []
        #Processes each heading
        for each in headings:
            #Puts it into a variable to simplify things
            heading = each[1]
            #Removes the first several characters
            heading = heading[7:]
            #Appends it to the table_attributes list
            table_attributes.append(heading)
        #Creates a blank headings string
        headings = ""
        #Processes each attribute
        for count in range(len(table_attributes)):
            #Calculates the length of the text and adds 5 for
visibility
            length = len(table_attributes[count]) + 5
            #Adds additional space for easy viewing for email address
            if count == 4:
                headings = headings + '{0[' + str(count) + ']:<21}'
            else:
                headings = headings + '{0[' + str(count) + ']:<13}'
        #Prints the headings
        print(headings.format(table_attributes))
         #Processes each row
        for each in data:
            #Creates a blank string for the row
            result = ""
            #Processes each "cell"
            for count in range(len(each)):
                #constructs the "cell", adding additional space for
email address
                if count == 4:
                   result = result + '{0[' + str(count) + ']:<21}'
                else:
                    result = result + '{0[' + str(count) + ']:<13}'
            #Prints the row
            print(result.format(each))
```

4. Variable Listing

Variable Name	Purpose	Instances
sql	Used as a temporary store for the sql code generated needed to perform a query on the database	10.1: 20, 26, 54, 68, 61, 63, 66, 70, 71, 100, 106, 109, 110, 114, 115, 119, 120, 130, 132. 10.2: 19, 23, 27, 29, 47, 48, 52, 55, 57, 60, 64, 66. 10.3: 19, 23, 42, 46, 49, 51, 54, 58, 59, 80, 86, 89, 90, 99, 100, 10.4: 19, 23, 31, 36, 40, 41, 59, 65,

		68, 69, 70, 74, 75. 10.12: 12, 21, 27, 36. 10.13: 6, 20, 25, 39, 45, 54, 58, 67, 71, 78, 82, 88, 10.18: 21, 30, 65, 69, 72, 74, 75, 80, 81, 117, 123, 126, 131, 132. 10.19: 23, 32, 68, 72, 75, 77, 79, 83, 84, 112, 117, 120, 121, 127, 131, 147, 148,
changes	Used as a temporary store for the variables in the database that need to be changed during an edit function	10.1: 34, 40, 42, 44, 46, 48, 50, 56. 10.2: 38, 44, 46, 50. 10.3: 30, 36, 38, 40, 44. 10.18: 49, 45, 47, 49, 51, 53, 55, 57, 59, 61, 63, 67. 10.19: 41, 47, 50, 52, 55, 57, 59, 61, 63, 65, 70.
parameters	Same as above, but used for viewing certain records.	10.1: 79, 84, 86, 88, 90, 92, 94, 96, 105. 10.3: 66, 71, 73, 75, 77, 85. 10.4: 48, 53, 55, 64. 10.18: 90, 95, 97, 99, 101, 103, 105, 107, 109, 111, 113, 122. 10.19: 92, 97, 99, 101, 103, 105, 107, 116
results	Used to store the output from the database query prior to processing.	10.1: 121, 124, 132, 136. 10.3: 100, 101, 102. 10.4: 75, 77. 10.10: 67, 71. 10.12: 38, 42.
option	Options that the user picked in the menus	10.5: 101, 103, 105, 135, 137, 140, 143, 145, 147, 149, 151, 153, 155, 157, 182, 187, 190, 192, 194, 196, 198, 200, 202, 221, 223, 226, 229, 231, 233, 235, 254, 256, 262, 264, 266, 268, 270, 285, 287, 290, 312, 313, 316, 318, 320, 322, 324, 10.9: 17, 18, 19. 10.11: 20, 22, 24,
heading(s)	Used when printing a table to store the table headings.	10.6: 13, 18, 26, 33, 36, 38, 106-112, 126-132, 166-172. 10.7: 58, 63, 71, 78, 80, 82, 101, 102, 122, 123, 151, 152, 10.9: 78, 83, 89, 94, 96, 123, 124, 140, 141, 167, 168. 10.10: 79, 90, 93, 106, 116, 119. 10.11: 131, 136, 144, 151, 153, 155, 175, 176, 198, 199, 238, 239,

Table_attributes	Used when printing a table to determine how many characters to use as a margin.	10.6 16, 24, 28, 30, 38. 10.7: 61, 69, 73, 75, 82. 10.9: 81, 87, 91, 93, 96. 10.11: 134, 142, 145, 148, 155,
each	This is only used in for loops where it is using items from a list, where "each" is the current item.	10.1: 124, 125, 136, 137. 10.4: 77, 78. 10.6: 18, 20, 40, 44, 51. 10.7: 63, 65, 84, 88, 95. 10.8: 83, 85, 98, 102, 106, 208, 209. 10.10: 71, 72, 83, 84, 85, 95, 99, 103, 110, 111, 121, 125, 129. 10.11: 136, 138, 157, 161, 168,
haveID And it's derivatives: haveStudentID haveAssignmentID haveClassID	Used as a temp store to determine if the user has an ID or not	10.6: 158, 159, 195, 196, 10.7: 143, 144, 173, 174, 10.8: 58, 59, 67, 68, 95, 96, 104, 105, 125, 126, 134, 135 10.9: 159, 160, 181, 182, 195, 196, 224, 225, 230, 231, 249, 250, 263, 264,
ID	Used to temporarily store an ID for an edit or a delete function	10.6: 163, 165, 200, 202, 10.7: 148, 151, 178, 180, 10.9: 164, 166, 186, 188, 200, 202, 10.11: 235, 237, 268, 269,
IDs	An array of IDs generated from an SQL statement. Mostly used for getting Assignment IDs.	10.1: 123, 126, 135, 137, 138,
data	Used to temporarily store data for printing tables in view/list functions.	10.6: 13, 40, 105, 112, 123, 132, 165, 172, 10.7: 58, 84, 100, 102, 118, 123, 151, 153, 10.9: 78, 98, 122, 124, 138, 141, 166, 168, 10.10: 28, 38, 50, 81, 85, 95, 108, 121, 10.11: 131, 157, 174, 176, 195, 199,237, 239,
Question	Used to store the question	10.6: 95, 98, 10.7: 49, 52, 10.8: 46, 49 10.9: 110, 113 10.11: 124, 127
Variable	Used to temporarily identify which variable the program is after	10.6: 55, 57, 59, 61, 63, 65, 67, 69, 71, 73, 75, 78, 84, 93, 95, 99, 10.7: 12, 14, 16, 18, 20, 22, 24, 26, 30, 36, 47, 49, 53, 10.8: 15, 17, 19, 21, 23, 27, 33, 41, 44, 46, 50, 51, 10.9: 27, 29, 31, 33, 35, 38, 44, 53, 61, 69, 108, 110, 114, 10.11: 34, 36, 38, 40, 42, 44, 46, 48, 50, 52, 54, 56, 59, 66, 73, 79, 85, 93, 101, 108, 115, 122, 124, 128,
Function	Used to determine what the program is wanting to do for the verification	10.6: 78, 81, 93, 99, 10.7: 30, 33, 47, 53 10.8: 27, 30, 44, 50, 51, 10.9: 48, 41,108, 114, 10.11: 59, 63, 122, 128,

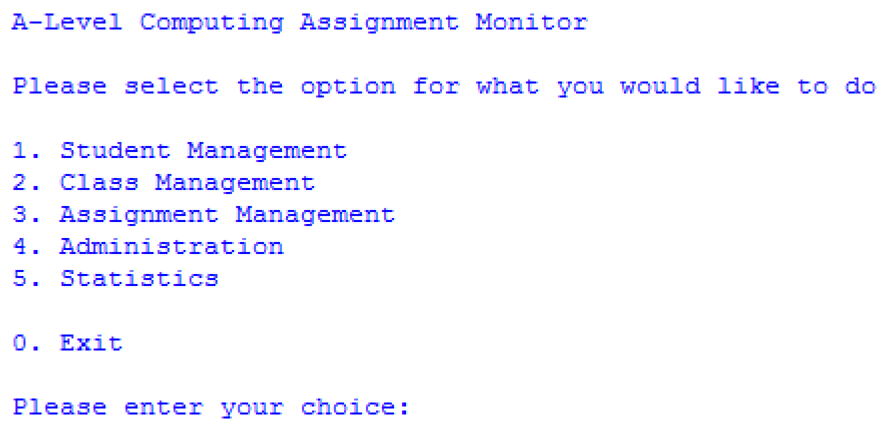
valid		10.5: 95, 96, 10.6: 96, 97, 99, 10.7: 50, 51, 53, 54, 10.8: 47, 48, 51, 10.9: 111, 112, 114, 115 10.11: 125, 126, 128 10.16: 60, 62
Entry	Used as a temp variable while verifying a user's input	10.6: 78, 81, 86, 91, 98, 99, 100, 10.7: 30, 33, 40, 41, 45, 52, 53, 56 10.8: 27, 30, 36, 37, 39, 42, 49, 50, 51, 52, 10.9: 38, 41, 47, 50, 55, 56, 58, 63, 64, 65, 71, 72, 74, 113, 114, 117, 10.11: 59-118, 127, 128, 129,
year	Stores the year of the class that it is getting the average results for	10.10: 22, 24
Assignments	Stores the assignments that apply to that year	10.10: 24
students	Stores the students in a class	10.10: 26
student_assignment_results	Stores ta students assignment result scores	10.10: 32, 36
Total_results	Used to store the summed up results	10.10: 40, 45, 49
average_results	Used to store the results from total_results averaged.	10.10: 41, 52
In_percentage	Used to determine if the user wants the statistics in percentages or numbers	10.10: 61,77
Templist	Used to temporarily store data while processing for printing	10.10: 69, 72, 74, 76
Dbname	Stores the filename of the database	10.12: 10, 16, 31
Smtpserver	Email server used to send emails	10.14: 6
smtpuser	Username for email server	10.14: 8
smtppass	Password for email server	10.14: 10
Smtpresult	Result of the sending email	10.14: 16,17

5. System Evidence

5.1 User Interface

5.1.1 Main Menu

This screenshot shows the CLI main menu for the program, with options to go to the sub-menus and exit the program.

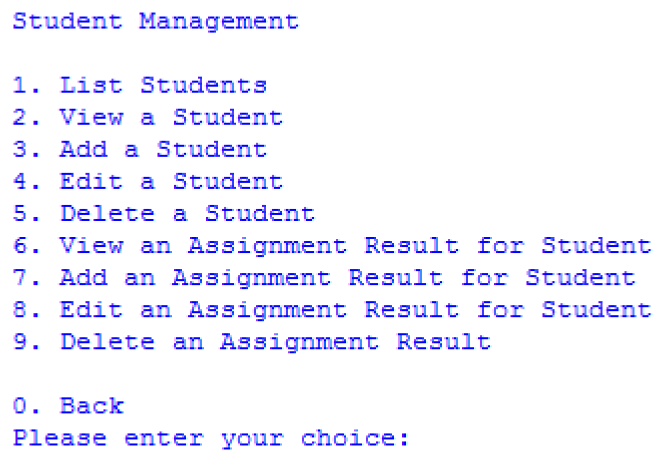


```
A-Level Computing Assignment Monitor

Please select the option for what you would like to do

1. Student Management
2. Class Management
3. Assignment Management
4. Administration
5. Statistics

0. Exit

Please enter your choice:
```

5.1.2 Student Management

5.1.2.1 Menu

This screenshot shows the Student Management menu for the program, allowing the user to pick one of the options or go back to the main menu



```
Student Management

1. List Students
2. View a Student
3. Add a Student
4. Edit a Student
5. Delete a Student
6. View an Assignment Result for Student
7. Add an Assignment Result for Student
8. Edit an Assignment Result for Student
9. Delete an Assignment Result

0. Back
Please enter your choice:
```

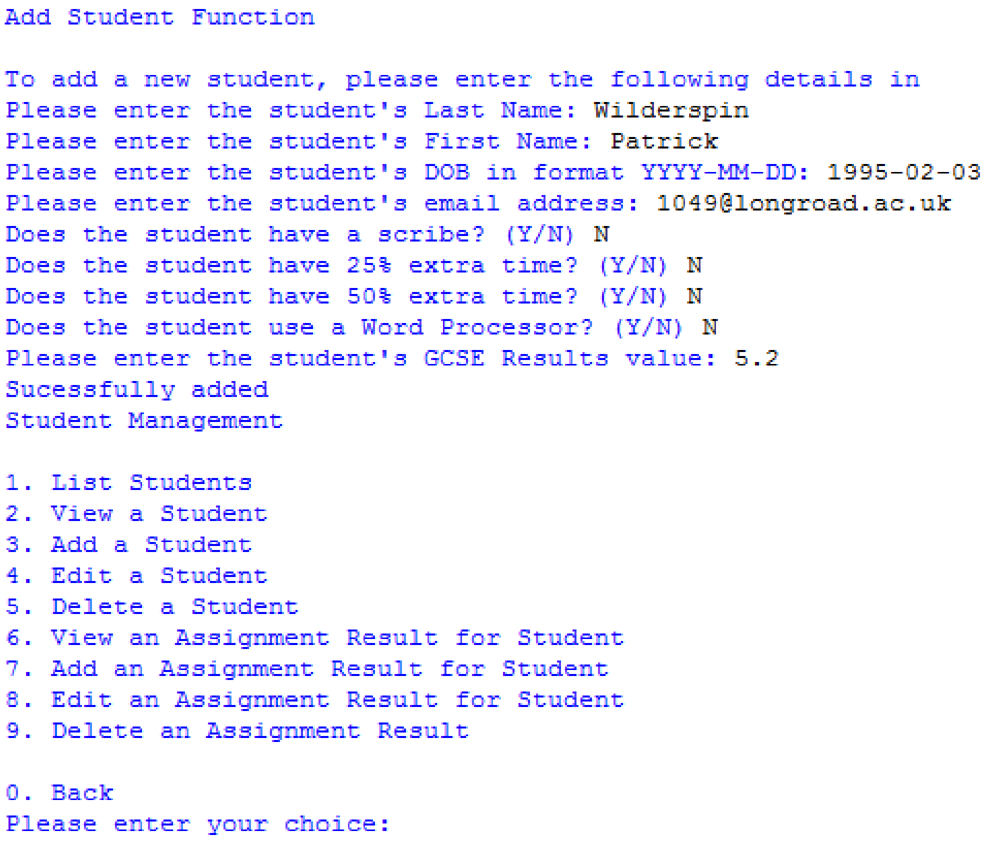
5.1.2.2 List Students

```
List Students
ID           LastName     FirstName    DOB          EMail                Scribe       25Extra      50Extra      WordProcessorGCSEResults  LastEmailed  Notes
1            Barham       Michael      1993-12-12   1341@longroad.ac.uk  True         True         False        False        4.3          0000-00-00
2            Wilderspin   Patrick      1995-02-03   1049@longroad.ac.uk  False        False        False        False        5.2          0000-00-00
3            Overhill     Ben          1994-10-15   1044@longroad.ac.uk  False        False        False        False        6.5          0000-00-00
4            Singer       Billy        1995-06-04   4022@longroad.ac.uk  False        False        False        False        5.4          0000-00-00
Student Management

1. List Students
2. View a Student
3. Add a Student
4. Edit a Student
5. Delete a Student
6. View an Assignment Result for Student
7. Add an Assignment Result for Student
8. Edit an Assignment Result for Student
9. Delete an Assignment Result

0. Back
Please enter your choice:
```

5.1.2.3 View a Student

```
View Student Function
Please input the details that you would like to use to find the student.
You can leave any value blank if you do not know it

Please enter the student's ID:
Please enter the student's Last Name:
Please enter the student's First Name:
Please enter the student's DOB in format YYYY-MM-DD:
Please enter the student's email address:
Does the student have a scribe? (Y/N) N
Does the student have 25% extra time? (Y/N)
Does the student have 50% extra time? (Y/N)
Does the student use a Word Processor? (Y/N)
Please enter the student's GCSE Results value:
ID           LastName     FirstName    DOB          EMail                Scribe       25Extra      50Extra      WordProcessorGCSEResults  LastEmailed  Notes
2            Wilderspin   Patrick      1995-02-03   1049@longroad.ac.uk  False        False        False        False        5.2          0000-00-00
3            Overhill     Ben          1994-10-15   1044@longroad.ac.uk  False        False        False        False        6.5          0000-00-00
4            Singer       Billy        1995-06-04   4022@longroad.ac.uk  False        False        False        False        5.4          0000-00-00
Student Management

1. List Students
2. View a Student
3. Add a Student
4. Edit a Student
5. Delete a Student
6. View an Assignment Result for Student
7. Add an Assignment Result for Student
8. Edit an Assignment Result for Student
9. Delete an Assignment Result

0. Back
Please enter your choice:
```

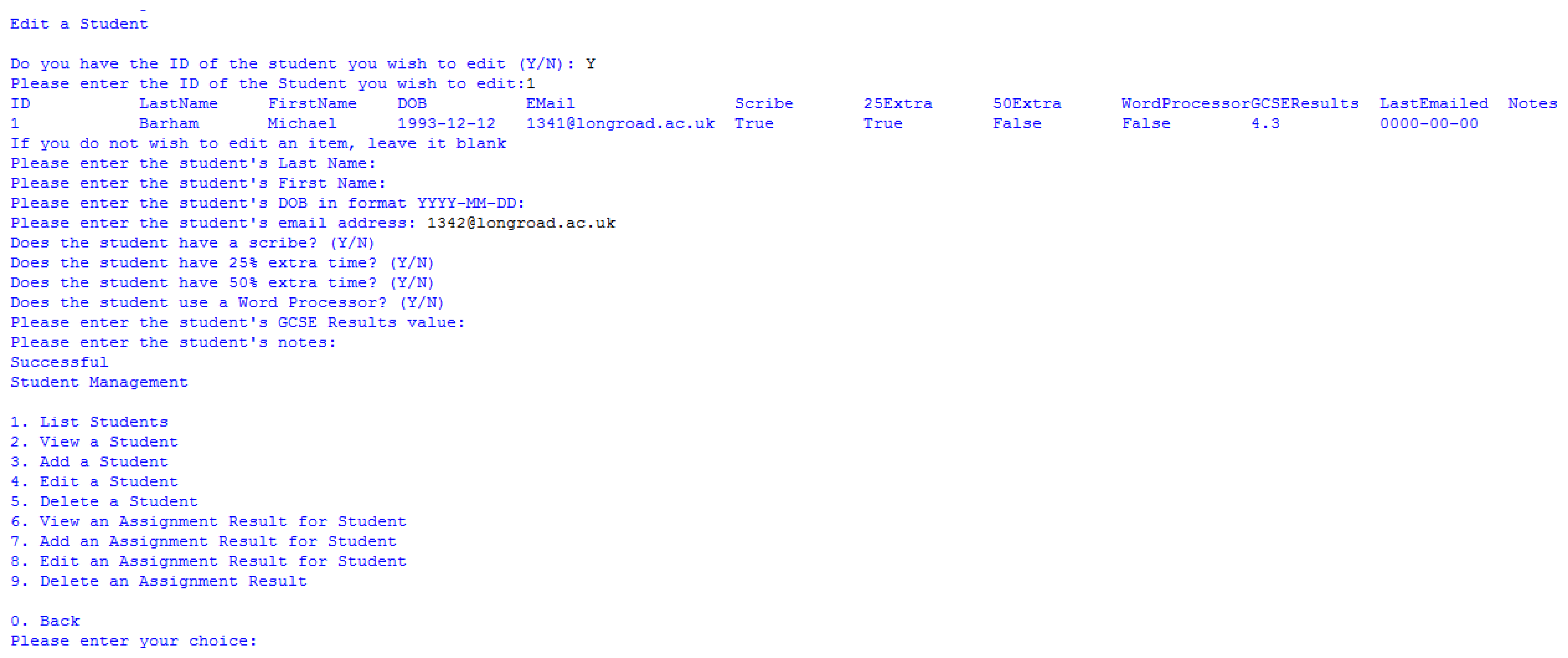
5.1.2.4 Add a Student



```
Add Student Function

To add a new student, please enter the following details in
Please enter the student's Last Name: Wilderspin
Please enter the student's First Name: Patrick
Please enter the student's DOB in format YYYY-MM-DD: 1995-02-03
Please enter the student's email address: 1049@longroad.ac.uk
Does the student have a scribe? (Y/N) N
Does the student have 25% extra time? (Y/N) N
Does the student have 50% extra time? (Y/N) N
Does the student use a Word Processor? (Y/N) N
Please enter the student's GCSE Results value: 5.2
Sucessfully added
Student Management

1. List Students
2. View a Student
3. Add a Student
4. Edit a Student
5. Delete a Student
6. View an Assignment Result for Student
7. Add an Assignment Result for Student
8. Edit an Assignment Result for Student
9. Delete an Assignment Result

0. Back
Please enter your choice:
```

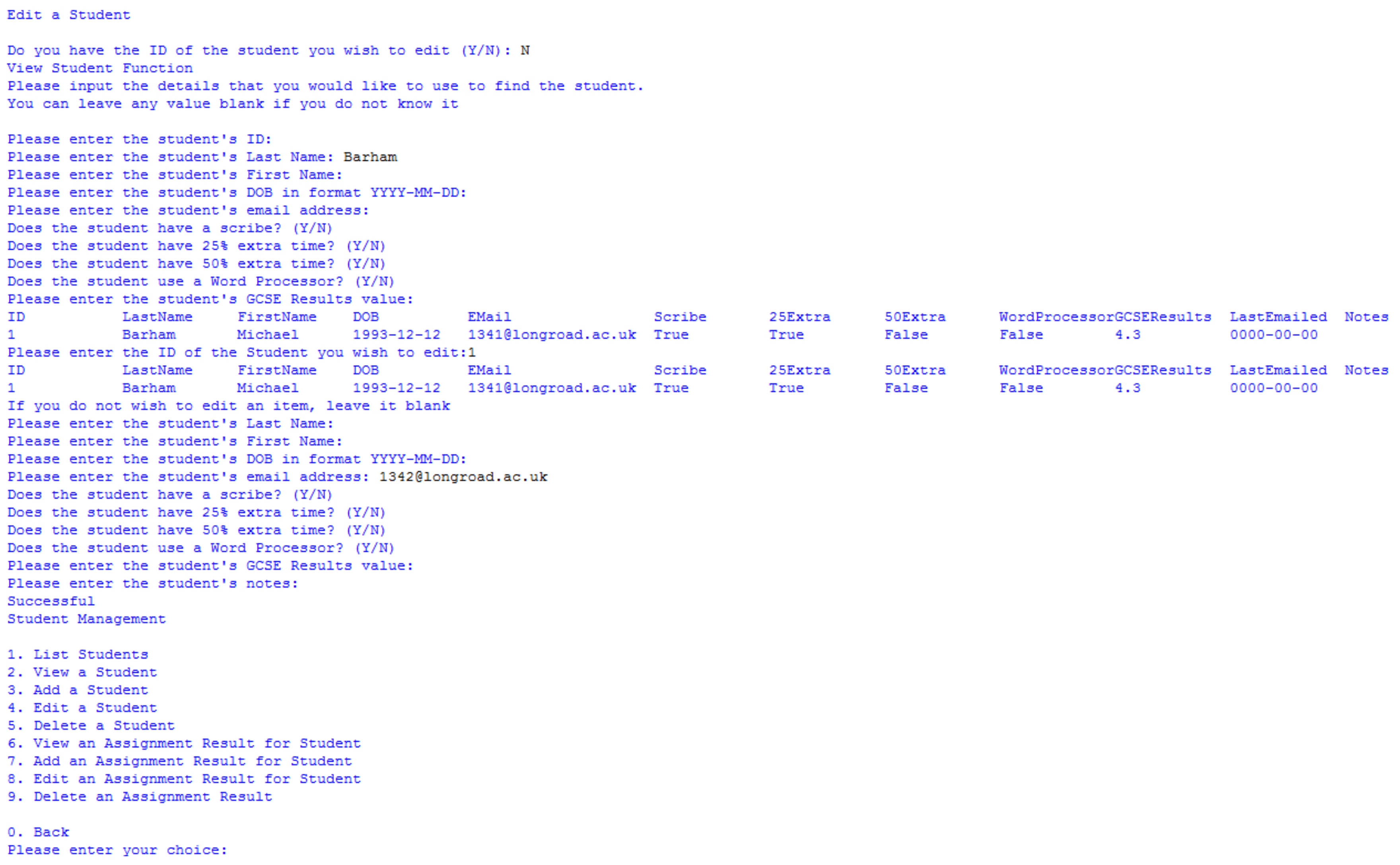
5.1.2.5 Edit a Student

With ID



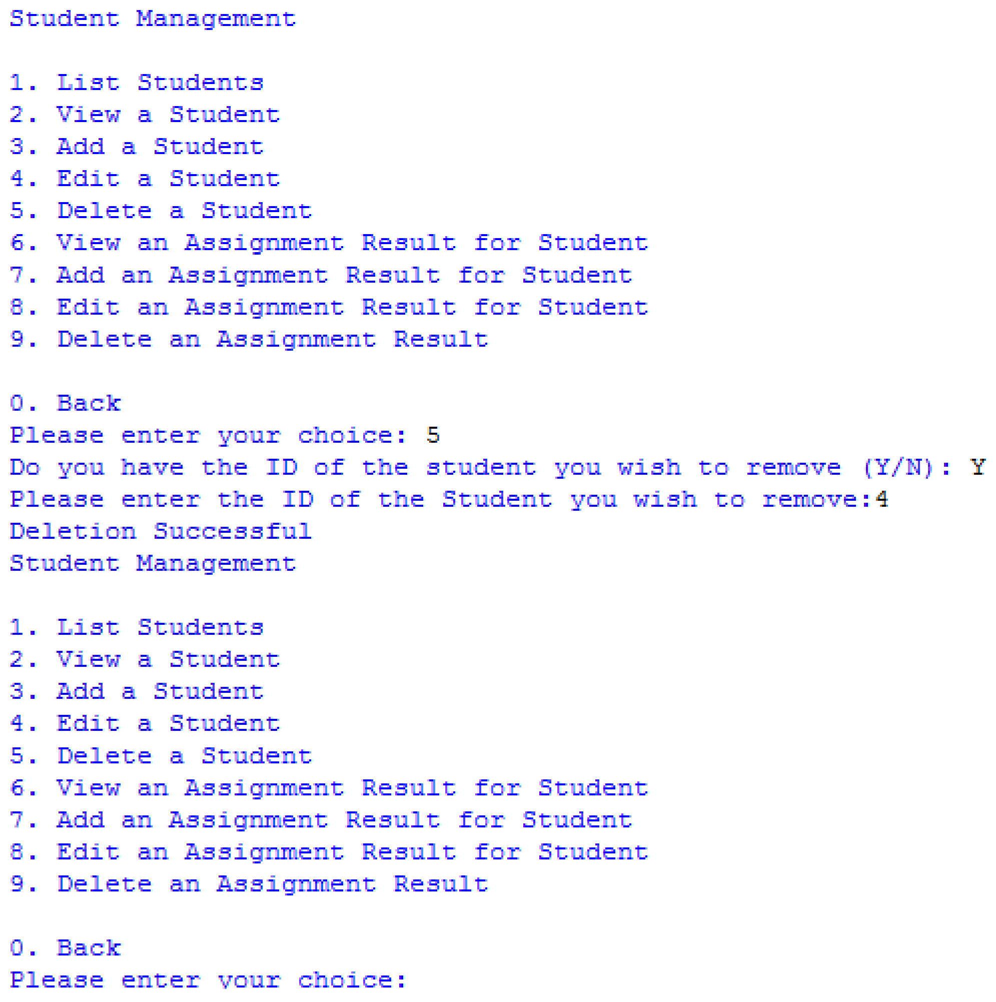
```
Edit a Student

Do you have the ID of the student you wish to edit (Y/N): Y
Please enter the ID of the Student you wish to edit:1
ID           LastName     FirstName    DOB          EMail                Scribe       25Extra      50Extra      WordProcessorGCSEResults  LastEmailed  Notes
1            Barham       Michael      1993-12-12   1341@longroad.ac.uk  True         True         False        False        4.3          0000-00-00
If you do not wish to edit an item, leave it blank
Please enter the student's Last Name:
Please enter the student's First Name:
Please enter the student's DOB in format YYYY-MM-DD:
Please enter the student's email address: 1342@longroad.ac.uk
Does the student have a scribe? (Y/N)
Does the student have 25% extra time? (Y/N)
Does the student have 50% extra time? (Y/N)
Does the student use a Word Processor? (Y/N)
Please enter the student's GCSE Results value:
Please enter the student's notes:
Successful
Student Management

1. List Students
2. View a Student
3. Add a Student
4. Edit a Student
5. Delete a Student
6. View an Assignment Result for Student
7. Add an Assignment Result for Student
8. Edit an Assignment Result for Student
9. Delete an Assignment Result

0. Back
Please enter your choice:
```

Without ID



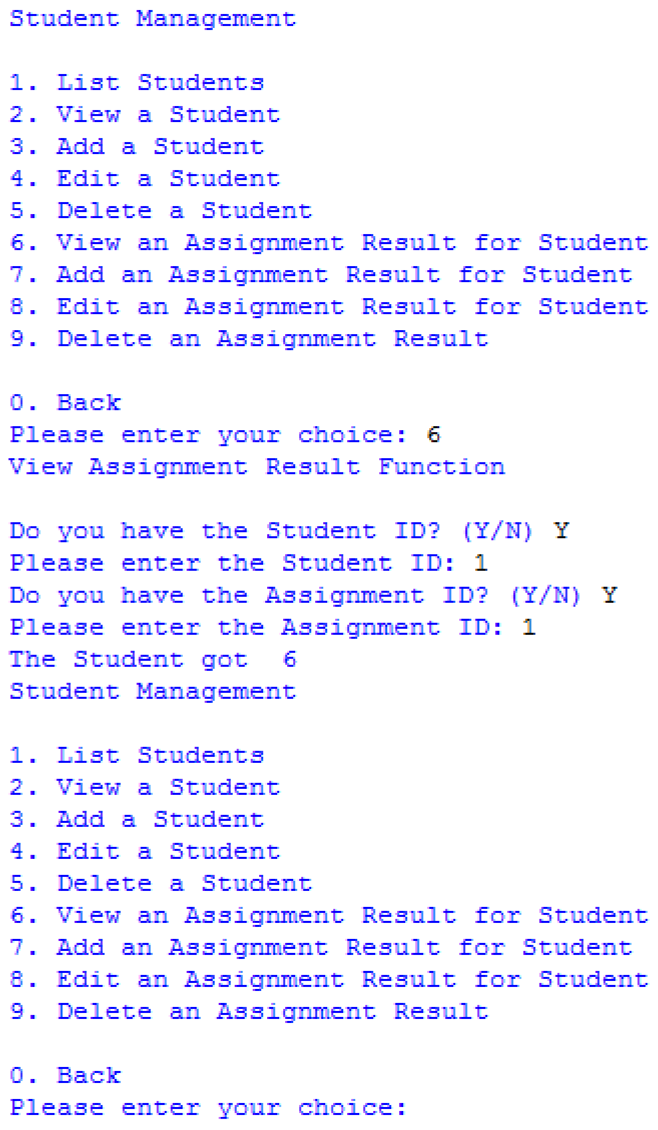
```
Edit a Student

Do you have the ID of the student you wish to edit (Y/N): N
View Student Function
Please input the details that you would like to use to find the student.
You can leave any value blank if you do not know it

Please enter the student's ID:
Please enter the student's Last Name: Barham
Please enter the student's First Name:
Please enter the student's DOB in format YYYY-MM-DD:
Please enter the student's email address:
Does the student have a scribe? (Y/N)
Does the student have 25% extra time? (Y/N)
Does the student have 50% extra time? (Y/N)
Does the student use a Word Processor? (Y/N)
Please enter the student's GCSE Results value:
ID           LastName     FirstName    DOB          EMail                Scribe       25Extra      50Extra      WordProcessorGCSEResults  LastEmailed  Notes
1            Barham       Michael      1993-12-12   1341@longroad.ac.uk  True         True         False        False        4.3          0000-00-00
Please enter the ID of the Student you wish to edit:1
ID           LastName     FirstName    DOB          EMail                Scribe       25Extra      50Extra      WordProcessorGCSEResults  LastEmailed  Notes
1            Barham       Michael      1993-12-12   1341@longroad.ac.uk  True         True         False        False        4.3          0000-00-00
If you do not wish to edit an item, leave it blank
Please enter the student's Last Name:
Please enter the student's First Name:
Please enter the student's DOB in format YYYY-MM-DD:
Please enter the student's email address: 1342@longroad.ac.uk
Does the student have a scribe? (Y/N)
Does the student have 25% extra time? (Y/N)
Does the student have 50% extra time? (Y/N)
Does the student use a Word Processor? (Y/N)
Please enter the student's GCSE Results value:
Please enter the student's notes:
Successful
Student Management

1. List Students
2. View a Student
3. Add a Student
4. Edit a Student
5. Delete a Student
6. View an Assignment Result for Student
7. Add an Assignment Result for Student
8. Edit an Assignment Result for Student
9. Delete an Assignment Result

0. Back
Please enter your choice:
```

5.1.2.6 Delete a Student



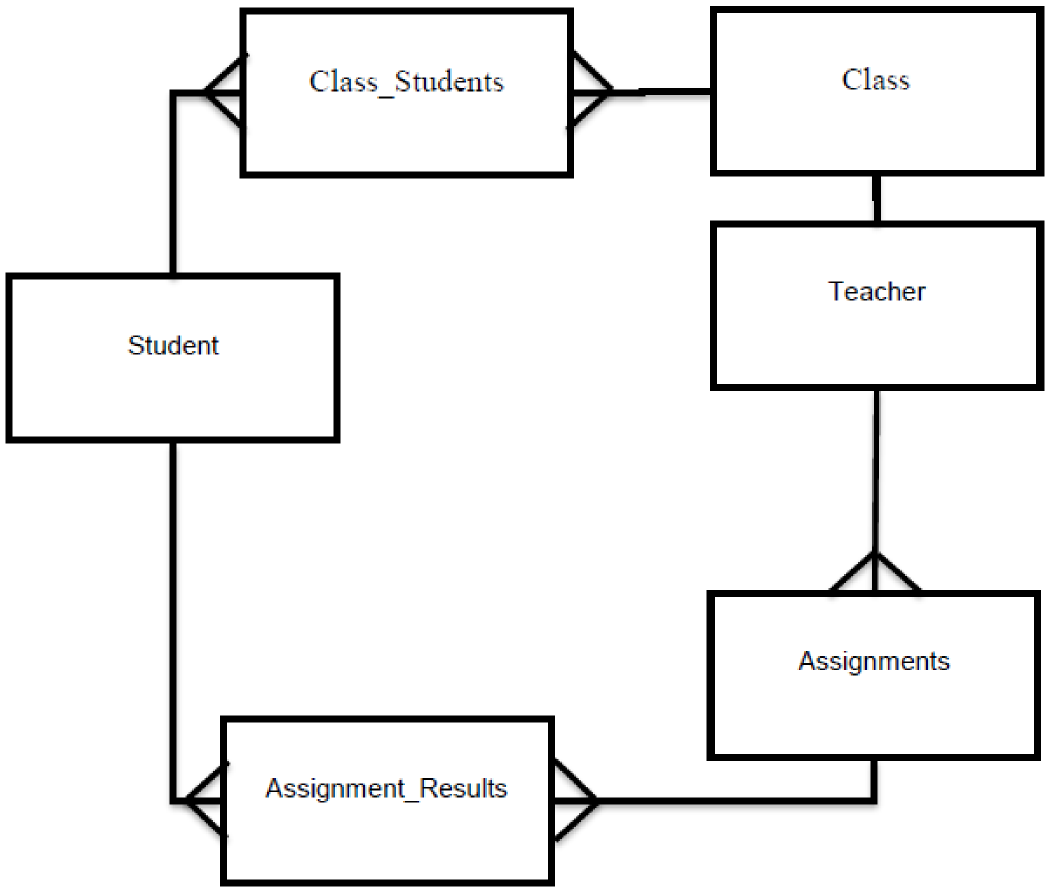
```
Student Management

1. List Students
2. View a Student
3. Add a Student
4. Edit a Student
5. Delete a Student
6. View an Assignment Result for Student
7. Add an Assignment Result for Student
8. Edit an Assignment Result for Student
9. Delete an Assignment Result

0. Back
Please enter your choice: 5
Do you have the ID of the student you wish to remove (Y/N): Y
Please enter the ID of the Student you wish to remove:4
Deletion Successful
Student Management

1. List Students
2. View a Student
3. Add a Student
4. Edit a Student
5. Delete a Student
6. View an Assignment Result for Student
7. Add an Assignment Result for Student
8. Edit an Assignment Result for Student
9. Delete an Assignment Result

0. Back
Please enter your choice:
```

5.1.2.7 View an Assignment Result



```
Student Management

1. List Students
2. View a Student
3. Add a Student
4. Edit a Student
5. Delete a Student
6. View an Assignment Result for Student
7. Add an Assignment Result for Student
8. Edit an Assignment Result for Student
9. Delete an Assignment Result

0. Back
Please enter your choice: 6
View Assignment Result Function

Do you have the Student ID? (Y/N) Y
Please enter the Student ID: 1
Do you have the Assignment ID? (Y/N) Y
Please enter the Assignment ID: 1
The Student got  6
Student Management

1. List Students
2. View a Student
3. Add a Student
4. Edit a Student
5. Delete a Student
6. View an Assignment Result for Student
7. Add an Assignment Result for Student
8. Edit an Assignment Result for Student
9. Delete an Assignment Result

0. Back
Please enter your choice:
```

5.2 ER Diagram



5.3 Database Table Views

Teacher Table

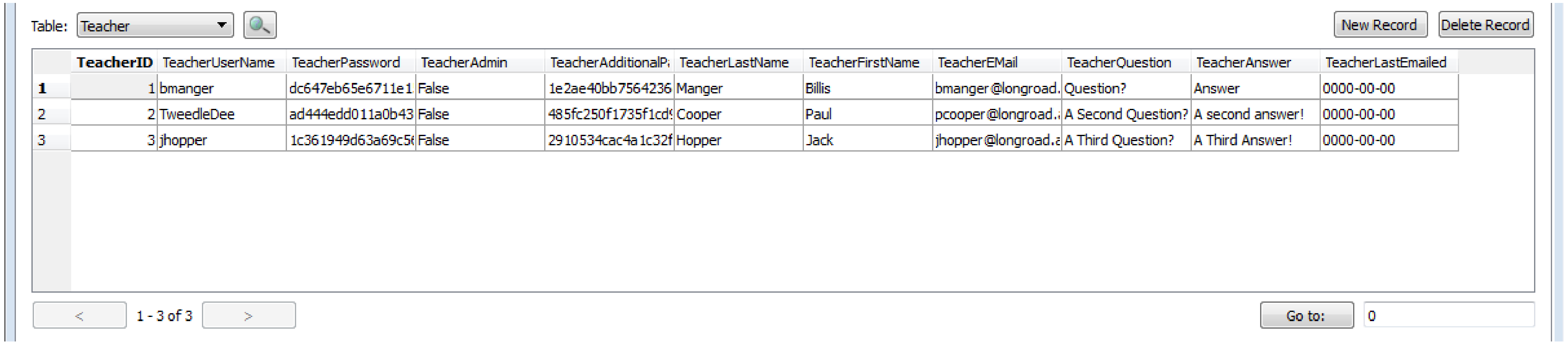


Table: Teacher

	TeacherID	TeacherUserName	TeacherPassword	TeacherAdmin	TeacherAdditionalP	TeacherLastName	TeacherFirstName	TeacherEMail	TeacherQuestion	TeacherAnswer	TeacherLastEmailed
1	1	bmanger	dc647eb65e6711e1	False	1e2ae40bb7564236	Manger	Billis	bmanger@longroad.	Question?	Answer	0000-00-00
2	2	TweedleDee	ad444edd011a0b43	False	485fc250f1735f1cd	Cooper	Paul	pcooper@longroad.	A Second Question?	A second answer!	0000-00-00
3	3	jhopper	1c361949d63a69c5	False	2910534cac4a1c32f	Hopper	Jack	jhopper@longroad.	A Third Question?	A Third Answer!	0000-00-00

1 - 3 of 3

Student Table

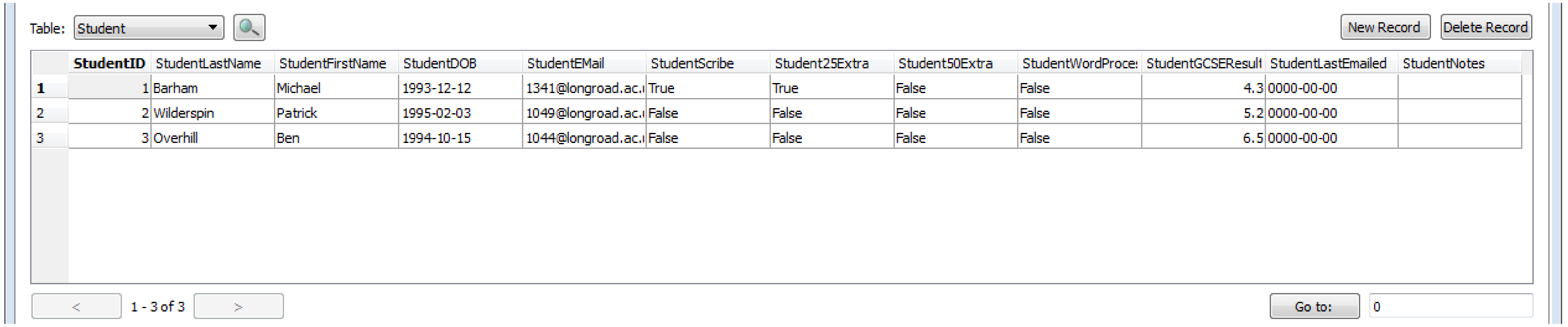
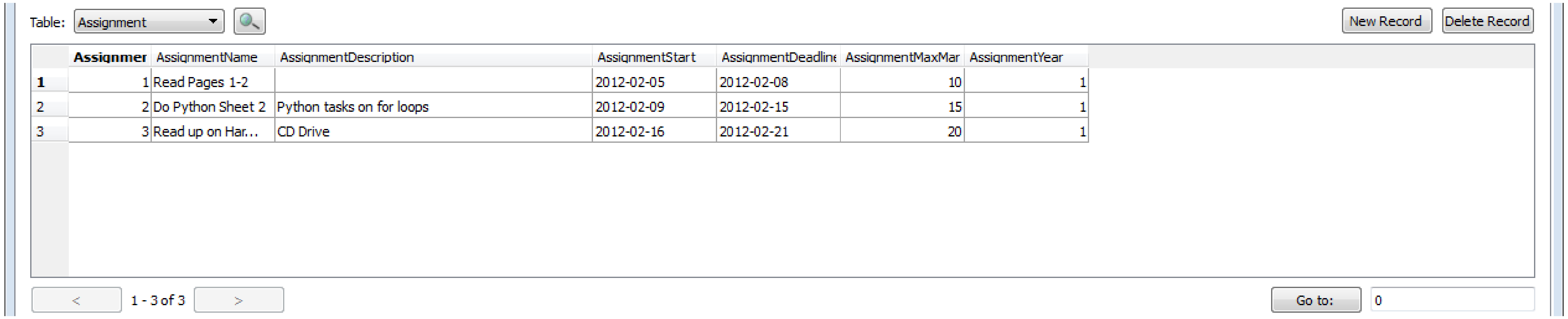


Table: Student

	StudentID	StudentLastName	StudentFirstName	StudentDOB	StudentEMail	StudentScribe	Student25Extra	Student50Extra	StudentWordProce	StudentGCSEResult	StudentLastEmailed	StudentNotes
1	1	Barham	Michael	1993-12-12	1341@longroad.ac.	True	True	False	False	4.3	0000-00-00	
2	2	Wilderspin	Patrick	1995-02-03	1049@longroad.ac.	False	False	False	False	5.2	0000-00-00	
3	3	Overhill	Ben	1994-10-15	1044@longroad.ac.	False	False	False	False	6.5	0000-00-00	

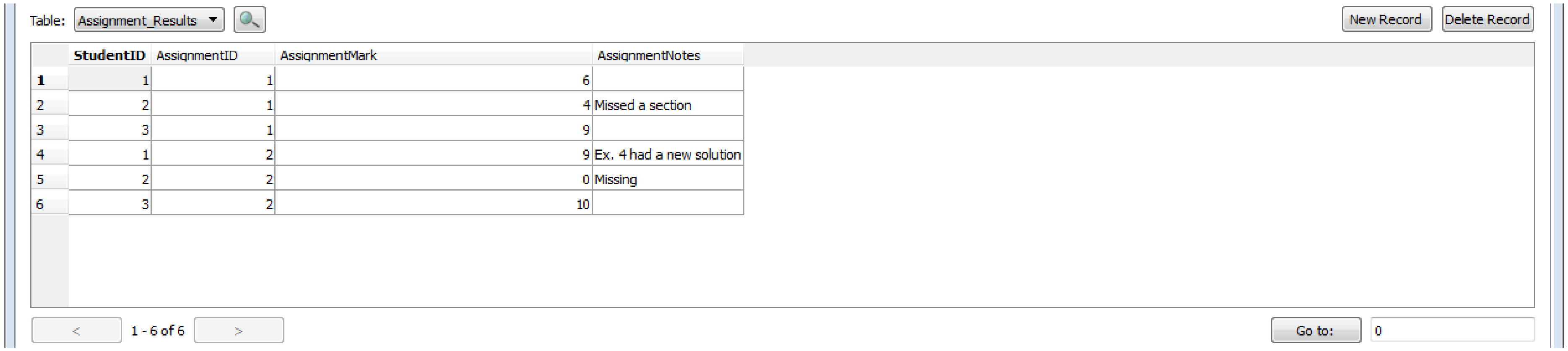
1 - 3 of 3

Assignment Table



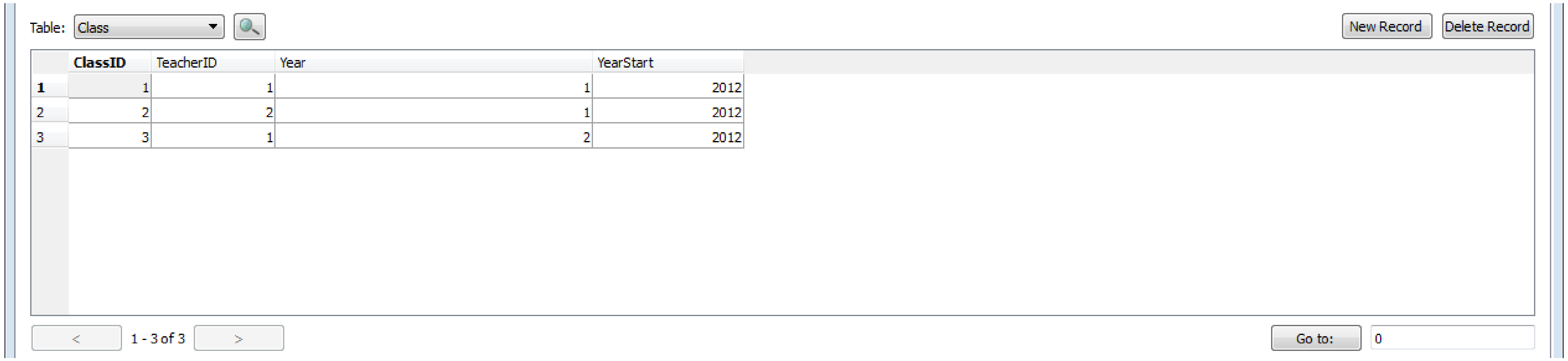
	Assignmer	AssignmentName	AssignmentDescription	AssignmentStart	AssignmentDeadline	AssignmentMaxMar	AssignmentYear
1	1	Read Pages 1-2		2012-02-05	2012-02-08	10	1
2	2	Do Python Sheet 2	Python tasks on for loops	2012-02-09	2012-02-15	15	1
3	3	Read up on Har...	CD Drive	2012-02-16	2012-02-21	20	1

Assignment_Results Table



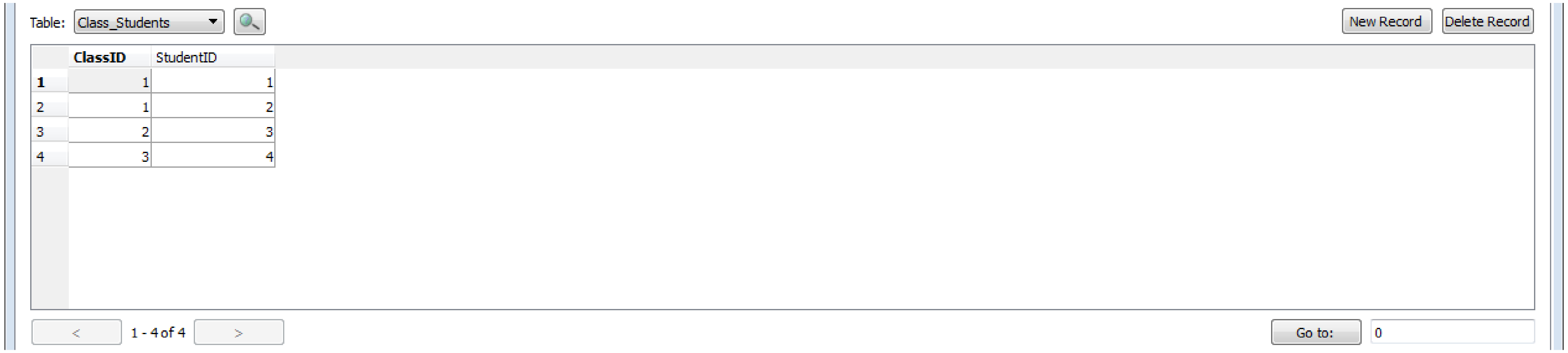
	StudentID	AssignmentID	AssignmentMark	AssignmentNotes
1	1	1	6	
2	2	1	4	Missed a section
3	3	1	9	
4	1	2	9	Ex. 4 had a new solution
5	2	2	0	Missing
6	3	2	10	

Class Table



	ClassID	TeacherID	Year	YearStart
1	1	1	1	2012
2	2	2	1	2012
3	3	1	2	2012

Class_Students Table



	ClassID	StudentID
1	1	1
2	1	2
3	2	3
4	3	4

5.4 Database SQL

5.4.1 Teacher Table

```
create table Teacher (
TeacherID integer,
TeacherUserName text,
TeacherPassword text,
TeacherAdmin integer,
TeacherAdditionalPassword text,
TeacherLastName text,
TeacherFirstName text,
TeacherEMail text,
TeacherQuestion text,
TeacherAnswer text,
TeacherLastEmailed text,
primary key (TeacherID))
```

5.4.2 Student Table

```
create table Student (
StudentID integer,
StudentLastName text,
StudentFirstName text,
StudentDOB text,
StudentEMail text,
StudentScribe integer,
Student25Extra integer,
Student50Extra integer,
StudentWordProcessor integer,
StudentGCSEResults real,
StudentLastEmailed text,
StudentNotes text,
primary key (StudentID)
```

5.4.3 Assignment Table

```
create table Assignment (
AssignmentID integer,
AssignmentName text,
AssignmentDescription text,
AssignmentStart text,
AssignmentDeadline text,
AssignmentMaxMark integer,
AssignmentYear integer,
primary key (AssignmentID)
```

5.4.4 Assignment_Results Table

```
create table Assignment_Results (
StudentID integer,
AssignmentID integer,
AssignmentMark integer,
AssignmentNotes text,
primary key (StudentID, AssignmentID),
foreign key (StudentID) references Student(StudentID) ON UPDATE
CASCADE ON DELETE RESTRICT,
foreign key (AssignmentID) references Assignment(AssignmentID) ON
UPDATE CASCADE ON DELETE RESTRICT)
```

5.4.5 Class Table

```
create table Class (
ClassID integer,
TeacherID integer,
Year integer,
YearStart integer,
primary key (ClassID),
foreign key (TeacherID) references Teacher(TeacherID) ON UPDATE
CASCADE ON DELETE RESTRICT)
```

5.4.6 Class_Students Table

```
create table Class_Students (
ClassID integer,
StudentID integer,
primary key (ClassID, StudentID),
foreign key (ClassID) references Class(ClassID) ON UPDATE CASCADE ON
DELETE RESTRICT,
foreign key (StudentID) references Student(StudentID) ON UPDATE
CASCADE ON DELETE RESTRICT)
```

5.5 SQL Queries

5.5.1 Student Controller

5.5.1.1 Adding a Student

```
"""insert into Student(StudentLastName, StudentFirstName, StudentDOB,
StudentEmail,StudentScribe,Student25Extra, Student50Extra,
StudentWordProcessor, StudentGCSEResults, StudentLastEmailed,
StudentNotes) values
('{0}','{1}','{2}','{3}','{4}','{5}','{6}','{7}','{8}','{9}','{10}')"
"".format(StudentLastName, StudentFirstName, StudentDOB,
StudentEmail, StudentScribe, Student25Extra, Student50Extra,
StudentWordProcessor, StudentGCSEResults, StudentLastEmailed,
StudentNotes)
```

5.5.1.2 Editing a Student

I have provided an example of the code here rather than the full code, which can be seen in the appendix on page.
update student set StudentLastName='Michael' where StudentID =1

5.5.1.3 Deleting a Student

```
"DELETE from student WHERE StudentID = {0}".format(StudentID)
```

6. Testing

6.1 Summary of Results

Overall, the testing went well and showed that the program worked. All the menus worked and forbid any invalid characters. The program mostly denied invalid input, although it did let some false characters in, though the majority of these would never actually be used.
While testing, I followed a plan and the majority of it worked fine, however I also tried a blanket database and if the user tried to skip a step or two, it would crash.

6.2 Known Issues

- If not all of the Assignments and Students in a class have marks, the statistics program will crash.
- While not a technical error, the use of Y/Ns can be frustrating to the user if they must switch. Alternative inputs (such as 1s and 0s and lowercase) should be allowed.

7. Code Explanations

7.1 Difficult Sections

7.1.1 Printing a Table

This function allows the user to print a table using the formatting ability, without needing to define each heading.

It first changes each heading, cuts off the un-needed beginning and places them into an array.

It then uses a for loop to add each heading to the headings string, with some space so it's not all squished together, and prints it.

It then uses the same principle with the data, where it constructs a string for each row.

```
def print_table(self,headings,data):
    #This function prints a table for a list/view function
    #Sets up a blank array for the attributes
    table_attributes = []
    #Processes each heading
    for each in headings:
        #Puts it into a variable to simplify things
        heading = each[1]
        #Removes the first several characters
        heading = heading[7:]
        #Appends it to the table_attributes list
        table_attributes.append(heading)
    #Creates a blank headings string
    headings = ""
    #Processes each attribute
    for count in range(len(table_attributes)):
        #Calculates the length of the text and adds 5 for
visibility
        #Adds additional space for easy viewing for email address
        if count == 4:
            headings = headings + '{0[' + str(count) + ']:<21}'
        else:
            headings = headings + '{0[' + str(count) + ']:<13}'
    #Prints the headings
    print(headings.format(table_attributes))
     #Processes each row
    for each in data:
        #Creates a blank string for the row
        result = ""
        #Processes each "cell"
        for count in range(len(each)):
            #constructs the "cell", adding additional space for
email address
            if count == 4:
               result = result + '{0[' + str(count) + ']:<21}'
            else:
                result = result + '{0[' + str(count) + ']:<13}'
        #Prints the row
        print(result.format(each))
```

7.2 Self-created Algorithms

7.2.1 Average Results for Class

This algorithm is designed to get the average results for a class. When it is passed a ClassID as a parameter, it will look-up the year that the class belongs to. It will then use this to get an array of assignmentIDs that this corresponds to. Then it'll look up a list of students that belong to the class.

It will then get all the assignment results for each student and place these into another array. Then cycle through each of these adding each assignment mark to the total, which it then divides by the number of items in the students list to get the average.

```
    def get_average_results_for_class(self,ClassID):
        #Gets the year of the class specified
        year = self.get_class_year(ClassID)
        #Gets assignments that the year applies to
        assignments = self.get_assignments_for_year(year)
        #Gets the student's IDs of the class specified
        students = self.get_students_in_class(ClassID)
        #Sets up blank list for loop to dump data into
        data = []
        #Loops round for each student
        for eachstudent in students:
            #Creates a blank array for their assignment results
            student_assignment_results = []
            #Loops round for each assignent
            for eachassignment in assignments:
                #Performs database query to get each result and
appends to database

student_assignment_results.append(self.get_assignment_result(eachstud
ent,eachassignment))
            #Appends each array to the overall data array
            data.append(student_assignment_results)
        #creates arrays for average column
        total_results = []
        average_results = []
        #Loops round each item in assignments
        for countassignment in range(len(assignments)):
            #Appends a "0" the beginning of the counter
            total_results.append(0)
            #Loops round each student
            for countstudent in
range(len(student_assignment_results)):
                #Adds the students result to the total results
                total_results[countassignment] =
total_results[countassignment] + data[countstudent][countassignment]
            #Gets the average result

average_results.append((total_results[countassignment]/len(students))
)
        return average_results,assignments
```

8. Settings

The program, as it stands (ie. No GUI or graphs), requires no additional settings to be made, however, there are a number of things that need to be checked for the prospective features.

8.1 GUI
For the GUI to function, the third-party library PyQT must be also install on any computers. This can be installed using an installer that can be found on the PyQT website, which would be included in any distribution.

8.2 Graphs
Like the GUI, graphs require a third-party library. The program would have been based on matplotlib, which can also be installed via an installer.

8.3 Email
Email was to be a part of the program and while, on the surface, it appears this doesn't require any settings, there are a number of things that must be checked.

- The email host, username and password must be filled out in the administration section
- The email host must support smtp, and this must be enabled.
- The ports required must **not** be blocked by the ISP, namely the college. The standard ports are 25 and 587.

9. Acknowledgements

9.1 Email code
The email code I used was taken from http://stackoverflow.com/questions/549391/python-3-0-smtplib. I turned it into a function and simplified it slightly.

9.2 PyQT and MatPlotLib
These each are taken from http://www.riverbankcomputing.co.uk/software/pyqt and http://www.matplotlib.org respectively.

10. Code Listing

10.1 assignment_controller.py

```
#Imports the data from the controller class
#for communication with the DB
from controller_class import *

#Creates a new class, using the db controller as its parent
class assignment_controller(database_controller):
    """Controller for the database connections with assignments"""
    #This sets up any values that I need for my class
    def __init__(self):
        #This inherits any of the values from the parent class
        super().__init__()

    def add_assignment(self, AssignmentName, AssignmentDescription, AssignmentStart, AssignmentDeadline,
AssignmentMaxMark, AssignmentYear):
        #This function allows the user to add an assignment to the database.

        #This SQL statement contains the details I need adding to the db
        #It uses the format ability to easily insert all the values into
        #the statement.
        sql = """insert into Assignment(AssignmentName, AssignmentDescription, AssignmentStart, AssignmentDeadline,
AssignmentMaxMark, AssignmentYear)
                values
                ('{0}','{1}','{2}','{3}','{4}','{5}')""".format(AssignmentName, AssignmentDescription,
AssignmentStart, AssignmentDeadline, AssignmentMaxMark, AssignmentYear)
        #Perform the operation
        self._query(sql)

    def edit_assignment(self, AssignmentID, AssignmentName=None, AssignmentDescription=None, AssignmentStart=None,
AssignmentDeadline=None, AssignmentMaxMark=None, AssignmentYear = None):
        #This function allows me to edit all of an Assignments’ values in one go
        #It uses named parameters to allow me to have them optional

        #Starts the list of changes needed
        changes = []

        #Checks each value to see if they're used
        #If used, it will append each change to the list as a list
        #I.e., a list of lists.
        if AssignmentName != None:
            changes.append(("AssignmentName",AssignmentName))
        if AssignmentDescription != None:
            changes.append(("AssignmentDescription",AssignmentDescription))
        if AssignmentStart != None:
            changes.append(("AssignmentStart",AssignmentStart))
        if AssignmentDeadline != None:
            changes.append(("AssignmentDeadline",AssignmentDeadline))
        if AssignmentMaxMark != None:
            changes.append(("AssignmentMaxMark",AssignmentMaxMark))
        if AssignmentYear != None:
            changes.append(("AssignmentYear",AssignmentYear))


        #This is the start of the sql statement that will be added to
        sql = "update Assignment set "
        #Iteration of each list within the changes list
        for update in changes:
            #This adds each update to the sql statement
            sql += "{0}='{1}', ".format(update[0],update[1])

        #Remove the last 2 characters ', '
        sql = sql[:-2]
        #Adds which ID to edit
        sql+= " where AssignmentID ='{0}'".format(AssignmentID)

        #Performs the query to the database
        self._query(sql)

    def delete_assignment(self,StudentID):
        #This function deletes a row from the table
        sql = "DELETE from Assignment WHERE AssignmentID = {0}".format(AssignmentID)
        self._query(sql)

    def find_assignment(self, AssignmentID=None, AssignmentName=None, AssignmentDescription=None,
AssignmentStart=None, AssignmentDeadline=None, AssignmentMaxMark=None, AssignmentYear = None):
        #This function is designed to find all the rows that match the following data.
        #It works in the same way as the update function.

        #Creates a new list
        parameters = []

        #Detects if the named parameters are used
        #if so, it will append them to the list
        if AssignmentID != None:
            parameters.append(("AssignmentID",AssignmentID))
        if AssignmentName != None:
            parameters.append(("AssignmentName",AssignmentName))
        if AssignmentDescription != None:
            parameters.append(("AssignmentDescription",AssignmentDescription))
        if AssignmentStart != None:
            parameters.append(("AssignmentStart",AssignmentStart))
        if AssignmentDeadline != None:
            parameters.append(("AssignmentDeadline",AssignmentDeadline))
        if AssignmentMaxMark != None:
            parameters.append(("AssignmentMaxMark",AssignmentMaxMark))
        if AssignmentYear != None:
            parameters.append(("AssignmentYear",AssignmentYear))

        #This begins the select command for the list
        #It's choosing only certain columns for the list, because of security.
        sql = """select *
                FROM Assignment
                where """

        #This adds all the parameters to the sql statement
        for parameter in parameters:
            sql = sql + "{0}='{1}' and ".format(parameter[0],parameter[1])

        #This removes the final " and" from the sql statement
        sql = sql[:-5]
        return self._select_query(sql)

    def assignment_headings(self):
        #Gets table information
        sql = "PRAGMA table_info(assignment)"
        return self._select_query(sql)

    def get_assignments_for_year(self,year):
        #Gets all assignment IDs for a year (i.e. AS or A2)
        sql = """select AssignmentID from Assignment
                where AssignmentYear = {0}""".format(year)
        results = self._select_query(sql)
        #Puts these into a simple array
        IDs = []
        for each in results:
            IDs.append(each[0])
        return IDs

    def get_assignment_max_mark(self,ID):
        #Gets a max mark for an assignment
        sql = """select AssignmentMaxMark from Assignment
                where AssignmentID = {0}""".format(ID)
        results = self._select_query(sql)
        #Put this into an array
        #It does this to prevent crashes if there's no MaxMark
        IDs = []
        for each in results:
            IDs.append(each[0])
        return IDs
```

10.2 assignment_results_controller.py

```python
#Imports the data from the controller class
#for communication with the DB
from controller_class import *

#Creates a new class, using the db controller as its parent
class assignment_results_controller(database_controller):
    """Controller for the database connections with an assignment result"""
    #This sets up any values that I need for my class
    def __init__(self):
        #This inherits any of the values from the parent class
        super().__init__()

    def add_assignment_result(self, StudentID, AssignmentID, AssignmentMark, AssignmentNotes):
        #This function allows the user to add a student to the database.

        #This SQL statement contains the details I need adding to the db
        #It uses the format ability to easily insert all the values into
        #the statement.
        sql = """insert into Assignment_Results(StudentID,AssignmentID,AssignmentMark,AssignmentNotes)
                values
                ('{0}','{1}','{2}','{3}')""".format(StudentID,AssignmentID,AssignmentMark, AssignmentNotes)
        #Perform the operation
        self._query(sql)

    def get_assignment_result(self,StudentID,AssignmentID):
        #Gets an assignment result for a student and assignment
        sql = """select AssignmentMark from Assignment_Results
                where StudentID = {0} and AssignmentID = {1}""".format(StudentID,AssignmentID)
        result = self._select_query(sql)
        #Returns just the result alone, rather than an array inside an array with a singular value
        return result[0][0]

    def edit_assignment_results(self, StudentID,AssignmentID,AssignmentMark=None, AssignmentNotes=None):
        #This function allows me to edit all of a student's values in one go
        #It uses named parameters to allow me to have them optional

        #Starts the list of changes needed
        changes = []

        #Checks each value to see if they're used
        #If used, it will append each change to the list as a list
        #Ie, a list of lists.
        if AssignmentMark != None:
            changes.append(("AssignmentMark",AssignmentMark))
        if AssignmentNotes != None:
            changes.append(("AssignmentNotes",AssignmentNotes))
        #This is the start of the sql statement that will be added to
        sql = "update Assignment_Results set "
        #Iteration of each list within the changes list
        for update in changes:
            #This adds each update to the sql statement
            sql += "{0}='{1}', ".format(update[0],update[1])

        #Remove the last 2 characters ', '
        sql = sql[:-2]
        #Adds which ID to edit
        sql+= " where StudentID ='{0}' and AssignmentID='{1}'".format(StudentID,AssignmentID)

        #Performs the query to the database
        self._query(sql)

    def delete_assignment_result(self,StudentID,AssignmentID):
        #This function deletes a row from the table
        sql = "DELETE from assignment_results where StudentID ='{0}' and 
AssignmentID='{1}'".format(StudentID,AssignmentID
      self._query(sql)

```

10.3 class_controller.py

```
#Imports the data from the controller class
#for communication with the DB
from controller_class import *

#Creates a new class, using the db controller as its parent
class class_controller(database_controller):
    """Controller for the database connections with a class"""
    #This sets up any values that I need for my class
    def __init__(self):
        #This inherits any of the values from the parent class
        super().__init__()

    def add_class(self, TeacherID, Year, YearStart):
        #This function allows the user to add a student to the database.

        #This SQL statement contains the details I need adding to the db
        #It uses the format ability to easily insert all the values into
        #the statement.
        sql = """insert into Class(TeacherID, Year, YearStart)
                values
                ('{0}','{1}','{2}')""".format(TeacherID, Year, YearStart)
        #Perform the operation
        self._query(sql)

    def edit_class(self, ClassID, TeacherID=None, Year=None, YearStart=None):
        #This function allows me to edit all of a student's values in one go
        #It uses named parameters to allow me to have them optional

        #Starts the list of changes needed
        changes = []

        #Checks each value to see if they're used
        #If used, it will append each change to the list as a list
        #i.e., a list of lists.
        if TeacherID != None:
            changes.append(("TeacherID",TeacherID))
        if Year != None:
            changes.append(("Year",Year))
        if YearStart != None:
            changes.append(("YearStart",YearStart))
        #This is the start of the sql statement that will be added to
        sql = "update class set "
        #Iteration of each list within the changes list
        for update in changes:
            #This adds each update to the sql statement
            sql += "{0}='{1}', ".format(update[0],update[1])

        #Remove the last 2 characters ', '
        sql = sql[:-2]
        #Adds which ID to edit
        sql+= " where ClassID ='{0}'".format(ClassID)

        #Performs the query to the database
        self._query(sql)

    def delete_class(self,ClassID):
        #This function deletes a row from the table
        sql = "DELETE from class WHERE ClassID = {0}".format(ClassID)
        self._query(sql)

    def find_class(self, ClassID=None, TeacherID=None, Year=None, YearStart=None):
        #This function is designed to find all the rows that match the following data.
        #It works in the same way as the update function.

        #Creates a new list
        parameters = []

        #Detects if Student the named parameters are used
        #if Student so, it will append them to the list
        if ClassID != None:
            parameters.append(("ClassID",ClassID))
        if TeacherID != None:
            parameters.append(("TeacherID",TeacherID))
        if Year != None:
            parameters.append(("Year",Year))
        if YearStart != None:
            parameters.append(("YearStart",YearStart))

        #This begins the select command for the list
        sql = """select *
                FROM class
                where """

        #This adds all the parameters to the sql statement
        for parameter in parameters:
            sql = sql + "{0}='{1}' and".format(parameter[0],parameter[1])

        #This removes the final " and" from the sql statement
        sql = sql[:-4]
        return self._select_query(sql)

    def class_headings(self):
        #Gets table information for class
        sql = "PRAGMA table_info(class)"
        return self._select_query(sql)

    def get_class_year(self,ClassID):
        #Gets the year (AS or A2) that a Class belongs to
        sql = """Select Year from Class where ClassID = {0}""".format(ClassID)
        results = self._select_query(sql)
        print(results)
        return results[0][0]
```

10.4 class_students_controller.py

```
#Imports the data from the controller class
#for communication with the DB
from controller_class import *

#Creates a new class, using the db controller as its parent
class class_students_controller(database_controller):
    """Controller for the database connections with a class's students"""
    #This sets up any values that I need for my class
    def __init__(self):
        #This inherits any of the values from the parent class
        super().__init__()

    def add_class_student(self, ClassID, StudentID):
        #This function allows the user to add a student to the database.

        #This SQL statement contains the details I need adding to the db
        #It uses the format ability to easily insert all the values into
        #the statement.
        sql = """insert into Class_Students(ClassID, StudentID)
                values
                ('{0}','{1}')""".format(ClassID, StudentID)
        #Perform the operation
        self._query(sql)

    def edit_class_student(self, Old_ClassID, Old_StudentID, New_ClassID, New_StudentID):
        #This function is for editing a class_student row
        #It does NOT use named values as it exists solely of a composite key (therefore, all entries are required
anyway)
        #so it would be faster to do it in one statement, rather than using named parameters and if statements

        sql = """update Class_Students set
                ClassID='{0}', StudentID='{1}'
```

```
                    where ClassID='{2}' and StudentID='{3}'""".format(New_ClassID, New_StudentID, Old_ClassID,
Old_StudentID)
        #Performs the query to the database
        self._query(sql)

    def delete_class_student(self,ClassID,StudentID):
        #This function deletes a row from the table
        sql = "DELETE from class_students WHERE ClassID = {0} and StudentID = {1}".format(ClassID,StudentID)
        self._query(sql)

    def find_class_student(self, ClassID=None, StudentID=None):
        #This function is designed to find all the rows that match the following data.
        #It works in the same way as the update function.

        #Creates a new list
        parameters = []

        #Detects if the named parameters are used
        #if so, it will append them to the list
        if ClassID != None:
            parameters.append(("ClassID",ClassID))
        if StudentID != None:
            parameters.append(("StudentID",StudentID))

        #This begins the select command for the list
        #It's choosing only certain columns for the list, because of security.
        sql = """select *
                FROM Class_Students
                where """

        #This adds all the parameters to the sql statement
        for parameter in parameters:
            sql = sql + "{0}='{1}' and ".format(parameter[0],parameter[1])

        #This removes the final " and" from the sql statement
        sql = sql[:-5]
        print(sql)
        return self._select_query(sql)

    def get_students_in_class(self,ClassID):
        #This function will get all students's IDs within a class in a straight list
        sql ="select StudentID FROM Class_Students where ClassID = {0}".format(ClassID)
        results = self._select_query(sql)
        students = []
        for each in results:
            students.append(each[0])
        return students
```

10.5 cli.py

```
#imports all the CLI sections
from CLI_Student_Management import *
from CLI_Class_Management import *
from CLI_Assignment_Management import *
from CLI_Assignment_Results_Management import *
from CLI_Statistics import *
from CLI_Administration import *

class CLI_Class():

    def __init__(self):
        #Nothing to initialise
        pass

    def Get_Option(self,list):
        valid = False
        while not valid:
            #Try is there to avoid the program crashing
            try:
                #Attempts to put the option into an integer
                #This removes possibilities of letter/blank options
                option = int(input("Please enter your choice: "))
                #Checks to see if the option is in the list
                if option in list:
                    #Gives the option back to the user
                    return option
                else:
                    #Prints an error
                    print("Please choose an option on the list")
            except:
                #Prints an error if the int function fails.
                print("That is not a valid integer. Please try again")


    def Student_Management(self):
        #Instantiates the Student Manager
        StudentManager = CLI_Student_Manager_Class()
        AssignmentResultManager = CLI_Assignment_Results_Manager_Class()
        #Sets up while loop
        exit = False
        while not exit:
            print("Student Management")
            print("")
            print("1. List Students")
            print("2. View a Student")
            print("3. Add a Student")
            print("4. Edit a Student")
            print("5. Delete a Student")
            print("6. View an Assignment Result for Student")
            print("7. Add an Assignment Result for Student")
            print("8. Edit an Assignment Result for Student")
            print("9. Delete an Assignment Result")
            print("")
            print("0. Back")
            #Calls Get_Option function for verification
            option = self.Get_Option([0,1,2,3,4,5,6,7,8,9])
            #Checks which option was chosen
            if option == 0:
                #Exits the loop
                exit = True
            elif option == 1:
                #Calls the CLI_list_student function
                StudentManager.CLI_list_student()
            elif option == 2:
                StudentManager.CLI_view_student()
            elif option == 3:
                StudentManager.CLI_add_student()
            elif option == 4:
                StudentManager.CLI_edit_student()
            elif option == 5:
                StudentManager.CLI_delete_student()
            elif option == 6:
                AssignmentResultManager.CLI_View_Assignment_Result()
            elif option == 7:
                AssignmentResultManager.CLI_Add_Assignment_Result()
            elif option == 8:
                AssignmentResultManager.CLI_Edit_Assignment_Result()
            elif option == 9:
                AssignmentResultManager.CLI_Delete_Assignment_Result()




    def Class_Management(self):
        #Instantiates the Class Manager class
        ClassManager = CLI_Class_Manager_Class()
        #Sets up the while loop
        exit = False
        while not exit:
            print("Class Management")
            print("")
            print("1. List Classes")
            print("2. View a Class")
            print("3. Add a Class")
            print("4. Edit a Class")
```

```
                print("5. Delete a Class")
                print("6. View Students in a Class")
                print("7. Add a Student to a Class")
                print("8. Remove a Student from a Class")
                print("")
                print("0. Back")
                #Calls Get_Option function for verification
                option = self.Get_Option([0,1,2,3,4,5,6,7,8])
                #Checks which option was chosen
                if option == 0:
                    #Exits the loop
                    exit = True
                elif option == 1:
                    #Calls the CLI_List_Class function
                    ClassManager.CLI_list_class()
                elif option == 2:
                    ClassManager.CLI_view_class()
                elif option == 3:
                    ClassManager.CLI_add_class()
                elif option == 4:
                    ClassManager.CLI_edit_class()
                elif option == 5:
                    ClassManager.CLI_delete_class()
                elif option == 6:
                    ClassManager.CLI_view_students_in_a_class()
                elif option == 7:
                    ClassManager.CLI_add_to_class()
                elif option == 8:
                    ClassManager.CLI_remove_from_class()

        def Assignment_Management(self):
            #Instantiates the Assignment Manager class
            AssignmentManager = CLI_Assignment_Manager_Class()
            #Sets up the while loop
            exit = False
            while not exit:
                print("Assignment Management")
                print("")
                print("1. List Assignments")
                print("2. View a Assignment")
                print("3. Add a Assignment")
                print("4. Edit a Assignment")
                print("5. Delete a Assignment")
                print("")
                print("0. Back")
                #Calls Get_Option function for verification
                option = self.Get_Option([0,1,2,3,4,5])
                #Checks which option was chosen
                if option == 0:
                    #Exits the loop
                    exit = True
                elif option == 1:
                    #Calls the CLI_list_assignment function
                    AssignmentManager.CLI_list_assignment()
                elif option == 2:
                    AssignmentManager.CLI_view_assignment()
                elif option == 3:
                    AssignmentManager.CLI_add_assignment()
                elif option == 4:
                    AssignmentManager.CLI_edit_assignment()
                elif option == 5:
                    AssignmentManager.CLI_delete_assignment()

        def Administration(self):
            #Instantiates the Administration Class
            Administration = CLI_Administration_Class()
            #Sets up the while loop
            exit = False
            while not exit:
                print("Administration")
                print("")
                print("1. List Teachers")
            print("2. View a Teacher")
            print("3. Add a Teacher")
            print("4. Edit a Teacher")
            print("5. Delete a Teacher")
            print("")
            print("0. Back")
            #Calls Get_Option function for verification
            option = self.Get_Option([0,1,2,3,4,5])
            #Checks which option was chosen
            if option == 0:
                #Exits the loop
                exit = True
            elif option == 1:
                #Calls the CLI_list_teacher class
                Administration.CLI_list_teacher()
            elif option == 2:
                Administration.CLI_view_teacher()
            elif option == 3:
                Administration.CLI_add_teacher()
            elif option == 4:
                Administration.CLI_edit_teacher()
            elif option == 5:
                Administration.CLI_delete_teacher()
            elif option == 6:
                Administration.CLI_email_settings()

    def Statistics(self):
        #Instantiates the Statistics Class
        Stats = CLI_Statistics_Class()
        #Sets up the while loop
        exit = False
        while not exit:
            print("Statistics")
            print("")
            print("1. Average Results for Class")
            print("")
            print("0. Back")
            #Calls Get_Option function for verification
            option = self.Get_Option([0,1])
            #Checks which option was chosen
            if option == 0:
                #Exits the loop
                exit = True
            elif option == 1:
                #Calls the CLI_average_results_for_class
                Stats.CLI_average_results_for_class()

    def Main_Menu(self):
        #Sets up the while loop
        exit = False
        while not exit:
            #Prints each of the options
            print("A-Level Computing Assignment Monitor")
            print("")
            print("Please select the option for what you would like to do")
            print("")
            print("1. Student Management")
            print("2. Class Management")
            print("3. Assignment Management")
            print("4. Administration")
            print("5. Statistics")
            print("")
            print("0. Exit")
            print("")
            #Calls the Get_Option function for verification and error checking
            option = self.Get_Option([0,1,2,3,4,5])
            if option == 1:
                #Calls the Student Management function
                self.Student_Management()
            elif option == 2:
                self.Class_Management()
            elif option == 3:
                self.Assignment_Management()
            elif option == 4:
                self.Administration()
            elif option == 5:
                self.Statistics()
            elif option == 0:
                #Changes exit to true, therefore ends the program
                exit = True

#Checks to see if this file is the file started
if __name__ == '__main__':
    #Instantiate the CLI Class and run it
    cli = CLI_Class()
    cli.Main_Menu()
```

10.6 CLI_Administration.py

```
#Imports the required python modules
from teacher_controller import *
import sys

#Creates the class
class CLI_Administration_Class(teacher_controller):
    """CLI Administration"""

    def __init__(self):
        #Inherites teacher_controller on instantiation
        super().__init__()

    def print_table(self,headings,data):
        #This function prints a table for a list/view function
        #Sets up a blank array for the attributes
        table_attributes = []
        #Processes each heading
        for each in headings:
            #Puts it into a variable to simplify things
            heading = each[1]
            #Removes the first several characters
            heading = heading[7:]
            #Appends it to the table_attributes list
            table_attributes.append(heading)
        #Creates a blank headings string
        headings = ""
        #Processes each attribute
        for count in range(len(table_attributes)):
            #Calculates the length of the text and adds 5 for visibility
            length = len(table_attributes[count]) + 5
            #Adds additional space for emails
            if count == 4:
                headings = headings + '{0[' + str(count) + ']:<19}'
            #Adds standard space for other variables
            else:
                headings = headings + '{0[' + str(count) + ']:<13}'
        #Prints the headings
        print(headings.format(table_attributes))
        #Processes each row
        for each in data:
            #Creates a blank string for the row
            result = ""
            #Processes each "cell"
            for count in range(len(each)):
                #constructs the "cell", adds more space for the email
                if count == 4:
                   result = result + '{0[' + str(count) + ']:<19}'
                else:
                    result = result + '{0[' + str(count) + ']:<13}'
            #Prints the row
            print(result.format(each))

    def get_teacher_question(self,variable):
        #Checks the variable, returns correct question
        if variable == "TeacherID":
            return "Please enter the Teacher's ID: "
        elif variable == "TeacherUserName":
            return "Please enter the Teacher's UserName: "
        elif variable == "TeacherLastName":
            return "Please enter the Teacher's Last Name: "
        elif variable == "TeacherFirstName":
            return "Please enter the Teacher's First Name: "
        elif variable == "TeacherEmail":
            return "Please enter the Teacher's Email address: "
        elif variable == "TeacherAdmin":
            return "Please enter the Teacher's Admin status: "
        elif variable == "TeacherPassword":
            return "Please enter the new Password: "
        elif variable == "TeacherAdditionalPassword":
            return "Please enter the new Additional Password: "
        elif variable == "TeacherQuestion":
            return "Please enter the new password recovery question: "
        elif variable == "TeacherAnswer":
            return "Please enter the new password recovery answer: "
        elif variable == "TeacherAdmin":
            return "Please enter the admin status of the teacher (Y,N): "

    def check_teacher_variable(self,variable,function,entry):
        #Checks to see if the function is not an add function
        #and therefore whether it should allow blank entries or not
        if (function == "find" or function == "edit") and entry == "":
            return True,None
        #Checks to see if it matches a variable
        elif variable == "TeacherAdmin":
            #Check to see if valid. If so, it'll return True to move on
            if entry == "Y":
                return True,True
            else:
                return True,False
        else:
            return True,entry

    def teacher_variable(self,variable,function):
        #combines the teacher question and check variable into an easy while loop
        question = self.get_teacher_question(variable)
        valid = False
        while not valid:
            entry = input(question)
            valid,entry = self.check_teacher_variable(variable,function,entry)
        return entry

    def CLI_list_teacher(self):
        print("List Teacher Function")
        #Gets a list of teachers, removes the passwords and the other secure items and prints
        data = self.find_teacher()
        headings = self.teacher_headings()
        headings.pop(10)
        headings.pop(9)
        headings.pop(8)
        headings.pop(4)
        headings.pop(2)
        self.print_table(headings,data)

    def CLI_view_teacher(self):
        print("View Teacher Function")
        #Asks the user to optionally input parameters
        TeacherID = self.teacher_variable("TeacherID","find")
        TeacherUserName = self.teacher_variable("TeacherUserName","find")
        TeacherLastName = self.teacher_variable("TeacherLastName","find")
        TeacherFirstName = self.teacher_variable("TeacherFirstName","find")
        TeacherEmail = self.teacher_variable("TeacherEmail","find")
        #Performs database query
        data = self.find_teacher(TeacherID,TeacherUserName,TeacherLastName,
                                 TeacherFirstName,TeacherEmail)
        #Prepares headings (including removing some un-needed headings and prints table
        headings = self.teacher_headings()
        headings.pop(10)
        headings.pop(9)
        headings.pop(8)
        headings.pop(4)
        headings.pop(2)
        self.print_table(headings,data)

    def CLI_add_teacher(self):
        print("Add Teacher Function")
        print("To add a new Teacher, please enter the following details in")
        #Asks the user to input variables. All fields are required.
        TeacherUserName = self.teacher_variable("TeacherUserName","add")
        TeacherLastName = self.teacher_variable("TeacherLastName","add")
        TeacherFirstName = self.teacher_variable("TeacherFirstName","add")
        TeacherEmail = self.teacher_variable("TeacherEmail","add")
        TeacherPassword = self.teacher_variable("TeacherPassword","add")
        TeacherAdditionalPassword = self.teacher_variable("TeacherAdditionalPassword","add")
        TeacherQuestion = self.teacher_variable("TeacherQuestion","add")
        TeacherAnswer = self.teacher_variable("TeacherAnswer","add")
        TeacherAdmin = self.teacher_variable("TeacherAdmin","add")
        #Adds data to the database
        self.add_teacher(TeacherUserName, TeacherPassword, TeacherAdmin,
            TeacherAdditionalPassword, TeacherLastName, TeacherFirstName,
            TeacherEmail, TeacherQuestion, TeacherAnswer, "0000-00-00")
        print("Sucessfully added")


    def CLI_edit_teacher(self):
        print("Edit a Teacher")
        print("")
        #Asks the user if they have an ID for the teacher already
        haveID = input("Do you have the ID of the Teacher you wish to edit (Y/N): ")
        if haveID == "N":
            #If not, performs the view teacher function for them to find it
            self.CLI_view_teacher()
        #Asks the user for the ID
        ID = input("Please enter the ID of the Teacher you wish to edit:")
        #Gets details on teacher and prints them in a table
        data = self.find_teacher(TeacherID=ID)
        headings = self.teacher_headings()
        headings.pop(10)
        headings.pop(9)
        headings.pop(8)
        headings.pop(4)
        headings.pop(2)
        self.print_table(headings,data)
        print("If you do not wish to edit an item, leave it blank")
        #Asks the user for variables to optionally change
        TeacherUserName = self.teacher_variable("TeacherUserName","edit")
        TeacherLastName = self.teacher_variable("TeacherLastName","edit")
        TeacherFirstName = self.teacher_variable("TeacherFirstName","edit")
        TeacherEmail = self.teacher_variable("TeacherEmail","edit")
        TeacherPassword = self.teacher_variable("TeacherPassword","edit")
        TeacherAdditionalPassword = self.teacher_variable("TeacherAdditionalPassword","edit")
        TeacherQuestion = self.teacher_variable("TeacherQuestion","edit")
        TeacherAnswer = self.teacher_variable("TeacherAnswer","edit")
        TeacherAdmin = self.teacher_variable("TeacherAdmin","edit")
        #Performs database query
        self.edit_teacher(ID,TeacherUserName, TeacherPassword, TeacherAdmin,
                    TeacherAdditionalPassword, TeacherLastName, TeacherFirstName,
                    TeacherEmail, TeacherQuestion, TeacherAnswer)

        print("Successful")


    def CLI_delete_teacher(self):
        print("Delete Teacher Function")
        #Asks the user if they have an ID for the teacher already
        haveID = input("Do you have the ID of the Teacher you wish to remove (Y/N): ")
        if haveID == "N":
            #If not, performs the view teacher function for them to find it
            self.CLI_view_teacher()
        #Asks the user for the ID
        ID = input("Please enter the ID of the teacher you wish to remove:")
        #Performs Database Query
        self.delete_teacher(ID)
        print("Deletion Successful")

    def CLI_Email_settings(self):
        #Stub function for email settings to be introduced later
        print("Edit Email Settings")
```

10.7 CLI_Assignment_Management.py

```python
#Imports the required python modules
from assignment_controller import *

#Creates the class
class CLI_Assignment_Manager_Class(assignment_controller):
    """CLI Assignment Manager"""

    def __init__(self):
        #Inherites assignment_manager on instantiation
        super().__init__()

    def get_assignment_question(self,variable):
        #Checks the variable, returns correct question
        if variable == "AssignmentID":
            return "Please enter the Assignment's ID: "
        elif variable == "AssignmentName":
            return "Please enter the Assignment's Name: "
        elif variable == "AssignmentDescription":
            return "Please enter the Assignment's Description: "
        elif variable == "AssignmentStart":
            return "Please enter the Assignment's Start Date: "
        elif variable == "AssignmentDeadline":
            return "Please enter the Assignment's Deadline: "
        elif variable == "AssignmentMaxMark":
            return "Please enter the Assignment's Max Mark: "
        elif variable == "AssignmentYear":
            return "Please enter the Year the assignment belongs to (1 or 2): "


    def check_assignment_variable(self,variable,function,entry):
        #Checks to see if the function is not an add function
        #and therefore whether it should allow blank entries or not
        if (function == "find" or function == "edit") and entry == "":
            return True,None
        #Checks to see if it matches a variable
        elif variable in ['AssignmentStart','AssignmentDeadline']:
            #Checks to see if valid. If so, it'll return True to move on
            try:
                #It does this by converting to a date time variable and back again
                entry = datetime.datetime.strptime(entry,"%Y-%m-%d").strftime("%Y-%m-%d")
                return True,entry
            except:
                return True,False
        else:
            return True,entry

    def assignment_variable(self,variable,function):
        #combines the assignment question and check variable into an easy while loop
        question = self.get_assignment_question(variable)
        valid = False
        while not valid:
            entry = input(question)
            valid,entry = self.check_assignment_variable(variable,function,entry)
            if not valid:
                print("That is not a valid entry. Please try again")
            return entry

    def print_table(self,headings,data):
        #This function prints a table for a list/view function
        #Sets up a blank array for the attributes
        table_attributes = []
        #Processes each heading
       for each in headings:
            #Puts it into a variable to simplify things
            heading = each[1]
            #Removes the first several characters
            heading = heading[10:]
            #Appends it to the table_attributes list
            table_attributes.append(heading)
```

```
         #Creates a blank headings string
         headings = ""
         #Processes each attribute
        for count in range(len(table_attributes)):
             #Calculates the length of the text and adds 5 for visibility
             length = len(table_attributes[count]) + 5
             #Adds additional space for easy viewing
             if count == 4:
                 headings = headings + '{0[' + str(count) + ']:<19}'
             else:
                 headings = headings + '{0[' + str(count) + ']:<13}'
         #Prints the headings
         print(headings.format(table_attributes))
          #Processes each row
        for each in data:
             #Creates a blank string for the row
            result = ""
             #Processes each "cell"
             for count in range(len(each)):
                 #constructs the "cell",
                 if count == 4:
                    result = result + '{0[' + str(count) + ']:<19}'
                 else:
                     result = result + '{0[' + str(count) + ']:<13}'
             #Prints the row
             print(result.format(each))

    def CLI_list_assignment(self):
        print("List Assignment Function")
        #Gets a list of assignments and prints
        data = self.find_assignment()
        headings = self.assignment_headings()
        self.print_table(headings,data)

    def CLI_view_assignment(self):
        print("View Assignment Function")
        print("Please input the details that you would like to use to find the assignment.")
        print("You can leave any value blank if you do not know it")
        print("")
        #Asks the user to optionally input parameters
        AssignmentID = self.assignment_variable("AssignmentID","find")
        AssignmentName = self.assignment_variable("AssignmentID","find")
        AssignmentDescription = self.assignment_variable("AssignmentDescription","find")
        AssignmentStart = self.assignment_variable("AssignmentStart","find")
        AssignmentDeadline = self.assignment_variable("AssignmentDeadline","find")
        AssignmentMaxMark = self.assignment_variable("AssignmentMaxMark","find")
        AssignmentYear = self.assignment_variable("AssignmentYear","find")
        #Performs database query
        data = self.find_assignment(AssignmentID,AssignmentName,AssignmentDescription,
                                    AssignmentStart,AssignmentDeadline,AssignmentMaxMark,
                                    AssignmentYear)
        #Prepares headings (including removing some un-needed headings and prints table
        headings = self.assignment_headings()
        self.print_table(headings,data)

    def CLI_add_assignment(self):
        print("Add Assignment Function")
        #Asks the user to input variables. All fields are required.
        AssignmentName = self.assignment_variable("AssignmentName","add")
        AssignmentDescription = self.assignment_variable("AssignmentDescription","add")
        AssignmentStart = self.assignment_variable("AssignmentStart","add")
        AssignmentDeadline = self.assignment_variable("AssignmentDeadline","add")
        AssignmentMaxMark = self.assignment_variable("AssignmentMaxMark","add")
        AssignmentYear = self.assignment_variable("AssignmentYear","add")
        #Adds data to the database
        self.add_assignment(AssignmentName,AssignmentDescription,
                            AssignmentStart,AssignmentDeadline,AssignmentMaxMark,
                            AssignmentYear)

    def CLI_edit_assignment(self):
        print("Edit an Assignment")
        print("")
        #Asks the user if they have an ID for the assignment already
        haveID = input("Do you have the ID of the Assignment you wish to edit (Y/N): ")
        if haveID == "N":
            #If not, performs the view assignment function for them to find it
            self.CLI_view_assignment()
        #Asks the user for the ID
        ID = input("Please enter the ID of the Assignment you wish to edit:")
        #Gets details on teach
er and prints them in a table
        data = self.find_assignment(AssignmentID=ID)
        headings = self.assignment_headings()
        self.print_table(headings,data)
        print("If you do not wish to edit an item, leave it blank")
        #Asks the user for variables to optionally change
        AssignmentName = self.assignment_variable("AssignmentID","edit")
        AssignmentDescription = self.assignment_variable("AssignmentDescription","edit")
        AssignmentStart = self.assignment_variable("AssignmentStart","edit")
        AssignmentDeadline = self.assignment_variable("AssignmentDeadline","edit")
        AssignmentMaxMark = self.assignment_variable("AssignmentMaxMark","edit")
        AssignmentYear = self.assignment_variable("AssignmentYear","edit")
        #Performs database query
        self.edit_assignment(AssignmentID, AssignmentName,AssignmentDescription,
                        AssignmentStart,AssignmentDeadline,AssignmentMaxMark,
                        AssignmentYear)


        print("Successful")

    def CLI_delete_assignment(self):
        print("Delete Assignment Function")
        #Asks the user if they have an ID for the assignment already
        haveID = input("Do you have the ID of the Assignment you wish to remove (Y/N): ")
        if haveID == "N":
            #If not, performs the view assignment function for them to find it
            self.CLI_view_assignment()
        #Asks the user for the ID
        ID = input("Please enter the ID of the Assignment you wish to remove:")
        #Performs Database Query
        self.delete_assignment(ID)
        print("Deletion Successful")
```

10.8 CLI_Assignment_Results_Management.py

```
#Imports the required python modules
from assignment_results_controller import *
from student_controller import *
from CLI_Student_Management import *
from CLI_Assignment_Management import *

#Creates the class
class CLI_Assignment_Results_Manager_Class(assignment_results_controller):
    """CLI Assignment Manager"""

    def __init__(self):
        #Inherites assignment_manager on instantiation
        super().__init__()

    def get_assignment_results_question(self,variable):
        #Checks the variable, returns correct question
        if variable == "StudentID":
            return "Please enter the student's ID: "
        elif variable == "AssignmentID":
            return "Please enter the assignment's ID: "
        elif variable == "AssignmentMark":
            return "Please enter the mark: "
        elif variable == "AssignmentNotes":
            return "Please enter any notes on the assignment: "


    def check_assignment_results_variable(self,variable,function,entry):
        #Checks to see if the function is not an add function
        #and therefore whether it should allow blank entries or not
        if (function == "find" or function == "edit") and entry == "":
            return True,None
        #Checks to see if it matches a variable
        elif variable in ["StudentID","AssignmentID","AssignmentMark"]:
            #Checks to see if valid. If so, it'll return True to move on
            try:
                entry = int(entry)
                return True,entry
            except ValueError:
                return False,entry
        #Due to notes being optional and can contain anything, this will always be valid
        elif variable == "AssignentNotes":
            return True,entry

    def assignment_result_variable(self,variable,function):
        #combines the assignment question and check variable into an easy while loop
        question = self.get_assignment_results_question(variable)
        valid = False
        while not valid:
            entry = input(question)
            print(variable,function,entry)
            valid,entry = self.check_assignment_results_variable(variable,function,entry)
        return entry

    def CLI_View_Assignment_Result(self):
        print("View Assignment Result Function")
        print("")
        #Asks the user if they have an ID for the student already
        haveStudentID = input("Do you have the Student ID? (Y/N) ")
        if haveStudentID != "Y":
            #If not, temporarily instantiates the class and
            #performs the view student function
            SM = CLI_Student_Manager_Class()
            SM.CLI_view_student()
        #Asks the user to input the ID
        StudentID = input("Please enter the Student ID: ")
        #Repeats for AssignmentID
        haveAssignmentID = input("Do you have the Assignment ID? (Y/N) ")
        if haveAssignmentID != "Y":
            AM = CLI_Assignment_Manager_Class()
            AM.CLI_view_assignment()
        AssignmentID = input("Please enter the Assignment ID: ")
        #Gets the result. In try and except incase the result doesn't exist
        try:
            print("The Student got ",self.get_assignment_result(StudentID,AssignmentID))
        except IndexError:
            print("This entry does not exist yet. You may need to create it first.")

    def CLI_Add_Assignment_Result(self):
        print("Add Assignment Result Function")
        print("")
        print("To add a result to a student's assignment, please enter the following details")
        #Asks the user to input the details. All are required.
        StudentID = self.assignment_result_variable("StudentID","add")
        AssignmentID = self.assignment_result_variable("AssignmentID","add")
        AssignmentMark = self.assignment_result_variable("AssignmentMark","add")
        AssignmentNotes = self.assignment_result_variable("AssignmentNotes","add")
        #Performs the DB query
        self.add_assignment_result(StudentID,AssignmentID,AssignmentMark,AssignmentNotes)
        print("Successful")

    def CLI_Edit_Assignment_Result(self):
        print("Edit Assignment Result Function")
        print("")
        #Asks the user if they have an ID for the student already
        haveStudentID = input("Do you have the Student ID? (Y/N) ")
        if haveStudentID != "Y":
            #If not, temporarily instantiates the class and
            #performs the view student function
            SM = CLI_Student_Manager_Class()
            SM.CLI_view_student()
        #Asks the user to input the ID
        StudentID = input("Please enter the Student ID: ")
        #Repeats for AssignmentID
        haveAssignmentID = input("Do you have the Assignment ID? (Y/N) ")
        if haveAssignmentID != "Y":
            AM = CLI_Assignment_Manager_Class()
            AM.CLI_view_assignment()
        AssignmentID = input("Please enter the Assignment ID: ")
        #In try and except in case the value doesn't exist yet
        try:
            #Prints the current value
            print("The Student got ",self.get_assignment_result(StudentID,AssignmentID))
            print("Please now enter the new values. Leave a line blank to not touch it")
            #Asks the user to enter the new value
            AssignmentMark = self.assignment_result_variable("AssignmentMark","edit")
            AssignmentNotes = self.assignment_result_variable("AssignmentNotes","edit")
            #Performs the DB query
            self.edit_assignment_results(StudentID,AssignmentID,AssignmentMark,AssignmentNotes)
            print("Successful")
        except IndexError:
            print("This entry does not exist yet. You need to create it before you can edit it.")

    def CLI_Delete_Assignment_Result(self):
        #Asks the user if they have an ID for the student already
        haveStudentID = input("Do you have the Student ID? (Y/N) ")
        if haveStudentID != "Y":
            #If not, temporarily instantiates the class and
            #performs the view student function
            SM = CLI_Student_Manager_Class()
            SM.CLI_view_student()
        #Asks the user to input the ID
        StudentID = input("Please enter the Student ID: ")
        #Repeats for AssignmentID
        haveAssignmentID = input("Do you have the Assignment ID? (Y/N) ")
        if haveAssignmentID != "Y":
            AM = CLI_Assignment_Manager_Class()
            AM.CLI_view_assignment()
        AssignmentID = input("Please enter the Assignment ID: ")
        #Performs the DB query
        self.delete_assignment_result(StudentID,AssignmentID)
        print("Successful")
```

10.9 CLI_Class_Management.py

```python
#Imports the required python modules
from class_controller import *
from student_controller import *
from class_students_controller import *

#Creates the class
class CLI_Class_Manager_Class(class_controller,class_students_controller):
    """CLI Class Manager"""

    def __init__(self):
        #Inherites class_controller and class_students_controller on instantiation
        super().__init__()

    def Get_Option(self,list):
        #Checks the variable, returns correct question
        try:
            option = int(input("Please enter your choice: "))
            if option in list:
                return option
            else:
                print("Please choose an option on the list")
                return self.Get_Option(list)
        except:
            print("That is not a valid integer. Please try again")
            return self.Get_Option(list)

    def get_class_question(self,variable):
        #Checks the variable, returns correct question
        if variable == "ClassID":
            return "Please enter the Class ID: "
        elif variable == "TeacherID":
            return "Please enter the Class's Teacher ID: "
        elif variable == "Year":
            return "Please enter which year the class is: "
        elif variable == "YearStart":
            return "Please enter the start year of the class: "

    def check_class_variable(self,variable,function,entry):
        #Checks to see if the function is not an add function
        #and therefore whether it should allow blank entries or not
        if (function == "find" or function == "edit") and entry == "":
            return True,None
        #Checks to see if it matches a variable
        elif variable == "ClassID":
            #tries to put it into an integer.
            try:
                if int(entry) > -1:
                    return True,entry
                else:
                    return False,entry
            except:
                return False,error
        elif variable == "TeacherID":
            try:
                if int(entry) > -1:
                    return True,entry
                else:
                    return False,entry
            except:
                return False,error
        elif variable == "Year":
            try:
                if int(entry) > -1:
                    return True,entry
                else:
                    return False,entry
            except:
                return False,error
        elif variable == "YearStart":
            try:
                if int(entry) in range(1990,2030):
                    return True,entry
                else:
                    return False,entry
            except:
                return False,error

    def print_table(self,headings,data):
        #This function prints a table for a list/view function
        #Sets up a blank array for the attributes
        table_attributes = []
        #Processes each heading
        for each in headings:
            #Puts it into a variable to simplify things
            heading = each[1]
            #Appends it to the table_attributes list
            table_attributes.append(heading)
        #Creates a blank headings string
        headings = ""
        #Processes each attribute
        for count in range(len(table_attributes)):
            #Calculates the length of the text and adds 5 for visibility
            length = len(table_attributes[count]) + 5
            headings = headings + '{0[' + str(count) + ']:<10}'
        #Prints the headings
        print(headings.format(table_attributes))
        #Processes each row
        for each in data:
            #Creates a blank string for the row
            result = ""
            #Processes each "cell"
            for count in range(len(each)):
                 #constructs the "cell", adds more space for readability
               result = result + '{0[' + str(count) + ']:<10}'
```

```
                #Prints the row
                print(result.format(each))

        def class_variable(self,variable,function):
            #combines the assignment question and check variable into an easy while loop
            question = self.get_class_question(variable)
            valid = False
            while not valid:
                entry = input(question)
                valid,entry = self.check_class_variable(variable,function,entry)
                if not valid:
                    print("That is not a valid entry. Please try again")
            return entry

        def CLI_list_class(self):
            print("List Class Function")
            #Gets a list of classes and prints
            data = self.find_class()
            headings = self.class_headings()
            self.print_table(headings,data)

        def CLI_view_class(self):
            print("View Class Function")
            #Asks the user to optionally input parameters
            print("Please input the details that you would like to use to find a class.")
            print("You can leave any value blank if you do not know it")
            print("")
            ClassID = self.class_variable("ClassID","find")
            TeacherID = self.class_variable("TeacherID","find")
            Year = self.class_variable("Year","find")
            YearStart = self.class_variable("YearStart","find")

            #Performs database query
            data = self.find_class()
            #Gets headers and prints data
            headings = self.class_headings()
            self.print_table(headings,data)


        def CLI_add_class(self):
            print("Add Class Function")
            print("To add a new class, please enter the following details in")
            #Asks the user to input the details. All are required.
            TeacherID = self.class_variable("TeacherID","add")
            Year = self.class_variable("Year","add")
            YearStart = self.class_variable("YearStart","add")
            #Performs the DB query
            self.add_class(TeacherID,Year,YearStart)
            print("Sucessfully added")

        def CLI_edit_class(self):
            print("Edit Class Function")
            print("")
            #Asks the user if they have an ID for the class already
            haveID = input("Do you have the ID of the Class you wish to edit (Y/N): ")
            if haveID == "N":
                #Calls view class function for the user to find the id
                self.CLI_view_class()
            #Asks the user to input the ID
            ID = input("Please enter the ID of the Class you wish to edit:")
            #Performs the DB query and prints data
            data = self.find_class(ClassID=ID)
            headings = self.class_headings()
            self.print_table(headings,data)
            print("If you do not wish to edit an item, leave it blank")
            #Asks user to optionally enter values to change
            TeacherID = self.class_variable("TeacherID","edit")
            Year = self.class_variable("Year","edit")
            YearStart = self.class_variable("YearStart","edit")
            #Performs the DB query
            self.edit_class(ClassID,TeacherID,Year,YearStart)
            print("Successful")

        def CLI_delete_class(self):
            print("Delete Class Function")
            #Asks the user if they have an ID for the class already
            haveID = input("Do you have the ID of the Class you wish to remove (Y/N): ")
            if haveID == "N":
                #Calls view class function for the user to find the id
                self.CLI_view_class()
            #Asks the user to input the ID
            ID = input("Please enter the ID of the Class you wish to remove:")
            #Performs the DB query
            self.delete_class(ID)
            print("Deletion Successful")

        def CLI_view_students_in_a_class(self):
            print("View students in a class function")
            print("")
            #Asks the user if they have an ID for the class already
            haveID = input("Do you have the class ID? (Y/N) ")
            if haveID == "N":
                #Calls view class function for the user to find the id
                self.CLI_view_class()
            #Asks the user to input the ID
            ID = input("Please input the class ID: ")
            #Performs the DB query
            students = self.get_students_in_class(ID)
            #Checks to see if there are students in the class (variables in the array)
            if len(students) == 0:
                print("No students are in this class")
            else:
                print("The following student IDs are in Class", ID)
                for each in students:
                    print(each)

        def CLI_add_to_class(self):
            print("Add Student to Class function")
            print("")
            #Asks the user if they have an ID for the student already
            haveStudent = input("Do you have the Student's ID? (Y/N) ")
            if haveStudent == "N":
                #If not, temporarily instantiates the class and
                #performs the view student function
                sc = student_controller()
                sc.view_student()
            #Asks the user to input the ID
            StudentID = input("Plese input the Student's ID: ")
            #Asks the user if they have an ID for the class already
            haveClassID = input("Do you have the Class's ID? (Y/N) ")
            if haveClassID == "N":
                #If not, temporarily instantiates the class and
                #performs the view class function
                cc = class_controller()
                cc.view_class()
            #Asks the user to input the ID
            ClassID = input("Please input the Class's ID: ")
            #Performs the DB Query
            self.add_class_student(ClassID,StudentID)
            print("Successful")

        def CLI_remove_from_class(self):
            print("Remove Student from Class function")
            print("")
            #Asks the user if they have an ID for the student already
            haveStudent = input("Do you have the Student's ID? (Y/N) ")
            if haveStudent == "N":
                #If not, temporarily instantiates the class and
                #performs the view student function
                sc = student_controller()
                sc.view_student()
            #Asks the user to input the ID
            StudentID = input("Plese input the Student's ID: ")
            #Asks the user if they have an ID for the class already
            haveClassID = input("Do you have the Class's ID? (Y/N) ")
            if haveClassID == "N":
                #If not, temporarily instantiates the class and
                #performs the view class function
                cc = class_controller()
                cc.view_class()
            #Asks the user to input the ID
            ClassID = input("Please input the Class's ID: ")
            #Performs the DB Query
            self.delete_class_student(ClassID,StudentID)
            print("Successful")
```

10.10 CLI_Statistics.py

```
    #Imports the required python modules

from assignment_controller import *
from assignment_results_controller import *
from class_controller import *
from class_students_controller import *
from student_controller import *
from teacher_controller import *


#Creates the class
class CLI_Statistics_Class(assignment_controller,assignment_results_controller,
                           class_controller, class_students_controller,
                           student_controller, teacher_controller):
    """CLI Statistics Manager"""

    def __init__(self):
        #Inherits the classes listed above.
        super().__init__()

    def get_average_results_for_class(self,ClassID):
        #Gets the year of the class specified
        year = self.get_class_year(ClassID)
        #Gets assignments that the year applies to
        assignments = self.get_assignments_for_year(year)
        #Gets the student's IDs of the class specified
        students = self.get_students_in_class(ClassID)
        #Sets up blank list for loop to dump data into
        data = []
        #Loops round for each student
        for eachstudent in students:
            #Creates a blank array for their assignment results
            student_assignment_results = []
```

```
      #Loops round for each assignent
      for eachassignment in assignments:
        #Performs database query to get each result and appends to database
        student_assignment_results.append(self.get_assignment_result(eachstudent,eachassignment))
      #Appends each array to the overall data array
      data.append(student_assignment_results)
    #creates arrays for average column
    total_results = []
    average_results = []
    #Loops round each item in assignments
    for countassignment in range(len(assignments)):
      #Appends a "0" the beginning of the counter
      total_results.append(0)
      #Loops round each student
      for countstudent in range(len(student_assignment_results)):
        #Adds the students result to the total results
        total_results[countassignment] = total_results[countassignment] +
data[countstudent][countassignment]
      #Gets the average result
      average_results.append((total_results[countassignment]/len(students)))
    return average_results,assignments

  def CLI_average_results_for_class(self):
    #Asks the user to input the class ID
    ClassID = input("Please input the class ID ")
    #Gets the average results
    avg_results,assignments = self.get_average_results_for_class(ClassID)
    #Asks the user if they want it in percentages or values
    in_percentage = input("Would you like the average results in terms of percentages? (Y/N)  ")
    #Starts blank array for assignment information
    assignment_info = []
    #Loops round each assignment
    for count in range(len(assignments)):
      #Gets the assignment information for ease of display purposes
      results = self.find_assignment(assignments[count])
      #begins a templist
```

```
        templist = []

        for each in results[0]:
            templist.append(each)
            #Appends each avg result to the array
            templist.append(avg_results[count])
            #Appends each assignment to the main assignment_info array
            assignment_info.append(templist)
    if in_percentage == "Y":
        #Creates table headings
        headings = ["Assignment ID","Assignment Name","Avg Percentage"]
        #Creates data array
        data = []
        #For each assignment, calculates the percentage and appends id, name and % to data array
        for each in assignment_info:
            percentage = "{0}%".format((each[7] / each[5]) * 100)
            data.append([each[0],each[1],percentage])
        ####Printing Table####
        #Creates a blank headings string
        heading = ""
        #Processes each attribute, adding spacing for ease of viewing
        for count in range(len(headings)):
            heading = heading + '{0[' + str(count) + ']:<16}'
        #Prints the heading
        print(heading.format(headings))
        #For each row
        for each in data:
            #Creates a blank string for the row
            result = ""
            #Processes each "cell"
            for count in range(len(each)):
                #constructs the "cell", adds more space for readability
                result = result + '{0[' + str(count) + ']:<16}'
            #prints each row
            print(result.format(each))
    else:
```

```python
            #Creates table headings
            headings = ["Assignment ID","Assignment Name","Average Mark","Max Mark"]
            #Creates data array
            data = []
            #For each assignment, it appends the id, name, Avg mark and Max Mark to data array
            for each in assignment_info:
                data.append([each[0],each[1],each[7],each[5]])
            ####Printing Table####
            #Creates a blank headings string
            heading = ""
            #Processes each attribute, adding spacing for ease of viewing
            for count in range(len(headings)):
                heading = heading + '{0[' + str(count) + ']:<16}'
            #Prints the heading
            print(heading.format(headings))
            #For each row
            for each in data:
                #Creates a blank string for the row
                result = ""
                #Processes each "cell"
                for count in range(len(each)):
                    #constructs the "cell", adds more space for readability
                    result = result + '{0[' + str(count) + ']:<16}'
                #prints each row
                print(result.format(each))
        print("")

    def predicted_result_for_student(self):
        #Incomplete stub function for processing a prediction on what a
        #students grade could be from a "moving 3 point average".
        A = 90
        B = 80
        C = 70
        D = 60
        E = 50
        print("Predicted Results Function")
        print("This gives a prediction on the grade based on the last 3 assignments")

#For testing purposes, checks to see if this function is being directly ran
#and runs the average results function with the first class
if __name__ == "__main__":
    stats = CLI_Statistics_Class()
    stats.average_results_for_class(1)
```

10.11 CLI_Student_Manager.py

```python
#Imports the required python modules
from student_controller import *
import re
import sys
import datetime

#Creates the class
class CLI_Student_Manager_Class(student_controller):
    """CLI Student Manager"""

    def __init__(self):
        #Inherites student_controller on instantiation
        super().__init__()

    def Get_Option(self,list):
            #Try is there to avoid the program crashing
        try:
            #Attempts to put the option into an integer
            #This removes possibilities of letter/blank options
            option = int(input("Please enter your choice: "))
            #Checks to see if the option is in the list
            if option in list:
                #Gives the option back to the user
                return option
            else:
                #Prints an error
                print("Please choose an option on the list")
                return self.Get_Option(list)
        except:
            #Prints an error if the int function fails.
            print("That is not a valid integer. Please try again")
```

```
                return self.Get_Option(list)

    def get_student_question(self,variable):
        #Checks the variable, returns correct question
        if variable == "StudentID":
            return "Please enter the student's ID: "
        elif variable == "StudentLastName":
            return "Please enter the student's Last Name: "
        elif variable == "StudentFirstName":
            return "Please enter the student's First Name: "
        elif variable == "StudentDOB":
            return "Please enter the student's DOB in format YYYY-MM-DD: "
        elif variable == "StudentEmail":
            return "Please enter the student's email address: "
        elif variable == "StudentScribe":
            return "Does the student have a scribe? (Y/N) "
        elif variable == "Student25Extra":
            return "Does the student have 25% extra time? (Y/N) "
        elif variable == "Student50Extra":
            return "Does the student have 50% extra time? (Y/N) "
        elif variable == "StudentWordProcessor":
            return "Does the student use a Word Processor? (Y/N) "
        elif variable == "StudentGCSEResults":
            return "Please enter the student's GCSE Results value: "
        elif variable == "StudentNotes":
            return "Please enter the student's notes: "

    def check_student_variable(self,variable,function,entry):
        #Checks to see if the function is not an add function
        #and therefore whether it should allow blank entries or not

        if (function == "find" or function == "edit") and entry == "":
            return True,None
        #Checks to see if it matches a variable
        elif variable == "StudentID":
            try:
                #Attempts to do an int, will deny if failed
                entry = int(entry)
                return True,entry
            except ValueError:
                return False,entry
        elif variable == "StudentLastName":
            #Length checks the name
            if len(entry) > 2:
                return True,entry
            else:
                return False,entry
        elif variable == "StudentFirstName":
            #Length checks the name
            if len(entry) > 2:
                return True,entry
            else:
                return False,entry
        elif variable == "StudentDOB":
            try:
                ##Attempts to convert the entry to a datetime variable and back. If successfull, it's valid.
                entry = datetime.datetime.strptime(entry,"%Y-%m-%d").strftime("%Y-%m-%d")
                return True,entry
            except ValueError:
                print("That is not an acceptable format for a date")
                return False,entry
        elif variable == "StudentEmail":
            #Compares the input with a regular expression.
            if re.match("^.+@longroad.ac.uk$",entry):
                return True,entry
            else:
                print("That is not an acceptable format.")
                print("The email address must end in @longroad.ac.uk")
                return False,entry
        elif variable in ["StudentScribe","Student25Extra","Student50Extra","StudentWordProcessor"]:
            if entry == "Y":
                return True,True
            elif entry == "N":
                return True,False
            else:
                return False,True
        elif variable == "StudentGCSEResults":
            try:
                #Attempts to convert to a float variable type, will deny if failed
                entry = float(entry)
                return True,entry
            except ValueError:
                return False,entry
        elif variable == "StudentNotes":
            #Due to being notes, will always accept
            return True,entry




    def student_variable(self,variable,function):
        #combines the assignment question and check variable into an easy while loop
        question = self.get_student_question(variable)
        valid = False
        while not valid:
            entry = input(question)
            valid,entry = self.check_student_variable(variable,function,entry)
        return entry

    def print_table(self,headings,data):
        #This function prints a table for a list/view function
        #Sets up a blank array for the attributes
        table_attributes = []
        #Processes each heading
        for each in headings:
            #Puts it into a variable to simplify things
            heading = each[1]
            #Removes the first several characters
            heading = heading[7:]
            #Appends it to the table_attributes list
            table_attributes.append(heading)
        #Creates a blank headings string
        headings = ""
        #Processes each attribute
        for count in range(len(table_attributes)):
            #Calculates the length of the text and adds 5 for visibility
            length = len(table_attributes[count]) + 5
            #Adds additional space for easy viewing for email address
            if count == 4:
                headings = headings + '{0[' + str(count) + ']:<21}'
            else:
                headings = headings + '{0[' + str(count) + ']:<13}'
        #Prints the headings
        print(headings.format(table_attributes))
         #Processes each row
        for each in data:
            #Creates a blank string for the row
            result = ""
            #Processes each "cell"
            for count in range(len(each)):
                #constructs the "cell", adding additional space for email address
                if count == 4:
                   result = result + '{0[' + str(count) + ']:<21}'
                else:
                    result = result + '{0[' + str(count) + ']:<13}'
            #Prints the row
            print(result.format(each))

    def CLI_list_student(self):
        print("List Students")
        #Gets a list of students and prints

        data = self.find_student()
        headings = self.student_headings()
        self.print_table(headings,data)

    def CLI_view_student(self):
        print("View Student Function")
        print("Please input the details that you would like to use to find the student.")
        print("You can leave any value blank if you do not know it")
        print("")
        #Asks the user to optionally input parameters
        StudentID = self.student_variable("StudentID","find")
        StudentLastName = self.student_variable("StudentLastName","find")
        StudentFirstName = self.student_variable("StudentFirstName","find")
        StudentDOB = self.student_variable("StudentDOB","find")
        StudentEmail = self.student_variable("StudentEmail","find")
        StudentScribe = self.student_variable("StudentScribe","find")
        Student25Extra = self.student_variable("Student25Extra","find")
        Student50Extra = self.student_variable("Student50Extra","find")
        StudentWordProcessor = self.student_variable("StudentWordProcessor","find")
        StudentGCSEResults = self.student_variable("StudentGCSEResults","find")
        #Performs database query
        data = self.find_student(StudentID, StudentLastName, StudentFirstName,
                     StudentDOB, StudentEmail, StudentScribe, Student25Extra,
                     Student50Extra, StudentGCSEResults)
        headings = self.student_headings()
        self.print_table(headings,data)

    def CLI_add_student(self):
        print("Add Student Function")
        print("")
        print("To add a new student, please enter the following details in")
        #Asks the user to input variables. All fields are required.
        StudentLastName = self.student_variable("StudentLastName","add")
        StudentFirstName = self.student_variable("StudentFirstName","add")
        StudentDOB = self.student_variable("StudentDOB","add")
        StudentEmail = self.student_variable("StudentEmail","add")
        StudentScribe = self.student_variable("StudentScribe","add")
        Student25Extra = self.student_variable("Student25Extra","add")
        Student50Extra = self.student_variable("Student50Extra","add")
        StudentWordProcessor = self.student_variable("StudentWordProcessor","add")
        StudentGCSEResults = self.student_variable("StudentGCSEResults","add")
        try:
        #Adds data to the database
            self.add_student(StudentLastName, StudentFirstName, StudentDOB, StudentEmail,
                    StudentScribe, Student25Extra, Student50Extra,StudentWordProcessor, StudentGCSEResults,
                    StudentLastEmailed = "0000-00-00", StudentNotes = "")
            print("Sucessfully added")
        except:
            #If Failure, it will cleanly exit with an error message
           print("Could not successfully be added. The following error occured:", sys.exc_info())


    def CLI_edit_student(self):
        print("Edit a Student")
        print("")
        #Asks the user if they have an ID for the student already
        haveID = input("Do you have the ID of the student you wish to edit (Y/N): ")
        if haveID == "N":
            #If not, performs the view student function for them to find it
            self.CLI_view_student()
        #Asks the user for the ID
        ID = input("Please enter the ID of the Student you wish to edit:")
        #Gets details on student and prints them in a table
        data = self.find_student(StudentID=ID)
        headings = self.student_headings()
        self.print_table(headings,data)
        print("If you do not wish to edit an item, leave it blank")
        #Asks the user for variables to optionally change
        StudentLastName = self.student_variable("StudentLastName","edit")
        StudentFirstName = self.student_variable("StudentFirstName","edit")
        StudentDOB = self.student_variable("StudentDOB","edit")
        StudentEmail = self.student_variable("StudentEmail","edit")
        StudentScribe = self.student_variable("StudentScribe","edit")
        Student25Extra = self.student_variable("Student25Extra","edit")
        Student50Extra = self.student_variable("Student50Extra","edit")
        StudentWordProcessor = self.student_variable("StudentWordProcessor","edit")
        StudentGCSEResults = self.student_variable("StudentGCSEResults","edit")
        StudentNotes = self.student_variable("StudentNotes","edit")
        #Performs database query
        self.edit_student(StudentLastName, StudentFirstName,
                     StudentDOB, StudentEmail, StudentScribe, Student25Extra,
                     Student50Extra, StudentGCSEResults, StudentNotes)
        print("Successful")



    def CLI_delete_student(self):
        #Asks the user if they have an ID for the student already
        haveID = input("Do you have the ID of the student you wish to remove (Y/N): ")
        if haveID == "N":
            #If not, performs the view student function for them to find it
            self.CLI_view_student()
        #Asks the user for the ID
        ID = input("Please enter the ID of the Student you wish to remove:")
        #Performs database query
        self.delete_student(ID)
        print("Deletion Successful")


    def CLI_email_student(self):
        #Stub function for emailing students as a later feature
        print("Email Student Function")
```

10.12 controller_class.py

```python
#Imports sqlite3 so we can use a DB
import sqlite3

class database_controller():
    """The controller for the database for security and ease of programming"""

    #This sets up any values that I need for my class
    def __init__(self):
        #dbname is the name of my database. I only need to edit it here
        self.dbname = "database.db"

    def _query(self,sql):
        #This is for adding, editing or removing data from the database

        #Opens a connection to the db and creates the cursor
        self.db = sqlite3.connect(self.dbname)
        self.cursor = self.db.cursor()
        #Enables me to use foreign keys, as it's disabled by default
        self.cursor.execute("PRAGMA foreign_keys = ON")
        #Executes the command
        self.cursor.execute(sql)
        #Saves the changes to the database
        self.db.commit()
        #Closes the connection
        self.cursor.close()

    def _select_query(self,sql):
        #This is for viewing data in the database

        #Opens a connection to the db and creates the cursor
        self.db = sqlite3.connect(self.dbname)
        self.cursor = self.db.cursor()
        #Enables me to use foreign keys, as it's disabled by default
        self.cursor.execute("PRAGMA foreign_keys = ON")
        #Executes the command
        self.cursor.execute(sql)
        #Fetches the data from the db
        results = self.cursor.fetchall()
        #closes connection
        self.cursor.close()
        #Returns the data for processing
        return results
```

10.13 database_creation.py

```
#Imports the SQLite3 module
import sqlite3

def create_teacher_database(db,cursor):
    #Creates the SQL code for the teacher table
    sql = """create table Teacher (
            TeacherID integer,
            TeacherUserName text,
            TeacherPassword text,
            TeacherAdmin integer,
            TeacherAdditionalPassword text,
            TeacherLastName text,
            TeacherFirstName text,
            TeacherEMail text,
            TeacherQuestion text,
            TeacherAnswer text,
            TeacherLastEmailed text,
            primary key (TeacherID))"""
    #Executes the SQL
    cursor.execute(sql)
    #Coomits the changes to the DB
    db.commit()

def create_student_database(db,cursor):
    sql = """create table Student (
            StudentID integer,
            StudentLastName text,
            StudentFirstName text,
            StudentDOB text,
            StudentEMail text,
            StudentScribe integer,
            Student25Extra integer,
            Student50Extra integer,
            StudentWordProcessor integer,
            StudentGCSEResults real,
            StudentLastEmailed text,
            StudentNotes text,
            primary key (StudentID))"""
    cursor.execute(sql)
    db.commit()



def create_assignment_database(db,cursor):
    sql = """create table Assignment (
                AssignmentID integer,
                AssignmentName text,
                AssignmentDescription text,
                AssignmentStart text,
                AssignmentDeadline text,
                AssignmentMaxMark integer,
                AssignmentYear integer,
                primary key (AssignmentID))"""
    cursor.execute(sql)
    db.commit()

def create_assignment_results_database(db,cursor):
    sql = """create table Assignment_Results (
                StudentID integer,
                AssignmentID integer,
                AssignmentMark integer,
                AssignmentNotes text,
                primary key (StudentID, AssignmentID),
                foreign key (StudentID) references Student(StudentID) ON UPDATE CASCADE ON DELETE RESTRICT,
                foreign key (AssignmentID) references Assignment(AssignmentID) ON UPDATE CASCADE ON DELETE
RESTRICT)"""
```

```
    cursor.execute(sql)
    db.commit()

def create_class_database(db,cursor):
    sql = """create table Class (
            ClassID integer,
            TeacherID integer,
            Year integer,
            YearStart integer,
            primary key (ClassID),
            foreign key (TeacherID) references Teacher(TeacherID) ON UPDATE CASCADE ON DELETE RESTRICT)"""
    cursor.execute(sql)
    db.commit()

def create_class_students_database(db,cursor):
    sql = """create table Class_Students (
            ClassID integer,
            StudentID integer,
            primary key (ClassID, StudentID),
            foreign key (ClassID) references Class(ClassID) ON UPDATE CASCADE ON DELETE RESTRICT,
            foreign key (StudentID) references Student(StudentID) ON UPDATE CASCADE ON DELETE RESTRICT)"""
    cursor.execute(sql)
    db.commit()


if __name__ == '__main__':
    #Connects to the Database
    db = sqlite3.connect("database.db")
    #Sets up the cursor
    cursor = db.cursor()
    #Enables use of foreign keys, not on by default
    cursor.execute("PRAGMA foreign_keys = ON")
    #creates the tables
    create_teacher_database(db,cursor)
    create_student_database(db,cursor)
    create_assignment_database(db,cursor)
    create_assignment_results_database(db,cursor)
    create_class_database(db,cursor)
    create_class_students_database(db,cursor)
```

10.14 email.py

```
#imports the required modules
import smtplib

def send_email(to,msg,teacher):
    #defines what server to connet to
    smtpserver = 'smtp'
    #Defines the username to sign in with
    smtpuser = 'admin@somedomain.com'
    #Defines the password to use
    smtppass = 'somepassword'
    #connects to the server
    session = smtplib.SMTP(smtpserver)
    #Logs into the server
    session.login(smtpuser, smtppass)
    #send email
    smtpresult = session.sendmail(teacher, [to], msg)
    return smtpresult
```

10.15 GUI_launch.py

```
#Import the PyQt libs and the gui sections
from PyQt4.QtCore import *
from PyQt4.QtGui import *
import sys
from gui_login import *

class LoginWindow(QMainWindow):
    def __init__(self):
        super().__init__()

        #Start with login layout
        self.setWindowTitle('Login')
        self.setCentralWidget(LoginWindow())

#Main program
if __name__ == '__main__':
    #create a new application
    application = QApplication(sys.argv)
    #Create main window
    window = LoginWindow()
    #show the window
    window.show()
    #raise window to the top of the window stack
    window.raise_()
    #monitor application for events
    application.exec_()

```

10.16 gui_login.py

```
from teacher_controller import *
import sys

class LoginWidget(QWidget):
    def __init__(self):
        super().__init__()
        #create a teacher controller
        self.TeacherController = teacher_controller()
        self.setWindowTitle('Login')
        #Create Components
        #Create Username Label and Line Edit
        self.UsernameLabel = QLabel('Username:')
        self.UsernameLineEdit = QLineEdit()
        #Create Password Label and Line Edit
        self.PasswordLabel = QLabel('Password:')
        self.PasswordLineEdit = QLineEdit()
        #Change Echo mode to password so the password is hidden
        self.PasswordLineEdit.EchoMode(2)
        #Create Login Button
        self.LoginButton = QPushButton("&Login")
        #Disable Login button (For re-enabling later)
        self.LoginButton.setEnabled(False)

        #Create the layout - Vertical Box
        self.LoginLayout = QVBoxLayout()
        #Add components to the layout
        self.LoginLayout.addWidget(self.UsernameLabel)
        self.LoginLayout.addWidget(self.UsernameLineEdit)
        self.LoginLayout.addWidget(self.PasswordLabel)
        self.LoginLayout.addWidget(self.PasswordLineEdit)
        self.LoginLayout.addWidget(self.LoginButton)

        #Create and set the Widget with the Layout
        self.LoginWidget = QWidget()
        self.setLayout(self.LoginLayout)

        #Connections
        #Enabling/disabling the login button
        self.UsernameLineEdit.textEdited.connect(self.ChangeLoginButton)
        self.PasswordLineEdit.textEdited.connect(self.ChangeLoginButton)
        #Log in on enter or button press
        self.UsernameLineEdit.returnPressed.connect(self.Login)
        self.PasswordLineEdit.returnPressed.connect(self.Login)
        self.LoginButton.clicked.connect(self.Login)

    def ChangeLoginButton(self):
        #Quick Length Check
        if len(self.UsernameLineEdit.text()) > 4:
            if len(self.PasswordLineEdit.text()) > 4:
                #If it's of length, enable the login button
                self.LoginButton.setEnabled(True)
            else:
                #If it's shorter than the length, disable the login button
                self.LoginButton.setEnabled(False)
        else:
            self.LoginButton.setEnabled(False)

    def Login(self):
        if self.LoginButton.isEnabled()) == "True":
            valid =
self.TeacherController.password_teacher_check(self.UsernameLineEdit.text(),self.PasswordLineEdit.text())
            if valid:
                print("Accepted")
            else:
                print("Denied")
```

10.17 gui_main_menu.py

```
#Import Core Libraries
from PyQt4.QtCore import *
from PyQt4.QtGui import *
import sys

#Import Functions
from GUI_ListStudents import *

class MainMenu(QMainWindow):
    def __init__(self):
        super().__init__()
        #Change Window title to "Main Menu"
        self.setWindowTitle('A-Level Computing Assignment Monitor')
        self.setMinimumWidth(700)
        self.setMaximumWidth(700)
        ##Setup a menu bar
        #Create MenuBar widget
        self.menuBar = QMenuBar()
        ##Add File Menu
        #Create Menu
        self.FileMenu = self.menuBar.addMenu("File")
        #Add Action
        self.File_BackupDatabase = self.FileMenu.addAction("Backup Database")
        self.File_Exit = self.FileMenu.addAction("Exit")
        ##Create Student Menu
        self.StudentMenu = self.menuBar.addMenu("Student Management")
        self.Student_ListStudents = self.StudentMenu.addAction("List Students")
        self.Student_ViewStudent = self.StudentMenu.addAction("View a Student")
        self.Student_AddStudent = self.StudentMenu.addAction("Add a Student")
        self.Student_EditStudent = self.StudentMenu.addAction("Edit a Student")
        self.Student_DeleteStudent = self.StudentMenu.addAction("Delete a Student")
        ##Create Class Menu
        self.ClassMenu = self.menuBar.addMenu("Class Management")
        self.Class_ViewClass = self.ClassMenu.addAction("View a Class")
        self.Class_AddClass = self.ClassMenu.addAction("Add a Class")
        self.Class_EditClass = self.ClassMenu.addAction("Edit a Class")
        self.Class_DeleteClass = self.ClassMenu.addAction("Delete a Class")
        ##Create Assignment Menu
        self.AssignmentMenu = self.menuBar.addMenu("Assignment Management")
        self.Assignment_ListAssignments = self.AssignmentMenu.addAction("List Assignments")
        self.Assignment_AddAssignment = self.AssignmentMenu.addAction("Add an Assignment")
        self.Assignment_EditAssignment = self.AssignmentMenu.addAction("Edit an Assignment")
        self.Assignment_DeleteAssignment = self.AssignmentMenu.addAction("Delete an Assignment")
        #Create Administration Menu
        self.AdministrationMenu = self.menuBar.addMenu("Administration")
        #Create a Teacher Menu under the Administration Menu
        self.Administration_TeacherMenu = self.AdministrationMenu.addMenu("Teachers")
        self.Administration_Teacher_ListTeachers = self.Administration_TeacherMenu.addAction("List the Teachers")
        self.Administration_Teacher_AddTeacher = self.Administration_TeacherMenu.addAction("Add a Teacher")
        self.Administration_Teacher_EditTeacher = self.Administration_TeacherMenu.addAction("Edit a Teacher")
        self.Administration_Teacher_DeleteTeacher = self.Administration_TeacherMenu.addAction("Delete a Teacher")
        self.Administration_Email = self.AdministrationMenu.addMenu("Email Settings")
        #Create Help Menu
        self.HelpMenu = self.menuBar.addMenu("Help")
        self.Help_Help = self.HelpMenu.addAction("Help!")
        #Sets standard Help shortcut
        self.Help_Help.setShortcut('F1')

        ##Main Space
        self.StudentsInDanger_Title_Label = QLabel("Students in danger: ")
        self.StudentsInDanger_List_Label = QLabel(self.StudentsInDanger())
        self.ViewClassProgress_Button = QPushButton("View a Class's Progress")
        self.ViewStudentsProgress_Button = QPushButton("View a Student's Progress")
        self.ShowStatisticsForClass_Button = QPushButton("Show Statistics for Class")
        self.ShowStatisticsForAllClasses_Button = QPushButton("Show Statistics for All Classes")

        #Create and add to layout
        self.MainMenu_Layout = QGridLayout()
        self.MainMenu_Layout.addWidget(self.StudentsInDanger_Title_Label,0,0)
        self.MainMenu_Layout.addWidget(self.StudentsInDanger_List_Label,0,1)
        self.MainMenu_Layout.addWidget(self.ViewClassProgress_Button,1,0)
        self.MainMenu_Layout.addWidget(self.ViewStudentsProgress_Button,1,1)
        self.MainMenu_Layout.addWidget(self.ShowStatisticsForClass_Button,2,0)
        self.MainMenu_Layout.addWidget(self.ShowStatisticsForAllClasses_Button,2,1)
        self.MainMenu_Widget = QWidget()
        self.MainMenu_Widget.setLayout(self.MainMenu_Layout)

        #Set Layout
        self.setCentralWidget(self.MainMenu_Widget)

        #Set the menu widget
        self.setMenuWidget(self.menuBar)

        ###Connections
        ##Menu Bar
        #File Menu
        self.File_BackupDatabase.triggered.connect(self.BackupDatabase)
        self.File_Exit.triggered.connect(self.Close)
        #Student Menu
        self.Student_ListStudents.triggered.connect(self.ListStudents)
        self.Student_ViewStudent.triggered.connect(self.ViewAStudent)
        self.Student_AddStudent.triggered.connect(self.AddStudent)
        self.Student_EditStudent.triggered.connect(self.EditStudent)
        self.Student_DeleteStudent.triggered.connect(self.DeleteStudent)



    def StudentsInDanger(self):
        #This will process which students are in danger for the GUI
        return "List of Students"

    def BackupDatabase(self):
        print("Backup dat shiz!")

    def Close(self):
        window.close()
        sys.exit()

    def ListStudents(self):
        self.setCentralWidget(ListStudents())

    def ViewAStudent(self):
        pass

    def AddStudent(self):
        pass

    def EditStudent(self):
        pass

    def DeleteStudent(self):
        pass


if __name__ == "__main__":
    application = QApplication(sys.argv)
    window = MainMenu()
    window.show()
    window.raise_()
    application.exec_()
```

10.18 student_controller.py

```
#Imports the data from the controller class
#for communication with the DB
from controller_class import *

#Creates a new class, using the db controller as it's parent
class student_controller(database_controller):
    """Controller for the database connections with a student"""
    #This sets up any values that I need for my class
    def __init__(self):
        #This inherits any of the values from the parent class
        super().__init__()

    def add_student(self, StudentLastName, StudentFirstName, StudentDOB, StudentEmail,
                    StudentScribe, Student25Extra, Student50Extra,StudentWordProcessor, StudentGCSEResults,
                    StudentLastEmailed, StudentNotes):
        #This function allows the user to add a student to the database.

        #This SQL statement contains the details I need adding to the db
        #It uses the format ability to easily insert all the values into
        #the statement.
        sql = """insert into Student(StudentLastName, StudentFirstName,
                StudentDOB, StudentEmail,StudentScribe,Student25Extra,
                Student50Extra, StudentWordProcessor, StudentGCSEResults, StudentLastEmailed, StudentNotes)
                values
                ('{0}','{1}','{2}','{3}','{4}','{5}','{6}','{7}','{8}','{9}','{10}')""".format(
                    StudentLastName, StudentFirstName, StudentDOB, StudentEmail,
                    StudentScribe, Student25Extra, Student50Extra, StudentWordProcessor,
                    StudentGCSEResults, StudentLastEmailed, StudentNotes)
        #Perform the operation
        self._query(sql)

    def edit_student(self, StudentID, StudentLastName=None, StudentFirstName=None,
                     StudentDOB=None, StudentEmail=None, StudentScribe=None, Student25Extra=None,
                     Student50Extra=None, StudentGCSEResults=None, StudentLastEmailed=None, StudentNotes=None):
        #This function allows me to edit all of a student's values in one go
        #It uses named parameters to allow me to have them optional

        #Starts the list of changes needed
        changes = []

        #Checks each value to see if Studentthey're used
        #if Studentused, it will append each change to the list as a list
        #Ie, a list of lists.
        if StudentLastName != None or "":
            changes.append(("StudentLastName",StudentLastName))
        if StudentFirstName != None or "":
            changes.append(("StudentFirstName",StudentFirstName))
        if StudentDOB != None or "":
            changes.append(("StudentDOB",StudentDOB))
        if StudentEmail != None or "":
            changes.append(("StudentEmail",StudentEmail))
        if StudentScribe != None or "":
            changes.append(("StudentScribe",StudentScribe))
        if Student25Extra != None or "":
            changes.append(("Student25Extra",Student25Extra))
        if Student50Extra != None or "":
            changes.append(("Student50Extra",Student50Extra))
        if StudentGCSEResults != None or "":
            changes.append(("StudentGCSEResults",StudentGCSEResults))
        if StudentLastEmailed != None or "":
            changes.append(("StudentLastEmailed",StudentLastEmailed))
        if StudentNotes != None:
            changes.append(("StudentNotes",Notes))
        #This is the start of the sql statement that will be added to
        sql = "update student set "
        #Iteration of each list within the changes list
        for update in changes:
            #This adds each update to the sql statement
            sql += "{0}='{1}', ".format(update[0],update[1])

        #Remove the last 2 characters ', '
        sql = sql[:-2]
        #Adds which ID to edit
        sql+= " where StudentID ='{0}'".format(StudentID)
        #Performs the query to the database
        self._query(sql)

    def delete_student(self,StudentID):
        #This function deletes a row from the table
        sql = "DELETE from student WHERE StudentID = {0}".format(StudentID)
        self._query(sql)

    def find_student(self, StudentID=None, StudentLastName=None, StudentFirstName=None,
                     StudentDOB=None, StudentEmail=None, StudentScribe=None, Student25Extra=None,
                     Student50Extra=None, StudentGCSEResults=None, StudentLastEmailed=None):
        #This function is designed to find all the rows that match the following data.
        #It works in the same way as the update function.

        #Creates a new list
        parameters = []

        #Detects if Student the named parameters are used
        #if Student so, it will append them to the list
        if StudentID != None or "":
            parameters.append(("StudentID",StudentID))
        if StudentLastName != None or "":
            parameters.append(("StudentLastName",StudentLastName))
        if StudentFirstName != None or "":
            parameters.append(("StudentFirstName",StudentFirstName))
        if StudentDOB != None or "":
            parameters.append(("StudentDOB",StudentDOB))
        if StudentEmail != None or "":
            parameters.append(("StudentEmail",StudentEmail))
        if StudentScribe != None or "":
            parameters.append(("StudentScribe",StudentScribe))
        if Student25Extra != None or "":
            parameters.append(("Student25Extra",Student25Extra))
        if Student50Extra != None or "":
            parameters.append(("Student50Extra",Student50Extra))
        if StudentGCSEResults != None or "":
            parameters.append(("StudentGCSEResults",StudentGCSEResults))
        if StudentLastEmailed != None or "":
            parameters.append(("StudentLastEmailed",StudentLastEmailed))

        #This begins the select command for the list
        #It's choosing only certain columns for the list, because of security.
        sql = """select *
                FROM student
                where """

        #This adds all the parameters to the sql statement
        for parameter in parameters:
            sql = sql + "{0}='{1}' and ".format(parameter[0],parameter[1])

        #This removes the final " and " from the sql statement
        sql = sql[:-5]
        return self._select_query(sql)

    def student_headings(self):
        #This function returns all the data about the table.
        sql = "PRAGMA table_info(student)"
        return self._select_query(sql)
```

10.19 teacher_controller.py

```
#Imports the data from the controller class
#for communication with the DB
from controller_class import *

#Creates a new class, using the db controller as it's parent
class teacher_controller(database_controller):
    """Controller for the database connections with a teacher"""
    #This sets up any values that I need for my class
    def __init__(self):
        #This inherits any of the values from the parent class
        super().__init__()

    def add_teacher(self, TeacherUserName, TeacherPassword, TeacherAdmin,
                    TeacherAdditionalPassword, TeacherLastName, TeacherFirstName,
                    TeacherEmail, TeacherQuestion, TeacherAnswer, TeacherLastEmailed):
        import hashlib
        #This function allows the user to add a teacher to the database.
        TeacherPassword = hashlib.md5(TeacherPassword.encode('utf-8')).hexdigest()
        TeacherAdditionalPassword = hashlib.md5(TeacherAdditionalPassword.encode('utf-8')).hexdigest()
        #This SQL statement contains the details I need adding to the db
        #It uses the format ability to easily insert all the values into
        #the statement.
        sql = """insert into Teacher(TeacherUserName, TeacherPassword,
                TeacherAdmin, TeacherAdditionalPassword, TeacherLastName,
                TeacherFirstName, TeacherEmail, TeacherQuestion,
                TeacherAnswer, TeacherLastEmailed)
                values
                ('{0}','{1}','{2}','{3}','{4}','{5}','{6}','{7}','{8}','{9}')""".format(TeacherUserName,
                TeacherPassword, TeacherAdmin, TeacherAdditionalPassword, TeacherLastName, TeacherFirstName,
                TeacherEmail, TeacherQuestion, TeacherAnswer, TeacherLastEmailed)
        #Perform the operation
        self._query(sql)

    def edit_teacher(self, TeacherID, TeacherUserName=None, TeacherPassword=None, TeacherAdmin=None,
                    TeacherAdditionalPassword=None, TeacherLastName=None, TeacherFirstName=None,
                    TeacherEmail=None, TeacherQuestion=None, TeacherAnswer=None):
        #This function allows me to edit all of a teachers values in one go
        #It uses named parameters to allow me to have them optional

        #Starts the list of changes needed
        changes = []

        #Checks each value to see if they're used
        #If used, it will append each change to the list as a list
        #Ie, a list of lists.
        if TeacherUserName != None:
            changes.append(("TeacherUserName",TeacherUserName))
        if TeacherPassword != None:
            TeacherPassword = hashlib.md5(TeacherPassword.encode('utf-8')).hexdigest()
            changes.append(("TeacherPassword",TeacherPassword))
        if TeacherAdmin != None:
            changes.append(("TeacherAdmin",TeacherAdmin))
        if TeacherAdditionalPassword != None:
            TeacherAdditionalPassword = hashlib.md5(TeacherAdditionalPassword.encode('utf-8')).hexdigest()
            changes.append(("TeacherAdditionalPassword",TeacherAdditionalPassword))
        if TeacherLastName != None:
            changes.append(("TeacherLastName",TeacherLastName))
        if TeacherFirstName != None:
            changes.append(("TeacherFirstName",TeacherFirstName))
        if TeacherEmail != None:
            changes.append(("TeacherEmail",TeacherEmail))
        if TeacherQuestion != None:
            changes.append(("TeacherQuestion",TeacherQuestion))
        if TeacherAnswer != None:
            changes.append(("TeacherAnswer",TeacherAnswer))

        #This is the start of the sql statement that will be added to
        sql = "update teacher set "
        #Iteration of each list within the changes list
        for update in changes:
            #This adds each update to the sql statement
            sql += "{0}='{1}', ".format(update[0],update[1])

        #Remove the last 2 characters ', '
        sql = sql[:-2]
        #Adds which ID to edit
        sql+= " where TeacherID='{0}'".format(TeacherID)
        #Performs the query to the database
        self._query(sql)
```

```

    def delete_teacher(self,TeacherID):
        #This function deletes a row from the table
        sql = "DELETE from teacher WHERE TeacherID = {0}".format(TeacherID)
        self._query(sql)

    def find_teacher(self, TeacherID=None, TeacherUserName=None, TeacherAdmin=None, TeacherLastName=None,
                     TeacherFirstName=None, TeacherEmail=None):
        #This function is designed to find all the rows that match the following data.
        #It works in the same way as the update function.

        #Creates a new list
        parameters = []

        #Detects if the named parameters are used
        #If so, it will append them to the list
        if TeacherID != None:
            parameters.append(("TeacherID",TeacherID))
        if TeacherUserName != None:
            parameters.append(("TeacherUserName",TeacherUserName))
        if TeacherAdmin != None:
            parameters.append(("TeacherAdmin",TeacherAdmin))
        if TeacherLastName != None:
            parameters.append(("TeacherLastName",TeacherLastName))
        if TeacherFirstName != None:
            parameters.append(("TeacherFirstName",TeacherFirstName))
        if TeacherEmail != None:
            parameters.append(("TeacherEmail",TeacherEmail))

        #This begins the select command for the list
        #It's choosing only certain columns for the list, because of security.
        sql = """select TeacherID,TeacherUserName,TeacherAdmin,TeacherLastName,TeacherFirstName,TeacherEmail
                FROM teacher
                where """

        #This adds all the parameters to the sql statement
        for parameter in parameters:
            sql = sql + "{0}='{1}' and".format(parameter[0],parameter[1])

        #This removes the final " and" from the sql statement
        sql = sql[:-4]
        return self._select_query(sql)

    def password_teacher_check(self,username,password):
        #Imports the hashlib
        import hashlib
        #Creates the SQL statement to get the password
        sql = """select TeacherPassword
        from Teacher
        where TeacherUserName = '{0}'""".format(username)
        #Puts the password into a list
        list = self._select_query(sql)
        #Check to void failing on no user
        if len(list) > 0:
            #Checks the entered password with the stored password
            if hashlib.md5(password.encode('utf-8')).hexdigest() == list[0][0]:
                #Accepts the password if matches
                return True
            else:
                #Rejects the password if not
                return False
        else:
            #Rejects the password if no user exists
            return False

    def teacher_headings(self):
        #This function returns all the data about the table.
        sql = "PRAGMA table_info(teacher)"
        return self._select_query(sql)
```