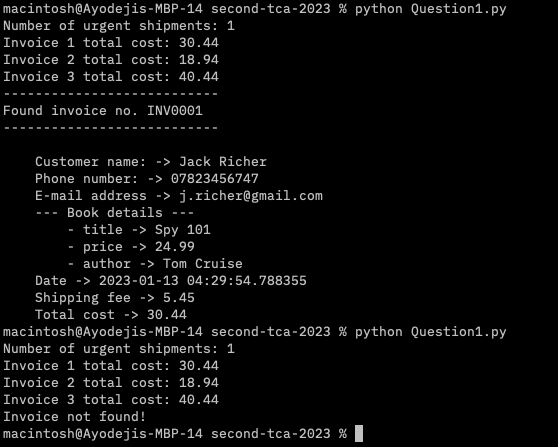
CSY1020 - TCA 2 - 2023

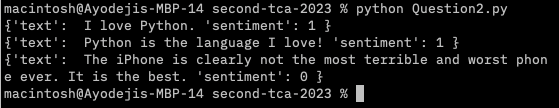
## Question 1

from datetime import datetime  
  
# ENUM to define shipping types cost  
NORMAL\_COST = 3.95  
URGENT\_COST = 5.45  
  
"""  
Every property has been set to private by default unless explicitly needed as this is good security practice (eg. in firewalls)  
"""  
  
class Customer:  
 def \_\_init\_\_(self, name: str, phone: str, email: str) -> None:  
 self.\_\_name: str = name  
 self.\_\_phone: str = phone  
 self.\_\_email: str = email  
  
 @property  
 def name(self):  
 return self.\_\_name  
  
 @name.setter  
 def name(self, val: str):  
 self.\_\_name = val  
  
 @property  
 def phone(self):  
 return self.\_\_phone  
  
 @phone.setter  
 def phone(self, val: str):  
 self.\_\_phone = val  
  
  
 @property  
 def email(self):  
 return self.\_\_email  
  
 @email.setter  
 def email(self, val: str):   
 self.\_\_email = val  
  
  
  
class Stock:  
 def \_\_init\_\_(self, book\_name: str, author: str, price: float) -> None:  
 self.\_\_book\_name: str = book\_name  
 self.\_\_author: str = author  
 self.\_\_price: float = price  
  
 @property  
 def book\_name(self):  
 return self.\_\_book\_name  
  
 @book\_name.setter  
 def book\_name(self, val: str):  
 self.\_\_book\_name = val  
  
 @property  
 def author(self):  
 return self.\_\_author  
  
 @author.setter  
 def author(self, val: str):  
 self.\_\_author = val  
  
 @property  
 def price(self):  
 return self.\_\_price  
  
 @price.setter  
 def price(self, val: float):  
 self.\_\_price = val  
  
  
  
class Order:  
 def \_\_init\_\_(self, customer: Customer, stock: Stock):  
 self.\_\_customer: Customer = customer  
 self.\_\_stock: Stock = stock  
  
 @property  
 def customer(self):  
 return self.\_\_customer  
  
 @customer.setter  
 def customer(self, val: Customer):  
 self.\_\_customer = val  
  
 @property  
 def stock(self):  
 return self.\_\_stock  
  
 @stock.setter  
 def stock(self, value):  
 self.\_\_stock = value  
  
  
  
  
class Shipping:  
 def \_\_init\_\_(self, order: Order, ship\_date: datetime) -> None:  
 self.\_\_order: Order = order  
 self.\_\_ship\_date: datetime = ship\_date  
 self.\_\_ship\_cost: float = 0.0  
 self.count\_urgent: int = 0  
  
 @property  
 def order(self):  
 return self.\_\_order  
  
 @property  
 def ship\_date(self):  
 return self.\_\_ship\_date  
   
 @property  
 def ship\_cost(self):  
 return self.\_\_ship\_cost  
  
 def set\_ship\_cost(self, val: float):  
 self.\_\_ship\_cost = val  
  
 def calc\_ship\_cost(self, is\_urgent: bool) -> float:  
 cost = NORMAL\_COST  
 if is\_urgent:  
 cost = URGENT\_COST  
 self.count\_urgent += 1  
 self.set\_ship\_cost(cost)  
 return cost  
  
  
  
class Invoice:  
 def \_\_init\_\_(self, invoice\_nbr: str, stock: Stock, ship\_order: Shipping) -> None:  
 self.\_\_invoice\_nbr: str = invoice\_nbr  
 self.\_\_stock: Stock = stock  
 self.\_\_ship\_order: Shipping = ship\_order  
 self.\_\_total\_cost: float = 0.0  
   
 @property  
 def invoice\_nbr(self):  
 return self.\_\_invoice\_nbr  
  
 def invoice(self):  
 shipping\_cost = self.\_\_ship\_order.ship\_cost  
 book\_cost = self.\_\_stock.price  
 total\_cost = shipping\_cost + book\_cost  
 self.\_\_total\_cost = total\_cost  
 return self.\_\_total\_cost  
  
 def \_\_str\_\_(self) -> str:  
 order = self.\_\_ship\_order.order  
 customer = order.customer  
 stock = order.stock  
 return f"""  
 Customer name: -> {customer.name}  
 Phone number: -> {customer.phone}  
 E-mail address -> {customer.email}  
 --- Book details ---  
 - title -> {stock.book\_name}  
 - price -> {stock.price}  
 - author -> {stock.author}  
 Date -> {self.\_\_ship\_order.ship\_date}  
 Shipping fee -> {self.\_\_ship\_order.ship\_cost}  
 Total cost -> {self.\_\_total\_cost:.2f}"""  
  
  
  
  
class BookStore:  
 def \_\_init\_\_(self):  
 self.invoices: list[Invoice] = []  
  
 def get\_invoices(self):  
 return self.invoices  
  
 def search\_invoice(self, nbr: str):  
 for invoice in self.invoices:  
 if invoice.invoice\_nbr == nbr:  
 print(f"---------------------------\nFound invoice no. {nbr}\n---------------------------")  
 print(invoice)  
 return  
  
 print("Invoice not found!")  
  
  
"""  
Test function to simulate various operations  
"""  
  
class Test:  
 def main(self):  
 customer1 = Customer("Jack Richer", "07823456747", "j.richer@gmail.com")  
 customer2 = Customer("Jack Ryan", "07126556919", "jackryan@protonmail.com")  
 customer3 = Customer("James Bond", "07123123784", "double007@outlook.com")   
   
   
 stock1 = Stock("Spy 101", "Tom Cruise", 24.99)  
 stock2 = Stock("Spy 102", "John Wick", 14.99)  
 stock3 = Stock("Spy 103", "Mary Jane", 34.99)  
   
   
 order1 = Order(customer1, stock1)  
 order2 = Order(customer2, stock2)  
 order3 = Order(customer3, stock3)  
   
   
 shipping1 = Shipping(order1, datetime.today())  
 shipping2 = Shipping(order2, datetime.today())  
 shipping3 = Shipping(order3, datetime.today())  
   
   
 shipping1.set\_ship\_cost(shipping1.calc\_ship\_cost(True))  
 shipping2.set\_ship\_cost(shipping2.calc\_ship\_cost(False))  
 shipping3.set\_ship\_cost(shipping3.calc\_ship\_cost(True))  
   
   
 invoice1 = Invoice("INV0001", stock1, shipping1)  
 invoice2 = Invoice("INV0002", stock2, shipping2)  
 invoice3 = Invoice("INV0003", stock3, shipping3)  
   
   
 bookstore = BookStore()  
   
 bookstore.invoices.append(invoice1)  
 bookstore.invoices.append(invoice2)  
 bookstore.invoices.append(invoice3)  
   
 print(f"Number of urgent shipments: {shipping1.count\_urgent}")  
 print(f"Invoice 1 total cost: {invoice1.invoice():.2f}")  
 print(f"Invoice 2 total cost: {invoice2.invoice():.2f}")  
 print(f"Invoice 3 total cost: {invoice3.invoice():.2f}")  
   
 bookstore.search\_invoice("INV0004")  
   
   
if \_\_name\_\_ == "\_\_main\_\_":  
 Test().main()



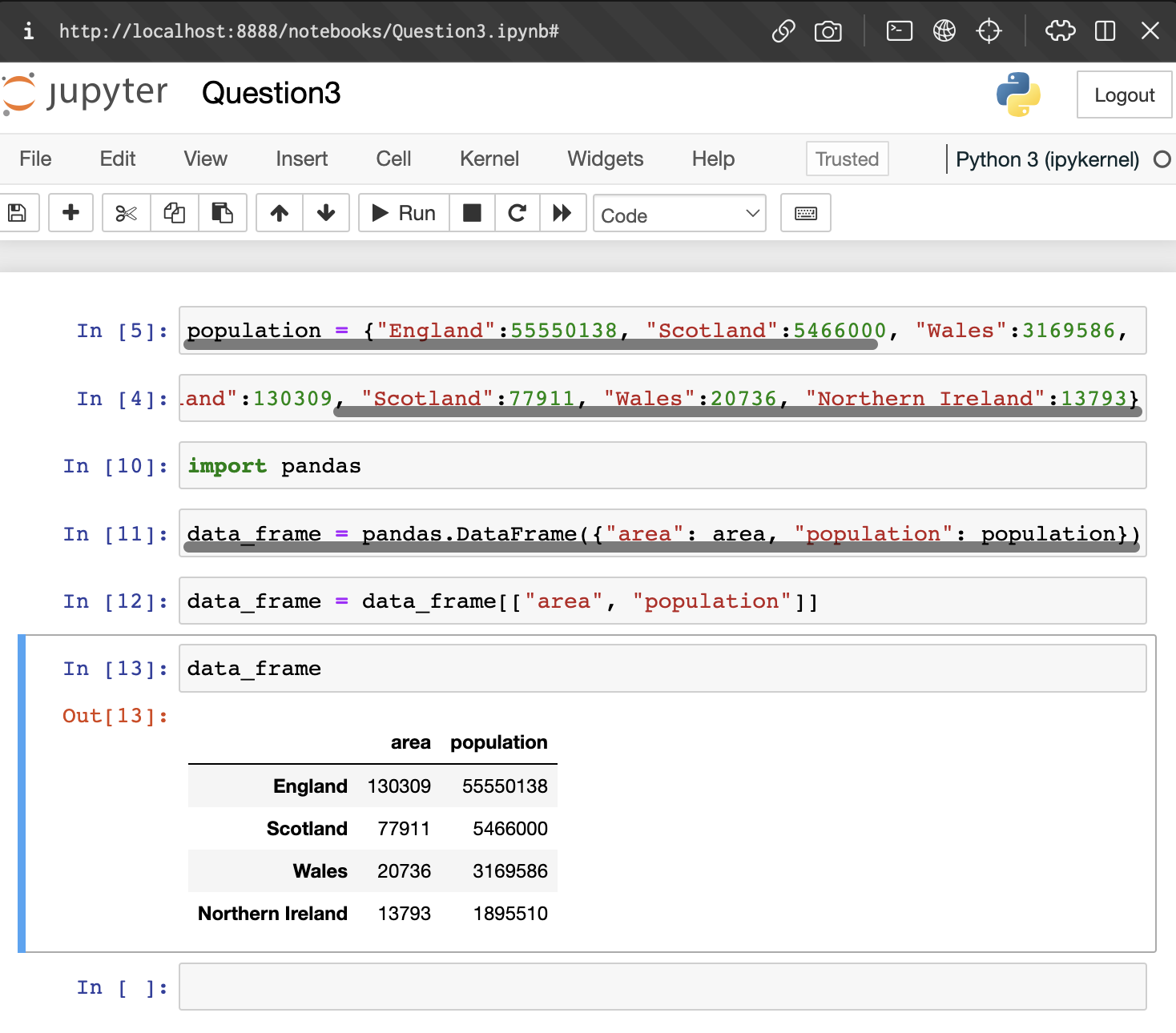
## Question 2

class SentimentLexicon:  
 def \_\_init\_\_(self):  
 self.dictionary: dict = self.load\_dictionary("./positive-words.txt", "./negative-words.txt")  
  
  
 def load\_file(self, src: str, sentiment\_val: int) -> dict:  
 # load a single file and return a dictionary that contains all the words that aren't commented out with a ;  
 result = {}  
 with open(src, "r") as file:  
 lines = file.readlines()  
 for line in lines:  
 if line.startswith(";") or line.strip() == "":  
 continue  
 result[line.lower().strip()] = sentiment\_val  
 return result  
   
  
 def load\_dictionary(self, pos\_src: str, neg\_src: str) -> dict:  
 # load all source files and return a mixed dictionary of good and bad words  
 pos\_dictionary = self.load\_file(pos\_src, 1)  
 neg\_dictionary = self.load\_file(neg\_src, -1)  
 return { \*\*pos\_dictionary, \*\*neg\_dictionary }  
  
  
class Classifier:  
 def \_\_init\_\_(self):  
 self.sentiment\_lexicon = SentimentLexicon()  
  
  
 def split\_txt(self, txt: str):  
 # This splits the sentence and removes pre-defined punctuations to make it easier to match with sentiment dictionary values   
 txt\_array = txt.lower().strip().split()  
 punctuations = [",", ".", "!"]  
 result = []  
 for word in txt\_array:  
 wrd\_stripped = word.strip()  
 for punctuation in punctuations:  
 if wrd\_stripped.startswith(punctuation) or wrd\_stripped.endswith(punctuation):  
 wrd\_stripped = wrd\_stripped.strip(punctuation)  
  
 result.append(wrd\_stripped)  
  
 return result  
  
  
  
 def classify(self, txt: str) -> int:  
 dictionary = self.sentiment\_lexicon.dictionary  
 total = 0  
 txt\_array = self.split\_txt(txt)  
  
 # check if word exists in the dictionary and add the value to the total if it exists, else add 0  
 for word in txt\_array:  
 score = dictionary[word] if word in dictionary else 0  
 total += score  
   
 # check if total is 0, positive or negative and return the right sentiment rating (1, -1 or 0)  
 sentiment = 0 if total == 0 else 1 if total > 0 else -1  
  
 return sentiment  
  
  
  
  
"""  
Main program to simulate a couple of sentiment analysis   
"""  
  
def main():  
 classifier = Classifier()  
 sentences = ["I love Python.", "Python is the language I love!", "The iPhone is clearly not the most terrible and worst phone ever. It is the best."]  
  
 # classify test cases and print them out  
 for sentence in sentences:  
 sentiment = classifier.classify(sentence)  
 print("{'text': ",sentence, "'sentiment':",sentiment, "}")  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()



## Question 3

{  
 "cells": [  
 {  
 "cell\_type": "code",  
 "execution\_count": 5,  
 "id": "a331687b",  
 "metadata": {},  
 "outputs": [],  
 "source": [  
 "population = {\"England\":55550138, \"Scotland\":5466000, \"Wales\":3169586, \"Northern Ireland\":1895510}"  
 ]  
 },  
 {  
 "cell\_type": "code",  
 "execution\_count": 4,  
 "id": "d35277aa",  
 "metadata": {},  
 "outputs": [],  
 "source": [  
 "area = {\"England\":130309, \"Scotland\":77911, \"Wales\":20736, \"Northern Ireland\":13793}"  
 ]  
 },  
 {  
 "cell\_type": "code",  
 "execution\_count": 10,  
 "id": "88016aef",  
 "metadata": {},  
 "outputs": [],  
 "source": [  
 "import pandas"  
 ]  
 },  
 {  
 "cell\_type": "code",  
 "execution\_count": 11,  
 "id": "fcfa9acc",  
 "metadata": {},  
 "outputs": [],  
 "source": [  
 "data\_frame = pandas.DataFrame({\"area\": area, \"population\": population})"  
 ]  
 },  
 {  
 "cell\_type": "code",  
 "execution\_count": 12,  
 "id": "b3b90f17",  
 "metadata": {},  
 "outputs": [],  
 "source": [  
 "data\_frame = data\_frame[[\"area\", \"population\"]]"  
 ]  
 },  
 {  
 "cell\_type": "code",  
 "execution\_count": 13,  
 "id": "0f60c63c",  
 "metadata": {},  
 "outputs": [  
 {  
 "data": {  
 "text/html": [  
 "<div>\n",  
 "<style scoped>\n",  
 " .dataframe tbody tr th:only-of-type {\n",  
 " vertical-align: middle;\n",  
 " }\n",  
 "\n",  
 " .dataframe tbody tr th {\n",  
 " vertical-align: top;\n",  
 " }\n",  
 "\n",  
 " .dataframe thead th {\n",  
 " text-align: right;\n",  
 " }\n",  
 "</style>\n",  
 "<table border=\"1\" class=\"dataframe\">\n",  
 " <thead>\n",  
 " <tr style=\"text-align: right;\">\n",  
 " <th></th>\n",  
 " <th>area</th>\n",  
 " <th>population</th>\n",  
 " </tr>\n",  
 " </thead>\n",  
 " <tbody>\n",  
 " <tr>\n",  
 " <th>England</th>\n",  
 " <td>130309</td>\n",  
 " <td>55550138</td>\n",  
 " </tr>\n",  
 " <tr>\n",  
 " <th>Scotland</th>\n",  
 " <td>77911</td>\n",  
 " <td>5466000</td>\n",  
 " </tr>\n",  
 " <tr>\n",  
 " <th>Wales</th>\n",  
 " <td>20736</td>\n",  
 " <td>3169586</td>\n",  
 " </tr>\n",  
 " <tr>\n",  
 " <th>Northern Ireland</th>\n",  
 " <td>13793</td>\n",  
 " <td>1895510</td>\n",  
 " </tr>\n",  
 " </tbody>\n",  
 "</table>\n",  
 "</div>"  
 ],  
 "text/plain": [  
 " area population\n",  
 "England 130309 55550138\n",  
 "Scotland 77911 5466000\n",  
 "Wales 20736 3169586\n",  
 "Northern Ireland 13793 1895510"  
 ]  
 },  
 "execution\_count": 13,  
 "metadata": {},  
 "output\_type": "execute\_result"  
 }  
 ],  
 "source": [  
 "data\_frame"  
 ]  
 },  
 {  
 "cell\_type": "code",  
 "execution\_count": null,  
 "id": "6b08e25c",  
 "metadata": {},  
 "outputs": [],  
 "source": []  
 }  
 ],  
 "metadata": {  
 "kernelspec": {  
 "display\_name": "Python 3 (ipykernel)",  
 "language": "python",  
 "name": "python3"  
 },  
 "language\_info": {  
 "codemirror\_mode": {  
 "name": "ipython",  
 "version": 3  
 },  
 "file\_extension": ".py",  
 "mimetype": "text/x-python",  
 "name": "python",  
 "nbconvert\_exporter": "python",  
 "pygments\_lexer": "ipython3",  
 "version": "3.10.7"  
 }  
 },  
 "nbformat": 4,  
 "nbformat\_minor": 5  
}



## Question 4

import pandas  
  
data = pandas.read\_excel("./food-data.xlsx")  
  
  
"""  
Calculate average price increase between September 2021 and 2022  
To understand how inflation has affected the price of these items  
"""  
avg\_yearly\_increase = (data["Price in September 2022 pence)"].mean() - data["Price in September 2021 (pence)"].mean()) / data["Price in September 2021 (pence)"].mean() \* 100  
print(f"-- Average yearly increase: {avg\_yearly\_increase:.2f}%")  
  
  
  
"""  
Top 3 most expensive food items in September 2021   
To understand which items had the highest profit margins or were considered "better" or as "luxury" in this time frame (2021)  
"""  
most\_expensive\_2021 = data.nlargest(3, "Price in September 2021 (pence)")  
print("-- Top 3 expensive food items in 2021\n", most\_expensive\_2021)  
  
  
  
"""  
Top 3 most expensive food items in September 2022  
To understand which items had the highest profit margins or were considered "better" or as "luxury" in this time frame (2022)  
"""  
most\_expensive\_2022 = data.nlargest(3, "Price in September 2022 pence)")  
print("-- Top 3 expensive food items in 2022\n", most\_expensive\_2022)  
  
  
  
"""  
Item that has the highest price increase percentage between September 2021 and September 2022  
To identify which item is most affected by price inflation  
"""  
highest\_price\_increase = (data['Price in September 2022 pence)'] - data['Price in September 2021 (pence)']) / data['Price in September 2021 (pence)'] \* 100  
highest\_price\_increase\_item = data.loc[highest\_price\_increase.idxmax()]['Item']  
print(f"-- The food item that has the highest price increase percentage between September 2021 and September 2022 is: {highest\_price\_increase\_item}")  
  
  
  
"""  
Item that has the highest price decrease percentage between September 2021 and September 2022  
To identify which item is least affected by price inflation  
"""  
highest\_price\_decrease = (data['Price in September 2022 pence)'] - data['Price in September 2021 (pence)']) / data['Price in September 2021 (pence)'] \* 100  
highest\_price\_decrease\_item = data.loc[highest\_price\_decrease.idxmin()]['Item']  
print(f"-- The food item that has the highest price decrease percentage between September 2021 and September 2022 is: {highest\_price\_decrease\_item}")  
  
  
  
"""  
The average price increase of each food item between September 2021 and September 2022  
To get an in-depth look into how each item was affected by price inflation (negatively and positively)   
"""  
price\_increase\_by\_items = (data.groupby('Item')['Price in September 2022 pence)'].mean() - data.groupby('Item')['Price in September 2021 (pence)'].mean()) / data.groupby('Item')['Price in September 2021 (pence)'].mean() \* 100  
print("-- Average price increase of each food item between September 2021 and September 2022\n", price\_increase\_by\_items)

