***Problem Definition***

At home, I’ve found for a while that, having siblings who might walk into my room when I’m not home, I would like to feel more secure with my belongings and I’d want my room to be sibling-proof.

Originally, my first idea was to have a conventional lock-and-key for my room; however it would be a hassle to have to remember to lock the door behind me every time I leave and it would also create issues if, for example, I left my wallet in my room and needed my parents to pick it up and drop it off to me, as they wouldn’t be able to go into my room to get it. Also, with me losing things regularly, I’m not usually great at keeping keys for a long period of time without losing them at least once, which would mean I’d more often than not lock myself out of my own room.

After thinking this over, I also had a conversation with my dad about how he’s also always losing his key for his office and wished that he could have an electronic lock of some form so that he wouldn’t need to worry about losing keys and getting locked out. This brought me to the idea of using RFID and NFC technology to create an electronic door lock.

RFID still uses their equivalent of keys, tags, which means that there is also the same risk of losing the tag as there is the key. However, with electronic door locks there can always be a variety of different options to implement if you lose the tag. Also, if I was to also use NFC, I could implement smartphone compatibility - I’d find this great because keys are much easier to lose than phones and I have my phone with me at all times.

Something else for me to consider was how I’d go about locking and unlocking the door physically – for example, whether to use an electromagnetic door lock or motorise a bolt. For this, an electromagnet would seem the most logical idea because, in addition to being much simpler to control, if the power was to cut to the system the door would unlock automatically, meaning that anyone inside the room wouldn’t be locked in.

***Investigation***

In order to investigate further into this task, I will need to produce a questionnaire in order to find out what else potential users of this system would like to see in the product and to find out the sorts of functionalities that are and aren’t so useful. I would aim to ask the end users questions on the above as well as the general questionnaire. I will also put a lot of research into existing products that have a similar functionality. As well as a questionnaire, I will also conduct an interview to hear a more detailed opinion from a potential user.

***Existing Similar Ideas***

* Digital door locks:

This includes keypads which will open and lock the door, but I cannot find an existing product that has the ability to log users entering.

* Car door locks:

Generally, most cars have buttons on the key fob which can open and close the car, but again generally don’t log the information or have any secondary unlocking methods.

***End Users***

People who might find it useful would be businesses that require monitoring of who comes in and out of their building and full control of this, as well people needing more advanced house security.

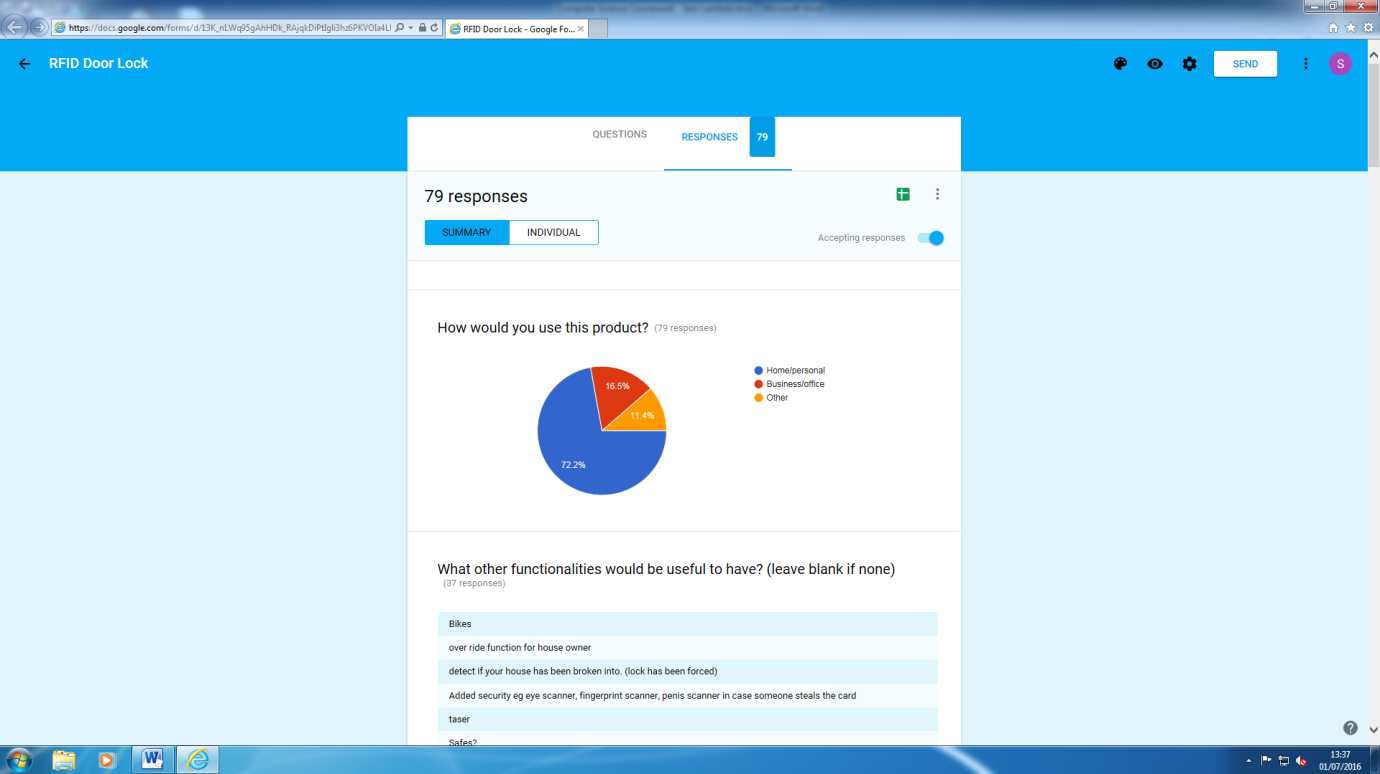
Additionally, I can see this being useful for people who supply foreign students with accommodation in their house as the lock could prevent each student from going into each other’s rooms.

***Hardware and Software***

As I said before, I will be using a Raspberry Pi and perhaps an Arduino to control the door and provide the information for the database for the app or website. I will use Python to write the code because it seems like the easiest option for the Raspberry Pi. In terms of the Arduino, if I was to use it I would get it to collect the data from the RFID reader and send it to the Pi to be interpreted and organised. The reason I don’t just use the Raspberry Pi is because the Arduino would allow me to do many more things mechanically that the Pi is incapable of, mainly because it has analogue as well as digital GPIO, instead of just the Pi’s digital GPIO. Additionally, the Pi can only output 5v, and conventional magnetic door locks require 12v, so the Arduino would be more useful to control the lock than the Pi as it is easier to cope with this.

***Investigation and Analysis I - Questionnaire***

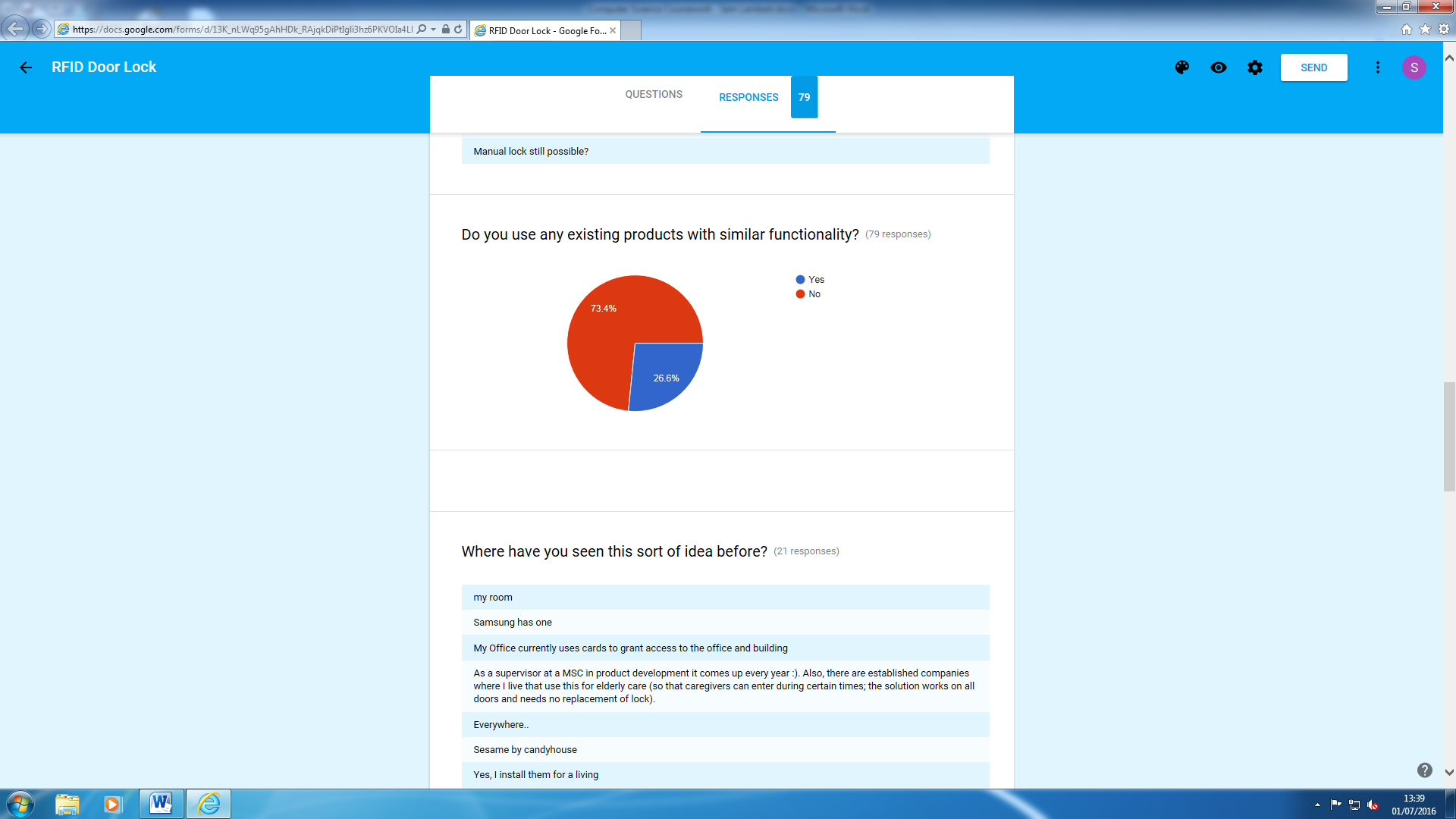
To begin my investigation, I’ve created a questionnaire to consolidate information about what potential users think about the idea and potential improvements.

The first question I asked was how people would use this product, and 75% of answers said that it would be for home/personal use, 15% said that it would be for business/office use and the remaining 10% had other uses in mind, such as safes. Therefore, for this project I will aim to specify it towards a home/personal system, however I will add functionalities in order to make it viable for businesses and offices and will make the entire system fully automated.

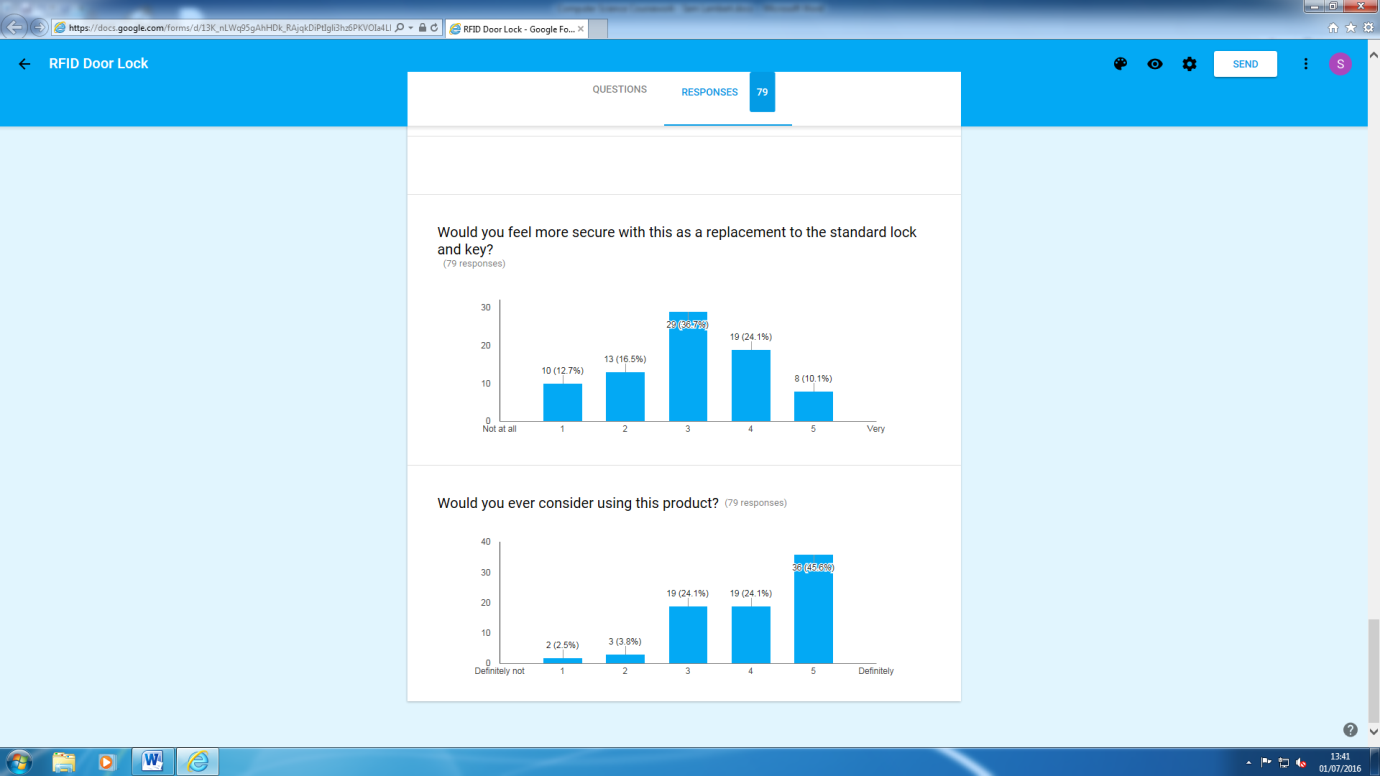
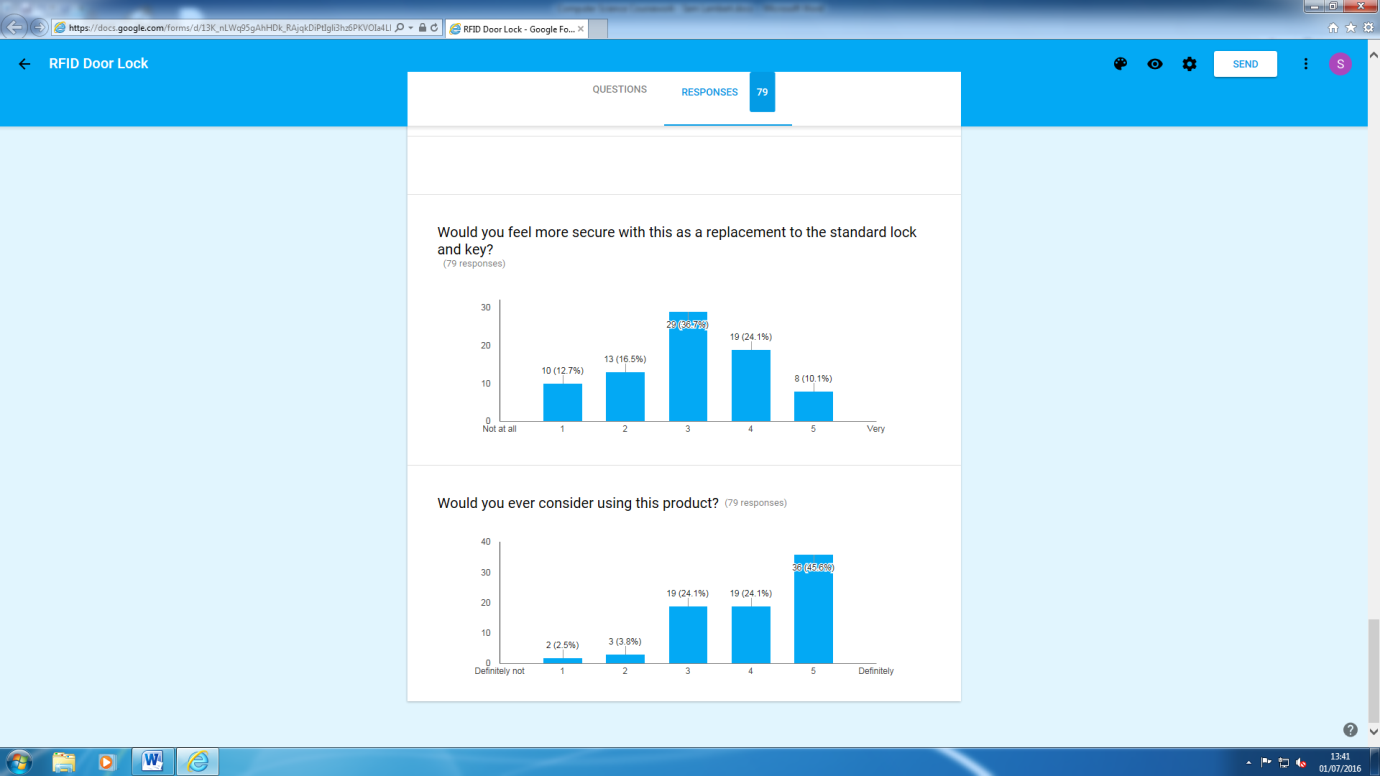
The next question was what other functionalities I could implement. I received lots of feedback in this area so have consolidated the list to viable and necessary additions. Some of the extra functionalities are as follows:

* Detection for if the lock is forced (if the house/room is being broken into)
* Storing the date/time someone entered the room
* Logging good tags/bad tags with date/time
* Card/key emulation on NFC smartphone
* Unlock/lock the door on a schedule
* Automatically lock behind you
* Physical override if system fails (cuts power to electromagnet)
* Success/fail indicator (e.g. green and red LED)
* LEDs to show current status of the door

The next question was whether people currently use products of similar functionality, which is vital to see how other similar systems work and also to see how unique the system would be for home use especially. 70% of responses said that they don’t currently use any similar system in their everyday life. The other 30% stated a variety of different uses they currently have for similar systems such as at the gym, in various product development firms and making their own projects. None of these show any specific home use which means that this will be relatively unique.



To make my project more unique, I asked what sorts of suggestions would make it more so. A fair few people said to focus more on what appeals to home users over office users, and other suggestions include knock pattern recognition and other unlocking options.

Finally, I asked how secure people would feel using the product (on a scale of 1-5, not very secure to very secure) - 40% of people gave a 3; 20% of people gave 2 and 4; 10% of people gave 1 and 5. This is extremely balanced, however the general consensus is that the majority of people would feel at least relatively secure while using the system. I also asked if they’d consider using the product (on the same scale) and only 5% said 1 and 2, and the majority said 5, which shows me that almost everyone who took the questionnaire would at least consider using it, if not definitely use it.

***Investigation and Analysis II - Interview***

I interviewed a potential client who was interested in this sort of project in order to obtain a few more direct and detailed answers to some of the questions I used in the questionnaire as well as a few different ones. Below is a transcript of the interview. I began by describing the product and functionalities to the interviewee and made sure that they understood how the system would fundamentally work, and then followed onto the questions.

***Investigation and Analysis II - Transcript***

{S - Interviewer, C - Interviewee}

***Q1: In what sort of environment would you be inclined to use this product (such as home or work)?***

C: With this particular idea, I would probably be most inclined to use this at home, however using it at work, say in office blocks or warehouses, obviously you see this sort of technology in a plethora of these sorts of places. The reason I’d prefer to have this at home is mainly because of that web interface you mentioned. Nowadays with all these technological advancements and hacking becoming easier as a result, having this particular system put in place in office blocks would mean that it could potentially be a target for malicious intent. I’d say this sort of thing would happen much less at home as houses with this sort of security are much less common than places where this security is used industrially and in businesses.

S: Excellent, so would you say then that you wouldn’t find this particularly secure wherever?

C: I’m not sure, I think I’d be fairly comfortable using this at home and I believe that many other people may find this even more secure than a key lock. The convenience is also a very big upside, mainly because carrying a tag rather than keys is much more compact, plus my lock at home is very stiff and can be frustrating to unlock so having something you can just touch to a reader to unlock is a very frustration-free solution.

***Q2: Do you have any suggestions for improvement?***

C: To be honest, if I was using this at home I’d want it to be as simple and easy to use. To have just the essentials and a few useful other features would mean less hassle to learn the system and also a lower selling price, which is definitely a desirable attribute for any product. I’d just aim to strip down the system to not be bulky or complicated to use and make it easy enough for a baby to use.

S: So, in terms of improvements you don’t think anything should be added, more that I should remove potentially unnecessary features?

C: I think what you have right now in terms of features is a good balance of the essentials to the potentially desirable but not necessary additions

***Q3: Do you think enough of these sorts of devices are used at the moment?***

C: I’m not sure if I can give a strong answer for this, mainly because it’s a matter of personal preference as to whether someone feels like buying them can be justified, because some people can find them easier and quicker and others will feel the opposite, also how secure someone would feel with technology locking their door as oppose to a physical metal bolt.

S: What about personally? Do you feel like if they were more available, would you be inclined to use them more?

C: I would say that currently home security systems are far too overpriced and overcomplicated, mainly because they integrate with lots of other features such as controlling multiple doors and operating sensors etc. – if something like this was simplified much more and made cheaper, I would definitely at least consider using it.

***User Requirements***

* Simplicity:
  + **It should be fully automated in order to reduce confusion for the user:**
    - **The door should open when the tag is put to the reader**
    - **The door should lock automatically when shut**
    - If the tag is held for 2 seconds, the door should stay unlocked indefinitely, until a tag is put to the reader again
    - If the power cuts, the door should automatically lock
* Hardware Layout:
  + **The key scanner must be linked to the breadboard, which is linked to the Raspberry Pi**
  + **All the electronics and wires will be stored in one place**
  + **Indicators:**
    - **A green LED will indicate if the door is open**
    - **A red LED will indicate if the door is locked**
    - If there is a failed attempt to lock/unlock the door, the red LED should flash 3 times.
  + The entire system will be powered through mains as the Raspberry Pi draws quite a lot of power, which wouldn’t be sensible for a battery.
* Web Interface:
  + **It should be an easy-to-use and simple web interface**
    - **View who opened the door and at what date/time**
    - **The date/time format should be ‘[Day (word)], [Day (date] [Month (word)] [Year], [Hours]:[Minutes]:[Seconds]’ (24 hour)**
      * **For example, for the 9th December 2016 at 2:46pm, the output should be ‘Fri, 09 Dec 2016, 14:46:00’**
  + **It will be hosted on the Raspberry Pi**
  + **A secure login page for access to the interface**
    - **Create password on first view**
      * At least 6 characters long
      * At least 1 number
    - **A password should be entered every time the page is loaded**

***Design***

During my design, I will be using a series of flowcharts to help make sense of the different scripts and functions and how they should interact with each other. By using flowcharts it will mean that I’ll be more confident when writing the code; hopefully, they will also allow me to keep the code as simple and optimised as possible as it will limit the processes down to what I need as opposed to perhaps some unnecessary functions and procedures. Although some of the flowcharts that I have made may seem very small, I feel like it is necessary to make sure I have covered all the possible processes that the system can perform in order to minimise risk of making errors.

***Flowchart 1***



This flowchart is the fundamental structure for the first function which should be run when the Pi is switched on and the system is booted. As you can see, when the ‘main’ script has been started, the script turns on the power to the electromagnetic door lock automatically. This means that if the Pi for some reason loses power, when power is restored the door will automatically lock for security purposes. After the door has been locked, the ‘door’ function is looped indefinitely. This function will be explained on the next page.

***Flowchart 2***



Here, the main body of the program is set out. This controls the door for when the user scans their tag, unlocking and locking the door dependant on its original state. This function is the ‘cross-roads’ for all of the other essential routines in the system, as it checks the tags RFID number against the user database and runs the lock and unlock functions.

***Flowchart 3***



The ‘unlockDoor’ function will be used exactly as it is labelled - to unlock the door. As well as just this, however, it will be used to change which LED indicator is on/off to the opposite state to signal that the door is open. It will make the variable ‘doorLocked’ equal to false so that this information can be used for other functions and for the program to know that the door is unlocked.

***Flowchart 4***



The ‘lockDoor’ function is essentially the same as the previous function except that before it locks the door, it will check that the door is closed before doing so. As the electromagnet will be powerful enough to hold the door shut against someone pulling very hard, it would most likely be hazardous to turn the lock on before the door is closed as it could slam if the door was near shut and therefore could present a danger to the user. It will, however, do exactly the opposite of the previous function in every other respect.

***Flowchart 5***



‘checkTag’ will also do exactly as it is labelled. It takes the value from the tempRFID variable (taken in the ‘door’ function before this function is called) and will query it in the user database. If the result comes back positive, then the ‘goodTag’ will be true, and vice versa if it comes back negative. This will then reflect as to whether the door is unlocked or locked.

***Development - Setup***

To start developing the project, I will first need the following list of components:

* Raspberry Pi
* Breadboard (with ribbon cable)
* PiOT Relay Board (from ModMyPi)
* Electromagnetic door lock
* Green/Red LEDs
* Resistors
* MFRC522 RFID Reader
* AC-12vDC adaptor

Throughout the development, I will be linking all of these components to work together using the various scripts.

To begin, I needed to install all of the relevant libraries that I would be able to use to control and interpret data that is sent from the RFID reader. Because creating a library for the reader was far out of my comfort zone in terms of programming and the specifications of this project, I decided to find a library created by <https://github.com/mxgxw>, the source code of which can be found at <https://github.com/mxgxw/MFRC522-python>. This library utilizes an hardware SPI C(++) extension for Python in order to work. This can be found at <https://github.com/lthiery/SPI-Py>. To install this extension, I used the git command line to clone the repository and then used the setup.py script to install the extension. Below are the terminal commands I used, in the order at which I used them.

~$ git clone <https://github.com/lthiery/SPI-Py>

~$ sudo apt-get install python-dev

~$ cd SPI-Py

~$ sudo python setup.py

In order to use SPI on the Raspberry Pi, I also had to enable it by removing it’s blacklist.

~$ cd /etc/modprobe.d/raspi-blacklist.conf

~$ sudo nano raspi-blacklist.conf

\_\_\_\_\_NANO \_\_\_\_\_

# blacklist spi-bcm2708

blacklist spi-bcm2835

The # is used to comment out the blacklist. The blacklist I have left uncommented is there for a reason which I will explain after the installation of the MFRC522 library.

Once I had enabled the SPI-Py extension, I then needed to fetch the library for the RFID reader. At this point, I decided to create a folder which would contain a directory for my libraries, resources and source code.

~$ cd ~

~$ mkdir rc522lock

~$ cd rc522lock

~$ mkdir lib

~$ mkdir resources

~$ mkdir src

From here, I then downloaded the MFRC522-python repository into the lib directory

~$ cd lib

~$ git clone <https://github.com/mxgxw/MFRC522-python>

As for the example code, I decided to keep the Read.py file for referencing against my main script, but I didn’t need the other files so I removed them to save space.

~$ rm Dump.py

~$ rm README.md

~$ rm Write.py

Before I had blacklisted the spi-bcm2835 module, I ran the Read.py script found within lib/MFRC522 and received the error ‘can't open device: no such file or directory’. I worked out that it was an issue with the SPI module by inserting a print statement before and after the program imports SPI. The first print statement appears and then follows the error, not the second print statement. I found that the current SPI version running on the Pi was spi-bcm2835. The issue was that the SPI extension I had installed was for version spi-bcm2708 and although it was enabled the Pi was using the later version. In order to fix the issue I had to blacklist the new version of the module and add the older version to the device tree in the boot configuration file.

~$ sudo nano /boot/config.txt

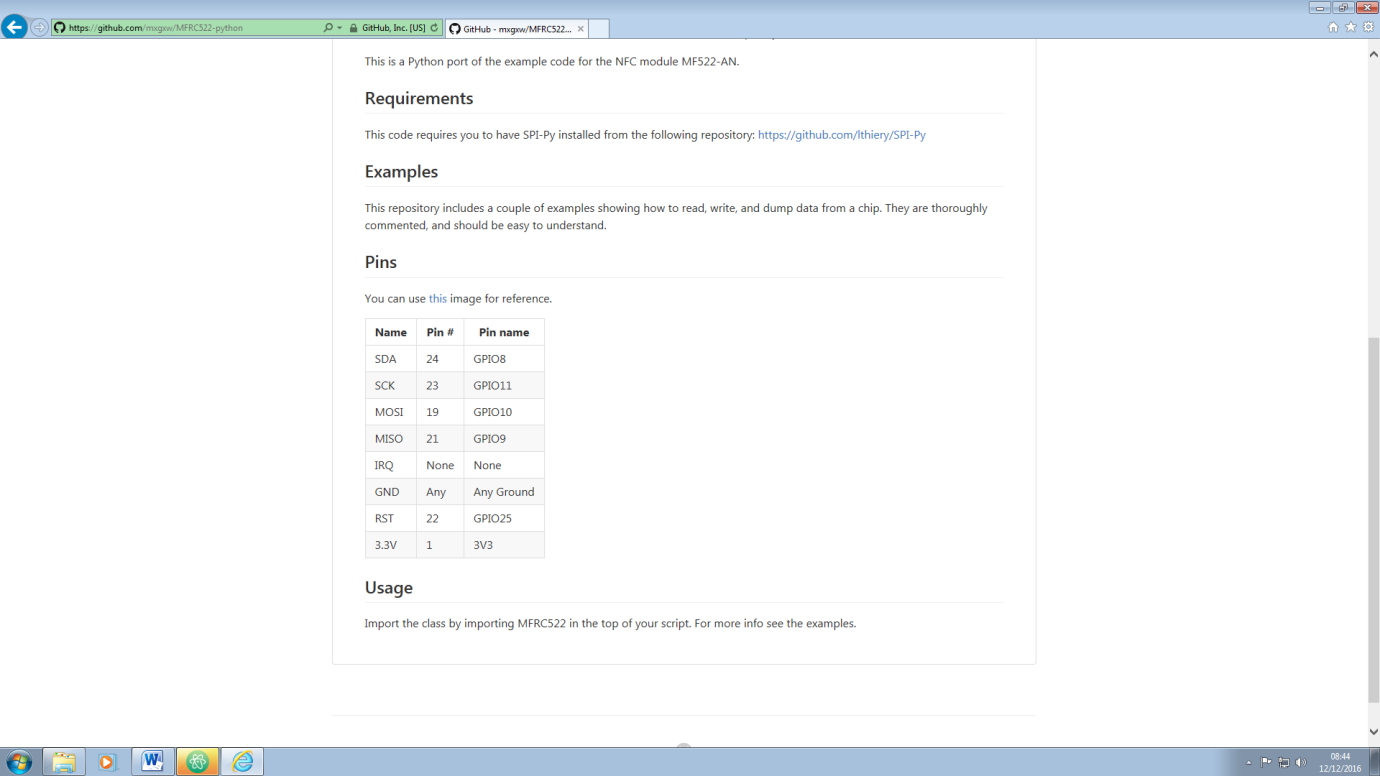
\_\_\_\_\_NANO \_\_\_\_\_

device\_tree\_param=spi=on

dtoverlay=spi-bcm2708

When the module was working properly, I was able to run the Read.py program to test the RC522 reader. For reference, the reader is hooked up to the breadboard (connected to the Pi) as shown below.

Board # reference for RPi



BCM reference for RPi

Name of socket on RC522

The main objective of running the program is to see how the reader returns an RFID tags unique ID number. I found that each tag has ‘blocks’ of numbers stored in a list with 6 indexes. From this point I decided that when it came to building the script, I’d effectively take all the numbers inside the list and concatenate them into one string. The reason I wouldn’t use an integer which would be as effective for a list of numbers is because I will be using the UIDs that are scanned to query a database. With SQL in Python, you query a database with SQL by passing a string to it with the SQL content. Another reason to have it as a string instead is because it is much easier to concatenate the numbers together compared with integers.

For example, I have 4 variables and each of them have different values as shown:

int1 = 5

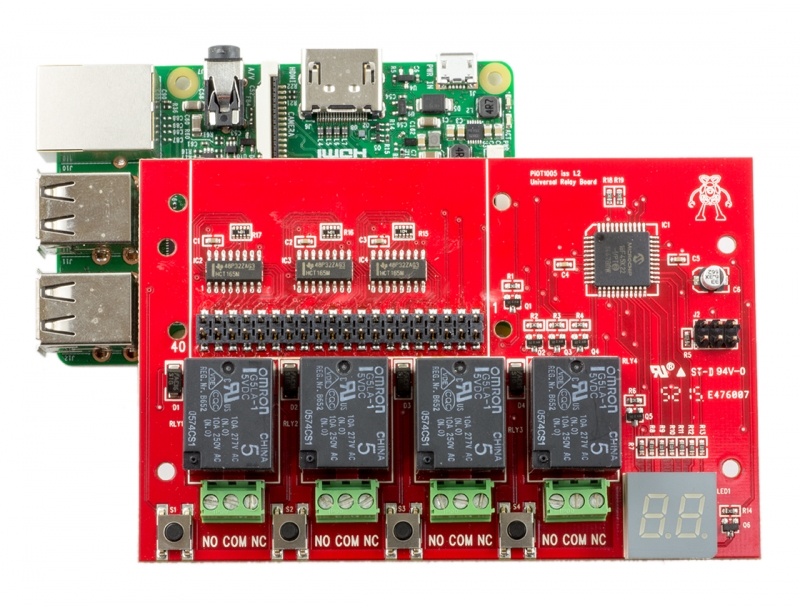
int2 = 3

str1 = “5”

str2 = “3”

If I was to do int1 + int2, it would return 8, as 5+3 is equal to 8. If I was to do str1 + str2 then it would return with 53, as when you add string together it concatenates the values rather than mathematically adds them.

All of this means that I can have a script which takes the RFID unique ID list and concatenate it into a single string. For example, if I have the list [24, 637, 15, 944, 67, 18], I can make that a string which reads 24637159446718 which works perfectly as a string to query a database.

Once I had verified that the reader was functional and had it working with the Pi, I decided to set the PiOT relay board up. Below is an image of the board. As you can see it is connected through all 40 GPIO pins in the socket (circled in blue).

[](https://www.modmypi.com/image/cache/data/rpi-products/gpio/40-pin-headers/40-pin-gpio-connector-header-extender-extra-tall-800x609.jpg)

At first I saw that being connected to all the pins could be an issue with connecting the breadboard however they’re all shipped with the extender shown on the right. I also asked the ModMyPi.com forums to see if the board utilised all 40 pins at once and if not if I’d be able to use the rest of the pins and the response I received was good, saying that the board used very little current from the 3.3v and 5v power pins and that it would only need to use 1 GPIO pin per relay, which could be configured with the on-board interface. I decided to configure the far right relay (circled in yellow) to use pin 3 in order to control it. I chose this pin because it’s at the back of the breadboard and isn’t likely that I’ll need to use it for anything else such as LEDs or other components. Also, the ribbon cable which I will be using to attach the Pi and relay board will run over the other three relays and therefore will obstruct and make it difficult to use them.

Lastly I wired up the LEDs to the breadboard. I used (BGN) pins 13 and 26 for the LED anodes and used a resistor for each LED to attach the cathodes to ground. After this was complete, I was ready to start building the code.

In terms of developing the code, I first had to import all of the libraries that I needed. Previously in the setup, I already had installed the SPI extension to allow me to use the MFRC522 repository, though I also needed access to the Raspberry Pi’s GPIO interface. Whilst I did this I also took the chance to create the database and link it to the program.

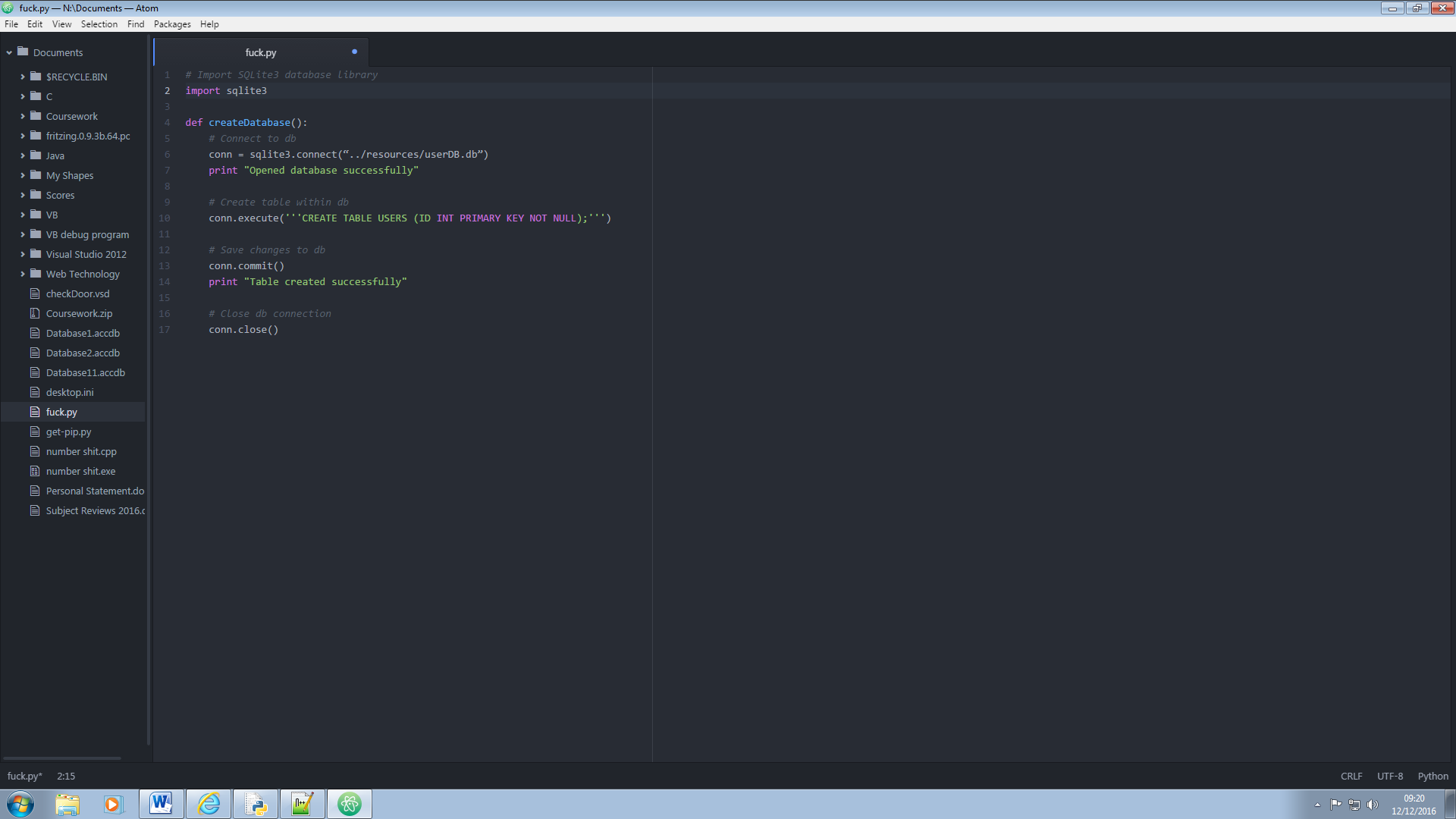
Before I did this, I decided to reorganise my directories and create new ones to cater for all of these programs. To easily do this and also to keep my code up to date, GitHub was by far the best option. I created a repository on GitHub and linked it to the directory on the Raspberry Pi. I then cloned it onto my PC so that I could easily code with vastly superior editors, push it up to my repository on GitHub and then pull it down on the Raspberry Pi to test.

The following is my directory tree, all of which is contained in the master parent directory.



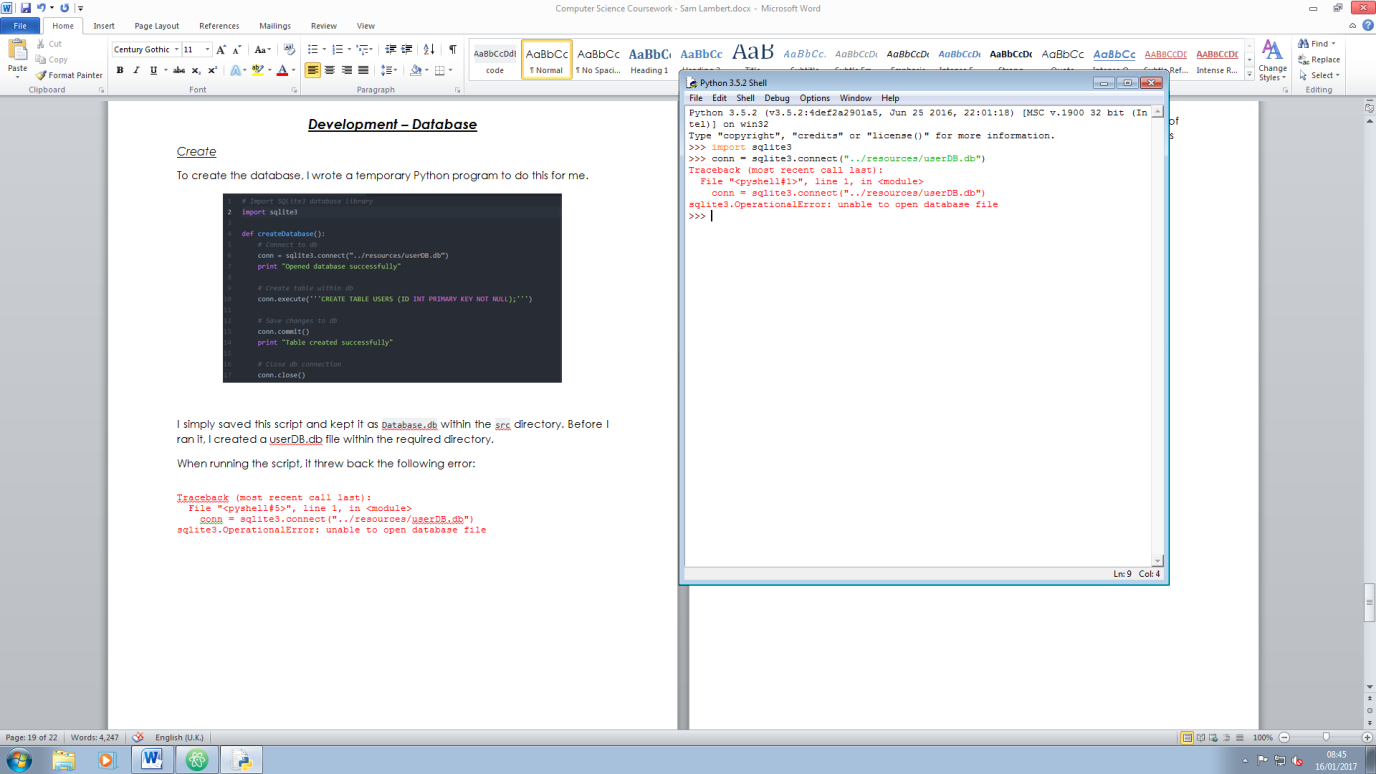
***Development – Database***

*Create*

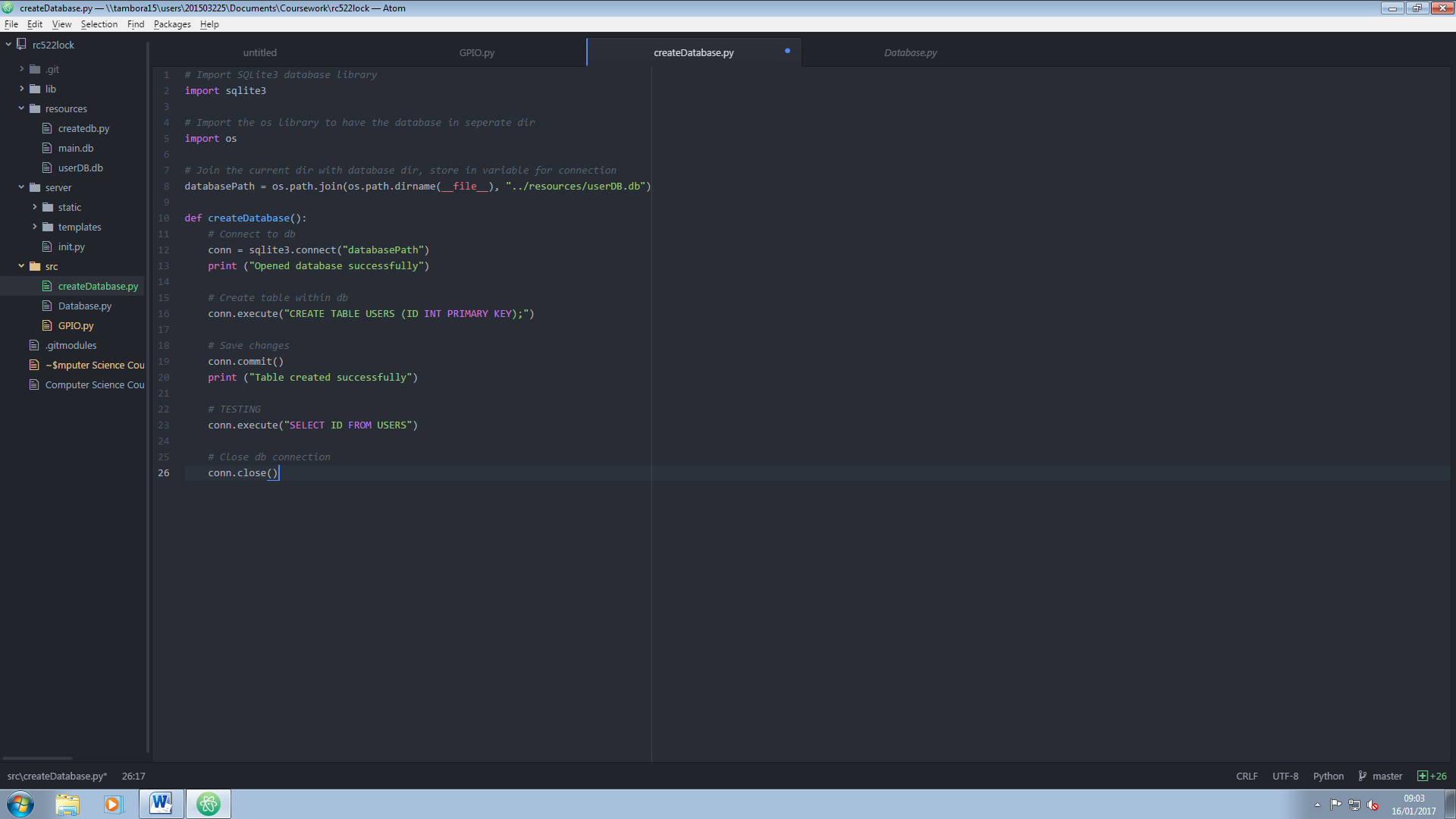
To create the database, I wrote a temporary Python program to do this for me.

I simply saved this script and kept it as Database.db within the src directory. Before I ran it, I created a userDB.db file within the required directory.

When running the script, it gave me the following error:



After researching similar errors, the fix for this was to define a variable at the start of the script by joining the path for the database onto the path of the python script.



*Test*

When running the final version of this code, both print statements were shown and the database was successfully created without errors. For additional testing, I added a statement (labelled # TESTING) which would return an error if the table wasn’t created as I needed it to. This didn’t return an error and therefore the database was created successfully

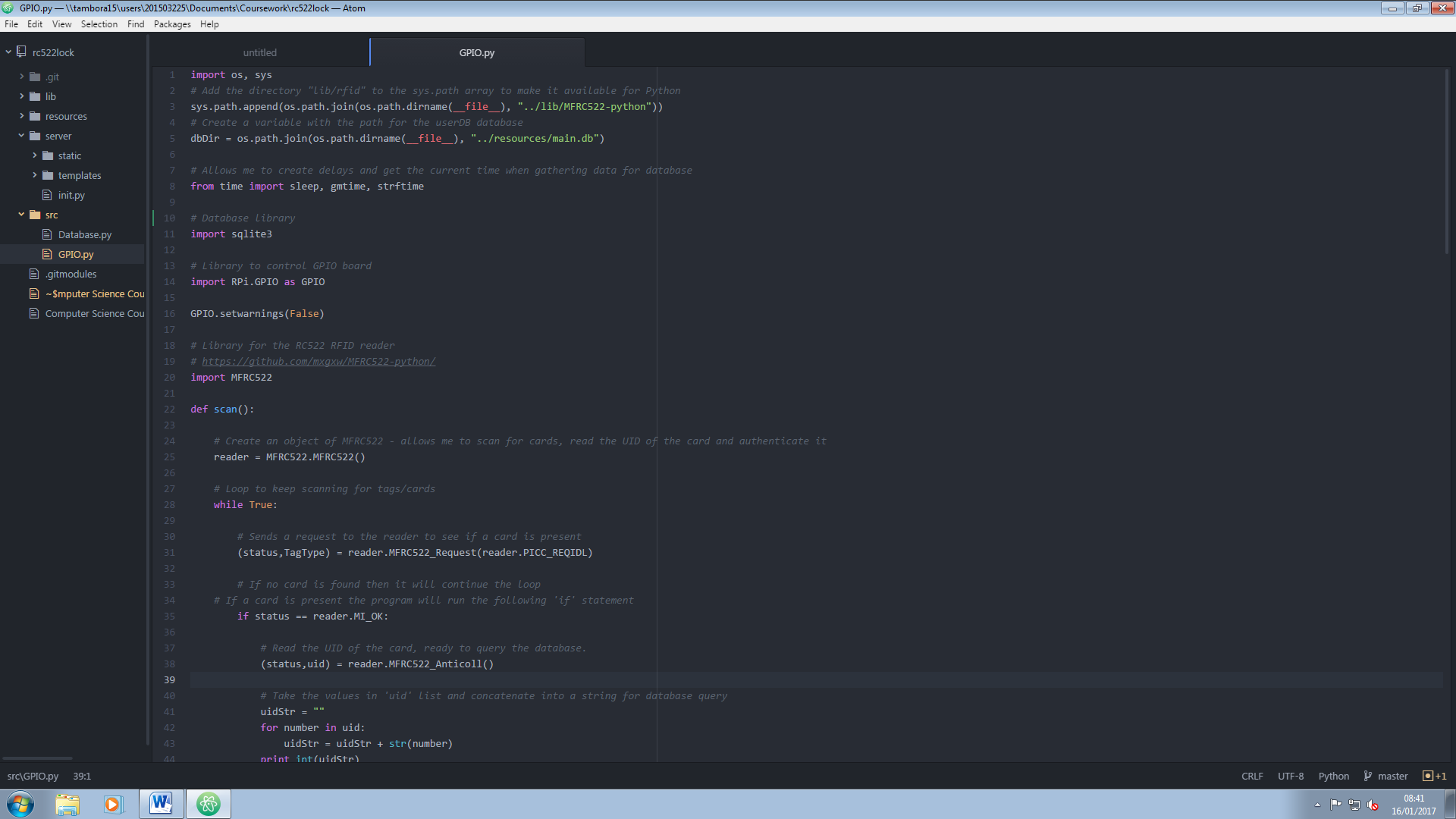
**ADD A PHOTO (EVIDENCE)**

***Development – Main Program***

This is the development for the main program which will be running 24/7 on the Raspberry Pi.

*Create*

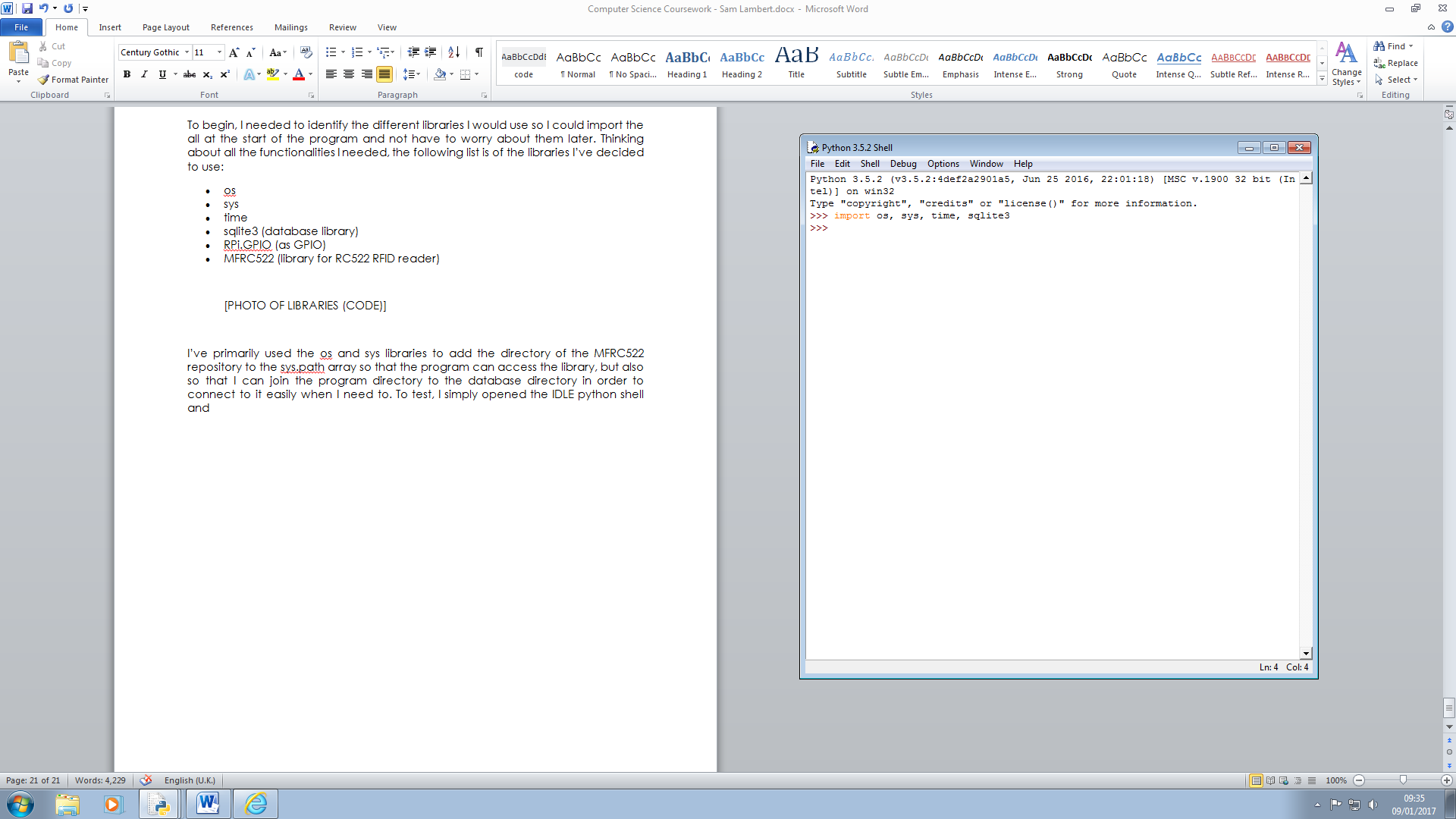
To begin, I needed to identify the different libraries I would use so I could import them all at the start of the program and not have to worry about them later. Thinking about all the functionalities I needed, the following list is of the libraries I’ve decided to use:

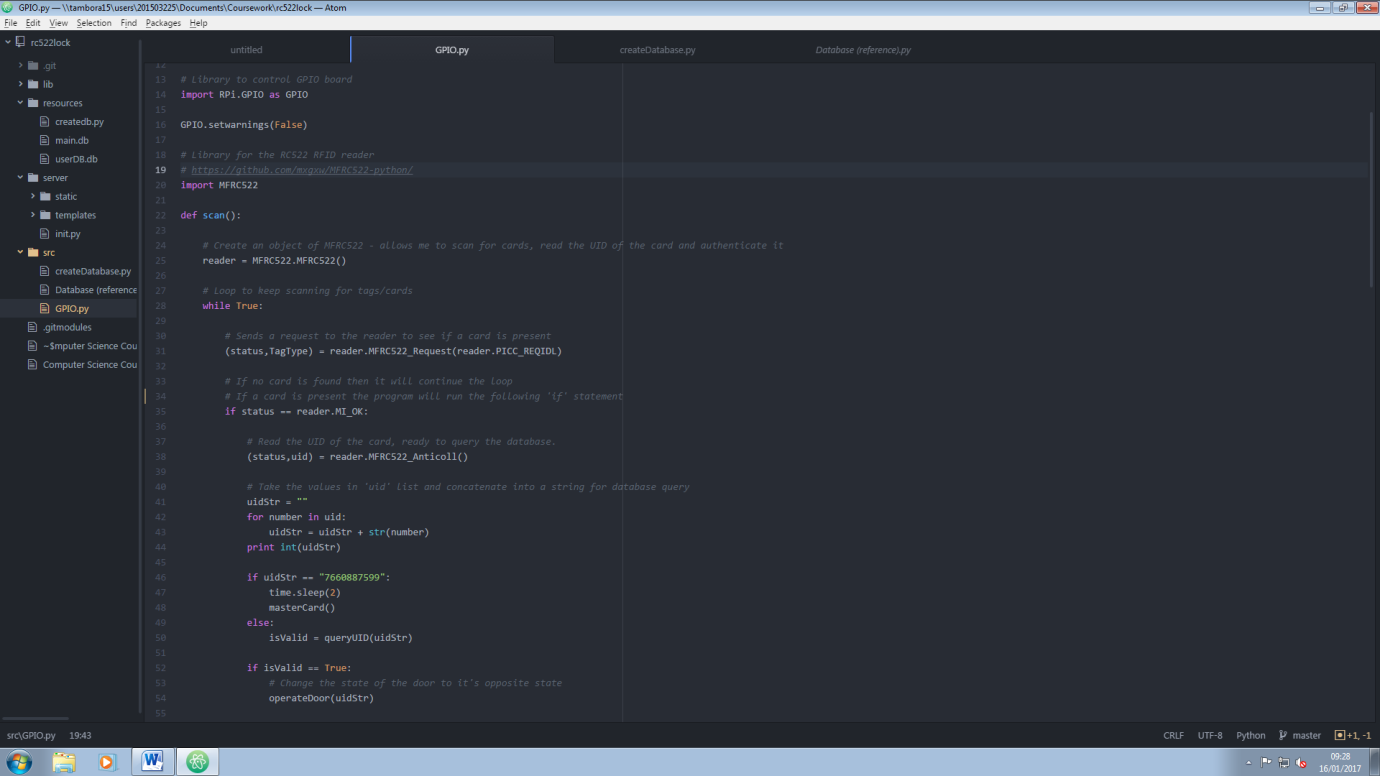
* os
* sys
* time
* sqlite3 (database library)
* RPi.GPIO (as GPIO)
* MFRC522 (library for RC522 RFID reader)

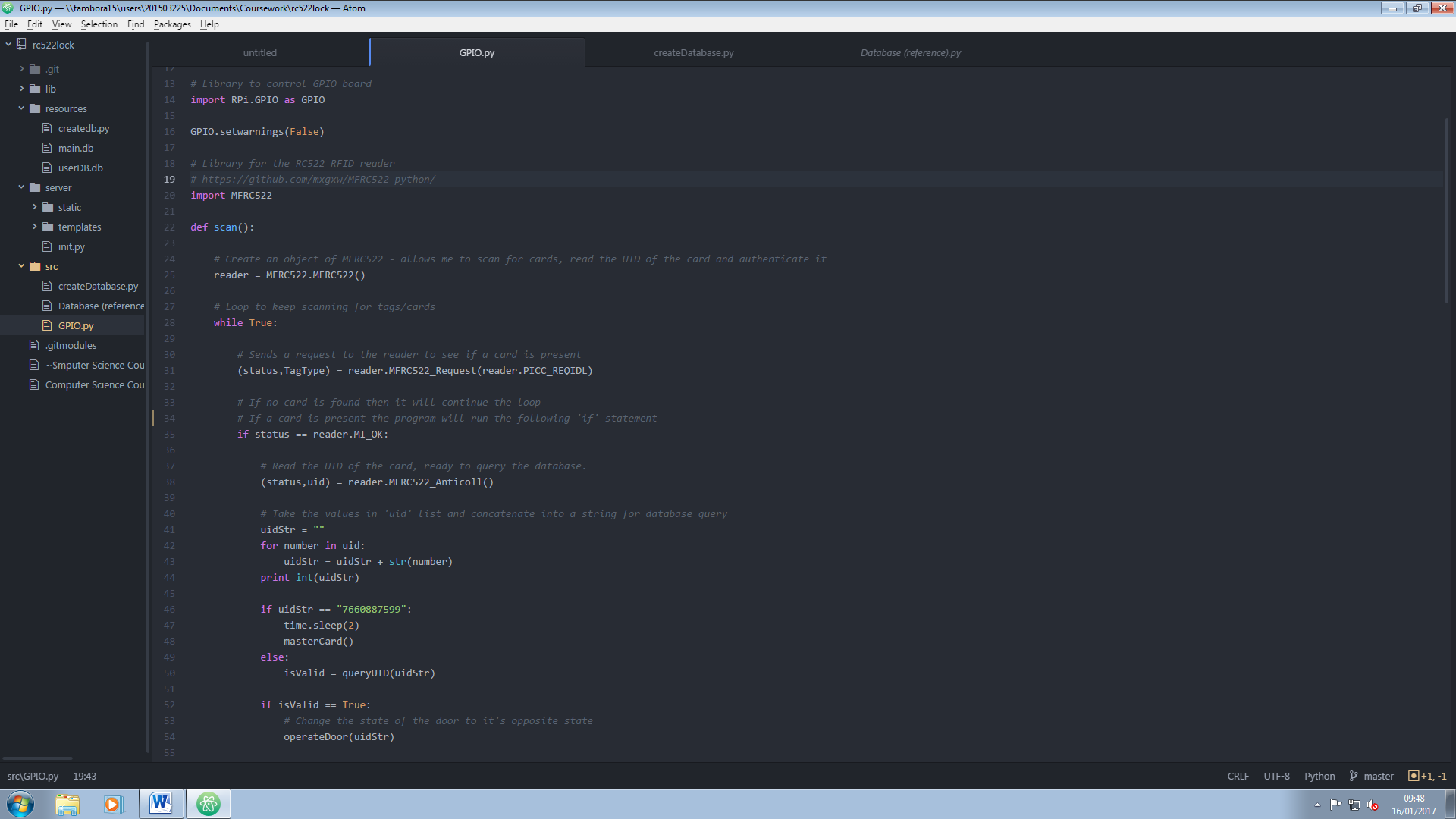
I’ve primarily used the os and sys libraries to add the directory of the MFRC522 repository to the sys.path array so that the program can access the library, but also so that I can join the program directory to the database directory in order to connect to it easily when I need to.

*Test*

I simply opened the IDLE python shell and attempted to import all of the libraries except the MFRC522 and RPi.GPIO libraries as they would only work on the Raspberry Pi

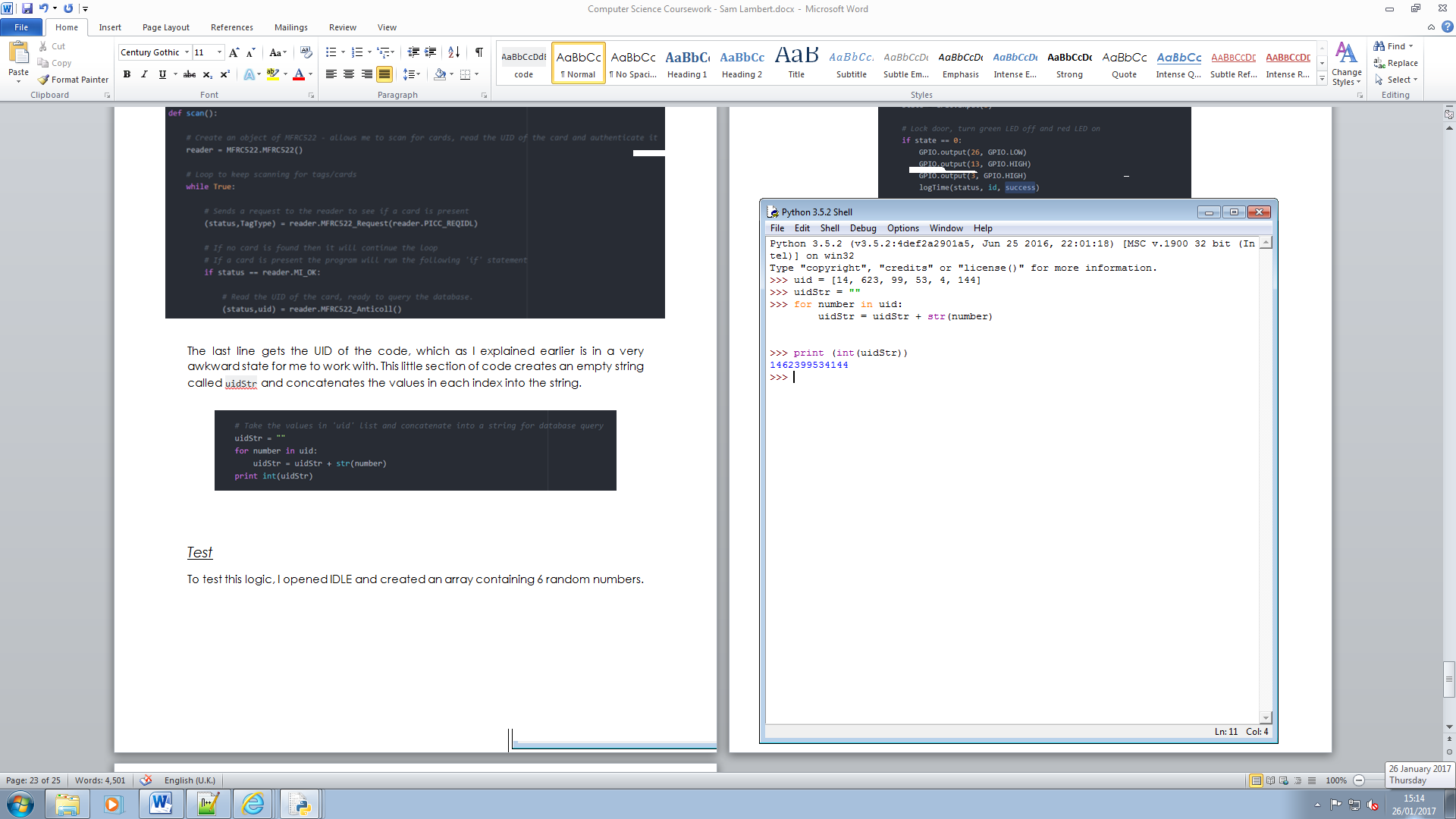


I had to then work out how to use the MFRC522 library to interact with the reader. Thankfully, the repository came with a resource which had an example program which effectively created a loop which continuously scanned for any present cards near the reader, and when it found a compatible card it read the UID. The script however was uncommented, so I played around with the example code and the reader to understand the code and then continue to add my own comments. The only bits that were essential from the example script were the following lines of code.

The last line gets the UID of the code, which as I explained earlier is in a very awkward state for me to work with. This little section of code creates an empty string called uidStr and concatenates the values in each index into the string.

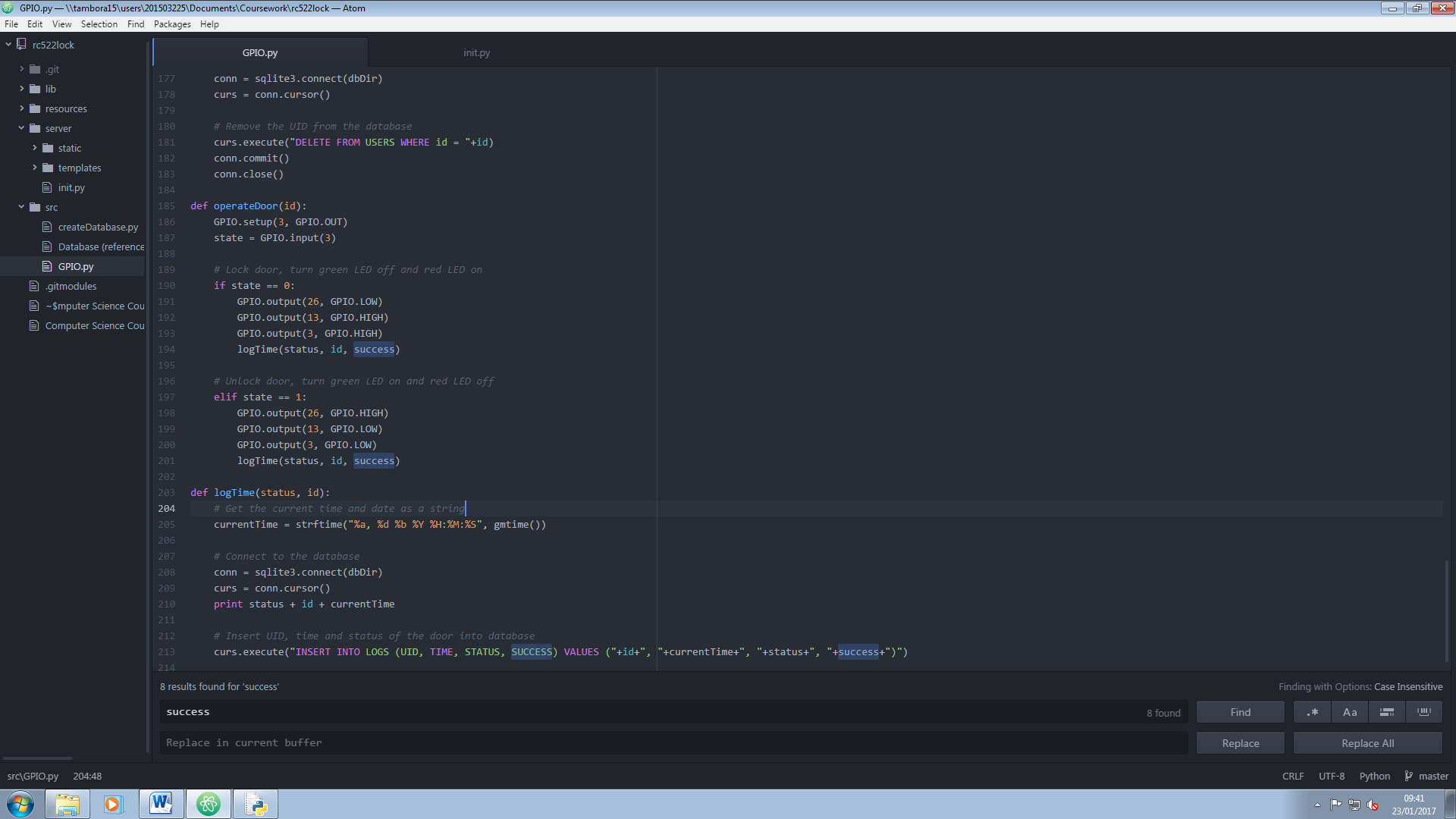
*Test*

To test this logic, I opened IDLE and created an array containing 6 random numbers: 14, 623, 99, 53, 4 and 144. The output I aimed for was 1462399534144



As you can see, the number output through the print function is as I wanted it to, and therefore this test was a success.

operateDoor()

Before I worked with the database to validate whether a card was valid, I created the script to operate the door, which would work with any compatible card for the time being. For the door operation, I built a separate function that I could call whenever I needed to. This also allows me to easily build in any emergency functions I choose.

The green LED is connected to pin 26 and the red LED is connected to 13. This small function reverses the current state of the door and switches the LED states. So, if the door is locked, it will unlock when the function is called, the green LED will turn on and the red LED will turn off, and vice versa. GPIO.input(3) gets the state of the door currently. 1 means that it is locked, 0 means it is unlocked.