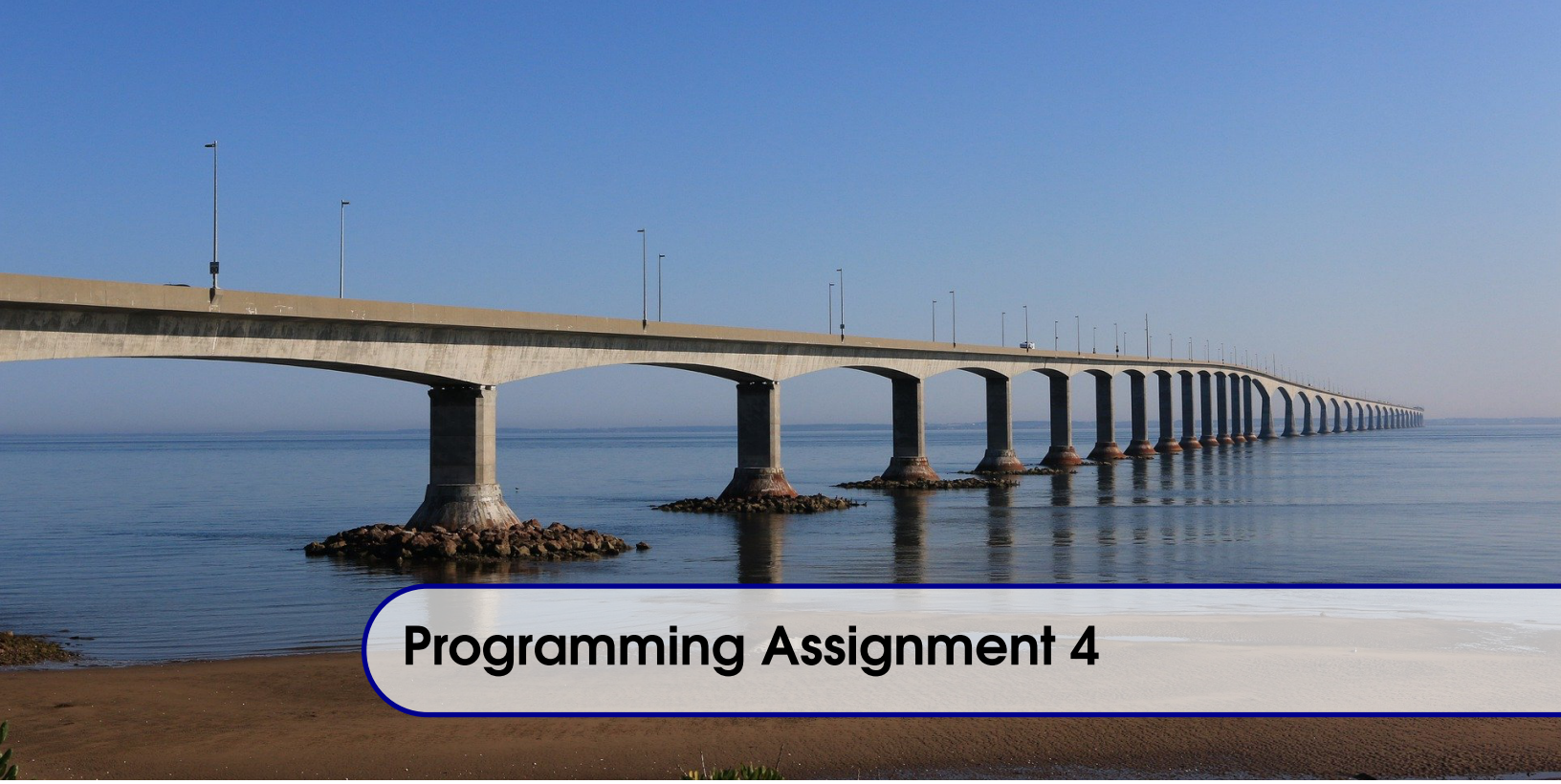

Programming Assignment 4

Assignment Objectives

By completing this assignment you are demonstrating your ability to:

- Design your own related set of classes in the Python programming language.
- Create a number of different classes with a variety of relationships.
- Read structured tabular data using the `csv` module.
- Use a `deque` object for storing and performing operations on a collection of objects.

Scenario

In the previous assignment, you worked on a Product class, a Store class, and a myStore class. In this assignment, you will be working on something similar but using stacks. You are given a Product class that will store information about the products, and a Store class that will store information about the stores. You are also given a Stack class that implements a stack using the deque module. Your objective is to design a StackStorage class that is going to create a database of your store's branches and products.

You need to use the given Stack class. You will not use the deque module or any other stack implementation in this assignment. Failure to follow this instruction will result in 0 mark on the assignment

Given

As part of this assignment, you have been given the following classes:

Product class

The Product class's constructor receives the following product information: store_id(string), product id (string), product name (string), products sold (int), products available (int), product cost (float), and product retail price (float). The *__str__* method represents the product information. In addition, the Product class has the following methods:
get_id():String - returns the product id.
get_name():String - returns the product name.
get_num_sold():String - returns the number of products sold.
get_num_available():String - returns the number of products available.
get_cost():String - returns the cost of the product.
get_price():String - returns the retail price of the product

Store class

The Store class's constructor receives the following store information: store id(string), store(branch) name (string), store city (string), store area (string). Each store maintains a dictionary of products where unique product ids act as keys and instances of product class act as values. The *__str__* method represents the store information. In addition, the Store class has the following methods:
get_store_id():String - returns the store id.
get_store_name():String - returns the store(branch) name.
get_city():String - returns the city name.
get_area():String - returns the store area.
get_products() - returns a dictionary of products.
add_product(pid, product_name, sold, available, cost, price):String - creates a new Product object and add it to the product dictionary.
remove_product(pid):String - removes the product with pid as its product id from the product dictionary.
count_products():Integer - returns the number of products in the product dictionary.

Stack class

The Stack class's constructor initializes a deque object. The Stack class has the following methods:
pop_element() - removes and returns the top element of the stack.
push_element(input_element) - adds the input element to the stack.
top_element - returns the top element of the stack.
stack_size():Integer - returns the number of elements in the stack.
print_stack(order:Integer) - print the stack elements from top-to-bottom if the order is 0, and from bottom-to-top if the order is 1.

asn4Inventory.csv

This supporting file contains the database of stores and products. Driver code is going to read this file and pass the data as a list to the constructor of the StackStorage class. Any row may contain the store data or the product data. There are 4 columns with labels: RecordID, Branch_or_StoreRecordID, City_or_Sold-Available, Area_or_Cost-RetailPrice. If "RecordID" starts with "S' then the respective row contains the store data (RecordID i.e. store ID, BranchName, City, Area - all as strings). If "RecordID" starts with "P' then the respective row contains the product data (RecordID i.e. product ID as a string, StoreID as a string, products sold and products available as a compound string separated by "-', price and retail cost as a compound string separated by "-').

Objective - StackStorage class

You need to write a StackStorage class that maintains the database of all stores and products. As part of this assignment, you have been given a StackStorage class with driver code. You need to write your code in the given StackStorage class. It must contain the following:

1. The class constructor will receive a list *dataList* that contains the rows of CSV file as its elements. So, *dataList* is a list of list elements. The constructor will initialize a new variable *dlist* using the input *dataList*. It will also initialize a new empty stack *store_obj_stack* using the given Stack class.
2. The *get_store_obj_stack* method will return the stack *store_obj_stack*.
3. The *store_in_stack* method will receive a *str_id* and check if a store with the same store id exists in the stack *store_obj_stack* or not. If such a store exists in the stack *store_obj_stack*, then it will return True, otherwise it will return False. Your original stack *store_obj_stack* should remain intact in the end i.e. it should have all the elements in the same order as it had at the start of this method.
4. The *build_stack* method will process the elements of *dlist* (one at a time) to add new stores and products to the stack *store_obj_stack* using the following instructions:
    - You will check if the list element represents the Store data or the Product data (*hint: check if the RecordID starts with "S" or "P"*).
    - If the list element represents the Store data, then create a Store object (using the Store class). If the store with the given store_id does not already exists in the stack *store_obj_stack*, then add this new store object to the stack. You may use *store_in_stack* method to check if the store object exists or not.
    - If the list element represents the Product data, then you will add the Product object to its respective store object. You will need to split the composite strings to get the values of the product attributes (*see asn4Inventory.csv section for more details*). If the store with the given store id does not exist, then you will print "*Store ID XX does not exist! Adding Product is not possible!*", where *XX* is the store id given in the Product data. If the store with the given store id already exists in the stack *store_obj_stack*, then you will add the product to its respective store object using the *add_product()* method of the Store class. You need to retrieve the store object with the give store id from the original stack *store_obj_stack* for adding a new product object to this store, but this should not alter the original order of store objects in the stack. To be clear, you may remove some store objects and then add them back to the stack to maintain the original order.
    - In the end, the constructor will print "*Constructor added XX store objects to the*

stack store_obj_stack!", where *XX* is the number of store objects added to the stack *store_obj_stack*.

5. The *get_odd_even_store_stacks* method will return two stacks, first that contains the store objects with odd store ids and the second that contains the store objects with even store ids. Please note that any given store id can be represented as a string of the format *Sxxx*, where the first letter is 'S' followed by a number of any length. If this number is even (odd), then you will consider the store id as even (odd). In this method, you will go through the stack elements (store objects), check if their store ids are even or odd, add the store objects to the even or odd stacks based on their ids, and return both stacks in the end. Your original stack *store_obj_stack* should remain intact in the end i.e. it should have all the elements in the same order as it had at the start of this method.
6. The *build_products_stack* method will receive a list of store ids and return a stack of products associated with the respective stores. You will go through the stack elements (store objects) and retrieve their products using the *get_products()* method. If the store object has its id in the input list of store ids, then you will add all products associated with that store object to the stack of products. Your original stack *store_obj_stack* should remain intact in the end i.e. it should have all the elements in the same order as it had at the start of this method.
7. You should provide appropriate docstrings and in-line commenting for your implemented methods.

Submitting your assignment

Your submission will consist of one zip file containing solutions to all problems (see below). This file will be submitted via a link provided on our Moodle page just below the assignment description. This file must be uploaded by 5pm (Moodle time) on the due date in order to be accepted. You can submit your solution any number of times prior to the cutoff time (each upload overwrites the previous one). Therefore, you can (and should) practice uploading your solution and verifying that it has arrived correctly.

Example:

Submission file: `asn4_studentnum.zip` , where `studentnum` is your student ID number.

Your zip file should contain a single Python file `StackStorage.py`, and a pdf file with the screenshots of a single sample run of your code. Please do not include the screenshots as separate image files. You need to combine the screenshots to a single pdf file. Marks may be deducted for incorrect file formats.

Driver Script and Sample Run

The *StackStorage.py* file has been provided as part of this assignment. This file includes the necessary driver code. You should write your solution in the given *StackStorage.py* file. A sample run of the completed code is shown below:

```
********************************************************************************
Printing Stack of Store objects:
No elements found!
********************************************************************************
Building Stack of Store objects...
Store ID S1010 does not exist! Adding Product is not possible!
Store ID S1015 does not exist! Adding Product is not possible!
Constructor added 11 store objects to the stack store_obj_stack!
********************************************************************************
Printing Stack of Store objects:
Stack elements - from top to bottom:
StoreID:S10111, Branch:Xmall, City:Jiukeng, Area:64 Johns st
StoreID:S10101, Branch:Downtown, City:Munich, Area:14 Mayfield Alley
StoreID:S1009, Branch:SouthPark, City:Cercal, Area:4 Charing Cross Plaza
StoreID:S1008, Branch:HaltonHills, City:Purral, Area:4496 Killdeer Court
StoreID:S1007, Branch:LoganPlaza, City:Bov, Area:89058 Monument Hill
StoreID:S1006, Branch:VictoriaPark, City:Munich, Area:24 May St
StoreID:S1005, Branch:HydePark, City:Ettelbruck, Area:84269 Crescent Oaks Way
StoreID:S1004, Branch:Downtown, City:Hekou, Area:5995 Merchant Trail
StoreID:S1003, Branch:CityPlaza, City:Jiukeng, Area:634 Jackson Parkway
StoreID:S1002, Branch:Uptown, City:Kangar, Area:689 Vera Pass
StoreID:S1001, Branch:Downtown, City:Gerakaroú, Area:5434 Daystar Circle
********************************************************************************
Printing Stack of Store objects with odd store ids:
Stack elements - from bottom to top:
StoreID:S10111, Branch:Xmall, City:Jiukeng, Area:64 Johns st
StoreID:S10101, Branch:Downtown, City:Munich, Area:14 Mayfield Alley
StoreID:S1009, Branch:SouthPark, City:Cercal, Area:4 Charing Cross Plaza
StoreID:S1007, Branch:LoganPlaza, City:Bov, Area:89058 Monument Hill
StoreID:S1005, Branch:HydePark, City:Ettelbruck, Area:84269 Crescent Oaks Way
StoreID:S1003, Branch:CityPlaza, City:Jiukeng, Area:634 Jackson Parkway
StoreID:S1001, Branch:Downtown, City:Gerakaroú, Area:5434 Daystar Circle
********************************************************************************
Printing Stack of Store objects with even store ids:
Stack elements - from bottom to top:
StoreID:S1008, Branch:HaltonHills, City:Purral, Area:4496 Killdeer Court
StoreID:S1006, Branch:VictoriaPark, City:Munich, Area:24 May St
StoreID:S1004, Branch:Downtown, City:Hekou, Area:5995 Merchant Trail
StoreID:S1002, Branch:Uptown, City:Kangar, Area:689 Vera Pass
********************************************************************************
```

Figure 1: Sample run - part 1

```
********************************************************************************
Printing stack of products for a given list of store ids:
Store ids: ['S10101', 'S1008', 'S1005', 'S1004', 'S1003']
Stack elements - from bottom to top:
StoreID: S10101;Product Name:Chocolate Bar ; Product ID:P2001; Sold:140; Available:97; Cost:2773.17; RetailPrice:2508.13
StoreID: S1008;Product Name:Juice; Product ID:P2007; Sold:231; Available:422; Cost:3008.33; RetailPrice:2900.32
StoreID: S1008;Product Name:Hummus; Product ID:P2003; Sold:421; Available:23; Cost:1783.66; RetailPrice:1672.12
StoreID: S1005;Product Name:Rootbeer; Product ID:P2002; Sold:654; Available:823; Cost:503.81; RetailPrice:489.65
StoreID: S1005;Product Name:Chocolate Bar ; Product ID:P2001; Sold:763; Available:231; Cost:748.47; RetailPrice:678.45
StoreID: S1005;Product Name:Soda; Product ID:P2005; Sold:323; Available:32; Cost:1896.99; RetailPrice:1232.12
StoreID: S1004;Product Name:Milk; Product ID:P2008; Sold:32; Available:94; Cost:1433.39; RetailPrice:1400.87
StoreID: S1003;Product Name:Chocolate Bar ; Product ID:P2001; Sold:400; Available:34; Cost:3758.18; RetailPrice:3409.12
StoreID: S1003;Product Name:Bread; Product ID:P2010; Sold:321; Available:94; Cost:4479.93; RetailPrice:4232.67
StoreID: S1003;Product Name:Rootbeer; Product ID:P2002; Sold:217; Available:876; Cost:2243.27; RetailPrice:2100.76
********************************************************************************
Printing to check if the Original Stack of Store objects is intact:
Stack elements - from top to bottom:
StoreID:S10111, Branch:Xmall, City:Jiukeng, Area:64 Johns st
StoreID:S10101, Branch:Downtown, City:Munich, Area:14 Mayfield Alley
StoreID:S1009, Branch:SouthPark, City:Cercal, Area:4 Charing Cross Plaza
StoreID:S1008, Branch:HaltonHills, City:Purral, Area:4496 Killdeer Court
StoreID:S1007, Branch:LoganPlaza, City:Bov, Area:89058 Monument Hill
StoreID:S1006, Branch:VictoriaPark, City:Munich, Area:24 May St
StoreID:S1005, Branch:HydePark, City:Ettelbruck, Area:84269 Crescent Oaks Way
StoreID:S1004, Branch:Downtown, City:Hekou, Area:5995 Merchant Trail
StoreID:S1003, Branch:CityPlaza, City:Jiukeng, Area:634 Jackson Parkway
StoreID:S1002, Branch:Uptown, City:Kangar, Area:689 Vera Pass
StoreID:S1001, Branch:Downtown, City:Gerakaroú, Area:5434 Daystar Circle
********************************************************************************
```

Figure 2: Sample run - part 2.