Assessment Information

In this assessment, you are required to complete the interactive program to be installed in a stamp-selling machine in Post Office. This machine sells postage stamps for two types of items (letter and parcel).

Software inputs and output

Inputs into your program will include:

- The price guides (as csv files) for letter and parcel
  - Economy Letters Price Guide ($).csv
  - Economy Parcel Price Guide_by Sea ($).csv
- The country-zone mapping csv file
  - Countries and Zones.csv

Each of these price guides will contain the price information for the postage type upon different item weights and destination zones. Note that these files should be loaded in a proper data structure when your software starts.

- The details of the items to be posted provided by users, including type (letter or parcel), weight, destination countries, quantities etc. You will need to design a clear and interactive way to acquire inputs from users.

- The pre-stored sales history. You will be given a pre-stored sales history file "**sales_history.csv**", which contains the history of stamp sales. Please refer to the section of **Functional requirements** regarding the format & description of this file.

Outputs from your program include:

- Upon each customer completed his/her shopping, your program will need to:
    - save the **invoice** along with the list of purchased stamps into a .txt file (refer to the section of Functional requirements).
    - append the purchased items to the sales record "**sales_history.csv**" (refer to the section of Functional requirements).
- Information and guidance to navigate and assist users during shopping.

Functional requirements

Your program should support continuous shopping activities in Post Office. If you demand the next user to restart the program each time they do shopping, your program does not fulfill this requirement. However, you could assume that it serves only one customer at a time, i.e., the next customer will only start shopping after the previous customer checkouts or cancel his/her shopping.

This program should provide the following functionalities as a stamp-selling machine.

- [Fun 1] Add an item to the shopping cart

Your program should allow the user to add items to his/her shopping cart. You will need to design an interactive way to acquire details of each item that he or she wants to purchase (such as type of post, e.g., letter or parcel, its weight, destination country, etc.).

[Challenge point] Available methods and pricing information

You will need to provide proper guidance for customers during their shopping experience. For example, once a customer submits the detail of a parcel, e.g., weight, destination country, you should let know the available methods that they can choose (e.g., letter or parcel, etc.) and the associated pricing.

[Challenge point] Checking duplicate item entry

If a user tries to add a duplicate item into the shopping cart, your program should prompt the user to confirm whether or not he/she wants to continue this action.

- [Fun 2] View shopping cart

Users should be able to view their shopping cart. Once requested, your program should print out **the list of items & costs**, as well as the **total cost of the items** in the shopping cart.

- [Fun 3] Amend an item

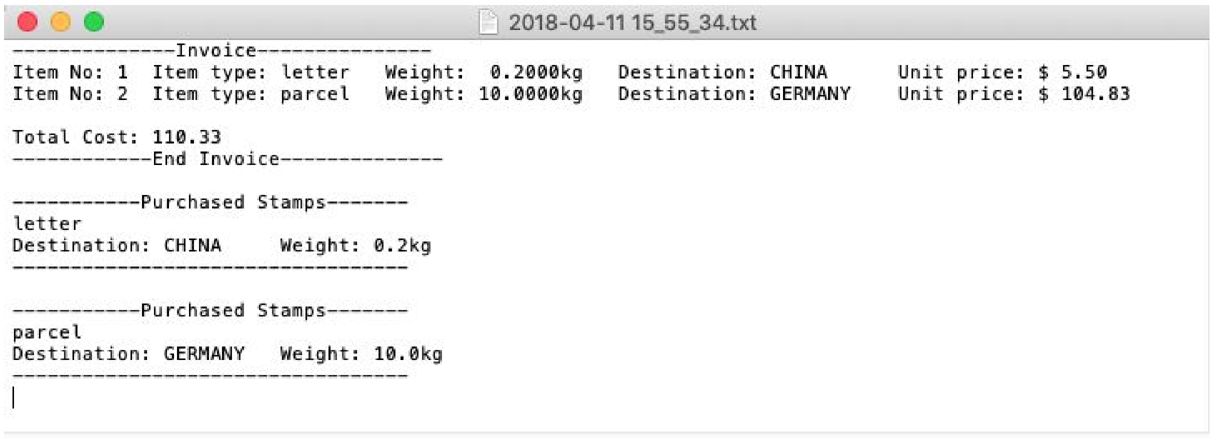
Your program should allow the user to amend an item in the shopping cart, **limited to its weight only.**

- [Fun 4] Remove an item from the shopping cart

Your program should allow the user to **delete** an item in the shopping cart. It should ask the users to **confirm** the deletion before removing an item from the cart.

- [Fun 5] Checkout

Once a customer chooses to check out, the program should print out the **invoice** along with a list of purchased stamps to a .txt file. This file should have a **unique** name, indicating the shopping **date and time** (see the example in the following figure). It is assumed that the list of stamps can be divided and used by the users in their post items. Then, staff in Post Office can check the stamps and arrange the delivery. The format of the invoice should look like the following figure.



```
--------------Invoice---------------
Item No: 1  Item type: letter   Weight:  0.2000kg  Destination: CHINA     Unit price: $ 5.50
Item No: 2  Item type: parcel   Weight: 10.0000kg  Destination: GERMANY   Unit price: $ 104.83

Total Cost: 110.33
------------End Invoice--------------

-----------Purchased Stamps-------
letter
Destination: CHINA     Weight: 0.2kg
----------------------------------

-----------Purchased Stamps-------
parcel
Destination: GERMANY   Weight: 10.0kg
----------------------------------
```

Then, your program will also need to **add** the record to the given "**sales_history.csv**" file. Each record contains the information for each sold item:

Field	Description
sale_id	this is a unique identifier for a sale record. You can choose to use an incremental key for the sale_id, e.g., the id for the current sale = the id of the previous sale + 1. Note that a sale only occurs when a customer successfully checks out. A sale (to a customer) could contain multiple stamp items, but they are of the same sale_id.
date_time	date & time of the purchased in *yyyy-mm-dd hh:mm:ss* format
weight	The weight of the item (in kg)
destination country	The destination country of where the item will be sent to
item type	The type of item is one of the following values: letter or parcel.
cost	The cost of the item

Note that you should not change the name and format of this file.

Exception handling

Your program should try to avoid or catch any kind of exceptions and errors, so the execution won't be interrupted and it won't enter abnormal states.

When handling exceptions related to user inputs, your system should return **meaningful error messages** so that the users can correct their inputs.

Examples of these errors are (but not limited to):

- User input errors, e.g., option entered is invalid, invalid country name, invalid weight, etc.
- if the shopping cart to be paid has zero cost, etc.

Documentation

Precise and concise comments/documents of your code are essential as part of the assessment criteria. Do Not include things that are irrelevant in your code as that will reduce your code readability.

In addition, you need to submit a **final report (pdf file)** which covers the following points (but not limited to):

1. Describing the **high-level design** of your programs
2. Designing the **user cases** to address **EACH** requirement mentioned above. In this section, you need to capture screenshots of the results while executing your program. Our accessor will mark the functionality part mainly based on the user cases and the captured results from executing your program.

Note: sufficient evidence to prove the implementation of the requirements shall be provided, otherwise marks will be deducted.

Grading Rubric

Grading Rubric

Parts	**mark allocation**
Functionality	70%
Code quality & architect & documentation	30%

Functionality	**mark allocation**
Add an item to the shopping cart	30%
View shopping cart	10%
Amend an item	20%
Remove an item from the shopping cart	20%
Checkout	20%

Penalties

- Late submission: -5% per day late, no submission accepted after 4 days.

For further details on late submission, **read the policy here**.

Submission

- Rename "rename_me" directory to your student ID. This directory should contain **stamp_selling.ipynb**, **input CSV files**, **sales_history.csv, sales invoices,** and **final_report.pdf**.
- Write all the code and documentation in **stamp_selling.ipynb**
- Do not include any unnecessary file in this folder
- Zip the containing folder with the same name (ie <student_id>.zip) and upload it