COMP1811 – Python Project Report

Name:	Israt Jahan Risma	Student ID	001339277
Partner's name:	Sanjida Khan	Student ID	001276650

1. BRIEF STATEMENT OF FEATURES YOU HAVE COMPLETED

THIS SECTION SHOULD BE THE SAME FOR ALL GROUP MEMBERS

1.1 Circle the parts of the coursework you have **fully completed and are fully working**. Please be accurate.	**Features** **F1:** i ☒ ii ☒ iii ☒ iv ☒ v ☒ vi ☒ **F2:** i ☒ ii ☒ iii ☒ iv ☒ v ☒ vi ☒ **F3:** i ☒ ii ☒ iii ☒ iv ☒ v ☒
1.2 Circle the parts of the coursework you have **partly completed or are partly working.**	**Features** **F1:** i ☐ ii ☐ iii ☐ iv ☐ v ☐ vi ☐ **F2:** i ☐ ii ☐ iii ☐ iv ☐ v ☐ vi ☐ **F3:** i ☐ ii ☐ iii ☐ iv ☐ v ☐
Briefly explain your answer if you circled any parts in 1.2	

2. Concise List of Bugs and Weaknesses

*A concise list of bugs and/or weaknesses in your work (if you don't think there are any, then say so). Bugs that are declared in this list will lose you fewer marks than ones that you don't declare! (**100-200 words**, but word count depends heavily on the number of bugs and weaknesses identified.)*

THIS SECTION SHOULD BE COMPLETED INDIVIDUALLY FOR F1 AND F2 AND AS A GROUP FOR F3.

2.1 Bugs

List each bug plus a brief description. A bug is code that causes an error or produces unexpected results.

1.The time module we imported but not used in the customer class. This can be considered unnecessary and might confuse someone reading the code. What we could do is remove the unused import time in customer class and fix it.

2.The time module is used as both a module and a variable name in the get_checkout_time method of the customer class which can lead to confusion and unexpected behavior so instead of that what we can do here is rename the time variable in the get_checkout_time method to avoid conflicts with the time module

3.we could not add 8 more self-service lane under the main individual self-service lane.

2.2 Weaknesses

List each weakness plus a brief description. A weakness is code that only works under limited scenarios and at some point produces erroneous or unexpected results or code/output that can be improved.

When executing the codes we faced some robust exception handling and making it vulnerable to runtime errors. For instance, there was no handling of potential exceptions that could occur during list operations, file operations or other critical sections of the code. however we implemented proper exception handling to the potential errors and provided meaningful error message.

3. DESCRIPTION OF THE FEATURES IMPLEMENTED

*Describe your implementation design and the choices made (e.g. choice of data structures, custom data types, code logic, choice of functions, etc) and indicate how the features developed were integrated. (**200-400 words***

THIS SECTION SHOULD BE COMPLETED INDIVIDUALLY FOR F1 AND F2 AND AS A GROUP FOR F3.

F1- Lane Management Features:(Israt)

CheckoutLane Implementation:

The CheckoutLane class represents the checkout lanes in the supermarket. It inherits from the Customer class, which is a design choice to leverage some shared functionality and attributes. Each lane has a name, type (regular or self-service), capacity, status, and a list of customers.

To manage customers efficiently, a list is used to store customer objects within each lane. This allows for dynamic customer handling, such as adding and removing customers based on lane capacity. The is_full method checks if a lane has reached its capacity, preventing further customer additions.

Methods like add_customer, remove_customer, open_lane, close_lane, and display_status handle lane operations. Opening and closing lanes are controlled by the status attribute, ensuring lanes operate according to defined rules.

The use of inheritance facilitates code reuse and abstraction, with the Customer class providing common attributes like name and basket size. However, it's important to note that in a supermarket scenario, a checkout lane should not inherit from a customer, as they represent different entities. A more appropriate design might involve composition or association rather than inheritance.

F2- Customer Management Features: (sanjida)

Customer Implementation:

The Customer class encapsulates customer-related details such as name, basket size, and lottery ticket status. The get_basket_size method returns the basket size, and the get_checkout_time method calculates the checkout time based on whether the customer uses self-service. The award_lottery_ticket method determines if the customer receives a lottery ticket.

The design choice to use a class for customers allows for clear encapsulation and organization of customer-specific functionalities. The lottery ticket randomization introduces variability, making the simulation more dynamic and engaging.

F3-(Both)

SupermarketSimulation Implementation:

The SupermarketSimulation class orchestrates the entire simulation, managing regular and self-service lanes along with customer interactions. The simulation initializes regular and self-service lanes, along with a list to keep track of waiting customers.

Methods like generate_customer, generate_initial_customers, assign_customer_to_lane, open_new_lane, close_empty_lanes, display_simulation_status, display_lottery_info, and run_simulation coordinate various aspects of the simulation.

The simulation loop in the run_simulation method controls the flow, generating customers, assigning them to lanes, opening new lanes, and displaying the status. The modular design ensures easy maintenance and extension of features.

In summary, the implementation design employs object-oriented principles, leveraging classes, methods, and inheritance to create a structured and modular simulation of a supermarket checkout system. The use of lists facilitates dynamic customer and lane management, and the randomization adds realism to customer behaviours. However, some design choices, such as inheritance between Customer and CheckoutLane, could be improved for better representation of supermarket entities.

4. Classes and OOP Features

List the classes you developed and provide an exposition on the choice of classes, class design, and OOP features implemented. List all the classes used in your program and include the attributes and behaviours for each. You may use a class diagram to illustrate these classes – do not include the class code here. Your narrative for section 4.2 should describe the design decisions you made, and the OOP techniques used (abstraction, encapsulation, inheritance/polymorphism). ***Note****: stating definitions here will not get you marks, you must clearly outline how you implemented the techniques in your code and WHY. (****400-600 words****)*

THIS SECTION SHOULD BE COMPLETED INDIVIDUALLY FOR F1 AND F2 AND AS A GROUP FOR F3.

4.1 Classes Used

F1- Lane Management Features:(Israt)

Class: Customer
Attributes: name, lanetype, capacity, status, timestamp, customers, Last_updated_time

F2- Customer Management Features: (sanjida)

Class:customer
Attributes: name, basket_size, lottery_ticket

F3-(Israt)

Class:SupermarketSimulation
Attribute:regular_lanes, self_service_lane, lanes, customer_waiting

4.2 Brief Explanation of Class Design and OOP Features Used

F1 – Lane Management Features:(Israt)

Inheritance: Inherit attributes and methods from the customers class allows it to share common functionality and data

Abstraction: It represents the abstraction of checkout lane with specific attributes.

Encapsulation: Attributes access is controlled through method and also encapsulated.

F2- Customer Management Features: (sanjida)

Abstraction: The class abstracts the properties and behaviors of a customer, including basket size, lottery ticket assignment, and checkout time calculation

Encapsulation: Attributes are encapsulated within the class, and access is controlled through getter methods and class methods.

F3-(Both)

Composition:Composes regular lanes and a self-service lane to represent the supermarket layout.

Abstraction:Manages the overall simulation, abstracting the complexities of customer generation, lane assignment, and resource optimization.

Encapsulation:Attributes and methods are encapsulated within the class, controlling access and maintaining the integrity of simulation-related data.

Polymorphism:Polymorphism is achieved through the use of shared methods like add_customer across different types of lanes.

5. Code for the Classes Created

*Add the **code for each of the classes you have implemented yourself** here. If you have contributed to parts of classes, please highlight those parts in a different colour and label them with your name. Copy and paste relevant code - actual code please, no screenshots! Make it easy for the tutor to read. Add an explanation if necessary – though your in-code comments should be clear enough. You will lose marks if screenshots are provided instead of code. **DO NOT provide a listing of the entire code. You will be marked down if a full code listing is provided, or you include the code as a screenshot**.*

THIS SECTION SHOULD BE COMPLETED INDIVIDUALLY FOR F1 AND F2 AND AS A GROUP FOR F3.

5.1 Class …

F1:

```
class CheckoutLane(Customer):

    def __init__(self, name, lanetype, capacity):
        super().__init__()


        self.name = name

        self.lanetype = lanetype

        self.capacity = capacity

        self.status = 'closed'

        self.timestamp = None

        self.customers = []
        self.last_updated_time = None


    def is_full(self):

        return len(self.customers) >= self.capacity


#F1:Adding customers to a lane:
    def add_customer(self, customer):

        self.customers.append(customer)


#F1:removing customer:
    def remove_customer(self, customer):

        self.customers.remove(customer)
#F1:open a lane:
    def open_lane(self):
```

5.2 CLASS …

5.3 CLASS …

F-3:

```
class SupermarketSimulation:


    def __init__(self, max_customers=40):

        self.regular_lanes = [CheckoutLane(f"L{i}", 'regular', 5) for i in
range(1, 6)]

        self.self_service_lane = CheckoutLane("L6", 'self-service', 15)

        self.lanes = self.regular_lanes + [self.self_service_lane]

        self.customers_waiting = []

    def reset_simulation(self):

        self.customers_waiting = []

        Customer.reset_counter()  # Reset the customer counter

        for lane in self.lanes:

            lane.status = 'closed'

            lane.timestamp = None

            lane.customers = []

            lane.last_updated_time = None
def display_simulation_status(self):

    total_customers_waiting = sum(len(lane.customers) for lane in self.lanes if
lane.status == 'open')

    print(

        f"Total number of customers waiting to check out at
{time.strftime('%H:%M:%S')} is: {total_customers_waiting}")

    for lane in self.lanes:

        lane.display_status()
def run_simulation(self, max_duration):

        # Run the simulation loop

        start_time = time.time()
```

```
        while time.time() - start_time < max_duration:

            self.reset_simulation()
            initial_customers = self.generate_initial_customers()
            self.customers_waiting.extend(initial_customers)

            for customer in initial_customers:
                self.assign_customer_to_lane(customer)

            new_customer = self.generate_customer()
            self.customers_waiting.append(new_customer)
            self.assign_customer_to_lane(new_customer)
            self.open_new_lane()
            self.close_empty_lanes()
            self.display_simulation_status()
            self.display_lottery_info()
            time.sleep(30)    # Simulate 30-second intervals

if __name__ == "__main__":
    simulation = SupermarketSimulation(max_customers=40)
    simulation.display_simulation_status()
    simulation.display_lottery_info()
    simulation.run_simulation(150)
```

…

6. TESTING

Describe the process you took to test your code and to make sure the program functions as required. ***Make sure you include a test plan and demonstrate thorough testing of your own code as well as the integrated code****.*

THIS SECTION SHOULD BE COMPLETED INDIVIDUALLY FOR F1 AND F2 AND AS A GROUP FOR F3.

F-1:

CheckoutLane Testing

Test Plan:

Capacity Management:

Add customers to a lane until it reaches its capacity.

Verify that the is_full method correctly indicates when a lane is full.

Customer Addition and Removal:

Add a customer to a lane.

Remove the customer from the lane.

Verify that the lane's customer list is updated correctly.

Lane Opening and Closing:

Open a closed lane.

Verify that the lane's status is now 'open.'

Close an open lane.

Verify that the lane's status is now 'closed.'

Display Status:

Display the status of a lane with customers.

Verify that the correct output is printed, showing the lane's name, type, status, and the number of customers.

Testing Implementation:

Capacity Management:

```
lane = CheckoutLane("TestLane", 'regular', 2)
assert not lane.is_full()
lane.add_customer(Customer())
lane.add_customer(Customer())
assert lane.is_full()
```

Customer Addition and Removal:

```
lane = CheckoutLane("TestLane", 'regular', 2)
customer = Customer()
lane.add_customer(customer)
assert customer in lane.customers
lane.remove_customer(customer)
assert customer not in lane.customers
```

Lane Opening and Closing:

```
lane = CheckoutLane("TestLane", 'regular', 2)
assert lane.status == 'closed'
lane.open_lane()
assert lane.status == 'open'
lane.close_lane()
assert lane.status == 'closed'
```

Display Status:

```
lane = CheckoutLane("TestLane", 'regular', 2)
lane.add_customer(Customer())
lane.display_status()  # Check the printed output
```

F2:

Customer Testing:

Test Plan:

Basket Size Generation:

Create multiple customers and check that their basket sizes fall within the expected range (1 to 30).

Checkout Time Calculation:

Calculate the checkout time for customers using both self-service and cashier lanes.

Verify that the calculated times are as expected based on the defined logic.

Lottery Ticket Awarding:

Create customers with different basket sizes.

Verify that customers with basket sizes greater than or equal to 10 sometimes receive a lottery ticket.

Display Customer Details:

Display the details of a customer.

Verify that the correct information is printed, including the name, basket size, lottery status, and checkout time.

Testing Implementation:

Basket Size Generation:

```
customer = Customer()
assert 1 <= customer.basket_size <= 30
```

Checkout Time Calculation:

```
customer = Customer()
self_service_time = customer.get_checkout_time(True)
cashier_time = customer.get_checkout_time(False)
assert self_service_time == customer.basket_size * 6
assert cashier_time == customer.basket_size * 4
```

Lottery Ticket Awarding:

```
customer1 = Customer()
assert customer1.lottery_ticket in [True, False]
customer2 = Customer()
assert customer2.lottery_ticket in [True, False]
```

Display Customer Details:

```
customer = Customer()
customer.display_customer_details()  # Check the printed output
```

F3:

SupermarketSimulation Testing

Test Plan:

Initial Configuration:

Verify that the supermarket simulation initializes with the correct number of regular and self-service lanes.

Generate Customers:

Generate a customer and check that their attributes are correctly initialized.

Assign Customers to Lanes:

Generate initial customers and assign them to lanes.

Verify that customers are assigned based on basket size and lane availability.

Open and Close Lanes:

Open a closed lane and verify its status.

Close an open lane and verify its status.

Display Simulation Status:

Display the simulation status and verify the correctness of the printed information.

Run Simulation:

Run the simulation for a specific duration.

Verify that the simulation executes without errors and produces the expected output.

Testing Implementation:

Initial Configuration:

```
simulation = SupermarketSimulation(max_customers=40)
assert len(simulation.regular_lanes) == 5
assert len(simulation.lanes) == 6
```

Generate Customers:

```
simulation = SupermarketSimulation(max_customers=40)
customer = simulation.generate_customer()
assert customer.name.startswith("C")
assert 1 <= customer.basket_size <= 30
```

Assign Customers to Lanes:

```
simulation = SupermarketSimulation(max_customers=40)
customer = simulation.generate_customer()
simulation.assign_customer_to_lane(customer)
assert customer in simulation.self_service_lane.customers or any(customer in lane.customers for lane in simulation.regular_lanes)
```

Open and Close Lanes:

```
simulation = SupermarketSimulation(max_customers=40)
lane = simulation.regular_lanes[0]
assert lane.status == 'closed'
lane.open_lane()
assert lane.status == 'open'
lane.close_lane()
assert lane.status == 'closed'
```

Display Simulation Status:

```
simulation = SupermarketSimulation(max_customers=40)
simulation.display_simulation_status()  # Check the printed output
```

Run Simulation:

```
simulation = SupermarketSimulation(max_customers=40)
simulation.run_simulation(150)
# Verify the execution of the simulation by inspecting the printed output during the run.
```

Group Reflection:

Testing for each component and the overall simulation involved creating specific scenarios to cover various functionalities. This approach ensures that each class and method behaves as expected. Additionally, running the entire simulation with specific durations allows for testing the integration of all components. Continuous monitoring of printed outputs during testing helps identify any unexpected behavior or errors in the code logic. Overall, the testing process ensures the reliability and correctness of the supermarket simulation

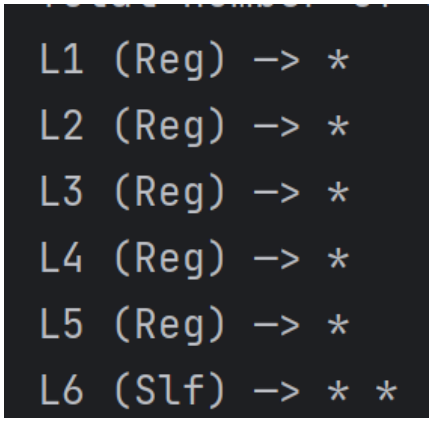
7. Annotated Screenshots Demonstrating Implementation

*Provide screenshots that demonstrate the features implemented running – i.e. showing the output produced by all of the subfeatures. Annotate each screenshot and if necessary, provide a brief description for **each** (**up to 100 words**) to explain the code in action.*

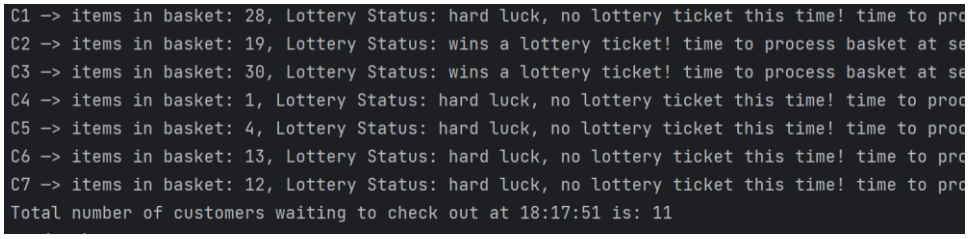
THIS SECTION SHOULD BE COMPLETED INDIVIDUALLY FOR F1 AND F2 AND AS A GROUP FOR F3.

7.1 Feature F1

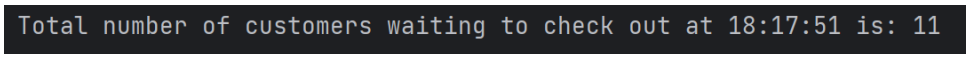
a. Sub-feature I- screenshots …



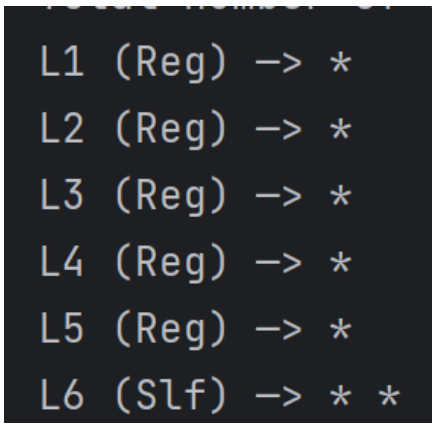
b. Sub-feature II- screenshots …



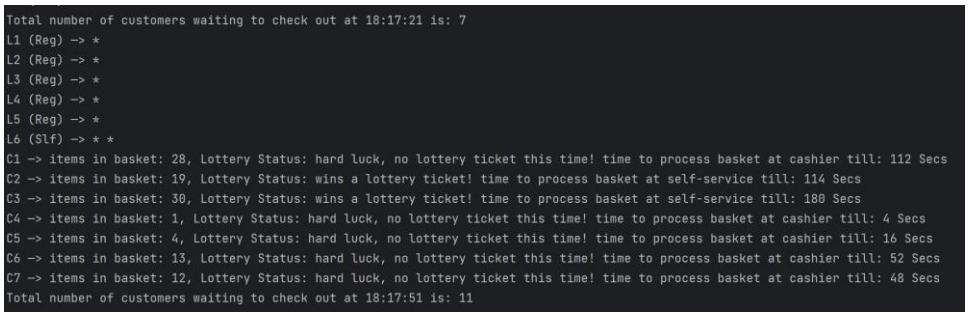
c. Sub-feature III- screenshots …



d. Sub-feature IV- screenshots …

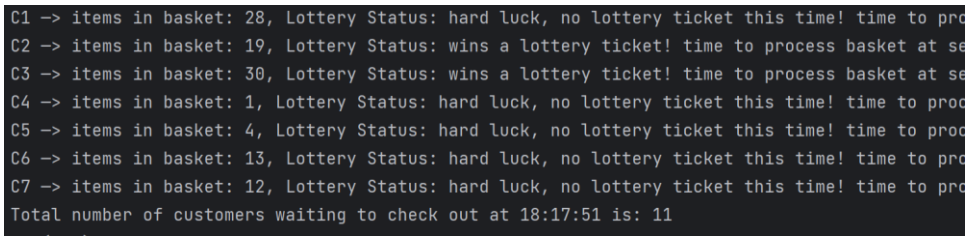


e. SUB-FEATURE V- SCREENSHOTS …

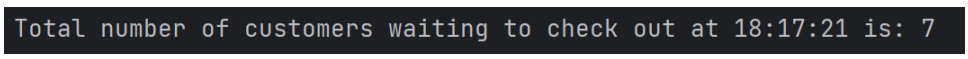


7.2 FEATURE F2

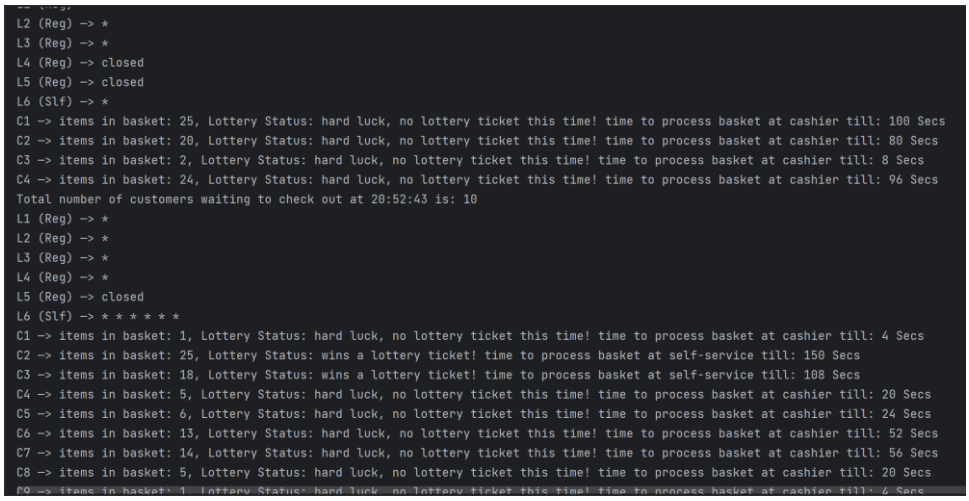
a. SUB-FEATURE I- SCREENSHOTS …



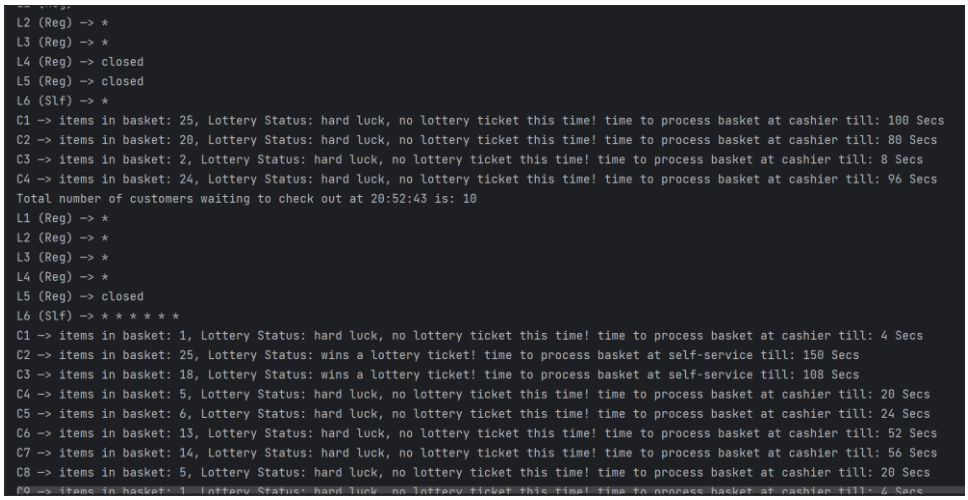
b. SUB-FEATURE II- SCREENSHOTS …



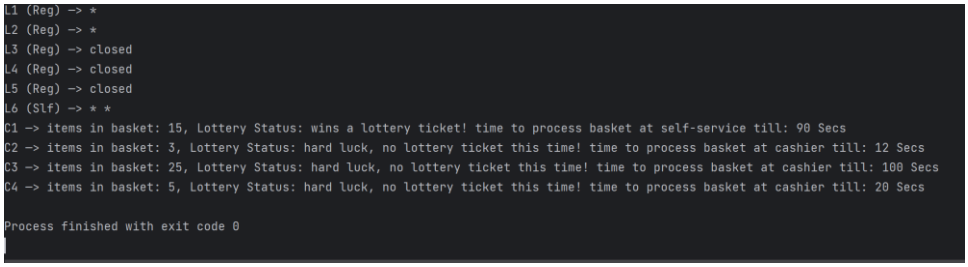
c. SUB-FEATURE III- SCREENSHOTS …



d. SUB-FEATURE IV- SCREENSHOTS …



e. SUB-FEATURE V- SCREENSHOTS …



7.3 FEATURE F3

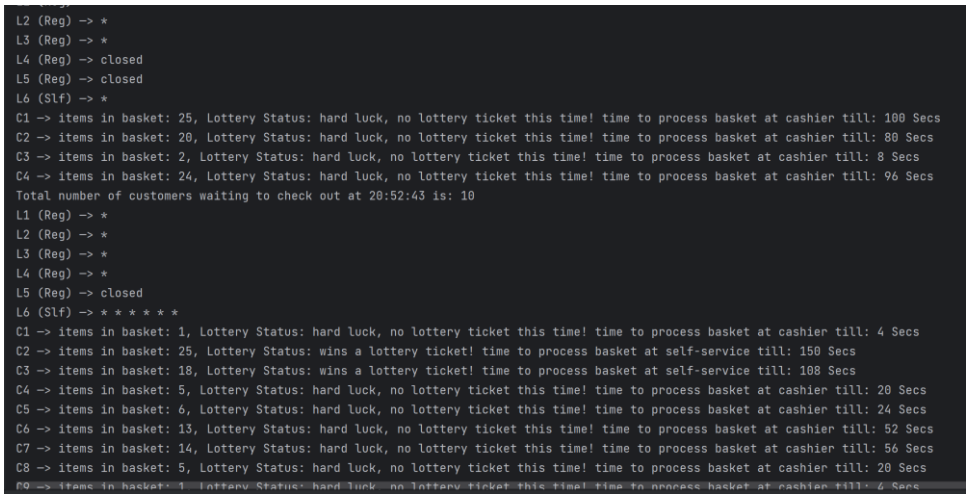
a. SUB-FEATURE I- SCREENSHOTS …

```
Total number of customers waiting to check out at 20:53:43 is: 6
L1 (Reg) -> *
L2 (Reg) -> *
L3 (Reg) -> *
L4 (Reg) -> *
L5 (Reg) -> closed
L6 (Slf) -> * *
```

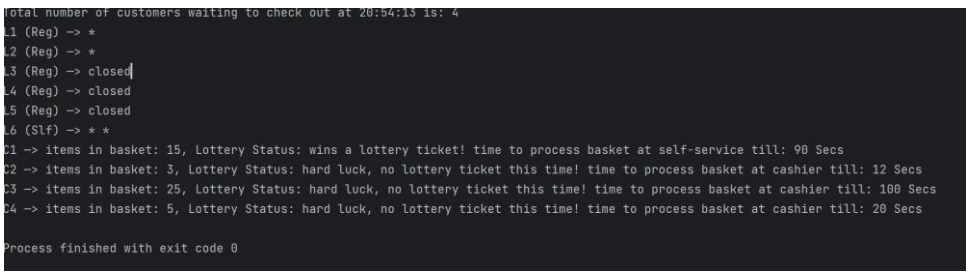
b. SUB-FEATURE II- SCREENSHOTS …

```
Total number of customers waiting to check out at 20:52:13 is: 4
```

c. SUB-FEATURE III- SCREENSHOTS …



d. SUB-FEATURE IV- SCREENSHOTS …



8. OpenAI Comparison

Provide the code generated using OpenAI along with a listing of the code you initially wrote from scratch in a table showing the generated and your code side-by-side for each feature. Examine and explain the generated code's design, describing its quality and efficiency compared to the initial code you wrote. The narrative must also describe how you used the generated code to improve your own code or describe how the generated code may be improved.

OpenAI Code:

The code defines classes for Customer and CheckoutLane, which are well-structured.
The generate_customerfunction creates a random customer.
The assign_to_lanefunction assigns a customer to an open lane based on certain conditions.
The simulation runs for a specified duration, generating random customers, assigning them to lanes, and processing checkouts.
The code handles opening and closing lanes based on the number of customers.

Own Code:
The code defines classes for Customer, CheckoutLane, and SupermarketSimulation.
The reset_simulationmethod resets the simulation parameters.
The generate_customermethod creates a new customer with a random basket size and a chance to win a lottery ticket.
The assign_customer_to_lanemethod assigns a customer to an appropriate lane based on their basket size.
The simulation loop generates initial customers, assigns them to lanes, introduces a new customer, and manages lane opening/closing.
There's a 30-second sleep between simulation iterations.

Feedback:
Both sets of code seem to have similar functionalities and are centered around a supermarket checkout simulation.
Your code seems well-structured, and you've effectively used classes to encapsulate different aspects of the simulation.
It's good to see that you've incorporated a sleep interval in your simulation to mimic a real-time scenario.
In the "Own Code," you've implemented methods to display the simulation status and lottery information, providing clarity and modularity.

9. SELF-ASSESSMENT

Please assess yourself objectively for each section shown below and then enter the total mark you expect to get. Marks for each assessment criterion are indicated between parentheses.

Code development (70)

a. Features Implemented [36] (group work and integration will be assessed here)

Partner A or Partner B features (up to 18)

Sub-features have not been implemented – 0
Attempted, not complete or very buggy – 1 to 5
Implemented and functioning without errors but not integrated – 6 to 10
Implemented and fully integrated but buggy – 11 to 15
Implemented, fully integrated and functioning without errors – 16 to 18

Group Features (up to 18)

Sub-features has not been implemented – 0
Attempted, not complete or very buggy – 1 to 5
Implemented and functioning without errors but not integrated – 6 to 10
Implemented and fully integrated but buggy – 11 to 15
Implemented, fully integrated and functioning without errors – 16 to 18

For this criterion I think I got: 32 out of 36

b. Use of OOP techniques [24]

Abstraction (up to 8)

No classes have been created – 0
Classes have been created superficially and not instantiated or used – 1 or 2
Classes have been created but only some have been instantiated and used – 3 or 4
Useful classes and objects have been created and used correctly – 5 or 6
The use of classes and objects exceeds the specification – 7 or 8

Encapsulation (up to 8)

No encapsulation has been used – 0
Class variables and methods have been encapsulated superficially – 1 to 3
Class variables and methods have been encapsulated correctly – 4 to 6
The use of encapsulation exceeds the specification – 6 to 8

Inheritance or polymorphism (up to 8)

No inheritance or polymorphism has been used – 0
Inheritance or polymorphism has been used superficially – – 1 to 3
Inheritance or polymorphism has been used correctly – 4 to 6
The use of inheritance or polymorphism exceeds the specification – 6 to 8

For this criterion I think I got: 22 out of 24

c. Quality of Code [10]

Code Duplication (up to 4)

Code contains too many unnecessary code repetition – 0
Regular occurrences of duplicate code – 1
Occasional duplicate code – 2
Very little duplicate code – 3

No duplicate code – 4

PEP8 Conventions and naming of variables, methods and classes (up to 3)

PEP8 and naming convention has not been used – 0
PEP8 and naming convention has been used occasionally – 1
PEP8 and naming convention has been used regularly – 2
PEP8 convention used professionally and all items have been named correctly – 3

In-code Comments (up to 3)

No in-code comments – 0
Code contains occasional in-code comments – 1
Code contains useful and regular in-code comments – 2
Thoroughly commented, good use of docstrings, and header comments describing.py files – 3

For this criterion I think I got: 8 out of 10

2. Documentation (20)

Design (up to 10) clear exposition about the design and decisions for OOP use

The documentation cannot be understood on first reading or is mostly incomplete – 0
The documentation is readable, but a section(s) are missing – 1 to 3
The documentation is complete – 4 to 6
The documentation is complete and of a high standard – 7 to 10

Testing (10)

Testing has not been demonstrated in the documentation – 0
A test plan has been included but is incomplete – 1 or 2
A test plan has been included with some appropriate test cases – 3 to 6
A full test plan has been included with thorough test cases and evidence of carrying it out – 7 to 10

For this criterion I think I got: 17 out of 20

3. Acceptance Test - Demonstration (10)

Final Demo (up to 10)

Not attended or no work demonstrated – 0
Work demonstrated was not up to the standard expected, superficial team contribution – 1 to 3
Work demonstrated was up to the standard expected, sufficient team contribution – 4 to 7
Work demonstrated exceeded the standard expected – 8 to 10

For this criterion I think I got: 6 out of 10

I think my overall mark would be: 84 out of 100

APPENDIX A: CODE LISTING

*Provide a complete listing of all the *.py files in your PyCharm project. Make sure your code is well commented and applies professional Python convention (refer to PEP 8 for details). The code listed here must match that uploaded to Moodle. Please copy and paste the actual code – no screenshots please! You will lose marks if screenshots are provided instead of code. Clearly label the parts each partner created with their name and SID.*

```
import random

import time
#F2:number of item and the basket size:
class Customer:

    counter = 1

    def __init__(self):

        self.name = f"C{Customer.counter}"

        Customer.counter += 1

        self.basket_size = random.randint(1, 30)

        self.lottery_ticket = self.award_lottery_ticket()


    def get_basket_size(self):

        return self.basket_size


#F2:Getting the checkout time:
    def get_checkout_time(self, self_service ):

        time = 6 if self_service else 4

        return self.basket_size * time


#F2:Award a lottery ticket :
    def award_lottery_ticket(self):

        return self.basket_size >= 10 and random.choice([True, False])



    def display_customer_details(self):

        lottery_status = 'wins a lottery ticket!' if self.lottery_ticket else 'hard luck, no
```

```
lottery ticket this time!'
        print(f"{self.name} -> items in basket: {self.basket_size}, "
              f"Lottery Status: {lottery_status} "
              f"time to process basket at {'self-service' if self.lottery_ticket else 'cashier'}
till: "
              f"{self.get_checkout_time(self.lottery_ticket)} Secs")


    @classmethod
    def reset_counter(cls):
        cls.counter = 1

#F1: generating a lane:
class CheckoutLane(Customer):
    def __init__(self, name, lanetype, capacity):
        super().__init__()

        self.name = name
        self.lanetype = lanetype
        self.capacity = capacity
        self.status = 'closed'
        self.timestamp = None
        self.customers = []
        self.last_updated_time = None

    def is_full(self):
        return len(self.customers) >= self.capacity

#F1:Adding customers to a lane:
    def add_customer(self, customer):
```

```
        self.customers.append(customer)


#F1:removing                                                    customer:
    def                             remove_customer(self,                   customer):

        self.customers.remove(customer)
#F1:open                                        a                               lane:
    def                                                             open_lane(self):

        if                      self.status                     ==                  'closed':

            self.status                             =                           'open'

            self.timestamp                      =                   time.strftime("%H:%M:%S")

            self.last_updated_time                      =                   self.timestamp

#F1:close                                       a                               lane:
    def                                                             close_lane(self):

        if                      self.status                     ==                  'open':

            self.status                             =                           'closed'

            self.timestamp                              =                           None

            self.last_updated_time                  =                   time.strftime("%H:%M:%S")

#F1:Display                                     lane                            status:
    def                                                             display_status(self):

        print(f"{self.name}  ({'Reg'  if  self.lanetype  ==  'regular'  else  'Slf'})  ->  "

              f"{' '.join('*' for _ in range(len(self.customers))) if self.status == 'open' else
'closed'}")

#f3
class                                                           SupermarketSimulation:


    def                             __init__(self,                      max_customers=40):

        self.regular_lanes  =  [CheckoutLane(f"L{i}",  'regular',  5)  for  i  in  range(1,  6)]

        self.self_service_lane      =      CheckoutLane("L6",      'self-service',      15)

        self.lanes      =      self.regular_lanes      +      [self.self_service_lane]

        self.customers_waiting                          =                           []
```

```
    def reset_simulation(self):
        self.customers_waiting = []
        Customer.reset_counter() # Reset the customer counter
        for lane in self.lanes:
            lane.status = 'closed'
            lane.timestamp = None
            lane.customers = []
            lane.last_updated_time = None
#F2:generate a customer:
    def generate_customer(self):
        return Customer()

    def generate_initial_customers(self):
        num_initial_customers = random.randint(1, 10)
        return [self.generate_customer() for _ in range(num_initial_customers)]

    def assign_customer_to_lane(self, customer):
        if customer.get_basket_size() < 10 and not self.self_service_lane.is_full():
            self.self_service_lane.add_customer(customer)
        else:
            available_lanes = [lane for lane in self.regular_lanes if not lane.is_full()]
            if available_lanes:
                shortest_lane = min(available_lanes, key=lambda lane: len(lane.customers))
                shortest_lane.add_customer(customer)
            else:
                print("All lanes are full. Unable to assign customer.")
```

```
    def open_new_lane(self):
        if all(lane.is_full() for lane in self.lanes):
            print("All lanes are full. Unable to open a new lane.")
        else:
            for lane in self.lanes:
                lane.open_lane()
#F1:
    def close_empty_lanes(self):
        for lane in self.lanes:
            if lane.status == 'open' and not lane.customers:
                lane.close_lane()
#F3:
    def display_simulation_status(self):
        total_customers_waiting = sum(len(lane.customers) for lane in self.lanes if lane.status
== 'open')
        print(
            f"Total number of customers waiting to check out at {time.strftime('%H:%M:%S')} is:
{total_customers_waiting}")
        for lane in self.lanes:
            lane.display_status()
#F2:
    def display_lottery_info(self):
        for customer in self.customers_waiting:
            customer.display_customer_details()
#F3:
    def run_simulation(self, max_duration):
        # Run the simulation loop
        start_time = time.time()
        while time.time() - start_time < max_duration:

            self.reset_simulation()
```

```
        initial_customers = self.generate_initial_customers()
        self.customers_waiting.extend(initial_customers)

        for customer in initial_customers:
            self.assign_customer_to_lane(customer)

        new_customer = self.generate_customer()
        self.customers_waiting.append(new_customer)
        self.assign_customer_to_lane(new_customer)
        self.open_new_lane()
        self.close_empty_lanes()
        self.display_simulation_status()
        self.display_lottery_info()
        time.sleep(30)  # Simulate 30-second intervals

if __name__ == "__main__":
    simulation = SupermarketSimulation(max_customers=40)
    simulation.display_simulation_status()
    simulation.display_lottery_info()
    simulation.run_simulation(150)
```