# 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** | **Features** | |  |  |  |  |  |
| **or are partly working.** | **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 | | | | | | | |

# 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.*

* 1. BUGS

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

Customer can switch lanes was not Implemented. Even though we are pushing all the customers to waiting lane when regular lane and self lane has no space left. However this feature partially implemented when customer needs to re-join to regular from waiting lane.

## 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.*

There might be a weakness in printing lanes situation. The printing function await for its function call from insertCustomer function . Then it prints lanes current situation.

# 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.*

Our code is divided in 4 different files. And they are f1.py , f2.py , f3.py (could be referred as f3) and sharedState.py.

**Features implemented in f1.py**:

The f1.py has a few classes containing some functionalities to serve different purposes. As the system requires some lanes for customers to enter for checking up on their products and a waiting line if regular and self-service have no longer space, We declared seven lists to maintain the process. We have three classes, with some functions in the file serving different purposes. The first class is holding a function that inserts customers in different lanes following the given requirements.

At begging, we started by inserting customers that had products less than 9 only in the self-serving lane. And then the rest will be inserted into different regular lanes. However, sometimes there might be an issue where the self-service lane is full and a newly generated customer has enough products to join the self-service lane. To solve this, we push them to the regular lane, following the requirements.

We face some challenges in doing that. Firstly, what should we do if we do not have enough spaces to let the newly generated customer join? We pushed the regular customer into waiting. Later, if we find spaces in the regular or self-service lane, we extract customers from the waiting line to different lanes.

However, this insertion function, known as insertCustomer, depends on other functions where new customers are generated (this feature is part of f2). When the insertCustomer function receives a new customer, it first calls another function called print\_stat, which is under the Lane\_situation class. As long as the function is called from insertCustomer, the print\_stat function starts printing current situations.

The f1.py file also has a class lane\_services; under this, there are six functions. All of them have similar working procedures. They clear the lanes after checking the products in their buckets. These lane clearing functions have some dependencies. They depend on a variable called is\_done. repeat the process as long as is\_done is true. These functions are also under the threading mechanism.

#### Features implemented in f2.py:

These files consist of two classes. One of them is called CreateCustomerElement, and the other is called Lottery. The first function declared as generate\_customer depends on a thread. So it works quite independently. When the thread started running, it generated between 1 and 10 customers and sent them to the generate\_product function under the same class, which returned a list of the generated customers. Each customer may have 1–30 products in their bucket. The the list of the customers product send to insertCustomer function.

The lottery class contains a function called lottery\_generator that receives a variable, which is the number of products. Depending on the size of the products, it rewards customers with a lottery. Also print the required time for using different tills.

**Features implemented in f3.py :**

The shared state known as is\_done is changed here. It contains a counter to count time. If the time is up, the function changes its value to false, and the whole system stops working. The file also controls threading. Running the file will trigger all the threads and functions as long as the shared state (is\_done) is set to false.

# 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.*

## Classes Used

Classes used in f1.py:

* + 1. Insertion,
    2. Lane\_services
    3. Lane\_situation

Classes used in f2.py:

1. CreateCustomerElement
2. Lottery

Implemented OOP feature :

1. Class and function
2. Abstraction
3. Polymorphism

## Brief Explanation of Class Design and OOP Features Used

### Insertion,

The insertion contains a function called insertCustomer. It receives lists and inserts the customers in lanes based on the given criteria. However, while inserting customers into the lane, we faced some problems. The system generates customers after every 30 seconds continuously. And we can't insert more than 40 people into lanes. At some point in the simulation, we might face some problems like insufficient spaces, customers waiting to join lanes, or the lane may have a few spaces left, but we are not allowed to push newly generated

customers into lanes as the process will exceed the customer limit. This function solves all the issues described here.

The lane\_service is created to handle the customer who joins the lane to checkout their products. It has six functions, and all are running using threads. As long as there are customers in the lane, each of the functions will delete customers from their own lane on a regular basis.

Another class called Lane\_situation has a function. print\_stat is the function. After generating a new customer, this function will be called from insertCustomer. Which will trigger the mechanism to print the lane situation, followed by some additional message.

In the f2.py file, we have two functions called Lottery and the other one is CreateCustomerElement.Create lottery Check the number of products in the customer bucket. If the amount is greater than 9, then it might reward a customer with a lottery, making him lucky. The other class has a customer generator function that generates customers after 30 seconds continuously. The amount is sent to another function, and we wait for it to return a list with customer bucket items. After receiving the list, this function sends it again to another function to insert them in different lanes.

### Implemented OOP feature like class and functions:

The whole system consists of many different classes and functions. We have four Python files in our system. Each has a unique task to perform. As we implemented the feature of inserting customers into the lane, we declared a class called Insertion, which also has a function named InsertCustomer. We have a class called Lane\_situation that contains a function that prints customers current situation across the lanes. It also indicates the lanes type (either regular or self). To clear the current lane customers, we are using multi-threading for each lane. The functions that are clearing the lanes are under a class called Lane\_services. The situation continues like this. To generate customers, their products, and the lottery, we are using classes and functions.

Abstraction:

As our file consists of 4 pages, we have followed this structure properly. We abstracted functions from a file to another file. Accessing shared variables from one file to another.

Polymorphism:

We declared our customer lanes as a global list in a file. Later, we will manipulate them for different functions. Inserting values into them by using a function. Deleting the values from

# 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.*

* 1. CLASS …

### Class and functions made by Md Mahbub Anam Tanim - From f1.py:

import time import copy

from datetime import datetime import sharedState

# declared lanes regular\_lane1 = [] regular\_lane2 = [] regular\_lane3 = [] regular\_lane4 = [] regular\_lane5 = [] self\_lane = [] waiting\_lane = []

class Insertion:

# function for inserting customers in different lanes # receive data through a list called a

def insertCustomer(self, a): from f2 import Lottery

# This function will be called after to print lane situation

lane\_inst = Lane\_situation() lane\_inst.print\_stat()

# total customers are waiting present\_lane\_length =

len(regular\_lane1)+len(regular\_lane2)+len(regular\_lane3)+len(regular\_lane4)+len(regular\_lane5)+len(self\_lane

)

# execute when the regular lane has no space but the self-lane has

#sort out the incoming, newly generated customer

If (len(regular\_lane1)+len(regular\_lane2)+len(regular\_lane3)+len(regular\_lane4)+len(regular\_lane5)) == 25 and len(self\_lane) < 15:

print("Reguler lane over flow!")

# Remember the indexes that need to be deleted.

remove\_ele = []

# Customers with more than nine products move to the waiting lane.

for pos in range(len(a)): if a[pos] > 9:

print("Customer with ",a[pos]," products has to join in waiting\_lane line") waiting\_lane.append(a[pos])

remove\_ele.append(pos)

# delete them from list a by using remove\_ele indexes.

for pos in reversed(remove\_ele): if 0 <= pos < len(a):

a.pop(pos)

# If the current total customers, along with lists, exceed exeed limit of 40

if (present\_lane\_length + len(a) ) > 40:

k = 0

while len(a) != k:

waiting\_lane.append(a[k]) # appending all customers in the waiting lane

k += 1

# return if there is no space left in any lanes

if present\_lane\_length == 40:

print("Both regular and selfservice lanes are full. Customer have to wait.") print("First in waiting\_lane will enter to lane first")

return

tmp = [] # temporary array to hold customers from the waiting lane

customer\_index = [] # Remember their indexes from the waiting lane.

lnth = 40 - present\_lane\_length # how many spaces left in regular and service lane

# If the current total customers, along with lists, exceed exeed limit of 40

if len(waiting\_lane) > 0 and lnth > 0:

tmp\_reg\_line = len(regular\_lane1) + len(regular\_lane2) + len(regular\_lane3) + len(regular\_lane4) + len(regular\_lane5) # total customers in regular lane

tmp\_slf\_line = len(self\_lane) # total customers in self services till k = 0

# logic: check space availability in lanes and customers are waiting

while k != len(waiting\_lane) and lnth != 0:

if tmp\_reg\_line < 25 and tmp\_slf\_line < 15: #If both the regular and self-lanes have space left,

tmp.append(waiting\_lane[k]) if waiting\_lane[k] > 9:

tmp\_reg\_line += 1 else:

tmp\_slf\_line += 1

customer\_index.append(k) # remember indexes for appended customer from waiting lnth -= 1

k += 1

elif tmp\_reg\_line == 25 and tmp\_slf\_line < 15: # regular space full, self service left if waiting\_lane[k] > 9:

k += 1

else:

tmp.append(waiting\_lane[k]) customer\_index.append(k) tmp\_slf\_line += 1

lnth -= 1

k += 1

elif tmp\_reg\_line < 25 and tmp\_slf\_line == 15: # regular space left, self service full tmp.append(waiting\_lane[k])

customer\_index.append(k) tmp\_reg\_line += 1

lnth -= 1

k += 1

# delete the customer from the waiting lane using the customer index list.

if customer\_index:

for pos in reversed(customer\_index): if 0 <= pos < len(waiting\_lane):

waiting\_lane.pop(pos)

# Copy the temporary customers to list a

if tmp:

a = copy.copy(tmp)

lottery\_ins = Lottery()

lottery\_ins.lottery\_generator(a) # Check the letter by calling the function.

# Push customers with products to different lanes.

i = 0

while i != len(a):

if len(self\_lane) < 15 and a[i] < 10: # total product in bucked less than 9 and has space left in self than 10 products

self\_lane.append(a[i])

else: #or we have spaces left in others lanes.

if len(regular\_lane1) < 5: regular\_lane1.append(a[i])

elif len(regular\_lane2) < 5: regular\_lane2.append(a[i])

elif len(regular\_lane3) < 5:

regular\_lane3.append(a[i]) elif len(regular\_lane4) < 5:

regular\_lane4.append(a[i])

elif len(regular\_lane5) < 5:

regular\_lane5.append(a[i]) i += 1

# customer checkout in different lanes also multithreaded class Lane\_services:

# delete customer from lane following criteria if customer exists def clearing\_regular\_lane1(self):

while sharedState.is\_done: if len(regular\_lane1) > 0:

time.sleep(regular\_lane1[0]\*4) regular\_lane1.pop(0)

else:

continue

# delete customer from lane following criteria if customer exists def clearing\_regular\_lane2(self):

while sharedState.is\_done: if len(regular\_lane2) > 0:

time.sleep(regular\_lane2[0]\*4) regular\_lane2.pop(0)

else:

continue

# delete customer from lane following criteria if customer exists def clearing\_regular\_lane3(self):

while sharedState.is\_done: if len(regular\_lane3) > 0:

time.sleep(regular\_lane3[0]\*4) regular\_lane3.pop(0)

else:

continue

# delete customer from lane following criteria if customer exists def clearing\_regular\_lane4(self):

while sharedState.is\_done: if len(regular\_lane4) > 0:

time.sleep(regular\_lane4[0]\*4) regular\_lane4.pop(0)

else:

continue

# delete customer from lane following criteria if customer exists def clearing\_regular\_lane5(self):

while sharedState.is\_done: if len(regular\_lane5) > 0:

time.sleep(regular\_lane5[0]\*4) regular\_lane5.pop(0)

else:

continue

# delete customer from lane following criteria if customer exists def clearing\_self\_lane(self):

while sharedState.is\_done: if len(self\_lane) > 0:

time.sleep(self\_lane[0]\*6) self\_lane.pop(0)

else:

continue

# class holds print\_stat function class Lane\_situation:

# print current situation of lanes after 30 sec continuously def print\_stat(self):

current\_datetime = datetime.now()

print("Total number of customers waiting\_lane to check out at", current\_datetime, " is: ", len(regular\_lane1)+len(regular\_lane2)+len(regular\_lane3)+len(regular\_lane4)+len(regular\_lane5)+len(self\_lane

))

lanes = [regular\_lane1, regular\_lane2, regular\_lane3, regular\_lane4, regular\_lane5, self\_lane, waiting\_lane] # creating a list containing lists for better iteration process

# idea generated from chat gpt

for i, lane in enumerate(lanes, start=1): length = len(lane)

if length < 1: if i == 7:

print("No customer in waiting\_lane") elif i == 6:

print(f"L{i}(Self) ->", end=" ") print("Open", end=" ")

else:

if i != 1:

print(f"L{i}(Reg) ->", end=" ") print("Close", end=" ")

else:

print(f"L{i}(Reg) ->", end=" ") print("Open", end=" ")

else:

if i == 7:

print(length,"customers are waiting\_lane to join in line") else:

if i == 6:

print(f"L{i}(Self) ->", end=" ") else:

print(f"L{i}(Reg) ->", end=" ")

for \_ in lane: print("\*", end=" ")

print("")

* 1. CLASS …

#### Class and function made by Nobel Ahmed: From f2.py:

import random

import time import sharedState

from f1 import Insertion

# class containing two functions class CreateCustomerElement:

def generate\_product(self,customer\_number): # receive parameter, generate bucket products and return customer\_with\_basket\_items = [random.randint(1, 30) for \_ in range(customer\_number)]

return customer\_with\_basket\_items

def generate\_customer(self): # Generate between 1-10 customers using thread insert\_ins = Insertion() # importing the class called Insertion from f1.py while sharedState.is\_done:

customer\_number = random.randint(1,10)

cus\_with\_prod = self.generate\_product(customer\_number) # calling function for generating items for each customer

insert\_ins.insertCustomer(cus\_with\_prod) # send them to the face insertion process in f1.py time.sleep(30)

class Lottery:

#generate a lottery for each customer in the list

def lottery\_generator(self, a): b = 1

print("### Customer details ###") for cus in a: #cus = customer

if cus > 9:

random\_boolean = bool(random.randint(0, 1)) # generate true or false for every customer entering in line with more thn 9 products in their buscket

if random\_boolean:

print(f'C{b}', "-> Items in basket: ", cus, ", wins a lottery ticket!") print("Time to process basket at cashier till :", cus\*4) print("Time to process basket at self-service till :", cus\*6)

else:

print(f'C{b}', "-> Items in basket: ", cus, ", hard luck!") print("Time to process basket at cashier till :", cus\*4) print("Time to process basket at self-service till :", cus\*6)

else:

print(f'C{b}', "-> Items in basket: ", cus) print("Time to process basket at cashier till :", cus\*4) print("Time to process basket at self-service till :", cus\*6)

b += 1

* 1. CLASS …

**Combined part:**

import threading import time import sharedState import f1

from f2 import CreateCustomerElement

from f1 import Lane\_services , Lane\_situation

customer\_ins = CreateCustomerElement() service\_ins = Lane\_services() situation\_ins = Lane\_situation()

# targeting functions to start thread

# idea of using thread was taken from chat gpt

# Each thread below is responsible for clearing lane.

service\_thread1 = threading.Thread(target=service\_ins.clearing\_regular\_lane1) service\_thread2 = threading.Thread(target=service\_ins.clearing\_regular\_lane2) service\_thread3 = threading.Thread(target=service\_ins.clearing\_regular\_lane3) service\_thread4 = threading.Thread(target=service\_ins.clearing\_regular\_lane4) service\_thread5 = threading.Thread(target=service\_ins.clearing\_regular\_lane5) service\_thread6 = threading.Thread(target=service\_ins.clearing\_self\_lane) generateThread = threading.Thread(target=customer\_ins.generate\_customer)

# thread starting generateThread.start() service\_thread1.start() service\_thread2.start() service\_thread3.start() service\_thread4.start() service\_thread5.start() service\_thread6.start()

t = 0

while sharedState.is\_done: if t > 300:

sharedState.is\_done = False time.sleep(30)

t += 30

# waiting threads to finish service\_thread1.join() service\_thread2.join() service\_thread3.join() service\_thread4.join() service\_thread5.join() service\_thread6.join() generateThread.join()

# 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.*

Testing code for f2.py:

### To check customer generation:

Initial approach was to create customer using thread .

import threading import random import time

def generate\_customer():

customer\_number = random.randint(1,10)

customer\_with\_basket\_items = [random.randint(1, 30) for \_ in range(customer\_number)] print(customer\_with\_basket\_items)

time.sleep(30)

generate\_thread = threading.Thread(target=generate\_customer) generate\_thread.start()

#### Once it was successful. It was implemented to the main f2.py file

**Later I upgraded the code using classes and connecting other functions:**

class CreateCustomerElement:

def generate\_product(self,customer\_number): # receive parameter, generate bucket products and return customer\_with\_basket\_items = [random.randint(1, 30) for \_ in range(customer\_number)]

return customer\_with\_basket\_items

def generate\_customer(self): # Generate between 1-10 customers using thread insert\_ins = Insertion() # importing the class called Insertion from f1.py while sharedState.is\_done:

customer\_number = random.randint(1,10)

cus\_with\_prod = self.generate\_product(customer\_number) # calling function for generating items for each customer

insert\_ins.insertCustomer(cus\_with\_prod) # send them to the face insertion process in f1.py time.sleep(30)

### And its threading moved to the f3.py files Generate lottery functions :

I created a list consist of some data : where a = [12,4,5,29,2,4,15]

Then itterating the list like this:

import random

a = [12,4,5,29,2,4,15]

for cus in a: #cus = customer b = 1

if cus > 9:

random\_boolean = bool(random.randint(0, 1)) # generate true or false for every customer entering in line with more thn 9 products in their buscket

if random\_boolean:

print("### Customer details ###")

print(f'C{b}', "-> Items in basket: ", cus, ", wins a lottery ticket!") print("Time to process basket at cashier till :", cus\*4) print("Time to process basket at self-service till :", cus\*6)

else:

print("### Customer details ###")

print(f'C{b}', "-> Items in basket: ", cus, ", hard luck!") print("Time to process basket at cashier till :", cus\*4) print("Time to process basket at self-service till :", cus\*6)

else:

print("### Customer details ###") print(f'C{b}', "-> Items in basket: ", cus)

print("Time to process basket at cashier till :", cus\*4) print("Time to process basket at self-service till :", cus\*6)

b += 1

#### Later updated Like this:

class Lottery:

# generate a lottery for each customer in the list def lottery\_generator(self, a):

b = 1

print("### Customer details ###") for cus in a: #cus = customer

if cus > 9:

random\_boolean = bool(random.randint(0, 1)) # generate true or false for every customer entering in line with more thn 9 products in their buscket

if random\_boolean:

print(f'C{b}', "-> Items in basket: ", cus, ", wins a lottery ticket!") print("Time to process basket at cashier till :", cus\*4) print("Time to process basket at self-service till :", cus\*6)

else:

print(f'C{b}', "-> Items in basket: ", cus, ", hard luck!") print("Time to process basket at cashier till :", cus\*4) print("Time to process basket at self-service till :", cus\*6)

else:

print(f'C{b}', "-> Items in basket: ", cus)

print("Time to process basket at cashier till :", cus\*4) print("Time to process basket at self-service till :", cus\*6)

b += 1

In this upgradation process now it depends on a function from f1.py. Which call the function and insert a list called a.

### From f1.py file:

**I was inserting customers in different lanes like this :**

a = [12,23,11,6,7,12]

lane1 = [] lane2 = [] lane3 = [] lane4 = [] lane5 = [] lane6 = [] waiting = [] i = 0

while len(a) != i:

if a[i] > 9 or len(lane6) > 15: if len(lane1) < 5:

lane1.append(a[i]) elif len(lane2) < 5:

lane2.append(a[i]) elif len(lane3) < 5:

lane3.append(a[i])

elif len(lane4) < 5:

lane1.append(a[i]) elif len(lane5) < 5:

lane5.append(a[i])

else:

lane6.append(a[i]) i += 1

print(lane1,lane2,lane3,lane4,lane5,lane6)

### I run this code completely fine . Also I face many problem to while generating customer continuously to insert them lanes. :

a = [12,23,11,6,7,12]

lane1 = [] lane2 = [] lane3 = [] lane4 = [] lane5 = [] lane6 = [] waiting = []

tmp = []

lnth = 40 - (lane1 + lane2 + lane3 + lane4 + lane5 + lane6) if len(lane7) > 0:

print(lane7, "are lane 7 member .Total has ", len(lane7)) k = 0

i = 0

print("number neede to full the all line: ", lnth, "again count is:", count, "connecting all separetly: ", (l1 + l2 + l3 + l4 + l5 + l6), " different approch: ",

l\_curr[0] + l\_curr[1] + l\_curr[2] + l\_curr[3] + l\_curr[4] + l\_curr[5]) regu\_line = lane1 + lane2 + lane3 + lane4 + lane5

while lnth != k: tmp.append(self.lane7[i]) k += 1

i += 1

print("the tmp array: ", tmp)

if len(lane7) > 0:

a = copy.copy(tmp) for p in range(lnth): self.lane7.pop(p)

print("after pooping from last array: ", self.lane7)

if tmp:

a = copy.tmp

i = 0

while len(a) != i:

if a[i] > 9 or len(lane6) > 15: if len(lane1) < 5:

lane1.append(a[i]) elif len(lane2) < 5:

lane2.append(a[i]) elif len(lane3) < 5:

lane3.append(a[i])

elif len(lane4) < 5: lane1.append(a[i])

elif len(lane5) < 5: lane5.append(a[i])

else:

lane6.append(a[i])

i += 1

print(lane1,lane2,lane3,lane4,lane5,lane6)

### This attempt almost failed. But after several hours of testing and bug fixing, I ended up doing these:

# declared lanes regular\_lane1 = [] regular\_lane2 = [] regular\_lane3 = [] regular\_lane4 = [] regular\_lane5 = [] self\_lane = [] waiting\_lane = []

class Insertion:

# function for inserting customers in different lanes # receive data through a list called a

def insertCustomer(self, a): from f2 import Lottery

# This function will be called after to print lane situation lane\_inst = Lane\_situation()

lane\_inst.print\_stat()

# total customers are waiting present\_lane\_length =

len(regular\_lane1)+len(regular\_lane2)+len(regular\_lane3)+len(regular\_lane4)+len(regular\_lane5)+len(self\_lane

)

# execute when the regular lane has no space but the self-lane has # sort out the incoming, newly generated customer

if (len(regular\_lane1)+len(regular\_lane2)+len(regular\_lane3)+len(regular\_lane4)+len(regular\_lane5)) == 25 and len(self\_lane) < 15:

print("Reguler lane over flow!")

# Remember the indexes that need to be deleted. remove\_ele = []

# Customers with more than nine products move to the waiting lane. for pos in range(len(a)):

if a[pos] > 9:

print("Customer with ",a[pos]," products has to join in waiting\_lane line") waiting\_lane.append(a[pos])

remove\_ele.append(pos)

# delete them from list a by using remove\_ele indexes. for pos in reversed(remove\_ele):

if 0 <= pos < len(a): a.pop(pos)

# If the current total customers, along with lists, exceed exeed limit of 40 if (present\_lane\_length + len(a) ) > 40:

k = 0

while len(a) != k:

waiting\_lane.append(a[k]) # appending all customers in the waiting lane k += 1

# return if there is no space left in any lanes if present\_lane\_length == 40:

print("Both regular and selfservice lanes are full. Customer have to wait.") print("First in waiting\_lane will enter to lane first")

return

tmp = [] # temporary array to hold customers from the waiting lane customer\_index = [] # Remember their indexes from the waiting lane.

lnth = 40 - present\_lane\_length # how many spaces left in regular and service lane

# If the current total customers, along with lists, exceed exeed limit of 40 if len(waiting\_lane) > 0 and lnth > 0:

tmp\_reg\_line = len(regular\_lane1) + len(regular\_lane2) + len(regular\_lane3) + len(regular\_lane4) + len(regular\_lane5) #total customers in regular lane

tmp\_slf\_line = len(self\_lane) # total customers in self services till

k = 0

# check space availability in lanes and customers are waiting while k != len(waiting\_lane) and lnth != 0:

if tmp\_reg\_line < 25 and tmp\_slf\_line < 15: #if both regular and self service lane has space left tmp.append(waiting\_lane[k])

if waiting\_lane[k] > 9: tmp\_reg\_line += 1

else:

tmp\_slf\_line += 1

customer\_index.append(k) # remember indexes for appended customer from waiting lnth -= 1

k += 1

elif tmp\_reg\_line == 25 and tmp\_slf\_line < 15: # regular space full, self service left if waiting\_lane[k] > 9:

k += 1

else:

tmp.append(waiting\_lane[k]) customer\_index.append(k) tmp\_slf\_line += 1

lnth -= 1

k += 1

elif tmp\_reg\_line < 25 and tmp\_slf\_line == 15: # regular space left, self service full tmp.append(waiting\_lane[k])

customer\_index.append(k) tmp\_reg\_line += 1

lnth -= 1

k += 1

# delete the customer from the waiting lane using the customer index list. if customer\_index:

for pos in reversed(customer\_index): if 0 <= pos < len(waiting\_lane):

waiting\_lane.pop(pos)

# copy the temporary customers to list a if tmp:

a = copy.copy(tmp)

lottery\_ins = Lottery()

lottery\_ins.lottery\_generator(a) # Check the letter by calling the function.

# Push customers with products to different lanes. i = 0

while i != len(a):

if len(self\_lane) < 15 and a[i] < 10: # total product in bucked less than 9 and has space left in self self\_lane.append(a[i])

else: # or we have spaces left in others lanes. if len(regular\_lane1) < 5:

regular\_lane1.append(a[i]) elif len(regular\_lane2) < 5:

regular\_lane2.append(a[i]) elif len(regular\_lane3) < 5:

regular\_lane3.append(a[i])

elif len(regular\_lane4) < 5: regular\_lane4.append(a[i])

elif len(regular\_lane5) < 5: regular\_lane5.append(a[i])

i += 1

### # class for checkout customers product in buscket from different lanes. This part is running independently using a thread.

**I have added a comment for the latest updated code.**

**At this stage, I tried to implement my lane-clearing mechanism. Which required a thread. So I implemented threading with Python and declared some non-empty lists to verify th**at my threading was working fine. And implemented like this:

import threading import time is\_done = True

lane1 = [12,12,23,4,22]

lane2 = [12,12,23,4,22]

lane3 = [12,12,23,4,22]

lane4 = [12,12,23,4,22]

lane5 = [12,12,23,4,22]

lane6 = [12,12,23,4,22]

lane7 = [12,12,23,4,22]

def clearing\_lane1(): global l1

while is\_done:

if len(lane1) > 0: frst\_ele = lane1[0]

lane1.pop(0) time.sleep(4)

else:

continue

def clearing\_lane2(self): global l2

while is\_done:

if len(lane2) > 0: frst\_ele = lane2[0] lane2.pop(0) time.sleep(4)

else:

continue

def clearing\_lane3(): global l3

while is\_done:

if len(lane3) > 0: frst\_ele = lane3[0] lane3.pop(0) time.sleep(4)

else:

continue

def clearing\_lane4(): global l4

while is\_done:

if len(lane4) > 0: frst\_ele = lane4[0] lane4.pop(0) time.sleep(4)

else:

continue

def clearing\_lane5(): global l5

while is\_done:

if len(lane5) > 0: frst\_ele = lane5[0] lane5.pop(0) time.sleep(4)

else:

continue

def clearing\_lane6(): global l6

while is\_done:

if len(lane6) > 0: frst\_ele = lane6[0] lane6.pop(0) time.sleep(4)

else:

continue

clearing\_thread1 = threading.Thread(target=clearing\_lane1) clearing\_thread2 = threading.Thread(target=clearing\_lane2) clearing\_thread3 = threading.Thread(target=clearing\_lane3) clearing\_thread4 = threading.Thread(target=clearing\_lane4) clearing\_thread5 = threading.Thread(target=clearing\_lane5) clearing\_thread6 = threading.Thread(target=clearing\_lane6)

clearing\_thread1.start() clearing\_thread2.start() clearing\_thread3.start() clearing\_thread4.start()

clearing\_thread5.start() clearing\_thread6.start()

t = 0

while is\_done: if t > 5:

print(lane1,lane2,lane3,lane4,lane5,lane6) if t > 720:

is\_done = False time.sleep(5)

t += 10

### Later, when updating my code, I implemented using classes and functions. In the f1.py file, I added a class called Lane\_services. Under this, all the lane clearing processes were initiated.

# customer checkout in different lanes also multithreaded class Lane\_services:

# delete customer from lane following criteria if customer exists def clearing\_regular\_lane1(self):

while sharedState.is\_done:

if len(regular\_lane1) > 0: time.sleep(regular\_lane1[0]\*4) regular\_lane1.pop(0)

else:

continue

# delete customer from lane following criteria if customer exists def clearing\_regular\_lane2(self):

while sharedState.is\_done: if len(regular\_lane2) > 0:

time.sleep(regular\_lane2[0]\*4) regular\_lane2.pop(0)

else:

continue

# delete customer from lane following criteria if customer exists def clearing\_regular\_lane3(self):

while sharedState.is\_done:

if len(regular\_lane3) > 0: time.sleep(regular\_lane3[0]\*4) regular\_lane3.pop(0)

else:

continue

# delete customer from lane following criteria if customer exists def clearing\_regular\_lane4(self):

while sharedState.is\_done:

if len(regular\_lane4) > 0: time.sleep(regular\_lane4[0]\*4) regular\_lane4.pop(0)

else:

continue

# delete customer from lane following criteria if customer exists def clearing\_regular\_lane5(self):

while sharedState.is\_done:

if len(regular\_lane5) > 0: time.sleep(regular\_lane5[0]\*4) regular\_lane5.pop(0)

else:

continue

# delete customer from lane following criteria if customer exists def clearing\_self\_lane(self):

while sharedState.is\_done: if len(self\_lane) > 0:

time.sleep(self\_lane[0]\*6) self\_lane.pop(0)

else:

continue

### For printing the code, I initialized the code as: It was my first attempt.

import time is\_done = True

lane1 = [12,12,23,4,22]

lane2 = [12,12,23,4,22]

lane3 = [12,12,23,4,22]

lane4 = [12,12,23,4,22]

lane5 = [12,12,23,4,22]

lane6 = [12,12,23,4,22]

lane7 = [12,12,23,4,22]

print(f"L{1}(Reg) ->", end=" ") for \_ in lane1:

print("\*",end=" ")

print("")

print(f"L{2}(Reg) ->", end=" ") for \_ in lane2:

print("\*",end=" ")

print("")

print(f"L{3}(Reg) ->", end=" ") for \_ in lane3:

print("\*",end=" ")

print("")

print(f"L{4}(Reg) ->", end=" ") for \_ in lane4:

print("\*",end=" ")

print("")

print(f"L{5}(Reg) ->", end=" ") for \_ in lane5:

print("\*",end=" ") print("")

print(f"L{6}(self) ->", end=" ") for \_ in lane6:

print("\*",end=" ")

### After testing, implementing, and a couple of bug fixes, it turned out: Added class and function methods:

# class holds print\_stat function class Lane\_situation:

# print current situation of lanes after 30 sec continuously

def print\_stat(self):

current\_datetime = datetime.now()

print("Total number of customers waiting\_lane to check out at", current\_datetime, " is: ", len(regular\_lane1)+len(regular\_lane2)+len(regular\_lane3)+len(regular\_lane4)+len(regular\_lane5)+len( self\_lane))

lanes = [regular\_lane1, regular\_lane2, regular\_lane3, regular\_lane4, regular\_lane5, self\_lane, waiting\_lane] # creating a list containing lists for better iteration process

# idea generated from chat gpt

for i, lane in enumerate(lanes, start=1): length = len(lane)

if length < 1: if i == 7:

print("No customer in waiting\_lane") elif i == 6:

print(f"L{i}(Self) ->", end=" ") print("Open", end=" ")

else:

if i != 1:

print(f"L{i}(Reg) ->", end=" ") print("Close", end=" ")

else:

print(f"L{i}(Reg) ->", end=" ") print("Open", end=" ")

else:

if i == 7:

print(length,"customers are waiting\_lane to join in line") else:

if i == 6:

print(f"L{i}(Self) ->", end=" ") else:

print(f"L{i}(Reg) ->", end=" ")

for \_ in lane: print("\*", end=" ")

print("")

### I finally used the thread like this (in my f3.py file):

import threading import time

import sharedState import f1

from f2 import CreateCustomerElement

from f1 import Lane\_services , Lane\_situation

customer\_ins = CreateCustomerElement() service\_ins = Lane\_services() situation\_ins = Lane\_situation()

# targeting functions to start thread

# idea of using thread was taken from chat gpt

# Each thread below is responsible for clearing lane.

service\_thread1 = threading.Thread(target=service\_ins.clearing\_regular\_lane1) service\_thread2 = threading.Thread(target=service\_ins.clearing\_regular\_lane2)

service\_thread3 = threading.Thread(target=service\_ins.clearing\_regular\_lane3) service\_thread4 = threading.Thread(target=service\_ins.clearing\_regular\_lane4) service\_thread5 = threading.Thread(target=service\_ins.clearing\_regular\_lane5) service\_thread6 = threading.Thread(target=service\_ins.clearing\_self\_lane) generateThread = threading.Thread(target=customer\_ins.generate\_customer)

# thread starting generateThread.start() service\_thread1.start() service\_thread2.start() service\_thread3.start() service\_thread4.start() service\_thread5.start() service\_thread6.start()

t = 0

while sharedState.is\_done: if t > 300:

sharedState.is\_done = False time.sleep(30)

t += 30

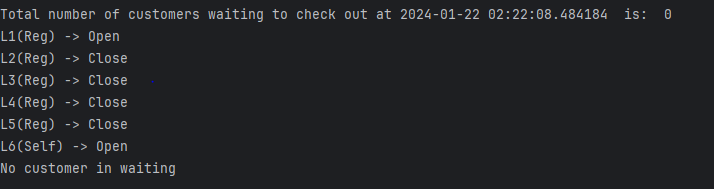
# waiting threads to finish service\_thread1.join() service\_thread2.join() service\_thread3.join() service\_thread4.join() service\_thread5.join() service\_thread6.join() generateThread.join()

# 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.*

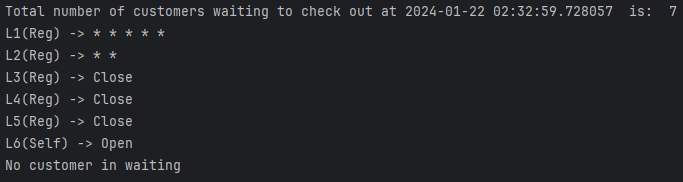
## Feature F1

* + 1. SUB-FEATURE I- SCREENSHOTS …

Initially when all the lanes ware empty nothing to show in lane. But two lane showing open. And they allow customer to enter.

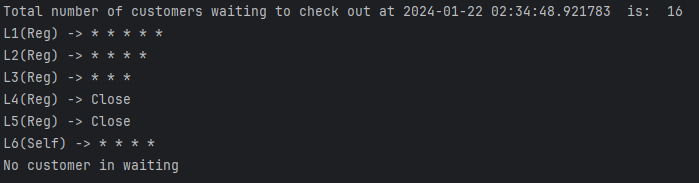
* + 1. SUB-FEATURE II- SCREENSHOTS …

Initially generated customer were inserted to the lane . But all the customer has products more then 9. So they were forced to join in new regular lane:

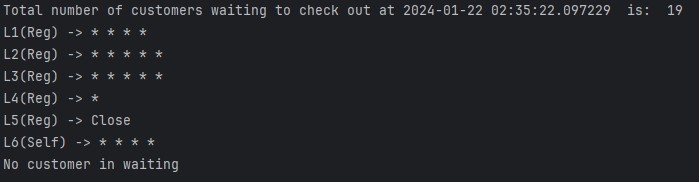


## Sub-feature iii- screenshots …

I have added two screen shot to demonstrate the change of lanes customers.

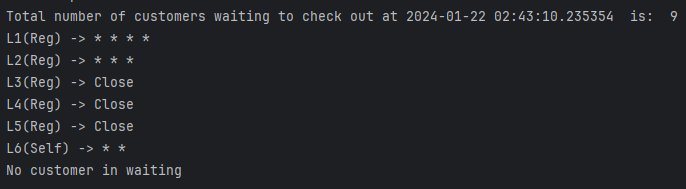
Initially:

After 30 second the lanes look like this:

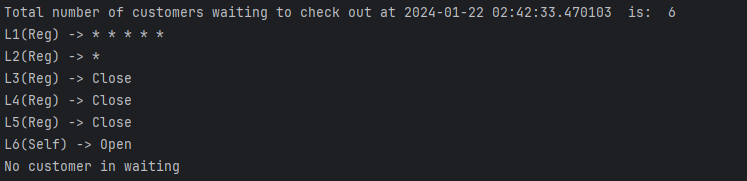


* + 1. SUB-FEATURE IV- SCREENSHOTS …

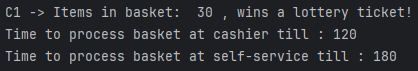
Lane that has no Customer showing close . When a lane is full customer join to the next lane



* + 1. SUB-FEATURE V- SCREENSHOTS …



## Feature F2

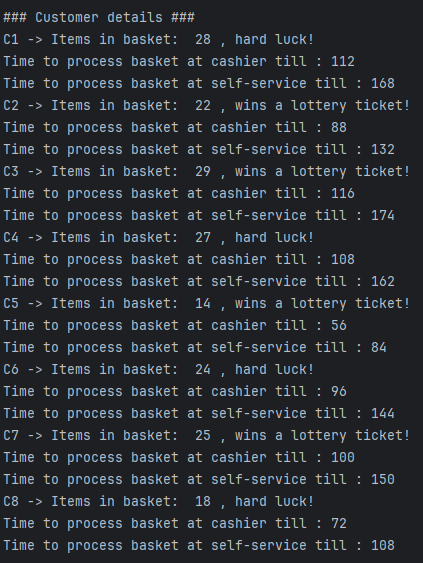
* + 1. SUB-FEATURE I- SCREENSHOTS …
    2. SUB-FEATURE II- SCREENSHOTS …

Here it’s showing customer’s basket products :

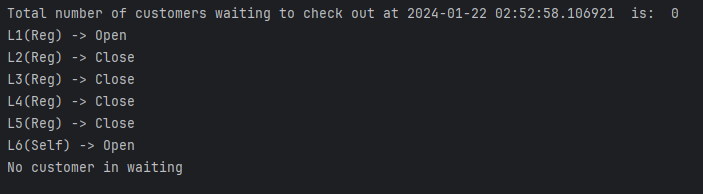


* + 1. SUB-FEATURE III- SCREENSHOTS …

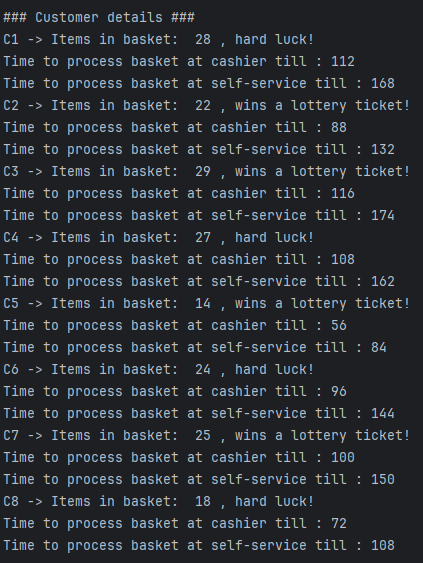


* + 1. SUB-FEATURE IV- SCREENSHOTS …
    2. SUB-FEATURE V- SCREENSHOTS …

## Feature F3

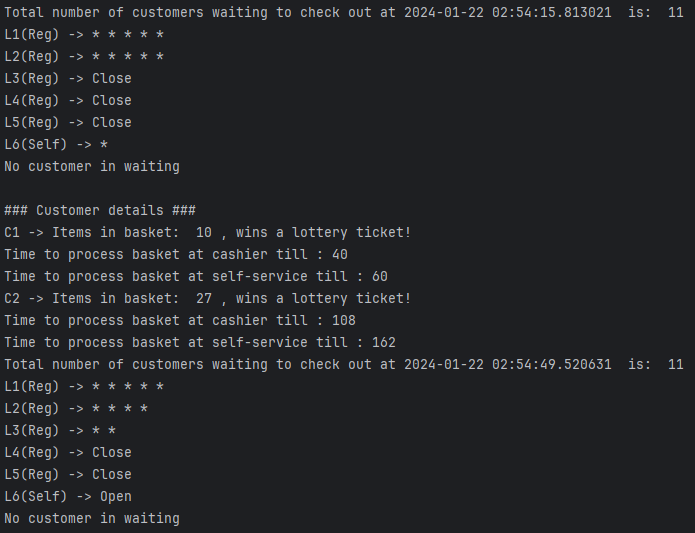
* + 1. SUB-FEATURE I- SCREENSHOTS …
    2. SUB-FEATURE II- SCREENSHOTS …

Just after the simulation started, a random amount of customer was generated. And shown in console.

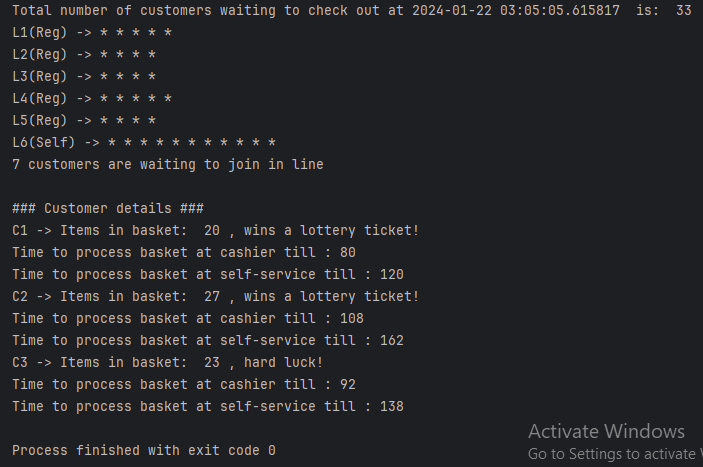


* + 1. SUB-FEATURE III- SCREENSHOTS …

Simulation printing lanes situation after 30 seconds:



* + 1. SUB-FEATURE IV- SCREENSHOTS …



# 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.*

***In f1.py file initially I was printing the lanes situation like this:***

print(f"L{1}(Reg) ->", end=" ") for \_ in lane1:

print("\*",end=" ")

print("")

print(f"L{2}(Reg) ->", end=" ") for \_ in lane2:

print("\*",end=" ")

print("")

print(f"L{3}(Reg) ->", end=" ") for \_ in lane3:

print("\*",end=" ")

print("")

print(f"L{4}(Reg) ->", end=" ") for \_ in lane4:

print("\*",end=" ")

print("")

print(f"L{5}(Reg) ->", end=" ") for \_ in lane5:

print("\*",end=" ") print("")

print(f"L{6}(self) ->", end=" ") for \_ in lane6:

print("\*",end=" ")

### Clearly, the code is not looking great. After a couple of tries with OpenAI, I was able to find an efficient way to do that.

**The code is:**

# class holds print\_stat function class Lane\_situation:

# print current situation of lanes after 30 sec continuously def print\_stat(self):

current\_datetime = datetime.now()

print("Total number of customers waiting\_lane to check out at", current\_datetime, " is: ", len(regular\_lane1)+len(regular\_lane2)+len(regular\_lane3)+len(regular\_lane4)+len(regular\_lane5)+len( self\_lane))

lanes = [regular\_lane1, regular\_lane2, regular\_lane3, regular\_lane4, regular\_lane5, self\_lane, waiting\_lane] # creating a list containing lists for better iteration process

# idea generated from chat gpt

for i, lane in enumerate(lanes, start=1):

length = len(lane) if length < 1:

if i == 7:

print("No customer in waiting\_lane") elif i == 6:

print(f"L{i}(Self) ->", end=" ") print("Open", end=" ")

else:

if i != 1:

print(f"L{i}(Reg) ->", end=" ") print("Close", end=" ")

else:

print(f"L{i}(Reg) ->", end=" ") print("Open", end=" ")

else:

if i == 7:

print(length,"customers are waiting\_lane to join in line") else:

if i == 6:

print(f"L{i}(Self) ->", end=" ") else:

print(f"L{i}(Reg) ->", end=" ")

for \_ in lane: print("\*", end=" ")

print("")

It is clearly efficient. Much more detailed data with various information showed

#### In f3.py function I was struggling to maintain thread and calling functions from other files: My initial code was :

sys = System\_management()

generate\_thread = threading.Thread(target=sys.generate\_customer) clearing\_thread1 = threading.Thread(target=sys.clearing\_lane1) clearing\_thread2 = threading.Thread(target=sys.clearing\_lane2) clearing\_thread3 = threading.Thread(target=sys.clearing\_lane3) clearing\_thread4 = threading.Thread(target=sys.clearing\_lane4) clearing\_thread5 = threading.Thread(target=sys.clearing\_lane5) clearing\_thread6 = threading.Thread(target=sys.clearing\_lane6)

generate\_thread.start() clearing\_thread1.start() clearing\_thread2.start() clearing\_thread3.start() clearing\_thread4.start() clearing\_thread5.start() clearing\_thread6.start()

### OpenAI updated the code after a couple of tries, and that’s turned out fine. Here is the updated code.

import threading import time

import sharedState import f1

from f2 import CreateCustomerElement

from f1 import Lane\_services , Lane\_situation

customer\_ins = CreateCustomerElement() service\_ins = Lane\_services() situation\_ins = Lane\_situation()

# targeting functions to start thread

# idea of using thread was taken from chat gpt

# Each thread below is responsible for clearing lane.

service\_thread1 = threading.Thread(target=service\_ins.clearing\_regular\_lane1) service\_thread2 = threading.Thread(target=service\_ins.clearing\_regular\_lane2) service\_thread3 = threading.Thread(target=service\_ins.clearing\_regular\_lane3) service\_thread4 = threading.Thread(target=service\_ins.clearing\_regular\_lane4) service\_thread5 = threading.Thread(target=service\_ins.clearing\_regular\_lane5) service\_thread6 = threading.Thread(target=service\_ins.clearing\_self\_lane) generateThread = threading.Thread(target=customer\_ins.generate\_customer)

# thread starting generateThread.start() service\_thread1.start() service\_thread2.start() service\_thread3.start() service\_thread4.start() service\_thread5.start() service\_thread6.start()

t = 0

while sharedState.is\_done: if t > 300:

sharedState.is\_done = False time.sleep(30)

t += 30

# waiting threads to finish service\_thread1.join() service\_thread2.join() service\_thread3.join() service\_thread4.join() service\_thread5.join() service\_thread6.join() generateThread.join()

#### OpenAI generated code

def clearing\_lane(self):

while len(self.lane1) > 0:

frst\_ele = self.lane1[0] time.sleep(frst\_ele) self.lane1.pop(0) global count

count -= 1

while len(self.lane2) > 0: frst\_ele = self.lane2[0] time.sleep(frst\_ele) self.lane2.pop(0) count -= 1

while len(self.lane3) > 0: frst\_ele = self.lane3[0] time.sleep(frst\_ele) self.lane3.pop(0) count -= 1

while len(self.lane4) > 0: frst\_ele = self.lane4[0] time.sleep(frst\_ele) self.lane4.pop(0) count -= 1

while len(self.lane5) > 0: frst\_ele = self.lane5[0]

time.sleep(frst\_ele) self.lane5.pop(0) count -= 1

while len(self.lane6) > 0: frst\_ele = self.lane6[0] time.sleep(frst\_ele) self.lane6.pop(0)

count -= 1

**After few modification**

# customer checkout in different lanes also multithreaded class Lane\_services:

# delete customer from lane following criteria if customer exists def clearing\_regular\_lane1(self):

while sharedState.is\_done: if len(regular\_lane1) > 0:

time.sleep(regular\_lane1[0]\*4) regular\_lane1.pop(0)

else:

continue

# delete customer from lane following criteria if customer exists def clearing\_regular\_lane2(self):

while sharedState.is\_done:

if len(regular\_lane2) > 0: time.sleep(regular\_lane2[0]\*4) regular\_lane2.pop(0)

else:

continue

# delete customer from lane following criteria if customer exists def clearing\_regular\_lane3(self):

while sharedState.is\_done: if len(regular\_lane3) > 0:

time.sleep(regular\_lane3[0]\*4) regular\_lane3.pop(0)

else:

continue

# delete customer from lane following criteria if customer exists def clearing\_regular\_lane4(self):

while sharedState.is\_done: if len(regular\_lane4) > 0:

time.sleep(regular\_lane4[0]\*4) regular\_lane4.pop(0)

else:

continue

# delete customer from lane following criteria if customer exists

def clearing\_regular\_lane5(self): while sharedState.is\_done:

if len(regular\_lane5) > 0: time.sleep(regular\_lane5[0]\*4) regular\_lane5.pop(0)

else:

continue

# delete customer from lane following criteria if customer exists def clearing\_self\_lane(self):

while sharedState.is\_done: if len(self\_lane) > 0:

time.sleep(self\_lane[0]\*6) self\_lane.pop(0)

else:

continue

# 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)

1. 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: 28 out of 36**

1. 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: 14 out of 24**

1. 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: 6 out of 10**

1. 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: 14 out of 20**

1. 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: 68 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.*

Here are all the added file listed below. f1.py

f2.py f3.py

sharedState.py

Reff:

Stack Overflow (2022). *Stack Overflow - Where Developers Learn, Share, & Build Careers*. [online] Stack Overflow. Available at: https://stackoverflow.com/.

OpenAI (2023). *ChatGPT*. [online] chat.openai.com. Available at: https://chat.openai.com/.