Multi-Paradigm Programming

Shop Assignment

You are tasked to add some additional functionality to the base code in files :

References

https://stackoverflow.com/questions/26949755/pass-pandas-dataframe-into-class

Functionality

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| The shop CSV should hold the initial cash value for the shop.  See file called **stock.csv**   |  |  |  | | --- | --- | --- | | **1000.30** |  |  | | **Coke Can** | 1.10 | 100 | | **Bread** | 0.7 | 30 | | **Spaghetti** | 1.20 | 100 | | **Tomato Sauce** | 0.80 | 100 | | **Big Bags** | 2.50 | 4 | |
| Read in customer orders from a CSV file.  from dataclasses import dataclass, field  from typing import List  import csv  """-------------------------------------------------------"""  #These classes are needed for the customer class to work  @dataclass  class Product:  name: str  price: float = 0.0  @dataclass  class ProductStock:  product: Product  quantity: int  """-------------------------------------------------------"""  @dataclass  class Customer:  name: str = ""  budget: float = 0.0  shopping\_list: List[ProductStock] = field(default\_factory=list)  def read\_customer(file\_path):  with open(file\_path) as csv\_file:  csv\_reader = csv.reader(csv\_file, delimiter=',')  first\_row = next(csv\_reader)  c = Customer(first\_row[0], float(first\_row[1]))  for row in csv\_reader:  name = row[0]  quantity = float(row[1])  p = Product(name)  ps = ProductStock(p, quantity)  c.shopping\_list.append(ps)  return c  """----------------------------------------------------------"""  # This function is needed for customer to work  def print\_product(p):  print(f'\nPRODUCT NAME: {p.name} \nPRODUCT PRICE: {p.price}')  """----------------------------------------------------------"""  def print\_customer(c):  print(f'CUSTOMER NAME: {c.name} \nCUSTOMER BUDGET: {c.budget}')    for item in c.shopping\_list:  print\_product(item.product)    print(f'{c.name} ORDERS {item.quantity} OF ABOVE PRODUCT')  cost = item.quantity \* item.product.price  print(f'The cost to {c.name} will be €{cost}')  c = read\_customer("../customer.csv")  print\_customer(c) |
| That file should include all the products they wish to buy and in what quantity.  It should also include their name and their budget.  See file called **customer.csv**   |  |  | | --- | --- | | **John** | **100.20** | | **Coke Can** | 10 | | **Bread** | 3 | | **Jam** | 1 | |
| The shop must be able to process the orders of the customer. Update the cash in the shop based on money received. |
| It is important that the state of the shop be consistent. |
| You should create customer test filles (CSVs) which cannot be completed by the shop e.g. customer wants 400 loaves of bread but the shop only has 20, or the customer wants 2 cans of coke but can only afford 1.  If these files don’t exist marks penalties will be applied. Know whether or not the shop can fill an order.  See files called  **customer\_too\_much\_bread.csv**  **customer\_not\_enough\_money.csv** |
| Thrown an appropriate error. |
| Operate in a live mode, where the user can enter a product by name, specify a quantity, and pay for it. The user should be able to buy many products in this way. |

BASE CODE PYTHON

|  |
| --- |
| from dataclasses import dataclass, field  from typing import List  import csv  @dataclass  class Product:  name: str  price: float = 0.0  @dataclass  class ProductStock:  product: Product  quantity: int  @dataclass  class Shop:  cash: float = 0.0  stock: List[ProductStock] = field(default\_factory=list)  @dataclass  class Customer:  name: str = ""  budget: float = 0.0  shopping\_list: List[ProductStock] = field(default\_factory=list)  def create\_and\_stock\_shop():  s = Shop()  with open('../stock.csv') as csv\_file:  csv\_reader = csv.reader(csv\_file, delimiter=',')  first\_row = next(csv\_reader)  s.cash = float(first\_row[0])  for row in csv\_reader:  p = Product(row[0], float(row[1]))  ps = ProductStock(p, float(row[2]))  s.stock.append(ps)  #print(ps)  return s    def read\_customer(file\_path):  with open(file\_path) as csv\_file:  csv\_reader = csv.reader(csv\_file, delimiter=',')  first\_row = next(csv\_reader)  c = Customer(first\_row[0], float(first\_row[1]))  for row in csv\_reader:  name = row[0]  quantity = float(row[1])  p = Product(name)  ps = ProductStock(p, quantity)  c.shopping\_list.append(ps)  return c    def print\_product(p):  print(f'\nPRODUCT NAME: {p.name} \nPRODUCT PRICE: {p.price}')  def print\_customer(c):  print(f'CUSTOMER NAME: {c.name} \nCUSTOMER BUDGET: {c.budget}')    for item in c.shopping\_list:  print\_product(item.product)    print(f'{c.name} ORDERS {item.quantity} OF ABOVE PRODUCT')  cost = item.quantity \* item.product.price  print(f'The cost to {c.name} will be €{cost}')    def print\_shop(s):  print(f'Shop has {s.cash} in cash')  for item in s.stock:  print\_product(item.product)  print(f'The Shop has {item.quantity} of the above')  #s = create\_and\_stock\_shop()  #print\_shop(s)  c = read\_customer("../customer.csv")  print\_customer(c) |

Notes

The above described functionality should be completed in Python and C. This is to be done in a procedural programming style.

The live mode, and the input files, should have the exact same behaviour in ALL implementations.

For example I should be able to use the Python implementation in the same way as the C one i.e. same CSV files, and the same process when doing an order in live mode.

The user experience of each implementation should be identical.

Marking Scheme

Python Program (20%)

Good Procedural Programming (NOT OOP) (10%)

Level of functionality (5%)

Live Mode (5%)

C Program (30%)

Good Procedural Programming (10%)

Level of functionality (10%)

Live Mode (10%)