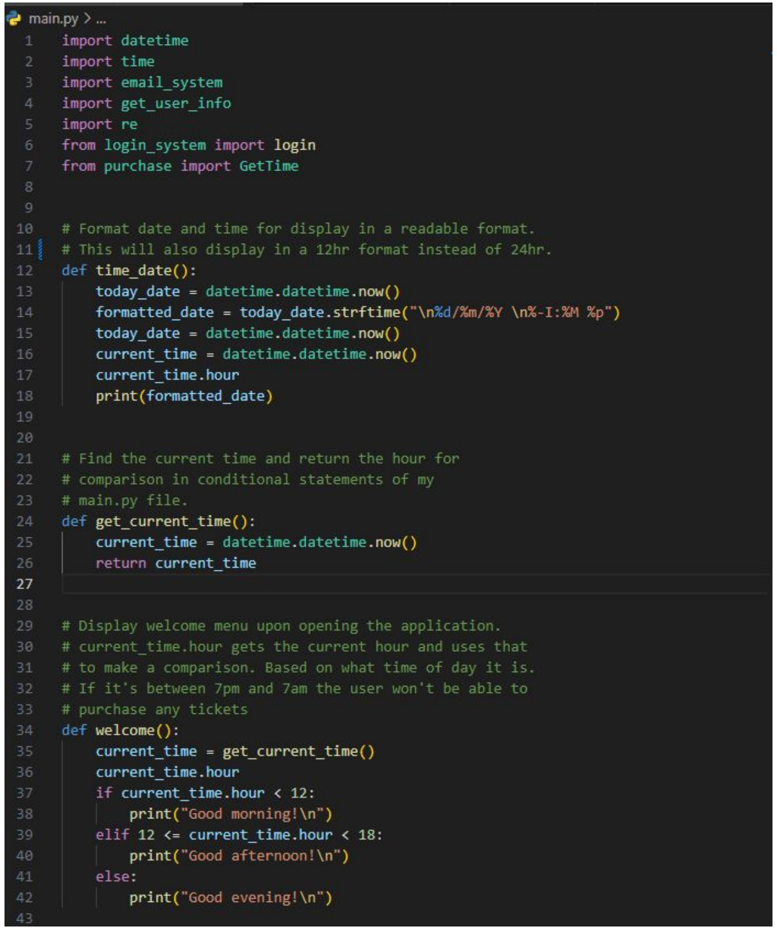
Isaac Eveans Mellonie T1A3

Code Overview

main.py



main.py > ...

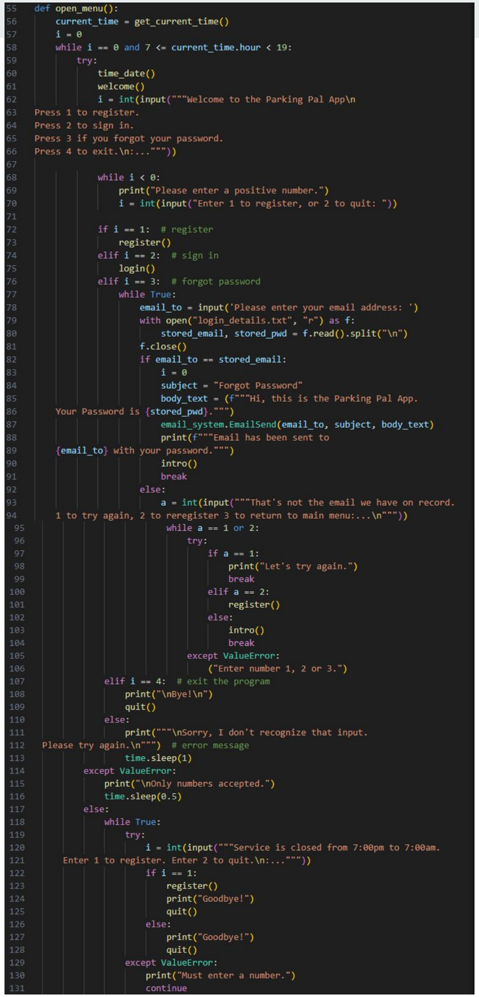
```python
import datetime
import time
import email_system
import get_user_info
import re
from login_system import login
from purchase import GetTime


# Format date and time for display in a readable format.
# This will also display in a 12hr format instead of 24hr.
def time_date():
    today_date = datetime.datetime.now()
    formatted_date = today_date.strftime("\n%d/%m/%Y \n%-I:%M %p")
    today_date = datetime.datetime.now()
    current_time = datetime.datetime.now()
    current_time.hour
    print(formatted_date)


# Find the current time and return the hour for
# comparison in conditional statements of my
# main.py file.
def get_current_time():
    current_time = datetime.datetime.now()
    return current_time


# Display welcome menu upon opening the application.
# current_time.hour gets the current hour and uses that
# to make a comparison. Based on what time of day it is.
# If it's between 7pm and 7am the user won't be able to
# purchase any tickets
def welcome():
    current_time = get_current_time()
    current_time.hour
    if current_time.hour < 12:
        print("Good morning!\n")
    elif 12 <= current_time.hour < 18:
        print("Good afternoon!\n")
    else:
        print("Good evening!\n")

```

Imports:

- Several Python standard libraries (datetime, time, and re) and external modules (email_system, get_user_info, login_system, and purchase) are imported for use in the application.

Date & Time Formatting:

- time_date(): Retrieves the current date and time, and displays it in a formatted manner using the 12-hour clock.
- get_current_time(): Retrieves and returns the current time.
- welcome(): Displays a greeting message depending on the current time of day.



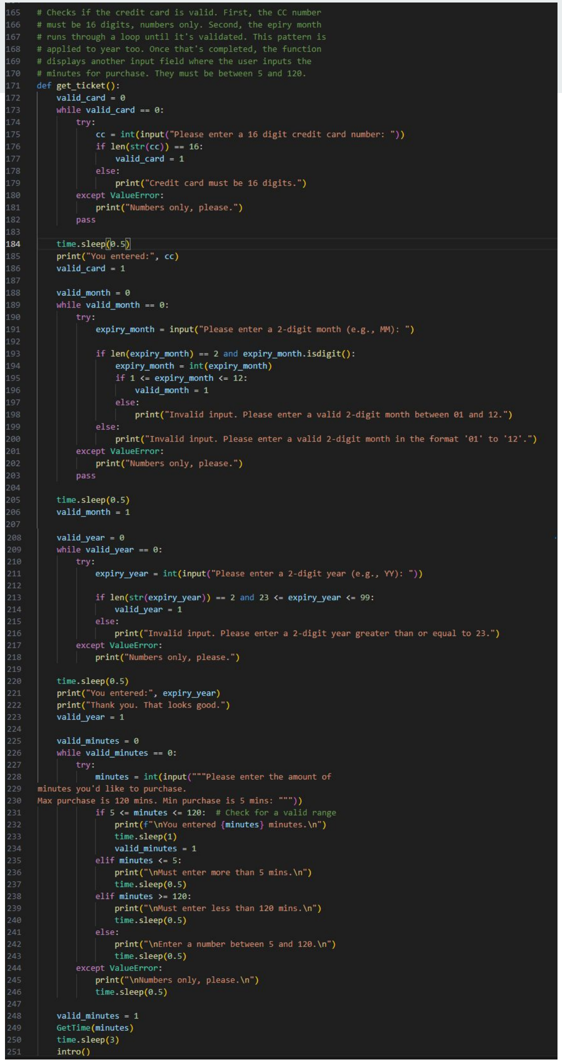
```
def open_menu():
    current_time = get_current_time()
    i = 0
    while i == 0 and 7 <= current_time.hour < 19:
        try:
            time_date()
            welcome()
            i = int(input("""Welcome to the Parking Pal App\n
Press 1 to register.
Press 2 to sign in.
Press 3 if you forgot your password.
Press 4 to exit.\n:..."""))

            while i < 0:
                print("Please enter a positive number.")
                i = int(input("Enter 1 to register, or 2 to quit: "))

            if i == 1:  # register
                register()
            elif i == 2:  # sign in
                login()
            elif i == 3:  # forgot password
                while True:
                    email_to = input('Please enter your email address: ')
                    with open("login_details.txt", "r") as f:
                        stored_email, stored_pwd = f.read().split("\n")
                    f.close()
                    if email_to == stored_email:
                        i = 0
                        subject = "Forgot Password"
                        body_text = (f"""Hi, this is the Parking Pal App.
    Your Password is {stored_pwd}.""")
                        email_system.EmailSend(email_to, subject, body_text)
                        print(f"""Email has been sent to
    {email_to} with your password.""")
                        intro()
                        break
                    else:
                        a = int(input("""That's not the email we have on record.
    1 to try again, 2 to reregister 3 to return to main menu:...\n"""))
                        while a == 1 or 2:
                            try:
                                if a == 1:
                                    print("Let's try again.")
                                    break
                                elif a == 2:
                                    register()
                                else:
                                    intro()
                                    break
                            except ValueError:
                                ("Enter number 1, 2 or 3.")
            elif i == 4:  # exit the program
                print("\nBye!\n")
                quit()
            else:
                print("""\nSorry, I don't recognize that input.
Please try again.\n""")  # error message
                time.sleep(1)
        except ValueError:
            print("\nOnly numbers accepted.")
            time.sleep(0.5)
        else:
            while True:
                try:
                    i = int(input("""Service is closed from 7:00pm to 7:00am.
    Enter 1 to register. Enter 2 to quit.\n:..."""))
                    if i == 1:
                        register()
                        print("Goodbye!")
                        quit()
                    else:
                        print("Goodbye!")
                        quit()
                except ValueError:
                    print("Must enter a number.")
                    continue
```

Open Menu - Main Application Interface:

- open_menu(): Presents the user with an interactive menu. The menu provides options for user registration, signing in, retrieving a forgotten password, and exiting the application. This function takes into account the time of day, as ticket purchases are restricted between 7 pm and 7 am.

User Authentication:

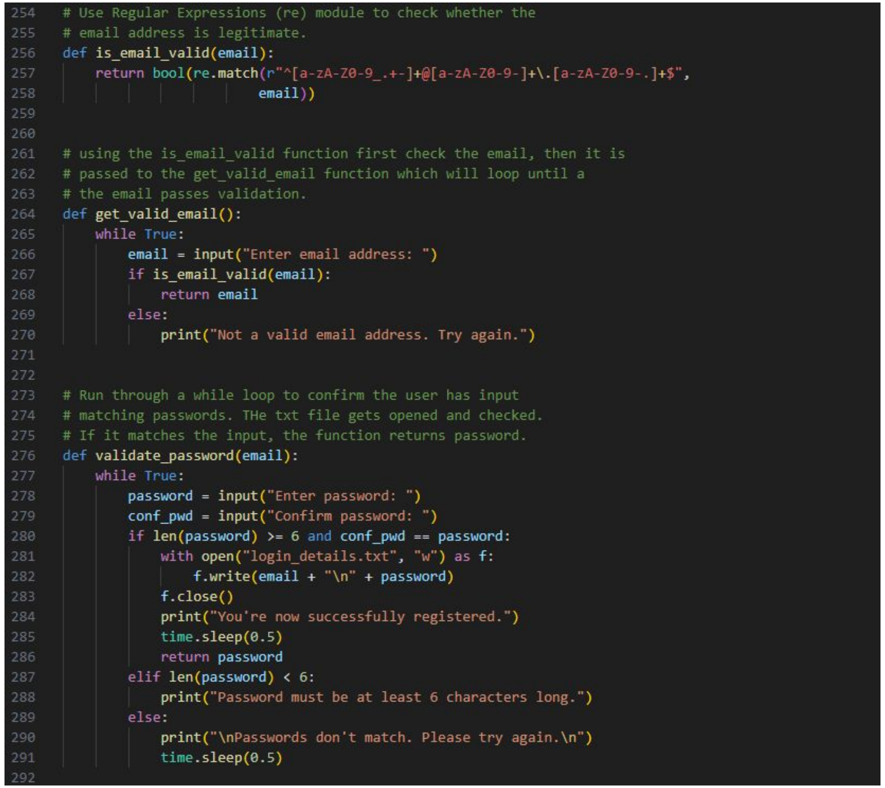
- login(): Allows users to input their email and password. It reads these credentials from a file (login_details.txt) and checks if the entered values match. If successful, the user can proceed to purchase tickets. Otherwise, they are prompted to retry or go back to the main menu.
- register(): Helps users register by validating their email address and ensuring the password meets certain criteria. After collecting other information, it saves the user details to a CSV file (login_details.csv).
- Password Recovery: Within the open_menu(), there's a section that aids users who forgot their password. The system sends an email to the user's registered email address with the password.



```python
# Checks if the credit card is valid. First, the CC number
# must be 16 digits, numbers only. Second, the epiry month
# runs through a loop until it's validated. This pattern is
# applied to year too. Once that's completed, the function
# displays another input field where the user inputs the
# minutes for purchase. They must be between 5 and 120.
def get_ticket():
    valid_card = 0
    while valid_card == 0:
        try:
            cc = int(input("Please enter a 16 digit credit card number: "))
            if len(str(cc)) == 16:
                valid_card = 1
            else:
                print("Credit card must be 16 digits.")
        except ValueError:
            print("Numbers only, please.")
        pass

    time.sleep(0.5)
    print("You entered:", cc)
    valid_card = 1

    valid_month = 0
    while valid_month == 0:
        try:
            expiry_month = input("Please enter a 2-digit month (e.g., MM): ")

            if len(expiry_month) == 2 and expiry_month.isdigit():
                expiry_month = int(expiry_month)
                if 1 <= expiry_month <= 12:
                    valid_month = 1
                else:
                    print("Invalid input. Please enter a valid 2-digit month between 01 and 12.")
            else:
                print("Invalid input. Please enter a valid 2-digit month in the format '01' to '12'.")
        except ValueError:
            print("Numbers only, please.")
        pass

    time.sleep(0.5)
    valid_month = 1

    valid_year = 0
    while valid_year == 0:
        try:
            expiry_year = int(input("Please enter a 2-digit year (e.g., YY): "))

            if len(str(expiry_year)) == 2 and 23 <= expiry_year <= 99:
                valid_year = 1
            else:
                print("Invalid input. Please enter a 2-digit year greater than or equal to 23.")
        except ValueError:
            print("Numbers only, please.")

    time.sleep(0.5)
    print("You entered:", expiry_year)
    print("Thank you. That looks good.")
    valid_year = 1

    valid_minutes = 0
    while valid_minutes == 0:
        try:
            minutes = int(input("""Please enter the amount of
minutes you'd like to purchase.
Max purchase is 120 mins. Min purchase is 5 mins: """))
            if 5 <= minutes <= 120:  # Check for a valid range
                print(f"\nYou entered {minutes} minutes.\n")
                time.sleep(1)
                valid_minutes = 1
            elif minutes <= 5:
                print("\nMust enter more than 5 mins.\n")
                time.sleep(0.5)
            elif minutes >= 120:
                print("\nMust enter less than 120 mins.\n")
                time.sleep(0.5)
            else:
                print("\nEnter a number between 5 and 120.\n")
                time.sleep(0.5)
        except ValueError:
            print("\nNumbers only, please.\n")
            time.sleep(0.5)

    valid_minutes = 1
    GetTime(minutes)
    time.sleep(3)
    intro()
```

Purchasing a Ticket:

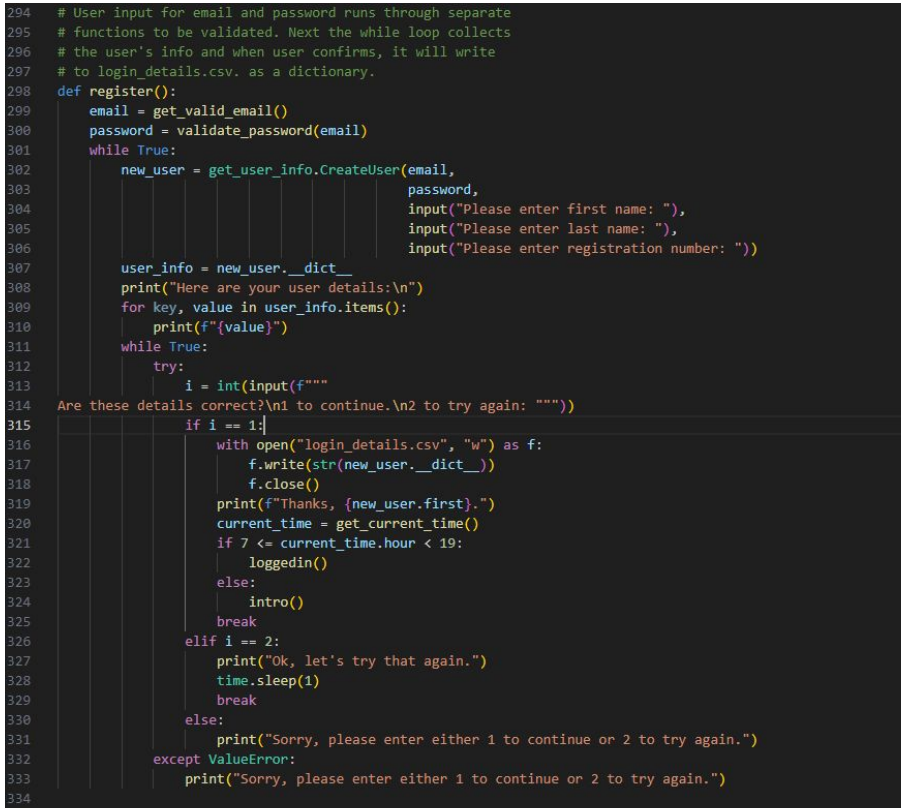
- User Input Validation: One of the primary focuses of this function is to ensure that user input is validated thoroughly. By employing try-except blocks and while loops, I've ensured that the user is prompted to provide the correct input format until they do so.
- Feedback and Redirection: Throughout the function, there are feedback messages, like "Numbers only, please," that guide the user in the right direction when they enter data that doesn't fit the required format. This approach ensures that users are always aware of any mistakes they make and understand how to correct them. The time.sleep() method is used to pace the feedback, giving users a moment to process the information.
- Modular Approach: The code is organized into different sections, each responsible for validating a specific type of input (credit card, expiry month, expiry year, and minutes). This modular approach makes the code easier to read and debug. If there's an issue or an enhancement needed for a particular input type, it can be addressed in its respective section without affecting the others.
- Reusability and Flow Control: Once the user has completed the ticket purchase process, the GetTime(minutes) function is called, emphasizing the modular design, where functions are used to manage specific tasks. After that, the intro() function is run, redirecting the user to the main application interface. This continuous loop ensures that users can navigate through multiple functionalities without having to restart the application.



```python
# Use Regular Expressions (re) module to check whether the
# email address is legitimate.
def is_email_valid(email):
    return bool(re.match(r"^[a-zA-Z0-9_.+-]+@[a-zA-Z0-9-]+\.[a-zA-Z0-9-.]+$",
                         email))


# using the is_email_valid function first check the email, then it is
# passed to the get_valid_email function which will loop until a
# the email passes validation.
def get_valid_email():
    while True:
        email = input("Enter email address: ")
        if is_email_valid(email):
            return email
        else:
            print("Not a valid email address. Try again.")


# Run through a while loop to confirm the user has input
# matching passwords. THe txt file gets opened and checked.
# If it matches the input, the function returns password.
def validate_password(email):
    while True:
        password = input("Enter password: ")
        conf_pwd = input("Confirm password: ")
        if len(password) >= 6 and conf_pwd == password:
            with open("login_details.txt", "w") as f:
                f.write(email + "\n" + password)
            f.close()
            print("You're now successfully registered.")
            time.sleep(0.5)
            return password
        elif len(password) < 6:
            print("Password must be at least 6 characters long.")
        else:
            print("\nPasswords don't match. Please try again.\n")
            time.sleep(0.5)

```

Data Validation:

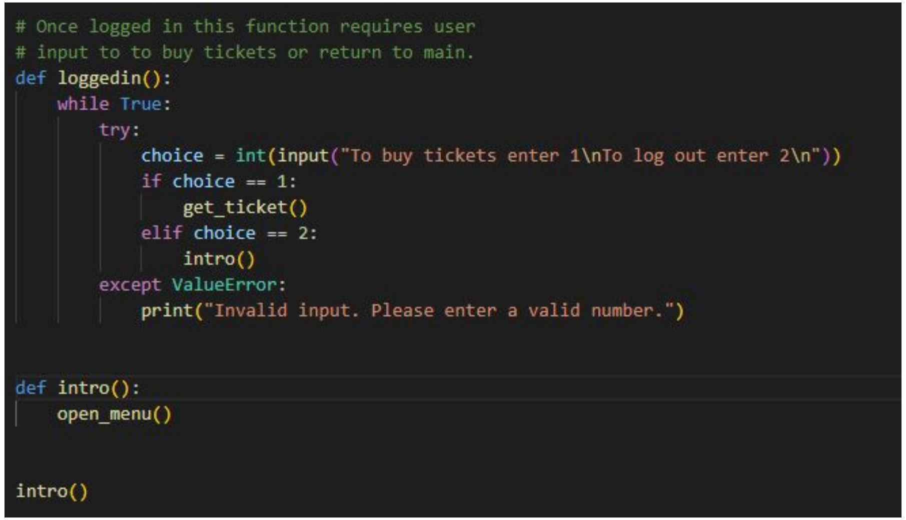
- is_email_valid(email): Uses regular expressions to verify the legitimacy of an email address.
- get_valid_email(): Repeatedly asks the user for a valid email address until a legitimate one is provided.
- validate_password(email): Ensures that the password entered is at least 6 characters long and matches its confirmation.



```python
# User input for email and password runs through separate
# functions to be validated. Next the while loop collects
# the user's info and when user confirms, it will write
# to login_details.csv. as a dictionary.
def register():
    email = get_valid_email()
    password = validate_password(email)
    while True:
        new_user = get_user_info.CreateUser(email,
                                            password,
                                            input("Please enter first name: "),
                                            input("Please enter last name: "),
                                            input("Please enter registration number: "))
        user_info = new_user.__dict__
        print("Here are your user details:\n")
        for key, value in user_info.items():
            print(f"{value}")
        while True:
            try:
                i = int(input(f"""
Are these details correct?\n1 to continue.\n2 to try again: """))
                if i == 1:
                    with open("login_details.csv", "w") as f:
                        f.write(str(new_user.__dict__))
                        f.close()
                    print(f"Thanks, {new_user.first}.")
                    current_time = get_current_time()
                    if 7 <= current_time.hour < 19:
                        loggedin()
                    else:
                        intro()
                    break
                elif i == 2:
                    print("Ok, let's try that again.")
                    time.sleep(1)
                    break
                else:
                    print("Sorry, please enter either 1 to continue or 2 to try again.")
            except ValueError:
                print("Sorry, please enter either 1 to continue or 2 to try again.")
```

The register() function is designed to guide a user through the registration process, ensuring they provide all necessary details and that these details are correctly formatted and valid.

- Email Gathering and Validation: Initially, the function prompts the user for an email address. This email is then validated to ensure it follows a standard email format.
- Password Gathering and Validation: After obtaining a valid email, the user is prompted to set a password. The function ensures that the password is between 6 and 12 characters and also requires the user to confirm their password to avoid mistakes.
- Collecting Additional User Info: Next, the function gathers more personal details from the user, including their first name, last name, and registration number.
- Display and Confirm User Details: Once all the details are collected, they're displayed back to the user for review. The user is then asked to confirm if the displayed information is accurate.
- Storing User Details: After the user confirms their details, the function saves this data to a file named login_details.csv. This ensures that the user's details are stored for future sessions or references.
- Post-Registration Navigation: The function checks the current time. Depending on the time of day, the user is directed either to the out of hours menu or a logged-in menu.

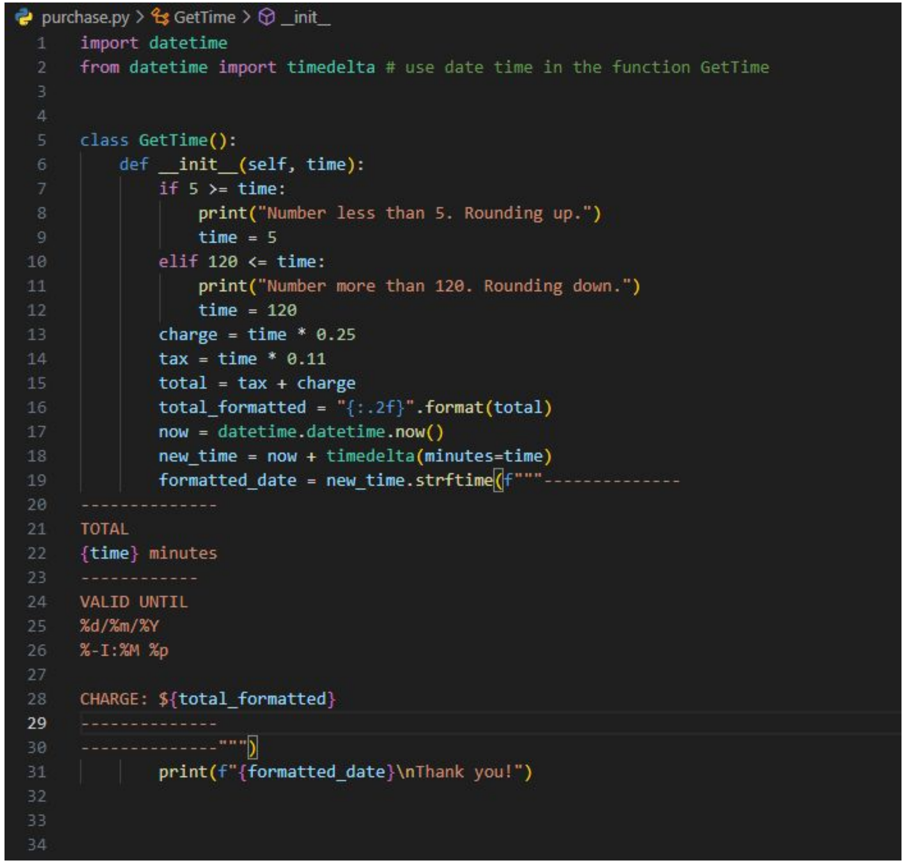


```
# Once logged in this function requires user
# input to to buy tickets or return to main.
def loggedin():
    while True:
        try:
            choice = int(input("To buy tickets enter 1\nTo log out enter 2\n"))
            if choice == 1:
                get_ticket()
            elif choice == 2:
                intro()
        except ValueError:
            print("Invalid input. Please enter a valid number.")


def intro():
    open_menu()


intro()
```

- Logged-In User Options: loggedin(): Presents logged-in users with the choice of buying tickets or logging out.
- Program Execution: The program starts by calling the intro() function, which in turn invokes open_menu(), initiating the main application loop.

Code Overview

purchase.py

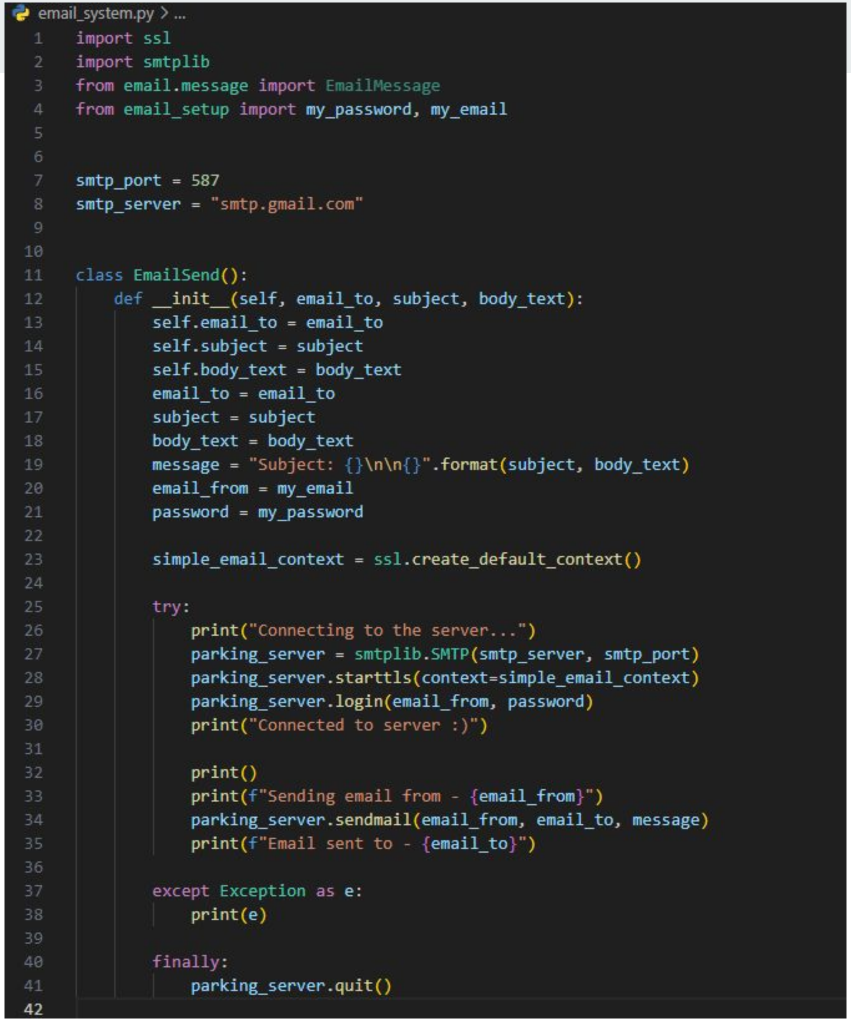


```python
import datetime
from datetime import timedelta # use date time in the function GetTime


class GetTime():
    def __init__(self, time):
        if 5 >= time:
            print("Number less than 5. Rounding up.")
            time = 5
        elif 120 <= time:
            print("Number more than 120. Rounding down.")
            time = 120
        charge = time * 0.25
        tax = time * 0.11
        total = tax + charge
        total_formatted = "{:.2f}".format(total)
        now = datetime.datetime.now()
        new_time = now + timedelta(minutes=time)
        formatted_date = new_time.strftime(f"""--------------
--------------
TOTAL
{time} minutes
------------
VALID UNTIL
%d/%m/%Y
%-I:%M %p

CHARGE: ${total_formatted}
--------------
--------------""")
        print(f"{formatted_date}\nThank you!")
```

- Boundary Handling for Time Input: The code includes conditions to check if the input time is outside the accepted range (5 to 120 minutes). If so, it automatically adjusts these values to the nearest boundary. This design choice minimizes potential errors and ensures consistent processing without rejecting outlier inputs.
- Utilization of datetime and timedelta: This simplifies the task of adding minutes to the current time.
- Dynamic Ticket Formatting: The use of string formatting combined with strftime from the datetime module enables the dynamic creation of ticket details based on user input. This design approach ensures adaptability.
- Clear Price Calculation Logic: Straightforward computation for the charge and tax based on the input minutes. By segregating each component (base charge, tax, total) and using clear variable names, the code makes its logic easily comprehensible to other developers.

Code Overview

email_system.py

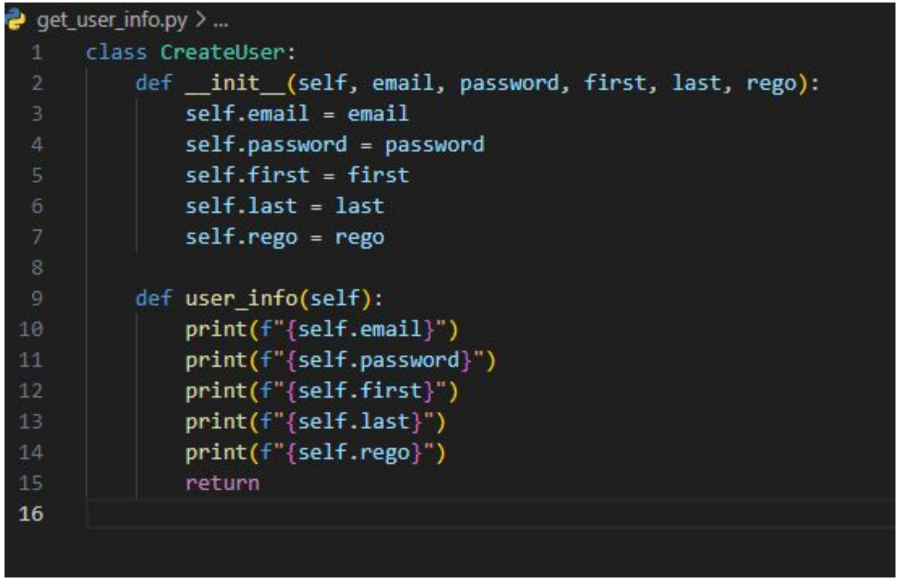

email_system.py > ...

```python
import ssl
import smtplib
from email.message import EmailMessage
from email_setup import my_password, my_email


smtp_port = 587
smtp_server = "smtp.gmail.com"


class EmailSend():
    def __init__(self, email_to, subject, body_text):
        self.email_to = email_to
        self.subject = subject
        self.body_text = body_text
        email_to = email_to
        subject = subject
        body_text = body_text
        message = "Subject: {}\n\n{}".format(subject, body_text)
        email_from = my_email
        password = my_password

        simple_email_context = ssl.create_default_context()

        try:
            print("Connecting to the server...")
            parking_server = smtplib.SMTP(smtp_server, smtp_port)
            parking_server.starttls(context=simple_email_context)
            parking_server.login(email_from, password)
            print("Connected to server :)")

            print()
            print(f"Sending email from - {email_from}")
            parking_server.sendmail(email_from, email_to, message)
            print(f"Email sent to - {email_to}")

        except Exception as e:
            print(e)

        finally:
            parking_server.quit()
```

- Encapsulation with a Class: The choice to encapsulate the email sending functionality within a class (EmailSend) promotes object-oriented programming principles. This design decision allows for easy scalability and organization.
- Separation of Credentials: Importing the my_password and my_email from an external module (email_setup) is a security-conscious decision. This separation keeps sensitive data out of the main code, which can be particularly beneficial when sharing or version-controlling the main script without exposing credentials.
- Exception Handling during Email Transmission: The use of the try-except-finally block during the email sending process accounts for runtime issues. By wrapping the email sending code inside this block, the program can handle exceptions, provide meaningful feedback to the user.

Code Overview

get_user_info.py



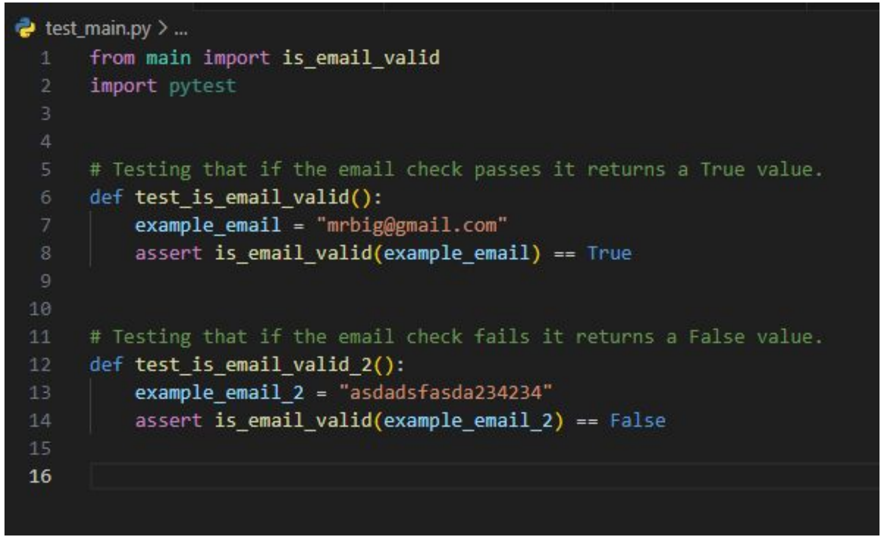
get_user_info.py > ...

```python
class CreateUser:
    def __init__(self, email, password, first, last, rego):
        self.email = email
        self.password = password
        self.first = first
        self.last = last
        self.rego = rego

    def user_info(self):
        print(f"{self.email}")
        print(f"{self.password}")
        print(f"{self.first}")
        print(f"{self.last}")
        print(f"{self.rego}")
        return

```

- Object-Oriented Approach with Encapsulation: By defining a CreateUser class, the design prioritizes object-oriented principles, encapsulating user-related attributes (like email, password, first and last names, and rego) within a single cohesive structure. This makes it easier to manage user-related data and functionalities as the program grows.
- Explicit User Info Display: The user_info method provides a direct way to output the user's details.

Code Overview

test_main.py

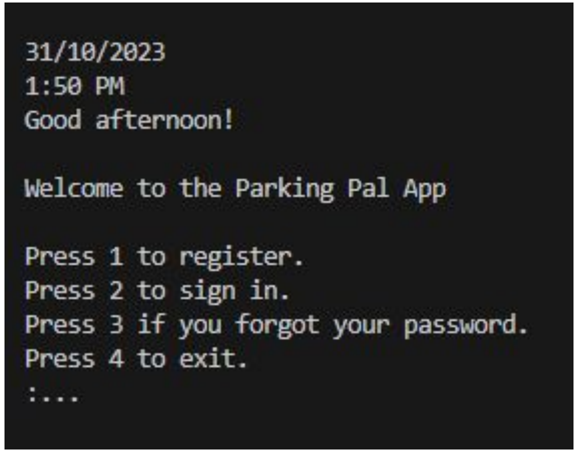

test_main.py > ...

```python
from main import is_email_valid
import pytest


# Testing that if the email check passes it returns a True value.
def test_is_email_valid():
    example_email = "mrbig@gmail.com"
    assert is_email_valid(example_email) == True


# Testing that if the email check fails it returns a False value.
def test_is_email_valid_2():
    example_email_2 = "asdadsfasda234234"
    assert is_email_valid(example_email_2) == False

```

- I imported pytest to run the testing on the function is_email_valid(). The tests are run on this function because it give an output. Most functions within my main don’t return a value which made the testing process difficult.

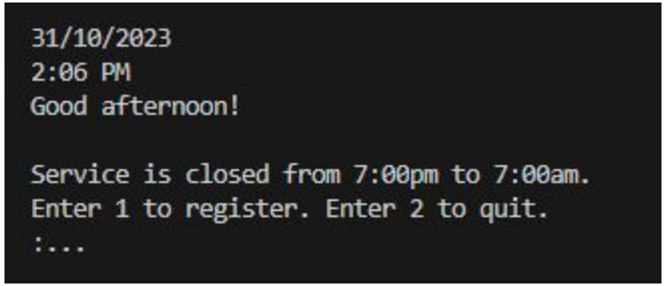
Application Walkthrough

1.



```
31/10/2023
1:50 PM
Good afternoon!

Welcome to the Parking Pal App

Press 1 to register.
Press 2 to sign in.
Press 3 if you forgot your password.
Press 4 to exit.
:...
```

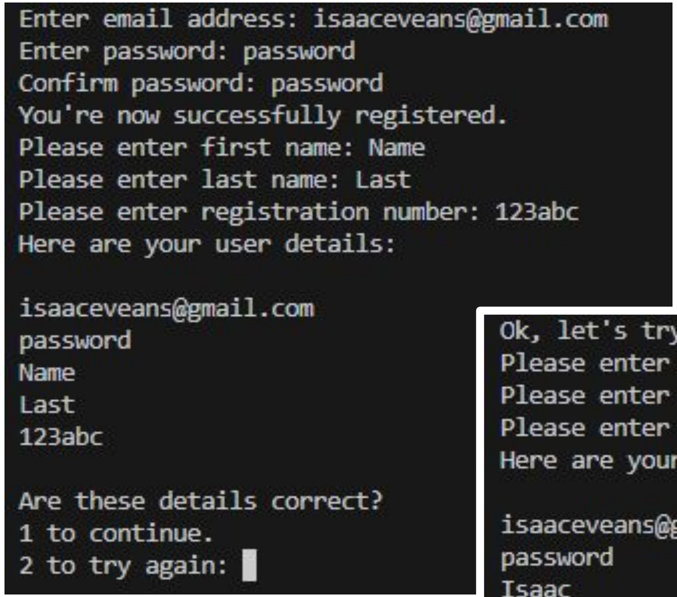
2.



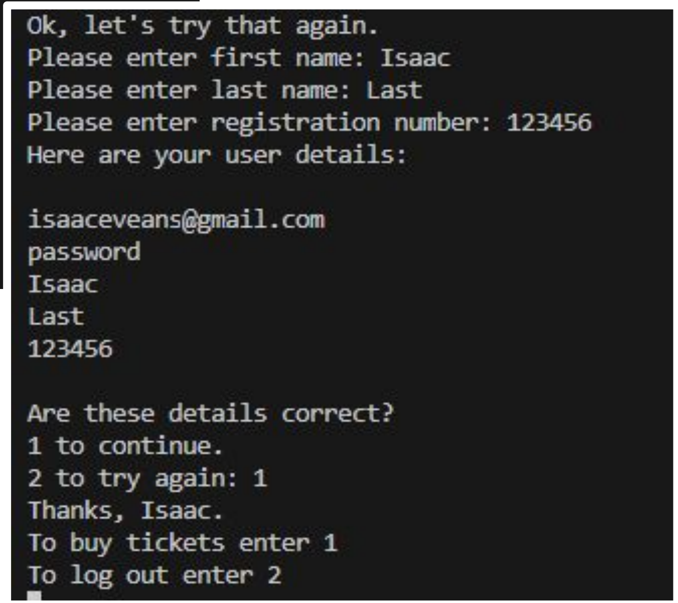
```
31/10/2023
2:06 PM
Good afternoon!

Service is closed from 7:00pm to 7:00am.
Enter 1 to register. Enter 2 to quit.
:...
```

- When the terminal app first opens, it begins by checking what the time is. If it’s within purchasing hours the full menu will appear. If not, the choices will be limited to registration only.

1. Shows the service hours menu
2. Shows the limited menu that will appear during off hours



```
Enter email address: isaaceveans@gmail.com
Enter password: password
Confirm password: password
You're now successfully registered.
Please enter first name: Name
Please enter last name: Last
Please enter registration number: 123abc
Here are your user details:

isaaceveans@gmail.com
password
Name
Last
123abc

Are these details correct?
1 to continue.
2 to try again: 
```

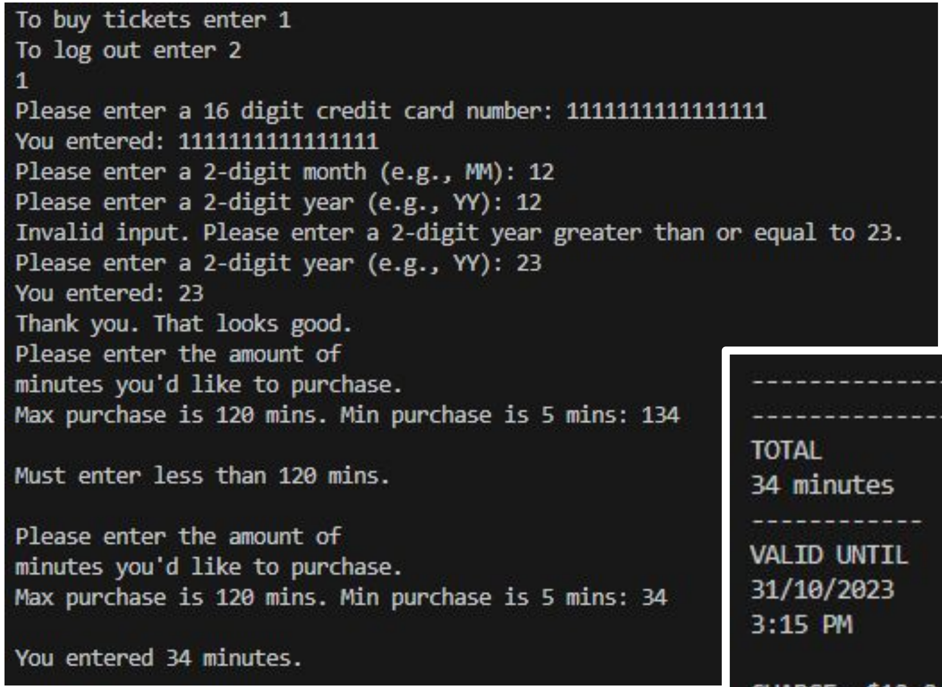


```
Ok, let's try that again.
Please enter first name: Isaac
Please enter last name: Last
Please enter registration number: 123456
Here are your user details:

isaaceveans@gmail.com
password
Isaac
Last
123456

Are these details correct?
1 to continue.
2 to try again: 1
Thanks, Isaac.
To buy tickets enter 1
To log out enter 2
```

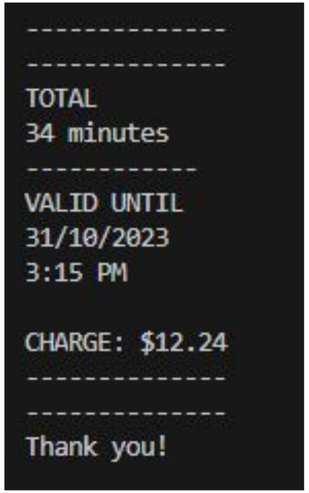
<u>Register:</u>
Must pass the email validation and also enter the password twice to confirm.

Next, the user will be prompted to put in their information and asked to confirm if it all looks good after.

If not, the process repeats itself. If yes, the next menu appears. Giving two option to continue to purchase tickets or sign out.



```
To buy tickets enter 1
To log out enter 2
1
Please enter a 16 digit credit card number: 1111111111111111
You entered: 1111111111111111
Please enter a 2-digit month (e.g., MM): 12
Please enter a 2-digit year (e.g., YY): 12
Invalid input. Please enter a 2-digit year greater than or equal to 23.
Please enter a 2-digit year (e.g., YY): 23
You entered: 23
Thank you. That looks good.
Please enter the amount of
minutes you'd like to purchase.
Max purchase is 120 mins. Min purchase is 5 mins: 134

Must enter less than 120 mins.

Please enter the amount of
minutes you'd like to purchase.
Max purchase is 120 mins. Min purchase is 5 mins: 34

You entered 34 minutes.
```



```
--------------
--------------
TOTAL
34 minutes
------------
VALID UNTIL
31/10/2023
3:15 PM

CHARGE: $12.24
--------------
--------------
Thank you!
```

Ticket Purchase

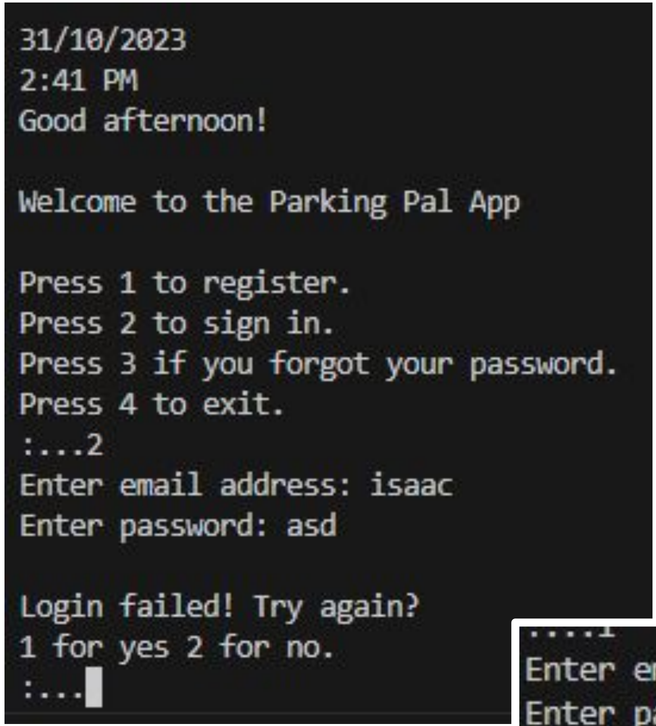
Enter the credit card number less than 16 digits. Once it passes the checks the month comes next which must be between 1 and 12. Once this passes checking the next stop asks for the year greater than 23 (current year).

After all of these details are corrects pass checking you can enter the amount of time you'd like to purchase in minutes.

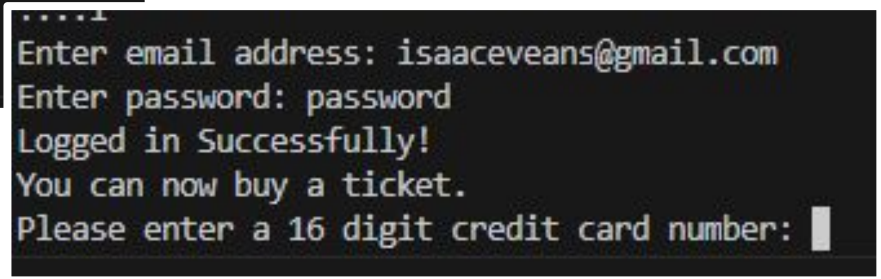
Using a combo of if, else and try, excepts for all of these steps to make sure the user only enters number.

Demonstrated here, one of the conditions is that the time must be between 5 and 120 mins. There's a prompt to alert the user to try again if this is False.

Once the conditions are met. The time will be sent to purchase.GetTime() which will calculate the cost and print out a receipt.



```
31/10/2023
2:41 PM
Good afternoon!

Welcome to the Parking Pal App

Press 1 to register.
Press 2 to sign in.
Press 3 if you forgot your password.
Press 4 to exit.
:...2
Enter email address: isaac
Enter password: asd

Login failed! Try again?
1 for yes 2 for no.
:...
```



```
:...1
Enter email address: isaaceveans@gmail.com
Enter password: password
Logged in Successfully!
You can now buy a ticket.
Please enter a 16 digit credit card number:
```

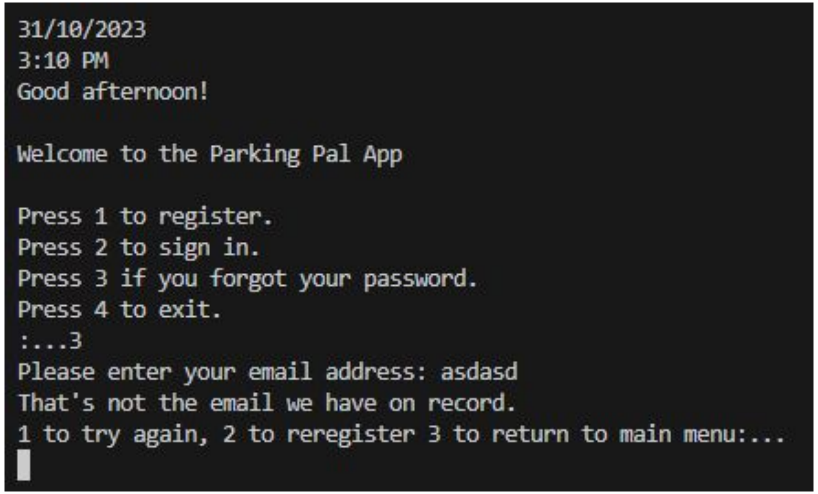
<u>SignIn</u>

If the email entered doesn't match the stored email then the login will fail. If the passwords don't match then the login will also fail. Both checks must pass for the program to accept. There's a way out of the loop though if the user can't remember their details.

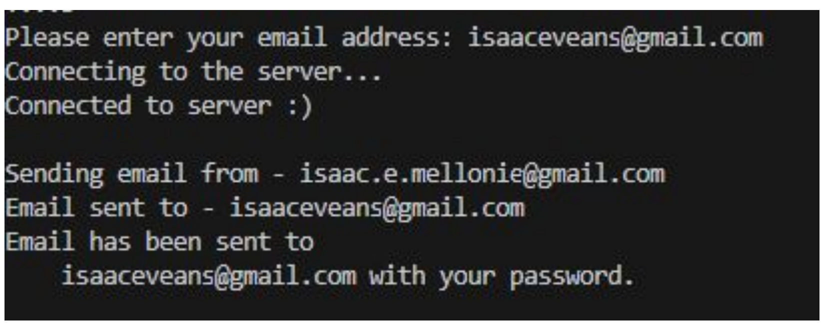
Once it's passed, the user will have access to the ticket purchase where they'll have to pass the same tests as registration.

<u>Forgot Password</u>

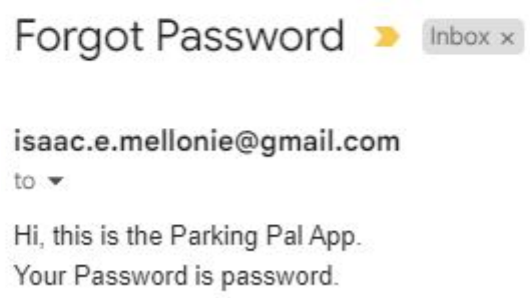
If the user has forgotten their password and they want to retrieve the information, they can enter their email that is stored locally. It must match the locally stored email for this to happen though. If they can't remember their password then they can try again, reregister, return to main menu. If they try again and the email matches the one kept on record teh email will be sent through.



```
31/10/2023
3:10 PM
Good afternoon!

Welcome to the Parking Pal App

Press 1 to register.
Press 2 to sign in.
Press 3 if you forgot your password.
Press 4 to exit.
:...3
Please enter your email address: asdasd
That's not the email we have on record.
1 to try again, 2 to reregister 3 to return to main menu:...
```



```
Please enter your email address: isaaceveans@gmail.com
Connecting to the server...
Connected to server :)

Sending email from - isaac.e.mellonie@gmail.com
Email sent to - isaaceveans@gmail.com
Email has been sent to
    isaaceveans@gmail.com with your password.
```



Forgot Password

Inbox ×

isaac.e.mellonie@gmail.com

to

Hi, this is the Parking Pal App.

Your Password is password.

Conclusion

I learned so much during this project. I’ve gone from having never used python, to building an application and running it in the terminal in the space of a month. It’s been an extremely rewarding process. The most enjoyable aspects of this project were when I could get something to function using a solution that I developed. It was painstakingly time consuming trying to find errors and reasons for things breaking that were working the day before. It taught be that from the get go, managing the project well saves you so much time and stress. I used so much of my time re-coding my functions due to them being so long. Due to this I would break down my functions into much smaller sized pieces and have them output for another function. I learnt this the hard way, but I guess this is normal when you’re just starting out. I found many useful tutorials and guides online that helped me to overcome the walls that I hit at different stages and without the internet I wouldn’t have succeeded! I’m eager to keep learning python in my free time away from the course as I feel I still need to work on my problem solving, logic and algorithmic thinking skills. Overall though, I feel I’m progressing along nicely and even though it feels like I’ve just started climbing this mountain, I will eventually get to a stage where I can help others on their journey too.