Development Log (I need to add screenshots for the first little bit)

19th Oct 2019 – Started work on a library program. The program consists of a 2D array with the name of the book and whether it is available to lend inside of each individual array. Functions written so far allow for one to take out and return a book to the library, changing a boolean inside of the array. I plan to add a function to add a new book to the library.

21st Oct 2019 – New function added which allows one to add a new book to the library. This is accomplished by appending a newly created array in to the 2D array. My next plan is to add date/time validation to the program, changing each array to have a “next in by” date, i.e. the date at which the book will be returned. This date is compared to the current date to see whether the book is available, and will ask how long the user needs it for and therefore when the book will be expected to be back in the library.

4th November 2019 – I have managed to retrieve the current date/time from python and been able to compare it to another using the substring function to compare dates. Next, I plan to alter the 2D array to accommodate dates instead of bools.

17th November 2019 – The 2D array now accommodates dates instead of booleans. My next plan is to use an SQL flat file database instead of a 2D array. To do this I will use SQLite and directly transfer my layout for a 2D array on to the flat file. In order to adapt my program, I will need to write functions that can read and write from an SQL database which will likely take much longer to do than previous iterations of the program.

13th February 2020 – I have not worked on coursework for a long time since last year, as I have been busy with exams and preparation for them. I have a basic structure for my program’s database mocked out in the SQL database editor I have made, as follows as shown in figure 1.

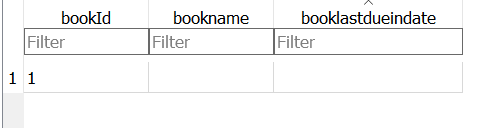


Figure 1

1st March 2020 – Lockdown seems imminent due to coronavirus, which presents a large risk to the progression of my coursework. I have chosen to reform my database to one which suits a database of labourers or handymen – a handyman can sign up to the service, listing their profession for clients to easily find and access and hire them. As such, each handyman will require a separate listing in the database, as will each customer. I have created a new database which is now fully relational and in 3rd normal form to make it as efficient as possible, presented below.

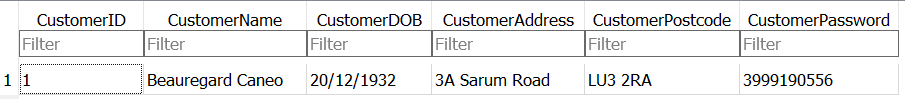


Figure 2

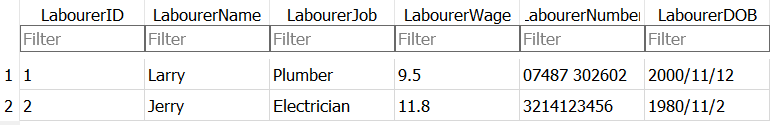


Figure 3

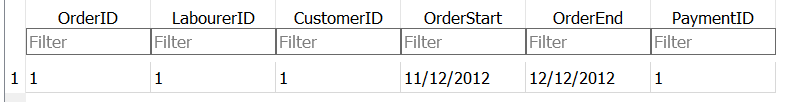


Figure 4

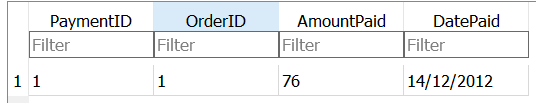


Figure 5

Customer Passwords are not stored as plaintext inside of the database; the password shown is a hashed version of the true password. I have yet to figure out what I will use for the hashing algorithm, so for now I’ve randomly generated a number to put in as a placeholder. I will likely need to use some ascii character code conversions to make it work.

Customer and Labourer tables have a many to many relationship, which I have bridged using the orders table as shown. In order to track payments I have implemented a payment table. While at first it may appear to not meet requirements for 3rd normal form, it is important to realise that not all of an order’s required payment will necessarily be paid all at once, and hence I have had to make it possible for multiple payments to assign to one order placed.

I have also denoted any fields where a primary or foreign key has been used with a 1.

2nd August 2020 – I have not done much work on my coursework in a long time. As such, I have realised that it is important I continue promptly to allow myself ample time to prepare for A Level exams when they come around next year.

I have made functions which allow me to read from my SQL database, which all follow the same template.

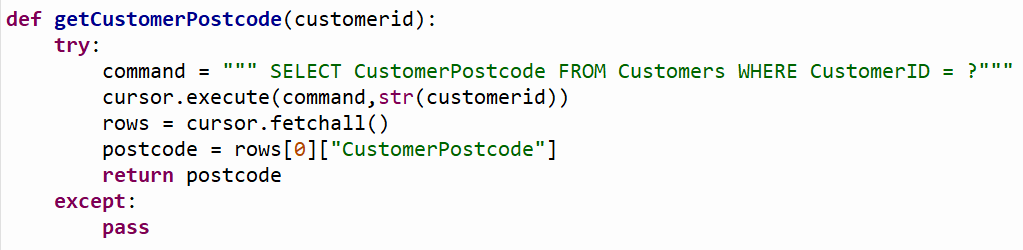


Figure 6

Initially, I had an issue as I replaced the question mark with an exclamation mark, which treated it as a wildcard and attempted to find anything like what I entered, which was blank. As a result, it threw an exception. I replaced it with a question mark, which allowed me to pass the customer id in to the statement easily.

All of the functions written follow this structure. However, in order for it to work, it is notable that the “cursor.fetchall()” function returns an array filled with dictionaries. This is done with the following function:

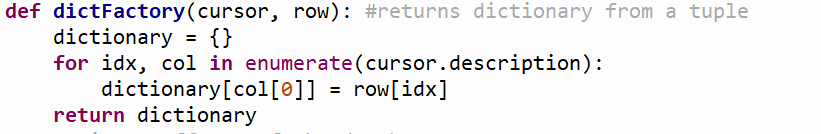


Figure 7

This returns a single dictionary. However, when the following function is run:

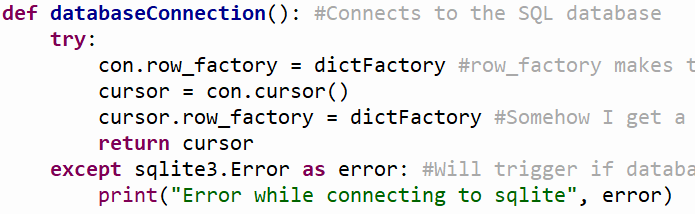


Figure 8

This is repeated for all instances inside of the database, giving the array of dictionaries as shown before.

The reason I have done this is to make it far easier to edit the contents of the dictionary if needed, as a tuple cannot be edited, and sometimes cannot be read either. This method makes it far easier to create functions that work with the contents of the database.

18th September 2020 – I have made an error with my program, as I neglected to realise I have called on the cursor variable which has only been defined in the local scope of the databaseConnection function, but has not been passed in to the new functions. As a result, the program will not work as the cursor will throw an object not defined error. I have amended the function as shown below:

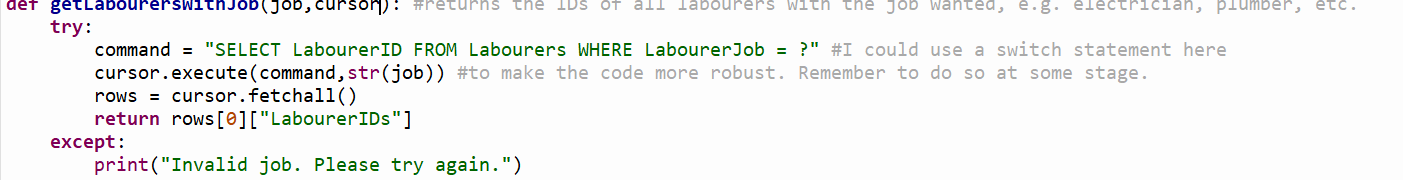


Figure 9

Try and except occasionally makes it rather difficult to debug code without removing them as well. During testing, I would simply run the program without calling the function and assume it was working. I realised this would not work as the program would assume the try and except loop worked with another piece of code to tell whether it was supposed to run something when that was not the case.

I found using the debugging tools built in to my IDE also provided a lot of help, especially with testing and finding issues with my code. Stepping through the code made it very easy without needing to go through the troubles of drawing out a trace table for every function, as well as see what my variables actually look like as I go through.

1st October – Began work on functions that will add to the database instead of just read from it. This provides a unique challenge, as the code is more complex due to needing to find the next needed ID for the table, as well as validate each input to prevent SQL injection.

Currently, my plan to prevent injection is to insert the values using the VALUES SQL function, which will forego any processing of the input itself, in a manner shown below:

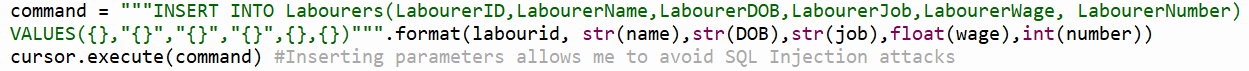
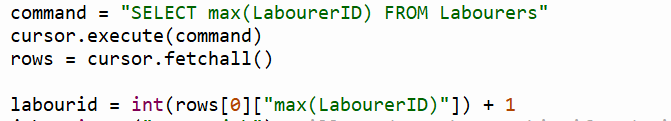


Figure 10

It is important to note that not having quotation marks around curly brackets which will store a string will cause the code to break.

20th October – More work done on a general function which adds to the database. I have done the following to find the max ID and find which ID needs to be found next:



The dictionary actually returns the key by the command I ran to get it, so for the retrieval function I have to call like so in order to get the ID.

4th November – Finished general writing function. Shown below.

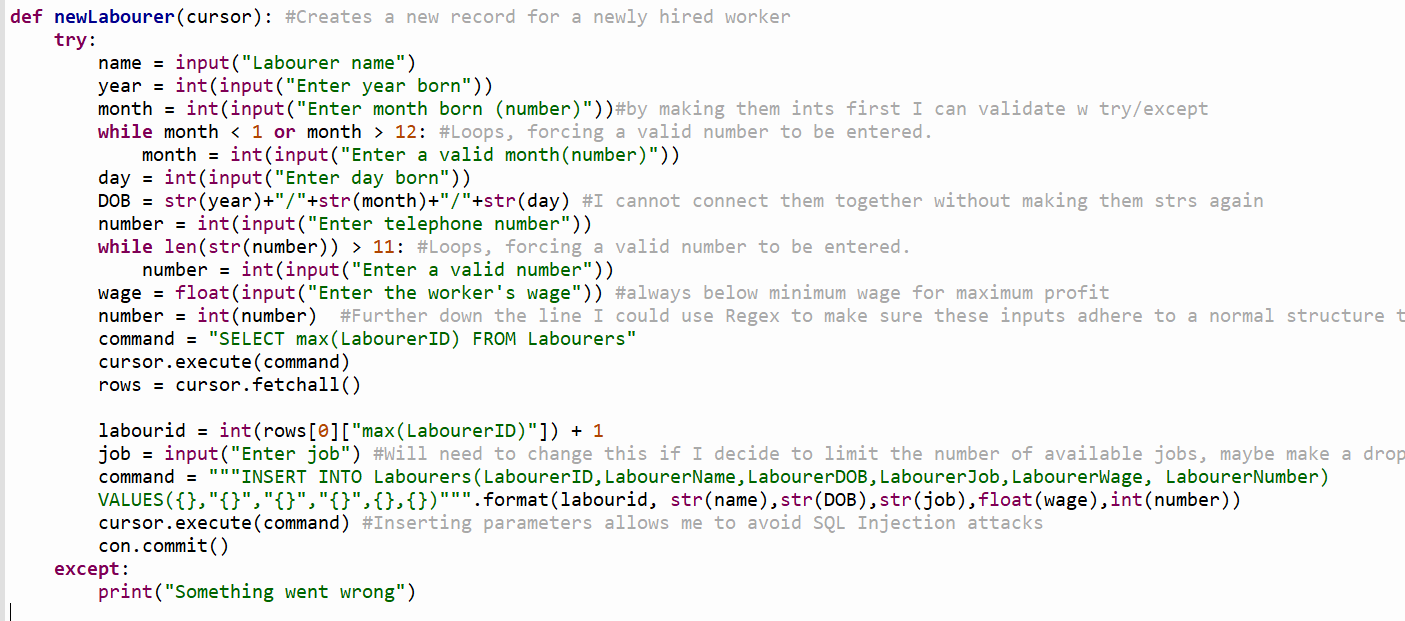


Figure 11

I repeated this structure for each of my tables to give a total of 4 writing functions. When I create my GUI I will likely need to make more functions which are made up of these functions, as well as parameterise these functions to suit the GUI format. This will make my time spent coding the GUI much shorter since I have much less which actually needs to be written up.

17th November – Started working on validation functions for dates as well as postcodes using REGEX. Understanding the syntax of regex is difficult but luckily the UK government provides a regular expression for programmers to use expressively for this purpose. However, it is important to know that the regular expression provided accommodates for any possible postcode, and the postcode does not necessarily need to exist for it to be used. As a result, I have considered using google waypoints API to counter this issue. However, I will look more in detail into this solution later as my current priority is finishing a decent GUI in good time.

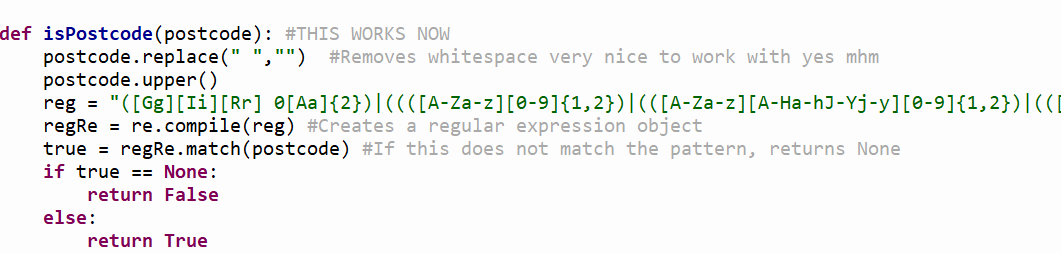


Figure 12

The full expression is as follows:

([Gg][Ii][Rr] 0[Aa]{2})|((([A-Za-z][0-9]{1,2})|(([A-Za-z][A-Ha-hJ-Yj-y][0-9]{1,2})|(([A-Za-z][0-9][A-Za-z])|([A-Za-z][A-Ha-hJ-Yj-y][0-9][A-Za-z]?))))\s?[0-9][A-Za-z]{2})

20th December – Finished a checkDate function which checks if the date entered is valid. Shown below.

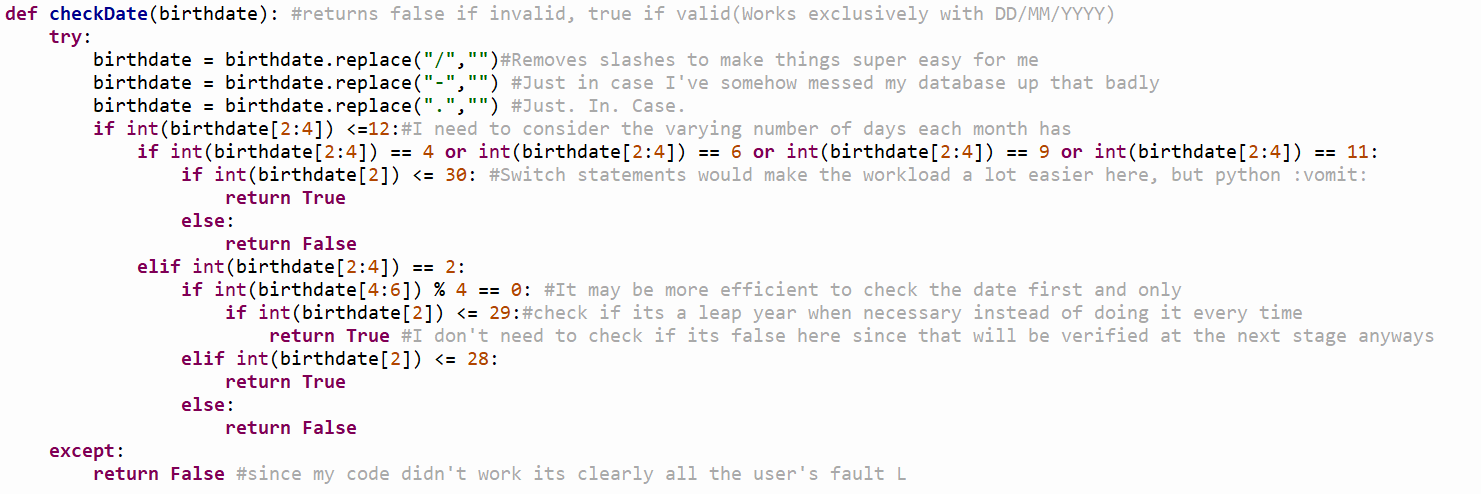


Figure 13

My database stores dates in the form of “DD/MM/YYYY” so ideally the date would not have hyphens or dots between the date, month and year. However, if I mess up along the way or if a user forcibly enters data directly in to the database, this will mean the date will still be considered valid.

I have made considerable use of the substring function provided by python, as items in the string can be called by index and length. This allows me to check specifically the date, month and year.

When considering the issue, it becomes apparent that different months have different numbers of days, and will hence need the month to be checked first. I have a set number of possible outcomes which are conveniently labelled from 1 to 12, which would be perfect for a switch statement. However, this isn’t possible in python, so I have been forced to simply use an if statement to catch all the possibilities.

27th December – I have created a password hashing function which is currently very primitive.

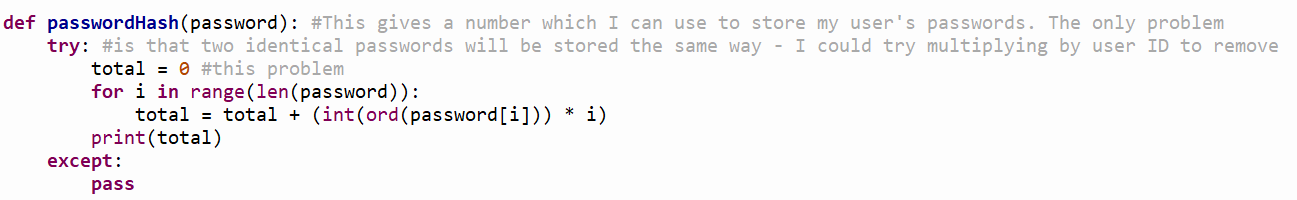


Figure 14

Print(total) would normally be a return statement but I have kept it that way for testing purposes.

The biggest problem with this is that 2 identical passwords will be stored in exactly the same way, meaning if someone gained access to the database, it would be possible for them to find out a user’s password with enough time which is much less than ideal. The solution I proposed in my code comments, multiplying by user id, would mean that it would be harder for the hacker to access passwords as he would need to figure out that he needs to divide each password by the user’s user ID first as well.

7th January – Began working on my GUI. Now my SQL is mostly finished, I can start drawing out what I want to make for my GUI and begin working on it. For now, I have decided to start out on working on showing and hiding a basic GUI, and being able to output a string when a button is clicked.

I have decided to use PyQt5 for this, as the Qt framework is well established, meaning it will be easier for me to get help with any large issues I encounter while using it.

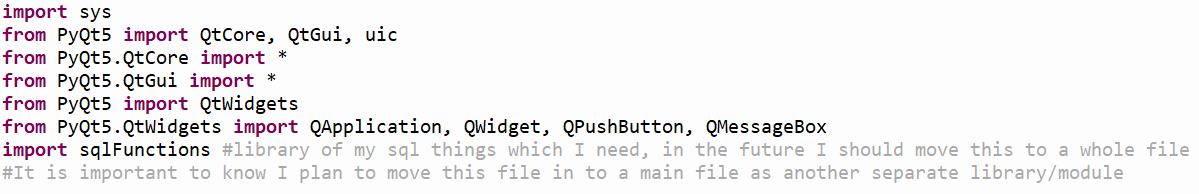


Figure 15

This code is used to import all the parts of PyQt5 which I will be using in my program.

Next, I need to initialise my window objects.

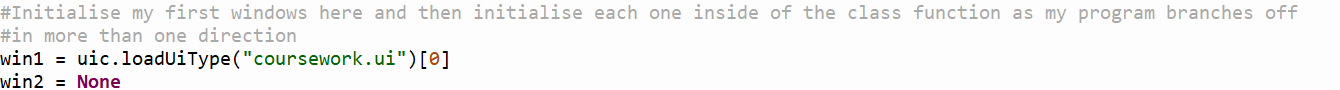


Figure 16

By initialising all the windows at the start of my program, I can simply show and hide each window as needed.

14th January – Made a function to initialise my first screens. Code is shown below.

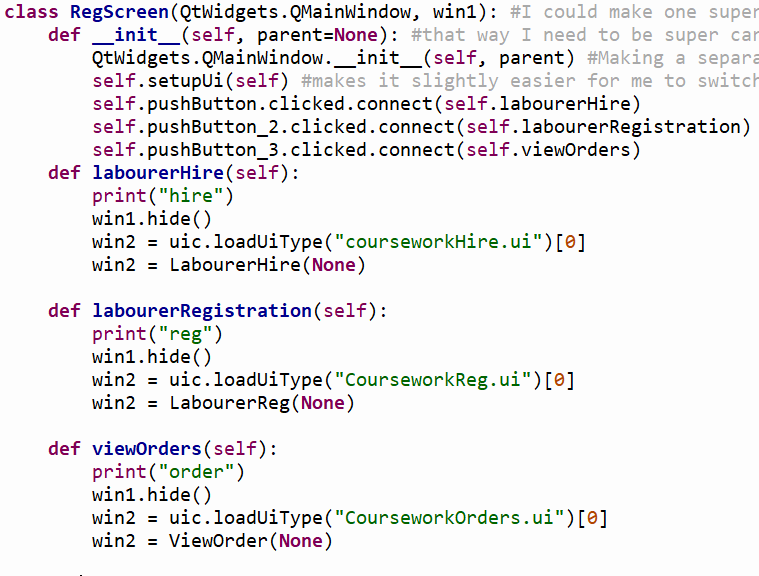


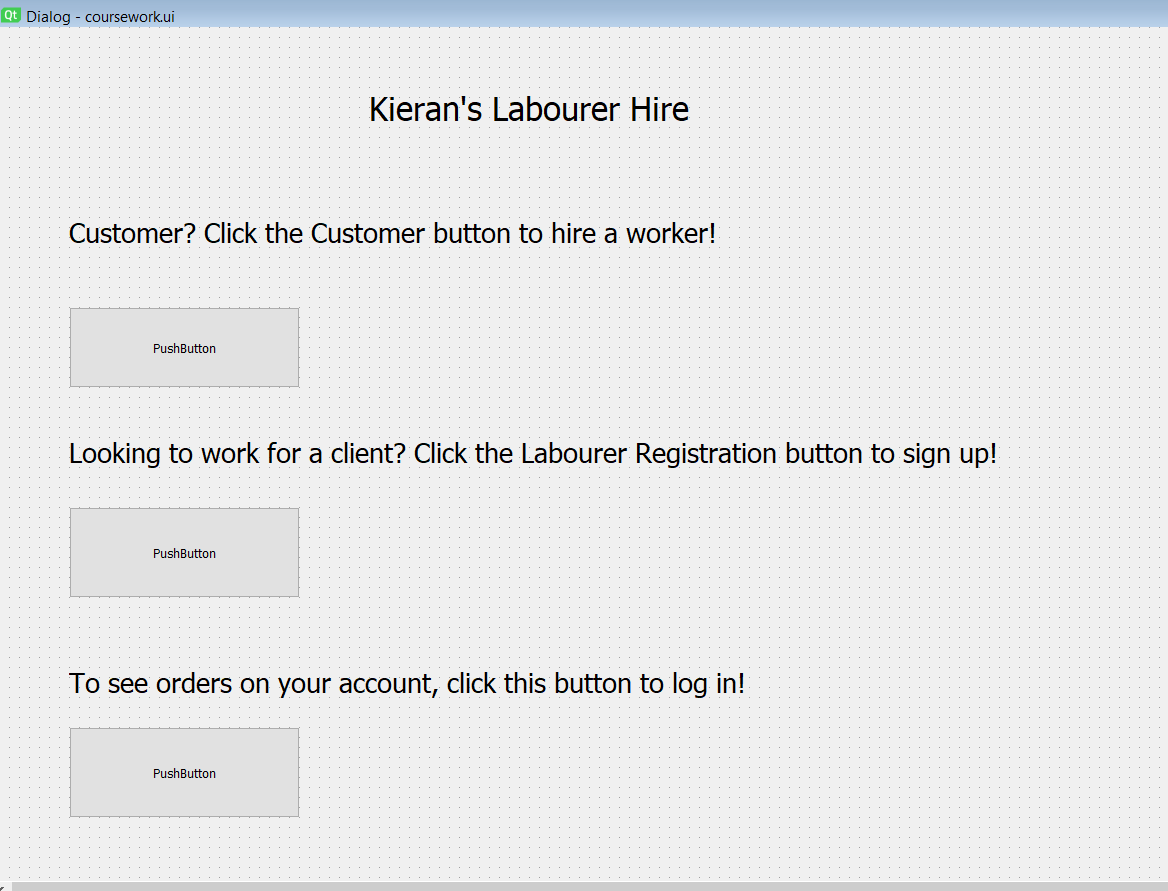
Figure 17

RegScreen is a class I will initialise at the start of the program. I will create classes for each respective window object I create, which will be able to perform functions accordingly. In order to go back and forth between different forms, I will implement the use of a stack which will store the windows being used in order. Each time the back button is pressed an item is popped off the stack, and each time a form is changed to the old form is added to the stack.

Each method I have specified inside of this class I plan to initialise in to another class.

As such, I will create a class for viewOrder, LabourerReg and LabourerHire.

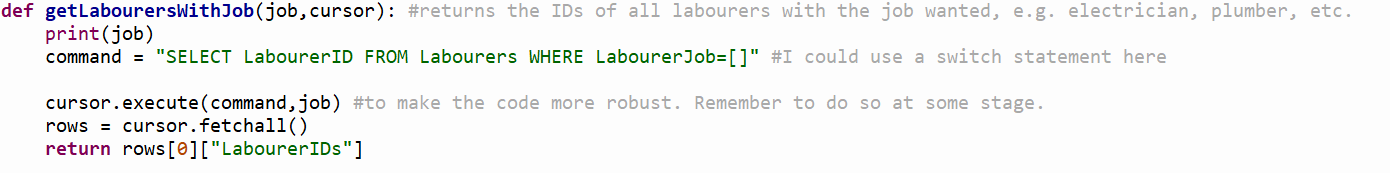
2nd February – Created my first GUIs using PyQt’s designer. The designer allows me to create UI files which allow me to specify object names from within the file, and I can perform in built functions with the object’s attributes using the imported libraries provided by the PyQt5 library.



Figure

Clicking each separate button runs each method, by running an event whenever the buttons are clicked. When this event occurs, I cause a method to run which launches the next window I require.

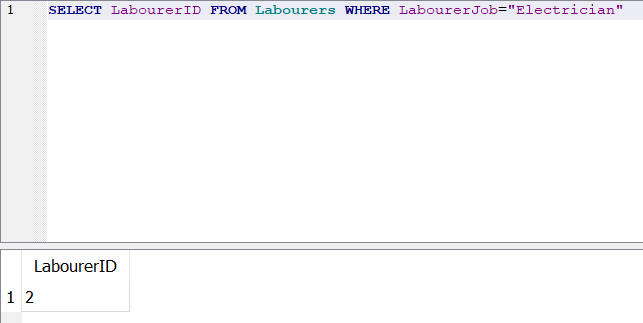
10th February – I have had some issues running my SQL functions. I believe I will need to test and reconsider the template I used previously, as it has failed in the case of the following function:



Figure

Currently, the job parameter being passed in is a string. However, the program throws an error, saying there is no column with no name. I do not know why, as no empty column names have been specified in the program.

I have as such tested the SQL code from inside the database browser. The code works perfectly fine without square brackets, but throws the same issue with them, so I believe the issue lies with the usage of the square brackets as a place holder value in side of the command itself.

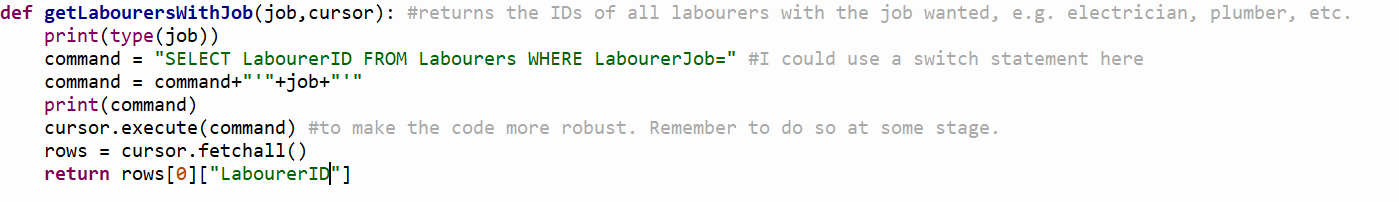


Figure



Figure

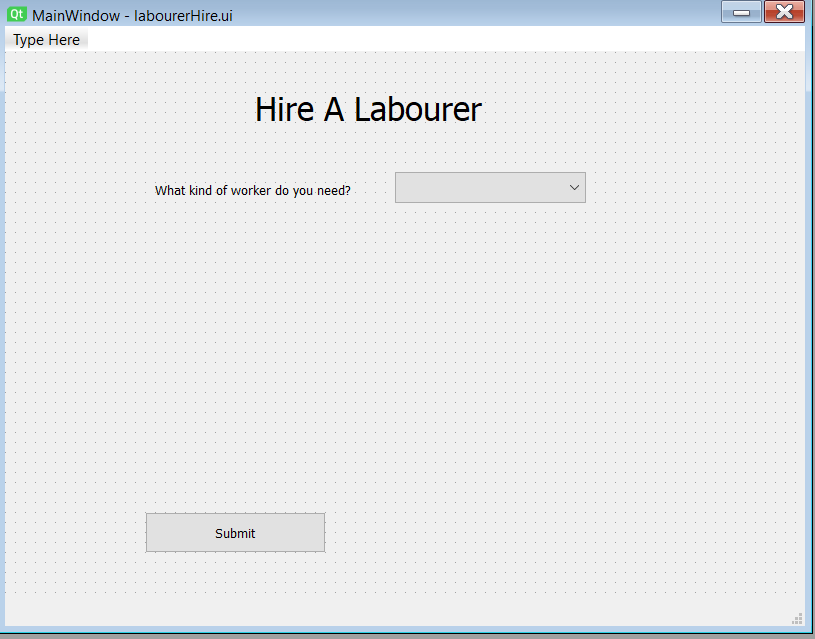
From this, I have realised the only reasons why this seemed to work perfectly fine previously was because my IDs are only single digits.



Figure

This solution I have used takes advantage of the fact that I know users cannot enter their own code in to the program, meaning I no longer need to consider the possibility of an SQL attack. Instead, this allows me to simply append the results from the combo box to the end of my statement, and avoids me having to make a separate SQL function for every possible combo box option.

I have made the combo box inside of the designer as such:



Figure

Right now, I do not have much I need to consider as far as choosing my workers goes. Workers can potentially just be suggested at random to the customer, but there is a lot I could do with this in the future, such as a graph colouring algorithm, usage of Google’s waypoints API to choose workers who are close by, and perhaps even looking at a worker’s previous hiring history to evenly distribute jobs.

The bigger problem at hand currently, however, is modifying my SQL functions to allow them to accommodate for cases where more than one character is passed in to the function. I will not tackle the problem for now – I think it’d be more efficient to handle the bug when I write the code for the GUI options which utilise those functions.

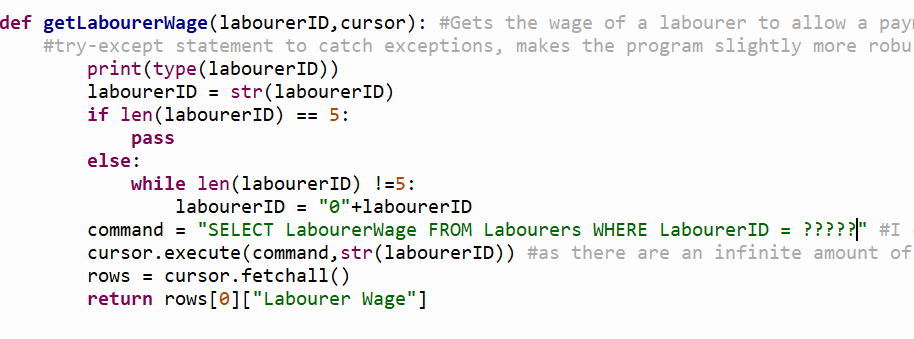


Figure 24

I attempted to implement a solution I suggested earlier, where a certain number of question marks would define how large my database could be. However, it seems that this isn’t possible, due to the way SQL handles the placeholder ? value. It treats each character as a separate entity when there are multiple in a row, rather than placing each character together to form a string.

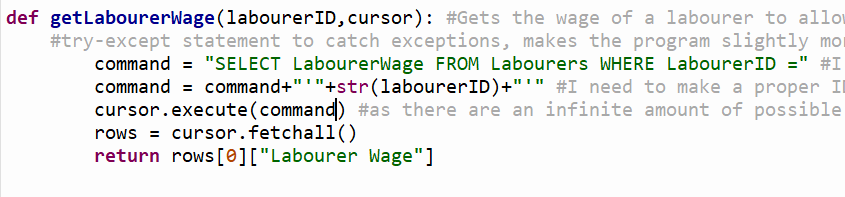
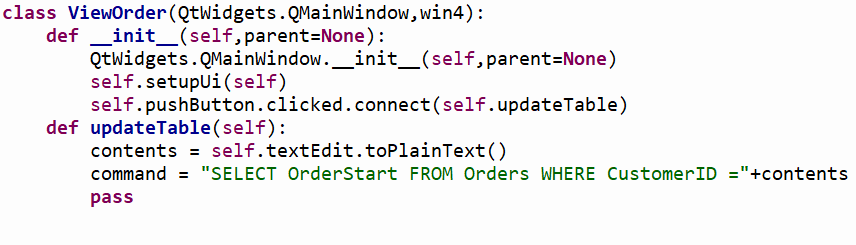


Figure 25

I have instead opted to use the same method as with the previous function. However, in order to keep using this, I will need to write a separate verification of ID function, so I know the user isn’t being naughty and trying to inject some code. I can probably do this with a simple attempted convert to INT attempt, and if it fails return false. I could also find the max ID and check if the INT falls within the range required from it.

16th February – Started working on a basis for my ViewOrders class.



Figure

Initially, the command did not function as I needed it to. However, after applying a quick fix, this was remedied.

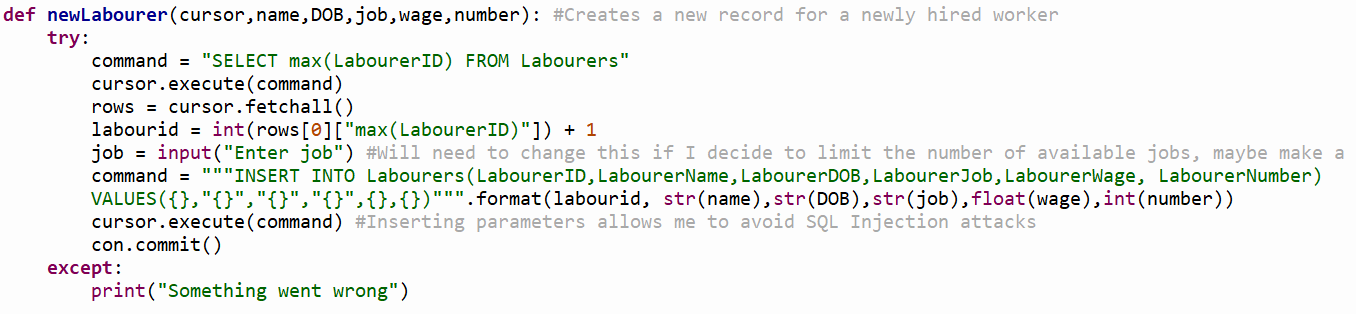


Figure

This meant everything was passed in to the SQL as one string, allowing it to run, whereas before the string contents was not placed as a string but instead as plaintext.

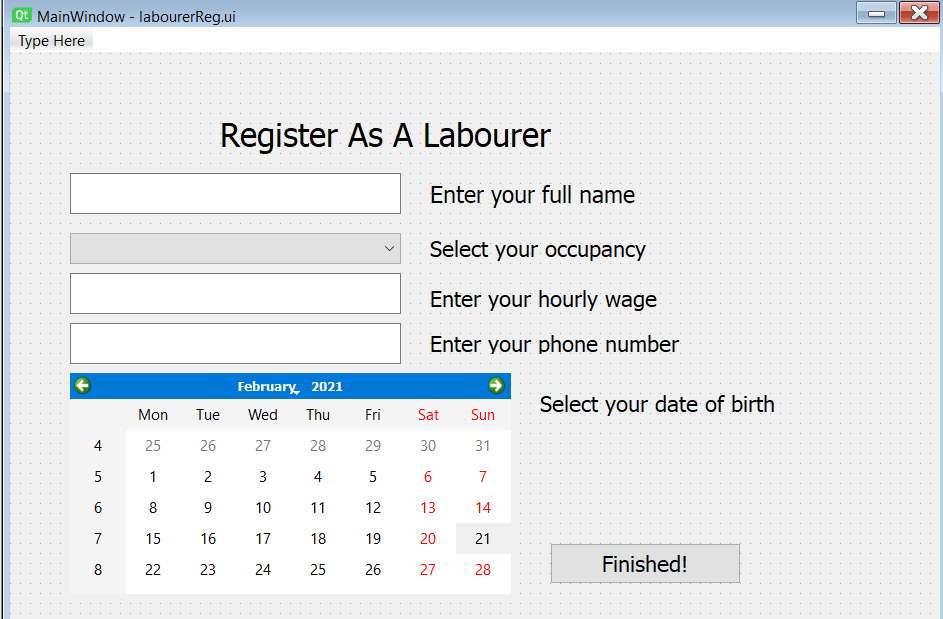
I implemented my table widget here. I plan to make this table widget much more developed in the future, however this class is certainly less essential than the ability to add and take orders out of the database. As such, I think I should focus on those aspects more in the future.

21st February – Did further work on my LabourerReg class. The first thing I need to do is to parameterize my newLabourer function inside of my sqlFunctions library. It now looks like this:



Figure

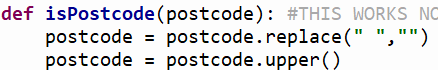
There was a lot stripped out from the function, but this can be easily fixed by simply using the checkDate function within my sqlFunctions library. Inside of my class’s method I will need to make sure this function returns a True Boolean value, as well as validate the rest of the input. In a sense, all I’m doing is moving the code from my library of SQL functions to my GUI library.



Figure

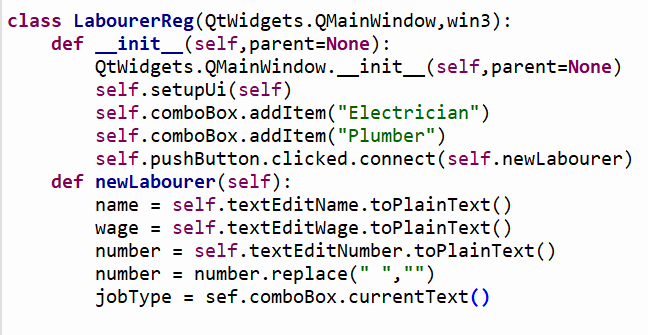
This is how my GUI now looks for the LabourerReg class.

While attempting to validate the inputted number from the GUI, I realised that my checkPostcode function didn’t work as intended.



Figure

Previously, this function would run the replace and upper functions. However, I didn’t save these changes, meaning the changes would mean nothing and a valid postcode wouldn’t be returned as valid as the postcode would be in varying forms to the form I intended to standardize it in to.



This is what I’ve written so far in the LabourerReg class. I need to write something to extract the date selected from the calendar widget and I will be good to run it in to the function from my library.