

Technical Report



ABENA[®]

Acknowledgements

The prototype of this device was made possible through the invaluable contribution of “Team Abena”:

- Abdallah Abdel Qader
- Alberto Miro
- Christina Joy Moses
- Judith Andrea Kröll
- Paul-Ioan Maghiari
- Povilas Janusauskas
- Rolandas Kraujelis
- Ronan Machado
- Sharva Yatin Nemane.

We would like to thank Roana de Oliveira Hansen for supervising us throughout this semester and Syddansk Universitet for giving us the opportunity to innovate and manufacture this device that has the potential to aid many in the future. We would also like to thank Abena, the company that gave us foundational advice that led to us creating a medical device.

Foreword

This report details the process of creating the “Sortena” – from conceptualizing the device to creating a prototype of the device. This project aims to make a device that simplifies the process of taking pills as a response to a medical condition. Initially targeted toward senior citizens, the prototype of this device has five containers where the user can store their pills. The prototype also has an in-built display with a graphical user interface (GUI) where the user can configure a profile and set their unique dosage schedule. Additionally, the device comes with a phone application that is intended for a nurse or other medical professional to set up their patient’s dosage schedule.

Index

Acknowledgements	2
Foreword	2
Index	3
Work distribution	1
Functionality & Design Research results	1
Risk assessment	4
Table for risk analysis	4
Mechanical risks	7
Embedded risks	8
Product delimitation and Choice benefit analysis (Abdallah)	9
Design	10
Dispensing Mechanisms ideation	13
Technology choice	15
Microcontroller	15
Motors	17
Electrical components	20
Electric circuits	21
Pump	21
Motors	21
Microcontroller	21
Material choices	22
Graphical User Interface	22
Language	22
GUI Toolkits	24
Clock & Home screen	25
Keyboard	27
Window Organisation	29
Database selection	30
Layout and Formatting	33
Phone Application	37
Conclusion	39
Bibliography	40

Appendix	42
Layout and Formatting	44

Work distribution

Table 1. Work distribution

TEAM MEMBER	CONTRIBUTION TO THE PROJECT
Abdallah Abdel Qader	-display programming -choice benefit analysis
Alberto Miro	-Motor selection & coding -App development
Christina Joy Moses	Coding the GUI: - layout and formatting - keyboard & windows (binding)
Judith Kröll	-Project organisation -Market opportunity identification for intelligent healthcare products -Design & function feedback sessions with potential users
Paul-loan Maghiari	-Mechanical design -Assembly and adjustment of prototype
Povilas Janusauskas	-Assembly of prototype -Power supply -Sensor Selection
Rolandas Kraujelis	-Sensor selection -Power supply
Ronan Machado	-Database setup and saving mechanism -Risk assessment
Sharva Yatin Nemane	-Home screen & buttons setup -Microcontroller and LCD setup

Functionality & Design Research results

When comparing “Sortena” to the competing products from the first report (PharmAdva MedaCube, Livi smart pill dispenser, e-pill MedSmart PLUS, Hero pill dispenser and Spencer -in-home Medication Dispenser), the main competing factor is still price, which is possible due to focusing this project on basic functionalities.

It is trimmed down to what the team considered the most necessary functions while undercutting competitors on price matters and independence of the product.

Secure transmission of health data is an important factor to take into consideration, especially when transferring personal information. There are ways to implement that, which would likely involve a specified service provider, which in the case of this prototype would go beyond the defined and achievable scope.

In regard to research for the desirable functions and visual components of the product mostly primary research was conducted, in the form of questionnaires, reaching 13 health and outpatient care workers and conducting more in-depth interviews and question sessions with two of them throughout the design and improvement process.

Since the interviews were conducted in German, here a summarized and put into structured textual outcome of the surveys.

When asked how the average home care patient would be described the portrayal landed on an around 80-year-old person on light to medium level of ambulatory care, getting checked on once a week, this is the patient which the following accounts and generalizations will be based on.

The patient tends to take around 5 different pills per day, with the times of pills taken divided into morning, noon, evening and night, signifying a pill intake just before bedtime.

A lot of medications are to be taken with food, implying that the pill dispenser location in the dining room would make sense. In case of a less mobile patient a flexible location setup is recommendable though.

When it comes to physical features the product should be built to accommodate people with gross motor skills, as in big buttons, no sharp edges, and no delicate components to handle with precision.

Another helpful feature will be a loud acoustic signal, since hearing does deteriorate in most people by the time, they reach the age of 80.

When putting this product on the market in different countries, the language settings should be adaptable, since it is not that common in the current older generation to fluently understand English and it is easier for everyone involved if no guesswork must be part of daily routines, rather sticking to the native language and implementing intuitive symbols and colors is advisable in this case.

There are different mediums that medication comes in, so adding a compartment containing for instance injections or a different drop zone for fizzy tablets meant to be dissolved in water were suggested for helping with that, possibly even adding a part responsible for grinding a pill, adding water and being able to flush that mixture to a stomach tube.

Having the opportunity to access a select few medications on demand like allergy medication that can be requested as needed and a maximum dosage per day was also brought up.

An additional suggestion was voice control, helping patients that would rather not rely on their motoric skill when interacting with the system.

When asked what the struggles with pill intake at the time being for most patients was, the problem most mentioned was the struggle to get the pills out of their packages before taking them.

Other suggestions from this line of questions were to install a detachable cup that can be used for direct intake of the pill, since they mostly get chugged in one go anyway and this way do not need to be steered from cartridge to hand to mouth but have a more direct route less prone drop pills and having to strain oneself looking for them and picking them up after. This cup is also suggested to be machine-washable, since throughout the course of a day many things tend to accumulate in the corners of mouths and the pill dispenser should be a sanitary product not fostering bacteria.

Relevant features for the pill dispenser's display and app features are primarily the alarms and reminders for taking pills for the patients and a special stress on information to family/caretaker if the medication has not been taken in a specific time frame. Normally a time frame of 2 hours is okay for delayed medication intake but for some of them like for example Parkinson medication a time frame of no more than half an hour is crucial to the well-being of the patient, so reminders and alert periods should be adjustable for the different medications.

Also suggested was an adaptable coupling of the pill intake with meal times, so the pills that need to be taken sober get taken beforehand.

Since elderly people generally do have quite regular meal times that should be easily realizable and not change too much after customized to a patient.

The interviews were very insightful, and information gained from them got evaluated together, followed by being adapted and applied to our scope and possibilities, as will be explained in later chapters of this report.

Risk assessment

Table for risk analysis

For this project, the risk analysis has only been performed towards the end of the project. It is generally preferred to have the risk analysis conducted during the development process, to try to predict future problems.

Table 2. Probability of occurrence

Probability of occurrence levels (P_o)	Geom. Ave. value	Minimum limit	Maximum limit	Class	Example
1	-	-	1.00E-06	Incredible	Residual risk
2	3E-0.6	1.00E-0.6	1.00E-06	Very improbable	Redundant systems
3	3E-0.5	1.00E-0.5	1.00E-04	Improbable	Robust mechanical or electronic parts
4	3E-0.4	0.0001	0.001	Remote	Mechanical or electronics device
5	0.32%	0.1%	1%	Occasional	Sensitive components
6	3.2%	1%	10%	Probable	Human errors (paying some attention)
7	32%	10%	100%	Frequent	Human errors (paying little/no attention)

The probability of a situation occurring is combined with the probability of a failure mode and is defined as the probability of occurrence. This probability then gets multiplied by the probability of a corresponding failure mode, which provide us with the final probability of occurrence (P_o). The probability of failure is highly dependent on the equipment/components being used.

Table 3. Probability of detection

Probability of detection levels (P_d)	Geom. Ave. value	Minimum limit	Maximum limit	Class	Example
1	99.997%	99.990%	99.999%	Automated, validated det.	Automated detection/prevention actions, validated and without need for human interference.
2	99.97%	99.90%	99.99%	Very high level of detection	In Line Test and Procedures in Production
3	99.7%	99.0%	99.9%	High level of detection	Trained, Alert Human Detection
4	97.0%	90.0%	99.0%	Remote	Informed Human Detection (Of Situation)
5	70.0%	90.0%	99.0%	Occasional	Non-Informed, Non-Alert Human Detection

Probability of detection will be used as the probability of a situation being identified by the code, sensors, or humans. In an ideal world, detection would mean that the corresponding failure mode can be ignored since no harm can happen.

Likelihood of harm			
$P_o + P_d$		P_{max} (harm)	Class
2 – 6	1	1.00E-06	Incredible
7	2	1.00E-05	Very Improbable
8	3	1.00E-04	Improbable
9	4	1.00E-03	Remote
10	5	1.00E-02	Occasional
11	6	1.00E-01	Probable
12	7	1.00E+00	Frequent

Table 4. Likelihood of harm

The likelihood of harm is the just the combination of the scores of both the probability of detection and occurrence. This score goes from frequent (occurs almost every time) to incredible (basically never happens)

Table 5. Severity levels

Severity level	Possible occurrence
5 – Life threatening	Death
4 – Severe/permanent injury	Permanent significant disability
3 – Transient disability	Transient but significant disability/permanent minor disability
2 – Transient or minor disability	Transient minor disability
1 – Inconvenient/No physical harm	No disability or physical complaints anticipated. Annoying complaints

The levels of severity show the seriousness of a given situation. It goes from inconvenience/no harm to life threatening.

Table 6. Risk categories

		Inconvenience. No physical harm	Transient or minor disability	Transient Disability	Severe. Permanent Injury	Life Threatening
Likelihood of harm		1	2	3	4	5
2 – 6	1	Broadly Acc	Broadly Acc	Broadly Acc	Broadly Acc	ALARP
7	2	Broadly Acc	Broadly Acc	Broadly Acc	ALARP	Intolerable
8	3	Broadly Acc	Broadly Acc	Broadly Acc	ALARP	Intolerable
9	4	Broadly Acc	Broadly Acc	ALARP	Intolerable	Intolerable
10	5	Broadly Acc	ALARP	Intolerable	Intolerable	Intolerable
11	6	Broadly Acc	ALARP	Intolerable	Intolerable	Intolerable
12	7	ALARP	Intolerable	Intolerable	Intolerable	Intolerable

In table 5 above, the likelihood of harm and the severity levels are combined to give us a score for the risk of each situation. The levels of risk are:

- Broadly acceptable (green)
- As low as reasonably possible (orange)
- Intolerable (red)

Mechanical risks

The most serious risk related to the mechanical aspect of the project was chosen, incorrect number of pills dispensed. This could lead to multiple dangerous situations in which the user could be put in harm.

Table 7. User situation and failure modes

User Situation	User event	Failure Mode	Failure Mode Cause	Probability of occurrence	Probability of detection	Severity level	Risk category
Incorrect number of pills dispensed	User does not get correct mediation	Not enough suction to lift pill	Wear and tear	2	4	2	Broadly accepted
			Shorted circuit	2	5	5	Intolerable
			Electric failure	2	5	5	Intolerable
		Vacuum is blocked	Wear and tear	2	4	2	Broadly accepted
			Misuse	3	5	4	ALARP
			Dirt in machine	3	5	3	ALARP
		Pill not dropped into container	Wear and tear	2	4	2	Broadly accepted
			Motor failure	3	5	5	Intolerable
			Software failure	2	5	5	Intolerable
		Pills stuck together	Wear and tear	2	5	2	Broadly accepted
			Dirt in machine	3	5	3	ALARP

Embedded risks

Table 8. User situation and failure modes

User Situation	User event	Failure Mode	Failure Mode Cause	Probability of occurrence	Probability of detection	Severity	Risk category
Incorrect data/loss of data	User does not get correct mediation	Input data is incorrect	User error	3	3	5	ALARP
			Software failure	2	5	5	Intolerable
		Data breach	Password of database leaked	1	5	5	ALARP
			Hacked	1	5	5	ALARP
		Loss of connection to database	Loss of power to Raspberry Pi	2	5	5	Intolerable
			Software failure	2	5	5	Intolerable
		Motor drift	Software failure	2	5	5	Intolerable
			Loss of power to Raspberry Pi	2	5	5	Intolerable

Product delimitation and Choice benefit analysis (Abdallah)

As for choosing to build a medical pill dispenser, there are a lot of decisions of what functions to include which to exclude, and why. First, we did market research to see what other companies and competitors have in store, and which functions they have in their dispensers. Our main demographic for our device is older people, therefore we decided to keep the functions as basic and simple as possible.

Some of the functions we were thinking of including:

In regard to the pill dispensing system, one of these would be implemented:

- Outputs the prescribed pills at a certain time and sends a signal
- Sorts pills into weekly assortments
- Outputs specific pills per person

In regard to security and protection:

- Protected medication with some sort of lock system, pin code or fingerprint

Notifications and App configuration:

- Late, skipped or missed dose notifications
- Caregiver notification
- Reporting of loved ones and when they took medications

Quality assurance

- Temperature and humidity sensor, which displays these values on the Serial LCD
- This function is to ensure that the user is aware of the storage conditions of their medication

The functions we included:

Pill dispensing system:

- Outputs the prescribed pills at a certain time

As noticed here, we tried to simplify the process for the user by keeping the main tasks as basic as possible. The main task is to aid and remind individuals with taking their pills.

Design

There have been multiple challenges the group faced when it comes to the thought process behind the design choice and its practicality. Firstly, the medical pill dispenser had to be able to not only pick pills of varied sizes and weights but also properly place them where a user could easily pick them up after the machine finished its process. Thus, multiple design choices were explored to find the best option.

Most designed would offer an uncomplicated way of dispensing the pills but they were pill specific and would run into problems if a different pill were to be placed in the storage containers, then the previous ones.

The first option the team came up with was a dispensing mechanism that would have containers for different pill sizes. This design would be efficient in practice, but it would inconvenience the user because it would require of them to know the specific pill size and shape every time, they refill the whole build.

As a result, the team produced the final design we have now which is not only able to dispense any pill of any size but also of any shape.

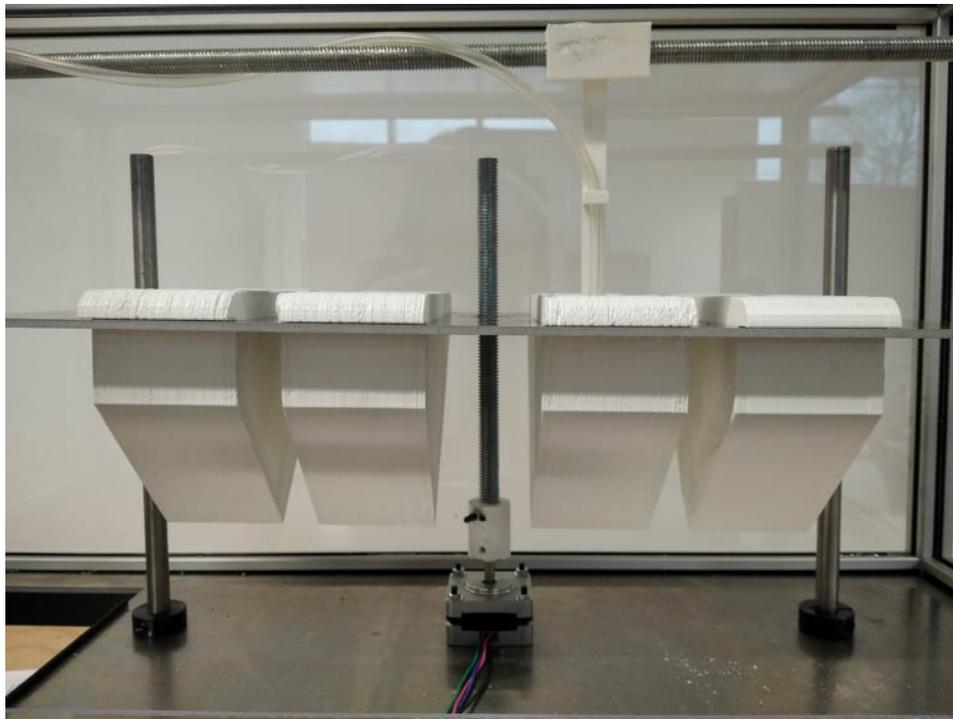


The mechanism is divided into two key parts: The lifting mechanism and the pickup point. When signal is being send to the robot that it must pick up a specific pill, the pickup point is traveling along the X-axis until it reaches right above one of the storage units for the pills. Then, the lifting mechanism that is working on the Y-axis is triggered to bring all storage units up, thus allowing the pickup point to take a pill out of the storage. The lifting mechanism lowers itself afterward with the pickup point returning to its original position and deactivating the vacuum pump, thus allowing the pill to drop into the cup you see above.

Outer Casing: Our first challenge was figuring out how to make the outer casing not only efficient for the user but also for manufacturing. So, we decided to stick with an uncomplicated design that would not only be very efficient mechanically but also look good and simple.

The upper part of the main body is see-through while having a door on the back used by the user to refill the storage units with pills. The lower body contains all the electronics inside of it and is solid, to be able to hold the weight of the top part pressing on it without risking it bending. It also has 3 spots where it is open to the user: One is for the LCD to be kept in place, another is for the user to pick up their pills from the respective cup you see in the picture once the process of dispensing is over, and the last opening is in the back to ensure easy maintenance for the whole build. For the corners we used Aluminium Bosch profiles which can be easily manipulated to replace parts of the casing in case it is required.

Lifting Mechanism: The design of the lifting mechanism is mostly based around the main motor and the plate holding all the storage units.

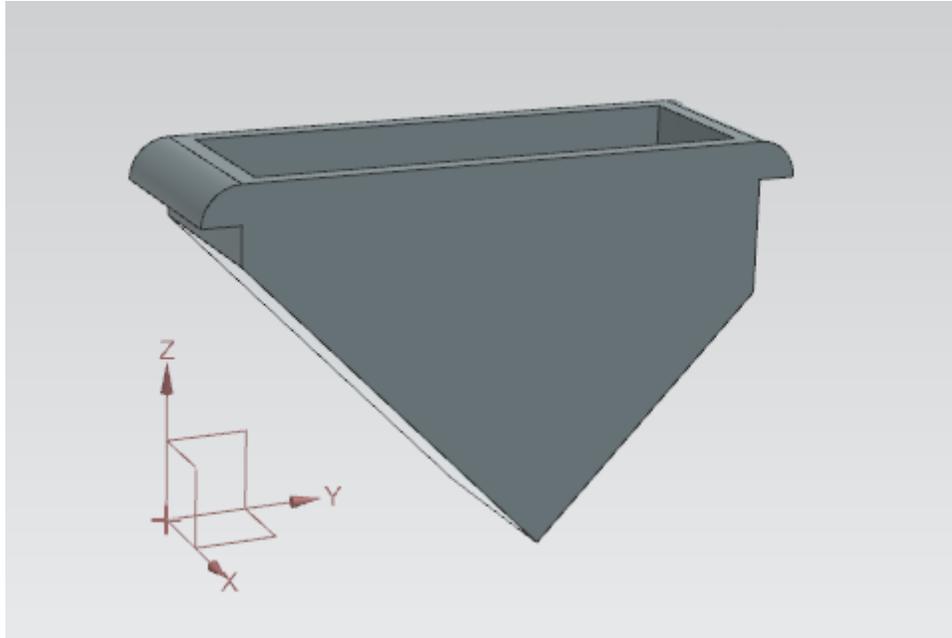


The motor shown in the picture above transfers all its energy through a converter into the threaded shaft connected to it. The plate is then lifted and down based on the rotation of the motor through the threaded shaft.

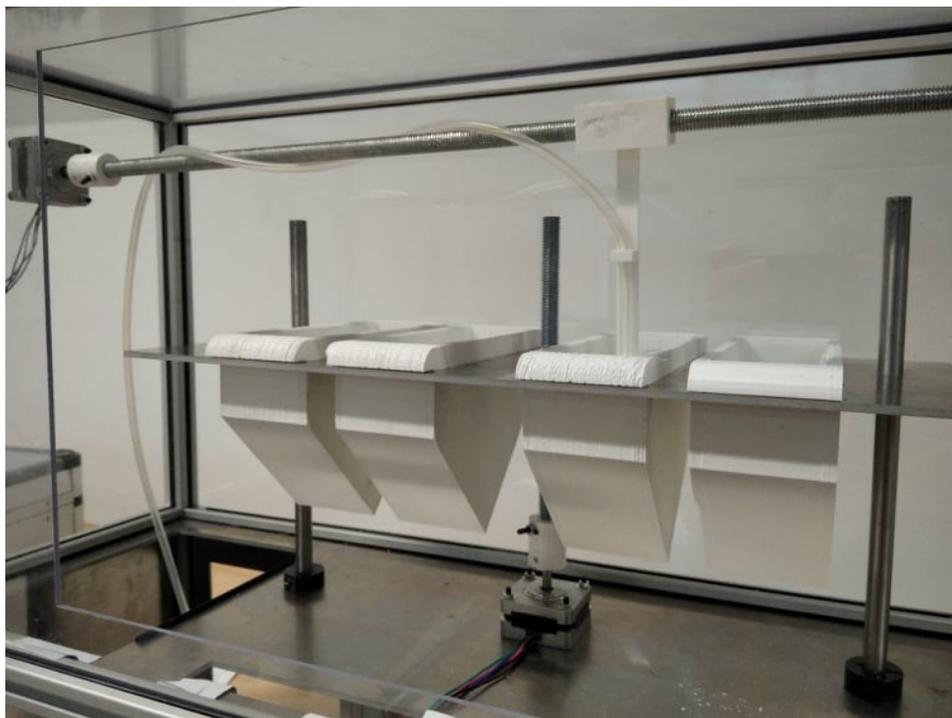
In order to keep the plate balanced and to ensure it is not rotating around the threaded shaft, 2 other non-moving shafts are added on each side. Both these shafts and the threaded rod are aligned on the center line of the plate for the pickup mechanism not to interfere with them. To reduce the friction between the shafts and the plate, bearings or bushes need to be added. The connector is strongly bonded to both the motor and the shaft through screws placed on the side.

Both the motor and the two non-moving shafts are connected to the base plate either through lockrings or through screws.

The storage units for the pills are specifically designed to be non-symmetric so that the pickup mechanism will always be able to reach the bottom no matter what. The containers are not attached to the main plate through screw so that in case a user needs to take one out to wash it can be easily achieved.

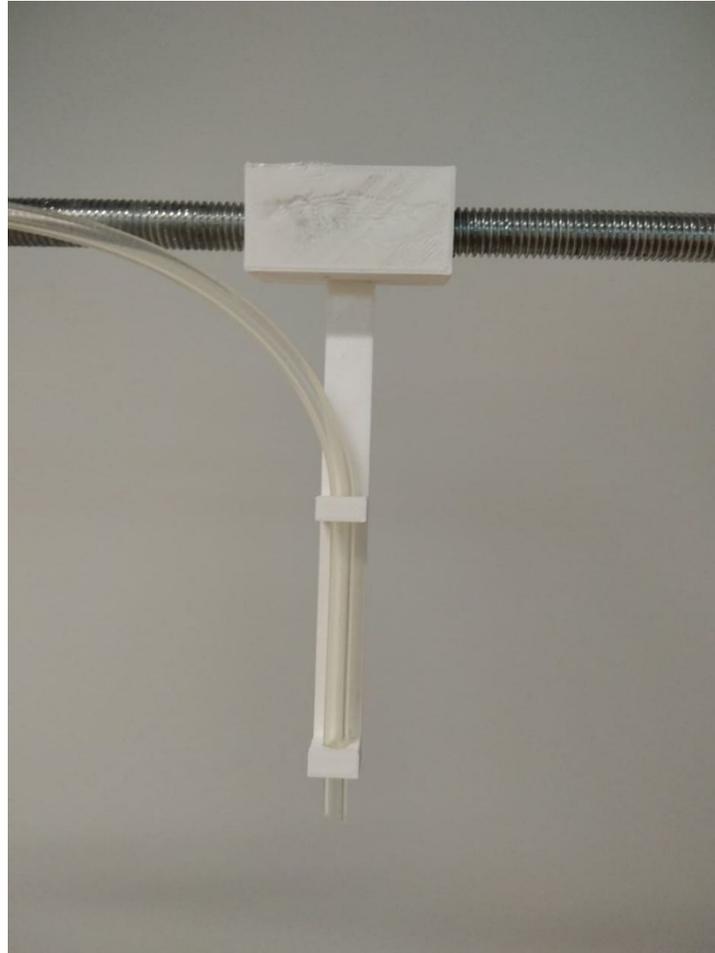


Pickup Mechanism: The pickup mechanism works on the same principle as the lifting mechanism when it comes to its movement.



A motor is connected to the side wall of the whole build. Again, the energy is being transferred to the threaded rod that in this case is used to move from one end to the other the tube of the vacuum pump.

A special holder is designed for the tube of the vacuum pump not only to help the tube travel along the X-axis, but also to move the tube from one end to the other.



There are multiple improvements that can be made for this specific holder like adding to its inside a special threaded bearing that would allow it to move more easily along the rod but also keep it in place more properly.

Dispensing Mechanisms ideation

Considering we went through what the final design is it would also be especially important to talk about the initial concepts and how this design has been achieved. Some information that is discussed here has also been discussed above for context but now we will dive deeper.

Since the main concept is to build a dispenser for pills the biggest problem as stated beforehand was how do we manage to achieve this without caring for pill size and shape. The initial idea that sounded plausible was a tray that will constantly be moved up and down to generate momentum for the pill to be then dropped onto a platform where it could be collected. This idea was quickly discarded since the

constant movement of the tray could generate so much momentum that all the pills will be dropped not just one. As a result this idea was discarded.

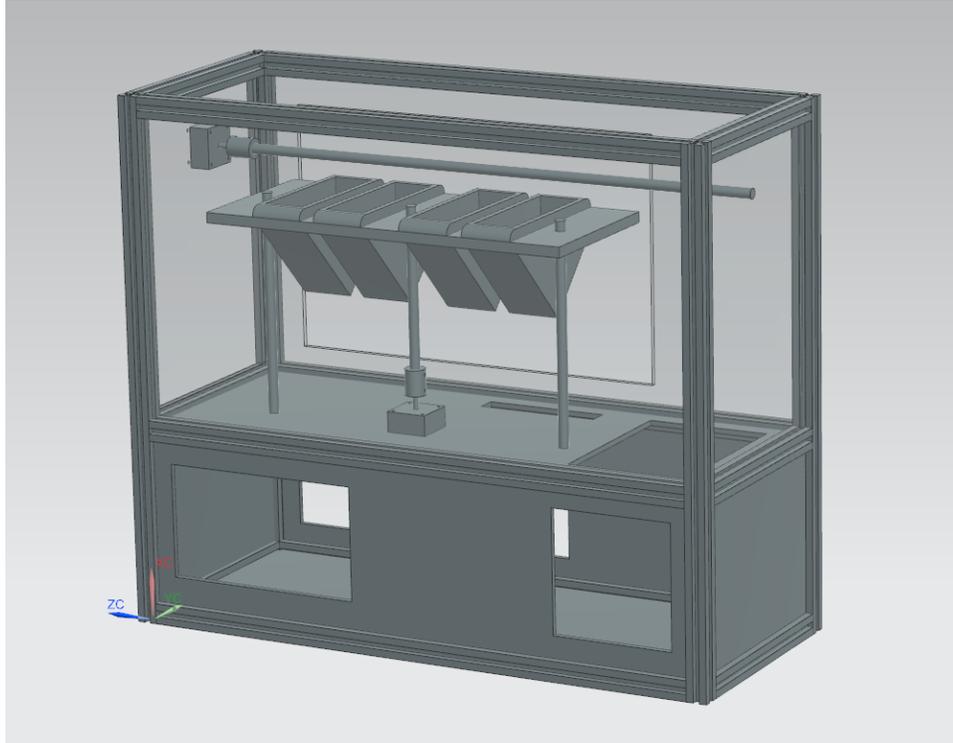
The second idea that came up to the group was regarding a rotating plate that would have on top of it the container for the pills. The container would have an opening that would allow one pill to drop onto the rotating plate. The rotating plate would itself have a whole that would allow the pill to fall. This build is flawed in the sense of the team needing to have a universal way of picking up any pill that would as mentioned before not being reliant on size or shape.

Visual representation of a concept for this idea can be found on this YouTube video:

[Pill Dispensing Mechanism](#)



Thus, the final idea for the main build came up. It was inspired by a 3D printer mechanism when it comes to its movement and picking up system. The concept could also be build exactly like a 3D printer but then the whole space would be taken. The design for the whole build was initially made in NX so that measurement can be made sure to be accurate and so that changes do not require a complete new physical piece redone.



The outer casing as stated has 2 layers, one with Aluminium 3mm plates and one with Lexan 3mm plates. They are held up together by profiles, each one having rubber railings inside it to hold the plates in place. Each hole the plates have is made specifically for either components mounted on the holes or for easy access inside the build. As mentioned, users are not supposed to have access to the bottom back hole in the aluminium plate.

Technology choice

Microcontroller

This would be a major decision which would affect the entire outlook of the project. Whichever microcontroller we choose should be the most suitable for our task as it will be the core around which all functionalities will revolve.

Our overall requirement is that the user should be able to make their selection on an external LCD touch screen. This should be interpreted by the microcontroller and then call functions; be it hardware or software protocols like communicating with the motors or saving data respectively.

Therefore, it will be preferable to choose a microcontroller that can implement hardware communication using a graphical user interface (GUI). As mentioned, we would also like to create a database to save user information. This entails a large storage size, meaning the microcontroller should have enough processing power to handle those files.

Since we plan on using motors and an LCD screen, it would be preferable that the microcontroller can power both those components. Hence the power ratings will also be analyzed. Finally when considering the cost, we have a large enough budget, however we would like to keep the costs under DKK 1000 as some will be dedicated to contingencies.

Having kept these requirements in mind, our research from websites like itfoss¹, zdnet² and electromaker³ yielded 3 options that were the most viable for our project. Below is a table listing the 3 options alongside the requirements that concern us.

Table 9. Microcontroller specifications

	Raspberry Pi 4B (RPI 4B)	BeagleBone Black (B3)	Le Potato (LeP)
CPU & RAM size	1.8 GHz & 8 GB (SDRAM)	1GHz ARM Cortex-A8 & 512 MB (DDRL). 4 GB flash memory.	1.5 GHz ARM Cortex-A53 & 2 GB DDR3-SDRAM. 750 MHz GPU.
Hardware interface	Bluetooth, 2 x micro-HDMI, Ethernet port, 2 x USB 2.0, 2 x USB 3.0, micro SDHC slot, Wi-Fi dual band module	1 x micro-HDMI port, Ethernet port, 1 x USB host, Wi-Fi module, 1 x USB mini	Bluetooth, 4 x USB ports, Ethernet port, Wi-Fi module
External connections	17 x GPIO (16 mA max each), 4 x UART, 4 x I2C connectors. 4 x SPI	46 x I/O pins, 4 x UART, 2 x SPI, 2 x I2C, A/D Converter	40 x I/O pins, alternate pin functions for I2C, SPI, PWM, UART and GPIO
Power ratings	600 mA when idle, 1.25 A under stress. 3 A power supply recommended.	210 – 460 mA at 5 V.	200 mA at 5 V (headless connection)
Cost	DKK 241.05	DKK 351.95	DKK 1040.98

The aftermath of the introduction of the coronavirus has left a shortage in “chips” or processors due to the influx in consumption of devices during these times. Therefore, the availability of the microprocessor should also be taken into consideration.

As visible from the table above, RPi 4B offers the highest processing power and the biggest RAM size. All of them are equipped with Bluetooth and Wi-Fi modules, which was a necessary requirement on our part. The RPi 4B only has 17 GPIO pins unlike B3 with 46 and LeP with 40. This will need to be cross-checked with the connections for the motors and the LCD screen. Since we will be running a GUI application, having a GPU chip would be beneficial, as offered by the LeP. The RPi needs a specific power supply, or else it will not be able to operate at high performance. Yet it is the cheapest of them all. It is also widely used and therefore will not be as rare as the LeP, the latter not available on RS Components.

Thus the obvious choice is the RPi 4B. Even if it is on backorder or takes longer to deliver, its popularity ensures that we will be able to find and prototype on a used one until the new one arrives.

¹ [Raspberry Pi Alternatives](#)

² [ZDNet](#)

³ [electromaker.io](#)

Motors

We created a decision matrix to evaluate which kind of motor would be best suited for our project. We started out by creating different criteria and grading them on a scale of 1-5, with 5 being the best.

	DC motor	Stepper motor	Servo motor
Control	2	5	5
Torque	3	5	2
Cost	5	3	2
Experience	4	3	2
Total	14	16	11

Table 10. - Motor choice decision

For the control criteria: Stepper and servo motors got a perfect rating for control since they have the possibility of calculating the angle of the shaft, thus eliminating the need for an encoder.

For torque criterion: Stepper motors got a perfect rating since they have maximum torque at low speeds, perfect feature for our project as we want to move all the components slowly.

For the cost criterion: Stepper motors are relatively cheap and come with a varying torque value, but they were higher priced compared to DC motors which led to a rating of 3.

For the experience criteria we all discussed our experience working with different motors. The most used was the DC motor since we had to use it for many projects, but closely followed by the Stepper Motor.

We ended up going with a Stepper Motor since it got the highest rating, and it got the best ratings in the most important criteria, control and torque, as we wanted to move all the components with accuracy at low speeds. If we could not find a stepper motor within our budget, then we would have probably gone with a DC motor. This would mean that we would have to include an encoder to measure the shaft angle and maybe a gear system if the motor does not meet the required torque. This reasons also influenced our decision to choose Stepper over DC motors.

Stepper Motor Driver Selection

We were looking for a stepper motor driver for a bipolar stepper motor that can handle a current of 0.4A. We were also looking for a stepper motor driver that can meet the requirements but a driver that had a good shipping time since we didn't have too much time.

Selected Motor and Driver

	Step Motor 39SH20-0404A
Cost (DKK)	469
Torque (Nm)	0.065
Steps per revolution	200
Supply Voltage (V)	2.64
Rated Current (A)	0.4

Table 11 - Motor specifications

The stepper motor we selected for the project is the bipolar Step Motor 39SH20-0404A.

We ended up selecting this Stepper Motor because we were looking for a small motor, with a decent torque, that can fit inside our project since we did not have too much space. There were only 2 motors available in the market, the motor we finally selected and a NEMA 17 pancake. Initially, we went with the Nema 17 Pancake because it was slightly cheaper, but since it was out of stock in all university suppliers, we finally used the Step Motor 39SH20-0404A.

The steps per revolution would dictate the speed of our robot, which was not necessarily vital in meeting the goals. The torque selected is enough to move up all the storage units and move the pump mechanism. We are planning to use 1 motor for moving the components in the Z-axis and other motor for moving the pump in the X-axis. Therefore, we had to order two of these motors which it is a cost of 939 DKK. The selected driver was the PModStep Driver since it met the requirements. As we had some problems with the delivery (it took 1 month to arrive), we ended up using the digital TB6600 stepper motor driver which we had some of those at the university and we also had experience using those drivers. The driver has an input voltage range from 9 to 40V and a current range of 0.5 – 4A, perfect knowing that our motor works at 0.4A, and it is designed to work with bipolar stepper motors. As the driver can only control one motor, we had to use two stepper drivers to control both motors.



Figure 1 - Step Motor 39SH20-0404A



Figure 2 - Motor Driver TB6600

Motor control

Both motors are connected to the TB6600 drivers, using A- A+ B- B+, as the motor is bipolar it only has 4 wires. Both drivers are connected to the power supply (battery or power source), this will feed the motors correctly. The negative pins (DIR- PUL-) are connected to the ground pins on the Raspberry-Pi. The PUL+ pins are connected to where we will send pulses to control the motors. Finally, the DIR+ pins are connected to on the Raspberry-Pi, where we will control the direction of rotation.

Running the motors

The motors are controlled by a code written in Python that is uploaded in the Raspberry-Pi. Initially we define all the pins, and we also create some variables. Each motor has a different code since they have different functionalities that will be executed every time their functions are called.

```
1 import RPi.GPIO as GPIO
2 GPIO.setmode(GPIO.BOARD)
3 from time import sleep
4
5 pinDIR1 = 10
6 pinSTEP1 = 8
7 pinDIR2 = 24
8 pinSTEP2 = 26
9 CW = 1
10 CCW = 0
11 numSteps1 = 800
12 numSteps2 = 0
13
14 GPIO.setup(pinDIR1, GPIO.OUT)
15 GPIO.setup(pinSTEP1, GPIO.OUT)
16 GPIO.setup(pinDIR2, GPIO.OUT)
17 GPIO.setup(pinSTEP2, GPIO.OUT)
```

In the first lines we import GPIO as .BOARD to be able to use the board pin numbering, and we also import “sleep” to set some pauses. Then we define the pins where the drivers are connected. We also create some variables to use them as shortcuts in a future for the direction and the steps. And finally, we define all the pins as Outputs since they are going to send data to the motor driver.

Figure 3 - Motors code I

The rest of the code is divided in two parts. On one hand, there is some code that will be executed, every time we call its correspondent function, for moving the components up and down. On the other hand, there is some code that will be executed, every time we call its correspondent function, for moving the pump mechanism from left to tight or vice versa.

```
19- def 1motor():
20     sleep(1.0)
21     GPIO.output(pinDIR1,CW)
22
23-     for x in range(numSteps1):
24         GPIO.output(pinSTEP1,GPIO.HIGH)
25         sleep(.005)
26         GPIO.output(pinSTEP1,GPIO.LOW)
27         sleep(.005)
28     sleep(60.0)
29     GPIO.output(pinDIR1,CCW)
30
31-     for x in range(numSteps1):
32         GPIO.output(pinSTEP1,GPIO.HIGH)
33         sleep(.005)
34         GPIO.output(pinSTEP1,GPIO.LOW)
35         sleep(.005)
36     sleep(1.0)
```

Here, there is created a function called “1motor” that will be executed every time we call that function for moving the storage units up and then down. In the line 21 we set the direction of the DIR pin as Clockwise (1). Then, knowing that 200 steps mean 1 full rotation and we defined previously numSteps1 to be 800, the motor will do 4 full-turns and the plate will be moved up. The sleep (.005) [line 25] expression means the turning velocity, when lower number the faster it goes. When doing the testing part, we consider this a reasonable velocity. Then the plate stays up for a minute, line 28, (enough time to pick a pill). And finally, we change the direction of turning (0) and it will go down in the same way it has gone up.

Figure 4 - Motors code II

```

50- def 2motor(Steps2):
51     sleep(1.0)
52     GPIO.output(pinDIR2, CW)
53
54-     for x in range(Steps2):
55         GPIO.output(pinSTEP2, GPIO.HIGH)
56         sleep(.005)
57         GPIO.output(pinSTEP2, GPIO.LOW)
58         sleep(.005)
59     sleep(60.0)
60     GPIO.output(pinDIR2, CCW)
61
62-     for x in range(Steps2):
63         GPIO.output(pinSTEP2, GPIO.HIGH)
64         sleep(.005)
65         GPIO.output(pinSTEP2, GPIO.LOW)
66         sleep(.005)
67     sleep(1.0)
68

```

In this part, there is created a function called “2motor”, that receives a value from the main part (Steps2). This value means the number of steps the motor has to do; therefore, it will determine the distance the pump will move along the X-axis. If the pump has to move only until the first storage, this value will be obviously lower than if it has to move until the last container. The rest of the code inside the function is the same as the code for the Motor 1. The only difference is that here we define the distance that must be moved.

Figure 5 - Motors code III

Electrical components

The pump, the microcontroller and the motors require electricity; therefore, the group had to think of a power supply. The first idea was to build a power supply from scratch. However, after talking with Power Electronics course lecturer Bente Olsen, students decided to buy a power supply online and then build voltage converters in order to match the required voltage. This decision was made because of lack of time and knowledge, as a supervisor, and safety precautions would be needed to not disturb the work of the University or prevent being electrocuted in case of an error. After analysing the needs of every electronic component in the project, a table was made:

Component	Voltage Required	Amperage Required
Pump	12 V	0.7 A
Motors, × 2	2×2.64 V	2×0.4 A
Microcontroller	5 V	3 A
Display and sensors	Will be connected to the Raspberry Pi	
Total required:	12 V	4.5 A

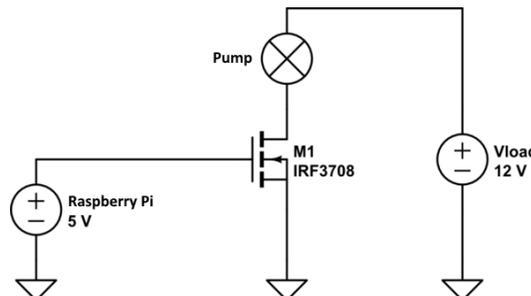
The voltage needed is 12 V as every component is connected in parallel. This means that every component has a 12 V source, while step-down converters lower the voltage if needed. In total, the device requires a 12V, 0.6A power supply for the vacuum pump, two parallel lines of 2.64V, 0.4A for the motors controlling the movement of the hose, which is used to pick up the pills, and a 5V and 3A power supply for the raspberry pi. Since the power adapter which the device is using to convert AC output from a European household power outlet already outputs a Voltage of 12 Volts, it can be directly used to power up the vacuum pump. However, to power the other components, reducing this voltage while also keeping in mind the current requirements is required. The amperage required is 4.5 A, since in parallel circuits every current is summed. To fulfil the needed parameters, a 12 V, 5 A Phihong PPL65U-120 power supply was chosen.

Electric circuits

The main part of the circuits is mostly comprised of utilizing different transistors. The voltage coming from the AC to DC adapter is first split into 4 parallel lines. A 12 V one which can be used to directly power the vacuum pump, as well as the others which are supplied into two different voltage regulator circuits so that supply of 5 V as well as two 2.64 V ones may be acquired. For the power supply of 2.6 Volts and 5V for the Raspberry Pi, LM317 and LM2578 voltage regulators are used respectively.

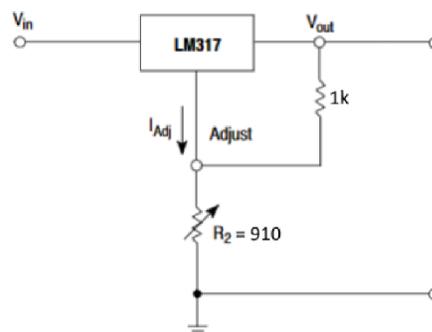
Pump

The vacuum pump already has the needed voltage, therefore there was no need to adjust it. However, if the voltage source is connected to the pump without any switch in-between, the pump is working constantly. To control when the vacuum pump is active, a transistor acting as a switch controlled by a digital output from the Raspberry Pi is used. The gate of the transistor IRF3708 is connected to the output of the Raspberry Pi which is set to either 5 or 0 Volts, the source is connected to ground, and the drain to the vacuum pump which is then also connected to 12V power supply. When the gate voltage is set to high, the transistor passes the 12 volts and when it is set to low, the device is completely off. This way it is possible to control when the pump must be active using the Raspberry Pi microcontroller.



Motors

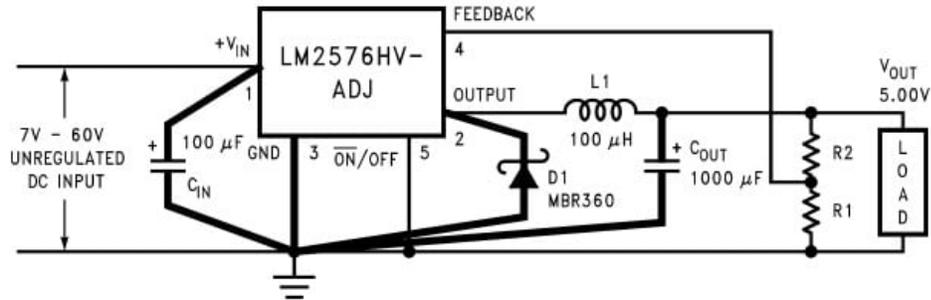
The motors require 2.64 V each. To power them, two voltage supply lines were used. To step-down the voltage, an LM317 linear voltage regulator was used. This voltage regulator is adjustable, therefore proper resistors must be chosen. After experimenting with different rated resistors, it was concluded that 1 k Ω and 910 Ω resistors were required. The circuit can be found below. If the output voltage is not stable, a capacitor can be added to prevent the ripple.



Microcontroller

Raspberry Pi requires the voltage supply of 5 V. To achieve this, the same method of using a step-down voltage regulator is used. However, the challenge here was finding a transistor which can withstand

providing a stable 3A output without frying. After searching through and testing various components, the group found one which satisfies all the requirements. The LM2576 adjustable voltage regulator can provide a stable output of 5V and 3A which is why it was used. Another advantage is that it can be supplied by any voltage between 7 and 60 volts while still outputting the same 5 volts. The picture below displays the circuit which was used.



Material choices

Considering that the build needs to be both stable and solid the choice of materials for most parts was made with these criteria in mind.

Another especially key factor we had to take into consideration was the price we wanted to spend on the materials since we needed to make the project as cost-effective as possible.

For the bottom part of the outer chasing, we went with aluminium since it was both cheap and solid enough to keep the rest of the structure on top of it. The upper part is made from Lexan to allow for both visibility and durability. Acrylic was also an option, but we considered Lexan to be more solid and reliable than Acrylic.

Both the threaded rod and the shaft are made from steel to ensure they will not bend or crack under pressure.

The connectors should normally have also been made out a steel or aluminium but considering the current situation the world finds itself in this could not be achieved in time. The storage units are made of 3D printed PLA.

Thus, most of the criteria we had have been achieved when it comes to the materials we chose.

Graphical User Interface

Language

For graphics rendering, C and C++ are the commonly go-to languages. The compilation of these languages into native machine code outperforms most other languages. This is an important feature in applications that require a high number of frames per second. Moreover, C++ is an object-oriented language, meaning it treats data as objects and classes. Concepts like polymorphism, inheritance and encapsulation give it a slight edge over other languages when it comes to object oriented programming (OOP). When programming in this language, the programmer has total control over memory management as there is no garbage collection. This is because C++ supports dynamic memory allocation (DMA). Furthermore, it is a very commonly used industry language; meaning there is a vast community

around it. The community size is very important, as it is there one can find solutions to most errors encountered.

These arguments raised would point to C++ being the ideal language to build our GUI in. We would need total control of the memory while making databases to store user information; made possible with DMA. Additionally, the inclusion of classes and inheritance gives it a big edge over its compatriot C. Also, being taught the basics of both variations of the language in previous semesters, we felt comfortable in using it.

However, C/C++ are not the easiest to learn. Even though cross-platform GUI libraries are available for C/C++ like QT, it is not often the easiest to learn due to the complex nature of the language itself. This reason, combined with the presence of global variables and pointers makes it an “unsafe” language. Meaning if the part of the memory is an incorrect type, it is possible to corrupt the entire program. Moreover, the language is also very strict regarding the syntax. A missing comma or semicolon will give a series of errors. It is also tough to write the code in a readable way, making it less user-friendly and more rigid.

We learned the basics of QT-programming last semester. Further research also proved that hardware programming through C++ tougher, although possible with minor abstractions. Considering the General Data Protection Regulation (GDPR), one of the seven principles include “Integrity and confidentiality (security)”, the appropriate security measures need to be in place to protect the user data. With this in mind, we couldn’t risk corrupting the data with internal errors.

We wanted to program in a language that can be used to build the GUI, control the motors, and communicate with the app. Using the same programming language throughout would ensure consistency, structured organization and avoid errors caused by confusion through different applications of a language (C/C++ is a static language while python is dynamic).

Having eliminated C/C++, research output suggested Python, Ruby, Java, .NET, and PHP. Even though these languages are dynamic, object oriented and can be used in developing cross-platform applications, we were biased towards Python. The cause of this bias included the immense advantages of learning Python and the impact it will have on automation in the future, its ability to converge hardware programming with a user interface and because it was the most associated language with a Raspberry Pi, the microcontroller of our choice.

Additionally, Python also has huge community platforms, with examples and Q&A platforms; something immensely helpful when programming. Although we were aware that learning a new programming language from scratch while doing a project is not ideal, we were encouraged by the easy-to-read and user-friendly syntax, its similarities to C when it came to hardware programming (GPIO pins instead of DDRD/C command) and that everyone responsible for code development was prepared to accept the challenge as well. Tutorials from unofficial and official YouTube channels like “Codemy” and “edureka!” would also assist enormously.

The programming language to build the GUI will be Python.

GUI Toolkits

While choosing a GUI toolkit, there is no singular correct solution. The various options available can be used to build a reliable GUI. Therefore, we must evaluate based on the factors that are most important to us. Some of our considerations include:

- Licensing requirements
- Storage
- Agility and prototyping speed of the framework
- Learning curve
- User community

PYQT

Before starting to compare these toolkits, it is vital to understand what each of these kits are used for. Even though these are toolkits used for building GUI's, their platform compatibility might be different.

PYQT is one of the most popular bindings for the QT cross-platform C++ framework. PYQT includes classes that cover user interfaces, network communication, threads, SQL databases, XML handling and other such technologies. It is compatible with Windows, Linux, iOS, and Android. This multi-platform compatibility makes this toolkit very attractive to develop cross-platform applications with a native feel on each platform.

In addition to these features, PYQT also offers coding versatility. Programming in QT is based around signals and slots. Meaning, it is possible to create contact between these objects. For example, if a user clicks the `close()`; button, the window's `close()`; function should be called. Other toolkits achieve this communication using pointers to a function; known as a callback. In QT however, every widget has a predefined signal and the response to the signal is called a slot. Although QT's widgets have predefined slots, the norm is to subclass widgets and add user-defined slots so we can handle the signals individually. Furthermore, QT also provides a broad variety of widgets, such as buttons or menus designed with an interface for all compatible platforms. Since it is also a commonly used UI system, there is a large variety of community platforms and documentations to further ease development.

However, amongst its multi-platform compatibility, its clever method of handling callback through signals and slots, there is one major disadvantage. If the application is not open source, a commercial license fee needs to be paid. However, a way around this is if the GPL or LGPL license requirements are met. Both these licenses allow the use of this library for commercial purposes. It means we may keep our source code private. Most commercial software comply with these requirements by dynamically linking to QT.

Tkinter

Tkinter is an open-source library notable for its simplicity. It is pre-installed in Python (in most cases). These characteristics gives it a strong position for beginners and intermediates to begin with. Amongst the numerous advantages it has, the ones impacting us the most were its layered approach, the accessibility it provides and its portability amongst platforms.

The layered approach that is used in designing Tkinter in the first place means the programmer is free to use all the advantages of the TK library. Regarding its accessibility, learning Tkinter is very intuitive and thus can be quick. This is because its implementation hides the detailed and complicated callback in

simple and intuitive methods. Similar to python, this is a continuation of the simplicity of the language to quickly build prototypes. This is a major benefit for beginner or intermediate programmers. Additionally, python scripts that use Tkinter do not require modifications to be ported from one platform to another. It is available on any platform that Python can be implemented on. Moreover, it also provides the native look and feel of the specific platform it runs on.

However, its simplistic approach does have numerous drawbacks as well. It being easy to learn comes at a cost of the execution speed. This may affect older or slower machines as most modern computers are fast enough to cope with the extra processing power it takes. It being too “simplistic” can also be its own shortcoming. This feature can prevent its use to design complex functions and use it across numerous devices all connected via a database. Although, it is an open-source application which is very beneficial to freelance programmers and advantageous when testing new toolkits.

Kivy

Similar to Tkinter, Kivy is an opensource multi-platform library for Python. It can run on iOS, Windows, Android, and GNU/Linux. It is used to develop applications that implement innovative and multi-touch UI. Kivy allows the developer to design and build an app once and use it across all devices. This allows the code to be reusable and deployable, enabling quick and easy interaction design and rapid prototyping.

If using Kivy, it is also possible to write code once and use it across all platforms. Also similar to the other 2 toolkits, Kivy offers easy to use widgets built with multi-touch support. Furthermore, it also provides extensive input support for input devices such as mouse, keyboard, TUIO and OS-specific multi-touch events. Unlike Tkinter however, this toolkit will allow us to create complex applications using numerous algorithms as the use of functions, regular expressions, etc. is possible. Kivy is also built around us creating a “Natural User Interface” or NUI. The idea behind this concept being that the user of this UI can easily learn to use the application with little to no instructions. Also, unlike Tkinter, Kivy does not attempt to use native controls or widgets, meaning all of its widgets are custom made. Consequently, this means Kivy applications will look the same across all platforms.

One major disadvantage of using Kivy for programmers learning to code in Python is that Kivy in itself has a design language called “kv”. This kvlng, as it is called, allows the programmer to create a widget tree in a declarative manner. This in turn helps bind the widget properties to each other or to callback in a natural manner. Fast prototyping and agile changes to the UI are possible with this.

It is not required to learn the entire kvlng to program in Kivy; however, if we do not make use of the array of functionalities provided to us by kvlng, our GUI may become too reliant on the example widgets already provided. Yet learning two new languages from scratch while also managing other tasks can prove to be quite tricky.

Since we are using Raspberry Pi and coding in Python not only has its benefits in the present, it will also be a valuable tool to possess in the future. Therefore, we will develop our GUI in Tkinter.

Clock & Home screen

On the GUI home screen, our aim was to make it look friendly, homely, and inviting. But simple and efficient at the same time. Because as mentioned earlier, we aimed this device mainly for older individuals, meaning simplifying the interface would be ideal to only keep the basics visible.

Time in general is a feature we thought would be useful in a home screen interface, hence we added it in the top of the screen. That includes the time in hours, minutes and seconds; as well as showing the time zone just below it.

Saturday 11:23:33 PM

Romance Standard Time Saturday

How did we build it?

First step was to import time. Time is just a python universal library and from there we can use it to access elements in relation with time.

```
import time
```

We create a new function called clock, and here we created a few variables we will be using later on in the code. We have to use the function called strftime which allows us to use different aspects of time.

```
def clock():  
    hour = time.strftime("%I")  
    minute = time.strftime("%M")  
    second = time.strftime("%S")  
    day = time.strftime("%A")  
    am_pm = time.strftime("%p")  
    timezone = time.strftime("%Z")
```

In the strftime function, we input the directive we are seeking. We used the following directives with the subsequent meanings:

%I → 12-hour clock as a decimal number

%M → minute as a decimal number

%S → second as a decimal number

%A → weekday name

%p → time in either AM or PM

%Z → time zone offset indicating a positive or negative time difference from UTC/GMT

These directives above are indeed case sensitive, meaning lower case letter means something different than upper case letter. Which we can see clearly with %p, as it's the only directive in lower case style.

Outside the function clock, we created two labels, one to display: day, hour, minute, second. And the other to display the time zone:

```

my_label = Label(root, text = "", font = ("Helvetica", 48), fg = "black")
my_label.pack(pady = 20)

my_label2 = Label(root, text = "", font = ("Helvetica", 14))
my_label2.pack(pady = 10)

```

Here we also changed the font to Helvetica and foreground colour to black. As well as increasing the font size of time, to make it easier to read.

Then back in the clock function, we print out the labels. We set the text to equal the variables we set up earlier, and to make it simpler, we can concatenate this as well.

We also use the function “after” to update our label after a certain period of time, since we are building a clock here, we update it every minute. So we have the first variable being 1000 milliseconds, and the second variable being the function clock.

```

#printing out the time
my_label.config(text = day + " " + hour + ":" + minute + ":" + second + " " + am_pm)
my_label.after(1000, clock) #updating after a certain time, 1000ms --> 1s

my_label2.config(text = timezone + " " + day)

```

So, what happens is that we call this function clock, it goes through the code, it reaches the function “after”, then every second it will run the whole function again, updating the time every second.

Keyboard

Something we also built in tkinter python was the keyboard. When a user goes to create their profile in the new user screen, they get the option to type their name in there. To do that we created a window with keyboard buttons on there.



Since we the keyboard would be displayed in front of another window, we thought it would be best if we made the keyboard transparent to a certain degree. Which we did using the following:

```

key.attributes("-alpha", 0.7)

```

Here we use the attribute alpha to change the transparency of the window. After trial and error, we decided that the opacity value of 0.7 would be the best value for alpha, because the transparency was perfect.

In regards to creating the buttons themselves, an example of a few can be seen here:

```
q = ttk.Button(key,text = 'Q' , width = 6, command = Lambda : press('Q'))
q.grid(row = 1 , column = 0, ipadx = 6 , ipady = 10)

w = ttk.Button(key,text = 'W' , width = 6, command = Lambda : press('W'))
w.grid(row = 1 , column = 1, ipadx = 6 , ipady = 10)

E = ttk.Button(key,text = 'E' , width = 6, command = Lambda : press('E'))
E.grid(row = 1 , column = 2, ipadx = 6 , ipady = 10)
```

We created a tkinter button for each keyboard button we needed. Using the grid geometry manager, we specified directly where the keys themselves would be located. And for the keys to actually work as buttons, as in when clicked they type, we created a function called “press”, and a global variable exp, when clicking a button, the key is stored in that variable, and converted and stored as a string. We also used the .set() method to set and change the stored values with in a tkinter variable.

```
exp = " " # global variable
# showing all data in display

def press(num):
    global exp
    exp=exp + str(num)
    equation.set(exp)
# end
```

Then for the enter and clear buttons, we created these functions to utilize them accordingly:

```
# function clear button

def clear():
    global exp
    exp = " "
    equation.set(exp)

# end
```

```
# Enter Button Work Next line Function

def action():
    exp = " Next Line : "
    equation.set(exp)

# end function coding
```

To place the buttons in the correct location, we also used the .grid() method to set the row and column of each button properly.

Button Q vs Button M:

```
q = ttk.Button(key,text = 'Q' , width = 6, command = Lambda : press('Q'))
q.grid(row = 1 , column = 0, ipadx = 6 , ipady = 10)
```

```
M = ttk.Button(key,text = 'M' , width = 6, command = Lambda : press('M'))
M.grid(row = 3 , column = 6, ipadx = 6 , ipady = 10)
```

Window Organisation

Since we have many different windows that play part in this GUI, organisation is crucial to keep track of how the windows are supposed to work together, and for the GUI to appear neat and friendly to the user.

To make the windows open on each other, and not scatter away when opened, we used the `overrideredirect()` function and set its flag to `True`.

```
root.overrideredirect(True)
```

Some windows that we have designed are set to keep being on top, when opened. This is done for the keyboard window for example. By using `-topmost` option in `.attributes()`, we make the window always be on top of all the other windows.

```
key.attributes('-topmost', 1)
```

Zero is considered false, hence we put 1 in there, as any other non-zero number means true.

To create different windows, and organize them accordingly, we used the `toplevel()` function. The first window we created was the root window and it showed a big window on the screen.

```
root = Tk()
```

To create a different window, we first have to define it, and set it equal to `Toplevel()`, if we run that we create a new window.

```
top = Toplevel()
```

Now anything we want to include, design or built on that new window, we have to do after that line.

```
usr1btn = Button(root, text="Hammond", font = myFont, width=10, height=5)
```

```
L1 = Label(top, text="Type").pack()
```

As we can see here, in button `usr1btn`, we insert `root` as the window we want it to appear on, while on label `L1`, we inserted `top`. This is a way to designate different windows that way.

Database selection

When it came to creating a database to store the user’s information, we had initially used SQLite. We had written the code and got it to work, but after conducting more research we realized that MySQL was a better option. Both have their advantages and disadvantages but when looking at our project it was clear that MySQL was the better option.

Table 1312. Database comparison

Comparison between SQLite and MySQL⁴	
MySQL	SQLite
Requires a database server to interact with the client over the network	Serverless embedded database that runs as part of the application and cannot connect with any other application over the network
Can handle multiple connections simultaneously	Can handle only one connection at a time
Highly scalable and can handle large volume of data very efficiently	Can handle only small set of data if the volume of data increased its performance degrades
Requires large space in the memory for its functioning (~ 600 Mb)	Requires some Kbs of space – very lightweight (~ 250Kb – 300 Kb)
Supports multiple user environments	Does not support multiple user environments
Can create multiple users with different levels of permissions and roles	Does not have this feature
Offers and supports many authentication methods to protect the unauthorized access of the database. Includes basic username and password protections to advance SSH authentication	Does not have any inbuilt authentication technique and the database files can be accessed by anyone. They can also read and update the data as well
Setting up the MySQL server requires many server configurations	Does not need any configuration and getting it up and running is very easy compared to MySQL
Has its own API	Does not have its own API

After considering the reasons listed in table 10 above, we decided to use the MySQL database. The main reasons were because it has its own API, is highly scalable, can create multiple users with different levels of permissions and offers better security (from basic username to passwords to SSH authentication). The only benefit we could see by using SQLite was that it was very lightweight (~ 250 Kb) and that the server configuration is much easier (do not need to set one up).

Table 1413. Database user_info

Variable name	What it stores
name_entry	User’s name
p1_entry	Total amount of pill 1 they want during the day
p2_entry	Total amount of pill 2 they want during the day
p3_entry	Total amount of pill 3 they want during the day
p4_entry	Total amount of pill 4 they want during the day

⁴ “Difference between MySQL and SQLite.” GeeksforGeeks, 7 May 2020, www.geeksforgeeks.org/difference-between-mysql-and-sqlite/.

p5_entry	Total amount of pill 5 they want during the day
p1_morning	Amount of pill 1 user wants in the morning
p1_afternoon	Amount of pill 1 user wants in the afternoon
p1_evening	Amount of pill 1 user wants in the evening
p2_morning	Amount of pill 2 user wants in the morning
p2_afternoon	Amount of pill 2 user wants in the afternoon
p2_evening	Amount of pill 2 user wants in the evening
p3_morning	Amount of pill 3 user wants in the morning
p3_afternoon	Amount of pill 3 user wants in the afternoon
p3_evening	Amount of pill 3 user wants in the evening
p4_morning	Amount of pill 4 user wants in the morning
p4_afternoon	Amount of pill 4 user wants in the afternoon
p4_evening	Amount of pill 4 user wants in the evening
p5_morning	Amount of pill 5 user wants in the morning
p5_afternoon	Amount of pill 5 user wants in the afternoon
p5_evening	Amount of pill 5 user wants in the evening
user_id	Unique id given to each user

The information that the database stores is the user's name, what pills they take, when they would like to take them and the number of pills they would like to take at certain times. The other database stores information about what pills are in the dispenser, the amount of each pill, and the time they want the pills to be dispensed. This can be seen more clearly in the tables 11 and 12.

Table 1514. Database local info

Variable name	What it stores
P1_name	Name of pill 1
P2_name	Name of pill 2
P3_name	Name of pill 3
P4_name	Name of pill 4
P5_name	Name of pill 5
P1_total_amount	Total amount of pill 1 in dispenser
P2_total_amount	Total amount of pill 2 in dispenser
P3_total_amount	Total amount of pill 3 in dispenser
P4_total_amount	Total amount of pill 4 in dispenser
P5_total_amount	Total amount of pill 5 in dispenser
P1_morning_time	Time they want pill 1 dispensed in morning
P1_afternoon_time	Time they want pill 1 dispensed in afternoon
P1_evening_time	Time they want pill 1 dispensed in evening
P2_morning_time	Time they want pill 2 dispensed in morning
P2_afternoon_time	Time they want pill 2 dispensed in afternoon
P2_evening_time	Time they want pill 2 dispensed in evening
P3_morning_time	Time they want pill 3 dispensed in morning
P3_afternoon_time	Time they want pill 3 dispensed in afternoon
P3_evening_time	Time they want pill 3 dispensed in evening
P4_morning_time	Time they want pill 4 dispensed in morning
P4_afternoon_time	Time they want pill 4 dispensed in afternoon
P4_evening_time	Time they want pill 4 dispensed in evening
P5_morning_time	Time they want pill 5 dispensed in morning
P5_afternoon_time	Time they want pill 5 dispensed in afternoon
P5_evening_time	Time they want pill 5 dispensed in evening
User_id	Unique id

Layout and Formatting

Review user information

User: Brody

Pills	Morning	Afternoon	Evening	Total
<i>Allocate</i>	3	1	1	5
<i>Brigade</i>	1	0	0	1
<i>Sherline</i>	0	2	0	2
<i>Caddy</i>	1	0	2	3
<i>Pristine</i>	0	2	0	2

Buttons: Edit user information, Save user information

Figure 7. A typical window in the GUI

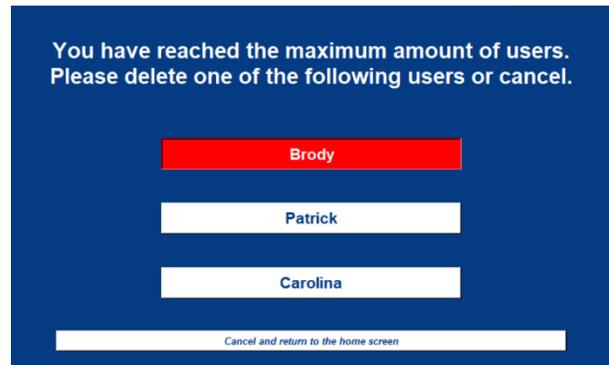


Figure 6. "Delete user" window

The G (graphics) in GUI is determined by the layout and formatting of each window. The GUI in the in-built display for this device was made in such a way that the “language” of the device is simple and quick to understand. The chosen color palette was based on the Abena logos. There are only four colors in this GUI – ‘Abena blue’, white, ‘Abena green’ and red. The primary color of the GUI is blue, and the complimentary color is white. Green is used for the user profile buttons and as an “active”⁵ color when clicking on most buttons. Red is used as an “active” color when clicking on the buttons that lead to deletion of the specified user. Most of the font is bold and some of the font is in italic to differentiate titles and subheadings from inputted values (such as the pill names).

Container Setup

Enter the name of the pills you have placed in container 1

Enter the number of pills in the container

Enter the name of the pills you have placed in container 2

Enter the number of pills in the container

Enter the name of the pills you have placed in container 3

Enter the number of pills in the container

Enter the name of the pills you have placed in container 4

Enter the number of pills in the container

Enter the name of the pills you have placed in container 5

Enter the number of pills in the container

Button: Save information

Figure 9. Frame with default border

Container Setup

Enter the name of the pills you have placed in container 1

Enter the number of pills in the container

Enter the name of the pills you have placed in container 2

Enter the number of pills in the container

Enter the name of the pills you have placed in container 3

Enter the number of pills in the container

Enter the name of the pills you have placed in container 4

Enter the number of pills in the container

Enter the name of the pills you have placed in container 5

Enter the number of pills in the container

Button: Save information

Figure 8. Frame with borderwidth=0

Firstly, it was helpful to divide each window in different parts. This was done with using the LabelFrame method. The syntax of a frame is as follows:

`*name* = LabelFrame (*master*, *options*)`

The name is the name given to the frame widget.

Master is the parent widget/window.

The options used in the code are in the table below.

⁵ explained later

Table 1615. Options used for Label Frame

Option used	Default attribute	Possible attributes (if relevant)	Description
bg		various colour names (e.g. white), hex code (erg. #043b82)	determines the background colour of the widget
borderwidth	2		determines the width of the border (0 was used to make the frame “invisible”)

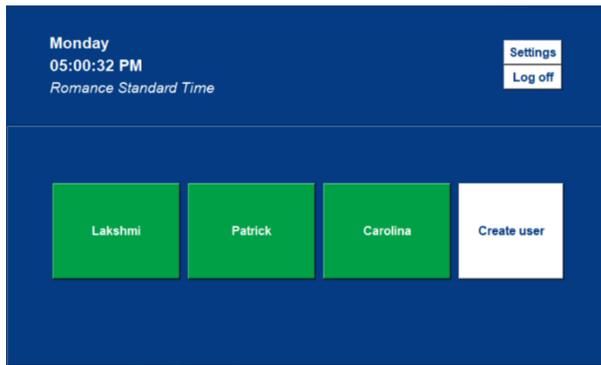


Figure 11. Grid method (frame border added for clarity)

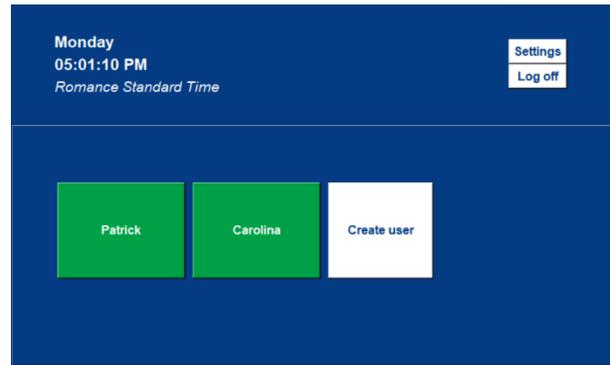


Figure 10. Grid method (frame border added for clarity)

There are multiple methods available to structure widgets onto the screen – pack, place, and grid. Pack was most commonly used as it was easy to arrange the widgets in a specific order and the fill and expand options made it simple to extend the widget in specific directions. Grid was used in the keyboard since there were a lot of buttons and on the home screen. The advantage of using the grid function, is that if there is a widget missing (for example, when only 2 users are created, as seen in Figure 10 and Figure 11), it adjusts the widgets accordingly, without compromising the intended layout of the widgets. Place was avoided as much as possible, as it required us to calculate the precise number of pixels if using the options x and y. However, when using anchor, relx and rely, the place() method was particularly helpful since it allowed us to overlay widgets and reduce the overall space occupied by a group of widgets (as seen in Figure 12). In Table 1716 below, the methods are explained in further detail.

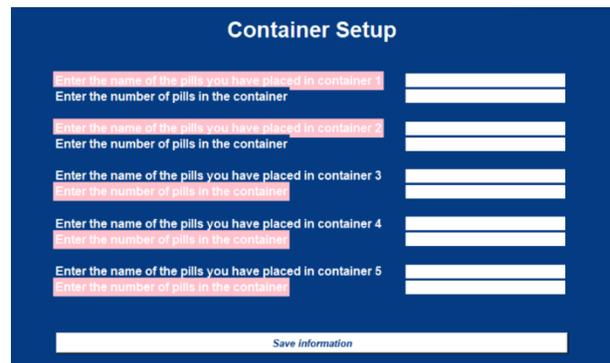


Figure 12. Place method allowing overlay (pink added on some labels for clarity)

Table 1716. Options used for pack(), grid() and place() explained

Method	Options used	Default attribute	Possible attributes (if relevant)	Description
pack()	side	TOP	TOP, BOTTOM, LEFT, RIGHT	determines which side of the parent to attach this widget to
	fill	NONE	NONE, X (only horizontal), Y (only vertical), BOTH	determines whether the widget must take up extra space
	expand	False	True, False	when set to 'True', the widget takes up extra space (fill determines whether it takes up space horizontally, vertically or in both directions)
grid()	row, column	0		row or column to put widget in (starts from 0)
	rowspan, colspan	1		how many rows or columns the widget occupies
	ipadx, ipady	0		number of pixels used to pad the widget on the inside
	padx, pady ⁶	0		number of pixels used to pad the widget on the outside
place()	anchor	NW	N, S, E, W, NE, NW, SE, SW	determines which corner/side the widget refers to for the other options
	relx, rely		0.0 - 1.0	determines how much to offset the widget by, as a fraction of the width and height of the parent

⁶ note padx and pady were used in many elements, not only in the place() method

In Table 1817 below, there is a list of options used for formatting in the code, that were not mentioned above.

Table 1817. Additional options used for formatting

Option	Possible attributes (if relevant)	Description
text		used to display text on top of the widget
font	<i>font family, point size, style modifiers (e.g. ("Helvetica", "22", "bold"))</i>	used to display the stylized text on the screen
fg	<i>various color names (e.g. red), hex code (e.g. #00a047)</i>	determines the foreground color of the widget
activeforeground		foreground color of the widget when it is clicked on ("active")
activebackground		background color of the widget when it is clicked on ("active")
width		sets the width of the widget
height		sets the height of the widget

Moreover, in certain windows, there were some practical reasons why the layout could not be optimal. The in-built display of the device is 800x480 pixels so that is the maximum window size. For example, in the first window that appears when clicking on "Create User" (let's call it "timeslot setup"). In this window, there are option menus that have ten options. Since there are five possible pills, there needs to be space for at least 14 options vertically, excluding titles and subheadings. So, compromises needed to be made, in order to accommodate all the available options. Therefore, the timeslot setup window does not look particularly aesthetically pleasing as seen in the figure.

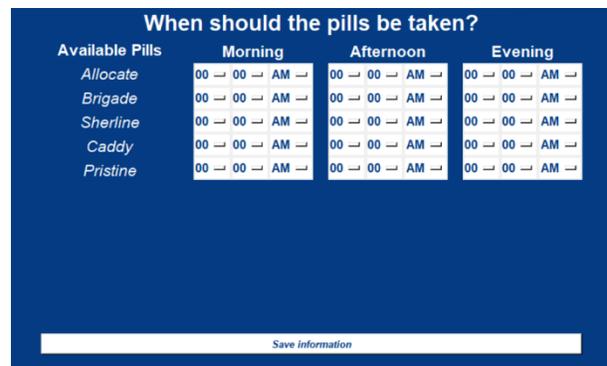


Figure 13. Timeslot Setup window

Configure was used in certain places to modify attributes of a widget after initialization (such as window background colors and option menus).

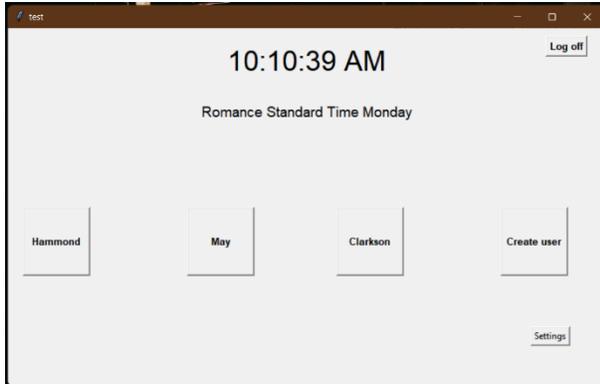


Figure 14. Initial version of the GUI without formatting

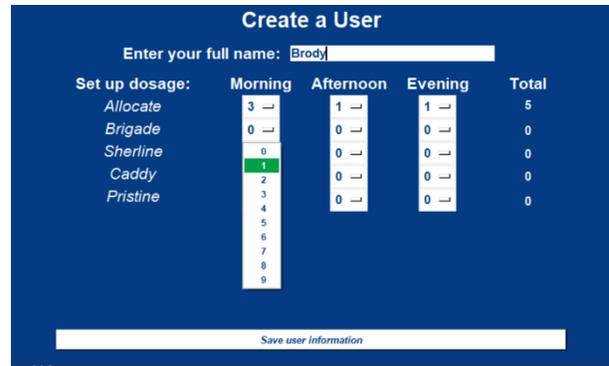


Figure 15. Option menus

In hindsight, the “language” of this GUI is simple, easy, and quickly understandable. In the pictures⁷ above, an overview of the “look overhaul” of the GUI is shown. One error I made is that I did not properly study the `ttk.Style()` method because the code would have had lesser lines if I used it. It would also allow for much, much easier style changes, whereas now, I would have to manually each and every widget and window configuration and more. Just to illustrate this point further, Figure 16 below is an example of formatting one option menu’s button and dropdown menu.

```
p5_afternoon_ampm_ddm=OptionMenu(noonAMPM_frame,p5_afternoon_ampm,*time_ampm_list)
p5_afternoon_ampm_ddm.config(borderwidth=0,width=2,font=("Helvetica","12","bold"),fg="#043b82",bg="white")
p5_afternoon_ampm_ddm["menu"].config(borderwidth=0,font=("Helvetica","9","bold"),fg="#043b82",bg="white",\
activeforeground="white",activebackground="#00a047")
p5_afternoon_ampm_ddm.pack()
```

Figure 16. One option menu's style configuration

Phone Application

The phone app has been created on MIT App Inventor; an online software designed exclusively for application design. It has many screens, that will be explained below, where you can insert, edit and delete some data, and also connect it to the Raspberry-Pi to send all the information.

The software has two different options when creating the app. First, it has a “Design” screen where you can modify all the components that will be viewed as buttons, labels, pictures... this is like the Hardware part of a PC. Then, it has an “Embedded” screen where you insert the code to program the app. For coding, it uses blocks instead of text and lines, to make the software more accessible and user-friendly to the users. Some examples of the block used will be attached in the Appendix.

⁷ all pictures of the GUI will be attached in Appendix for clearer viewing

So, the steps to follow when creating an app in MIT App Inventor are:

1. Edit the user interface, adding all the components that you want to be visible by the user.
2. Program all the visible components in the screen, as buttons, links, databases, passwords...

There is space for 3 users in the app, there is no relation between them. Each one is a unique user inside the system.

All the data is saved in variables in the same way as it is done in the GUI, this means that the app deals with around 160 variables.

There is implemented a Database in the app that saves all the data and variables used and stored in the app: usernames, dispensing time, amount, names... This has been done so as not to have to enter all the values whenever the application is started. So, no matter if the app is closed or minimized, all the values are saved in the database. It is needed to say that the database is a completely different database than the one using in the Raspberry Pi.

Now, all the different screens and functionalities will be explained:

INITIAL PASSWORD SCREEN: the app will request the user to type a password in order to go inside the system. This will make the app only usable by the registered users.

MAIN SCREEN: in this screen you will see all the users registered in the system. It is also possible to create a new user, as well as edit some information or even adjust some settings about the pills.

PILL SETTINGS SCREEN: the app gives the opportunity to enter the name of the pills that are going to be dispensed, as well as the amount of each different pill.

NEW USER SCREEN: if you click on the new user button in the main screen you will be redirected here. When creating a user, you will need to type your name (otherwise you are not going to be able to create a user), and the time you want each pill to be dispensed. The app has the feature of distinguishing between morning, afternoon, and evening. Then it will request you the amount of each pill to be dispensed.

NEW USER CREATED: when a new user is created 3 new buttons appear to know the status of the user:

- **INFORMATION ABOUT USER:** if you press the Information button, a new screen will be opened. Here, you will see a schematic that contains your name, name of the pills, dispensing time as well as the amount. At the bottom, the app has the functionality to connect a Bluetooth device, in this case the Raspberry Pi will be connected. So, when both devices are connected, you will be able to send all user information (including all the things mentioned before) to the Raspberry Pi by pressing the "Send all Info" button. All the data will be stored in the Raspberry Pi.
- **EDIT USER SCREEN:** if this button is pressed, the new user screen will be opened again. Here, you will be able to edit anything that has to be changed. Since the app has a Database, it will remember all the names, times, numbers, etc. This is more user-friendly since the user will only need to change the desired value, instead of defining everything again.
- **DELETE BUTTON:** if this button is pressed, means that a user wants to be removed from the system. Then, it will appear an alert to make sure that it was not a mistake. If the "Yes" button is

pressed, all the data related to that user will be removed from the app as well as from the Database.

All the different screens are attached to the Appendix.

Conclusion

The first goal of the project was to draft up an idea for a useful and necessary product in the healthcare sector with integrated sensors and intelligent reading that the partnered company Abena would be able to implement in the future.

This is followed by the development of a product prototype, which was both fulfilled within the intended time frame.

The company representative from Abena was not able to advise much when it came to the creation of the prototype and technical details since the team chose to go with the development of a new product rather than improving upon an already existing one implemented by the company.

Due to the current situation with covid and deliveries of materials a lot of ideas regarding the mechanical part of the project had to be delayed or cancelled completely. This is mostly visible in the pickup point part of the machine or the lifting mechanism since certain parts would take too long to deliver or in experience from earlier orders throughout the semester kept postponing delivery times.

After those unexpected bottlenecks and delays with component deliveries the first version of a physical prototype was able to be assembled and as a concept mechanically achieves its goal.

The missing parts mentioned earlier are not strictly vital to the function of a prototype, but were hard to work around after settling on a set-up and counting on ordered components to arrive.

Their timely arrival would have improved the pill dispenser in its functionality, letting the team transform it into the better-rounded product that was originally planned.

The different subsystems of lifting mechanism, pump-assisted pill pick up and the code for interactive display and app function correctly, whereas the system assembled as a whole has not been able to be successfully tested in regard to functionality, solely the fit of the real components has been able to be tested and adjusted until this point.

In conclusion, after a semester's work the project demonstrates the functionalities of the smart pill dispenser that was developed together and the team is confident that the product can be further developed into a valuable item Abena will be able to distribute and continue on their mission to help people.

Bibliography

<https://ico.org.uk/for-organisations/guide-to-data-protection/guide-to-the-general-data-protection-regulation-gdpr/principles/>
<https://maker.pro/custom/tutorial/which-programming-language-should-i-choose-graphics-and-guis>
<https://www.quora.com/Could-I-use-C-for-hardware-programming>
<https://www.quora.com/What-can-Python-do-that-C-can-t>
<https://www.decipherzone.com/blog-detail/top-programming-languages-for-desktop-apps-in-2021>
<https://www.youtube.com/user/edurekaIN>
<https://www.youtube.com/channel/UCFB0dxMudkws1q8w5NJEAmw>
<https://realpython.com/python-pyqt-gui-calculator/>
<https://doc.qt.io/archives/qt-4.8/signalsandslots.html>
<https://www.gnu.org/philosophy/selling.en.html>
<https://www.geeksforgeeks.org/what-is-kivy/>
<https://itsfoss.com/raspberry-pi-alternatives/>
<https://www.geeksforgeeks.org/difference-between-mysql-and-sqlite/>
<https://hevodata.com/learn/sqlite-vs-mysql/>
<https://www.educba.com/mysql-vs-sqlite/>
https://www.youtube.com/watch?v=ruohUTTo8Kw&ab_channel=Codemy.com
<https://docs.python.org/3/library/time.html>
<https://stackoverflow.com/questions/18394597/is-there-a-way-to-create-transparent-windows-with-tkinter>
<https://masterprograming.com/how-to-create-virtual-onscreen-keyboard-using-python-and-tkinter/>
<https://www.quora.com/What-does-the-set-variable-do-in-Python-3-Tkinter>
<https://stackoverflow.com/questions/65046290/what-does-the-one-in-tk-wm-attributes-topmost-1-mean>
[Create New Windows in tKinter - Python Tkinter GUI Tutorial #14](#)



<https://microcontrollerslab.com/lm2576-step-down-voltage-regulator/>

<https://electronics.stackexchange.com/questions/334454/mosfet-switch-not-working-as-expected>
ON Semiconductor, "1.5 A Adjustable Output, Positive Voltage Regulator", LM317 datasheet, Jan. 2002

<https://youtube.com/playlist?list=PLCC34OHNcOtoC6GglhF3ncJ5rLwQrLGnV>

https://www.tutorialspoint.com/python/tk_pack.htm

https://www.tutorialspoint.com/python/tk_grid.htm

https://www.tutorialspoint.com/python/tk_place.htm

https://www.tutorialspoint.com/python/tk_labelframe.htm

https://www.tutorialspoint.com/python/tk_colors.htm

https://www.tutorialspoint.com/python/tk_fonts.htm

<https://www.delftstack.com/howto/python-tkinter/how-to-set-font-of-tkinter-text-widget/>

<https://www.geeksforgeeks.org/python-add-style-to-tkinter-button/>

<https://stackoverflow.com/questions/42511123/utility-of-config-in-python-tkinter/42511195>

<https://www.pythontutorial.net/tkinter/tkinter-window/>

Appendix



PASSWORD :

NEXT

Wrong password!

Figure 17-Password screen

Enter your full name:

Amoxil	MORNING	AFTERNOON	EVENING
Messil	MORNING	AFTERNOON	EVENING
Antiviral	MORNING	AFTERNOON	EVENING

GO BACK **SAVE**

Figure 22 - New User Screen I

Name of pill:

Amount of pills:

Added succesfully! **GO BACK** **ADD** Now you can go back

Figure 19-Pills settings screen

Write the amount to take:

	MORNING	AFTERNOON	EVENING
Amoxil	<input type="text" value="1"/>	<input type="text" value="2"/>	<input type="text" value="0"/>
Messil	<input type="text" value="1"/>	<input type="text" value="0"/>	<input type="text" value="10"/>
Antiviral	<input type="text" value="2"/>	<input type="text" value="5"/>	<input type="text" value="1"/>

!! If no pill is wanted write 0 !!

GO BACK **SAVE USER INFORMATION**

Figure 21 - New User Screen II

USER SCREEN

Alberto **INFORMATION** **EDIT** **DELETE**

Are you sure you want to delete all user information?

YES **NO**

PILL SETTINGS

NEW USER **NEW USER**

Figure 18 - Main Screen

Enter your full name:

Amoxil	MORNING	AFTERNOON	EVENING
Messil	MORNING	AFTERNOON	EVENING

9 10

Aceptar **Cancelar**

GO BACK **SAVE**

Figure 20 - Selecting the time for pill dispense

INFORMATION:

	MORNING	AFTERNOON	EVENING
Amoxil	9 : 10	15 : 0	:
*Amount:	1	2	0
Messil	10 : 10	:	22 : 10
*Amount:	1	0	10
Antiviral	7 : 30	14 : 0	0 : 0
*Amount:	2	5	1



Figure 24 - User information screen



Figure 23 - Bluetooth connection

Layout and Formatting
A typical window in the GUI

Review user information

User: Brody

Pills	Morning	Afternoon	Evening	Total
<i>Allocate</i>	3	1	1	5
<i>Brigade</i>	1	0	0	1
<i>Sherline</i>	0	2	0	2
<i>Caddy</i>	1	0	2	3
<i>Pristine</i>	0	2	0	2

"Delete user" windowFrame with default border

**You have reached the maximum amount of users.
Please delete one of the following users or cancel.**

Brody

Patrick

Carolina

Frame with default border

Container Setup

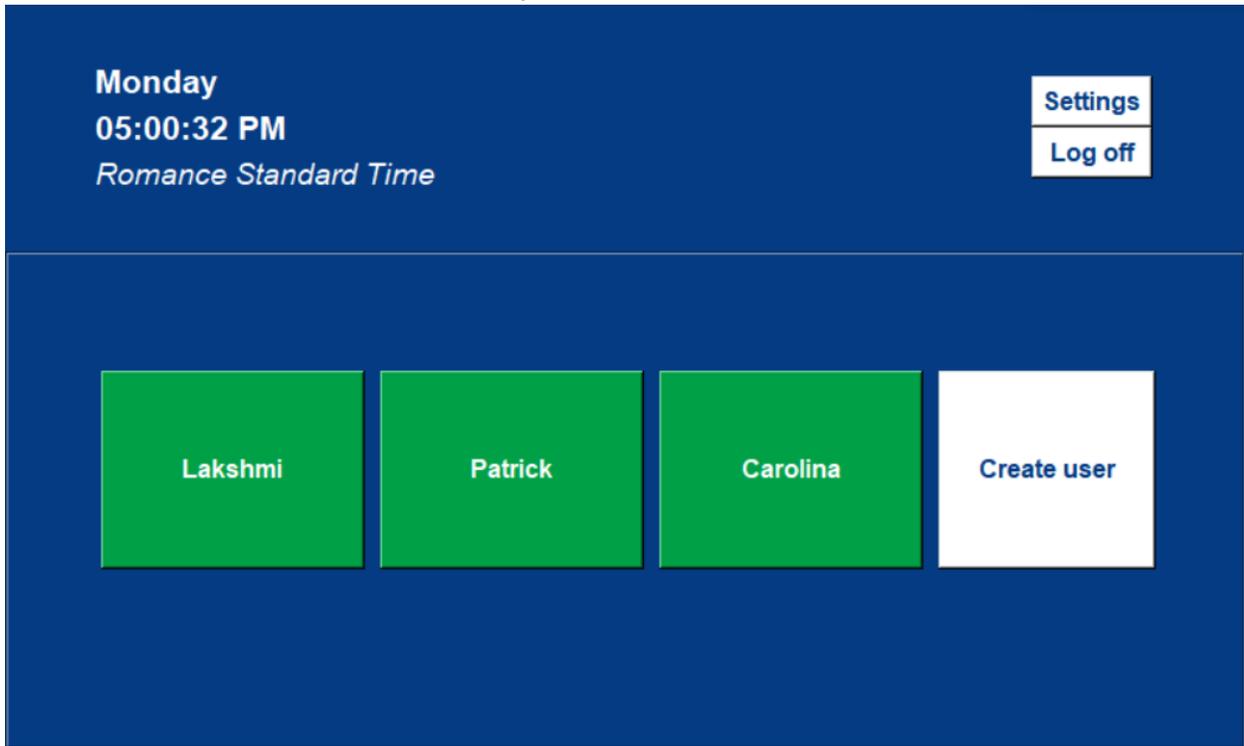
Enter the name of the pills you have placed in container 1	<input type="text"/>
Enter the number of pills in the container	<input type="text"/>
Enter the name of the pills you have placed in container 2	<input type="text"/>
Enter the number of pills in the container	<input type="text"/>
Enter the name of the pills you have placed in container 3	<input type="text"/>
Enter the number of pills in the container	<input type="text"/>
Enter the name of the pills you have placed in container 4	<input type="text"/>
Enter the number of pills in the container	<input type="text"/>
Enter the name of the pills you have placed in container 5	<input type="text"/>
Enter the number of pills in the container	<input type="text"/>

Frame with borderwidth=0

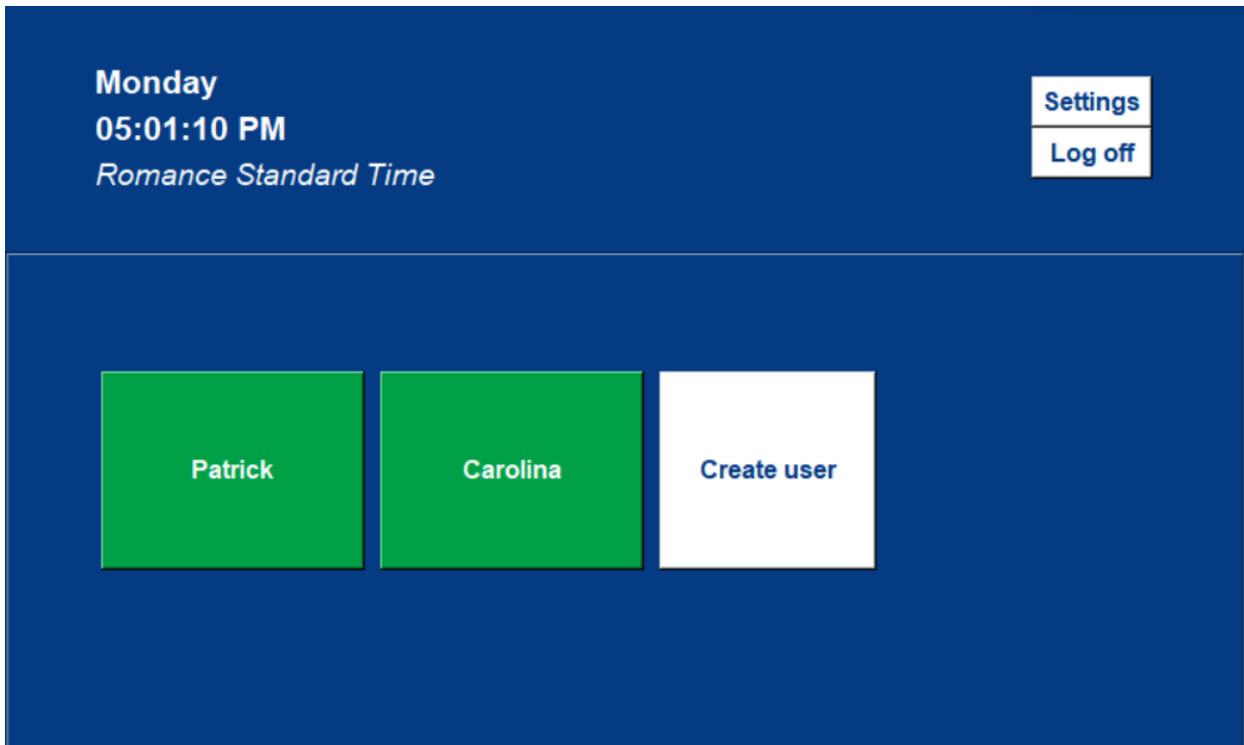
Container Setup

Enter the name of the pills you have placed in container 1	<input type="text"/>
Enter the number of pills in the container	<input type="text"/>
Enter the name of the pills you have placed in container 2	<input type="text"/>
Enter the number of pills in the container	<input type="text"/>
Enter the name of the pills you have placed in container 3	<input type="text"/>
Enter the number of pills in the container	<input type="text"/>
Enter the name of the pills you have placed in container 4	<input type="text"/>
Enter the number of pills in the container	<input type="text"/>
Enter the name of the pills you have placed in container 5	<input type="text"/>
Enter the number of pills in the container	<input type="text"/>

Grid method (frame border added for clarity)



Grid method (frame border added for clarity)



Place method allowing overlay (pink added on some labels for clarity)

Container Setup

Enter the name of the pills you have placed in container 1	<input type="text"/>
Enter the number of pills in the container	<input type="text"/>
Enter the name of the pills you have placed in container 2	<input type="text"/>
Enter the number of pills in the container	<input type="text"/>
Enter the name of the pills you have placed in container 3	<input type="text"/>
Enter the number of pills in the container	<input type="text"/>
Enter the name of the pills you have placed in container 4	<input type="text"/>
Enter the number of pills in the container	<input type="text"/>
Enter the name of the pills you have placed in container 5	<input type="text"/>
Enter the number of pills in the container	<input type="text"/>

Timeslot Setup window

When should the pills be taken?

Available Pills	Morning			Afternoon			Evening					
<i>Allocate</i>	00	→	00	→	AM	→	00	→	00	→	AM	→
<i>Brigade</i>	00	→	00	→	AM	→	00	→	00	→	AM	→
<i>Sherline</i>	00	→	00	→	AM	→	00	→	00	→	AM	→
<i>Caddy</i>	00	→	00	→	AM	→	00	→	00	→	AM	→
<i>Pristine</i>	00	→	00	→	AM	→	00	→	00	→	AM	→

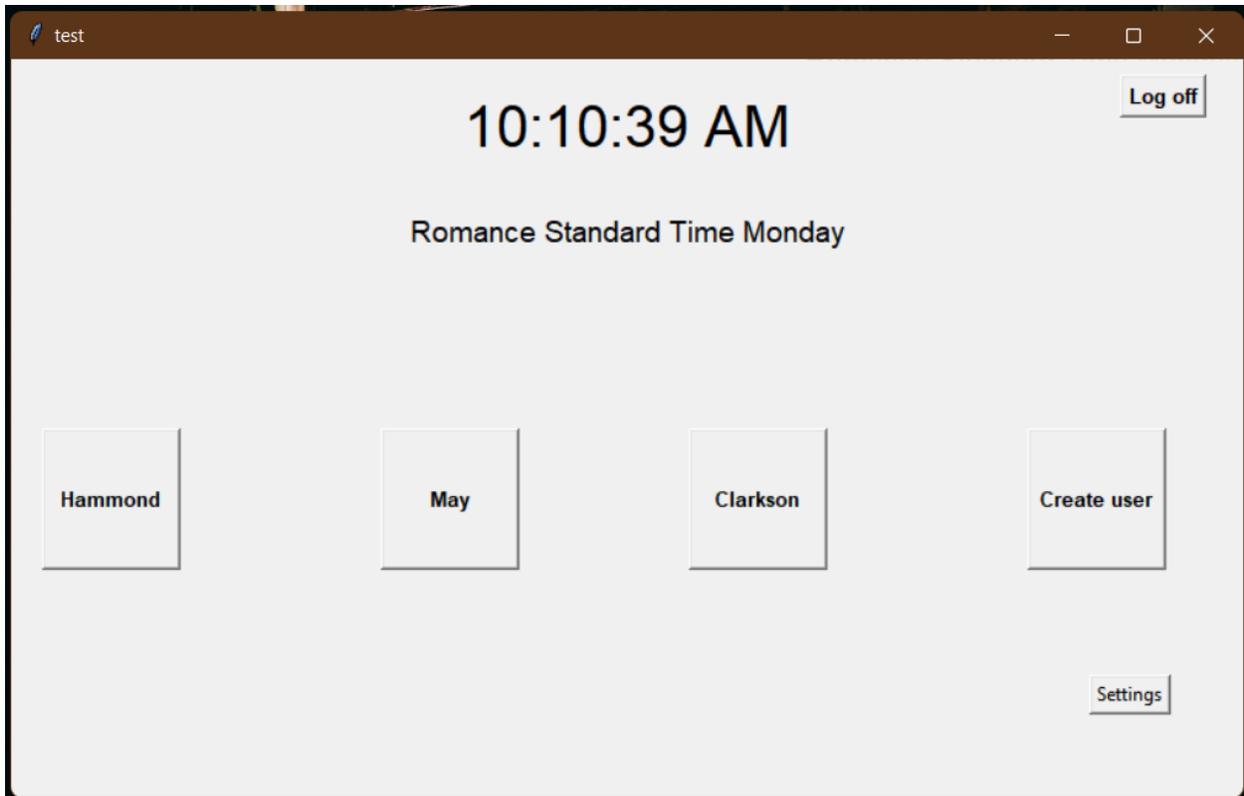
Option menus

Create a User

Enter your full name:

Set up dosage:	Morning	Afternoon	Evening	Total
<i>Allocate</i>	<input type="text" value="3"/>	<input type="text" value="1"/>	<input type="text" value="1"/>	5
<i>Brigade</i>	<input type="text" value="0"/>	<input type="text" value="0"/>	<input type="text" value="0"/>	0
<i>Sherline</i>	<input type="text" value="0"/>	<input type="text" value="0"/>	<input type="text" value="0"/>	0
<i>Caddy</i>	<input type="text" value="1"/>	<input type="text" value="0"/>	<input type="text" value="0"/>	0
<i>Pristine</i>	<input type="text" value="2"/>	<input type="text" value="0"/>	<input type="text" value="0"/>	0
	<input type="text" value="3"/>			
	<input type="text" value="4"/>			
	<input type="text" value="5"/>			
	<input type="text" value="6"/>			
	<input type="text" value="7"/>			
	<input type="text" value="8"/>			
	<input type="text" value="9"/>			

Figure 14. Initial version of the GUI without formatting




```

from tkinter import *
import tkinter as tk
from tkinter import ttk
import time
import mysql.connector
import tkinter.font
#from PIL import ImageTk,Image

def start_gui():
    global root
    root=Tk()
    root.geometry("800x480")
    root.overrideredirect(True)
    root.configure(bg="#043b82")

    #database that contains all information regarding each user
    #connecting to MySQL database
    mydb_user_info=mysql.connector.connect(
        host="localhost",
        user="root",
        passwd="password123",
        database="user_info"
    )

    #creating a cursor and intializing it
    my_cursor=mydb_user_info.cursor()
    my_cursor=mydb_user_info.cursor(buffered=True)

    #creating a database and table for database (if necessary)
    my_cursor.execute("CREATE DATABASE IF NOT EXISTS user_info")
    my_cursor.execute("CREATE TABLE IF NOT EXISTS users (name_entry VARCHAR(255),\
        p1_entry INT(5),p2_entry INT(5),p3_entry INT(5),p4_entry INT(5),p5_entry INT(5),\
        p1_morning INT(5),p1_afternoon INT(5),p1_evening INT(5),\
        p2_morning INT(5),p2_afternoon INT(5),p2_evening INT(5),\
        p3_morning INT(5),p3_afternoon INT(5),p3_evening INT(5),\
        p4_morning INT(5),p4_afternoon INT(5),p4_evening INT(5),\
        p5_morning INT(5),p5_afternoon INT(5),p5_evening INT(5),\
        user_id INT AUTO_INCREMENT PRIMARY KEY)")

    #printing out contents of the table above to terminal
    my_cursor.execute("SELECT * FROM users")
    result=my_cursor.fetchall()
    for x in result:
        print(x)

    #database to store names and amount of pills
    #connecting to MySQL database
    mydb_local_info=mysql.connector.connect(
        host="localhost",
        user="root",
        passwd="password123",
        database="local_info"
    )

    #creating a cursor and initializing it
    my_cursor_local_info=mydb_local_info.cursor()
    my_cursor_local_info=mydb_local_info.cursor(buffered=True)

    #creating a database and table for database (if necessary)
    my_cursor_local_info.execute("CREATE DATABASE IF NOT EXISTS local_info")
    my_cursor_local_info.execute("CREATE TABLE IF NOT EXISTS info (\
        p1_name VARCHAR(255),p2_name VARCHAR(255),p3_name VARCHAR(255),p4_name VARCHAR(255),p5_name VARCHAR(255),\
        p1_total_amount VARCHAR(255),p2_total_amount VARCHAR(255),p3_total_amount VARCHAR(255),\
        p4_total_amount VARCHAR(255),p5_total_amount VARCHAR(255),\
        p1_morning_time VARCHAR(255),p1_afternoon_time VARCHAR(255),p1_evening_time VARCHAR(255),\
        p2_morning_time VARCHAR(255),p2_afternoon_time VARCHAR(255),p2_evening_time VARCHAR(255),\
        p3_morning_time VARCHAR(255),p3_afternoon_time VARCHAR(255),p3_evening_time VARCHAR(255),\
        p4_morning_time VARCHAR(255),p4_afternoon_time VARCHAR(255),p4_evening_time VARCHAR(255),\
        p5_morning_time VARCHAR(255),p5_afternoon_time VARCHAR(255),p5_evening_time VARCHAR(255),\
        user_id_local INT AUTO_INCREMENT PRIMARY KEY)")

    # #printing out contents of the table above to terminal
    # my_cursor_local_info.execute("SELECT * FROM info")
    # result=my_cursor_local_info.fetchall()
    # for x in result:
    #     print(x)

```

```
'''
def openkeyboard(event):
    key = tk.Tk() #key window name
    key.geometry("800x480")
    key.overrideredirect(True)
    key.configure(bg="#043b82")

    # function coding start
    exp = " " # global variable
    # showing all data in display

    def press(num):
        global exp
        exp=exp + str(num)
        equation.set(exp)
    # end

    # function clear button

    def clear():
        global exp
        exp = " "
        equation.set(exp)
    # end

    # Enter Button Work Next line Function

    def action():
        exp = " Next Line : "
        equation.set(exp)
    # end function coding

    # Tab Button Function
    #def Tab():
        #exp = " TAB : "
        #equation.set(exp)
    # END Tab Button Fucntion

    # Size window size
    key.geometry('800x195') # normal size
    key.maxsize(width=800, height=195) # maximum size
    key.minsize(width= 800 , height = 195) # minimum size
    # end window size

    #key.configure(bg = 'white') # add background color
    #key.wm_attributes("-transparentcolor", True)
    # entry box
    key.attributes("-alpha", 0.7)
    #equation = tk.StringVar()
    #Dis_entry = ttk.Entry(key,state= 'readonly',textvariable = equation)
    #Dis_entry.grid(rowspan= 1 , columnspan = 100, ipadx = 999 , ipady = 20)
    # end entry box

    # add all button line wise

    # First Line Button

    q = ttk.Button(key,text = 'Q' , width = 6, command = lambda : press('Q'))
    q.grid(row = 1 , column = 0, ipadx = 6 , ipady = 10)

    w = ttk.Button(key,text = 'W' , width = 6, command = lambda : press('W'))
    w.grid(row = 1 , column = 1, ipadx = 6 , ipady = 10)

    E = ttk.Button(key,text = 'E' , width = 6, command = lambda : press('E'))
    E.grid(row = 1 , column = 2, ipadx = 6 , ipady = 10)

    R = ttk.Button(key,text = 'R' , width = 6, command = lambda : press('R'))
    R.grid(row = 1 , column = 3, ipadx = 6 , ipady = 10)

    T = ttk.Button(key,text = 'T' , width = 6, command = lambda : press('T'))
    T.grid(row = 1 , column = 4, ipadx = 6 , ipady = 10)

    Y = ttk.Button(key,text = 'Y' , width = 6, command = lambda : press('Y'))
    Y.grid(row = 1 , column = 5, ipadx = 6 , ipady = 10)
```

```
U = ttk.Button(key,text = 'U' , width = 6, command = lambda : press('U'))
U.grid(row = 1 , column = 6, ipadx = 6 , ipady = 10)

I = ttk.Button(key,text = 'I' , width = 6, command = lambda : press('I'))
I.grid(row = 1 , column = 7, ipadx = 6 , ipady = 10)

O = ttk.Button(key,text = 'O' , width = 6, command = lambda : press('O'))
O.grid(row = 1 , column = 8, ipadx = 6 , ipady = 10)

P = ttk.Button(key,text = 'P' , width = 6, command = lambda : press('P'))
P.grid(row = 1 , column = 9, ipadx = 6 , ipady = 10)

hyphn = ttk.Button(key,text = '-' , width = 6, command = lambda : press('-'))
hyphn.grid(row = 1 , column = 10, ipadx = 6 , ipady = 10)

clear = ttk.Button(key,text = 'Clear' , width = 6, command = clear)
clear.grid(row = 1 , column = 13, ipadx = 20 , ipady = 10)

# Second Line Button
A = ttk.Button(key,text = 'A' , width = 6, command = lambda : press('A'))
A.grid(row = 2 , column = 0, ipadx = 6 , ipady = 10)

S = ttk.Button(key,text = 'S' , width = 6, command = lambda : press('S'))
S.grid(row = 2 , column = 1, ipadx = 6 , ipady = 10)

D = ttk.Button(key,text = 'D' , width = 6, command = lambda : press('D'))
D.grid(row = 2 , column = 2, ipadx = 6 , ipady = 10)

F = ttk.Button(key,text = 'F' , width = 6, command = lambda : press('F'))
F.grid(row = 2 , column = 3, ipadx = 6 , ipady = 10)

G = ttk.Button(key,text = 'G' , width = 6, command = lambda : press('G'))
G.grid(row = 2 , column = 4, ipadx = 6 , ipady = 10)

H = ttk.Button(key,text = 'H' , width = 6, command = lambda : press('H'))
H.grid(row = 2 , column = 5, ipadx = 6 , ipady = 10)

J = ttk.Button(key,text = 'J' , width = 6, command = lambda : press('J'))
J.grid(row = 2 , column = 6, ipadx = 6 , ipady = 10)

K = ttk.Button(key,text = 'K' , width = 6, command = lambda : press('K'))
K.grid(row = 2 , column = 7, ipadx = 6 , ipady = 10)

L = ttk.Button(key,text = 'L' , width = 6, command = lambda : press('L'))
L.grid(row = 2 , column = 8, ipadx = 6 , ipady = 10)

enter = ttk.Button(key,text = 'Enter' , width = 6, command = action)
enter.grid(row = 2 , column = 8, colspan = 10, rowspan= 1, ipadx = 16 , ipady = 16)

# third line Button
Z = ttk.Button(key,text = 'Z' , width = 6, command = lambda : press('Z'))
Z.grid(row = 3 , column = 0, ipadx = 6 , ipady = 10)

X = ttk.Button(key,text = 'X' , width = 6, command = lambda : press('X'))
X.grid(row = 3 , column = 1, ipadx = 6 , ipady = 10)

C = ttk.Button(key,text = 'C' , width = 6, command = lambda : press('C'))
C.grid(row = 3 , column = 2, ipadx = 6 , ipady = 10)

V = ttk.Button(key,text = 'V' , width = 6, command = lambda : press('V'))
V.grid(row = 3 , column = 3, ipadx = 6 , ipady = 10)

B = ttk.Button(key, text= 'B' , width = 6 , command = lambda : press('B'))
B.grid(row = 3 , column = 4 , ipadx = 6 ,ipady = 10)

N = ttk.Button(key,text = 'N' , width = 6, command = lambda : press('N'))
N.grid(row = 3 , column = 5, ipadx = 6 , ipady = 10)

M = ttk.Button(key,text = 'M' , width = 6, command = lambda : press('M'))
M.grid(row = 3 , column = 6, ipadx = 6 , ipady = 10)

#Fourth Line Button
space = ttk.Button(key,text = 'Space' , width = 6, command = lambda : press(' '))
space.grid(row = 4 , colspan = 14 , ipadx = 160 , ipady = 10)
```

```

... #key.mainloop() # using ending point
...

#clock on the home screen
def clock():
    hour=time.strftime("%I")
    minute=time.strftime("%M")
    second=time.strftime("%S")
    day=time.strftime("%A")
    am_pm=time.strftime("%p")
    timezone=time.strftime("%Z")

    day_label.config(text=day)
    time_label.config(text=hour+":"+minute+":"+second+" "+am_pm)
    time_label.after(1000,clock)
    zone_label.config(text=timezone)

#setup of pills in the containers
def settings_window():
    settings=Tk()
    settings.geometry("800x480")
    settings.overrideredirect(True)
    settings.configure(bg="#043b82")

    settings_label=Label(settings,text="Container Setup",font=("Helvetica", "22", "bold"),fg="white",bg="#043b82")
    settings_label.pack(pady=10)

    def save_settings():
        #getting the settings data for the 'info' table
        insert_name="INSERT INTO info(p1_name,p2_name,p3_name,p4_name,p5_name,\
        p1_total_amount,p2_total_amount,p3_total_amount,p4_total_amount,p5_total_amount)\
        VALUES(%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s)"
        data=(p1_name_entry.get(),p2_name_entry.get(),p3_name_entry.get(),p4_name_entry.get(),p5_name_entry.get(),\
        p1_amount_entry.get(),p2_amount_entry.get(),p3_amount_entry.get(),p4_amount_entry.get(),p5_amount_entry.get())

        #entering the settings data into the 'info' table
        my_cursor_local_info.execute(insert_name,data)

        #committing changes to the 'local_info' database
        mydb_local_info.commit()

    #labels so the user knows what to input in the settings
    setInfo_frame=LabelFrame(settings,borderwidth=0,bg="#043b82")
    setInfo_frame.pack(side=TOP,fill=BOTH,expand=True)

    p1_name_label=Label(setInfo_frame,text="Enter the name of the pills you have placed in container 1",\
        font=("Helvetica", "12", "bold"),fg="white",bg="#043b82")
    p1_name_label.place(anchor=NW,relx=0.07,relly=0.07)

    p1_amount_label=Label(setInfo_frame,text="Enter the number of pills in the container",\
        font=("Helvetica", "12", "bold"),fg="white",bg="#043b82")
    p1_amount_label.place(anchor=NW,relx=0.07,relly=0.13)

    p2_name_label=Label(setInfo_frame,text="Enter the name of the pills you have placed in container 2",\
        font=("Helvetica", "12", "bold"),fg="white",bg="#043b82")
    p2_name_label.place(anchor=NW,relx=0.07,relly=0.25)

    p2_amount_label=Label(setInfo_frame,text="Enter the number of pills in the container",\
        font=("Helvetica", "12", "bold"),fg="white",bg="#043b82")
    p2_amount_label.place(anchor=NW,relx=0.07,relly=0.31)

    p3_name_label=Label(setInfo_frame,text="Enter the name of the pills you have placed in container 3",\
        font=("Helvetica", "12", "bold"),fg="white",bg="#043b82")
    p3_name_label.place(anchor=NW,relx=0.07,relly=0.43)

    p3_amount_label=Label(setInfo_frame,text="Enter the number of pills in the container",\
        font=("Helvetica", "12", "bold"),fg="white",bg="#043b82")
    p3_amount_label.place(anchor=NW,relx=0.07,relly=0.49)

    p4_name_label=Label(setInfo_frame,text="Enter the name of the pills you have placed in container 4",\
        font=("Helvetica", "12", "bold"),fg="white",bg="#043b82")
    p4_name_label.place(anchor=NW,relx=0.07,relly=0.61)

    p4_amount_label=Label(setInfo_frame,text="Enter the number of pills in the container",\
        font=("Helvetica", "12", "bold"),fg="white",bg="#043b82")
    p4_amount_label.place(anchor=NW,relx=0.07,relly=0.67)

```

```

p5_name_label=Label(setInfo_frame,text="Enter the name of the pills you have placed in container 5",\
    font=("Helvetica", "12", "bold"),fg="white",bg="#043b82")
p5_name_label.place(anchor=NW,relx=0.07,relly=0.79)

p5_amount_label=Label(setInfo_frame,text="Enter the number of pills in the container",\
    font=("Helvetica", "12", "bold"),fg="white",bg="#043b82")
p5_amount_label.place(anchor=NW,relx=0.07,relly=0.85)

#entry boxes for the user to input the pill names and amounts
p1_name_entry=Entry(setInfo_frame,borderwidth=0,width=30,font=("Helvetica", "10", "bold"),fg="#043b82",bg="white")
p1_name_entry.place(anchor=NE,relx=0.92,relly=0.08)

p1_amount_entry=Entry(setInfo_frame,borderwidth=0,width=30,font=("Helvetica", "10", "bold"),fg="#043b82",bg="white")
p1_amount_entry.place(anchor=NE,relx=0.92,relly=0.14)

p2_name_entry=Entry(setInfo_frame,borderwidth=0,width=30,font=("Helvetica", "10", "bold"),fg="#043b82",bg="white")
p2_name_entry.place(anchor=NE,relx=0.92,relly=0.26)

p2_amount_entry=Entry(setInfo_frame,borderwidth=0,width=30,font=("Helvetica", "10", "bold"),fg="#043b82",bg="white")
p2_amount_entry.place(anchor=NE,relx=0.92,relly=0.32)

p3_name_entry=Entry(setInfo_frame,borderwidth=0,width=30,font=("Helvetica", "10", "bold"),fg="#043b82",bg="white")
p3_name_entry.place(anchor=NE,relx=0.92,relly=0.44)

p3_amount_entry=Entry(setInfo_frame,borderwidth=0,width=30,font=("Helvetica", "10", "bold"),fg="#043b82",bg="white")
p3_amount_entry.place(anchor=NE,relx=0.92,relly=0.50)

p4_name_entry=Entry(setInfo_frame,borderwidth=0,width=30,font=("Helvetica", "10", "bold"),fg="#043b82",bg="white")
p4_name_entry.place(anchor=NE,relx=0.92,relly=0.62)

p4_amount_entry=Entry(setInfo_frame,borderwidth=0,width=30,font=("Helvetica", "10", "bold"),fg="#043b82",bg="white")
p4_amount_entry.place(anchor=NE,relx=0.92,relly=0.68)

p5_name_entry=Entry(setInfo_frame,borderwidth=0,width=30,font=("Helvetica", "10", "bold"),fg="#043b82",bg="white")
p5_name_entry.place(anchor=NE,relx=0.92,relly=0.80)

p5_amount_entry=Entry(setInfo_frame,borderwidth=0,width=30,font=("Helvetica", "10", "bold"),fg="#043b82",bg="white")
p5_amount_entry.place(anchor=NE,relx=0.92,relly=0.86)

#button to save the data given in this (settings) window
setBtn_frame=LabelFrame(settings,borderwidth=0,bg="#043b82")
setBtn_frame.pack(side=BOTTOM,fill=X,padx=60,pady=20)

save_button=Button(setBtn_frame,text="Save information",command=Lambda:[save_settings(),settings.destroy()],\
    font=("Helvetica", "10", "bold italic"),fg="#043b82",bg="white",activeforeground="white",activebackground="#00a047")
save_button.pack(fill=X,expand=True)

def enter_info_window():
    top=Tk()
    top.geometry("800x480")
    top.overrideRedirect(True)
    top.configure(bg="#043b82")

    #1st window when clicking on "Create User" that is used to set up times to take pills
    timeSetup_label=Label(top,text="When should the pills be taken?",font=("Helvetica", "22", "bold"),fg="white",bg="#043b82")
    timeSetup_label.pack()

    #retrieving most recent data from 'info' table
    my_cursor_local_info.execute("SELECT * FROM info ORDER BY user_id_local DESC LIMIT 1")
    most_local_info=my_cursor_local_info.fetchall()

    TS_frame=LabelFrame(top,borderwidth=0,bg="#043b82")
    TS_frame.pack(side=TOP,fill=X,padx=30)

    TSpill_frame=LabelFrame(TS_frame,borderwidth=0,bg="#043b82")
    TSpill_frame.pack(side=LEFT,fill=X,expand=True,padx=10)

    pills_label=Label(TSpill_frame,text="Available Pills",font=("Helvetica", "16", "bold"),fg="white",bg="#043b82")
    pills_label.pack(side=TOP,fill=X,expand=True,pady=1)

    p1TS_name=Label(TSpill_frame,text=str(most_local_info[0][0]),font=("Helvetica", "16", "italic"),fg="white",bg="#043b82")
    p1TS_name.pack(pady=1)

    p2TS_name=Label(TSpill_frame,text=str(most_local_info[0][1]),font=("Helvetica", "16", "italic"),fg="white",bg="#043b82")
    p2TS_name.pack(pady=1)

```

```
p3TS_name=Label(TSpill_frame,text=str(most_local_info[0][2]),font=("Helvetica", "16", "italic"),fg="white",bg="#043b82",
p3TS_name.pack(pady=1)

p4TS_name=Label(TSpill_frame,text=str(most_local_info[0][3]),font=("Helvetica", "16", "italic"),fg="white",bg="#043b82",
p4TS_name.pack(pady=1)

p5TS_name=Label(TSpill_frame,text=str(most_local_info[0][4]),font=("Helvetica", "16", "italic"),fg="white",bg="#043b82",
p5TS_name.pack(pady=1)

#lists for the drop down menus
time_hour_list=["00", "01", "02", "03", "04", "05", "06", "07", "08", "09", "10", "11", "12"]
time_minute_list=["00", "15", "30", "45"]
time_ampm_list=["AM", "PM"]

#variables for the drop down menus
#setting up variables for the morning hours
p1_morning_hour=StringVar(top)
p1_morning_hour.set(time_hour_list[0])

p2_morning_hour=StringVar(top)
p2_morning_hour.set(time_hour_list[0])

p3_morning_hour=StringVar(top)
p3_morning_hour.set(time_hour_list[0])

p4_morning_hour=StringVar(top)
p4_morning_hour.set(time_hour_list[0])

p5_morning_hour=StringVar(top)
p5_morning_hour.set(time_hour_list[0])

#setting up variables for the afternoon hours
p1_afternoon_hour=StringVar(top)
p1_afternoon_hour.set(time_hour_list[0])

p2_afternoon_hour=StringVar(top)
p2_afternoon_hour.set(time_hour_list[0])

p3_afternoon_hour=StringVar(top)
p3_afternoon_hour.set(time_hour_list[0])

p4_afternoon_hour=StringVar(top)
p4_afternoon_hour.set(time_hour_list[0])

p5_afternoon_hour=StringVar(top)
p5_afternoon_hour.set(time_hour_list[0])

#setting up variables for the evening hours
p1_evening_hour=StringVar(top)
p1_evening_hour.set(time_hour_list[0])

p2_evening_hour=StringVar(top)
p2_evening_hour.set(time_hour_list[0])

p3_evening_hour=StringVar(top)
p3_evening_hour.set(time_hour_list[0])

p4_evening_hour=StringVar(top)
p4_evening_hour.set(time_hour_list[0])

p5_evening_hour=StringVar(top)
p5_evening_hour.set(time_hour_list[0])

#setting up variables for the morning minutes
p1_morning_minute=StringVar(top)
p1_morning_minute.set(time_minute_list[0])

p2_morning_minute=StringVar(top)
p2_morning_minute.set(time_minute_list[0])

p3_morning_minute=StringVar(top)
p3_morning_minute.set(time_minute_list[0])

p4_morning_minute=StringVar(top)
p4_morning_minute.set(time_minute_list[0])
```

```
p5_morning_minute=StringVar(top)
p5_morning_minute.set(time_minute_list[0])

#setting up variables for the afternoon minutes
p1_afternoon_minute=StringVar(top)
p1_afternoon_minute.set(time_minute_list[0])

p2_afternoon_minute=StringVar(top)
p2_afternoon_minute.set(time_minute_list[0])

p3_afternoon_minute=StringVar(top)
p3_afternoon_minute.set(time_minute_list[0])

p4_afternoon_minute=StringVar(top)
p4_afternoon_minute.set(time_minute_list[0])

p5_afternoon_minute=StringVar(top)
p5_afternoon_minute.set(time_minute_list[0])

#setting up variables for the evening minutes
p1_evening_minute=StringVar(top)
p1_evening_minute.set(time_minute_list[0])

p2_evening_minute=StringVar(top)
p2_evening_minute.set(time_minute_list[0])

p3_evening_minute=StringVar(top)
p3_evening_minute.set(time_minute_list[0])

p4_evening_minute=StringVar(top)
p4_evening_minute.set(time_minute_list[0])

p5_evening_minute=StringVar(top)
p5_evening_minute.set(time_minute_list[0])

#setting up AM/PM variables for the morning
p1_morning_ampm=StringVar(top)
p1_morning_ampm.set(time_ampm_list[0])

p2_morning_ampm=StringVar(top)
p2_morning_ampm.set(time_ampm_list[0])

p3_morning_ampm=StringVar(top)
p3_morning_ampm.set(time_ampm_list[0])

p4_morning_ampm=StringVar(top)
p4_morning_ampm.set(time_ampm_list[0])

p5_morning_ampm=StringVar(top)
p5_morning_ampm.set(time_ampm_list[0])

#setting up AM/PM variables for the afternoon
p1_afternoon_ampm=StringVar(top)
p1_afternoon_ampm.set(time_ampm_list[0])

p2_afternoon_ampm=StringVar(top)
p2_afternoon_ampm.set(time_ampm_list[0])

p3_afternoon_ampm=StringVar(top)
p3_afternoon_ampm.set(time_ampm_list[0])

p4_afternoon_ampm=StringVar(top)
p4_afternoon_ampm.set(time_ampm_list[0])

p5_afternoon_ampm=StringVar(top)
p5_afternoon_ampm.set(time_ampm_list[0])

#setting up AM/PM variables for the evening
p1_evening_ampm=StringVar(top)
p1_evening_ampm.set(time_ampm_list[0])

p2_evening_ampm=StringVar(top)
p2_evening_ampm.set(time_ampm_list[0])

p3_evening_ampm=StringVar(top)
p3_evening_ampm.set(time_ampm_list[0])
```

```
p4_evening_ampm=StringVar(top)
p4_evening_ampm.set(time_ampm_list[0])

p5_evening_ampm=StringVar(top)
p5_evening_ampm.set(time_ampm_list[0])

#placing the morning section in the window
TSmorning_frame=LabelFrame(TS_frame,borderwidth=0,bg="#043b82")
TSmorning_frame.pack(side=LEFT,padx=10)

morning_time_label=Label(TSmorning_frame,text="Morning",font=("Helvetica","16","bold"),fg="white",bg="#043b82")
morning_time_label.pack(side=TOP,fill=X)

#placing the morning hour column in the morning section
morHOUR_frame=LabelFrame(TSmorning_frame,borderwidth=0,bg="#043b82")
morHOUR_frame.pack(side=LEFT)

#creating and placing drop down menus for the morning hours
p1_morning_hour_ddm=OptionMenu(morHOUR_frame,p1_morning_hour,*time_hour_list)
p1_morning_hour_ddm.config(borderwidth=0,width=1,font=("Helvetica","12","bold"),fg="#043b82",bg="white")
p1_morning_hour_ddm["menu"].config(borderwidth=0,font=("Helvetica","9","bold"),fg="#043b82",bg="white",\
    activeforeground="white",activebackground="#00a047")
p1_morning_hour_ddm.pack()

p2_morning_hour_ddm=OptionMenu(morHOUR_frame,p2_morning_hour,*time_hour_list)
p2_morning_hour_ddm.config(borderwidth=0,width=1,font=("Helvetica","12","bold"),fg="#043b82",bg="white")
p2_morning_hour_ddm["menu"].config(borderwidth=0,font=("Helvetica","9","bold"),fg="#043b82",bg="white",\
    activeforeground="white",activebackground="#00a047")
p2_morning_hour_ddm.pack()

p3_morning_hour_ddm=OptionMenu(morHOUR_frame,p3_morning_hour,*time_hour_list)
p3_morning_hour_ddm.config(borderwidth=0,width=1,font=("Helvetica","12","bold"),fg="#043b82",bg="white")
p3_morning_hour_ddm["menu"].config(borderwidth=0,font=("Helvetica","9","bold"),fg="#043b82",bg="white",\
    activeforeground="white",activebackground="#00a047")
p3_morning_hour_ddm.pack()

p4_morning_hour_ddm=OptionMenu(morHOUR_frame,p4_morning_hour,*time_hour_list)
p4_morning_hour_ddm.config(borderwidth=0,width=1,font=("Helvetica","12","bold"),fg="#043b82",bg="white")
p4_morning_hour_ddm["menu"].config(borderwidth=0,font=("Helvetica","9","bold"),fg="#043b82",bg="white",\
    activeforeground="white",activebackground="#00a047")
p4_morning_hour_ddm.pack()

p5_morning_hour_ddm=OptionMenu(morHOUR_frame,p5_morning_hour,*time_hour_list)
p5_morning_hour_ddm.config(width=1, bg="white", fg="#043b82", font=("Helvetica", "12", "bold"), borderwidth=0)
p5_morning_hour_ddm["menu"].config(borderwidth=0,font=("Helvetica","9","bold"),fg="#043b82",bg="white",\
    activeforeground="white",activebackground="#00a047")
p5_morning_hour_ddm.pack()

#placing the morning minutes column in the morning section
morMIN_frame=LabelFrame(TSmorning_frame,borderwidth=0,bg="#043b82")
morMIN_frame.pack(side=LEFT)

#creating and placing drop down menus for the morning minutes
p1_morning_minute_ddm=OptionMenu(morMIN_frame,p1_morning_minute,*time_minute_list)
p1_morning_minute_ddm.config(borderwidth=0,width=1,font=("Helvetica","12","bold"),fg="#043b82",bg="white")
p1_morning_minute_ddm["menu"].config(borderwidth=0,font=("Helvetica","9","bold"),fg="#043b82",bg="white",\
    activeforeground="white",activebackground="#00a047")
p1_morning_minute_ddm.pack()

p2_morning_minute_ddm=OptionMenu(morMIN_frame,p2_morning_minute,*time_minute_list)
p2_morning_minute_ddm.config(borderwidth=0,width=1,font=("Helvetica","12","bold"),fg="#043b82",bg="white")
p2_morning_minute_ddm["menu"].config(borderwidth=0,font=("Helvetica","9","bold"),fg="#043b82",bg="white",\
    activeforeground="white",activebackground="#00a047")
p2_morning_minute_ddm.pack()

p3_morning_minute_ddm=OptionMenu(morMIN_frame,p3_morning_minute,*time_minute_list)
p3_morning_minute_ddm.config(width=1, bg="white", fg="#043b82", font=("Helvetica", "12", "bold"), borderwidth=0)
p3_morning_minute_ddm["menu"].config(bg="white", fg="#043b82", font=("Helvetica", "9", "bold"), borderwidth=0)
p3_morning_minute_ddm.pack()

p4_morning_minute_ddm=OptionMenu(morMIN_frame,p4_morning_minute,*time_minute_list)
p4_morning_minute_ddm.config(borderwidth=0,width=1,font=("Helvetica","12","bold"),fg="#043b82",bg="white")
p4_morning_minute_ddm["menu"].config(borderwidth=0,font=("Helvetica","9","bold"),fg="#043b82",bg="white",\
    activeforeground="white",activebackground="#00a047")
p4_morning_minute_ddm.pack()

p5_morning_minute_ddm=OptionMenu(morMIN_frame,p5_morning_minute,*time_minute_list)
```

```

p5_morning_minute_ddm.config(borderwidth=0,width=1,font=("Helvetica","12","bold"),fg="#043b82",bg="white")
p5_morning_minute_ddm["menu"].config(borderwidth=0,font=("Helvetica","9","bold"),fg="#043b82",bg="white",\
    activeforeground="white",activebackground="#00a047")
p5_morning_minute_ddm.pack()

#placing the morning AM/PM column in the morning section
morAMPM_frame=LabelFrame(TSmorning_frame,borderwidth=0,bg="#043b82")
morAMPM_frame.pack(side=LEFT)

#creating and placing drop down menus for AM/PM in the morning section
p1_morning_ampm_ddm=OptionMenu(morAMPM_frame,p1_morning_ampm,*time_ampm_list)
p1_morning_ampm_ddm.config(borderwidth=0,width=2,font=("Helvetica","12","bold"),fg="#043b82",bg="white")
p1_morning_ampm_ddm["menu"].config(borderwidth=0,font=("Helvetica","9","bold"),fg="#043b82",bg="white",\
    activeforeground="white",activebackground="#00a047")
p1_morning_ampm_ddm.pack()

p2_morning_ampm_ddm=OptionMenu(morAMPM_frame,p2_morning_ampm,*time_ampm_list)
p2_morning_ampm_ddm.config(borderwidth=0,width=2,font=("Helvetica","12","bold"),fg="#043b82",bg="white")
p2_morning_ampm_ddm["menu"].config(borderwidth=0,font=("Helvetica","9","bold"),fg="#043b82",bg="white",\
    activeforeground="white",activebackground="#00a047")
p2_morning_ampm_ddm.pack()

p3_morning_ampm_ddm=OptionMenu(morAMPM_frame,p3_morning_ampm,*time_ampm_list)
p3_morning_ampm_ddm.config(borderwidth=0,width=2,font=("Helvetica","12","bold"),fg="#043b82",bg="white")
p3_morning_ampm_ddm["menu"].config(borderwidth=0,font=("Helvetica","9","bold"),fg="#043b82",bg="white",\
    activeforeground="white",activebackground="#00a047")
p3_morning_ampm_ddm.pack()

p4_morning_ampm_ddm=OptionMenu(morAMPM_frame,p4_morning_ampm,*time_ampm_list)
p4_morning_ampm_ddm.config(borderwidth=0,width=2,font=("Helvetica","12","bold"),fg="#043b82",bg="white")
p4_morning_ampm_ddm["menu"].config(borderwidth=0,font=("Helvetica","9","bold"),fg="#043b82",bg="white",\
    activeforeground="white",activebackground="#00a047")
p4_morning_ampm_ddm.pack()

p5_morning_ampm_ddm=OptionMenu(morAMPM_frame,p5_morning_ampm,*time_ampm_list)
p5_morning_ampm_ddm.config(borderwidth=0,width=2,font=("Helvetica","12","bold"),fg="#043b82",bg="white")
p5_morning_ampm_ddm["menu"].config(borderwidth=0,font=("Helvetica","9","bold"),fg="#043b82",bg="white",\
    activeforeground="white",activebackground="#00a047")
p5_morning_ampm_ddm.pack()

#placing the afternoon section in the window
TSafternoon_frame=LabelFrame(TS_frame,borderwidth=0,bg="#043b82")
TSafternoon_frame.pack(side=LEFT,padx=10)

afternoon_time_label=Label(TSafternoon_frame,text="Afternoon",font=("Helvetica","16","bold"),fg="white",bg="#043b82")
afternoon_time_label.pack(side=TOP,fill=X)

#placing the afternoon hour column in the afternoon section
noonHOUR_frame=LabelFrame(TSafternoon_frame,borderwidth=0,bg="#043b82")
noonHOUR_frame.pack(side=LEFT)

#creating and placing drop down menus for the afternoon hours
p1_afternoon_hour_ddm=OptionMenu(noonHOUR_frame,p1_afternoon_hour,*time_hour_list)
p1_afternoon_hour_ddm.config(borderwidth=0,width=1,font=("Helvetica","12","bold"),fg="#043b82",bg="white")
p1_afternoon_hour_ddm["menu"].config(borderwidth=0,font=("Helvetica","9","bold"),fg="#043b82",bg="white",\
    activeforeground="white",activebackground="#00a047")
p1_afternoon_hour_ddm.pack()

p2_afternoon_hour_ddm=OptionMenu(noonHOUR_frame,p2_afternoon_hour,*time_hour_list)
p2_afternoon_hour_ddm.config(borderwidth=0,width=1,font=("Helvetica","12","bold"),fg="#043b82",bg="white")
p2_afternoon_hour_ddm["menu"].config(borderwidth=0,font=("Helvetica","9","bold"),fg="#043b82",bg="white",\
    activeforeground="white",activebackground="#00a047")
p2_afternoon_hour_ddm.pack()

p3_afternoon_hour_ddm=OptionMenu(noonHOUR_frame,p3_afternoon_hour,*time_hour_list)
p3_afternoon_hour_ddm.config(width=1,bg="white",fg="#043b82",font=("Helvetica","12","bold"),borderwidth=0)
p3_afternoon_hour_ddm["menu"].config(borderwidth=0,font=("Helvetica","9","bold"),fg="#043b82",bg="white",\
    activeforeground="white",activebackground="#00a047")
p3_afternoon_hour_ddm.pack()

p4_afternoon_hour_ddm=OptionMenu(noonHOUR_frame,p4_afternoon_hour,*time_hour_list)
p4_afternoon_hour_ddm.config(borderwidth=0,width=1,font=("Helvetica","12","bold"),fg="#043b82",bg="white")
p4_afternoon_hour_ddm["menu"].config(borderwidth=0,font=("Helvetica","9","bold"),fg="#043b82",bg="white",\
    activeforeground="white",activebackground="#00a047")
p4_afternoon_hour_ddm.pack()

p5_afternoon_hour_ddm=OptionMenu(noonHOUR_frame,p5_afternoon_hour,*time_hour_list)

```

```
p5_afternoon_hour_ddm.config(borderwidth=0,width=1,font=("Helvetica","12","bold"),fg="#043b82",bg="white")
p5_afternoon_hour_ddm["menu"].config(borderwidth=0,font=("Helvetica","9","bold"),fg="#043b82",bg="white",\
    activeforeground="white",activebackground="#00a047")
p5_afternoon_hour_ddm.pack()

#placing the afternoon minutes column in the afternoon section
noonMIN_frame=LabelFrame(TSafternoon_frame,borderwidth=0,bg="#043b82")
noonMIN_frame.pack(side=LEFT)

#creating and placing drop down menus for the afternoon minutes
p1_afternoon_minute_ddm=OptionMenu(noonMIN_frame,p1_afternoon_minute,*time_minute_list)
p1_afternoon_minute_ddm.config(borderwidth=0,width=1,font=("Helvetica","12","bold"),fg="#043b82",bg="white")
p1_afternoon_minute_ddm["menu"].config(borderwidth=0,font=("Helvetica","9","bold"),fg="#043b82",bg="white",\
    activeforeground="white",activebackground="#00a047")
p1_afternoon_minute_ddm.pack()

p2_afternoon_minute_ddm=OptionMenu(noonMIN_frame,p2_afternoon_minute,*time_minute_list)
p2_afternoon_minute_ddm.config(borderwidth=0,width=1,font=("Helvetica","12","bold"),fg="#043b82",bg="white")
p2_afternoon_minute_ddm["menu"].config(borderwidth=0,font=("Helvetica","9","bold"),fg="#043b82",bg="white",\
    activeforeground="white",activebackground="#00a047")
p2_afternoon_minute_ddm.pack()

p3_afternoon_minute_ddm=OptionMenu(noonMIN_frame,p3_afternoon_minute,*time_minute_list)
p3_afternoon_minute_ddm.config(borderwidth=0,width=1,font=("Helvetica","12","bold"),fg="#043b82",bg="white")
p3_afternoon_minute_ddm["menu"].config(borderwidth=0,font=("Helvetica","9","bold"),fg="#043b82",bg="white",\
    activeforeground="white",activebackground="#00a047")
p3_afternoon_minute_ddm.pack()

p4_afternoon_minute_ddm=OptionMenu(noonMIN_frame,p4_afternoon_minute,*time_minute_list)
p4_afternoon_minute_ddm.config(borderwidth=0,width=1,font=("Helvetica","12","bold"),fg="#043b82",bg="white")
p4_afternoon_minute_ddm["menu"].config(borderwidth=0,font=("Helvetica","9","bold"),fg="#043b82",bg="white",\
    activeforeground="white",activebackground="#00a047")
p4_afternoon_minute_ddm.pack()

p5_afternoon_minute_ddm=OptionMenu(noonMIN_frame,p5_afternoon_minute,*time_minute_list)
p5_afternoon_minute_ddm.config(borderwidth=0,width=1,font=("Helvetica","12","bold"),fg="#043b82",bg="white")
p5_afternoon_minute_ddm["menu"].config(borderwidth=0,font=("Helvetica","9","bold"),fg="#043b82",bg="white",\
    activeforeground="white",activebackground="#00a047")
p5_afternoon_minute_ddm.pack()

#placing the morning AM/PM column in the afternoon section
noonAMPM_frame=LabelFrame(TSafternoon_frame,borderwidth=0,bg="#043b82")
noonAMPM_frame.pack(side=LEFT)

#creating and placing drop down menus for AM/PM in the afternoon section
p1_afternoon_ampm_ddm=OptionMenu(noonAMPM_frame,p1_afternoon_ampm,*time_ampm_list)
p1_afternoon_ampm_ddm.config(borderwidth=0,width=2,font=("Helvetica","12","bold"),fg="#043b82",bg="white")
p1_afternoon_ampm_ddm["menu"].config(borderwidth=0,font=("Helvetica","9","bold"),fg="#043b82",bg="white",\
    activeforeground="white",activebackground="#00a047")
p1_afternoon_ampm_ddm.pack()

p2_afternoon_ampm_ddm=OptionMenu(noonAMPM_frame,p2_afternoon_ampm,*time_ampm_list)
p2_afternoon_ampm_ddm.config(borderwidth=0,width=2,font=("Helvetica","12","bold"),fg="#043b82",bg="white")
p2_afternoon_ampm_ddm["menu"].config(borderwidth=0,font=("Helvetica","9","bold"),fg="#043b82",bg="white",\
    activeforeground="white",activebackground="#00a047")
p2_afternoon_ampm_ddm.pack()

p3_afternoon_ampm_ddm=OptionMenu(noonAMPM_frame,p3_afternoon_ampm,*time_ampm_list)
p3_afternoon_ampm_ddm.config(borderwidth=0,width=2,font=("Helvetica","12","bold"),fg="#043b82",bg="white")
p3_afternoon_ampm_ddm["menu"].config(borderwidth=0,font=("Helvetica","9","bold"),fg="#043b82",bg="white",\
    activeforeground="white",activebackground="#00a047")
p3_afternoon_ampm_ddm.pack()

p4_afternoon_ampm_ddm=OptionMenu(noonAMPM_frame,p4_afternoon_ampm,*time_ampm_list)
p4_afternoon_ampm_ddm.config(borderwidth=0,width=2,font=("Helvetica","12","bold"),fg="#043b82",bg="white")
p4_afternoon_ampm_ddm["menu"].config(borderwidth=0,font=("Helvetica","9","bold"),fg="#043b82",bg="white",\
    activeforeground="white",activebackground="#00a047")
p4_afternoon_ampm_ddm.pack()

p5_afternoon_ampm_ddm=OptionMenu(noonAMPM_frame,p5_afternoon_ampm,*time_ampm_list)
p5_afternoon_ampm_ddm.config(borderwidth=0,width=2,font=("Helvetica","12","bold"),fg="#043b82",bg="white")
p5_afternoon_ampm_ddm["menu"].config(borderwidth=0,font=("Helvetica","9","bold"),fg="#043b82",bg="white",\
    activeforeground="white",activebackground="#00a047")
p5_afternoon_ampm_ddm.pack()

#placing the evening section in the evening section
TSevening_frame=LabelFrame(TS_frame,borderwidth=0,bg="#043b82")
```

```

TSevening_frame.pack(side=LEFT,padx=10)

evening_time_label=Label(TSevening_frame,text="Evening",font=("Helvetica","16","bold"),fg="white",bg="#043b82")
evening_time_label.pack(side=TOP,fill=X)

#placing the evening hour column in the window
eveHOUR_frame=LabelFrame(TSevening_frame,borderwidth=0,bg="#043b82")
eveHOUR_frame.pack(side=LEFT)

#creating and placing drop down menus for the evening hours
p1_evening_hour_ddm=OptionMenu(eveHOUR_frame,p1_evening_hour,*time_hour_list)
p1_evening_hour_ddm.config(borderwidth=0,width=1,font=("Helvetica","12","bold"),fg="#043b82",bg="white")
p1_evening_hour_ddm["menu"].config(borderwidth=0,font=("Helvetica","9","bold"),fg="#043b82",bg="white",\
    activeforeground="white",activebackground="#00a047")
p1_evening_hour_ddm.pack()

p2_evening_hour_ddm=OptionMenu(eveHOUR_frame,p2_evening_hour,*time_hour_list)
p2_evening_hour_ddm.config(borderwidth=0,width=1,font=("Helvetica","12","bold"),fg="#043b82",bg="white")
p2_evening_hour_ddm["menu"].config(borderwidth=0,font=("Helvetica","9","bold"),fg="#043b82",bg="white",\
    activeforeground="white",activebackground="#00a047")
p2_evening_hour_ddm.pack()

p3_evening_hour_ddm=OptionMenu(eveHOUR_frame,p3_evening_hour,*time_hour_list)
p3_evening_hour_ddm.config(borderwidth=0,width=1,font=("Helvetica","12","bold"),fg="#043b82",bg="white")
p3_evening_hour_ddm["menu"].config(borderwidth=0,font=("Helvetica","9","bold"),fg="#043b82",bg="white",\
    activeforeground="white",activebackground="#00a047")
p3_evening_hour_ddm.pack()

p4_evening_hour_ddm=OptionMenu(eveHOUR_frame,p4_evening_hour,*time_hour_list)
p4_evening_hour_ddm.config(borderwidth=0,width=1,font=("Helvetica","12","bold"),fg="#043b82",bg="white")
p4_evening_hour_ddm["menu"].config(borderwidth=0,font=("Helvetica","9","bold"),fg="#043b82",bg="white",\
    activeforeground="white",activebackground="#00a047")
p4_evening_hour_ddm.pack()

p5_evening_hour_ddm=OptionMenu(eveHOUR_frame,p5_evening_hour,*time_hour_list)
p5_evening_hour_ddm.config(borderwidth=0,width=1,font=("Helvetica","12","bold"),fg="#043b82",bg="white")
p5_evening_hour_ddm["menu"].config(borderwidth=0,font=("Helvetica","9","bold"),fg="#043b82",bg="white",\
    activeforeground="white",activebackground="#00a047")
p5_evening_hour_ddm.pack()

#placing the evening minutes column in the evening section
eveMIN_frame=LabelFrame(TSevening_frame,borderwidth=0,bg="#043b82")
eveMIN_frame.pack(side=LEFT)

#creating and placing drop down menus for the evening minutes
p1_evening_minute_ddm=OptionMenu(eveMIN_frame,p1_evening_minute,*time_minute_list)
p1_evening_minute_ddm.config(borderwidth=0,width=1,font=("Helvetica","12","bold"),fg="#043b82",bg="white")
p1_evening_minute_ddm["menu"].config(borderwidth=0,font=("Helvetica","9","bold"),fg="#043b82",bg="white",\
    activeforeground="white",activebackground="#00a047")
p1_evening_minute_ddm.pack()

p2_evening_minute_ddm=OptionMenu(eveMIN_frame,p2_evening_minute,*time_minute_list)
p2_evening_minute_ddm.config(borderwidth=0,width=1,font=("Helvetica","12","bold"),fg="#043b82",bg="white")
p2_evening_minute_ddm["menu"].config(borderwidth=0,font=("Helvetica","9","bold"),fg="#043b82",bg="white",\
    activeforeground="white",activebackground="#00a047")
p2_evening_minute_ddm.pack()

p3_evening_minute_ddm=OptionMenu(eveMIN_frame,p3_evening_minute,*time_minute_list)
p3_evening_minute_ddm.config(borderwidth=0,width=1,font=("Helvetica","12","bold"),fg="#043b82",bg="white")
p3_evening_minute_ddm["menu"].config(borderwidth=0,font=("Helvetica","9","bold"),fg="#043b82",bg="white",\
    activeforeground="white",activebackground="#00a047")
p3_evening_minute_ddm.pack()

p4_evening_minute_ddm=OptionMenu(eveMIN_frame,p4_evening_minute,*time_minute_list)
p4_evening_minute_ddm.config(borderwidth=0,width=1,font=("Helvetica","12","bold"),fg="#043b82",bg="white")
p4_evening_minute_ddm["menu"].config(borderwidth=0,font=("Helvetica","9","bold"),fg="#043b82",bg="white",\
    activeforeground="white",activebackground="#00a047")
p4_evening_minute_ddm.pack()

p5_evening_minute_ddm=OptionMenu(eveMIN_frame,p5_evening_minute,*time_minute_list)
p5_evening_minute_ddm.config(borderwidth=0,width=1,font=("Helvetica","12","bold"),fg="#043b82",bg="white")
p5_evening_minute_ddm["menu"].config(borderwidth=0,font=("Helvetica","9","bold"),fg="#043b82",bg="white",\
    activeforeground="white",activebackground="#00a047")
p5_evening_minute_ddm.pack()

#placing the evening AM/PM column in the evening section
eveAMPM_frame=LabelFrame(TSevening_frame,borderwidth=0,bg="#043b82")

```

```

eveAMPM_frame.pack(side=LEFT)

#creating and placing drop down menus for AM/PM in the evening section
p1_evening_ampm_ddm=OptionMenu(eveAMPM_frame,p1_evening_ampm,*time_ampm_list)
p1_evening_ampm_ddm.config(borderwidth=0,width=2,font=("Helvetica", "12", "bold"),fg="#043b82",bg="white")
p1_evening_ampm_ddm["menu"].config(borderwidth=0,font=("Helvetica", "9", "bold"),fg="#043b82",bg="white",\
    activeforeground="white",activebackground="#00a047")
p1_evening_ampm_ddm.pack()

p2_evening_ampm_ddm=OptionMenu(eveAMPM_frame,p2_evening_ampm,*time_ampm_list)
p2_evening_ampm_ddm.config(borderwidth=0,width=2,font=("Helvetica", "12", "bold"),fg="#043b82",bg="white")
p2_evening_ampm_ddm["menu"].config(borderwidth=0,font=("Helvetica", "9", "bold"),fg="#043b82",bg="white",\
    activeforeground="white",activebackground="#00a047")
p2_evening_ampm_ddm.pack()

p3_evening_ampm_ddm=OptionMenu(eveAMPM_frame,p3_evening_ampm,*time_ampm_list)
p3_evening_ampm_ddm.config(borderwidth=0,width=2,font=("Helvetica", "12", "bold"),fg="#043b82",bg="white")
p3_evening_ampm_ddm["menu"].config(borderwidth=0,font=("Helvetica", "9", "bold"),fg="#043b82",bg="white",\
    activeforeground="white",activebackground="#00a047")
p3_evening_ampm_ddm.pack()

p4_evening_ampm_ddm=OptionMenu(eveAMPM_frame,p4_evening_ampm,*time_ampm_list)
p4_evening_ampm_ddm.config(borderwidth=0,width=2,font=("Helvetica", "12", "bold"),fg="#043b82",bg="white")
p4_evening_ampm_ddm["menu"].config(borderwidth=0,font=("Helvetica", "9", "bold"),fg="#043b82",bg="white",\
    activeforeground="white",activebackground="#00a047")
p4_evening_ampm_ddm.pack()

p5_evening_ampm_ddm=OptionMenu(eveAMPM_frame,p5_evening_ampm,*time_ampm_list)
p5_evening_ampm_ddm.config(borderwidth=0,width=2,font=("Helvetica", "12", "bold"),fg="#043b82",bg="white")
p5_evening_ampm_ddm["menu"].config(borderwidth=0,font=("Helvetica", "9", "bold"),fg="#043b82",bg="white",\
    activeforeground="white",activebackground="#00a047")
p5_evening_ampm_ddm.pack()

#submitting user information to database
def submit_local():
    #retrieving most recent data from 'info' table
    my_cursor_local_info.execute("SELECT * FROM info ORDER BY user_id_local DESC LIMIT 1")
    local_recent_update_info=my_cursor_local_info.fetchall()
    #print(local_recent_update_info)

    p1_morning_time_insert=p1_morning_hour.get()+". "+p1_morning_minute.get()+". "+p1_morning_ampm.get()
    p1_afternoon_time_insert=p1_afternoon_hour.get()+". "+p1_afternoon_minute.get()+". "+p1_afternoon_ampm.get()
    p1_evening_time_insert=p1_evening_hour.get()+". "+p1_evening_minute.get()+". "+p1_evening_ampm.get()

    p2_morning_time_insert=p2_morning_hour.get()+". "+p2_morning_minute.get()+". "+p2_morning_ampm.get()
    p2_afternoon_time_insert=p2_afternoon_hour.get()+". "+p2_afternoon_minute.get()+". "+p2_afternoon_ampm.get()
    p2_evening_time_insert=p2_evening_hour.get()+". "+p2_evening_minute.get()+". "+p2_evening_ampm.get()

    p3_morning_time_insert=p3_morning_hour.get()+". "+p3_morning_minute.get()+". "+p3_morning_ampm.get()
    p3_afternoon_time_insert=p3_afternoon_hour.get()+". "+p3_afternoon_minute.get()+". "+p3_afternoon_ampm.get()
    p3_evening_time_insert=p3_evening_hour.get()+". "+p3_evening_minute.get()+". "+p3_evening_ampm.get()

    p4_morning_time_insert=p4_morning_hour.get()+". "+p4_morning_minute.get()+". "+p4_morning_ampm.get()
    p4_afternoon_time_insert=p4_afternoon_hour.get()+". "+p4_afternoon_minute.get()+". "+p4_afternoon_ampm.get()
    p4_evening_time_insert=p4_evening_hour.get()+". "+p4_evening_minute.get()+". "+p4_evening_ampm.get()

    p5_morning_time_insert=p5_morning_hour.get()+". "+p5_morning_minute.get()+". "+p5_morning_ampm.get()
    p5_afternoon_time_insert=p5_afternoon_hour.get()+". "+p5_afternoon_minute.get()+". "+p5_afternoon_ampm.get()
    p5_evening_time_insert=p5_evening_hour.get()+". "+p5_evening_minute.get()+". "+p5_evening_ampm.get()

    user_id_local=local_recent_update_info[0][25]

    #entering the data into the 'info' table
    local_info_submit="""UPDATE info SET \
        p1_morning_time=%s,p1_afternoon_time=%s,p1_evening_time=%s,\
        p2_morning_time=%s,p2_afternoon_time=%s,p2_evening_time=%s,\
        p3_morning_time=%s,p3_afternoon_time=%s,p3_evening_time=%s,\
        p4_morning_time=%s,p4_afternoon_time=%s,p4_evening_time=%s,\
        p5_morning_time=%s,p5_afternoon_time=%s,p5_evening_time=%s \
        WHERE user_id_local=%s"""

    local_values_submit=(p1_morning_time_insert,p1_afternoon_time_insert,p1_evening_time_insert,\
        p2_morning_time_insert,p2_afternoon_time_insert,p2_evening_time_insert,\
        p3_morning_time_insert,p3_afternoon_time_insert,p3_evening_time_insert,\
        p4_morning_time_insert,p4_afternoon_time_insert,p4_evening_time_insert,\
        p5_morning_time_insert,p5_afternoon_time_insert,p5_evening_time_insert,user_id_local)

```

```

#committing changes to the 'local_info' database
my_cursor_local_info.execute(local_info_submit,local_values_submit)
mydb_local_info.commit()

#button to save information regarding when to take pills
TSbtn_frame=LabelFrame(top,borderwidth=0,bg="#043b82")
TSbtn_frame.pack(padx=40,pady=20,fill=BOTH,expand=True)

next_window_button=Button(TSbtn_frame,text="Save information",\
    command=Lambda:[create_new_user_window(),submit_local(),top.destroy()],\
    font=("Helvetica","10","bold italic"),fg="#043b82",bg="white",activeforeground="white",activebackground="#00a047")
next_window_button.pack(side=BOTTOM,fill=X)

def create_new_user_window():
    new_user=Tk()
    new_user.geometry("800x480")
    new_user.overridereirect(True)
    new_user.configure(bg="#043b82")

#function to calculate the total amount of pills
def add_total_pill_amount(*args):
    p1_total=int(p1_morning_var.get()+int(p1_afternoon_var.get()+int(p1_evening_var.get()))
    p2_total=int(p2_morning_var.get()+int(p2_afternoon_var.get()+int(p2_evening_var.get()))
    p3_total=int(p3_morning_var.get()+int(p3_afternoon_var.get()+int(p3_evening_var.get()))
    p4_total=int(p4_morning_var.get()+int(p4_afternoon_var.get()+int(p4_evening_var.get()))
    p5_total=int(p5_morning_var.get()+int(p5_afternoon_var.get()+int(p5_evening_var.get()))

    p1_total_label=Label(NUtotal_frame,text=str(p1_total),font=("Helvetica","12","bold"),fg="white",bg="#043b82")
    p1_total_label.place(anchor=N,relx=0.5,relly=0.17)

    p2_total_label=Label(NUtotal_frame,text=str(p2_total),font=("Helvetica","12","bold"),fg="white",bg="#043b82")
    p2_total_label.place(anchor=N,relx=0.5,relly=0.35)

    p3_total_label=Label(NUtotal_frame,text=str(p3_total),font=("Helvetica","12","bold"),fg="white",bg="#043b82")
    p3_total_label.place(anchor=N,relx=0.5,relly=0.52)

    p4_total_label=Label(NUtotal_frame,text=str(p4_total),font=("Helvetica","12","bold"),fg="white",bg="#043b82")
    p4_total_label.place(anchor=N,relx=0.5,relly=0.68)

    p5_total_label=Label(NUtotal_frame,text=str(p5_total),font=("Helvetica","12","bold"),fg="white",bg="#043b82")
    p5_total_label.place(anchor=N,relx=0.5,relly=0.86)

#submitting user information to database
def submit():
    check=StringVar()
    check=0
    #USE THIS TO GET THE MOST RECENT USER CREATED
    my_cursor.execute("SELECT * FROM users ORDER BY user_id ASC LIMIT 4")
    most_recent_user_check=my_cursor.fetchall()

    #doing math
    p1_total_insert=int(p1_morning_var.get()+int(p1_afternoon_var.get()+int(p1_evening_var.get()))
    p2_total_insert=int(p2_morning_var.get()+int(p2_afternoon_var.get()+int(p2_evening_var.get()))
    p3_total_insert=int(p3_morning_var.get()+int(p3_afternoon_var.get()+int(p3_evening_var.get()))
    p4_total_insert=int(p4_morning_var.get()+int(p4_afternoon_var.get()+int(p4_evening_var.get()))
    p5_total_insert=int(p5_morning_var.get()+int(p5_afternoon_var.get()+int(p5_evening_var.get()))

    for x in range(1,4):
        if most_recent_user_check[x-1][0]=="Start_name":
            sql_command=""UPDATE users SET name_entry=%s,\
                p1_entry=%s,p2_entry=%s,p3_entry=%s,p4_entry=%s,p5_entry=%s,\
                p1_morning=%s,p1_afternoon=%s,p1_evening=%s,\
                p2_morning=%s,p2_afternoon=%s,p2_evening=%s,\
                p3_morning=%s,p3_afternoon=%s,p3_evening=%s,\
                p4_morning=%s,p4_afternoon=%s,p4_evening=%s,\
                p5_morning=%s,p5_afternoon=%s,p5_evening=%s WHERE user_id=%s""

            values=(NUname_entry.get(),p1_total_insert,p2_total_insert,p3_total_insert,p4_total_insert,p5_total_insert,\
                p1_morning_var.get(),p1_afternoon_var.get(),p1_evening_var.get(),\
                p2_morning_var.get(),p2_afternoon_var.get(),p2_evening_var.get(),\
                p3_morning_var.get(),p3_afternoon_var.get(),p3_evening_var.get(),\
                p4_morning_var.get(),p4_afternoon_var.get(),p4_evening_var.get(),\
                p5_morning_var.get(),p5_afternoon_var.get(),p5_evening_var.get(),x)

            my_cursor.execute(sql_command,values)

```

```

        mydb_user_info.commit()
        check=1
        if check==1:
            break

#new window to list users
def list_users(name_list):
    list_users_query=Tk()
    list_users_query.geometry("800x480")
    list_users_query.overrideredirect(True)
    list_users_query.configure(bg="#043b82")

    list_name="SELECT * FROM users WHERE name_entry=%s"
    list_name_values=(name_list, )

    most_recent_user_show=my_cursor.execute(list_name,list_name_values)
    most_recent_user_show=my_cursor.fetchall()

    LQtitle_label=Label(list_users_query,text="Review user information",font=("Helvetica", "22", "bold"),fg="white",
    LQtitle_label.pack()

    LQuser_label=Label(list_users_query,text="User: "+str(most_recent_user_show[0][0]),\
        font=("Helvetica", "16", "bold"),fg="white",bg="#043b82")
    LQuser_label.pack(side=TOP,pady=30)

    LQ_frame=LabelFrame(list_users_query,borderwidth=0,bg="#043b82")
    LQ_frame.pack()

    #placing the pills column in the window
    LQpill_frame=LabelFrame(LQ_frame,borderwidth=0,bg="#043b82")
    LQpill_frame.pack(side=LEFT,padx=20)

    LQpill_label=Label(LQpill_frame,text="Pills",font=("Helvetica", "16", "bold"),fg="white",bg="#043b82")
    LQpill_label.pack(side=TOP,fill=X,expand=True,pady=10)

    p1LQ_label=Label(LQpill_frame,text=str(most_local_info[0][0]),font=("Helvetica", "16", "italic"),fg="white",bg='
    p1LQ_label.pack()

    p2LQ_label=Label(LQpill_frame,text=str(most_local_info[0][1]),font=("Helvetica", "16", "italic"),fg="white",bg='
    p2LQ_label.pack()

    p3LQ_label=Label(LQpill_frame,text=str(most_local_info[0][2]),font=("Helvetica", "16", "italic"),fg="white",bg='
    p3LQ_label.pack()

    p4LQ_label=Label(LQpill_frame,text=str(most_local_info[0][3]),font=("Helvetica", "16", "italic"),fg="white",bg='
    p4LQ_label.pack()

    p5LQ_label=Label(LQpill_frame,text=str(most_local_info[0][4]),font=("Helvetica", "16", "italic"),fg="white",bg='
    p5LQ_label.pack()

    #placing the morning column in the window
    LQmorn_frame=LabelFrame(LQ_frame,borderwidth=0,bg="#043b82")
    LQmorn_frame.pack(side=LEFT,padx=20)

    LQmorn_label=Label(LQmorn_frame,text="Morning",font=("Helvetica", "16", "bold"),fg="white",bg="#043b82")
    LQmorn_label.pack(side=TOP,fill=X,expand=True,pady=10)

    p1_LQmorn_label=Label(LQmorn_frame,text=str(most_recent_user_show[0][6]),font=("Helvetica", "16", "bold"),fg="w
    p1_LQmorn_label.pack()

    p2_LQmorn_label=Label(LQmorn_frame,text=str(most_recent_user_show[0][9]),font=("Helvetica", "16", "bold"),fg="w
    p2_LQmorn_label.pack()

    p3_LQmorn_label=Label(LQmorn_frame,text=str(most_recent_user_show[0][12]),font=("Helvetica", "16", "bold"),fg="t
    p3_LQmorn_label.pack()

    p4_LQmorn_label=Label(LQmorn_frame,text=str(most_recent_user_show[0][15]),font=("Helvetica", "16", "bold"),fg="t
    p4_LQmorn_label.pack()

    p5_LQmorn_label=Label(LQmorn_frame,text=str(most_recent_user_show[0][18]),font=("Helvetica", "16", "bold"),fg="t
    p5_LQmorn_label.pack()

    #placing the afternoon column in the window
    LQnoon_frame=LabelFrame(LQ_frame,borderwidth=0,bg="#043b82")
    LQnoon_frame.pack(side=LEFT,padx=20)

```

```

LQnoon_label=Label(LQnoon_frame,text="Afternoon",font=("Helvetica", "16", "bold"),fg="white",bg="#043b82")
LQnoon_label.pack(side=TOP,fill=X,expand=True,pady=10)

p1_LQnoon_label=Label(LQnoon_frame,text=str(most_recent_user_show[0][7]),font=("Helvetica", "16", "bold"),fg="white",bg="#043b82")
p1_LQnoon_label.pack()

p2_LQnoon_label=Label(LQnoon_frame,text=str(most_recent_user_show[0][10]),font=("Helvetica", "16", "bold"),fg="white",bg="#043b82")
p2_LQnoon_label.pack()

p3_LQnoon_label=Label(LQnoon_frame,text=str(most_recent_user_show[0][13]),font=("Helvetica", "16", "bold"),fg="white",bg="#043b82")
p3_LQnoon_label.pack()

p4_LQnoon_label=Label(LQnoon_frame,text=str(most_recent_user_show[0][16]),font=("Helvetica", "16", "bold"),fg="white",bg="#043b82")
p4_LQnoon_label.pack()

p5_LQnoon_label=Label(LQnoon_frame,text=str(most_recent_user_show[0][19]),font=("Helvetica", "16", "bold"),fg="white",bg="#043b82")
p5_LQnoon_label.pack()

#placing the evening column in the window
LQeve_frame=LabelFrame(LQ_frame,borderwidth=0,bg="#043b82")
LQeve_frame.pack(side=LEFT,padx=20)

LQeve_label=Label(LQeve_frame,text="Evening",font=("Helvetica", "16", "bold"),fg="white",bg="#043b82")
LQeve_label.pack(side=TOP,fill=X,expand=True,pady=10)

p1_LQeve_label=Label(LQeve_frame,text=str(most_recent_user_show[0][8]),font=("Helvetica", "16", "bold"),fg="white",bg="#043b82")
p1_LQeve_label.pack()

p2_LQeve_label=Label(LQeve_frame,text=str(most_recent_user_show[0][11]),font=("Helvetica", "16", "bold"),fg="white",bg="#043b82")
p2_LQeve_label.pack()

p3_LQeve_label=Label(LQeve_frame,text=str(most_recent_user_show[0][14]),font=("Helvetica", "16", "bold"),fg="white",bg="#043b82")
p3_LQeve_label.pack()

p4_LQeve_label=Label(LQeve_frame,text=str(most_recent_user_show[0][17]),font=("Helvetica", "16", "bold"),fg="white",bg="#043b82")
p4_LQeve_label.pack()

p5_LQeve_label=Label(LQeve_frame,text=str(most_recent_user_show[0][20]),font=("Helvetica", "16", "bold"),fg="white",bg="#043b82")
p5_LQeve_label.pack()

#placing the total column in the window
LQtotal_frame=LabelFrame(LQ_frame,borderwidth=0,bg="#043b82")
LQtotal_frame.pack(side=LEFT,padx=20)

LQtotal_label=Label(LQtotal_frame,text="Total",font=("Helvetica", "16", "bold"),fg="white",bg="#043b82")
LQtotal_label.pack(side=TOP,fill=X,expand=True,pady=10)

p1_LQtotal_label=Label(LQtotal_frame,text=str(most_recent_user_show[0][1]),font=("Helvetica", "16", "bold"),fg="white",bg="#043b82")
p1_LQtotal_label.pack()

p2_LQtotal_label=Label(LQtotal_frame,text=str(most_recent_user_show[0][2]),font=("Helvetica", "16", "bold"),fg="white",bg="#043b82")
p2_LQtotal_label.pack()

p3_LQtotal_label=Label(LQtotal_frame,text=str(most_recent_user_show[0][3]),font=("Helvetica", "16", "bold"),fg="white",bg="#043b82")
p3_LQtotal_label.pack()

p4_LQtotal_label=Label(LQtotal_frame,text=str(most_recent_user_show[0][4]),font=("Helvetica", "16", "bold"),fg="white",bg="#043b82")
p4_LQtotal_label.pack()

p5_LQtotal_label=Label(LQtotal_frame,text=str(most_recent_user_show[0][5]),font=("Helvetica", "16", "bold"),fg="white",bg="#043b82")
p5_LQtotal_label.pack()

#placing the buttons in the window
LQbtn_frame=LabelFrame(list_users_query,borderwidth=0,bg="#043b82")
LQbtn_frame.pack(side=BOTTOM,padx=40,pady=20)

#button to edit the data given in this window
edit_data_button=Button(LQbtn_frame,text="Edit user information",\
    command=Lambda:[edit_user_info(str(most_recent_user_show[0][0])),list_users_query.destroy()],\
    width=30,font=("Helvetica", "10", "bold italic"),fg="#043b82",bg="white",activeforeground="white",activebackground="white")
edit_data_button.pack(side=LEFT,padx=40)

#button to save the data given in this window
save_data_button=Button(LQbtn_frame,text="Save user information",command=Lambda:[list_users_query.destroy()],\
    width=30,font=("Helvetica", "10", "bold italic"),fg="#043b82",bg="white",activeforeground="white",activebackground="white")
save_data_button.pack(side=RIGHT,padx=40)

```

```

#commit changes to database
mydb_user_info.commit()
create_user_buttons()

#new window to edit user info
def edit_user_info(name_edit):
    edit_user=Tk()
    edit_user.geometry("800x480")
    edit_user.overrideredirect("True")
    edit_user.configure(bg="#043b82")

def update_user_info():
    sql="""UPDATE users SET name_entry=%s,\
        p1_entry=%s,p2_entry=%s,p3_entry=%s,p4_entry=%s,p5_entry=%s,\
        p1_morning=%s,p1_afternoon=%s,p1_evening=%s,\
        p2_morning=%s,p2_afternoon=%s,p2_evening=%s,\
        p3_morning=%s,p3_afternoon=%s,p3_evening=%s,\
        p4_morning=%s,p4_afternoon=%s,p4_evening=%s,\
        p5_morning=%s,p5_afternoon=%s,p5_evening=%s \
        WHERE user_id=%s """

    name_entry=EUname_entry.get()
    p1_entry=int(p1_morning_var_edit.get()+int(p1_afternoon_var_edit.get()+int(p1_evening_var_edit.get()
    p2_entry=int(p2_morning_var_edit.get()+int(p2_afternoon_var_edit.get()+int(p2_evening_var_edit.get()
    p3_entry=int(p3_morning_var_edit.get()+int(p3_afternoon_var_edit.get()+int(p3_evening_var_edit.get()
    p4_entry=int(p4_morning_var_edit.get()+int(p4_afternoon_var_edit.get()+int(p4_evening_var_edit.get()
    p5_entry=int(p5_morning_var_edit.get()+int(p5_afternoon_var_edit.get()+int(p5_evening_var_edit.get()
    p1_morning=p1_morning_var_edit.get()
    p2_morning=p2_morning_var_edit.get()
    p3_morning=p3_morning_var_edit.get()
    p4_morning=p4_morning_var_edit.get()
    p5_morning=p5_morning_var_edit.get()
    p1_afternoon=p1_afternoon_var_edit.get()
    p2_afternoon=p2_afternoon_var_edit.get()
    p3_afternoon=p3_afternoon_var_edit.get()
    p4_afternoon=p4_afternoon_var_edit.get()
    p5_afternoon=p5_afternoon_var_edit.get()
    p1_evening=p1_evening_var_edit.get()
    p2_evening=p2_evening_var_edit.get()
    p3_evening=p3_evening_var_edit.get()
    p4_evening=p4_evening_var_edit.get()
    p5_evening=p5_evening_var_edit.get()
    user_id=most_recent_user_show[0][21]

    values=(name_entry,p1_entry,p2_entry,p3_entry,p4_entry,p5_entry,\
        p1_morning,p1_afternoon,p1_evening,\
        p2_morning,p2_afternoon,p2_evening,\
        p3_morning,p3_afternoon,p3_evening,\
        p4_morning,p4_afternoon,p4_evening,\
        p5_morning,p5_afternoon,p5_evening, user_id)

    my_cursor.execute(sql,values)
    mydb_user_info.commit()

def show_edit(*args):
    p1_total_show=int(p1_morning_var_edit.get()+int(p1_afternoon_var_edit.get()+int(p1_evening_var_edit
    p2_total_show=int(p2_morning_var_edit.get()+int(p2_afternoon_var_edit.get()+int(p2_evening_var_edit
    p3_total_show=int(p3_morning_var_edit.get()+int(p3_afternoon_var_edit.get()+int(p3_evening_var_edit
    p4_total_show=int(p4_morning_var_edit.get()+int(p4_afternoon_var_edit.get()+int(p4_evening_var_edit
    p5_total_show=int(p5_morning_var_edit.get()+int(p5_afternoon_var_edit.get()+int(p5_evening_var_edit

    p1_total_label_edit=Label(EUtotal_frame,text=str(p1_total_show),font=("Helvetica", "12", "bold"),fg="whi
    p1_total_label_edit.place(anchor=N,relx=0.5,relx=0.17)

    p2_total_label_edit=Label(EUtotal_frame,text=str(p2_total_show),font=("Helvetica", "12", "bold"),fg="whi
    p2_total_label_edit.place(anchor=N,relx=0.5,relx=0.35)

    p3_total_label_edit=Label(EUtotal_frame,text=str(p3_total_show),font=("Helvetica", "12", "bold"),fg="whi
    p3_total_label_edit.place(anchor=N,relx=0.5,relx=0.52)

    p4_total_label_edit=Label(EUtotal_frame,text=str(p4_total_show),font=("Helvetica", "12", "bold"),fg="whi
    p4_total_label_edit.place(anchor=N,relx=0.5,relx=0.68)

    p5_total_label_edit=Label(EUtotal_frame,text=str(p5_total_show),font=("Helvetica", "12", "bold"),fg="whi
    p5_total_label_edit.place(anchor=N,relx=0.5,relx=0.86)

```

```

'''
#getting most recently created user
my_cursor.execute("SELECT * FROM users ORDER BY user_id DESC LIMIT 1")
most_recent_user=my_cursor.fetchall()
'''

edit_name="SELECT * FROM users WHERE name_entry=%s"
edit_name_values=(name_list,)

most_recent_user=my_cursor.execute(edit_name,edit_name_values)
most_recent_user=my_cursor.fetchall()

EUtitle_label=Label(edit_user,text="Edit User",font=("Helvetica","22","bold"),fg="white",bg="#043b82")
EUtitle_label.pack()

editUser_frame=LabelFrame(edit_user,borderwidth=0,bg="#043b82")
editUser_frame.pack()

#creating and placing a section to edit the username
EUsname_frame=LabelFrame(editUser_frame,borderwidth=0,bg="#043b82",padx=100)
EUsname_frame.pack(side=TOP,fill=X,expand=True,pady=10)

EUsname_label=Label(EUsname_frame,text="Enter your full name: ",font=("Helvetica","16","bold"),fg="white",bg="#043b82")
EUsname_label.pack(side=LEFT)

EUsname_entry=Entry(EUsname_frame,borderwidth=0,width=30,font=("Helvetica","12","bold"),fg="#043b82",bg="white")
#EUsname_entry.bind("<Button-1>",openkeyboard)
EUsname_entry.pack(side=LEFT,padx=10)
EUsname_entry.insert(0,most_recent_user[0][0])

#creating and placing the main body of the window
EUmain_frame=LabelFrame(editUser_frame,borderwidth=0,bg="#043b82")
EUmain_frame.pack(side=TOP,fill=X,expand=True)

#placing the pill column in the window
EUpill_frame=LabelFrame(EUmain_frame,borderwidth=0,bg="#043b82")
EUpill_frame.pack(side=LEFT,fill=BOTH,expand=True,padx=10)

EUsdosage_label=Label(EUpill_frame,text="Check dosage:",font=("Helvetica","16","bold"),fg="white",bg="#043b82")
EUsdosage_label.pack(side=TOP,fill=X)

p1EUs_label=Label(EUpill_frame,text=most_local_info[0][0],font=("Helvetica","16","italic"),fg="white",bg="#043b82")
p1EUs_label.pack()

p2EUs_label=Label(EUpill_frame,text=most_local_info[0][1],font=("Helvetica","16","italic"),fg="white",bg="#043b82")
p2EUs_label.pack()

p3EUs_label=Label(EUpill_frame,text=most_local_info[0][2],font=("Helvetica","16","italic"),fg="white",bg="#043b82")
p3EUs_label.pack()

p4EUs_label=Label(EUpill_frame,text=most_local_info[0][3],font=("Helvetica","16","italic"),fg="white",bg="#043b82")
p4EUs_label.pack()

p5EUs_label=Label(EUpill_frame,text=most_local_info[0][4],font=("Helvetica","16","italic"),fg="white",bg="#043b82")
p5EUs_label.pack()

#list for the drop down menus
Number_list_edit=["0","1","2","3","4","5","6","7","8","9"]

#variables for the drop down menus
#setting up variables for the morning
p1_morning_var_edit=StringVar(edit_user)
p1_morning_var_edit.set(Number_list_edit[0])

p2_morning_var_edit=StringVar(edit_user)
p2_morning_var_edit.set(Number_list_edit[0])

p3_morning_var_edit=StringVar(edit_user)
p3_morning_var_edit.set(Number_list_edit[0])

p4_morning_var_edit=StringVar(edit_user)
p4_morning_var_edit.set(Number_list_edit[0])

p5_morning_var_edit=StringVar(edit_user)
p5_morning_var_edit.set(Number_list_edit[0])

```

```

#setting up variables for the afternoon
p1_afternoon_var_edit=StringVar(edit_user)
p1_afternoon_var_edit.set(Number_list_edit[0])

p2_afternoon_var_edit=StringVar(edit_user)
p2_afternoon_var_edit.set(Number_list_edit[0])

p3_afternoon_var_edit=StringVar(edit_user)
p3_afternoon_var_edit.set(Number_list_edit[0])

p4_afternoon_var_edit=StringVar(edit_user)
p4_afternoon_var_edit.set(Number_list_edit[0])

p5_afternoon_var_edit=StringVar(edit_user)
p5_afternoon_var_edit.set(Number_list_edit[0])

#setting up variables for the evening
p1_evening_var_edit=StringVar(edit_user)
p1_evening_var_edit.set(Number_list_edit[0])

p2_evening_var_edit=StringVar(edit_user)
p2_evening_var_edit.set(Number_list_edit[0])

p3_evening_var_edit=StringVar(edit_user)
p3_evening_var_edit.set(Number_list_edit[0])

p4_evening_var_edit=StringVar(edit_user)
p4_evening_var_edit.set(Number_list_edit[0])

p5_evening_var_edit=StringVar(edit_user)
p5_evening_var_edit.set(Number_list_edit[0])

#placing the morning column in the window
EUmorn_frame=LabelFrame(EUmain_frame,borderwidth=0,bg="#043b82")
EUmorn_frame.pack(side=LEFT, padx=10)

EUmorn_label=Label(EUmorn_frame,text="Morning",font=("Helvetica", "16", "bold"),fg="white",bg="#043b82")
EUmorn_label.pack(side=TOP, fill=X)

#creating and placing drop down menus for the morning
p1_morning_edit=OptionMenu(EUmorn_frame,p1_morning_var_edit,*Number_list_edit)
p1_morning_edit.config(borderwidth=0,width=1,font=("Helvetica", "12", "bold"),fg="#043b82",bg="white")
p1_morning_edit["menu"].config(borderwidth=0,font=("Helvetica", "9", "bold"),fg="#043b82",bg="white",\
    activeforeground="white",activebackground="#00a047")
p1_morning_edit.pack()

p2_morning_edit=OptionMenu(EUmorn_frame,p2_morning_var_edit,*Number_list_edit)
p2_morning_edit.config(borderwidth=0,width=1,font=("Helvetica", "12", "bold"),fg="#043b82",bg="white")
p2_morning_edit["menu"].config(borderwidth=0,font=("Helvetica", "9", "bold"),fg="#043b82",bg="white",\
    activeforeground="white",activebackground="#00a047")
p2_morning_edit.pack()

p3_morning_edit=OptionMenu(EUmorn_frame,p3_morning_var_edit,*Number_list_edit)
p3_morning_edit.config(borderwidth=0,width=1,font=("Helvetica", "12", "bold"),fg="#043b82",bg="white")
p3_morning_edit["menu"].config(borderwidth=0,font=("Helvetica", "9", "bold"),fg="#043b82",bg="white",\
    activeforeground="white",activebackground="#00a047")
p3_morning_edit.pack()

p4_morning_edit=OptionMenu(EUmorn_frame,p4_morning_var_edit,*Number_list_edit)
p4_morning_edit.config(borderwidth=0,width=1,font=("Helvetica", "12", "bold"),fg="#043b82",bg="white")
p4_morning_edit["menu"].config(borderwidth=0,font=("Helvetica", "9", "bold"),fg="#043b82",bg="white",\
    activeforeground="white",activebackground="#00a047")
p4_morning_edit.pack()

p5_morning_edit=OptionMenu(EUmorn_frame,p5_morning_var_edit,*Number_list_edit)
p5_morning_edit.config(borderwidth=0,width=1,font=("Helvetica", "12", "bold"),fg="#043b82",bg="white")
p5_morning_edit["menu"].config(borderwidth=0,font=("Helvetica", "9", "bold"),fg="#043b82",bg="white",\
    activeforeground="white",activebackground="#00a047")
p5_morning_edit.pack()

#placing the afternoon column in the window
EUnoon_frame=LabelFrame(EUmain_frame,borderwidth=0,bg="#043b82")
EUnoon_frame.pack(side=LEFT, padx=10)

EUnoon_label=Label(EUnoon_frame,text="Afternoon",font=("Helvetica", "16", "bold"),fg="white",bg="#043b82")

```

```

EUnoon_label.pack(side=TOP,fill=X)

#creating and placing drop down menus for the afternoon
p1_afternoon_edit=OptionMenu(EUnoon_frame,p1_afternoon_var_edit,*Number_list_edit)
p1_afternoon_edit.config(borderwidth=0,width=1,font=("Helvetica","12","bold"),fg="#043b82",bg="white")
p1_afternoon_edit["menu"].config(borderwidth=0,font=("Helvetica","9","bold"),fg="#043b82",bg="white",\
    activeforeground="white",activebackground="#00a047")
p1_afternoon_edit.pack()

p2_afternoon_edit=OptionMenu(EUnoon_frame,p2_afternoon_var_edit,*Number_list_edit)
p2_afternoon_edit.config(borderwidth=0,width=1,font=("Helvetica","12","bold"),fg="#043b82",bg="white")
p2_afternoon_edit["menu"].config(borderwidth=0,font=("Helvetica","9","bold"),fg="#043b82",bg="white",\
    activeforeground="white",activebackground="#00a047")
p2_afternoon_edit.pack()

p3_afternoon_edit=OptionMenu(EUnoon_frame,p3_afternoon_var_edit,*Number_list_edit)
p3_afternoon_edit.config(borderwidth=0,width=1,font=("Helvetica","12","bold"),fg="#043b82",bg="white")
p3_afternoon_edit["menu"].config(borderwidth=0,font=("Helvetica","9","bold"),fg="#043b82",bg="white",\
    activeforeground="white",activebackground="#00a047")
p3_afternoon_edit.pack()

p4_afternoon_edit=OptionMenu(EUnoon_frame,p4_afternoon_var_edit,*Number_list_edit)
p4_afternoon_edit.config(borderwidth=0,width=1,font=("Helvetica","12","bold"),fg="#043b82",bg="white")
p4_afternoon_edit["menu"].config(borderwidth=0,font=("Helvetica","9","bold"),fg="#043b82",bg="white",\
    activeforeground="white",activebackground="#00a047")
p4_afternoon_edit.pack()

p5_afternoon_edit=OptionMenu(EUnoon_frame,p5_afternoon_var_edit,*Number_list_edit)
p5_afternoon_edit.config(borderwidth=0,width=1,font=("Helvetica","12","bold"),fg="#043b82",bg="white")
p5_afternoon_edit["menu"].config(borderwidth=0,font=("Helvetica","9","bold"),fg="#043b82",bg="white",\
    activeforeground="white",activebackground="#00a047")
p5_afternoon_edit.pack()

#placing the evening column in the window
EUeve_frame=LabelFrame(EUmain_frame,borderwidth=0,bg="#043b82")
EUeve_frame.pack(side=LEFT,padx=10)

EUeve_label=Label(EUeve_frame,text="Morning",font=("Helvetica","16","bold"),fg="white",bg="#043b82")
EUeve_label.pack(side=TOP,fill=X)

#creating and placing drop down menus for the evening
p1_evening_edit=OptionMenu(EUeve_frame,p1_evening_var_edit,*Number_list_edit)
p1_evening_edit.config(borderwidth=0,width=1,font=("Helvetica","12","bold"),fg="#043b82",bg="white")
p1_evening_edit["menu"].config(borderwidth=0,font=("Helvetica","9","bold"),fg="#043b82",bg="white",\
    activeforeground="white",activebackground="#00a047")
p1_evening_edit.pack()

p2_evening_edit=OptionMenu(EUeve_frame,p2_evening_var_edit,*Number_list_edit)
p2_evening_edit.config(borderwidth=0,width=1,font=("Helvetica","12","bold"),fg="#043b82",bg="white")
p2_evening_edit["menu"].config(borderwidth=0,font=("Helvetica","9","bold"),fg="#043b82",bg="white",\
    activeforeground="white",activebackground="#00a047")
p2_evening_edit.pack()

p3_evening_edit=OptionMenu(EUeve_frame,p3_evening_var_edit,*Number_list_edit)
p3_evening_edit.config(borderwidth=0,width=1,font=("Helvetica","12","bold"),fg="#043b82",bg="white")
p3_evening_edit["menu"].config(borderwidth=0,font=("Helvetica","9","bold"),fg="#043b82",bg="white",\
    activeforeground="white",activebackground="#00a047")
p3_evening_edit.pack()

p4_evening_edit=OptionMenu(EUeve_frame,p4_evening_var_edit,*Number_list_edit)
p4_evening_edit.config(borderwidth=0,width=1,font=("Helvetica","12","bold"),fg="#043b82",bg="white")
p4_evening_edit["menu"].config(borderwidth=0,font=("Helvetica","9","bold"),fg="#043b82",bg="white",\
    activeforeground="white",activebackground="#00a047")
p4_evening_edit.pack()

p5_evening_edit=OptionMenu(EUeve_frame,p5_evening_var_edit,*Number_list_edit)
p5_evening_edit.config(borderwidth=0,width=1,font=("Helvetica","12","bold"),fg="#043b82",bg="white")
p5_evening_edit["menu"].config(borderwidth=0,font=("Helvetica","9","bold"),fg="#043b82",bg="white",\
    activeforeground="white",activebackground="#00a047")
p5_evening_edit.pack()

#placing the total column in the window
EUtotal_frame=LabelFrame(EUmain_frame,borderwidth=0,bg="#043b82")
EUtotal_frame.pack(side=RIGHT,fill=Y,expand=True,padx=10)

EUtotal_label=Label(EUtotal_frame,text="Total",font=("Helvetica","16","bold"),fg="white",bg="#043b82")
EUtotal_label.pack(side=TOP)

```

```

#code to update the total amount of pills shown on screen
p1_morning_var_edit.trace('w',show_edit)
p2_morning_var_edit.trace('w',show_edit)
p3_morning_var_edit.trace('w',show_edit)
p4_morning_var_edit.trace('w',show_edit)
p5_morning_var_edit.trace('w',show_edit)
p1_afternoon_var_edit.trace('w',show_edit)
p2_afternoon_var_edit.trace('w',show_edit)
p3_afternoon_var_edit.trace('w',show_edit)
p4_afternoon_var_edit.trace('w',show_edit)
p5_afternoon_var_edit.trace('w',show_edit)
p1_evening_var_edit.trace('w',show_edit)
p2_evening_var_edit.trace('w',show_edit)
p3_evening_var_edit.trace('w',show_edit)
p4_evening_var_edit.trace('w',show_edit)
p5_evening_var_edit.trace('w',show_edit)

#button to save the data given in this window
EUBtn_frame=LabelFrame(edit_user,borderwidth=0,bg="#043b82")
EUBtn_frame.pack(side=BOTTOM,fill=X,padx=60,pady=20)

update_info_button=Button(EUBtn_frame,text="Update user information",\
    command=Lambda:[update_user_info(),list_users(EUname_entry.get()),edit_user.destroy()],\
    font=("Helvetica",10,"bold italic"),fg="#043b82",bg="white",activeforeground="white",activebackground="white")
update_info_button.pack(side=BOTTOM,fill=X)

#2nd window when clicking on "Create User" that is used to set up dosages for the user
NUtitle_label=Label(new_user,text="Create a User",font=("Helvetica",22,"bold"),fg="white",bg="#043b82")
NUtitle_label.pack()

newUser_frame=LabelFrame(new_user,borderwidth=0,bg="#043b82")
newUser_frame.pack()

#creating and placing a section to write the username
NUname_frame=LabelFrame(newUser_frame,borderwidth=0,bg="#043b82",padx=100)
NUname_frame.pack(side=TOP,fill=X,expand=True,pady=10)

NUname_label=Label(NUname_frame,text="Enter your full name:",font=("Helvetica",16,"bold"),fg="white",bg="#043b82")
NUname_label.pack(side=LEFT)

NUname_entry=Entry(NUname_frame,borderwidth=0,width=30,font=("Helvetica",12,"bold"),fg="#043b82",bg="white")
#NUname_entry.bind("<Button-1>",openkeyboard)
NUname_entry.pack(side=LEFT,padx=10)

#creating and placing the main body of the window
NUmain_frame=LabelFrame(newUser_frame,borderwidth=0,bg="#043b82")
NUmain_frame.pack(side=TOP,fill=X,expand=True)

#placing the pills column in the window
NUpill_frame=LabelFrame(NUmain_frame,borderwidth=0,bg="#043b82")
NUpill_frame.pack(side=LEFT,fill=BOTH,expand=True,padx=10)

NUsdosage_label=Label(NUpill_frame,text="Set up dosage:",font=("Helvetica",16,"bold"),fg="white",bg="#043b82")
NUsdosage_label.pack(side=TOP,fill=X)

p1NU_label=Label(NUpill_frame,text=str(most_local_info[0][0]),font=("Helvetica",16,"italic"),fg="white",bg="#043b82")
p1NU_label.pack()

p2NU_label=Label(NUpill_frame,text=str(most_local_info[0][1]),font=("Helvetica",16,"italic"),fg="white",bg="#043b82")
p2NU_label.pack()

p3NU_label=Label(NUpill_frame,text=str(most_local_info[0][2]),font=("Helvetica",16,"italic"),fg="white",bg="#043b82")
p3NU_label.pack()

p4NU_label=Label(NUpill_frame,text=str(most_local_info[0][3]),font=("Helvetica",16,"italic"),fg="white",bg="#043b82")
p4NU_label.pack()

p5NU_label=Label(NUpill_frame,text=str(most_local_info[0][4]),font=("Helvetica",16,"italic"),fg="white",bg="#043b82")
p5NU_label.pack()

#list for the drop down menus
Number_list=["0","1","2","3","4","5","6","7","8","9"]

```

```
#variables for the drop down menus
#setting up variables for the morning
p1_morning_var=StringVar(new_user)
p1_morning_var.set(Number_list[0])

p2_morning_var=StringVar(new_user)
p2_morning_var.set(Number_list[0])

p3_morning_var=StringVar(new_user)
p3_morning_var.set(Number_list[0])

p4_morning_var=StringVar(new_user)
p4_morning_var.set(Number_list[0])

p5_morning_var=StringVar(new_user)
p5_morning_var.set(Number_list[0])

#setting up variables for the afternoon
p1_afternoon_var=StringVar(new_user)
p1_afternoon_var.set(Number_list[0])

p2_afternoon_var=StringVar(new_user)
p2_afternoon_var.set(Number_list[0])

p3_afternoon_var=StringVar(new_user)
p3_afternoon_var.set(Number_list[0])

p4_afternoon_var=StringVar(new_user)
p4_afternoon_var.set(Number_list[0])

p5_afternoon_var=StringVar(new_user)
p5_afternoon_var.set(Number_list[0])

#setting up variables for the evening
p1_evening_var=StringVar(new_user)
p1_evening_var.set(Number_list[0])

p2_evening_var=StringVar(new_user)
p2_evening_var.set(Number_list[0])

p3_evening_var=StringVar(new_user)
p3_evening_var.set(Number_list[0])

p4_evening_var=StringVar(new_user)
p4_evening_var.set(Number_list[0])

p5_evening_var=StringVar(new_user)
p5_evening_var.set(Number_list[0])

#placing the morning section in the window
NUmorn_frame=LabelFrame(NUmain_frame,borderwidth=0,bg="#043b82")
NUmorn_frame.pack(side=LEFT, padx=10)

NUmorn_label=Label(NUmorn_frame,text="Morning",font=("Helvetica","16","bold"),fg="white",bg="#043b82")
NUmorn_label.pack(side=TOP,fill=X)

#creating and placing drop down menus for the morning
p1_morning=OptionMenu(NUmorn_frame,p1_morning_var,*Number_list)
p1_morning.config(borderwidth=0,width=1,font=("Helvetica","12","bold"),fg="#043b82",bg="white")
p1_morning["menu"].config(borderwidth=0,font=("Helvetica","9","bold"),fg="#043b82",bg="white",\
    activeforeground="white",activebackground="#00a047")
p1_morning.pack()

p2_morning=OptionMenu(NUmorn_frame,p2_morning_var,*Number_list)
p2_morning.config(borderwidth=0,width=1,font=("Helvetica","12","bold"),fg="#043b82",bg="white")
p2_morning["menu"].config(borderwidth=0,font=("Helvetica","9","bold"),fg="#043b82",bg="white",\
    activeforeground="white",activebackground="#00a047")
p2_morning.pack()

p3_morning=OptionMenu(NUmorn_frame,p3_morning_var,*Number_list)
p3_morning.config(borderwidth=0,width=1,font=("Helvetica","12","bold"),fg="#043b82",bg="white")
p3_morning["menu"].config(borderwidth=0,font=("Helvetica","9","bold"),fg="#043b82",bg="white",\
    activeforeground="white",activebackground="#00a047")
p3_morning.pack()

p4_morning=OptionMenu(NUmorn_frame,p4_morning_var,*Number_list)
p4_morning.config(borderwidth=0,width=1,font=("Helvetica","12","bold"),fg="#043b82",bg="white")
```

```

p4_morning["menu"].config(borderwidth=0,font=("Helvetica", "9", "bold"),fg="#043b82",bg="white",\
    activeforeground="white",activebackground="#00a047")
p4_morning.pack()

p5_morning=OptionMenu(NUmorn_frame,p5_morning_var,*Number_list)
p5_morning.config(borderwidth=0,width=1,font=("Helvetica", "12", "bold"),fg="#043b82",bg="white")
p5_morning["menu"].config(borderwidth=0,font=("Helvetica", "9", "bold"),fg="#043b82",bg="white",\
    activeforeground="white",activebackground="#00a047")
p5_morning.pack()

#placing the afternoon section in the window
NUnoon_frame=LabelFrame(NUmain_frame,borderwidth=0,bg="#043b82")
NUnoon_frame.pack(side=LEFT, padx=10)

NUnoon_label=Label(NUnoon_frame,text="Afternoon",font=("Helvetica", "16", "bold"),fg="white",bg="#043b82")
NUnoon_label.pack(side=TOP,fill=X)

#creating and placing drop down menus for the afternoon
p1_afternoon=OptionMenu(NUnoon_frame,p1_afternoon_var,*Number_list)
p1_afternoon.config(borderwidth=0,width=1,font=("Helvetica", "12", "bold"),fg="#043b82",bg="white")
p1_afternoon["menu"].config(borderwidth=0,font=("Helvetica", "9", "bold"),fg="#043b82",bg="white",\
    activeforeground="white",activebackground="#00a047")
p1_afternoon.pack()

p2_afternoon=OptionMenu(NUnoon_frame,p2_afternoon_var,*Number_list)
p2_afternoon.config(borderwidth=0,width=1,font=("Helvetica", "12", "bold"),fg="#043b82",bg="white")
p2_afternoon["menu"].config(borderwidth=0,font=("Helvetica", "9", "bold"),fg="#043b82",bg="white",\
    activeforeground="white",activebackground="#00a047")
p2_afternoon.pack()

p3_afternoon=OptionMenu(NUnoon_frame,p3_afternoon_var,*Number_list)
p3_afternoon.config(borderwidth=0,width=1,font=("Helvetica", "12", "bold"),fg="#043b82",bg="white")
p3_afternoon["menu"].config(bg="white", fg="#043b82", font=("Helvetica", "9", "bold"), borderwidth=0)
p3_afternoon.pack()

p4_afternoon=OptionMenu(NUnoon_frame,p4_afternoon_var,*Number_list)
p4_afternoon.config(borderwidth=0,width=1,font=("Helvetica", "12", "bold"),fg="#043b82",bg="white")
p4_afternoon["menu"].config(borderwidth=0,font=("Helvetica", "9", "bold"),fg="#043b82",bg="white",\
    activeforeground="white",activebackground="#00a047")
p4_afternoon.pack()

p5_afternoon=OptionMenu(NUnoon_frame,p5_afternoon_var,*Number_list)
p5_afternoon.config(borderwidth=0,width=1,font=("Helvetica", "12", "bold"),fg="#043b82",bg="white")
p5_afternoon["menu"].config(borderwidth=0,font=("Helvetica", "9", "bold"),fg="#043b82",bg="white",\
    activeforeground="white",activebackground="#00a047")
p5_afternoon.pack()

#placing the evening section in the window
NUeve_frame=LabelFrame(NUmain_frame,borderwidth=0,bg="#043b82")
NUeve_frame.pack(side=LEFT, padx=10)

NUeve_label=Label(NUeve_frame,text="Evening",font=("Helvetica", "16", "bold"),fg="white",bg="#043b82")
NUeve_label.pack(side=TOP,fill=X)

#creating and placing drop down menus for the evening
p1_evening=OptionMenu(NUeve_frame,p1_evening_var,*Number_list)
p1_evening.config(borderwidth=0,width=1,font=("Helvetica", "12", "bold"),fg="#043b82",bg="white")
p1_evening["menu"].config(borderwidth=0,font=("Helvetica", "9", "bold"),fg="#043b82",bg="white",\
    activeforeground="white",activebackground="#00a047")
p1_evening.pack()

p2_evening=OptionMenu(NUeve_frame,p2_evening_var,*Number_list)
p2_evening.config(borderwidth=0,width=1,font=("Helvetica", "12", "bold"),fg="#043b82",bg="white")
p2_evening["menu"].config(borderwidth=0,font=("Helvetica", "9", "bold"),fg="#043b82",bg="white",\
    activeforeground="white",activebackground="#00a047")
p2_evening.pack()

p3_evening=OptionMenu(NUeve_frame,p3_evening_var,*Number_list)
p3_evening.config(borderwidth=0,width=1,font=("Helvetica", "12", "bold"),fg="#043b82",bg="white")
p3_evening["menu"].config(borderwidth=0,font=("Helvetica", "9", "bold"),fg="#043b82",bg="white",\
    activeforeground="white",activebackground="#00a047")
p3_evening.pack()

p4_evening=OptionMenu(NUeve_frame,p4_evening_var,*Number_list)
p4_evening.config(borderwidth=0,width=1,font=("Helvetica", "12", "bold"),fg="#043b82",bg="white")
p4_evening["menu"].config(borderwidth=0,font=("Helvetica", "9", "bold"),fg="#043b82",bg="white",\
    activeforeground="white",activebackground="#00a047")

```

```

p4_evening.pack()

p5_evening=OptionMenu(NUeve_frame,p5_evening_var,*Number_list)
p5_evening.config(borderwidth=0,width=1,font=("Helvetica","12","bold"),fg="#043b82",bg="white")
p5_evening["menu"].config(borderwidth=0,font=("Helvetica","9","bold"),fg="#043b82",bg="white",\
activeforeground="white",activebackground="#00a047")
p5_evening.pack()

#placing the total column in the window
NUtotal_frame=LabelFrame(NUmain_frame,borderwidth=0,bg="#043b82")
NUtotal_frame.pack(side=RIGHT,fill=Y,expand=True,padx=10)

NUtotal_label=Label(NUtotal_frame,text="Total",font=("Helvetica","16","bold"),fg="white",bg="#043b82")
NUtotal_label.pack(side=TOP)

#code to update the total amount of pills shown on screen
p1_morning_var.trace('w',add_total_pill_amount)
p2_morning_var.trace('w',add_total_pill_amount)
p3_morning_var.trace('w',add_total_pill_amount)
p4_morning_var.trace('w',add_total_pill_amount)
p5_morning_var.trace('w',add_total_pill_amount)
p1_afternoon_var.trace('w',add_total_pill_amount)
p2_afternoon_var.trace('w',add_total_pill_amount)
p3_afternoon_var.trace('w',add_total_pill_amount)
p4_afternoon_var.trace('w',add_total_pill_amount)
p5_afternoon_var.trace('w',add_total_pill_amount)
p1_evening_var.trace('w',add_total_pill_amount)
p2_evening_var.trace('w',add_total_pill_amount)
p3_evening_var.trace('w',add_total_pill_amount)
p4_evening_var.trace('w',add_total_pill_amount)
p5_evening_var.trace('w',add_total_pill_amount)

#button to save the data given in this window
NUbtn_frame=LabelFrame(new_user,borderwidth=0,bg="#043b82")
NUbtn_frame.pack(side=BOTTOM,fill=X,padx=60,pady=20)

check_info_button=Button(NUbtn_frame,text="Save user information",\
command=Lambda:[submit(),list_users(NUname_entry.get()),new_user.destroy()],\
font=("Helvetica","10","bold italic"),fg="#043b82",bg="white",activeforeground="white",activebackground="#00a047")
check_info_button.pack(side=BOTTOM,fill=X)

#creating user profiles (shown when clicked on relevant button on the homescreen)
def show_user_window(user_name_show):
    show_user=Tk()
    show_user.geometry("800x480")
    show_user.overrideredirect(True)
    show_user.configure(bg="#043b82")

    name="SELECT * FROM users WHERE name_entry=%s"
    name_value=(user_name_show, )

    #getting user information
    search_result_user=my_cursor.execute(name,name_value)
    search_result_user=my_cursor.fetchall()

    #code to get the most recent local_info inputted
    my_cursor_local_info.execute("SELECT * FROM info ORDER BY user_id_local DESC LIMIT 1")
    search_result_local=my_cursor_local_info.fetchall()

    showUser_label=Label(show_user,text="User profile of "+str(search_result_user[0][0]),font=("Helvetica","22","bold"),fg="white",bg="#043b82")
    showUser_label.pack(pady=10)

    showUser_frame=LabelFrame(show_user,borderwidth=0,bg="#043b82")
    showUser_frame.pack(pady=10,fill=BOTH,expand=True,padx=40)

    #placing the pills column in the window
    SUPill_frame=LabelFrame(showUser_frame,borderwidth=0,bg="#043b82")
    SUPill_frame.pack(side=LEFT,fill=X,expand=True,padx=20)

    SUDosage_label=Label(SUPill_frame,text=" ",font=("Helvetica","16","bold"),fg="white",bg="#043b82")
    SUDosage_label.pack(side=TOP,fill=X,expand=True,pady=10)

    p1SU_label=Label(SUPill_frame,text=str(search_result_local[0][0]),font=("Helvetica","16","italic"),fg="white",bg="#043b82")

```

```
p1SU_label.pack()

p2SU_label=Label(SUpill_frame,text=str(search_result_local[0][1]),font=("Helvetica","16","italic"),fg="white",bg="#043b82")
p2SU_label.pack()

p3SU_label=Label(SUpill_frame,text=str(search_result_local[0][2]),font=("Helvetica","16","italic"),fg="white",bg="#043b82")
p3SU_label.pack()

p4SU_label=Label(SUpill_frame,text=str(search_result_local[0][3]),font=("Helvetica","16","italic"),fg="white",bg="#043b82")
p4SU_label.pack()

p5SU_label=Label(SUpill_frame,text=str(search_result_local[0][4]),font=("Helvetica","16","italic"),fg="white",bg="#043b82")
p5SU_label.pack()

#placing the morning column in the window
SUMorn_frame=LabelFrame(showUser_frame,borderwidth=0,bg="#043b82")
SUMorn_frame.pack(side=LEFT,padx=20)

SUMorn_label=Label(SUMorn_frame,text="Morning",font=("Helvetica","16","bold"),fg="white",bg="#043b82")
SUMorn_label.pack(side=TOP,fill=X,expand=True,pady=10)

p1_SUMorn_label=Label(SUMorn_frame,text=str(search_result_user[0][6]),font=("Helvetica","16","bold"),fg="white",bg="#043b82")
p1_SUMorn_label.pack()

p2_SUMorn_label=Label(SUMorn_frame,text=str(search_result_user[0][9]),font=("Helvetica","16","bold"),fg="white",bg="#043b82")
p2_SUMorn_label.pack()

p3_SUMorn_label=Label(SUMorn_frame,text=str(search_result_user[0][12]),font=("Helvetica","16","bold"),fg="white",bg="#043b82")
p3_SUMorn_label.pack()

p4_SUMorn_label=Label(SUMorn_frame,text=str(search_result_user[0][15]),font=("Helvetica","16","bold"),fg="white",bg="#043b82")
p4_SUMorn_label.pack()

p5_SUMorn_label=Label(SUMorn_frame,text=str(search_result_user[0][18]),font=("Helvetica","16","bold"),fg="white",bg="#043b82")
p5_SUMorn_label.pack()

#placing the afternoon column in the window
SUnoos_frame=LabelFrame(showUser_frame,borderwidth=0,bg="#043b82")
SUnoos_frame.pack(side=LEFT,padx=20)

SUnoos_label=Label(SUnoos_frame,text="Afternoon",font=("Helvetica","16","bold"),fg="white",bg="#043b82")
SUnoos_label.pack(side=TOP,fill=X,expand=True,pady=10)

p1_SUnoos_label=Label(SUnoos_frame,text=str(search_result_user[0][7]),font=("Helvetica","16","bold"),fg="white",bg="#043b82")
p1_SUnoos_label.pack()

p2_SUnoos_label=Label(SUnoos_frame,text=str(search_result_user[0][10]),font=("Helvetica","16","bold"),fg="white",bg="#043b82")
p2_SUnoos_label.pack()

p3_SUnoos_label=Label(SUnoos_frame,text=str(search_result_user[0][13]),font=("Helvetica","16","bold"),fg="white",bg="#043b82")
p3_SUnoos_label.pack()

p4_SUnoos_label=Label(SUnoos_frame,text=str(search_result_user[0][16]),font=("Helvetica","16","bold"),fg="white",bg="#043b82")
p4_SUnoos_label.pack()

p5_SUnoos_label=Label(SUnoos_frame,text=str(search_result_user[0][19]),font=("Helvetica","16","bold"),fg="white",bg="#043b82")
p5_SUnoos_label.pack()

#placing the evening column in the window
SUEve_frame=LabelFrame(showUser_frame,borderwidth=0,bg="#043b82")
SUEve_frame.pack(side=LEFT,padx=20)

SUEve_label=Label(SUEve_frame,text="Evening",font=("Helvetica","16","bold"),fg="white",bg="#043b82")
SUEve_label.pack(side=TOP,fill=X,expand=True,pady=10)

p1_SUEve_label=Label(SUEve_frame,text=str(search_result_user[0][8]),font=("Helvetica","16","bold"),fg="white",bg="#043b82")
p1_SUEve_label.pack()

p2_SUEve_label=Label(SUEve_frame,text=str(search_result_user[0][11]),font=("Helvetica","16","bold"),fg="white",bg="#043b82")
p2_SUEve_label.pack()

p3_SUEve_label=Label(SUEve_frame,text=str(search_result_user[0][14]),font=("Helvetica","16","bold"),fg="white",bg="#043b82")
p3_SUEve_label.pack()

p4_SUEve_label=Label(SUEve_frame,text=str(search_result_user[0][17]),font=("Helvetica","16","bold"),fg="white",bg="#043b82")
p4_SUEve_label.pack()
```

```

p5_SUeve_label=Label(SUeve_frame,text=str(search_result_user[0][20]),font=("Helvetica", "16", "bold"),fg="white",bg="#043b82")
p5_SUeve_label.pack()

#placing the total column in the window
SUtotal_frame=LabelFrame(showUser_frame,borderwidth=0,bg="#043b82")
SUtotal_frame.pack(side=LEFT,padx=20)

SUtotal_label=Label(SUtotal_frame,text="Total",font=("Helvetica", "16", "bold"),fg="white",bg="#043b82")
SUtotal_label.pack(side=TOP,fill=X,expand=True,pady=10)

p1_SUtotal_label=Label(SUtotal_frame,text=str(search_result_user[0][1]),font=("Helvetica", "16", "bold"),fg="white",bg="#043b82")
p1_SUtotal_label.pack()

p2_SUtotal_label=Label(SUtotal_frame,text=str(search_result_user[0][2]),font=("Helvetica", "16", "bold"),fg="white",bg="#043b82")
p2_SUtotal_label.pack()

p3_SUtotal_label=Label(SUtotal_frame,text=str(search_result_user[0][3]),font=("Helvetica", "16", "bold"),fg="white",bg="#043b82")
p3_SUtotal_label.pack()

p4_SUtotal_label=Label(SUtotal_frame,text=str(search_result_user[0][4]),font=("Helvetica", "16", "bold"),fg="white",bg="#043b82")
p4_SUtotal_label.pack()

p5_SUtotal_label=Label(SUtotal_frame,text=str(search_result_user[0][5]),font=("Helvetica", "16", "bold"),fg="white",bg="#043b82")
p5_SUtotal_label.pack()

#button to return to the homescreen
Subtn_frame=LabelFrame(show_user,borderwidth=0,bg="#043b82")
Subtn_frame.pack(side=BOTTOM,fill=X,padx=60,pady=20)

done_button=Button(Subtn_frame,text="Return to the home screen",command=Lambda:[show_user.destroy()],\
font=("Helvetica", "10", "bold italic"),fg="#043b82",bg="white",activeforeground="white",activebackground="#00a047")
done_button.pack(fill=X,expand=True)

#creating buttons on the homescreen when a new user has been created
def create_user_buttons():
    global user1_button,user2_button,user3_button
    #prints out contents of table to terminal
    my_cursor.execute("SELECT * FROM users")
    button_result=my_cursor.fetchall()

    #placing buttons depending on the number of users
    if button_result[0][0] != "Start_name":
        user1_button=Button(homeUsers_frame,text=button_result[0][0],command=Lambda:[show_user_window(button_result[0][0]),\
width=16,height=6,font=("Helvetica", "12", "bold"),fg="white",bg="#00a047",activeforeground="#00a047",activebackground="#00a047"])
        user1_button.grid(row=0,column=1,padx=5)

    if button_result[1][0] != "Start_name":
        user2_button=Button(homeUsers_frame,text=button_result[1][0],command=Lambda:[show_user_window(button_result[1][0]),\
width=16,height=6,font=("Helvetica", "12", "bold"),fg="white",bg="#00a047",activeforeground="#00a047",activebackground="#00a047"])
        user2_button.grid(row=0,column=2,padx=5)

    if button_result[2][0] != "Start_name":
        user3_button=Button(homeUsers_frame,text=button_result[2][0],command=Lambda:[show_user_window(button_result[2][0]),\
width=16,height=6,font=("Helvetica", "12", "bold"),fg="white",bg="#00a047",activeforeground="#00a047",activebackground="#00a047"])
        user3_button.grid(row=0,column=3,padx=5)

    if button_result[0][0] != "Start_name" or button_result[1][0] != "Start_name" or button_result[2][0] != "Start_name":
        enter_info_button=Button(homeUsers_frame,text="Create user",command=check_user_amount,\
width=13,height=6,font=("Helvetica", "12", "bold"),fg="#043b82",bg="white",activeforeground="white",activebackground="white")
        enter_info_button.grid(row=0,column=4,padx=5)

    #otherwise, placing the 'Create User' button in the center if there are no existing users
    if button_result[0][0] == "Start_name" and button_result[1][0] == "Start_name" and button_result[2][0] == "Start_name":
        enter_info_button=Button(homeUsers_frame,text="Create user",command=check_user_amount,\
width=14,height=6,font=("Helvetica", "12", "bold"),fg="#043b82",bg="white",activeforeground="white",activebackground="white")
        enter_info_button.grid(row=0,column=0,padx=5)

#deleting the button for the user when the user is deleted
def delete_user_buttons(delete_user):
    #prints out contents of table to terminal
    my_cursor.execute("SELECT * FROM users")
    button_delete=my_cursor.fetchall()

    #print(delete_user)

    if delete_user==1:

```



```
#placing elements on the homescreen
home_frame=LabelFrame(root,borderwidth=0,bg="#043b82",padx=10,pady=5)
home_frame.pack(fill=X,pady=30)

#frame for the time elements on the homescreen
time_frame=LabelFrame(home_frame,borderwidth=0,fg="white",bg="#043b82")
time_frame.pack(side=LEFT,padx=45)

#labels used in the clock function
day_label=Label(time_frame,text="",font=("Helvetica",16,"bold"),fg="white",bg="#043b82")
day_label.pack(anchor=W)

time_label=Label(time_frame,text="",font=("Helvetica",16,"bold"),fg="white",bg="#043b82")
time_label.pack(anchor=W)

zone_label=Label(time_frame,text="",font=("Helvetica",14,"italic"),fg="white",bg="#043b82")
zone_label.pack(anchor=W)

clock()

#frame for the user buttons & 'Create user' on the homescreen
homeUsers_frame=LabelFrame(root,borderwidth=0,bg="#043b82",padx=55,pady=75)
homeUsers_frame.pack(fill=BOTH,expand=True)
create_user_buttons()

HUtop_frame=LabelFrame(home_frame,borderwidth=0,bg="#043b82")
HUtop_frame.pack(side=RIGHT,padx=50)

#button to input what is in the containers of the device
settings_button=Button(HUtop_frame,text="Settings",command=settings_window,\
    font=("Helvetica",12,"bold"),fg="#043b82",bg="white",activeforeground="white",activebackground="#00a047")
settings_button.pack(fill=BOTH)

#button to exit the program
off_button=Button(HUtop_frame,text="Log off",command=root.quit,\
    font=("Helvetica",12,"bold"),fg="#043b82",bg="white",activeforeground="white",activebackground="#00a047")
off_button.pack(fill=BOTH)

root.mainloop()

if __name__ == '__main__':
    def restart_program():
        root.destroy()
        start_gui()
    start_gui()
```