FINAL PROJECT

ALGO & PROGRAMMING

[2022]

[Binus International University]

Authored by :

Christopher Alexander Tjiandra

L1BC

[Specifications]

- **Vision & Mission :**

➢ **Create a program that can realize how parking lots work in the form of a digital system.**

- **Project Description :**

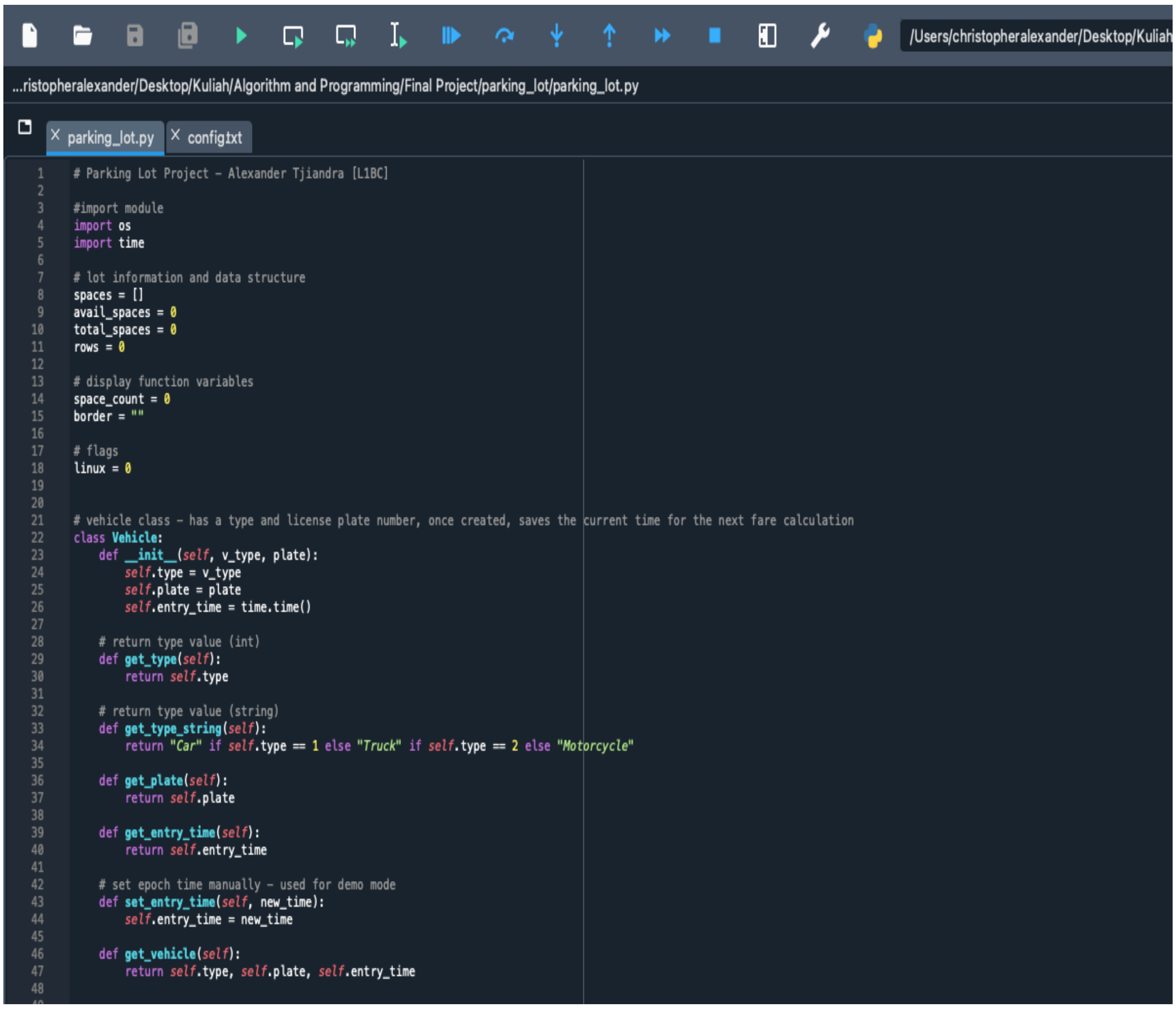
➢ **This "Parking Lot" project is a project that realizes how a parking lot works. There will be an option where we will park our vehicle, the process when we want to get out of the parking lot, the price to be paid, and there will also be an option to view a description of the vehicle being parked.**

[Component / Code description]

- **Files :**

  - **parking_lot.py**
    - **This file contains the source code that I created to run this parking lot program.**
  - **config.txt**
    - **This file is a config file that will be used in my source code by reading this config.txt file and connecting it to my source code.**

1.parking_lot.py



/Users/christopheralexander/Desktop/Kuliah

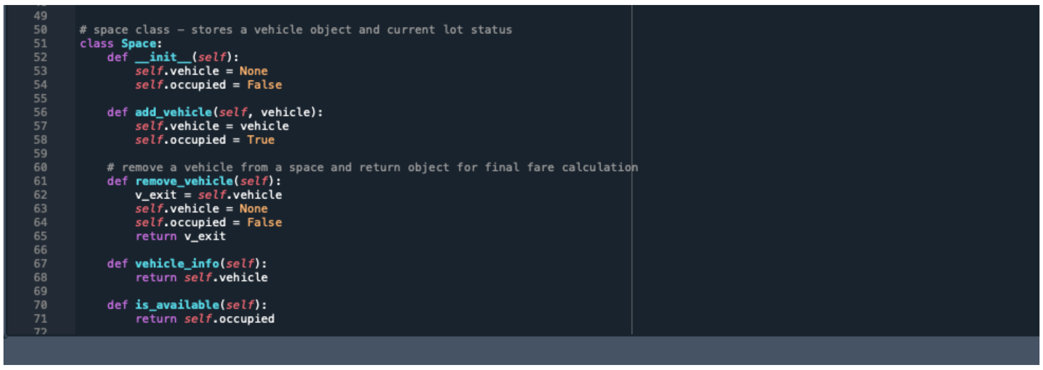
...ristopheralexander/Desktop/Kuliah/Algorithm and Programming/Final Project/parking_lot/parking_lot.py

parking_lot.py | config.txt

```python
# Parking Lot Project - Alexander Tjiandra [L1BC]

#import module
import os
import time

# lot information and data structure
spaces = []
avail_spaces = 0
total_spaces = 0
rows = 0

# display function variables
space_count = 0
border = ""

# flags
linux = 0


# vehicle class - has a type and license plate number, once created, saves the current time for the next fare calculation
class Vehicle:
    def __init__(self, v_type, plate):
        self.type = v_type
        self.plate = plate
        self.entry_time = time.time()

    # return type value (int)
    def get_type(self):
        return self.type

    # return type value (string)
    def get_type_string(self):
        return "Car" if self.type == 1 else "Truck" if self.type == 2 else "Motorcycle"

    def get_plate(self):
        return self.plate

    def get_entry_time(self):
        return self.entry_time

    # set epoch time manually - used for demo mode
    def set_entry_time(self, new_time):
        self.entry_time = new_time

    def get_vehicle(self):
        return self.type, self.plate, self.entry_time

```

class Vehicle :

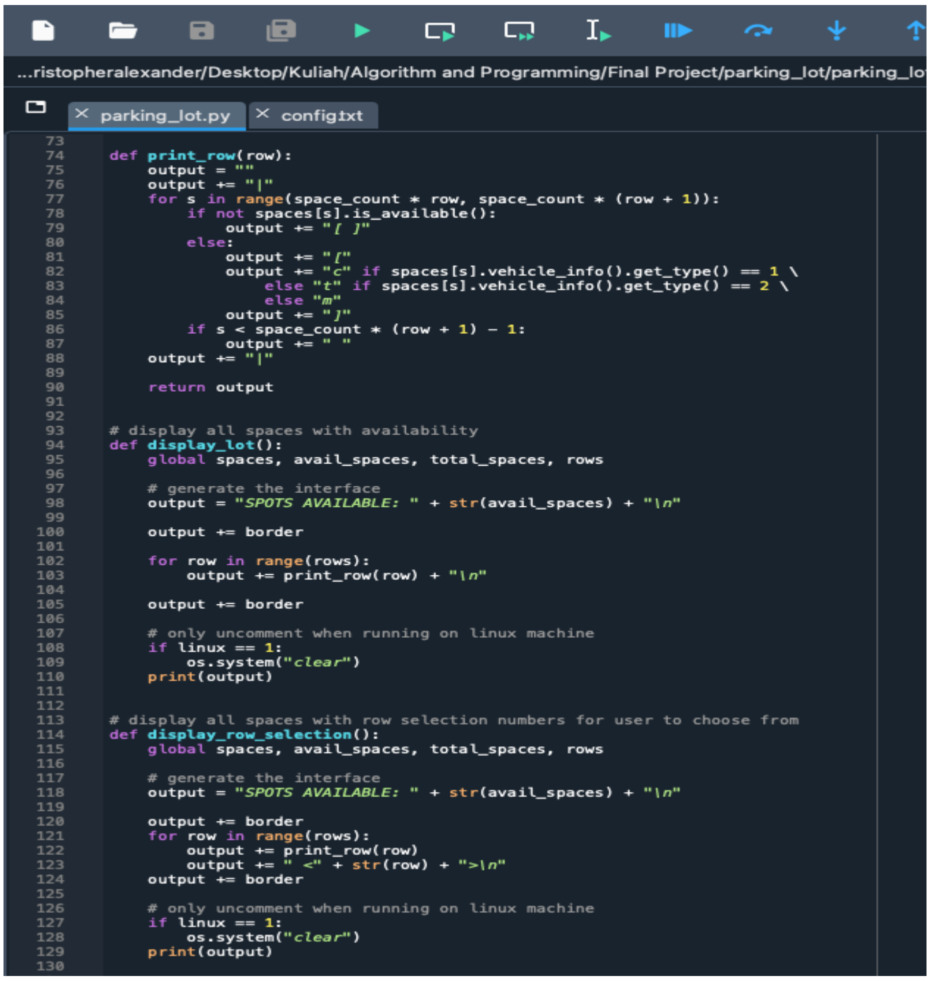
- This class has a type of license plate number, and also a type of vehicle (car, truck, motorcycle).



```python

# space class - stores a vehicle object and current lot status
class Space:
    def __init__(self):
        self.vehicle = None
        self.occupied = False

    def add_vehicle(self, vehicle):
        self.vehicle = vehicle
        self.occupied = True

    # remove a vehicle from a space and return object for final fare calculation
    def remove_vehicle(self):
        v_exit = self.vehicle
        self.vehicle = None
        self.occupied = False
        return v_exit

    def vehicle_info(self):
        return self.vehicle

    def is_available(self):
        return self.occupied

```

class Space :

- The class that contains how to enter the vehicle object into the parking lot and shows the status of the available places. In addition, this class also contains a program to remove the vehicle from the previous parking lot.


...ristopheralexander/Desktop/Kuliah/Algorithm and Programming/Final Project/parking_lot/parking_lo
parking_lot.py
config.txt

```python
def print_row(row):
    output = ""
    output += "|"
    for s in range(space_count * row, space_count * (row + 1)):
        if not spaces[s].is_available():
            output += "[ ]"
        else:
            output += "["
            output += "c" if spaces[s].vehicle_info().get_type() == 1 \
                else "t" if spaces[s].vehicle_info().get_type() == 2 \
                else "m"
            output += "]"
        if s < space_count * (row + 1) - 1:
            output += " "
    output += "|"

    return output


# display all spaces with availability
def display_lot():
    global spaces, avail_spaces, total_spaces, rows

    # generate the interface
    output = "SPOTS AVAILABLE: " + str(avail_spaces) + "\n"

    output += border

    for row in range(rows):
        output += print_row(row) + "\n"

    output += border

    # only uncomment when running on linux machine
    if linux == 1:
        os.system("clear")
    print(output)


# display all spaces with row selection numbers for user to choose from
def display_row_selection():
    global spaces, avail_spaces, total_spaces, rows

    # generate the interface
    output = "SPOTS AVAILABLE: " + str(avail_spaces) + "\n"

    output += border
    for row in range(rows):
        output += print_row(row)
        output += " <" + str(row) + ">\n"
    output += border

    # only uncomment when running on linux machine
    if linux == 1:
        os.system("clear")
    print(output)
```

def print_row( ) :

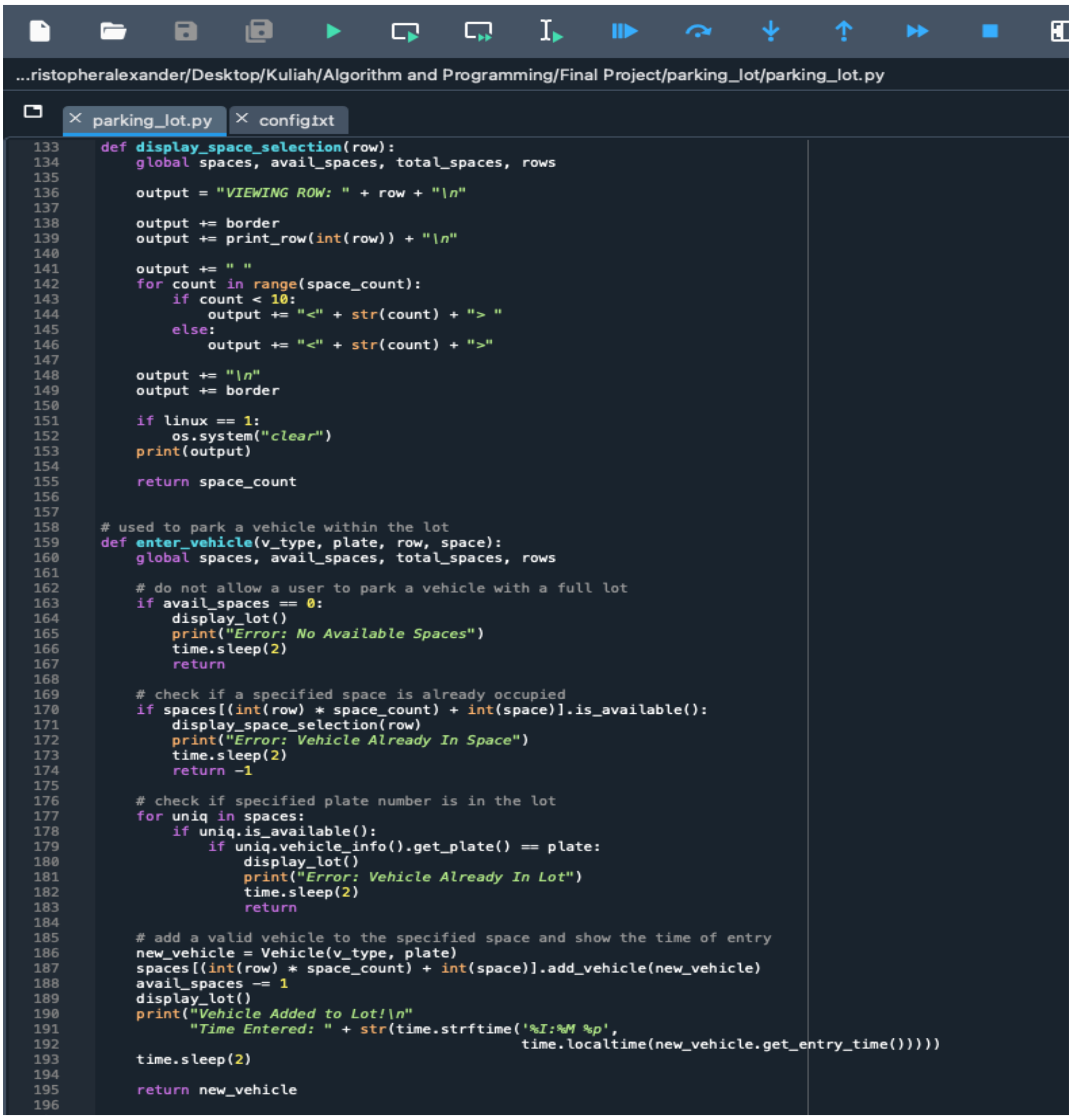
- Functions as a program to print vehicles into rows in the form of several categories (c for cars, t for trucks, and m for motorcycles)

def display_lot( ) :

- Function to display all available spaces/lot

def display_row_selection( ) :

- Function to display all spaces with row selection numbers for user


...ristopheralexander/Desktop/Kuliah/Algorithm and Programming/Final Project/parking_lot/parking_lot.py
parking_lot.py
config.txt

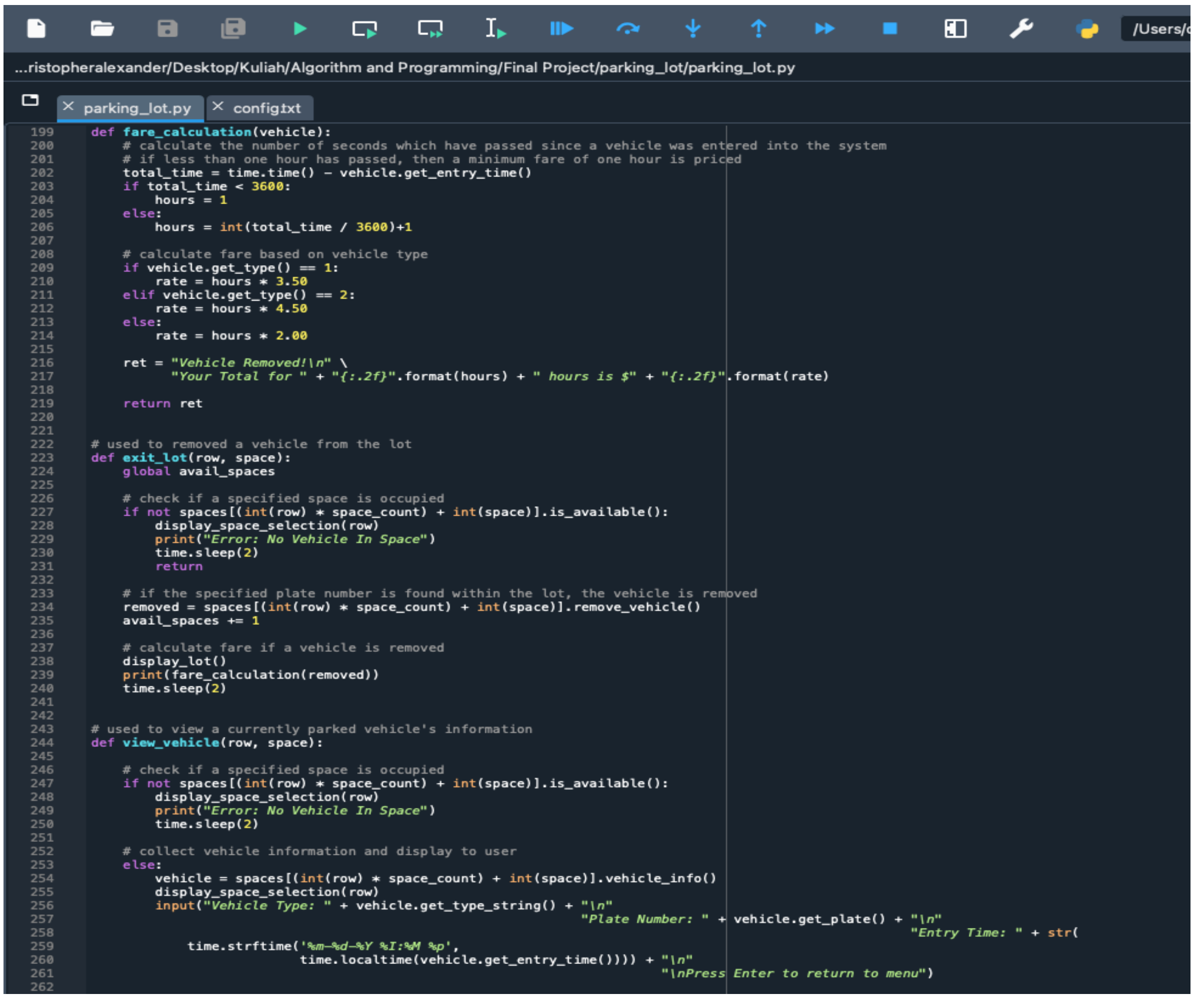
```python
def display_space_selection(row):
    global spaces, avail_spaces, total_spaces, rows

    output = "VIEWING ROW: " + row + "\n"

    output += border
    output += print_row(int(row)) + "\n"

    output += " "
    for count in range(space_count):
        if count < 10:
            output += "<" + str(count) + "> "
        else:
            output += "<" + str(count) + ">"

    output += "\n"
    output += border

    if linux == 1:
        os.system("clear")
    print(output)

    return space_count


# used to park a vehicle within the lot
def enter_vehicle(v_type, plate, row, space):
    global spaces, avail_spaces, total_spaces, rows

    # do not allow a user to park a vehicle with a full lot
    if avail_spaces == 0:
        display_lot()
        print("Error: No Available Spaces")
        time.sleep(2)
        return

    # check if a specified space is already occupied
    if spaces[(int(row) * space_count) + int(space)].is_available():
        display_space_selection(row)
        print("Error: Vehicle Already In Space")
        time.sleep(2)
        return -1

    # check if specified plate number is in the lot
    for uniq in spaces:
        if uniq.is_available():
            if uniq.vehicle_info().get_plate() == plate:
                display_lot()
                print("Error: Vehicle Already In Lot")
                time.sleep(2)
                return

    # add a valid vehicle to the specified space and show the time of entry
    new_vehicle = Vehicle(v_type, plate)
    spaces[(int(row) * space_count) + int(space)].add_vehicle(new_vehicle)
    avail_spaces -= 1
    display_lot()
    print("Vehicle Added to Lot!\n"
          "Time Entered: " + str(time.strftime('%I:%M %p',
                                               time.localtime(new_vehicle.get_entry_time()))))
    time.sleep(2)

    return new_vehicle

```

Def display_space_selection(row) :

- Function to display a specified row with selection numbers.

Def enter_vehicle( )

- Function used to park a vehicle with the available lot/spot. If the spot is empty, then user can park the vehicle there, and do not allow the user to park a vehicle with a full lot.


/Users/c
...ristopheralexander/Desktop/Kuliah/Algorithm and Programming/Final Project/parking_lot/parking_lot.py
parking_lot.py
config.txt

```python
def fare_calculation(vehicle):
    # calculate the number of seconds which have passed since a vehicle was entered into the system
    # if less than one hour has passed, then a minimum fare of one hour is priced
    total_time = time.time() - vehicle.get_entry_time()
    if total_time < 3600:
        hours = 1
    else:
        hours = int(total_time / 3600)+1

    # calculate fare based on vehicle type
    if vehicle.get_type() == 1:
        rate = hours * 3.50
    elif vehicle.get_type() == 2:
        rate = hours * 4.50
    else:
        rate = hours * 2.00

    ret = "Vehicle Removed!\n" \
          "Your Total for " + "{:.2f}".format(hours) + " hours is $" + "{:.2f}".format(rate)

    return ret


# used to removed a vehicle from the lot
def exit_lot(row, space):
    global avail_spaces

    # check if a specified space is occupied
    if not spaces[(int(row) * space_count) + int(space)].is_available():
        display_space_selection(row)
        print("Error: No Vehicle In Space")
        time.sleep(2)
        return

    # if the specified plate number is found within the lot, the vehicle is removed
    removed = spaces[(int(row) * space_count) + int(space)].remove_vehicle()
    avail_spaces += 1

    # calculate fare if a vehicle is removed
    display_lot()
    print(fare_calculation(removed))
    time.sleep(2)


# used to view a currently parked vehicle's information
def view_vehicle(row, space):

    # check if a specified space is occupied
    if not spaces[(int(row) * space_count) + int(space)].is_available():
        display_space_selection(row)
        print("Error: No Vehicle In Space")
        time.sleep(2)

    # collect vehicle information and display to user
    else:
        vehicle = spaces[(int(row) * space_count) + int(space)].vehicle_info()
        display_space_selection(row)
        input("Vehicle Type: " + vehicle.get_type_string() + "\n"
                                                             "Plate Number: " + vehicle.get_plate() + "\n"
                                                                                                      "Entry Time: " + str(
            time.strftime('%m-%d-%Y %I:%M %p',
                          time.localtime(vehicle.get_entry_time()))) + "\n"
                                                                        "\nPress Enter to return to menu")
```

Def fare_calculation(vehicle) :

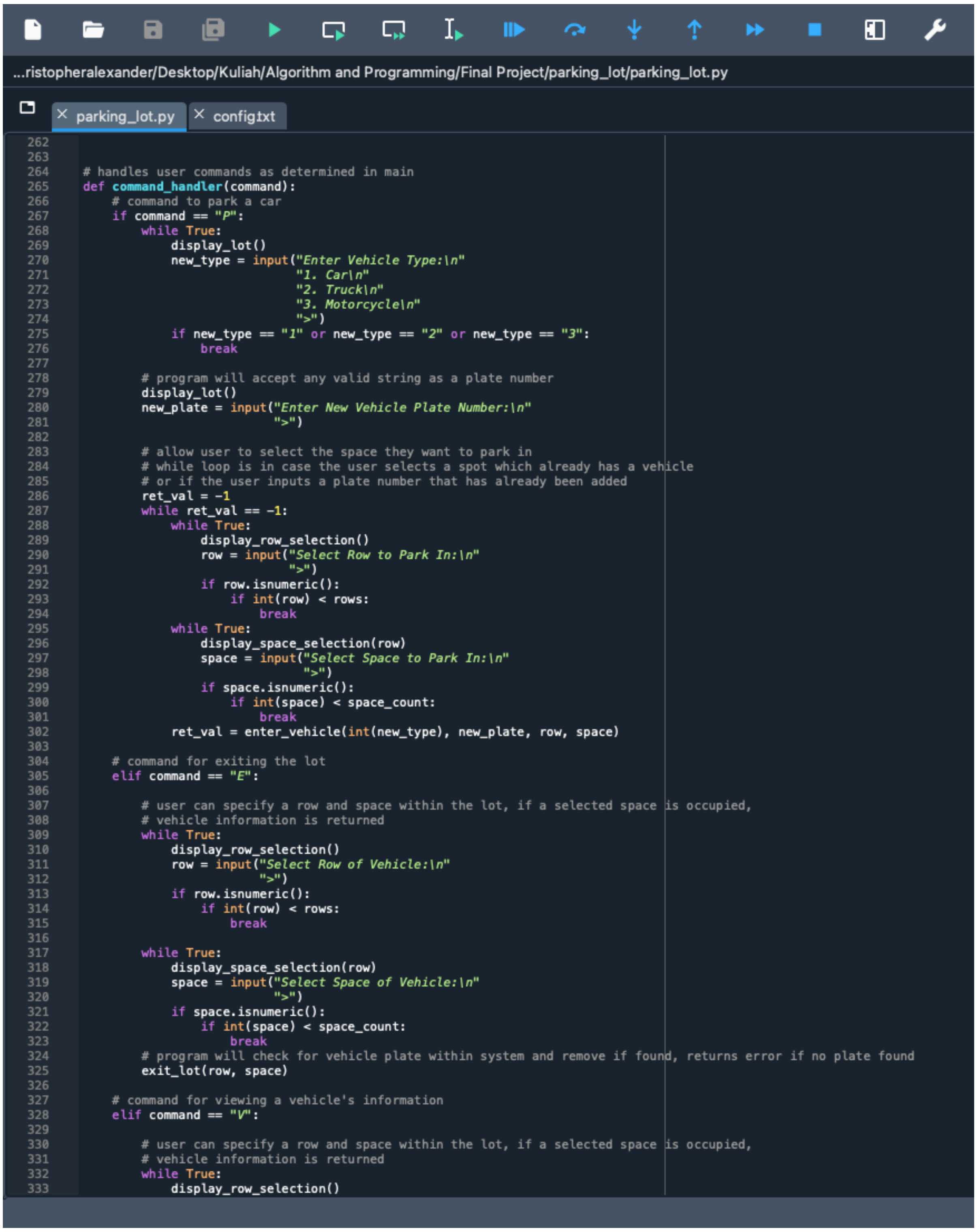
- Function that include some formula to calculate the fare of vehicle that parked on.

Def exit_lot(row, space) :

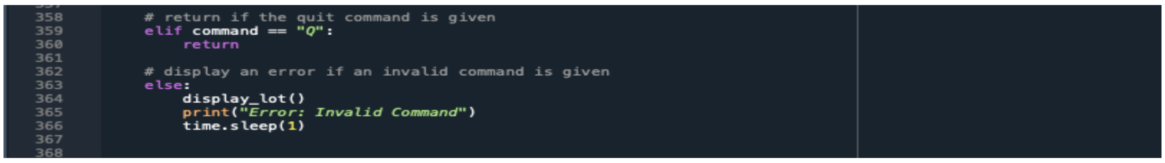
- Function used to removed a vehicle from the parking lot.

Def view_vehicle(row, space) :

- Function used to view a currently parked vehicle information, such as plate number, and time entered.


...ristopheralexander/Desktop/Kuliah/Algorithm and Programming/Final Project/parking_lot/parking_lot.py
parking_lot.py
config.txt

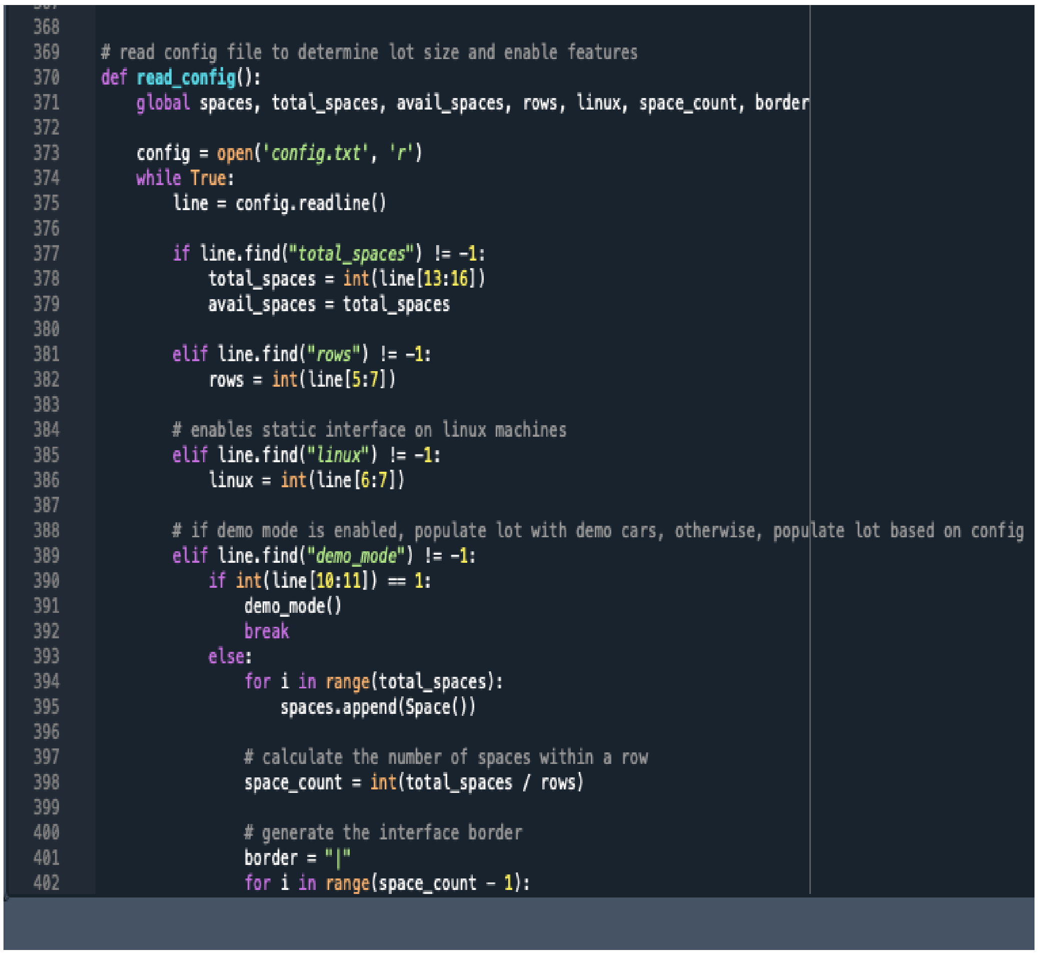
```python


# handles user commands as determined in main
def command_handler(command):
    # command to park a car
    if command == "P":
        while True:
            display_lot()
            new_type = input("Enter Vehicle Type:\n"
                             "1. Car\n"
                             "2. Truck\n"
                             "3. Motorcycle\n"
                             ">")
            if new_type == "1" or new_type == "2" or new_type == "3":
                break

        # program will accept any valid string as a plate number
        display_lot()
        new_plate = input("Enter New Vehicle Plate Number:\n"
                          ">")

        # allow user to select the space they want to park in
        # while loop is in case the user selects a spot which already has a vehicle
        # or if the user inputs a plate number that has already been added
        ret_val = -1
        while ret_val == -1:
            while True:
                display_row_selection()
                row = input("Select Row to Park In:\n"
                            ">")
                if row.isnumeric():
                    if int(row) < rows:
                        break
            while True:
                display_space_selection(row)
                space = input("Select Space to Park In:\n"
                              ">")
                if space.isnumeric():
                    if int(space) < space_count:
                        break
            ret_val = enter_vehicle(int(new_type), new_plate, row, space)

    # command for exiting the lot
    elif command == "E":

        # user can specify a row and space within the lot, if a selected space is occupied,
        # vehicle information is returned
        while True:
            display_row_selection()
            row = input("Select Row of Vehicle:\n"
                        ">")
            if row.isnumeric():
                if int(row) < rows:
                    break

        while True:
            display_space_selection(row)
            space = input("Select Space of Vehicle:\n"
                          ">")
            if space.isnumeric():
                if int(space) < space_count:
                    break
        # program will check for vehicle plate within system and remove if found, returns error if no plate found
        exit_lot(row, space)

    # command for viewing a vehicle's information
    elif command == "V":

        # user can specify a row and space within the lot, if a selected space is occupied,
        # vehicle information is returned
        while True:
            display_row_selection()
```



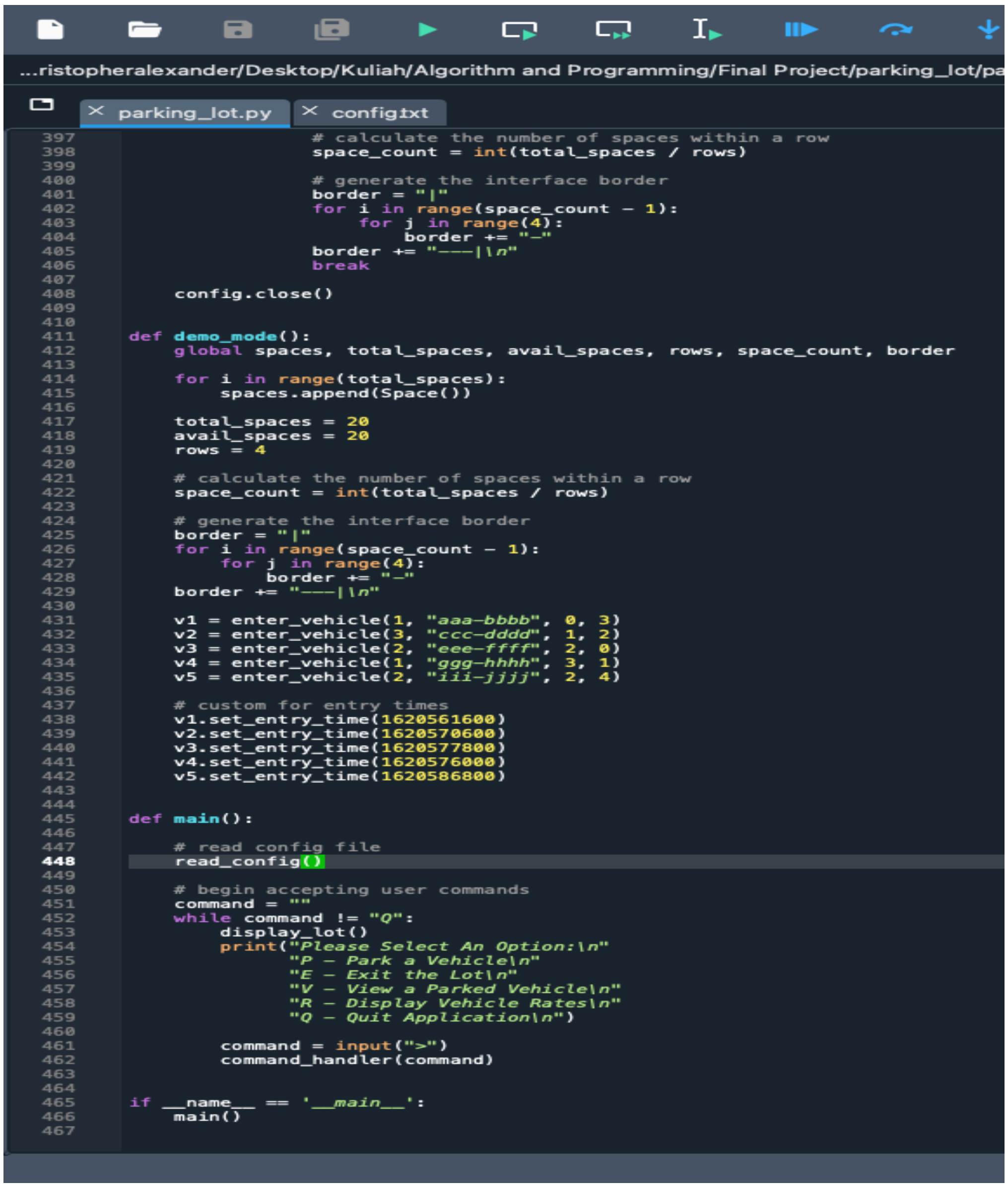
```python
    # return if the quit command is given
    elif command == "Q":
        return

    # display an error if an invalid command is given
    else:
        display_lot()
        print("Error: Invalid Command")
        time.sleep(1)
```

Def command_handler(command) :

- Used for the user as their command handler. This means that this function contain several options that will be used when you want to run this parking lot program. For example there's an option when we can choose which vehicle do we park in there, is it car, motorcycle, or truck.



```python

# read config file to determine lot size and enable features
def read_config():
    global spaces, total_spaces, avail_spaces, rows, linux, space_count, border

    config = open('config.txt', 'r')
    while True:
        line = config.readline()

        if line.find("total_spaces") != -1:
            total_spaces = int(line[13:16])
            avail_spaces = total_spaces

        elif line.find("rows") != -1:
            rows = int(line[5:7])

        # enables static interface on linux machines
        elif line.find("linux") != -1:
            linux = int(line[6:7])

        # if demo mode is enabled, populate lot with demo cars, otherwise, populate lot based on config
        elif line.find("demo_mode") != -1:
            if int(line[10:11]) == 1:
                demo_mode()
                break
            else:
                for i in range(total_spaces):
                    spaces.append(Space())

                # calculate the number of spaces within a row
                space_count = int(total_spaces / rows)

                # generate the interface border
                border = "|"
                for i in range(space_count - 1):
```



...ristopheralexander/Desktop/Kuliah/Algorithm and Programming/Final Project/parking_lot/pa

```python
                # calculate the number of spaces within a row
                space_count = int(total_spaces / rows)

                # generate the interface border
                border = "|"
                for i in range(space_count - 1):
                    for j in range(4):
                        border += "-"
                border += "---|\n"
                break

    config.close()


def demo_mode():
    global spaces, total_spaces, avail_spaces, rows, space_count, border

    for i in range(total_spaces):
        spaces.append(Space())

    total_spaces = 20
    avail_spaces = 20
    rows = 4

    # calculate the number of spaces within a row
    space_count = int(total_spaces / rows)

    # generate the interface border
    border = "|"
    for i in range(space_count - 1):
        for j in range(4):
            border += "-"
    border += "---|\n"

    v1 = enter_vehicle(1, "aaa-bbbb", 0, 3)
    v2 = enter_vehicle(3, "ccc-dddd", 1, 2)
    v3 = enter_vehicle(2, "eee-ffff", 2, 0)
    v4 = enter_vehicle(1, "ggg-hhhh", 3, 1)
    v5 = enter_vehicle(2, "iii-jjjj", 2, 4)

    # custom for entry times
    v1.set_entry_time(1620561600)
    v2.set_entry_time(1620570600)
    v3.set_entry_time(1620577800)
    v4.set_entry_time(1620576000)
    v5.set_entry_time(1620586800)


def main():

    # read config file
    read_config()

    # begin accepting user commands
    command = ""
    while command != "Q":
        display_lot()
        print("Please Select An Option:\n"
              "P - Park a Vehicle\n"
              "E - Exit the Lot\n"
              "V - View a Parked Vehicle\n"
              "R - Display Vehicle Rates\n"
              "Q - Quit Application\n")

        command = input(">")
        command_handler(command)


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

Def read_config( ) :

- Function that will read the config file (config.txt) to determine the lot size and some enable features.

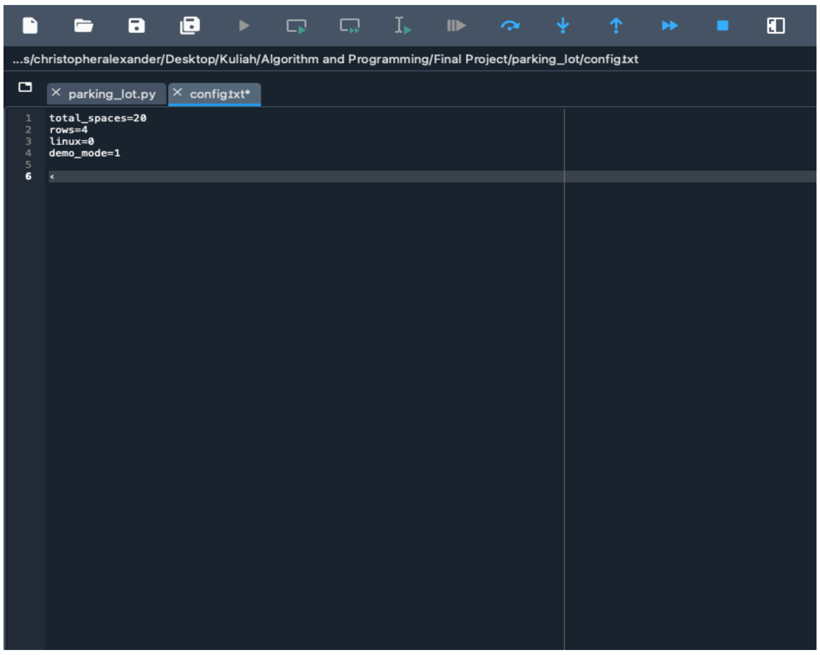
Def demo_mode( ) :

- Function for demo mode, contains information about several examples of parked vehicles and will later show the remaining lots available when the demo vehicle is parked.

Def main( ) :

- Main function.

2.Config.txt



...s/christopheralexander/Desktop/Kuliah/Algorithm and Programming/Final Project/parking_lot/config.txt

```
total_spaces=20
rows=4
linux=0
demo_mode=1

```

Config file which contain some information code that will be used in parking_lot.py

3.References

https://medium.com/@JuanPabloHerrera/create-a-parking-lot-simulation-using-oop-in-python-ff3472554265

https://github.com/apoorva-dave/ParkingLot

https://codereview.stackexchange.com/questions/225354/parking-lot-object-oriented-design-python

Object Oriented Programming in Python #2 | 12 mins | Coding Parking Lot