My Project aims to solve the problem of the Supply Management System. The Shortest explanation of the problem is: My system will handle the flow of goods from suppliers to customers. It will support the ordering, movement, and storage of goods and materials. The registered customer is placing an order for materials, specifies the quantity where and when it should be delivered. My system will check all the warehouses to see if such materials are available, choose the closest to the delivery destination warehouse and confirm the order. The customer can always check the state of the order.

I used Python and SQ Lite system for my project. And I used Princeton University's Standard Library which includes basic color and drawing libraries.

This part of the codes are actually solution for the problem which is writing up :

# Iterate over all warehouses

for i in warehouses:

# Iterate over the products in stock on the current warehouse

for j in i.in\_stock:

# If the desired product is in stock

if str(j[0]) == product\_str and j[1] >= qty:

# Select the product if it is not selected before

if product == None:

product = j[0]

# Add the warehouse to the list of warehouses that have stocks

has\_stock.append(i)

# Initialize the minimum distance

min\_dist = float("inf")

# Initliaze the minimum-distanced warehouse

min\_dist\_wh = None

# Iterate over all the warehouses that have stocks

for i in has\_stock:

# If the distance between store and the current warehouse is less than the last minimum disttance

if abs(((store.x - i.x)\*\*2 + (store.y - i.y)\*\*2)\*\*(1/2)) < min\_dist:

# Update the distance and the minimum-distanced warehouse

min\_dist = abs(((store.x - i.x)\*\*2 + (store.y - i.y)\*\*2)\*\*(1/2))

min\_dist\_wh = i

These codes are a code that automatically selects the warehouse closest to the selected market and containing sufficient items.

These are the Order panel's codes and explanations:

from tkinter import \*

from tkinter import messagebox

from lib.product import Product

from lib.store import Store

from lib.warehouse import Warehouse

import os

import sqlite3

DIR = os.path.dirname(os.path.abspath(\_\_file\_\_)) # Get the directory

DB\_DIR = DIR + "/db" # Get the database directory

DB\_PATH = DB\_DIR + "/order.db" # Get the database file

# SQL queries

SHOW\_TABLES = "SELECT name FROM sqlite\_master WHERE type='table'"

CREATE\_PRODUCT = "CREATE TABLE Product(name varchar(20), brand varchar(20), price int, PRIMARY KEY(brand, name))"

CREATE\_STORE = "CREATE TABLE Store(id varchar(4) PRIMARY KEY, name varchar(20), address varchar(10) UNIQUE)"

CREATE\_WAREHOUSE = "CREATE TABLE Warehouse(id varchar(4) PRIMARY KEY, name varchar(20), address varchar(10) UNIQUE)"

CREATE\_SELLS = "CREATE TABLE Sells(product\_name varchar(20), product\_brand varchar(20), store\_id varchar(4), quantity int, PRIMARY KEY(product\_brand, product\_name, store\_id), FOREIGN KEY(product\_brand, product\_name) REFERENCES Product(brand, name), FOREIGN KEY(store\_id) REFERENCES Store(id))"

CREATE\_OWNS = "CREATE TABLE Owns(product\_name varchar(20), product\_brand varchar(20), warehouse\_id varchar(4), quantity int, PRIMARY KEY(product\_brand, product\_name, warehouse\_id), FOREIGN KEY(product\_brand, product\_name) REFERENCES Product(brand, name), FOREIGN KEY(warehouse\_id) REFERENCES Warehouse(id))"

SELECT\_PRODUCT = "SELECT \* FROM Product"

SELECT\_STORE = "SELECT \* FROM Store"

SELECT\_WAREHOUSE = "SELECT \* FROM Warehouse"

SELECT\_OWNS = "SELECT product\_brand, product\_name, quantity FROM Owns WHERE warehouse\_id=?"

SELECT\_SELLS = "SELECT product\_brand, product\_name, quantity FROM Sells WHERE store\_id=?"

# Method for initializing the database

def init\_db():

global cursor

global sql

# If the database directory does not exist, create it

if not os.path.exists(DB\_DIR):

os.mkdir(DB\_DIR)

# If the database file does not exist, create it

if not os.path.exists(DB\_PATH):

with open(DB\_PATH, 'w'): pass

# Create database instance with the database file

sql = sqlite3.connect(DB\_PATH)

# Create the SQL cursor

cursor = sql.cursor()

# Get the database tables

cursor.execute(SHOW\_TABLES)

tables = cursor.fetchall()

# If the database tables are not valid, drop the database and recreate all the tables

if len(tables) != 5 or ('Store',) not in tables or ('Product',) not in tables or ('Warehouse',) not in tables or ('Sells',) not in tables or ('Owns',) not in tables:

with open(DB\_PATH, 'w'): pass # Overwrite the db file

cursor.execute(CREATE\_STORE)

cursor.execute(CREATE\_WAREHOUSE)

cursor.execute(CREATE\_PRODUCT)

cursor.execute(CREATE\_SELLS)

cursor.execute(CREATE\_OWNS)

sql.commit()

# Method for initializing the screen

def init\_screen():

global stores

global products

# Initiliaze the main screen

main\_screen = Tk()

main\_screen.resizable(False, False)

main\_screen.title("Order")

# Initialize the left frame

left\_frame = Frame(main\_screen, borderwidth=5, relief="groove")

left\_frame.grid(row=0,column=0,padx=10,pady=10)

# Initialize the rigth frame

right\_frame = Frame(main\_screen, borderwidth=5, relief="groove")

right\_frame.grid(row=0,column=1,padx=10,pady=10)

global store\_stock\_var

global product\_var

global quantity\_var

global store\_var

# Variable for product dropdown menu selection

product\_var = StringVar()

# If the value of product\_var changes, run variable check

product\_var.trace("w", lambda name, index, mode, sv=product\_var: var\_check())

# Variable for quantity entry

quantity\_var = IntVar()

# If the value of quantity\_var changes, run variable check

quantity\_var.trace("w", lambda name, index, mode, sv=quantity\_var: var\_check())

# Variable for store dropdown menu selection

store\_var = StringVar()

# If the value of store\_var changes, run variable check

store\_var.trace("w", lambda name, index, mode, sv=store\_var: var\_check())

# Variable for store stock dropdown menu selection

store\_stock\_var = StringVar()

# If the value of store\_stock\_var changes, change the contents of listbox

store\_stock\_var.trace("w", lambda name, index, mode, sv=store\_stock\_var: update\_stocks())

# "Product to be delivered:" label

product\_label = Label(left\_frame, text="Product to be delivered:")

product\_label.grid(row=0,column=0,sticky=N+S+W,padx=10,pady=3)

# Product dropdown menu

global product\_dd

product\_dd = OptionMenu(left\_frame, product\_var, \*products)

product\_dd.grid(row=1,column=0,sticky=NSEW,padx=10,pady=3)

# "Quantity:" label

quantity\_label = Label(left\_frame, text="Quantity:")

quantity\_label.grid(row=2,column=0,sticky=N+S+W,padx=10,pady=3)

# Quantity entry

global quantity\_entry

quantity\_entry = Entry(left\_frame, textvariable=quantity\_var,width=40)

quantity\_entry.grid(row=3,column=0,sticky=NSEW,padx=10,pady=3)

# "Store to be delivered to:" label

store\_label = Label(left\_frame, text="Store to be delivered to:")

store\_label.grid(row=4,column=0,sticky=N+S+W,padx=10,pady=3)

# Store dropdown menu

global store\_dd

store\_dd = OptionMenu(left\_frame, store\_var, \*stores)

store\_dd.grid(row=5,column=0,sticky=NSEW,padx=10,pady=3)

# "Order Now" button

global order\_button

order\_button = Button(left\_frame, text="Order Now!",command=deliver,state=DISABLED)

order\_button.grid(row=6,column=0,sticky=NSEW,padx=10,pady=11)

# "Check the stocks of:" label

store\_stock\_label = Label(right\_frame, text="Check the stocks of:")

store\_stock\_label.grid(row=0,column=0,sticky=N+S+W,padx=10,pady=3)

# Store stocks dropdown menu

global store\_stock\_dd

store\_stock\_dd = OptionMenu(right\_frame, store\_stock\_var, \*stores)

store\_stock\_dd.grid(row=1,column=0,sticky=NSEW,padx=10,pady=3)

# Stock listbox

global stocks

stocks = Listbox(right\_frame, exportselection=False,height=9,width=40,state=DISABLED)

stocks.grid(row=2,column=0,rowspan=5,sticky=NSEW,padx=10,pady=8)

# Start the main screen loop

main\_screen.mainloop()

# Method for updating the listbox to ch<nge its contents based on he

def update\_stocks():

global store\_stock\_var

global stocks

global stores

# Get the selected store

store\_name = store\_stock\_var.get()

# Enable the listbox and delete its contents

stocks.configure(state=NORMAL)

stocks.delete(0, END)

# If nothing is selected, disable the listbox again and return

if store\_name == "":

stocks.configure(state=DISABLED)

return

# Iterate over all stores

for i in stores:

# If the selected store is the current one

if str(i) == store\_name:

# Add all the stocks to the listbox

for j in i.in\_stock:

stocks.insert(END, "{}, Qty: {}".format(j[0], j[1]))

# Disable the listbox again and return

stocks.configure(state=DISABLED)

return

# Method for checking the variables for being valid

def var\_check():

global order\_button

global product\_var

global quantity\_var

global store\_var

# Try to get the quantity, if the entered value is not integer, it will raise an exception

try:

qty = quantity\_var.get()

# If an exception occurs, disable the button and return

except:

order\_button.configure(state=DISABLED)

return

# If the variables are empty or not valid, disable the button

if qty <= 0 or store\_var.get() == "" or product\_var.get() == "":

order\_button.configure(state=DISABLED)

# Otherwise, enable the button

else:

order\_button.configure(state=NORMAL)

# Method for delivering a product to a store

def deliver():

global product\_var

global quantity\_var

global store\_var

global store\_stock\_var

global products

global stores

global warehouses

global product\_dd

global quantity\_entry

global store\_dd

global store\_stock\_dd

# Disable all the input elements

product\_dd.configure(state=DISABLED)

quantity\_entry.configure(state=DISABLED)

store\_dd.configure(state=DISABLED)

store\_stock\_dd.configure(state=DISABLED)

# Get the input values

qty = quantity\_var.get()

product\_str = product\_var.get()

store\_str = store\_var.get()

# Select the store

for i in stores:

if str(i) == store\_str:

store = i

break

# List to hold all the warehouses that have the desired amount of stocks of the product

has\_stock = []

# For holding the desired product

product = None

# Iterate over all warehouses

for i in warehouses:

# Iterate over the products in stock on the current warehouse

for j in i.in\_stock:

# If the desired product is in stock

if str(j[0]) == product\_str and j[1] >= qty:

# Select the product if it is not selected before

if product == None:

product = j[0]

# Add the warehouse to the list of warehouses that have stocks

has\_stock.append(i)

# If there is no warehouse that has the desired amount of stocks, show an error message, reset and enable all the input elements, and return

if len(has\_stock) == 0:

messagebox.showerror("Out of Stock", "There is no warehouse with such a stock of the desired product.")

product\_dd.configure(state=NORMAL)

quantity\_entry.configure(state=NORMAL)

store\_dd.configure(state=NORMAL)

store\_stock\_dd.configure(state=NORMAL)

product\_var.set("")

store\_var.set("")

quantity\_var.set(0)

return

# Initialize the minimum distance

min\_dist = float("inf")

# Initliaze the minimum-distanced warehouse

min\_dist\_wh = None

# Iterate over all the warehouses that have stocks

for i in has\_stock:

# If the distance between store and the current warehouse is less than the last minimum disttance

if abs(((store.x - i.x)\*\*2 + (store.y - i.y)\*\*2)\*\*(1/2)) < min\_dist:

# Update the distance and the minimum-distanced warehouse

min\_dist = abs(((store.x - i.x)\*\*2 + (store.y - i.y)\*\*2)\*\*(1/2))

min\_dist\_wh = i

# Deliver the desired amount of stocks of the product from the minimum-distanced warehouse to the selected store

min\_dist\_wh.deliver(store, product, qty, stores, warehouses, cursor, sql)

# Select the store to the stock check

store\_stock\_var.set(store\_str)

# Update the listbox

update\_stocks()

# Enable and reset all the input elements

product\_dd.configure(state=NORMAL)

quantity\_entry.configure(state=NORMAL)

store\_dd.configure(state=NORMAL)

store\_stock\_dd.configure(state=NORMAL)

product\_var.set("")

store\_var.set("")

quantity\_var.set(0)

# Method for getting values from database

def get\_from\_db():

global sql

global cursor

global products

global warehouses

global stores

# Initialize the lists

products = []

warehouses = []

stores = []

# Get all the products fro database

cursor.execute(SELECT\_PRODUCT)

pds = cursor.fetchall()

# Add the products to the products list

for i in pds:

products.append(Product(i[0], i[1], i[2]))

# If there is no product, add an empty string to the list

if len(products) == 0:

products.append("")

# Get all the stores from database

cursor.execute(SELECT\_STORE)

sts = cursor.fetchall()

# Iterate over all stores in database

for i in sts:

# Get all the products that the store sells

cursor.execute(SELECT\_SELLS, (i[0],))

pds\_stock = cursor.fetchall()

# Create a list to hold the products in stock

stock = []

# Iterate over all products that the store sells on database

for j in pds\_stock:

# Find the product

for k in products:

if j[0] == k.brand and j[1] == k.name:

# Add the product and its quantity to the store stock list

stock.append([k, j[2]])

break

# Get the x and y coordinates of the store from database

x = int(i[2].lstrip("(").rstrip(")").split(",")[0])

y = int(i[2].lstrip("(").rstrip(")").split(",")[1])

# Add the store with all its values to the stores list

stores.append(Store(x, y, i[1], stock, i[0]))

# If there is no store, add an empty string to the list

if len(stores) == 0:

stores.append("")

# Get all the warehouses from database

cursor.execute(SELECT\_WAREHOUSE)

whs = cursor.fetchall()

# Iterate over all warehouses

for i in whs:

# Get all the products that the warehouse owns

cursor.execute(SELECT\_OWNS, (i[0],))

pds\_stock = cursor.fetchall()

# Create a list to hold the products in stock

stock = []

# Iterate over all products that the warehouse owns on database

for j in pds\_stock:

# Find the product

for k in products:

# Add the product and its quantity to the warehouse stock list

if j[0] == k.brand and j[1] == k.name:

stock.append([k, j[2]])

break

# Get the x and y coordinates of the warehouse from database

x = int(i[2].lstrip("(").rstrip(")").split(",")[0])

y = int(i[2].lstrip("(").rstrip(")").split(",")[1])

# Add the warehouse with all its values to the warehouses list

warehouses.append(Warehouse(x, y, i[1], stock, i[0]))

# If there is no warehouse, add an empty string to the list

if len(warehouses) == 0:

warehouses.append("")

# If the script run directly

if \_\_name\_\_ == '\_\_main\_\_':

init\_db() # Initlialize database

get\_from\_db() # Get values from database

init\_screen() # Initialize the GUI

Admin panel:

from random import randint

import sqlite3

import os

import sys

DIR = os.path.dirname(os.path.abspath(\_\_file\_\_)) # Get the directory

DB\_DIR = DIR + "/db" # Get the database directory

DB\_PATH = DB\_DIR + "/order.db" # Get the database file

# SQL queries

SHOW\_TABLES = "SELECT name FROM sqlite\_master WHERE type='table'"

CREATE\_PRODUCT = "CREATE TABLE Product(name varchar(20), brand varchar(20), price int, PRIMARY KEY(brand, name))"

CREATE\_STORE = "CREATE TABLE Store(id varchar(4) PRIMARY KEY, name varchar(20), address varchar(10) UNIQUE)"

CREATE\_WAREHOUSE = "CREATE TABLE Warehouse(id varchar(4) PRIMARY KEY, name varchar(20), address varchar(10) UNIQUE)"

CREATE\_SELLS = "CREATE TABLE Sells(product\_name varchar(20), product\_brand varchar(20), store\_id varchar(4), quantity int, PRIMARY KEY(product\_brand, product\_name, store\_id), FOREIGN KEY(product\_brand, product\_name) REFERENCES Product(brand, name), FOREIGN KEY(store\_id) REFERENCES Store(id))"

CREATE\_OWNS = "CREATE TABLE Owns(product\_name varchar(20), product\_brand varchar(20), warehouse\_id varchar(4), quantity int, PRIMARY KEY(product\_brand, product\_name, warehouse\_id), FOREIGN KEY(product\_brand, product\_name) REFERENCES Product(brand, name), FOREIGN KEY(warehouse\_id) REFERENCES Warehouse(id))"

SELECT\_PRODUCT = "SELECT \* FROM Product"

SELECT\_STORE = "SELECT \* FROM Store"

SELECT\_WAREHOUSE = "SELECT \* FROM Warehouse"

SELECT\_OWNS = "SELECT product\_brand, product\_name, quantity FROM Owns WHERE warehouse\_id=?"

SELECT\_SELLS = "SELECT product\_brand, product\_name, quantity FROM Sells WHERE store\_id=?"

ADD\_WAREHOUSE = "INSERT INTO Warehouse VALUES(?, ?, ?)"

DELETE\_WAREHOUSE = "DELETE FROM Warehouse WHERE id=?"

ADD\_OWNS = "INSERT INTO Owns VALUES(?, ?, ?, ?)"

UPDATE\_OWNS = "UPDATE Owns SET quantity=? WHERE (product\_name, product\_brand, warehouse\_id) = (?, ?, ?)"

DELETE\_OWNS = "DELETE FROM Owns WHERE (product\_name, product\_brand, warehouse\_id) = (?, ?, ?)"

DELETE\_ALL\_OWNS = "DELETE FROM Owns WHERE warehouse\_id=?"

ADD\_STORE = "INSERT INTO Store VALUES(?, ?, ?)"

DELETE\_STORE = "DELETE FROM Store WHERE id=?"

ADD\_SELLS = "INSERT INTO SELLS VALUES(?, ?, ?, ?)"

UPDATE\_SELLS = "UPDATE Sells SET quantity=? WHERE (product\_name, product\_brand, store\_id) = (?, ?, ?)"

DELETE\_SELLS = "DELETE FROM Sells WHERE (product\_name, product\_brand, store\_id) = (?, ?, ?)"

DELETE\_ALL\_SELLS = "DELETE FROM Sells WHERE store\_id=?"

ADD\_PRODUCT = "INSERT INTO Product VALUES (?, ?, ?)"

DELETE\_PRODUCT = "DELETE FROM Product WHERE (name, brand) = (?, ?)"

DELETE\_PD\_FR\_SELLS = "DELETE FROM Sells WHERE product\_name=? AND product\_brand=?"

DELETE\_PD\_FR\_OWNS = "DELETE FROM Owns WHERE product\_name=? AND product\_brand=?"

UPDATE\_PRICE = "UPDATE Product SET price=? WHERE (name, brand) = (?, ?)"

# If the database directory does not exist, create it

if not os.path.exists(DB\_DIR):

os.mkdir(DB\_DIR)

# If the database file does not exist, create it

if not os.path.exists(DB\_PATH):

with open(DB\_PATH, 'w'):

pass

# Create database instance with the database file

sql = sqlite3.connect(DB\_PATH)

# Create the SQL cursor

cursor = sql.cursor()

# Get the database tables

cursor.execute(SHOW\_TABLES)

tables = cursor.fetchall()

# If the database tables are not valid, drop the database and recreate all the tables

if len(tables) != 5 or ('Store',) not in tables or ('Product',) not in tables or ('Warehouse',) not in tables or ('Sells',) not in tables or ('Owns',) not in tables:

with open(DB\_PATH, 'w'): pass # Overwrite the db file

cursor.execute(CREATE\_STORE)

cursor.execute(CREATE\_WAREHOUSE)

cursor.execute(CREATE\_PRODUCT)

cursor.execute(CREATE\_SELLS)

cursor.execute(CREATE\_OWNS)

sql.commit()

# Main program loop

while True:

# Print the option table

print("\nWelcome to Order Admin Panel!")

print("----------------------------------")

print(" 1) List Warehouses")

print(" 2) List Stocks of A Warehouse")

print(" 3) Add Warehouse")

print(" 4) Delete Warehouse")

print(" 5) Add Product to Warehouse Stock")

print(" 6) Remove Product from Warehouse Stock")

print(" 7) List Stores")

print(" 8) List Stocks of A Store")

print(" 9) Add Store")

print("10) Delete Store")

print("11) Add Product to Store Stock")

print("12) Remove Product from Store Stock")

print("13) List Products")

print("14) Add Product")

print("15) Delete Product")

print("16) Update Product Price")

print("17) Exit\n")

# Loop until the user enters a valid option

while True:

# Get the input

raw\_input = input("Please select a command: ")

# If it is not integer, continue

try:

inp = int(raw\_input)

except:

print("Invalid option.")

continue

# If the given integer not in the boundaries of the options, continue

if inp < 1 or inp > 17:

print("Invalid option.")

continue

# Break if it is valid

break

# Option 1 / List Warehouses

if inp == 1:

# Get warehouses

cursor.execute(SELECT\_WAREHOUSE)

whs = cursor.fetchall()

# Print warehouses

print()

for i in whs:

print("{} #{}, (x,y)={}".format(i[1], i[0], i[2]))

continue

# Option 2 / List Stocks of A Warehouse

if inp == 2:

# Print warehouses label

print()

print("Warehouses:")

cursor.execute(SELECT\_WAREHOUSE)

whs = cursor.fetchall()

# Gather warehouse IDs in a list and print warehouses

ids = []

for i in whs:

ids.append(i[0])

print("{} #{}".format(i[1], i[0], i[2]))

print()

# Loop until the user enters a valid input

while True:

# Get an ID from the user

id = input("Please enter the warehouse ID: ").strip()

# If the given ID not in the IDs list, continue

if id not in ids:

print("Incorrect ID.")

continue

# Break if the ID is valid

break

# Get the products that the warehouse with the ID owns

cursor.execute(SELECT\_OWNS, (id,))

pds = cursor.fetchall()

# Print products

print()

for i in pds:

print("Brand: {}, Name: {}, Qty: {}".format(i[0], i[1], i[2]))

continue

# Option 3 / Add Warehouse

if inp == 3:

# Get warehouses

cursor.execute(SELECT\_WAREHOUSE)

whs = cursor.fetchall()

# List for holding used addresses

used\_addresses = []

# List for holding used IDs

ids = []

# Gather addresses and IDs of warehouses

for i in whs:

ids.append(i[0])

used\_addresses.append(i[2])

# Get stores

cursor.execute(SELECT\_STORE)

sts = cursor.fetchall()

# Gather addresses and IDs of stores

for i in sts:

ids.append(i[0])

used\_addresses.append(i[2])

# Get a name

print()

name = input("Name: ").strip()

# Print used addresses

print("Used Addresses: {}".format(", ".join(used\_addresses)))

# Loop until the user enters a valid input

while True:

# Loop until the user enters a valid x

while True:

# Get x as string

x\_str = input("x: ").strip()

# Try to convert it to integer, continue if cannot convert

try:

x = int(x\_str)

except:

print("Invalid x-coordinate.")

continue

# If the given x is not in bounds, continue

if x < 0 or x > 9:

print("Invalid x-coordinate. Must be between 0 and 9.")

continue

# Break if the x is valid

break

# Loop until the user enter a valid y

while True:

# Get the y as string

y\_str = input("y: ").strip()

# Try to convert it to integer, continue if cannot convert

try:

y = int(y\_str)

except:

print("Invalid y-coordinate.")

continue

# If the given y is not in bounds, continue

if y < 0 or y > 9:

print("Invalid y-coordinate. Must be between 0 and 9.")

continue

# Break if the y is valid

break

# Check if there is a warehouse with the given x and y addresses, it there is, continue

found = False

for i in whs:

if i[2] == "({},{})".format(x,y):

print("Invalid x and y. Both must be unique.")

found = True

break

if found:

continue

# Check if there is a store with the given x and y addresses, it there is, continue

for i in sts:

if i[2] == "({},{})".format(x,y):

print("Invalid x and y. Both must be unique.")

found = True

break

if found:

continue

# Break if the given address is valid

break

# Generate an unused ID

while True:

id = "{:04d}".format(randint(0,9999))

if id not in ids:

break

# Insert warehouse into the database

cursor.execute(ADD\_WAREHOUSE, (id, name, "({},{})".format(x,y)))

sql.commit()

# Print success message

print()

print("{} #{} is created!".format(name, id))

continue

# Option 4 / Delete Warehouse

if inp == 4:

# Print warehouses label

print()

print("Warehouses:")

# Get warehouses

cursor.execute(SELECT\_WAREHOUSE)

whs = cursor.fetchall()

# Gather warehouse IDs and print warehouses

ids = []

for i in whs:

ids.append(i[0])

print("{} #{}".format(i[1], i[0], i[2]))

print()

# Loop until the user enters a valid warehouse ID

while True:

# Get ID

id = input("Enter warehouse ID: ").strip()

# If the ID is not in IDs, continue

if id not in ids:

print("Invalid ID.")

continue

# Break if the ID is valid

break

# Delete warehouse stocks from database

cursor.execute(DELETE\_ALL\_OWNS, (id,))

sql.commit()

# Delete warehouse from database

cursor.execute(DELETE\_WAREHOUSE, (id,))

sql.commit()

# Print success message

print()

print("Warehouse #{} is deleted!".format(id))

continue

# Option 5 / Add Product to Warehouse Stock

if inp == 5:

# Get all products

cursor.execute(SELECT\_PRODUCT)

pds = cursor.fetchall()

# List to hold product names and brands as tuples

pd\_tuples = []

# Print products header

print()

print("Products:")

# Add all the products as tuples to the list and print products

for i in pds:

pd\_tuples.append((i[1], i[0]))

print("Brand: {}, Name: {}, ${}".format(i[1], i[0], i[2]))

print()

# Loop until the user enters a valid product brand and name

while True:

# Get brand and name

brand = input("Brand: ").strip()

name = input("Name: ").strip()

# If the brand and name not in tuples, continue

if (brand,name) not in pd\_tuples:

print("Invalid product.")

continue

# Break if the product is valid

break

# Print warehouses header

print()

print("Warehouses:")

# Get warehouses

cursor.execute(SELECT\_WAREHOUSE)

whs = cursor.fetchall()

# Gather warehouse IDs in a list and print warehouses

ids = []

for i in whs:

ids.append(i[0])

print("{} #{}".format(i[1], i[0], i[2]))

print()

# Loop until the user enters a valid warehouse ID

while True:

# Get ID

id = input("Warehouse ID: ").strip()

# If the ID is not in IDs list, continue

if id not in ids:

print("Invalid ID.")

continue

# Break if the ID is valid

break

# Loop until the user enters a valid quantity

while True:

# Get quantity as string

qty\_str = input("Quantity: ").strip()

# Try to convert it to int, continue if cannot convert

try:

qty = int(qty\_str)

except:

print("Invalid quantity.")

continue

# If quantity is below 1, continue

if qty < 1:

print("Invalid quantity. Cannot be less than 1.")

continue

# Break if the quantity is valid

break

# Get all the products that are already in the stock of the selected warehouse

cursor.execute(SELECT\_OWNS, (id,))

owns = cursor.fetchall()

# Variable for holding if the selected product is already in the stock

found = False

# Iterate over all the products in stock

for i in owns:

# If the product is in stock

if i[0] == brand and i[1] == name:

# Compute the new quantity

new\_qty = int(i[2]) + qty

# Update the quantity

cursor.execute(UPDATE\_OWNS, (new\_qty, name, brand, id))

sql.commit()

# Print success message

print()

print("{} stocks of {} {} is added to the stock of Warehouse #{}!".format(qty, brand, name, id))

# Set variable as found

found = True

break

# If the product is found, skip the rest of the block

if found:

continue

# Add new product to the stock

cursor.execute(ADD\_OWNS, (name, brand, id, qty))

sql.commit()

# Print success message

print()

print("{} stocks of {} {} is added to the stock of Warehouse #{}!".format(qty, brand, name, id))

continue

# Option 6 / Remove Product from Warehouse Stock

if inp == 6:

# Print warehouses header

print()

print("Warehouses:")

# Get warehouses

cursor.execute(SELECT\_WAREHOUSE)

whs = cursor.fetchall()

# Gather warehouse IDs in a list and print warehouses

ids = []

for i in whs:

ids.append(i[0])

print("{} #{}".format(i[1], i[0], i[2]))

print()

# Loop until the user enters a valid ID

while True:

# Get ID

id = input("Warehouse ID: ").strip()

# If the ID is not in the IDs list, continue

if id not in ids:

print("Invalid ID.")

continue

# Break if the ID is valid

break

# Get all the products in the selected warehouses's stock

cursor.execute(SELECT\_OWNS, (id,))

in\_stock = cursor.fetchall()

# Print items in stock header

print()

print("Items in stock:")

# Gather all the products as brand and name tuples, gather the quantities as values to the tuple keys, and print products

pd\_tuples = []

qtys = {}

for i in in\_stock:

pd\_tuples.append((i[0], i[1]))

qtys["({}, {})".format(i[0], i[1])] = int(i[2])

print("Brand: {}, Name: {}, Qty: {}".format(i[0], i[1], i[2]))

print()

# Loop until the user enters a valid product brand and name

while True:

# Get brand and name

brand = input("Brand: ").strip()

name = input("Name: ").strip()

# If the brand and name not in product tuples, continue

if (brand,name) not in pd\_tuples:

print("Invalid product.")

continue

# Break if the product is valid

break

# Loop until the user enters a valid quantity

while True:

# Get quantity as string

qty\_str = input("Quantity: ").strip()

# Try to convert it to int, continue if cannot convert

try:

qty = int(qty\_str)

except:

print("Invalid quantity.")

continue

# If the given quantity is less than 0 or more than the total quantity of the product, continue

if qty < 1 or qty > qtys["({}, {})".format(brand, name)]:

print("Invalid quantity. Cannot be less than 1 or more than total quantity.")

continue

# Break if the quantity is valid

break

# If the total quantity - given quantity equals 0, delete the product from stock

if qtys["({}, {})".format(brand, name)] - qty == 0:

cursor.execute(DELETE\_OWNS, (name, brand, id))

sql.commit()

# If there is still an amount quantity, update the quantity

else:

cursor.execute(UPDATE\_OWNS, (qtys["({}, {})".format(brand, name)] - qty, name, brand, id))

sql.commit()

# Print success message

print()

print("{} stocks of {} {} is deleted from Warehouse #{}!".format(qty, brand, name, id))

continue

# Option 7 / List Stores

if inp == 7:

# Get stores

cursor.execute(SELECT\_STORE)

sts = cursor.fetchall()

# Print stores

print()

for i in sts:

print("{} #{}, (x,y)={}".format(i[1], i[0], i[2]))

continue

# Option 8 / List Stocks of A Store

if inp == 8:

# Print stores label

print()

print("Stores:")

cursor.execute(SELECT\_STORE)

sts = cursor.fetchall()

# Gather store IDs in a list and print stores

ids = []

for i in sts:

ids.append(i[0])

print("{} #{}".format(i[1], i[0], i[2]))

print()

# Loop until the user enters a valid input

while True:

# Get an ID from the user

id = input("Please enter the store ID: ").strip()

# If the given ID not in the IDs list, continue

if id not in ids:

print("Incorrect ID.")

continue

# Break if the ID is valid

break

# Get the products that the store with the ID sells

cursor.execute(SELECT\_SELLS, (id,))

pds = cursor.fetchall()

# Print products

print()

for i in pds:

print("Brand: {}, Name: {}, Qty: {}".format(i[0], i[1], i[2]))

continue

# Option 9 / Add Store

if inp == 9:

# Get warehouses

cursor.execute(SELECT\_WAREHOUSE)

whs = cursor.fetchall()

# List for holding used addresses

used\_addresses = []

# List for holding used IDs

ids = []

# Gather addresses and IDs of warehouses

for i in whs:

ids.append(i[0])

used\_addresses.append(i[2])

# Get stores

cursor.execute(SELECT\_STORE)

sts = cursor.fetchall()

# Gather addresses and IDs of stores

for i in sts:

ids.append(i[0])

used\_addresses.append(i[2])

# Get a name

print()

name = input("Name: ").strip()

# Print used addresses

print("Used Addresses: {}".format(", ".join(used\_addresses)))

# Loop until the user enters a valid input

while True:

# Loop until the user enters a valid x

while True:

# Get x as string

x\_str = input("x: ").strip()

# Try to convert it to integer, continue if cannot convert

try:

x = int(x\_str)

except:

print("Invalid x-coordinate.")

continue

# If the given x is not in bounds, continue

if x < 0 or x > 9:

print("Invalid x-coordinate. Must be between 0 and 9.")

continue

# Break if the x is valid

break

# Loop until the user enter a valid y

while True:

# Get the y as string

y\_str = input("y: ").strip()

# Try to convert it to integer, continue if cannot convert

try:

y = int(y\_str)

except:

print("Invalid y-coordinate.")

continue

# If the given y is not in bounds, continue

if y < 0 or y > 9:

print("Invalid y-coordinate. Must be between 0 and 9.")

continue

# Break if the y is valid

break

# Check if there is a warehouse with the given x and y addresses, it there is, continue

found = False

for i in whs:

if i[2] == "({},{})".format(x,y):

print("Invalid x and y. Both must be unique.")

found = True

break

if found:

continue

# Check if there is a store with the given x and y addresses, it there is, continue

for i in sts:

if i[2] == "({},{})".format(x,y):

print("Invalid x and y. Both must be unique.")

found = True

break

if found:

continue

# Break if the given address is valid

break

# Generate an unused ID

while True:

id = "{:04d}".format(randint(0,9999))

if id not in ids:

break

# Insert store into the database

cursor.execute(ADD\_STORE, (id, name, "({},{})".format(x,y)))

sql.commit()

# Print success message

print()

print("{} #{} is created!".format(name, id))

continue

# Option 10 / Delete Store

if inp == 10:

# Print stores label

print()

print("Stores:")

# Get stores

cursor.execute(SELECT\_STORE)

sts = cursor.fetchall()

# Gather stores IDs and print stores

ids = []

for i in sts:

ids.append(i[0])

print("{} #{}".format(i[1], i[0], i[2]))

print()

# Loop until the user enters a valid store ID

while True:

# Get ID

id = input("Enter store ID: ").strip()

# If the ID is not in IDs, continue

if id not in ids:

print("Invalid ID.")

continue

# Break if the ID is valid

break

# Delete store stocks from database

cursor.execute(DELETE\_ALL\_SELLS, (id,))

sql.commit()

# Delete store from database

cursor.execute(DELETE\_STORE, (id,))

sql.commit()

# Print success message

print()

print("Store #{} is deleted!".format(id))

continue

# Option 11 / Add Product to Store Stock

if inp == 11:

# Get all products

cursor.execute(SELECT\_PRODUCT)

pds = cursor.fetchall()

# List to hold product names and brands as tuples

pd\_tuples = []

# Print products header

print()

print("Products:")

# Add all the products as tuples to the list and print products

for i in pds:

pd\_tuples.append((i[1], i[0]))

print("Brand: {}, Name: {}, ${}".format(i[1], i[0], i[2]))

print()

# Loop until the user enters a valid product brand and name

while True:

# Get brand and name

brand = input("Brand: ").strip()

name = input("Name: ").strip()

# If the brand and name not in tuples, continue

if (brand,name) not in pd\_tuples:

print("Invalid product.")

continue

# Break if the product is valid

break

# Print stores header

print()

print("Stores:")

# Get stores

cursor.execute(SELECT\_STORE)

sts = cursor.fetchall()

# Gather store IDs in a list and print stores

ids = []

for i in sts:

ids.append(i[0])

print("{} #{}".format(i[1], i[0], i[2]))

print()

# Loop until the user enters a valid store ID

while True:

# Get ID

id = input("Store ID: ").strip()

# If the ID is not in IDs list, continue

if id not in ids:

print("Invalid ID.")

continue

# Break if the ID is valid

break

# Loop until the user enters a valid quantity

while True:

# Get quantity as string

qty\_str = input("Quantity: ").strip()

# Try to convert it to int, continue if cannot convert

try:

qty = int(qty\_str)

except:

print("Invalid quantity.")

continue

# If quantity is below 1, continue

if qty < 1:

print("Invalid quantity. Cannot be less than 1.")

continue

# Break if the quantity is valid

break

# Get all the products that are already in the stock of the selected store

cursor.execute(SELECT\_SELLS, (id,))

sells = cursor.fetchall()

# Variable for holding if the selected product is already in the stock

found = False

# Iterate over all the products in stock

for i in sells:

# If the product is in stock

if i[0] == brand and i[1] == name:

# Compute the new quantity

new\_qty = int(i[2]) + qty

# Update the quantity

cursor.execute(UPDATE\_SELLS, (new\_qty, name, brand, id))

sql.commit()

# Print success message

print()

print("{} stocks of {} {} is added to the stock of Store #{}!".format(qty, brand, name, id))

# Set variable as found

found = True

break

# If the product is found, skip the rest of the block

if found:

continue

# Add new product to the stock

cursor.execute(ADD\_SELLS, (name, brand, id, qty))

sql.commit()

# Print success message

print()

print("{} stocks of {} {} is added to the stock of Store #{}!".format(qty, brand, name, id))

continue

# Option 12 / Remove Product from Store Stock

if inp == 12:

# Print stores header

print()

print("Stores:")

# Get stores

cursor.execute(SELECT\_STORE)

sts = cursor.fetchall()

# Gather store IDs in a list and print stores

ids = []

for i in sts:

ids.append(i[0])

print("{} #{}".format(i[1], i[0], i[2]))

print()

# Loop until the user enters a valid ID

while True:

# Get ID

id = input("Store ID: ").strip()

# If the ID is not in the IDs list, continue

if id not in ids:

print("Invalid ID.")

continue

break

# Get all the products in the selected store's stock

cursor.execute(SELECT\_SELLS, (id,))

in\_stock = cursor.fetchall()

# Print items in stock header

print()

print("Items in stock:")

# Gather all the products as brand and name tuples, gather the quantities as values to the tuple keys, and print products

pd\_tuples = []

qtys = {}

for i in in\_stock:

pd\_tuples.append((i[0], i[1]))

qtys["({}, {})".format(i[0], i[1])] = int(i[2])

print("Brand: {}, Name: {}, Qty: {}".format(i[0], i[1], i[2]))

print()

# Loop until the user enters a valid product brand and name

while True:

# Get brand and name

brand = input("Brand: ").strip()

name = input("Name: ").strip()

# If the brand and name not in product tuples, continue

if (brand,name) not in pd\_tuples:

print("Invalid product.")

continue

# Break if the product is valid

break

# Loop until the user enters a valid quantity

while True:

# Get quantity as string

qty\_str = input("Quantity: ").strip()

# Try to convert it to int, continue if cannot convert

try:

qty = int(qty\_str)

except:

print("Invalid quantity.")

continue

# If the given quantity is less than 0 or more than the total quantity of the product, continue

if qty < 1 or qty > qtys["({}, {})".format(brand, name)]:

print("Invalid quantity. Cannot be less than 1 or more than total quantity.")

continue

# Break if the quantity is valid

break

# If the total quantity - given quantity equals 0, delete the product from stock

if qtys["({}, {})".format(brand, name)] - qty == 0:

cursor.execute(DELETE\_SELLS, (name, brand, id))

sql.commit()

# If there is still an amount quantity, update the quantity

else:

cursor.execute(UPDATE\_SELLS, (qtys["({}, {})".format(brand, name)] - qty, name, brand, id))

sql.commit()

# Print success message

print()

print("{} stocks of {} {} is deleted from Store #{}!".format(qty, brand, name, id))

continue

# Option 13 / List Products

if inp == 13:

# Get products

cursor.execute(SELECT\_PRODUCT)

pds = cursor.fetchall()

# Print products

print()

for i in pds:

print("Brand: {}, Name: {}, ${}".format(i[1], i[0], i[2]))

continue

# Option 14 / Add Product

if inp == 14:

# Print products header

print()

print("Products:")

# Get products

cursor.execute(SELECT\_PRODUCT)

pds = cursor.fetchall()

# Gather all the products as brand and name tuples and print products

tuples = []

for i in pds:

tuples.append((i[1], i[0]))

print("Brand:{}, Name: {}".format(i[1], i[0]))

print()

# Loop until the user enters a valid product brand and name

while True:

# Get brand and name

brand = input("Brand: ").strip()

name = input("Name: ").strip()

# If the brand and name not in product tuples, continue

if (brand,name) in tuples:

print("Product exists.")

continue

# Break if the product is valid

break

# Loop until the user enters a valid price

while True:

# Get price as string

pr\_str = input("Price: ").strip()

# Try to convert it to int, continue if cannot convert

try:

price = int(pr\_str)

except:

print("Invalid price.")

continue

# If the given price is less than 1, continue

if price < 1:

print("Invalid price. Cannot be less than 1.")

continue

# Break if the price is valid

break

# Add product

cursor.execute(ADD\_PRODUCT, (name, brand, price))

sql.commit()

# Print success message

print()

print("{} {} with the price ${} is added!".format(brand, name, price))

continue

# Option 15 / Delete Product

if inp == 15:

# Print products header

print()

print("Products:")

# Get products

cursor.execute(SELECT\_PRODUCT)

pds = cursor.fetchall()

# Gather all the products as brand and name tuples and print products

tuples = []

for i in pds:

tuples.append((i[1], i[0]))

print("Brand:{}, Name: {}".format(i[1], i[0]))

print()

# Loop until the user enters a valid product brand and name

while True:

# Get brand and name

brand = input("Brand: ").strip()

name = input("Name: ").strip()

# If the brand and name not in product tuples, continue

if (brand,name) not in tuples:

print("Product does not exist.")

continue

# Break if the product is valid

break

# Delete product from store stocks

cursor.execute(DELETE\_PD\_FR\_SELLS, (name, brand))

sql.commit()

# Delete product from warehouse stocks

cursor.execute(DELETE\_PD\_FR\_OWNS, (name, brand))

sql.commit()

# Delete product

cursor.execute(DELETE\_PRODUCT, (name, brand))

sql.commit()

# Print success message

print()

print("{} {} is deleted!".format(brand, name))

continue

# Option 16 / Update Product Price

if inp == 16:

# Print products header

print()

print("Products:")

# Get products

cursor.execute(SELECT\_PRODUCT)

pds = cursor.fetchall()

# Gather all the products as brand and name tuples and print products

tuples = []

for i in pds:

tuples.append((i[1], i[0]))

print("Brand: {}, Name: {}, ${}".format(i[1], i[0], i[2]))

print()

# Loop until the user enters a valid product brand and name

while True:

# Get brand and name

brand = input("Brand: ").strip()

name = input("Name: ").strip()

# If the brand and name not in product tuples, continue

if (brand,name) not in tuples:

print("Product does not exist.")

continue

# Break if the product is valid

break

# Loop until the user enters a valid price

while True:

# Get price as string

pr\_str = input("Price: ").strip()

# Try to convert it to int, continue if cannot convert

try:

price = int(pr\_str)

except:

print("Invalid price.")

continue

# If the given price is less than 1, continue

if price < 1:

print("Invalid price. Cannot be less than 1.")

continue

# Break if the price is valid

break

# Update product

cursor.execute(UPDATE\_PRICE, (price, name, brand))

sql.commit()

# Print success message

print()

print("Price of {} {} is changed to {}!".format(brand, name, price))

continue

# Option 17 / Exit

if inp == 17:

# Exit the system

sys.exit(0)

This is create.sql database creation codes:

CREATE TABLE Store(

id varchar(4) PRIMARY KEY,

name varchar(20),

address varchar(10) UNIQUE

);

CREATE TABLE Warehouse(

id varchar(4) PRIMARY KEY,

name varchar(20),

address varchar(10) UNIQUE

);

CREATE TABLE Product(

name varchar(20),

brand varchar(20),

price int,

PRIMARY KEY(brand, name)

);

CREATE TABLE Sells(

product\_name varchar(20),

product\_brand varchar(20),

store\_id varchar(4),

quantity int,

PRIMARY KEY(product\_brand, product\_name, market\_id),

FOREIGN KEY(product\_brand, product\_name) REFERENCES Product(brand, name),

FOREIGN KEY(store\_id) REFERENCES Store(id)

);

CREATE TABLE Owns(

product\_name varchar(20),

product\_brand varchar(20),

warehouse\_id varchar(4),

quantity int,

PRIMARY KEY(product\_brand, product\_name, warehouse\_id),

FOREIGN KEY(product\_brand, product\_name) REFERENCES Product(brand, name),

FOREIGN KEY(warehouse\_id) REFERENCES Warehouse(id)

);

This is warehouse.py :

from random import randint

import lib.stddraw as std

from lib.picture import Picture

from pathlib import Path

# SQL queries

DELETE\_STOCK = "DELETE FROM Owns WHERE (product\_name, product\_brand, warehouse\_id) = (?, ?, ?)"

UPDATE\_STOCK = "UPDATE Owns SET quantity=? WHERE (product\_name, product\_brand, warehouse\_id) = (?, ?, ?)"

# Warehouse class

class Warehouse:

# Initialize a warehouse with x, y, and its name. Optionally, products in stock and ID can be given.

def \_\_init\_\_(self, x, y, name, in\_stock=None, id="{:04d}".format(randint(0,9999))) -> None:

# Set the fields

self.id = id

self.name = name

self.x = x

self.y = y

# Set the in\_stock field

if in\_stock is None:

self.in\_stock = []

else:

self.in\_stock = in\_stock

# Return a formatted string of the warehouse

def \_\_str\_\_(self) -> str:

return "{} #{}".format(self.name, self.id)

# Draw the warehouse

def draw(self):

# Get the picture and place it onto the canvas

std.picture(Picture(Path(\_\_file\_\_).parent.resolve().parent.resolve().joinpath("src", "warehouse.png")),self.x+0.5, self.y+0.5)

# Write its name and ID on top of the warehouse

std.setPenColor(std.BLACK)

std.setFontSize(10)

std.text(self.x+0.5,self.y+1.15,str(self))

# Remove a desired amount of quantity of the given product from the stock

def remove\_stock(self, product, quantity, cursor, sql):

# Iterate over all the products in stock

for i in self.in\_stock:

# If the current product is the given one

if str(i[0]) == str(product):

# If the product quantity will be zero after the removing operating, remove the product directly both from list and database

if i[1] - quantity == 0:

self.in\_stock.remove(i)

cursor.execute(DELETE\_STOCK, (product.name, product.brand, self.id))

sql.commit()

# If there will be more than 0 stocks left, update the quantity both on list and database

else:

i[1] -= quantity

cursor.execute(UPDATE\_STOCK, (i[1], product.name, product.brand, self.id))

sql.commit()

break

# Draw a road from this warehouse to the given store

def \_draw\_road(self, store, delivered=False):

# Get the distance

dist = round(((store.x - self.x)\*\*2 + (store.y - self.y)\*\*2)\*\*(1/2))

# Compute the step count

step\_count = dist \* 3

# Compute the x and y step values

x\_step = (store.x - self.x) / step\_count

y\_step = (store.y - self.y) / step\_count

# Set the pen color

# If the product is not delivered yet, set it to red. Otherwise, set it to green.

if not delivered:

std.setPenColor(std.RED)

else:

std.setPenColor(std.DARK\_GREEN)

# For every step

for i in range(step\_count):

# Draw a circle for the step

std.filledCircle(self.x + 0.5 + x\_step\*i, self.y + 0.5 + y\_step\*i, 0.1)

# If not delivered yet, wait for a random amount of time between 0.4 and 1.2 second.

if not delivered:

std.show(randint(400,1200))

# Deliver an amount of stocks of a product from this warehouse to the given store

def deliver(self, store, product, quantity, stores\_list, warehouses\_list, cursor, sql):

# Create a canvas

std.setCanvasSize(550, 600)

std.setXscale(0,10)

std.setYscale(0,11)

# Draw the stores

for i in stores\_list:

i.draw()

# Draw the warehouses

for i in warehouses\_list:

i.draw()

# Set the font family, font size, color

std.setFontFamily("arial")

std.setFontSize(20)

std.setPenColor(std.BLACK)

# Draw the "Preparing items..." text on top of the canvas

std.boldText(5, 10.6, "Preparing items...")

# Wait for a random amount of time which is based on the quantity

std.show(randint(100, 600) \* quantity)

# Remove stocks of the product from this warehouse

self.remove\_stock(product, quantity, cursor, sql)

# Add stocks of the product to the store

store.add\_stock(product, quantity, cursor, sql)

# Clear the canvaas

std.clear()

# Draw the stores

for i in stores\_list:

i.draw()

# Draw the warehouses

for i in warehouses\_list:

i.draw()

# Set the font family, font size, color

std.setFontFamily("arial")

std.setFontSize(20)

std.setPenColor(std.BLACK)

# Draw the delivering statement text on top of the canvas

std.boldText(5, 10.6, "Delivering from {} #{} to {} #{}...".format(self.name, self.id, store.name, store.id))

# Draw a road between this warehouse and the store

self.\_draw\_road(store)

# Clear the canvas

std.clear()

# Draw the stores

for i in stores\_list:

i.draw()

# Draw the warehouses

for i in warehouses\_list:

i.draw()

# Draw the delivered road

self.\_draw\_road(store, delivered=True)

# Set the font family, font size, color

std.setFontFamily("arial")

std.setPenColor(std.DARK\_GREEN)

std.setFontSize(20)

# Draw the "Delivered!" text on top of the canvas

std.boldText(5, 10.6, "Delivered!")

# Show the canvas and wait for 1.5 seconds

std.show(1500)

# Hide the canvas

std.hide()

This is store.py store class:

from random import randint

import lib.stddraw as std

from lib.picture import Picture

from pathlib import Path

# SQL queries

ADD\_STOCK = "INSERT INTO Sells VALUES(?, ?, ?, ?)"

UPDATE\_STOCK = "UPDATE Sells SET quantity=? WHERE (product\_name, product\_brand, store\_id) = (?, ?, ?)"

# Store class

class Store:

# Initialize a store with x, y, and its name. Optionally, products in stock and ID can be given.

def \_\_init\_\_(self, x, y, name, in\_stock=None, id="{:04d}".format(randint(0,9999))) -> None:

# Set the fields

self.id =id

self.name = name

self.x = x

self.y = y

# Set the in\_stock field

if in\_stock is None:

self.in\_stock = []

else:

self.in\_stock = in\_stock

# Return a formatted string of the store

def \_\_str\_\_(self) -> str:

return "{} #{}".format(self.name, self.id)

# Add a desired amount of quantity of the given product to the stock

def add\_stock(self, product, quantity, cursor, sql):

# Iterate over all products in stock

for i in self.in\_stock:

# If the given product is in the stock already

if i[0].name == product.name and i[0].brand == product.brand:

# Update its quantity both on list and database

i[1] += quantity

cursor.execute(UPDATE\_STOCK, (i[1], product.name, product.brand, self.id))

sql.commit()

return

# If the given product is a new product, add it both to list and database

cursor.execute(ADD\_STOCK, (product.name, product.brand, self.id, quantity))

sql.commit()

self.in\_stock.append([product, quantity])

# Draw the stock

def draw(self):

# Get the picture and place it onto the canvas

std.picture(Picture(Path(\_\_file\_\_).parent.resolve().parent.resolve().joinpath("src", "store.png")),self.x+0.5, self.y+0.5)

# Write its name and ID on top of the warehouse

std.setPenColor(std.BLACK)

std.setFontSize(10)

std.text(self.x+0.5,self.y+1.15,str(self))

This part of code is lib for drawing lines in based canvas it is Lib from Princeton University's Standard Library :

"""

stddraw.py

The stddraw module defines functions that allow the user to create a

drawing. A drawing appears on the canvas. The canvas appears

in the window. As a convenience, the module also imports the

commonly used Color objects defined in the color module.

"""

import time

import os

import sys

import lib.color

import string

os.environ['PYGAME\_HIDE\_SUPPORT\_PROMPT'] = 'hide'

import pygame

import pygame.gfxdraw

import pygame.font

import tkinter as Tkinter

import tkinter.messagebox as tkMessageBox

import tkinter.filedialog as tkFileDialog

#-----------------------------------------------------------------------

# Define colors so clients need not import the color module.

from lib.color import WHITE

from lib.color import BLACK

from lib.color import RED

from lib.color import GREEN

from lib.color import BLUE

from lib.color import CYAN

from lib.color import MAGENTA

from lib.color import YELLOW

from lib.color import DARK\_RED

from lib.color import DARK\_GREEN

from lib.color import DARK\_BLUE

from lib.color import GRAY

from lib.color import DARK\_GRAY

from lib.color import LIGHT\_GRAY

from lib.color import ORANGE

from lib.color import VIOLET

from lib.color import PINK

from lib.color import BOOK\_BLUE

from lib.color import BOOK\_LIGHT\_BLUE

from lib.color import BOOK\_RED

#-----------------------------------------------------------------------

# Default Sizes and Values

\_BORDER = 0.0

#\_BORDER = 0.05

\_DEFAULT\_XMIN = 0.0

\_DEFAULT\_XMAX = 1.0

\_DEFAULT\_YMIN = 0.0

\_DEFAULT\_YMAX = 1.0

\_DEFAULT\_CANVAS\_SIZE = 512

\_DEFAULT\_PEN\_RADIUS = .005 # Maybe change this to 0.0 in the future.

\_DEFAULT\_PEN\_COLOR = lib.color.BLACK

\_DEFAULT\_FONT\_FAMILY = 'Helvetica'

\_DEFAULT\_FONT\_SIZE = 12

\_xmin = None

\_ymin = None

\_xmax = None

\_ymax = None

\_fontFamily = \_DEFAULT\_FONT\_FAMILY

\_fontSize = \_DEFAULT\_FONT\_SIZE

\_canvasWidth = float(\_DEFAULT\_CANVAS\_SIZE)

\_canvasHeight = float(\_DEFAULT\_CANVAS\_SIZE)

\_penRadius = None

\_penColor = \_DEFAULT\_PEN\_COLOR

\_keysTyped = []

# Has the window been created?

\_windowCreated = False

#-----------------------------------------------------------------------

# Begin added by Alan J. Broder

#-----------------------------------------------------------------------

# Keep track of mouse status

# Has the mouse been left-clicked since the last time we checked?

\_mousePressed = False

# The position of the mouse as of the most recent mouse click

\_mousePos = None

#-----------------------------------------------------------------------

# End added by Alan J. Broder

#-----------------------------------------------------------------------

#-----------------------------------------------------------------------

def \_pygameColor(c):

"""

Convert c, an object of type color.Color, to an equivalent object

of type pygame.Color. Return the result.

"""

r = c.getRed()

g = c.getGreen()

b = c.getBlue()

return pygame.Color(r, g, b)

#-----------------------------------------------------------------------

# Private functions to scale and factor X and Y values.

def \_scaleX(x):

return \_canvasWidth \* (x - \_xmin) / (\_xmax - \_xmin)

def \_scaleY(y):

return \_canvasHeight \* (\_ymax - y) / (\_ymax - \_ymin)

def \_factorX(w):

return w \* \_canvasWidth / abs(\_xmax - \_xmin)

def \_factorY(h):

return h \* \_canvasHeight / abs(\_ymax - \_ymin)

#-----------------------------------------------------------------------

# Begin added by Alan J. Broder

#-----------------------------------------------------------------------

def \_userX(x):

return \_xmin + x \* (\_xmax - \_xmin) / \_canvasWidth

def \_userY(y):

return \_ymax - y \* (\_ymax - \_ymin) / \_canvasHeight

#-----------------------------------------------------------------------

# End added by Alan J. Broder

#-----------------------------------------------------------------------

#-----------------------------------------------------------------------

def setCanvasSize(w=\_DEFAULT\_CANVAS\_SIZE, h=\_DEFAULT\_CANVAS\_SIZE):

"""

Set the size of the canvas to w pixels wide and h pixels high.

Calling this function is optional. If you call it, you must do

so before calling any drawing function.

"""

global \_background

global \_surface

global \_canvasWidth

global \_canvasHeight

global \_windowCreated

if (w < 1) or (h < 1):

raise Exception('width and height must be positive')

\_canvasWidth = w

\_canvasHeight = h

\_background = pygame.display.set\_mode([w, h])

pygame.display.set\_caption('Delivery Tracking')

\_surface = pygame.Surface((w, h))

\_surface.fill(\_pygameColor(WHITE))

\_windowCreated = True

def setXscale(min=\_DEFAULT\_XMIN, max=\_DEFAULT\_XMAX):

"""

Set the x-scale of the canvas such that the minimum x value

is min and the maximum x value is max.

"""

global \_xmin

global \_xmax

min = float(min)

max = float(max)

if min >= max:

raise Exception('min must be less than max')

size = max - min

\_xmin = min - \_BORDER \* size

\_xmax = max + \_BORDER \* size

def setYscale(min=\_DEFAULT\_YMIN, max=\_DEFAULT\_YMAX):

"""

Set the y-scale of the canvas such that the minimum y value

is min and the maximum y value is max.

"""

global \_ymin

global \_ymax

min = float(min)

max = float(max)

if min >= max:

raise Exception('min must be less than max')

size = max - min

\_ymin = min - \_BORDER \* size

\_ymax = max + \_BORDER \* size

def hide():

global \_background

global \_canvasWidth

global \_canvasHeight

\_background = pygame.display.set\_mode([\_canvasWidth, \_canvasHeight], flags=pygame.HIDDEN)

def unhide():

global \_background

global \_canvasWidth

global \_canvasHeight

\_background = pygame.display.set\_mode([\_canvasWidth, \_canvasHeight], flags=pygame.SHOWN)

def setPenRadius(r=\_DEFAULT\_PEN\_RADIUS):

"""

Set the pen radius to r, thus affecting the subsequent drawing

of points and lines. If r is 0.0, then points will be drawn with

the minimum possible radius and lines with the minimum possible

width.

"""

global \_penRadius

r = float(r)

if r < 0.0:

raise Exception('Argument to setPenRadius() must be non-neg')

\_penRadius = r \* float(\_DEFAULT\_CANVAS\_SIZE)

def setPenColor(c=\_DEFAULT\_PEN\_COLOR):

"""

Set the pen color to c, where c is an object of class color.Color.

c defaults to stddraw.BLACK.

"""

global \_penColor

\_penColor = c

def setFontFamily(f=\_DEFAULT\_FONT\_FAMILY):

"""

Set the font family to f (e.g. 'Helvetica' or 'Courier').

"""

global \_fontFamily

\_fontFamily = f

def setFontSize(s=\_DEFAULT\_FONT\_SIZE):

"""

Set the font size to s (e.g. 12 or 16).

"""

global \_fontSize

\_fontSize = s

#-----------------------------------------------------------------------

def \_makeSureWindowCreated():

global \_windowCreated

if not \_windowCreated:

setCanvasSize()

\_windowCreated = True

#-----------------------------------------------------------------------

# Functions to draw shapes, text, and images on the background canvas.

def \_pixel(x, y):

"""

Draw on the background canvas a pixel at (x, y).

"""

\_makeSureWindowCreated()

xs = \_scaleX(x)

xy = \_scaleY(y)

pygame.gfxdraw.pixel(

\_surface,

int(round(xs)),

int(round(xy)),

\_pygameColor(\_penColor))

def point(x, y):

"""

Draw on the background canvas a point at (x, y).

"""

\_makeSureWindowCreated()

x = float(x)

y = float(y)

# If the radius is too small, then simply draw a pixel.

if \_penRadius <= 1.0:

\_pixel(x, y)

else:

xs = \_scaleX(x)

ys = \_scaleY(y)

pygame.draw.ellipse(

\_surface,

\_pygameColor(\_penColor),

pygame.Rect(

xs-\_penRadius,

ys-\_penRadius,

\_penRadius\*2.0,

\_penRadius\*2.0),

0)

def \_thickLine(x0, y0, x1, y1, r):

"""

Draw on the background canvas a line from (x0, y0) to (x1, y1).

Draw the line with a pen whose radius is r.

"""

xs0 = \_scaleX(x0)

ys0 = \_scaleY(y0)

xs1 = \_scaleX(x1)

ys1 = \_scaleY(y1)

if (abs(xs0-xs1) < 1.0) and (abs(ys0-ys1) < 1.0):

filledCircle(x0, y0, r)

return

xMid = (x0+x1)/2

yMid = (y0+y1)/2

\_thickLine(x0, y0, xMid, yMid, r)

\_thickLine(xMid, yMid, x1, y1, r)

def line(x0, y0, x1, y1):

"""

Draw on the background canvas a line from (x0, y0) to (x1, y1).

"""

THICK\_LINE\_CUTOFF = 3 # pixels

\_makeSureWindowCreated()

x0 = float(x0)

y0 = float(y0)

x1 = float(x1)

y1 = float(y1)

lineWidth = \_penRadius \* 2.0

if lineWidth == 0.0: lineWidth = 1.0

if lineWidth < THICK\_LINE\_CUTOFF:

x0s = \_scaleX(x0)

y0s = \_scaleY(y0)

x1s = \_scaleX(x1)

y1s = \_scaleY(y1)

pygame.draw.line(

\_surface,

\_pygameColor(\_penColor),

(x0s, y0s),

(x1s, y1s),

int(round(lineWidth)))

else:

\_thickLine(x0, y0, x1, y1, \_penRadius/\_DEFAULT\_CANVAS\_SIZE)

def circle(x, y, r):

"""

Draw on the background canvas a circle of radius r centered on

(x, y).

"""

\_makeSureWindowCreated()

x = float(x)

y = float(y)

r = float(r)

ws = \_factorX(2.0\*r)

hs = \_factorY(2.0\*r)

# If the radius is too small, then simply draw a pixel.

if (ws <= 1.0) and (hs <= 1.0):

\_pixel(x, y)

else:

xs = \_scaleX(x)

ys = \_scaleY(y)

pygame.draw.ellipse(

\_surface,

\_pygameColor(\_penColor),

pygame.Rect(xs-ws/2.0, ys-hs/2.0, ws, hs),

int(round(\_penRadius)))

def filledCircle(x, y, r):

"""

Draw on the background canvas a filled circle of radius r

centered on (x, y).

"""

\_makeSureWindowCreated()

x = float(x)

y = float(y)

r = float(r)

ws = \_factorX(2.0\*r)

hs = \_factorY(2.0\*r)

# If the radius is too small, then simply draw a pixel.

if (ws <= 1.0) and (hs <= 1.0):

\_pixel(x, y)

else:

xs = \_scaleX(x)

ys = \_scaleY(y)

pygame.draw.ellipse(

\_surface,

\_pygameColor(\_penColor),

pygame.Rect(xs-ws/2.0, ys-hs/2.0, ws, hs),

0)

def rectangle(x, y, w, h):

"""

Draw on the background canvas a rectangle of width w and height h

whose lower left point is (x, y).

"""

global \_surface

\_makeSureWindowCreated()

x = float(x)

y = float(y)

w = float(w)

h = float(h)

ws = \_factorX(w)

hs = \_factorY(h)

# If the rectangle is too small, then simply draw a pixel.

if (ws <= 1.0) and (hs <= 1.0):

\_pixel(x, y)

else:

xs = \_scaleX(x)

ys = \_scaleY(y)

pygame.draw.rect(

\_surface,

\_pygameColor(\_penColor),

pygame.Rect(xs, ys-hs, ws, hs),

int(round(\_penRadius)))

def filledRectangle(x, y, w, h):

"""

Draw on the background canvas a filled rectangle of width w and

height h whose lower left point is (x, y).

"""

global \_surface

\_makeSureWindowCreated()

x = float(x)

y = float(y)

w = float(w)

h = float(h)

ws = \_factorX(w)

hs = \_factorY(h)

# If the rectangle is too small, then simply draw a pixel.

if (ws <= 1.0) and (hs <= 1.0):

\_pixel(x, y)

else:

xs = \_scaleX(x)

ys = \_scaleY(y)

pygame.draw.rect(

\_surface,

\_pygameColor(\_penColor),

pygame.Rect(xs, ys-hs, ws, hs),

0)

def square(x, y, r):

"""

Draw on the background canvas a square whose sides are of length

2r, centered on (x, y).

"""

\_makeSureWindowCreated()

rectangle(x-r, y-r, 2.0\*r, 2.0\*r)

def filledSquare(x, y, r):

"""

Draw on the background canvas a filled square whose sides are of

length 2r, centered on (x, y).

"""

\_makeSureWindowCreated()

filledRectangle(x-r, y-r, 2.0\*r, 2.0\*r)

def polygon(x, y):

"""

Draw on the background canvas a polygon with coordinates

(x[i], y[i]).

"""

global \_surface

\_makeSureWindowCreated()

# Scale X and Y values.

xScaled = []

for xi in x:

xScaled.append(\_scaleX(float(xi)))

yScaled = []

for yi in y:

yScaled.append(\_scaleY(float(yi)))

points = []

for i in range(len(x)):

points.append((xScaled[i], yScaled[i]))

points.append((xScaled[0], yScaled[0]))

pygame.draw.polygon(

\_surface,

\_pygameColor(\_penColor),

points,

int(round(\_penRadius)))

def filledPolygon(x, y):

"""

Draw on the background canvas a filled polygon with coordinates

(x[i], y[i]).

"""

global \_surface

\_makeSureWindowCreated()

# Scale X and Y values.

xScaled = []

for xi in x:

xScaled.append(\_scaleX(float(xi)))

yScaled = []

for yi in y:

yScaled.append(\_scaleY(float(yi)))

points = []

for i in range(len(x)):

points.append((xScaled[i], yScaled[i]))

points.append((xScaled[0], yScaled[0]))

pygame.draw.polygon(\_surface, \_pygameColor(\_penColor), points, 0)

def text(x, y, s):

"""

Draw string s on the background canvas centered at (x, y).

"""

\_makeSureWindowCreated()

x = float(x)

y = float(y)

xs = \_scaleX(x)

ys = \_scaleY(y)

font = pygame.font.SysFont(\_fontFamily, \_fontSize)

text = font.render(s, 1, \_pygameColor(\_penColor))

textpos = text.get\_rect(center=(xs, ys))

\_surface.blit(text, textpos)

def boldText(x, y, s):

"""

Draw string s as a bold text on the background canvas centered at (x, y).

"""

\_makeSureWindowCreated()

x = float(x)

y = float(y)

xs = \_scaleX(x)

ys = \_scaleY(y)

font = pygame.font.SysFont(\_fontFamily, \_fontSize, True)

text = font.render(s, 1, \_pygameColor(\_penColor))

textpos = text.get\_rect(center=(xs, ys))

\_surface.blit(text, textpos)

def picture(pic, x=None, y=None):

"""

Draw pic on the background canvas centered at (x, y). pic is an

object of class picture.Picture. x and y default to the midpoint

of the background canvas.

"""

global \_surface

\_makeSureWindowCreated()

# By default, draw pic at the middle of the surface.

if x is None:

x = (\_xmax + \_xmin) / 2.0

if y is None:

y = (\_ymax + \_ymin) / 2.0

x = float(x)

y = float(y)

xs = \_scaleX(x)

ys = \_scaleY(y)

ws = pic.width()

hs = pic.height()

picSurface = pic.\_surface # violates encapsulation

\_surface.blit(picSurface, [xs-ws/2.0, ys-hs/2.0, ws, hs])

def clear(c=WHITE):

"""

Clear the background canvas to color c, where c is an

object of class color.Color. c defaults to stddraw.WHITE.

"""

\_makeSureWindowCreated()

\_surface.fill(\_pygameColor(c))

def save(f):

"""

Save the window canvas to file f.

"""

\_makeSureWindowCreated()

#if sys.hexversion >= 0x03000000:

# # Hack because Pygame without full image support

# # can handle only .bmp files.

# bmpFileName = f + '.bmp'

# pygame.image.save(\_surface, bmpFileName)

# os.system('convert ' + bmpFileName + ' ' + f)

# os.system('rm ' + bmpFileName)

#else:

# pygame.image.save(\_surface, f)

pygame.image.save(\_surface, f)

#-----------------------------------------------------------------------

def \_show():

"""

Copy the background canvas to the window canvas.

"""

\_background.blit(\_surface, (0, 0))

pygame.display.flip()

\_checkForEvents()

def \_showAndWaitForever():

"""

Copy the background canvas to the window canvas. Then wait

forever, that is, until the user closes the stddraw window.

"""

\_makeSureWindowCreated()

\_show()

QUANTUM = .1

while True:

time.sleep(QUANTUM)

\_checkForEvents()

def show(msec=float('inf')):

"""

Copy the background canvas to the window canvas, and

then wait for msec milliseconds. msec defaults to infinity.

"""

if msec == float('inf'):

\_showAndWaitForever()

\_makeSureWindowCreated()

\_show()

\_checkForEvents()

# Sleep for the required time, but check for events every

# QUANTUM seconds.

QUANTUM = .1

sec = msec / 1000.0

if sec < QUANTUM:

time.sleep(sec)

return

secondsWaited = 0.0

while secondsWaited < sec:

time.sleep(QUANTUM)

secondsWaited += QUANTUM

\_checkForEvents()

#-----------------------------------------------------------------------

def \_saveToFile():

"""

Display a dialog box that asks the user for a file name. Save the

drawing to the specified file. Display a confirmation dialog box

if successful, and an error dialog box otherwise. The dialog boxes

are displayed using Tkinter, which (on some computers) is

incompatible with Pygame. So the dialog boxes must be displayed

from child processes.

"""

import subprocess

\_makeSureWindowCreated()

stddrawPath = os.path.realpath(\_\_file\_\_)

childProcess = subprocess.Popen(

[sys.executable, stddrawPath, 'getFileName'],

stdout=subprocess.PIPE)

so, se = childProcess.communicate()

fileName = so.strip()

if sys.hexversion >= 0x03000000:

fileName = fileName.decode('utf-8')

if fileName == '':

return

if not fileName.endswith(('.jpg', '.png')):

childProcess = subprocess.Popen(

[sys.executable, stddrawPath, 'reportFileSaveError',

'File name must end with ".jpg" or ".png".'])

return

try:

save(fileName)

childProcess = subprocess.Popen(

[sys.executable, stddrawPath, 'confirmFileSave'])

except (pygame.error) as e:

childProcess = subprocess.Popen(

[sys.executable, stddrawPath, 'reportFileSaveError', str(e)])

def \_checkForEvents():

"""

Check if any new event has occured (such as a key typed or button

pressed). If a key has been typed, then put that key in a queue.

"""

global \_surface

global \_keysTyped

#-------------------------------------------------------------------

# Begin added by Alan J. Broder

#-------------------------------------------------------------------

global \_mousePos

global \_mousePressed

#-------------------------------------------------------------------

# End added by Alan J. Broder

#-------------------------------------------------------------------

\_makeSureWindowCreated()

for event in pygame.event.get():

if event.type == pygame.QUIT:

#sys.exit()

pass

elif event.type == pygame.KEYDOWN:

\_keysTyped = [event.unicode] + \_keysTyped

elif (event.type == pygame.MOUSEBUTTONUP) and \

(event.button == 3):

\_saveToFile()

#---------------------------------------------------------------

# Begin added by Alan J. Broder

#---------------------------------------------------------------

# Every time the mouse button is pressed, remember

# the mouse position as of that press.

elif (event.type == pygame.MOUSEBUTTONDOWN) and \

(event.button == 1):

\_mousePressed = True

\_mousePos = event.pos

#---------------------------------------------------------------

# End added by Alan J. Broder

#---------------------------------------------------------------

#-----------------------------------------------------------------------

# Functions for retrieving keys

def hasNextKeyTyped():

"""

Return True if the queue of keys the user typed is not empty.

Otherwise return False.

"""

global \_keysTyped

return \_keysTyped != []

def nextKeyTyped():

"""

Remove the first key from the queue of keys that the the user typed,

and return that key.

"""

global \_keysTyped

return \_keysTyped.pop()

#-----------------------------------------------------------------------

# Begin added by Alan J. Broder

#-----------------------------------------------------------------------

# Functions for dealing with mouse clicks

def mousePressed():

"""

Return True if the mouse has been left-clicked since the

last time mousePressed was called, and False otherwise.

"""

global \_mousePressed

if \_mousePressed:

\_mousePressed = False

return True

return False

def mouseX():

"""

Return the x coordinate in user space of the location at

which the mouse was most recently left-clicked. If a left-click

hasn't happened yet, raise an exception, since mouseX() shouldn't

be called until mousePressed() returns True.

"""

global \_mousePos

if \_mousePos:

return \_userX(\_mousePos[0])

raise Exception(

"Can't determine mouse position if a click hasn't happened")

def mouseY():

"""

Return the y coordinate in user space of the location at

which the mouse was most recently left-clicked. If a left-click

hasn't happened yet, raise an exception, since mouseY() shouldn't

be called until mousePressed() returns True.

"""

global \_mousePos

if \_mousePos:

return \_userY(\_mousePos[1])

raise Exception(

"Can't determine mouse position if a click hasn't happened")

#-----------------------------------------------------------------------

# End added by Alan J. Broder

#-----------------------------------------------------------------------

#-----------------------------------------------------------------------

# Initialize the x scale, the y scale, and the pen radius.

setXscale()

setYscale()

setPenRadius()

pygame.font.init()

#-----------------------------------------------------------------------

# Functions for displaying Tkinter dialog boxes in child processes.

def \_getFileName():

"""

Display a dialog box that asks the user for a file name.

"""

root = Tkinter.Tk()

root.withdraw()

reply = tkFileDialog.asksaveasfilename(initialdir='.')

sys.stdout.write(reply)

sys.stdout.flush()

sys.exit()

def \_confirmFileSave():

"""

Display a dialog box that confirms a file save operation.

"""

root = Tkinter.Tk()

root.withdraw()

tkMessageBox.showinfo(title='File Save Confirmation',

message='The drawing was saved to the file.')

sys.exit()

def \_reportFileSaveError(msg):

"""

Display a dialog box that reports a msg. msg is a string which

describes an error in a file save operation.

"""

root = Tkinter.Tk()

root.withdraw()

tkMessageBox.showerror(title='File Save Error', message=msg)

sys.exit()

#-----------------------------------------------------------------------

def \_regressionTest():

"""

Perform regression testing.

"""

clear()

setPenRadius(.5)

setPenColor(ORANGE)

point(0.5, 0.5)

show(0.0)

setPenRadius(.25)

setPenColor(BLUE)

point(0.5, 0.5)

show(0.0)

setPenRadius(.02)

setPenColor(RED)

point(0.25, 0.25)

show(0.0)

setPenRadius(.01)

setPenColor(GREEN)

point(0.25, 0.25)

show(0.0)

setPenRadius(0)

setPenColor(BLACK)

point(0.25, 0.25)

show(0.0)

setPenRadius(.1)

setPenColor(RED)

point(0.75, 0.75)

show(0.0)

setPenRadius(0)

setPenColor(CYAN)

for i in range(0, 100):

point(i / 512.0, .5)

point(.5, i / 512.0)

show(0.0)

setPenRadius(0)

setPenColor(MAGENTA)

line(.1, .1, .3, .3)

line(.1, .2, .3, .2)

line(.2, .1, .2, .3)

show(0.0)

setPenRadius(.05)

setPenColor(MAGENTA)

line(.7, .5, .8, .9)

show(0.0)

setPenRadius(.01)

setPenColor(YELLOW)

circle(.75, .25, .2)

show(0.0)

setPenRadius(.01)

setPenColor(YELLOW)

filledCircle(.75, .25, .1)

show(0.0)

setPenRadius(.01)

setPenColor(PINK)

rectangle(.25, .75, .1, .2)

show(0.0)

setPenRadius(.01)

setPenColor(PINK)

filledRectangle(.25, .75, .05, .1)

show(0.0)

setPenRadius(.01)

setPenColor(DARK\_RED)

square(.5, .5, .1)

show(0.0)

setPenRadius(.01)

setPenColor(DARK\_RED)

filledSquare(.5, .5, .05)

show(0.0)

setPenRadius(.01)

setPenColor(DARK\_BLUE)

polygon([.4, .5, .6], [.7, .8, .7])

show(0.0)

setPenRadius(.01)

setPenColor(DARK\_GREEN)

setFontSize(24)

text(.2, .4, 'hello, world')

show(0.0)

#import picture as p

#pic = p.Picture('saveIcon.png')

#picture(pic, .5, .85)

#show(0.0)

# Test handling of mouse and keyboard events.

setPenColor(BLACK)

print('Left click with the mouse or type a key')

while True:

if mousePressed():

filledCircle(mouseX(), mouseY(), .02)

if hasNextKeyTyped():

print(nextKeyTyped())

show(0.0)

# Never get here.

show()

#-----------------------------------------------------------------------

def \_main():

"""

Dispatch to a function that does regression testing, or to a

dialog-box-handling function.

"""

import sys

if len(sys.argv) == 1:

\_regressionTest()

elif sys.argv[1] == 'getFileName':

\_getFileName()

elif sys.argv[1] == 'confirmFileSave':

\_confirmFileSave()

elif sys.argv[1] == 'reportFileSaveError':

\_reportFileSaveError(sys.argv[2])

if \_\_name\_\_ == '\_\_main\_\_':

\_main()

This is product.py product class:

# Product class

class Product:

# Create a product with name, brand, and price.

def \_\_init\_\_(self, name, brand, price) -> None:

# Set the fields

self.name = name

self.brand = brand

self.price = price

# Return a formatted string of the product

def \_\_str\_\_(self) -> str:

return "{} {}, ${}".format(self.brand, self.name, self.price)

This is picture.py again lib:

"""

picture.py

The picture module defines the Picture class.

"""

#-----------------------------------------------------------------------

import os

import lib.color

os.environ['PYGAME\_HIDE\_SUPPORT\_PROMPT'] = 'hide'

import pygame

#-----------------------------------------------------------------------

\_DEFAULT\_WIDTH = 512

\_DEFAULT\_HEIGHT = 512

#-----------------------------------------------------------------------

class Picture:

"""

A Picture object models an image. It is initialized such that

it has a given width and height and contains all black pixels.

Subsequently you can load an image from a given JPG or PNG file.

"""

def \_\_init\_\_(self, arg1=None, arg2=None):

"""

If both arg1 and arg2 are None, then construct self such that

it is all black with \_DEFAULT\_WIDTH and height \_DEFAULT\_HEIGHT.

If arg1 is not None and arg2 is None, then construct self by

reading from the file whose name is arg1.

If neither arg1 nor arg2 is None, then construct self such that

it is all black with width arg1 and and height arg2.

"""

if (arg1 is None) and (arg2 is None):

maxW = \_DEFAULT\_WIDTH

maxH = \_DEFAULT\_HEIGHT

self.\_surface = pygame.Surface((maxW, maxH))

self.\_surface.fill((0, 0, 0))

elif (arg1 is not None) and (arg2 is None):

fileName = arg1

try:

self.\_surface = pygame.image.load(fileName)

except pygame.error:

raise IOError()

elif (arg1 is not None) and (arg2 is not None):

maxW = arg1

maxH = arg2

self.\_surface = pygame.Surface((maxW, maxH))

self.\_surface.fill((0, 0, 0))

else:

raise ValueError()

#-------------------------------------------------------------------

#def load(self, f):

# """

# Change self by reading from the file whose name is f. The

# dimensions of the read image override the dimensions specified

# in the constructor.

# """

#if sys.hexversion >= 0x03000000:

# # Hack because Pygame without full image support

# # can handle only .bmp files.

# bmpFileName = f + '.bmp'

# os.system('convert ' + f + ' ' + bmpFileName)

# self.\_surface = pygame.image.load(bmpFileName)

# os.system('rm ' + bmpFileName)

#else:

# self.\_surface = pygame.image.load(f)

# self.\_surface = pygame.image.load(f)

#-------------------------------------------------------------------

def save(self, f):

"""

Save self to the file whose name is f.

"""

#if sys.hexversion >= 0x03000000:

# # Hack because Pygame without full image support

# # can handle only .bmp files.

# bmpFileName = f + '.bmp'

# pygame.image.save(self.\_surface, bmpFileName)

# os.system('convert ' + bmpFileName + ' ' + f)

# os.system('rm ' + bmpFileName)

#else:

# pygame.image.save(self.\_surface, f)

pygame.image.save(self.\_surface, f)

#-------------------------------------------------------------------

def width(self):

"""

Return the width of self.

"""

return self.\_surface.get\_width()

#-------------------------------------------------------------------

def height(self):

"""

Return the height of self.

"""

return self.\_surface.get\_height()

#-------------------------------------------------------------------

def get(self, x, y):

"""

Return the color of self at location (x, y).

"""

pygameColor = self.\_surface.get\_at((x, y))

return lib.color.Color(pygameColor.r, pygameColor.g, pygameColor.b)

#-------------------------------------------------------------------

def set(self, x, y, c):

"""

Set the color of self at location (x, y) to c.

"""

pygameColor = pygame.Color(c.getRed(), c.getGreen(),

c.getBlue(), 0)

self.\_surface.set\_at((x, y), pygameColor)

This is color.py color lib:

"""

color.py

The color module defines the Color class and some popular Color

objects.

"""

#-----------------------------------------------------------------------

class Color:

"""

A Color object models an RGB color.

"""

#-------------------------------------------------------------------

def \_\_init\_\_(self, r=0, g=0, b=0):

"""

Construct self such that it has the given red (r),

green (g), and blue (b) components.

"""

self.\_r = r # Red component

self.\_g = g # Green component

self.\_b = b # Blue component

#-------------------------------------------------------------------

def getRed(self):

"""

Return the red component of self.

"""

return self.\_r

#-------------------------------------------------------------------

def getGreen(self):

"""

Return the green component of self.

"""

return self.\_g

#-------------------------------------------------------------------

def getBlue(self):

"""

Return the blue component of self.

"""

return self.\_b

#-------------------------------------------------------------------

def \_\_str\_\_(self):

"""

Return the string equivalent of self, that is, a

string of the form '(r, g, b)'.

"""

#return '#%02x%02x%02x' % (self.\_r, self.\_g, self.\_b)

return '(' + str(self.\_r) + ', ' + str(self.\_g) + ', ' + \

str(self.\_b) + ')'

#-----------------------------------------------------------------------

# Some predefined Color objects:

WHITE = Color(255, 255, 255)

BLACK = Color( 0, 0, 0)

RED = Color(255, 0, 0)

GREEN = Color( 0, 255, 0)

BLUE = Color( 0, 0, 255)

CYAN = Color( 0, 255, 255)

MAGENTA = Color(255, 0, 255)

YELLOW = Color(255, 255, 0)

DARK\_RED = Color(128, 0, 0)

DARK\_GREEN = Color( 0, 128, 0)

DARK\_BLUE = Color( 0, 0, 128)

GRAY = Color(128, 128, 128)

DARK\_GRAY = Color( 64, 64, 64)

LIGHT\_GRAY = Color(192, 192, 192)

ORANGE = Color(255, 200, 0)

VIOLET = Color(238, 130, 238)

PINK = Color(255, 175, 175)

# Shade of blue used in Introduction to Programming in Java.

# It is Pantone 300U. The RGB values are approximately (9, 90, 166).

BOOK\_BLUE = Color( 9, 90, 166)

BOOK\_LIGHT\_BLUE = Color(103, 198, 243)

# Shade of red used in Algorithms 4th edition

BOOK\_RED = Color(150, 35, 31)

#-----------------------------------------------------------------------

def \_main():

"""

For testing:

"""

c1 = Color(128, 128, 128)

print(c1)

print(c1.getRed())

print(c1.getGreen())

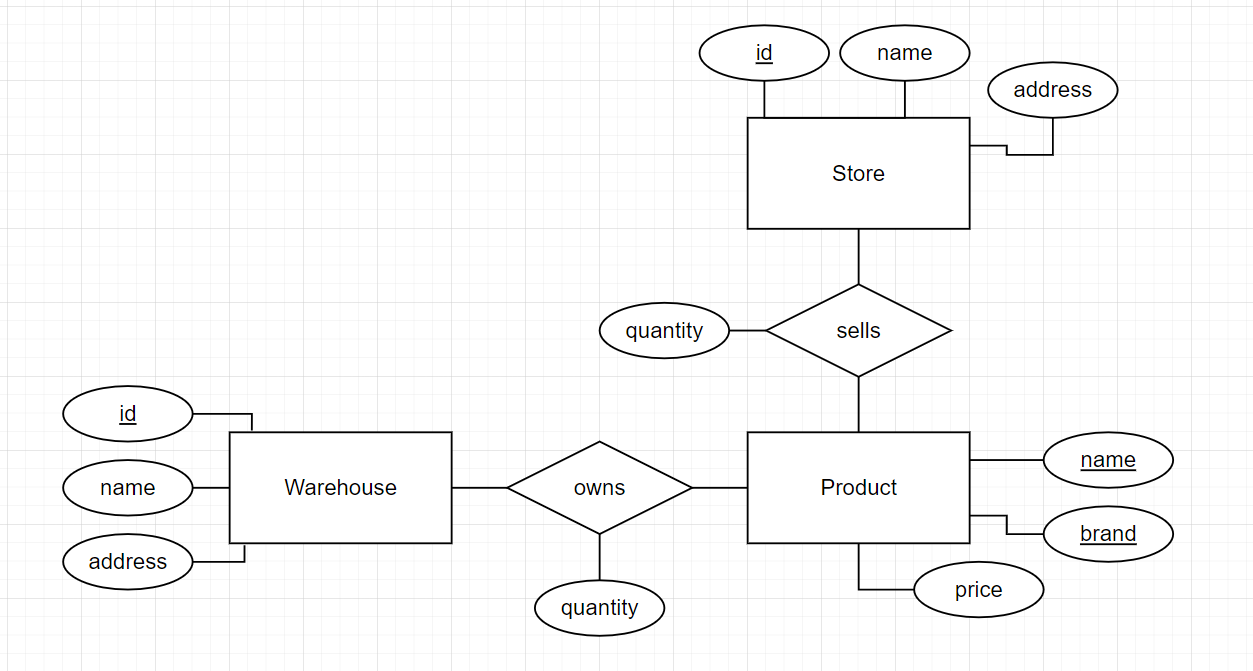
print(c1.getBlue())

if \_\_name\_\_ == '\_\_main\_\_':

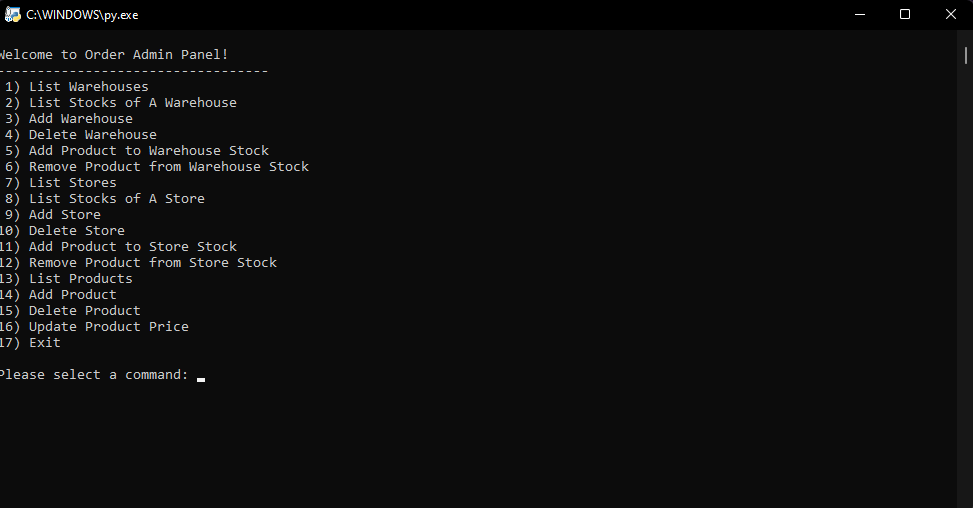
\_main()

All the explanations are writing under the codes every individual piece has own explanation.

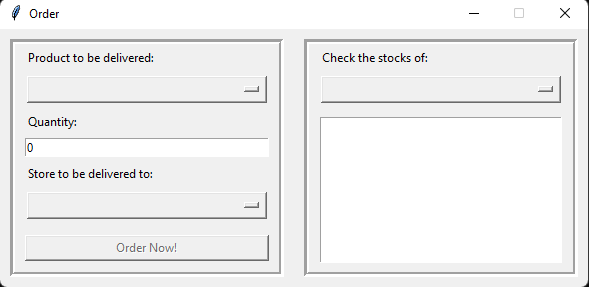
This Is UML For this project



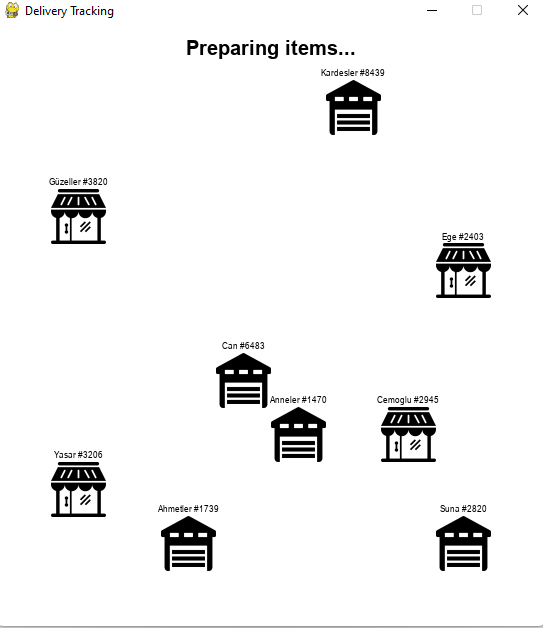
This is Control panel :

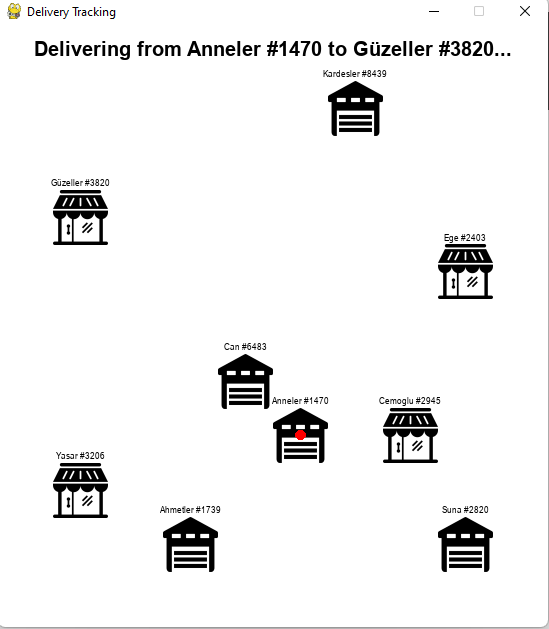


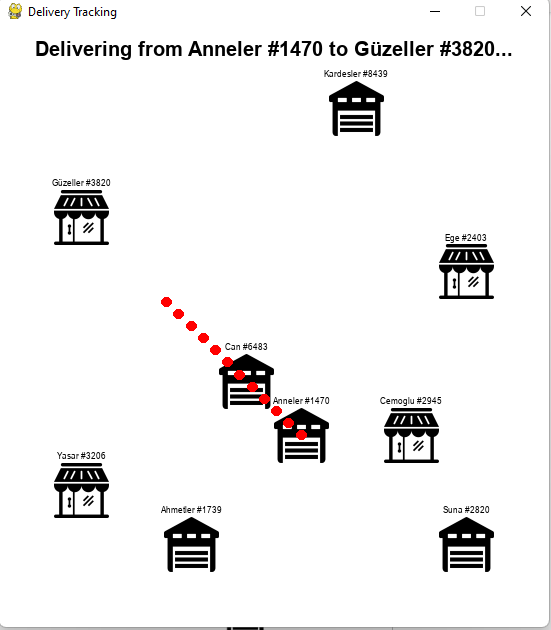
This is order panel:

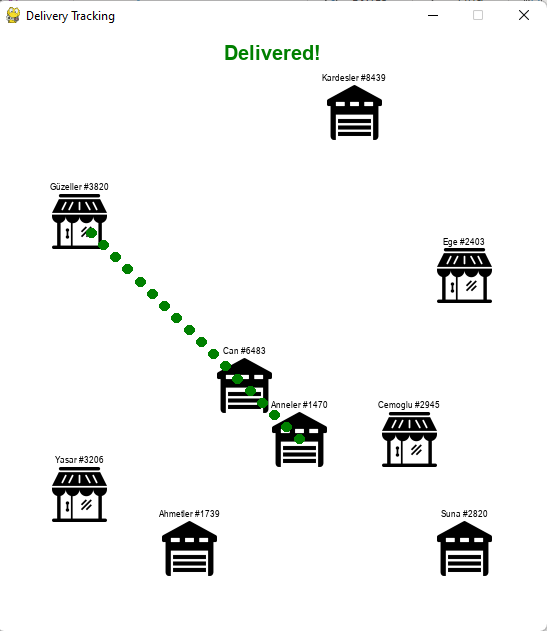


This is map while order is while delivering:









Cem Tuna ALTUNTAS 38304 L2