# COMP1811 – Python Project Report

| Name: | Harmanpreet Singh | Student ID | 001289568 |
|---|---|---|---|
| Partner's name: | Sebastian Carp | Student ID | 001289569 |

**Sebastian Carp – F1**

**Harmanpreet Singh - F2**

## 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**<br>**F1:** i ☒ ii ☒ iii ☒ iv ☒ v ☒ vi ☒<br>**F2:** i ☒ ii ☒ iii ☒ iv ☒ v ☒ vi ☒<br>**F3:** i ☒ ii ☒ iii ☒ iv ☒ v ☒ |
| 1.2 Circle the parts of the coursework you have **partly completed or are partly working.** | **Features**<br>**F1:** i ☐ ii ☐ iii ☐ iv ☐ v ☐ vi ☐<br>**F2:** i ☐ ii ☐ iii ☐ iv ☐ v ☐ vi ☐<br>**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.*

- F3: "threading.Lock()" has been used to handle the printing of lanes and customers in them each time a customer checks out. This keeps everything in order and prevents overlapping when two customers check out at the same time (either from different lanes or because they had the same number of items in the self-service lane), however in a scenario when a customer completes checkout and new customers are added at the same time, the printing will overlap. With "threading.Lock()" there might also be small delays before the next batch of customers is added every 30 seconds (i.e. 1 to 5 seconds)

## 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.*

## 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: I have created the skeleton for the customers from F2 to go onto. For this to be achievable, the lanes have been defined as variables of subclasses (RegLane and SelfService) inheriting the attributes from the parent class "Lane". Many functions of managing the lanes have been created within the regular_lane.py file, as it was the easiest one to work around without my partner's code being complete and only the determined data types. Some functions take arguments depending on their functionality, but most do not. Both me and my partner have agreed to use the total number of customers in a lane to display the customers rather than listing each customer, for better code efficiency and visibility when running the simulation.

F2: Randcustomer is responsible to create the customer data, the class encapsulates methods for generating random items, checkout time for self-checkout and regular lane.

The data is represented using dictionaries, which contains attributes such as customer, number of items, lottery status, this allows the data to be easily accessible I was able to assign each customer a random number within the same class by using the for loop and random to make the number creation for the items truly random for each customer.

F3: We chose to use one file for most of F3 to easily manoeuvre the existing skeleton of the code (the lanes) with the newly added instances of "Customer". The chosen data types (int, str, bool) for each of the partners part of the code (F1 and F2) were kept in the implementation within F3 and were used accordingly. Both parts of the code were merged into a couple of functions which were called at the end in the "simulation( )" function. The code logic is as follows: there is a for loop with 8 iterations which creates a random number of customers between 1 to 10 each time, effectively making it impossible to get past 80 customers, therefore following the coursework specification of not having more than 40 customers waiting to join any line. The customers are then stored in memory as variables and added to their lanes. Threading was used to run the checkout for customers for both type of lanes at the same time, following different functionalities for each. Python is an interpreted language, which means that order matters.

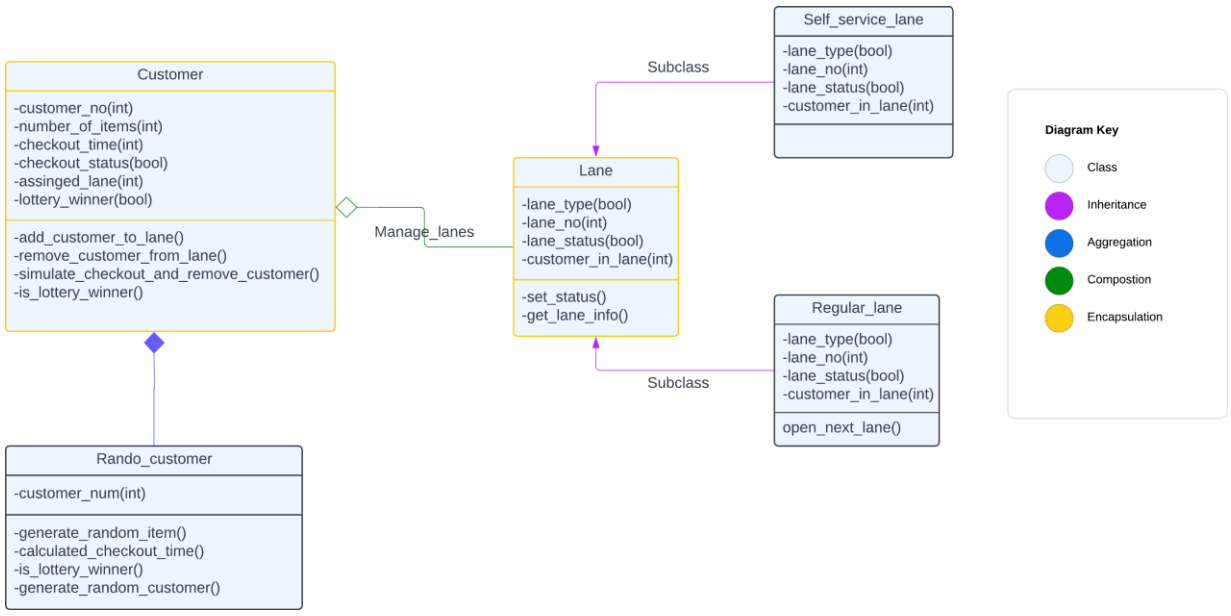
# 4. Classes and OOP Features

*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

- Lane
- RegLane(Lane)
- SelfService(Lane)
- RandCustomer
- Customer




## 4.2 Brief Explanation of Class Design and OOP Features Used

F1: Classes Lane, RegLane, SelfService and Customer all contain features of F1. For example, inheritance has been used for RegLane and SelfService, with the parent class being Lane. This means that the two classes are now modelled by the attributes of their parent class: lane_type (str), lane_no (int), lane_status (str), customers_in_lane (int). The same principle is followed for the Customer class, which takes customer_number (int), number_of_items (int), checkout_time (int), checkout_status (str) as attributes. By defining these classes, I am effectively creating

robust and reusable code for any future implementations. I have implemented this to create objects such as the lanes (subfeature i.) or randomly generated customers with the help of F2 from those classes / subclasses as variables, which are essential for the structure of the code. With the customers being defined, I used a method within the RegLane subclass which runs a series of checks and then assigns the customer to the lane he is supposed to be in (subfeature ii.). The customers are mapped at the same time with their assigned lane, to be able to remove them within another method from the same class (subfeature iii.). For opening / closing a lane, I have created a method withing the RegLane class which uses a loop to check if the next lane needs to be opened if conditions are met (subfeature iv). Displaying all the lanes status happens in an abstract function, which calls the needed methods inside the classes with a for loop. (subfeature v) These lanes are static, only some of their attributes will change depending on multiple factors within the instances created from the Customer class, specifically a customer's basket size, their number and their calculated checkout time based on what lane they are on and their checkout status, which uses a real time elapsing system since the start of the simulation.

F2: Randcustomer class encapsulates the behavior related to generating random customers, details shown inside the code such as random item generation, checkout time for both regular lane and self-checkout lane is hidden within the class, therefore allows the user to use the method without any need to understand each part of the code. The use of static methods like generate_random_items, is_lottery_winner is used to perform specific tasks related to customer behavior. These methods can be called without creating any instance for the class. Customer information is represented using a dictionary containing various attributes such as customer ID, number of items, lottery status, and checkout times. This allows for a structured and easily accessible representation of customer data. The code uses random to generate random amounts of customer data (customer_no, number_of_items), this adds variability in the number of items, as well as the chance of winning lottery, by enclosing the information presentation within the class methods, the code additionally exemplifies encapsulation. The last loop summarizes the specifics of how the customer information is shown by iterating through the created customers and printing their details.

F3: Subfeature i of F3 has been implemented within F1, when the lane variables have been defined. Subfeature ii is initiated within F2 and used inside the simulation() function inside the loop to create new customers each 30 seconds. Subfeature iii uses the objects as args, which are now stored in lists, and their encapsulated methods are accessed with the use of threads, to allow for simultaneous running and updating of lanes each time a customer checks out, therefore proving the methods work together.

# 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 … Lane – Sebastian Carp F1

```
class Lane:
    def __init__(self, lane_type, lane_no, lane_status, customers_in_lane):  # arguments to be
inherited by subclasses
        self.lane_type = lane_type
        self.lane_no = lane_no
        self.lane_status = lane_status
        self.customers_in_lane = customers_in_lane

    def set_status(self, lane_status):  # update status to open / closed
        self.lane_status = lane_status

    def get_lane_info(self):  # returns the lane info for a specific lane
        info = f'Lane Number ({self.lane_type}): {self.lane_no},' ' ' \
               f'Status: {self.lane_status},' ' ' \
               f'Customers in lane: {self.customers_in_lane}'
        return info
```

## 5.2 Class … RegLane(Lane) - Sebastian Carp F1

```
class RegLane(Lane):
    """
    Subclass of "Lane" for the regular lanes, inheriting its attributes
    Contains method for checking if the next lane needs to be opened or not
    """
    def __init__(self, lane_type, lane_no, lane_status, customers_in_lane):
        super().__init__(lane_type, lane_no, lane_status, customers_in_lane)  # inheriting
from parent class

    # checking if the next lane needs to be opened or not
```

```
    def open_next_lane(self):
        for lane in list_of_regular_lanes:
            if self.customers_in_lane >= 5 and lane.lane_status == 'closed' and
lane.customers_in_lane > 0:  # check
                # if conditions met
                lane.set_status('open')  # opens next lane if conditions met
                break  # exit the loop once a lane is opened to open only the next one
```

## 5.3 Class ...SelfService(Lane) - Sebastian Carp F1

```
class SelfService(Lane):
    """
    Subclass of "Lane" for the self-service lane, inheriting its attributes
    """
    def __init__(self, lane_type, lane_no, lane_status, customers_in_lane):
        super().__init__(lane_type, lane_no, lane_status, customers_in_lane)
```

## 5.4 Class ...Customer - Sebastian Carp & Harmanpreet Singh

```
class Customer:
    """
    Class to which all the customers will be modelled by
    """
    def __init__(self, customer_number, number_of_items, checkout_time, checkout_status):
        self.customer_number = customer_number
        self.number_of_items = number_of_items
        self.checkout_time = checkout_time
        self.checkout_status = checkout_status
        self.assigned_lane = None  # Initialize initial assigned_lane attribute
        self.lottery_winner = False  # Initialize lottery_winner attribute

    def add_customer_to_lane(self):
        """
        Adds customers to lanes based on basket size, lane criteria and lane capacity
        :return:
        """
        if self.number_of_items < 10:  # If a customer has less than 10 items, he can go to
self-checkout
            if lane6.customers_in_lane >= 15:  # If the self-checkout lane is full, customers
will go to a regular lane
                for lane in list_of_regular_lanes:  # Checking which regular lane is best to
join
                    if lane.customers_in_lane < 5:
                        lane.customers_in_lane += 1
                        self.assigned_lane = lane  # Assigns customer to regular lane
                        break
                else:
                    min_lane = min(list_of_regular_lanes, key=lambda x: x.customers_in_lane)
                    min_lane.customers_in_lane += 1
                    self.assigned_lane = min_lane  # Assigns customer to next less crowded
lane
            else:
                lane6.customers_in_lane += 1  # Adding customer to self-checkout lane
                self.assigned_lane = lane6
        elif self.number_of_items >= 10:  # If a customer has more than 10 items, he needs to
go to a regular lane
            for lane in list_of_regular_lanes:  # Checking which regular lane is best to join
                if lane.customers_in_lane < 5:
                    lane.customers_in_lane += 1
                    self.assigned_lane = lane  # Assigns customer to lane
                    break
            else:
                min_lane = min(list_of_regular_lanes, key=lambda x: x.customers_in_lane)
                min_lane.customers_in_lane += 1
                self.assigned_lane = min_lane  # Assigns customer to next less crowded lane

    def remove_customer_from_lane(self):
        """
        Removes a customer from their lane based on basket size
        :return:
        """
        if self.checkout_status == 'complete':
            if self.number_of_items < 10:
```

```
                lane6.customers_in_lane -= 1
            if self.number_of_items > 10:
                if self.assigned_lane:
                    self.assigned_lane.customers_in_lane -= 1



def simulate_checkout_and_remove_customer(self):
        """
        Using real elapsed time since the start of the simulation to set a customers checkout
status to complete
        and remove them from their lane
        :return:
        """
        start_time = time.time()  # Record the start time of the simulation

        while time.time() - start_time < self.checkout_time:
            # Keep checking if the checkout time has elapsed
            pass

        self.checkout_status = 'complete'  # Update the checkout status to 'complete'

        # Check if the customer wins the lottery
        self.lottery_winner = self.checkout_status == 'complete' and self.is_lottery_winner()
        if self.lottery_winner:  # If the customer meets the conditions for winning the
lottery
            print(f'Customer {self.customer_number} has won the lottery')  # Prints the
customer number which has won

        self.remove_customer_from_lane()  # Removes the customer from the lane

        # Printing the lanes info with a lock to ensure synchronized printing
        with print_lock:
            print_lanes_info()

    def is_lottery_winner(self):
        """
        Determine if the customer wins the lottery
        :return:
        """
        return self.number_of_items >= 10 and random.randint(1, 10) == 1
```

## 5.5 Class ...RandCustomer - Harmanpreet Singh F2

```
class RandCustomer:
    def __init__(self, customer_num):
        """
        Initializes a Customer object with a given customer number.
        """
        self.customer_num = customer_num

    @staticmethod
    def generate_random_items():
        """
        Generates a random number of items between 1 and 30 for each customer.
        """
        return random.randint(1, 30)

    @staticmethod
    def calculate_checkout_times(num_items):
        """
        Calculates the time it takes to check out each customer's items in the regular lane
and self-checkout.
        """
        regular_lane_time = (num_items * 4) #checkout_time for regular_lane
        self_checkout_time = (num_items * 6) #for self_checkout, calculate the time
        return round(regular_lane_time, 2), round(self_checkout_time, 2)

    @staticmethod
    def is_lottery_winner(num_items):
        """
        Determine if the customer wins the lottery
        :return:
        """
        return num_items >= 10 and random.randint(1, 10) == 1# creating 10% chances of anyone
with more than 10 items wins lottery

    def generate_random_customers(self):
        """
        Generates a list of randomly generated customers.
        """
        customers = []
        for customer_number in range(1, self.customer_num + 1):
            number_of_items = self.generate_random_items()
            regular_lane_time, self_checkout_time =
self.calculate_checkout_times(number_of_items)
            checkout_status = 'incomplete'

            won_lottery = self.is_lottery_winner(number_of_items)

            customer_info = {
                'customer_id': customer_number,
                'num_items': number_of_items,
                'won_lottery': won_lottery,
                'self_checkout_time': self_checkout_time,
                'regular_lane_time': regular_lane_time,
                'checkout_status': checkout_status
```

```
            }

            customers.append(customer_info)

    return customers
```

# 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.*

Test Plan for Python Code

1 Purpose

The purpose of this testing is to make sur that every bit of the code that we have presented over the course of 2 months is working as it should or are there any issues that would create future problems.

1.1 Scope

Throughout this testing we will be testing each part of our codes (F1, F2, F3)

Testing each individual code (unit testing) helps us understand that all the pieces are working plus with a test for the overall (integration testing) working code to make sure those pieces are working together to give us the result

1.2 Objectives

- Validate the correctness of the code.

- Ensure the code meets functional and non-functional requirements.

- Identify and fix defects.

2. Classes: lane, reglane, self-service, randcustomer, customer

2.1 Functions: Display_lane_saturation, get_total_customers_in_all_lanes, timestamps, print_lane_info, simulate_checkout_for_lane, generate_random_customer, get_customer_in_lane, simulation

3. Test Approach

3.1 Testing Levels

- Unit Testing

3.2 Testing Types

- Functional Testing

3.3 Test Techniques

- White Box Testing

4. Test Environment

4.1 Hardware

- Laptop, keyboard, mouse, stylus

4.2 Software

- Python [3.10]

- unittest

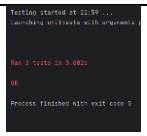
5 Test Deliverables

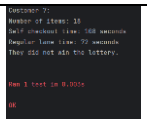
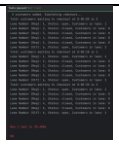
- Test cases

- Test execution reports

6. Test Cases

6.1 Unit Testing and 6.2 Integration testing

| Case | Test case | Inputs | Expected Output | Actual Output | pass/Fail | Corrective Action |
|---|---|---|---|---|---|---|
| F1(lane_ parent) | If lane setup works ok, the code for all the testing was improved GPT |  | The test should pass, as this is the base of the whole code |  | pass | n/a |
| F1(self-service) | If the self service is working fine |  | The should be working fine |  | pass | |

| | | | | | | |
|---|---|---|---|---|---|---|
| F1(reg_lane) | If the regular lane is working |   | They should be working fine |  | Pass and fail | Fix the printing issue |
| F2 | If the customers are being generated with random item number |  | Prints the number of customers it produced, and prints the details for them |  | pass | n/a |
| F3 | To check if the whole code is working fine |  | The should be working fine |  | pass | n/a |

F1:

Lane_parent:

```
import unittest
from lane_parent import Lane

class TestLane(unittest.TestCase):

    def setUp(self):
        # Create an instance of Lane for testing
        self.lane = Lane(lane_type="Checkout", lane_no=1, lane_status="Open", customers_in_lane=5)

    def test_initialization(self):
        self.assertEqual(self.lane.lane_type, second: "Checkout")
        self.assertEqual(self.lane.lane_no, second: 1)
        self.assertEqual(self.lane.lane_status, second: "Open")
        self.assertEqual(self.lane.customers_in_lane, second: 5)

    def test_set_status(self):
        # Test setting status to "Closed"
        self.lane.set_status("Closed")
        self.assertEqual(self.lane.lane_status, second: "Closed")

        # Test setting status back to "Open"
        self.lane.set_status("Open")
        self.assertEqual(self.lane.lane_status, second: "Open")

    def test_get_lane_info(self):
        expected_info = 'Lane Number (Checkout): 1, Status: Open, Customers in lane: 5'
        self.assertEqual(self.lane.get_lane_info(), expected_info)

if __name__ == '__main__':
    unittest.main()
```

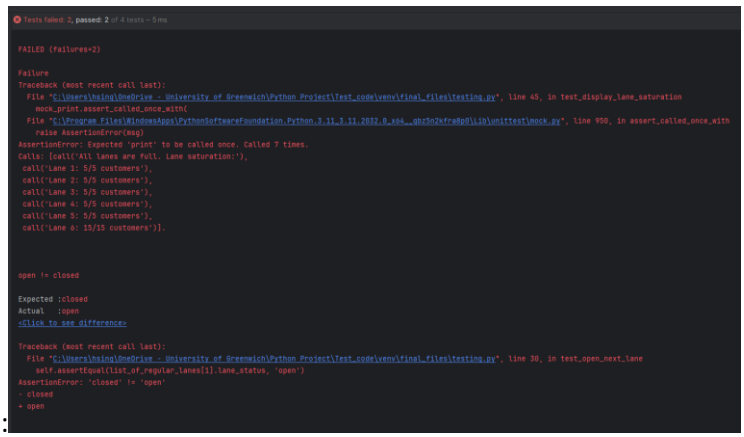
```python
import unittest
from unittest.mock import patch
from regular_lane import lane6, print_self_service_lane_info
from self_service import SelfService


class TestSelfServiceLane(unittest.TestCase):

    def test_self_service_lane_attributes(self):
        self.assertIsInstance(lane6, SelfService)
        self.assertEqual(lane6.lane_type, second: 'Slf')
        self.assertEqual(lane6.lane_no, second: 6)
        self.assertEqual(lane6.lane_status, second: 'open')
        self.assertEqual(lane6.customers_in_lane, second: 0)

    def test_print_self_service_lane_info(self):
        with patch('builtins.print') as mock_print:
            print_self_service_lane_info()

        # Check if the correct print statement is called
        mock_print.assert_called_with(lane6.get_lane_info())


if __name__ == '__main__':
    unittest.main()
```

Reglane:

Self_service:

```python
import unittest
from unittest.mock import patch
from datetime import datetime
from regular_lane import (
    list_of_regular_lanes,
    list_of_all_lanes,
    display_lane_saturation,
    get_total_customers_in_all_lanes,
    timestamp, )


class TestRegularLane(unittest.TestCase):

    def setUp(self):
        # Reset the lanes before each test
        for lane in list_of_regular_lanes:
            lane.customers_in_lane = 0
            lane.set_status('closed')
        for lane in list_of_all_lanes:
            lane.customers_in_lane = 0
            lane.set_status('closed')

    def test_open_next_lane(self):
        lane = list_of_regular_lanes[0]
        lane.customers_in_lane = 6  # Simulate a lane with more than 5 customers

        # Ensure that the next closed lane is opened
        with patch('builtins.print'):
            lane.open_next_lane()
            self.assertEqual(list_of_regular_lanes[1].lane_status, second: 'open')

        # Ensure that only one lane is opened
        self.assertEqual(list_of_regular_lanes[2].lane_status, second: 'closed')

    def test_display_lane_saturation(self):
        # Simulate all lanes and lane6 being full
        for lane in list_of_regular_lanes:
            lane.customers_in_lane = 5
        list_of_all_lanes[5].customers_in_lane = 15

        with patch('builtins.print') as mock_print:
            display_lane_saturation()
```

Reglane output:

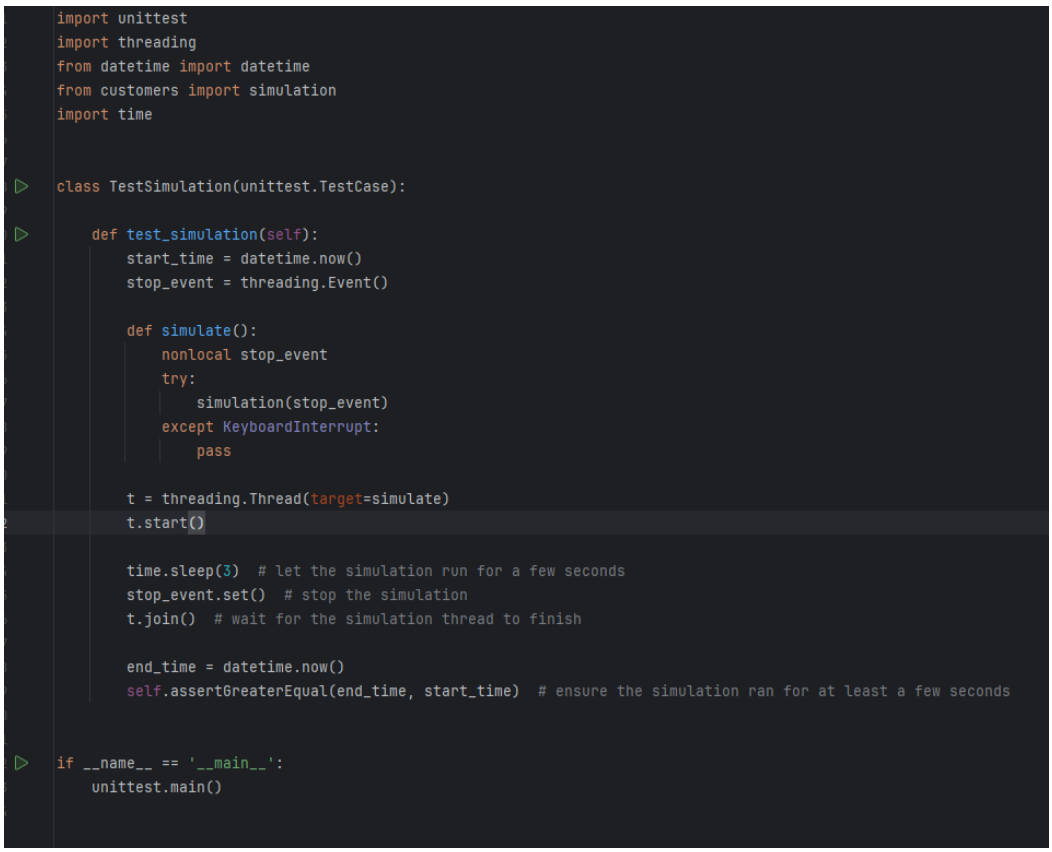


```
    def test_display_lane_saturation(self):
        # Simulate all lanes and lane6 being full
        for lane in list_of_regular_lanes:
            lane.customers_in_lane = 5
        list_of_all_lanes[5].customers_in_lane = 15

        with patch('builtins.print') as mock_print:
            display_lane_saturation()

        # Ensure that the correct saturation information is printed
        mock_print.assert_called_once_with(
            'All lanes are full. Lane saturation:',
        )

    def test_get_total_customers_in_all_lanes(self):
        # Simulate customers in some lanes
        list_of_regular_lanes[0].customers_in_lane = 3
        list_of_regular_lanes[1].customers_in_lane = 2
        list_of_all_lanes[5].customers_in_lane = 7

        total_customers = get_total_customers_in_all_lanes()

        # Ensure that the total number of customers is correct
        self.assertEqual(total_customers, second: 12)

    def test_timestamp(self):
        start_time = datetime( year: 2024, month: 1, day: 22, hour: 22, minute: 55, second: 0)  # Replace with a specific start time
        elapsed_time_info = timestamp(start_time)

        # Ensure that the elapsed time is in the correct format
        self.assertRegex(elapsed_time_info, expected_regex: r'\d+:\d+:\d+')

    # Add more test cases as needed


if __name__ == '__main__':
    unittest.main()
```

F2:

```
import unittest
from random_customer import RandCustomer
import random


class TestRandCustomer(unittest.TestCase):
    def setUp(self):
        self.customer_count = random.randint( a: 1,  b: 10)
        self.rand_customer = RandCustomer(self.customer_count)

    def test_generate_random_customers(self):
        generated_customers = self.rand_customer.generate_random_customers()
        self.assertEqual(len(generated_customers), self.customer_count)

        for customer in generated_customers:
            self.assertTrue(1 <= customer['num_items'] <= 30)
            self.assertTrue(1 <= customer['customer_id'] <= self.customer_count)
            self.assertTrue(1 <= customer['self_checkout_time'] <= (customer['num_items'] * 6))
            self.assertTrue(1 <= customer['regular_lane_time'] <= (customer['num_items'] * 4))
            self.assertEqual(customer['checkout_status'],  second: 'incomplete')
            self.assertTrue(type(customer['won_lottery']) is bool)


if __name__ == '__main__':
    unittest.main()
```

F3:



```python
import unittest
import threading
from datetime import datetime
from customers import simulation
import time


class TestSimulation(unittest.TestCase):

    def test_simulation(self):
        start_time = datetime.now()
        stop_event = threading.Event()

        def simulate():
            nonlocal stop_event
            try:
                simulation(stop_event)
            except KeyboardInterrupt:
                pass

        t = threading.Thread(target=simulate)
        t.start()

        time.sleep(3)  # let the simulation run for a few seconds
        stop_event.set()  # stop the simulation
        t.join()  # wait for the simulation thread to finish

        end_time = datetime.now()
        self.assertGreaterEqual(end_time, start_time)  # ensure the simulation ran for at least a few seconds


if __name__ == '__main__':
    unittest.main()
```

# 7. Annotated Screenshots Demonstrating Implementation

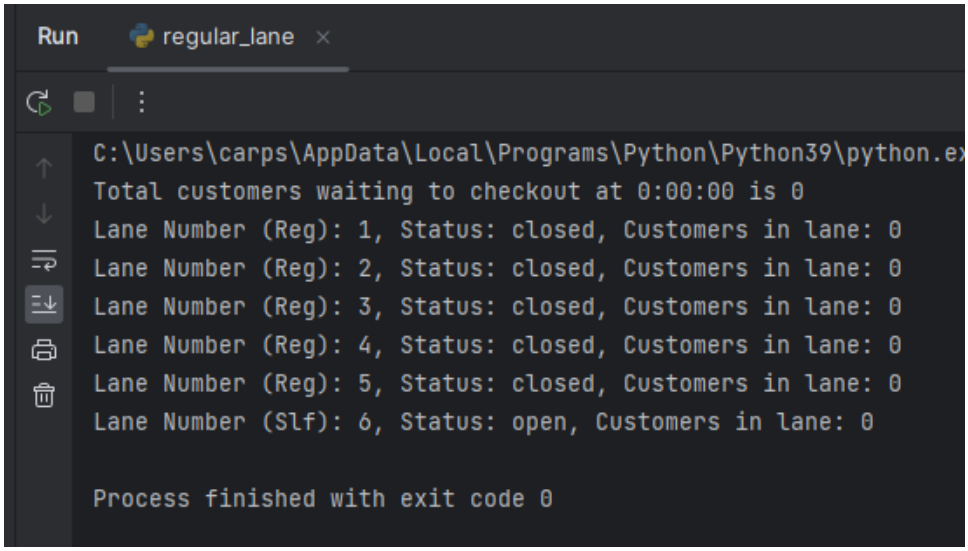
*Provide screenshots that demonstrate the features implemented running – i.e. showing the output produced by all of the subfeatures. Annotate each screenshot and if necessary, provide a brief description for **each** (**up to 100 words**) to explain the code in action.*

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

## 7.1 Feature F1 - Sebastian Carp

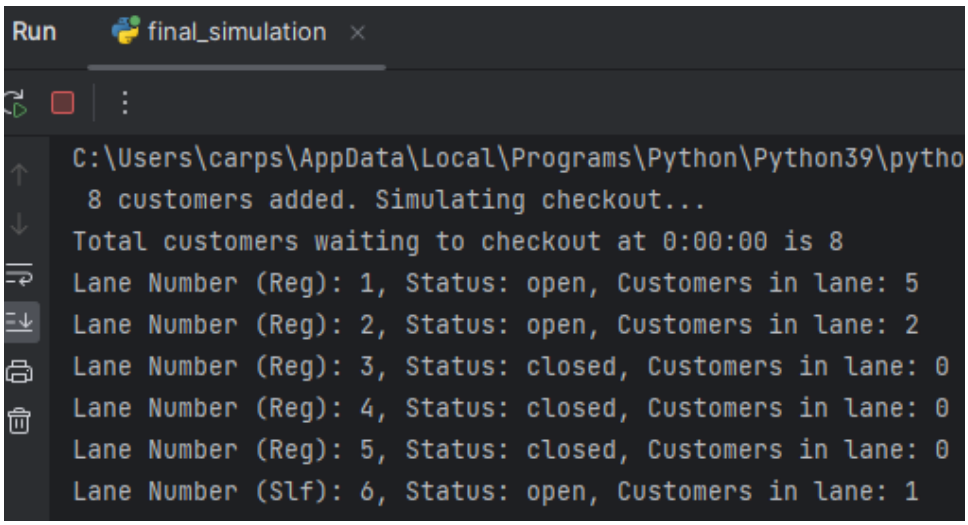
### a. Sub-feature I- screenshots …

Generate a Lane: Data structure for checkout lanes to hold the lane type, lane status, customers in lane, timestamp when created and the total amount of customers waiting to checkout




### b. Sub-feature II- screenshots …

Add a Customer to a Lane: adds generated customers to lanes based on their basket size, lane criteria and lane capacity.




### c. SUB-FEATURE III- SCREENSHOTS …

Remove customers from a lane: if there are customers in lanes, they will be checked out based on their checkout time, which is calculated depending on which type of lane they are in. Customers in regular lanes must wait for the customer before them to finish checking out before they can start their own checkout time and customers in self-checkout lanes can go 8 at a time.

```
9 customers added. Simulating checkout...
Total customers waiting to checkout at 0:00:30 is 12
Lane Number (Reg): 1, Status: open, Customers in lane: 5
Lane Number (Reg): 2, Status: open, Customers in lane: 4
Lane Number (Reg): 3, Status: closed, Customers in lane: 0
Lane Number (Reg): 4, Status: closed, Customers in lane: 0
Lane Number (Reg): 5, Status: closed, Customers in lane: 0
Lane Number (Slf): 6, Status: open, Customers in lane: 3
Total customers waiting to checkout at 0:00:37 is 11
Lane Number (Reg): 1, Status: open, Customers in lane: 5
Lane Number (Reg): 2, Status: open, Customers in lane: 4
Lane Number (Reg): 3, Status: closed, Customers in lane: 0
Lane Number (Reg): 4, Status: closed, Customers in lane: 0
Lane Number (Reg): 5, Status: closed, Customers in lane: 0
Lane Number (Slf): 6, Status: open, Customers in lane: 2
```

### d. SUB-FEATURE IV- SCREENSHOTS …

Open / Close a lane: when customers are added, lanes are opened appropriately based on the current lane saturation. If a lane removed all its customers, it closes automatically.

```
Total customers waiting to checkout at 0:04:36 is 4
Lane Number (Reg): 1, Status: open, Customers in lane: 3
Lane Number (Reg): 2, Status: closed, Customers in lane: 0
Lane Number (Reg): 3, Status: open, Customers in lane: 1
Lane Number (Reg): 4, Status: closed, Customers in lane: 0
Lane Number (Reg): 5, Status: closed, Customers in lane: 0
Lane Number (Slf): 6, Status: open, Customers in lane: 0
```

### e. SUB-FEATURE V- SCREENSHOTS …

Display Lane Status: displays the status of each lane. In this scenario, I have waited for the simulation to finish, to even display all the regular lanes closing when every customer has been removed.

```
Total customers waiting to checkout at 0:05:30 is 2
Lane Number (Reg): 1, Status: open, Customers in lane: 1
Lane Number (Reg): 2, Status: closed, Customers in lane: 0
Lane Number (Reg): 3, Status: open, Customers in lane: 1
Lane Number (Reg): 4, Status: closed, Customers in lane: 0
Lane Number (Reg): 5, Status: closed, Customers in lane: 0
Lane Number (Slf): 6, Status: open, Customers in lane: 0
Total customers waiting to checkout at 0:05:35 is 1
Lane Number (Reg): 1, Status: closed, Customers in lane: 0
Lane Number (Reg): 2, Status: closed, Customers in lane: 0
Lane Number (Reg): 3, Status: open, Customers in lane: 1
Lane Number (Reg): 4, Status: closed, Customers in lane: 0
Lane Number (Reg): 5, Status: closed, Customers in lane: 0
Lane Number (Slf): 6, Status: open, Customers in lane: 0
Total customers waiting to checkout at 0:06:28 is 0
Lane Number (Reg): 1, Status: closed, Customers in lane: 0
Lane Number (Reg): 2, Status: closed, Customers in lane: 0
Lane Number (Reg): 3, Status: closed, Customers in lane: 0
Lane Number (Reg): 4, Status: closed, Customers in lane: 0
Lane Number (Reg): 5, Status: closed, Customers in lane: 0
Lane Number (Slf): 6, Status: open, Customers in lane: 0

Process finished with exit code 0
```
a

## 7.2 Feature F2 - Harmanpreet Singh

### a. Sub-feature I- screenshots (generating customer)

The image below shows the simulation for F2 generates a random number of customers each time—up to ten at a time.

```
There are a total of 7 customers

Customer 1:
```

Each time you get different amount of customers

```
There are a total of 10 customers

Customer 1:
```

### b. Sub-feature II- screenshots (number of items)

Number of items generated each time the loop goes back to generate customer

```
Customer 1:
Number of items: 4
```

```
Customer 2:
Number of items: 23
```

### c. SUB-FEATURE III- SCREENSHOTS (TIME FOR CHECKOUT)

Simple math used to create the checkout_time for both regular lanes and self-service lanes :

```
Self checkout time: 24 seconds
Regular lane time: 16 seconds
```

### d. SUB-FEATURE IV- SCREENSHOTS (LOTTERY AWARD)

created a statement to create a 10% probability that the consumer will win the lottery even if they purchase more than 10 products by just using Random (random.randint(1, 10)).

```
Customer 1:
Number of items: 4
Self checkout time: 24 seconds
Regular lane time: 16 seconds
They did not win the lottery.
```

```
Customer 2:
Number of items: 14
Self checkout time: 84 seconds
Regular lane time: 56 seconds
Congratulations! They won the lottery.
```

### e. SUB-FEATURE V- SCREENSHOTS (DISPLAY THE CUSTOMER BASKET AND PROCESSING TIME)

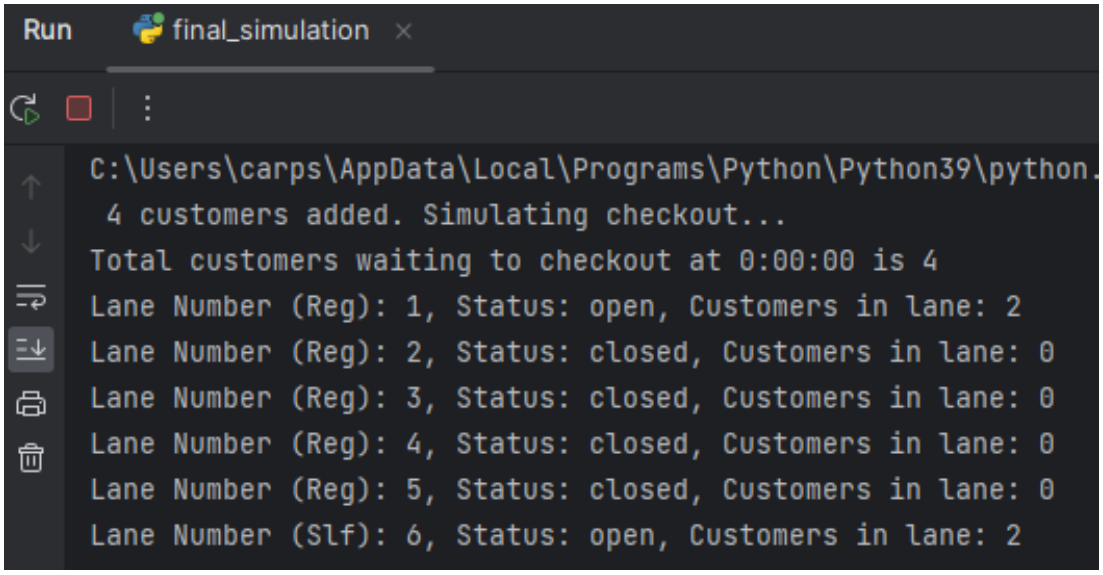
In the last, all the methods have been printed together to create a customer with all the data (customer_no,number of items, time it takes to checkout, if they won the lottery or not)

```
Customer 4:
Number of items: 21
Self checkout time: 126 seconds
Regular lane time: 84 seconds
Congratulations! They won the lottery.

Customer 5:
Number of items: 9
Self checkout time: 54 seconds
Regular lane time: 36 seconds
They did not win the lottery.
```

## 7.3 Feature F3 - Sebastian Carp & Harmanpreet Singh

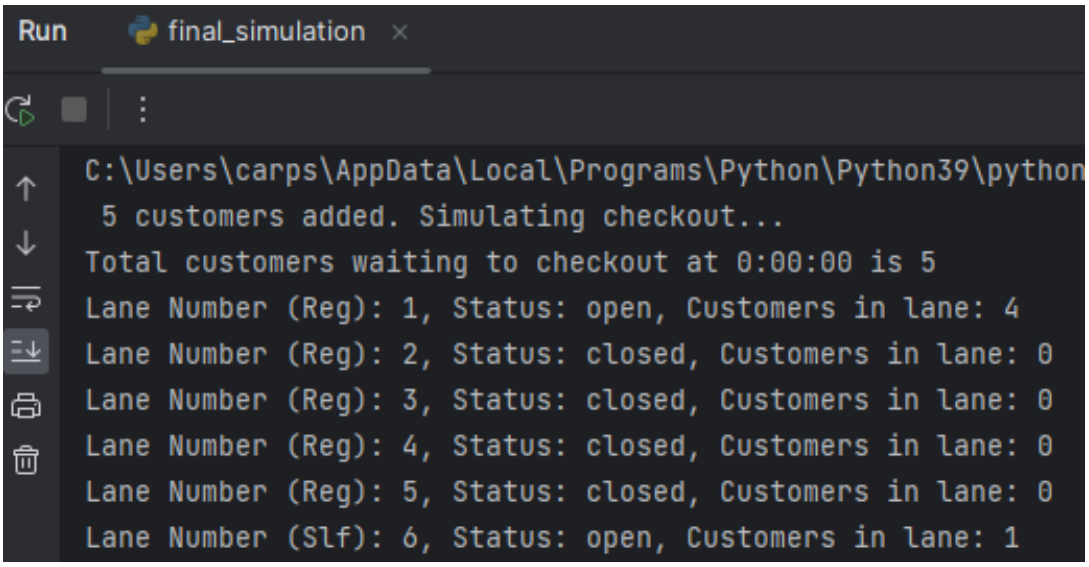
### a. Sub-feature I- screenshots …

Set up Lanes : Initially, 2 lanes are open (1 self-service and one regular), while others remain closed



### b. SUB-FEATURE II- SCREENSHOTS …

Initiate the simulation: Record the timestamp and assume a random number of up to 10 customers join their appropriate lanes.




### c. SUB-FEATURE III- SCREENSHOTS …

Run continuous simulation: keeps adding customers every 30 seconds and removes them after their checkout time is completed. The simulation does not update at fixed time

intervals, but rather when any customer leaves a lane. This has been done for better understanding of customer in an out flow.

```
 7 customers added. Simulating checkout...
Total customers waiting to checkout at 0:02:34 is 15
Lane Number (Reg): 1, Status: open, Customers in lane: 5
Lane Number (Reg): 2, Status: open, Customers in lane: 5
Lane Number (Reg): 3, Status: open, Customers in lane: 3
Lane Number (Reg): 4, Status: closed, Customers in lane: 0
Lane Number (Reg): 5, Status: closed, Customers in lane: 0
Lane Number (Slf): 6, Status: open, Customers in lane: 2
Total customers waiting to checkout at 0:02:48 is 14
Lane Number (Reg): 1, Status: open, Customers in lane: 4
Lane Number (Reg): 2, Status: open, Customers in lane: 5
Lane Number (Reg): 3, Status: open, Customers in lane: 3
Lane Number (Reg): 4, Status: closed, Customers in lane: 0
Lane Number (Reg): 5, Status: closed, Customers in lane: 0
Lane Number (Slf): 6, Status: open, Customers in lane: 2
Total customers waiting to checkout at 0:02:55 is 13
Lane Number (Reg): 1, Status: open, Customers in lane: 4
Lane Number (Reg): 2, Status: open, Customers in lane: 4
Lane Number (Reg): 3, Status: open, Customers in lane: 3
Lane Number (Reg): 4, Status: closed, Customers in lane: 0
Lane Number (Reg): 5, Status: closed, Customers in lane: 0
Lane Number (Slf): 6, Status: open, Customers in lane: 2
Total customers waiting to checkout at 0:03:01 is 12
Lane Number (Reg): 1, Status: open, Customers in lane: 4
Lane Number (Reg): 2, Status: open, Customers in lane: 4
Lane Number (Reg): 3, Status: open, Customers in lane: 3
Lane Number (Reg): 4, Status: closed, Customers in lane: 0
Lane Number (Reg): 5, Status: closed, Customers in lane: 0
Lane Number (Slf): 6, Status: open, Customers in lane: 1
 1 customers added. Simulating checkout...
Total customers waiting to checkout at 0:03:07 is 13
Lane Number (Reg): 1, Status: open, Customers in lane: 4
Lane Number (Reg): 2, Status: open, Customers in lane: 4
Lane Number (Reg): 3, Status: open, Customers in lane: 3
Lane Number (Reg): 4, Status: closed, Customers in lane: 0
Lane Number (Reg): 5, Status: closed, Customers in lane: 0
Lane Number (Slf): 6, Status: open, Customers in lane: 2
```

### d. SUB-FEATURE IV- SCREENSHOTS …

End the simulation: simulation automatically times out when all the customers have been removed (loop end)

```
Total customers waiting to checkout at 0:05:35 is 1
Lane Number (Reg): 1, Status: closed, Customers in lane: 0
Lane Number (Reg): 2, Status: closed, Customers in lane: 0
Lane Number (Reg): 3, Status: open, Customers in lane: 1
Lane Number (Reg): 4, Status: closed, Customers in lane: 0
Lane Number (Reg): 5, Status: closed, Customers in lane: 0
Lane Number (Slf): 6, Status: open, Customers in lane: 0
Total customers waiting to checkout at 0:06:28 is 0
Lane Number (Reg): 1, Status: closed, Customers in lane: 0
Lane Number (Reg): 2, Status: closed, Customers in lane: 0
Lane Number (Reg): 3, Status: closed, Customers in lane: 0
Lane Number (Reg): 4, Status: closed, Customers in lane: 0
Lane Number (Reg): 5, Status: closed, Customers in lane: 0
Lane Number (Slf): 6, Status: open, Customers in lane: 0

Process finished with exit code 0
```

# 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.*

1st “improved” code:

Ai logic: “In this improved version: I removed unnecessary nested conditions and simplified the logic. I used target_lane as a variable to hold the initial assignment based on the number of items, which improves code readability. I consolidated the logic for choosing between self-checkout and regular lanes based on the conditions. I used a single block of code to find the best regular lane, avoiding code duplication. This version is more concise, easier to read, and eliminates redundant checks.”

This might seem accurate at first, however when we changed this bit of the and ran the simulation, the results instantly started to bug out, not following any of the instructions accordingly.

The AI generated code can potentially be improved by modifying bits from the rest of the file, however this is really time consuming and can induce programmers in error.

See snippet attached:

```
 8 customers added. Simulating checkout...
Total customers waiting to checkout at 0:00:00 is 8
Lane Number (Reg): 1, Status: open, Customers in lane: 1
Lane Number (Reg): 2, Status: closed, Customers in lane: 1
Lane Number (Reg): 3, Status: closed, Customers in lane: 1
Lane Number (Reg): 4, Status: closed, Customers in lane: 1
Lane Number (Reg): 5, Status: closed, Customers in lane: 1
Lane Number (Slf): 6, Status: open, Customers in lane: 3
Total customers waiting to checkout at 0:00:31 is 7
Lane Number (Reg): 1, Status: open, Customers in lane: 1
Lane Number (Reg): 2, Status: closed, Customers in lane: 1
Lane Number (Reg): 3, Status: closed, Customers in lane: 1
Lane Number (Reg): 4, Status: closed, Customers in lane: 1
Lane Number (Reg): 5, Status: closed, Customers in lane: 1
 2 customers added. Simulating checkout...
Lane Number (Slf): 6, Status: open, Customers in lane: 1
Total customers waiting to checkout at 0:00:31 is 8
Total customers waiting to checkout at 0:00:31 is 8Lane Number (Reg): 1, Status: open, Customers in lane: 2

Lane Number (Reg): 1, Status: open, Customers in lane: 2
Lane Number (Reg): 2, Status: closed, Customers in lane: 2Lane Number (Reg): 2, Status: closed, Customers in lane: 2
Lane Number (Reg): 3, Status: closed, Customers in lane: 1
Lane Number (Reg): 4, Status: closed, Customers in lane: 1

Lane Number (Reg): 5, Status: closed, Customers in lane: 1Lane Number (Reg): 3, Status: closed, Customers in lane: 1
Lane Number (Reg): 4, Status: closed, Customers in lane: 1
Lane Number (Reg): 5, Status: closed, Customers in lane: 1
Lane Number (Slf): 6, Status: open, Customers in lane: 1

Lane Number (Slf): 6, Status: open, Customers in lane: 1
Total customers waiting to checkout at 0:00:40 is 8
Lane Number (Reg): 1, Status: open, Customers in lane: 2
```

| *Code written from scratch* | *OpenAi improved code* |
|---|---|

```
def add_customer_to_lane(self):
    """
    Adds customers to lanes based on basket size, lane criteria and lane capacity
    :return:
    """
    if self.number_of_items < 10:  # If a customer has less than 10 items, he can go to self-checkout
        if lane6.customers_in_lane >= 15:  # If the self-checkout lane is full, customers will go to a regular lane
            for lane in list_of_regular_lanes:  # Checking which regular lane is best to join
                if lane.customers_in_lane < 5:
                    lane.customers_in_lane += 1
                    self.assigned_lane = lane  # Assigns customer to regular lane
                    break
            else:
                min_lane = min(list_of_regular_lanes, key=lambda x: x.customers_in_lane)
                min_lane.customers_in_lane += 1
                self.assigned_lane = min_lane  # Assigns customer to next less crowded lane
        else:
            lane6.customers_in_lane += 1  # Adding customer to self-checkout lane
            self.assigned_lane = lane6
    elif self.number_of_items >= 10:  # If a customer has more than 10 items, he needs to go to a regular lane
        for lane in list_of_regular_lanes:  # Checking which regular lane is best to join
            if lane.customers_in_lane < 5:
                lane.customers_in_lane += 1
                self.assigned_lane = lane  # Assigns customer to lane
                break
        else:
            min_lane = min(list_of_regular_lanes, key=lambda x: x.customers_in_lane)
            min_lane.customers_in_lane += 1
            self.assigned_lane = min_lane  # Assigns customer to next less crowded lane
```

```
def add_customer_to_lane(self):
    """
    Adds customers to lanes based on basket size, lane criteria, and lane capacity
    :return:
    """
    target_lane = lane6 if self.number_of_items < 10 else None

    if target_lane and target_lane.customers_in_lane >= 15:
        # If self-checkout lane is full, find the best regular lane
        best_regular_lane = min(list_of_regular_lanes, key=lambda x: x.customers_in_lane)
        best_regular_lane.customers_in_lane += 1
        self.assigned_lane = best_regular_lane
    elif target_lane:
        # Assign the customer to the self-checkout lane
        target_lane.customers_in_lane += 1
        self.assigned_lane = target_lane
    else:
        # If the customer has more than 10 items, find the best regular lane
        best_regular_lane = min(list_of_regular_lanes, key=lambda x: x.customers_in_lane)
        best_regular_lane.customers_in_lane += 1
        self.assigned_lane = best_regular_lane
```

2nd improved code:
From what we have seen throughout development, most of the times the AI generated code did not even work, throwing unresolved references or syntax errors. However, the good part about using artificial intelligence is that as a programmer you can get ideas relatively fast when you are stuck. For example, in this part of the code, OpenAI has managed to compress every line efficiently, resulting in faster execution time:

```
def print_lanes_info():  # improved by OpenAi
    """
    Returns the info of all lanes, including saturation if lanes are full
    :return:
    """

    # checks for lanes with 0 customers in them and closes them
    for lane in list_of_regular_lanes:
        if lane.customers_in_lane == 0:
            lane.set_status('closed')

    # prints elapsed time and customers waiting to check out
    print(
        f'Total customers waiting to checkout at {timestamp(simulation_start_time)} is '
        f'{get_total_customers_in_all_lanes()}')

    # goes through a series of checks to decide if the next lane should open
    # prints all the regular lanes info
    for idx, lane in enumerate(list_of_regular_lanes, start=1):
        lane.open_next_lane()
        print(lane.get_lane_info())

    # prints the self-service lane info
    print_self_service_lane_info()

    # upon the lanes being full, prints lane saturation
    display_lane_saturation()
```

The changes made are not significant, with 80% of the code snippet remaining the same, however a loop was added for the printing of the regular lanes and the function looks cleaner now. We consider that if OpenAI is used responsibly, any project will be way more efficient at the end.

3rd improved code:
Another example of AI use, and the last one within our project is getting an idea of what to use to run the simulation for the lanes. In the simulation() function, OpenAI provided these two lines:

```
threads = [threading.Thread(target=simulate_checkout_for_lane, args=(lane_customers[i],)) for
i in range(5)]
threads += [threading.Thread(target=customer.simulate_checkout_and_remove_customer) for
customer i
            lane6_customers_to_checkout]
```

Because of too many generations attempted by the AI until we got to these two final lines, we managed to get confused when the solution to the problem we were facing was quite simple, so it is important to find a balance point and be precise when searching or asking for an idea.

# 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: 19 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: 9 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: 16 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: 8 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.*

## *List of .py files :*

### 1. *lane_parent.py:*

```
"""
lane_parent.py contains the parent class for "Lane" which defines arguments which will be
inherited by any subclasses
"""
# Author Name: Sebastian Carp
# Author Student ID: 001289569
# Python interpreter used: Python 3.10


class Lane:
    def __init__(self, lane_type, lane_no, lane_status, customers_in_lane):  # arguments to be
inherited by subclasses
        self.lane_type = lane_type
        self.lane_no = lane_no
        self.lane_status = lane_status
        self.customers_in_lane = customers_in_lane

    def set_status(self, lane_status):  # update status to open / closed
        self.lane_status = lane_status

    def get_lane_info(self):  # returns the lane info for a specific lane
        info = f'Lane Number ({self.lane_type}): {self.lane_no},' ' ' \
               f'Status: {self.lane_status},' ' ' \
               f'Customers in lane: {self.customers_in_lane}'
        return info
```

## 2. *regular_lane.py:*

```
"""
regular_lane.py has the functionality of inheriting from the parent class "Lane" in the
subclass RegLane
                also defines the structure (variables) for the 5 regular lanes used within the
simulation
"""
# Author Name: Sebastian Carp
# Author Student ID: 001289569
# Python interpreter used: Python 3.10

# imports for python file to be functional:
from lane_parent import Lane
from self_service import print_self_service_lane_info
from self_service import lane6
from datetime import datetime
def display_lane_saturation():
    """
    In case all lanes are full, display lane saturation (the amount of customers in each lane)
    """
    if all(lane.customers_in_lane >= 5 for lane in list_of_regular_lanes) and
lane6.customers_in_lane >= 15:
        print('All lanes are full. Lane saturation:')
        for lane in list_of_regular_lanes:
            print(f"Lane {lane.lane_no}: {lane.customers_in_lane}/5 customers")
        print(f'Lane {lane6.lane_no}: {lane6.customers_in_lane}/15 customers')


def get_total_customers_in_all_lanes():
    """
    Returns the sum of the customers in all lanes as an integer
    :return:
    """
    total_customers = sum(lane.customers_in_lane for lane in list_of_all_lanes)
    return total_customers


def timestamp(start_time):
    """
    Returns the elapsed time since the start of the simulation
    :param start_time:
    :return:
    """
    current_time = datetime.now()
    elapsed_time = current_time - start_time
    elapsed_time_info = str(elapsed_time).split('.')[0]  # formatting elapsed time
    return elapsed_time_info
```

```
# marks the start of the simulation with the current time
simulation_start_time = datetime.now()


def print_lanes_info():  # improved by OpenAi
    """
    Returns the info of all lanes, including saturation if lanes are full
    :return:
    """

    # checks for lanes with 0 customers in them and closes them
    for lane in list_of_regular_lanes:
        if lane.customers_in_lane == 0:
            lane.set_status('closed')

    # prints elapsed time and customers waiting to check out
    print(
        f'Total customers waiting to checkout at {timestamp(simulation_start_time)} is '
        f'{get_total_customers_in_all_lanes()}')

    # goes through a series of checks to decide if the next lane should open
    # prints all the regular lanes info
    for idx, lane in enumerate(list_of_regular_lanes, start=1):
        lane.open_next_lane()
        print(lane.get_lane_info())

    # prints the self-service lane info
    print_self_service_lane_info()

    # upon the lanes being full, prints lane saturation
    display_lane_saturation()


class RegLane(Lane):
    """
    Subclass of "Lane" for the regular lanes, inheriting its attributes
    Contains method for checking if the next lane needs to be opened or not
    """
    def __init__(self, lane_type, lane_no, lane_status, customers_in_lane):
        super().__init__(lane_type, lane_no, lane_status, customers_in_lane)  # inheriting
from parent class

    # checking if the next lane needs to be opened or not
    def open_next_lane(self):
        for lane in list_of_regular_lanes:
            if self.customers_in_lane >= 5 and lane.lane_status == 'closed' and
lane.customers_in_lane > 0:  # check
                # if conditions met
                lane.set_status('open')  # opens next lane if conditions met
                break  # exit the loop once a lane is opened to open only the next one
```

```
# assigns lane type, lane number, lane status, and number of customers in lane
# data types for each attribute are mapped (str, int, str, int)
lane1 = RegLane('Reg', 1, 'open', 0)
lane2 = RegLane('Reg', 2, 'closed', 0)
lane3 = RegLane('Reg', 3, 'closed', 0)
lane4 = RegLane('Reg', 4, 'closed', 0)
lane5 = RegLane('Reg', 5, 'closed', 0)

# defining lists for lanes which will be used to manage customers
list_of_regular_lanes = [lane1, lane2, lane3, lane4, lane5]  # list of regular lanes
list_of_all_lanes = [lane1, lane2, lane3, lane4, lane5, lane6]  # list of all lanes
Self
```

### 3. *self_service.py:*

```
"""
regular_lane.py has the functionality of inheriting from the parent class "Lane" in the
subclass RegLane
                also defines the structure (variables) for the 5 regular lanes used within the
simulation
"""
# Author Name: Sebastian Carp
# Author Student ID: 001289569
# Python interpreter used: Python 3.10

# imports for python file to be functional:
from lane_parent import Lane


def print_self_service_lane_info():
    """
    Prints the info of the self-service lane
    :return:
    """
    print(lane6.get_lane_info())  # uses parent class method


class SelfService(Lane):
    """
    Subclass of "Lane" for the self-service lane, inheriting its attributes
    """
    def __init__(self, lane_type, lane_no, lane_status, customers_in_lane):
        super().__init__(lane_type, lane_no, lane_status, customers_in_lane)


# assigns lane type, lane number, lane status, and number of customers in lane
# data types for each attribute are mapped (str, int, str, int)
lane6 = SelfService('Slf', 6, "open", 0)
```

### 4. random_customer.py:

```
"""
random_customer.py has the functionality of generating random customers with a random amount
of items in their basket
"""
# Author Name: Harmanpreet Singh
# Author Student ID: 001289568
# Python interpreter used: Python 3.10

# random_customer.py

import random
class RandCustomer:
    def __init__(self, customer_num):
        """
        Initializes a Customer object with a given customer number.
        """
        self.customer_num = customer_num

    @staticmethod
    def generate_random_items():
        """
        Generates a random number of items between 1 and 30 for each customer.
        """
        return random.randint(1, 30)

    @staticmethod
    def calculate_checkout_times(num_items):
        """
        Calculates the time it takes to check out each customer's items in the regular lane
and self-checkout.
        """
        regular_lane_time = (num_items * 4)  # checkout_time for regular_lane
        self_checkout_time = (num_items * 6)  # for self_checkout, calculate the time
        return round(regular_lane_time, 2), round(self_checkout_time, 2)
```

```
    @staticmethod
    def is_lottery_winner(num_items):
        """
        Determine if the customer wins the lottery
        :return:
        """
        return num_items >= 10 and random.randint(1,10) == 1  # creating 10% chances of anyone
with more than 10 items wins lottery

    def generate_random_customers(self):

        """
        Generates a list of randomly generated customers.
        """
        customers = []
        for customer_number in range(1, self.customer_num + 1):
            number_of_items = self.generate_random_items()  # generates random of items
            regular_lane_time, self_checkout_time = self.calculate_checkout_times(
                number_of_items)  # calculates the checkout_time
            checkout_status = 'incomplete'

            won_lottery = self.is_lottery_winner(number_of_items)

            customer_info = {
                'customer_id': customer_number,
                'num_items': number_of_items,
                'won_lottery': won_lottery,
                'self_checkout_time': self_checkout_time,
                'regular_lane_time': regular_lane_time,
                'checkout_status': checkout_status
            }

            customers.append(customer_info)

        return customers


customer_in_total = random.randint(1, 10)  # generates a random number of customers
print(f"There are a total of {customer_in_total} customers")  # prints how many customers
there are in total
c = RandCustomer(customer_in_total)  # creates a variable of the class

# / uses a method within the class to assign the customer info and
# stores the data in a variable "generated_customers" /
generated_customers = c.generate_random_customers()

# prints the customer information
for customer in generated_customers:
    print(f"\nCustomer {customer['customer_id']}:")
    print(f"Number of items: {customer['num_items']}")
    print(f"Self checkout time: {customer['self_checkout_time']} seconds")
    print(f"Regular lane time: {customer['regular_lane_time']} seconds")
    if customer['won_lottery']:
```

```
        print("Congratulations! They won the lottery.")
    else:
        print("They did not win the lottery.")
```

5. customer.py:

```
"""
customers.py is used to run the supermarket simulation
            inheritance, abstraction and encapsulation have all been used for the file to run
accordingly
"""
# Author Name: Sebastian Carp, Harmanpreet Singh
# Author Student ID: 001289569, 001289568
# Python interpreter used: Python 3.10


import random
import time
import threading
from regular_lane import print_lanes_info, list_of_regular_lanes, lane1, lane2, lane3, lane4,
lane5, lane6
# define a threading lock for printing
# / Each lane info update may seem 'laggy', but this ensures no printing
# overlapping when two customers check out at the same time, no matter the lane.
# The only time that overlapping might happen is when
# customers are generated and a customer checks out at the same time /
print_lock = threading.Lock()class Customer:
    """
    Class to which all the customers will be modelled by
    """
    def __init__(self, customer_number, number_of_items, checkout_time, checkout_status):
        self.customer_number = customer_number
        self.number_of_items = number_of_items
        self.checkout_time = checkout_time
        self.checkout_status = checkout_status
        self.assigned_lane = None  # Initialize initial assigned_lane attribute
        self.lottery_winner = False  # Initialize lottery_winner attribute

    def add_customer_to_lane(self):
        """
        Adds customers to lanes based on basket size, lane criteria and lane capacity
        :return:
        """
        if self.number_of_items < 10:  # If a customer has less than 10 items, he can go to
self-checkout
            if lane6.customers_in_lane >= 15:  # If the self-checkout lane is full, customers
will go to a regular lane
                for lane in list_of_regular_lanes:  # Checking which regular lane is best to
join
                    if lane.customers_in_lane < 5:
                        lane.customers_in_lane += 1
                        self.assigned_lane = lane  # Assigns customer to regular lane
                        break
```

```
                else:
                    min_lane = min(list_of_regular_lanes, key=lambda x: x.customers_in_lane)
                    min_lane.customers_in_lane += 1
                    self.assigned_lane = min_lane  # Assigns customer to next less crowded
lane
            else:
                lane6.customers_in_lane += 1  # Adding customer to self-checkout lane
                self.assigned_lane = lane6
        elif self.number_of_items >= 10:  # If a customer has more than 10 items, he needs to
go to a regular lane
            for lane in list_of_regular_lanes:  # Checking which regular lane is best to join
                if lane.customers_in_lane < 5:
                    lane.customers_in_lane += 1
                    self.assigned_lane = lane  # Assigns customer to lane
                    break
            else:
                min_lane = min(list_of_regular_lanes, key=lambda x: x.customers_in_lane)
                min_lane.customers_in_lane += 1
                self.assigned_lane = min_lane  # Assigns customer to next less crowded lane

    def remove_customer_from_lane(self):
        """
        Removes a customer from their lane based on basket size
        :return:
        """
        if self.checkout_status == 'complete':
            if self.number_of_items < 10:
                lane6.customers_in_lane -= 1
            if self.number_of_items > 10:
                if self.assigned_lane:
                    self.assigned_lane.customers_in_lane -= 1

    def simulate_checkout_and_remove_customer(self):
        """
        Using real elapsed time since the start of the simulation to set a customers checkout
status to complete
        and remove them from their lane
        :return:
        """
        start_time = time.time()  # Record the start time of the simulation

        while time.time() - start_time < self.checkout_time:
            # Keep checking if the checkout time has elapsed
            pass

        self.checkout_status = 'complete'  # Update the checkout status to 'complete'

        # Check if the customer wins the lottery
        self.lottery_winner = self.checkout_status == 'complete' and self.is_lottery_winner()
        if self.lottery_winner:  # If the customer meets the conditions for winning the
lottery
            print(f'Customer {self.customer_number} has won the lottery')  # Prints the
customer number which has won
```

```
            self.remove_customer_from_lane()  # Removes the customer from the lane

            # Printing the lanes info with a lock to ensure synchronized printing
            with print_lock:
                print_lanes_info()

    def is_lottery_winner(self):
        """
        Determine if the customer wins the lottery
        :return:
        """
        return self.number_of_items >= 10 and random.randint(1, 10) == 1


def simulate_checkout_for_lane(customers):
    """
    Removes the customers given
    :param customers:
    :return:
    """
    for customer in customers:
        customer.simulate_checkout_and_remove_customer()


def generate_random_customer(customer_number):
    """
    Generates a random customer
    :param customer_number:
    :return:
    """
    number_of_items = random.randint(1, 30)  # Random number of items between 1 and 30
    checkout_time = number_of_items * 6 if number_of_items < 10 else number_of_items * 4
    # checkout time --> x items * 6 for self-checkout; x items * 4 for regular lanes
    checkout_status = 'incomplete'  # Initial checkout status

    # Stores each customer as a variable of the Customer class
    return Customer(customer_number, number_of_items, checkout_time, checkout_status)


def get_customers_in_lane(customers, lane):
    """
    Returns the customers within a specific lane
    :param customers:
    :param lane:
    :return:
    """
    return [customer for customer in customers if customer.assigned_lane == lane]


def simulation(stop_event):
    """
```

```
    Generates, assigns and checks out customers; displays simulation in python console
    """
    # The simulation automatically times out after all the customers have been removed
    for _ in range(8):

        if stop_event.is_set():  # use for GUI
            break  # Exit the loop if the stop event is set

        # / Loops 8 times with a random number of new customers between 1 and 10 so customers
waiting
        # to join any lane cannot exceed 40 (80 maximum customers within the lanes and waiting
to join any lane) /

        # Generate a random number of customers between 1 and 10
        num_customers = random.randint(1, 10)

        # Generate a list of random customer variables
        random_customers = [generate_random_customer(i) for i in range(1, num_customers + 1)]

        # Add the randomly generated customers to the lanes
        for customer in random_customers:
            customer.add_customer_to_lane()

        # Get customers in each lane
        lanes = [lane1, lane2, lane3, lane4, lane5, lane6]
        lane_customers = [get_customers_in_lane(random_customers, lane) for lane in lanes]
        lane6_customers_to_checkout = lane_customers[-1][:8]

        # Print confirmation of customers added to lanes, print lanes info
        print(f" {num_customers} customers added. Simulating checkout...")
        print_lanes_info()

        # Start new threads for newly added customers
        # Checks out newly added customers from their lanes in order
        # Use of threads allows the loop to continue
        threads = [threading.Thread(target=simulate_checkout_for_lane,
args=(lane_customers[i],)) for i in range(5)]
        threads += [threading.Thread(target=customer.simulate_checkout_and_remove_customer)
for customer in
                    lane6_customers_to_checkout]


# Start all threads
        for thread in threads:
            thread.start()

        # Print customers each 30 seconds
        time.sleep(30)

    # Prints lanes without any customers left
    print_lanes_info()
```

## 6. *Final_simulation.py:*

```
"""
final_simulation.py has been created with the purpose of executing the simulation cleanly,
                    without any code being visible
"""
# Author Name: Sebastian Carp, Harmanpreet Singh
# Author Student ID: 001289569, 001289568
# Python interpreter used: Python 3.10

from customers import simulation
import threading


def run_simulation():
    # Create a threading.Event to be able to stop the simulation if needed
    stop_event = threading.Event()

    # Call the simulation function
    simulation(stop_event)


if __name__ == "__main__":
    # Run the simulation in this file
    run_simulation()
```