Assignment 2 (P&C) – Testing

This part of Assignment 2 is mandatory for all customers targeting at grades of Pass, Credit, Distinction and High Distinction.

Consider a scenario where a store manager maintains a 'list.csv' file, used to track the remaining promotional balance for each customer. For this particular campaign, customers have the opportunity to select three unique items: a computer mouse, a bag, and a study lamp. Importantly, customers may not choose multiple items from the same category - for instance, purchasing two computer mice is disallowed.

The price ranges for these items vary. A computer mouse can cost anywhere between $1 to $300; the bag also ranges from $1 to $300; and the study lamp is priced between $1 to $400. Rest assured, the store has ample stock to fulfill all orders, and customers are given the flexibility to select one, two, or all three items as per their preference.

The specifications for the new program, record.py consists of its four main operations:

- **Add** function: This feature permits the manager to integrate a new customer into the system. By executing the command python record.py add <customer ID> <surname> <given name> <expense for the mouse> <expense for the bag> <expense for the lamp>, the manager can input the prices of the selected items for the new customer. Enter the price of zero if a particular item is not selected by the customer.
- **Delete** function: This function enables the manager to remove a customer from the system. To do this, run the command python record.py delete <customer ID>.
- **Update** function: This feature provides the manager with the ability to update the expenses of the items selected by an existing customer. This can be done by running the command python record.py update <customer ID> <surname> <given name> <expense for the mouse> <expense for the bag> <expense for the lamp>. Enter the price of zero if a particular item is not selected by the customer.
- **Report** function: This function gives the manager the ability to review the profit of promotion campaign for each customer. By executing the command python record.py report, the manager can view the expenditure. The expenditure is classified as "Lost" for a certain customer if it falls within the range of $800 to $1000, both inclusive. It is designated as "Low Profit" if it is equal to or greater than $500, but less than $800. Should the expenditure be less than $500, it is categorized as "High Profit".

The detailed description for these functions of the program record.py is given in Appendix A.

You are required to play the role as a software tester, undertaking the following tasks.

Task 1: Traceability matrix (3 points)

Traceability matrix is a useful tool supporting a good test planning as well as efficient test management and control. You are required to create and maintain a traceability matrix (in a tabular format) to map and trace requirements with test assets (including test conditions, test cases, and execution status) created in the whole testing process.

Hint: A sample traceability matrix will be provided in a template ('SQT_Assignment2_template.doc'). Relevant information about traceability matrix can be found in lectures (e.g., slides 73-75 in Week 5's lecture and slides 16, 22&41 in Week 6's lecture).

Task 2: Functional testing (19 points)

You are required to undertake all activities for the dynamic testing of all four major functions described in the beginning of this document and detailed in Appendix A. Specifically, you should complete the following three sub-tasks:

Sub-task 2.1: Test analysis (4 points)

You are required to analyse the requirements given in Appendix A on all four major functions of record.py, and define corresponding **test conditions**, which are normally elaborated into high-level test cases. Note that each defined test condition should be associated with a professional description (in one or two sentences) about "what to test". All test conditions should also be put into the traceability matrix created in Task 1 to map with requirements.

Hint: Relevant information and examples about test analysis and test conditions can be found in lectures (e.g., slides 15-25 in Week 6's lecture).

Sub-task 2.2: Test design (10 points)

You are required to design **concrete test cases** based on the test conditions defined in sub-task 2.1. Note that:

- Each test case should contain concrete values for input data as well as expected result such that they can be directly executed.
- One test condition may be associated with one or more concrete test cases.
- You should select and apply appropriate techniques (e.g., black-box and white-box) when designing test cases for each particular function ("add", "delete", "update", or "report" in record.py).
- All test cases should also be put into the traceability matrix created in Task 1 to map with test conditions as well as requirements.

Hint: Four sample test cases are provided in a template pytest script 'test_assignment2_xxxxxx.py' (further described in Appendix B) and briefly described in the sample document 'SQT_Assignment2_template.doc'. Relevant information and examples about test design, test cases, and concrete testing techniques can be found in lectures (e.g., slides 68-69 in Week 5's lecture, slides 26-30 in Week 6's lecture, and Week 8&9's lecture content). You have also practiced different techniques for designing test cases in our labs (e.g., lab work in Week 6, 8&9).

Sub-task 2.3: Test implementation, execution and reporting (5 points)

You are required to **schedule and execute** all test cases designed in sub-task 2.2. You should also **analyse and report** the results of test execution. Note that:

- Some test cases may have dependency relationships and different orders among them may affect the test results. For example, you may have a test case TC#1 that adds a customer with ID '123456' and another test case TC#2 that deletes the customer with ID '123456'. In this case, it is better to schedule the execution of TC#1 before TC#2. In a word, as an important activity for test implementation, you should schedule the execution order of test cases in a smart and efficient way.
- Although a template pytest script ('test_assignment2_xxxxxx.py') has been provided, it is NOT mandatory for you to execute test cases using pytest. You can choose whatever tool(s) you prefer to undertake the test execution activity. Manual execution without the support of any automation tool is even allowed.
- All test execution results (pass and/or fail) should be analysed, summarised and reported. They should also be recorded in the traceability matrix created in Task 1.

Hint: Relevant information and examples about test implementation, execution and reporting can be found in lectures (e.g., slides 31-41 in Week 6's lecture).

Task 3: Non-functional review (3 points)

You are required to undertake some static testing activities, especially review, for the non-functional characteristics of the program record.py. You should identify some potential issues of the program with respect to one or more quality characteristics, e.g., **performance efficiency and security**. You can also suggest some solutions to address these issues such that the program can be improved.

Hint: Relevant information about non-functional characteristics and static testing can be found in lectures (e.g., Week 3, 10&11's lecture content).

Submission

This is an **individual** assignment, which totals 25 points, 25% of the whole assessment of this unit. This assignment is mandatory to all customers. You are required to complete all three tasks and compose your answers into one single document in .doc, .docx, or .pdf file.

Every customer should submit his/her own work by **23.59pm Sunday the 26th of May 2024**. The assignment should be submitted via the assessment submission system in Canvas which integrates with the Turnitin plagiarism checking service. Also provided in the Canvas' submission system is a rubric detailing the assessment criteria for each of the above tasks.

You will be penalised 10% of the assessment's worth for each calendar day the task is late, up to a maximum of 5 days. After 5 calendar days, a zero result will be recorded. Customers with special circumstances (acute illness, loss or bereavement, hardship or trauma) may apply for an extension up to five days.

Appendix A: Description of program *record.py*

This description can be treated as an informal yet detailed requirements for each major function of the program *record.py*.

Functional hierarchy

The overall functional hierarchy of the program *record.py* is shown in Figure 1.

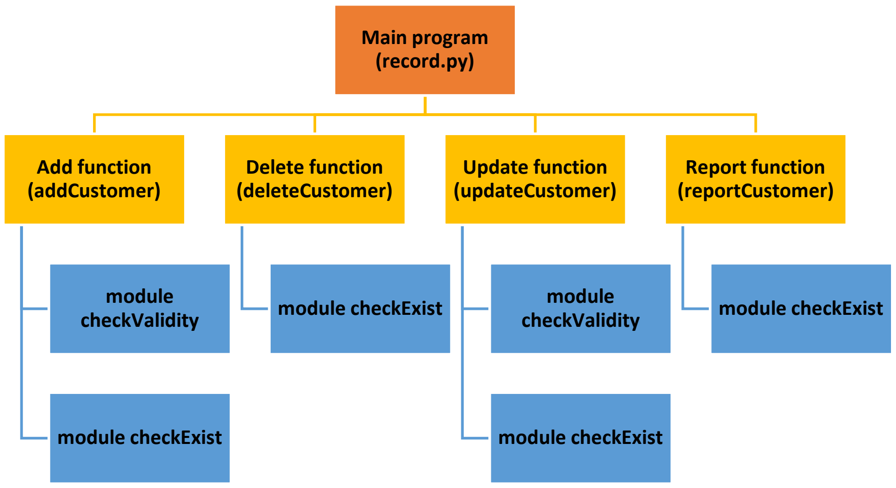


Figure 1 Functional hierarchy of record.py.

The program has four major functions "add", "delete", "update", and "report", which are implemented by modules 'addCustomer', 'deleteCustomer', 'updateCustomer', and 'reportCustomer', respectively. All these four modules will make use of a submodule 'checkExist' for checking whether a given customer ID exists in the file "list.csv". The modules 'addCustomer' and 'updateCustomer' will further utilise another submodule 'checkValidity' for checking whether the input customer information is valid or not. The required format for each customer's information is given as follows.

Required format for customer information

Each row of the file "list.csv" represents the information of a customer, containing six items, which are the customer's ID, surname, given name, and expenses for the mouse, bag and lamp. Table 1 gives the required format and/or value range for each item.

Table 1 Required format for customer information

Item	*Required format/value range*
Customer ID	Must be 6 digits, no letter allowed, e.g., 123456
Surname	Must contain letters only, length between 2 and 35 (inclusive)
Given name	Must contain letters only, length between 2 and 35 (inclusive)
Expense for the mouse	Must be an integer, value between 0 and 300 (inclusive)
Expense for the bag	Must be an integer, value between 0 and 300 (inclusive)
Expense for the lamp	Must be an integer, value between 0 and 400 (inclusive)

“Add” function

The module ‘addCustomer’ is executed as follows:

1. Check whether the provided customer information satisfies the required format/value range given in Table 1. If not, the customer information cannot be added, and the module will return a string “not added”. If all the information is valid, go to the next step.
2. Check whether the provided customer ID exists in the file “list.csv”. If it already exists, the customer information cannot be added and the module will return a string “not added”. If the ID is new (that is, it does not exist in the file), go to the next step.
3. Add the customer information as a new row in the file “list.csv” and return a string “added”.

“Delete” function

The module ‘deleteCustomer’ is executed as follows:

1. Check whether the provided customer ID exists in the file “list.csv”. If the ID does not exist in the file, the module will return a string “not exist”. If the ID exists in the file, go to the next step.
2. Delete the row related to the ID from the file “list.csv” and return a string “deleted”.

“Update” function

The module ‘updateCustomer’ is executed as follows:

1. Check whether the provided customer information satisfies the required format/value range given in Table 1. If not, the customer information cannot be added and the module will return a string “not updated”. If all the information is valid, go to the next step.
2. Check whether the provided customer ID exists in the file “list.csv”. If the ID does not exist in the file, the module will return a string “not updated”. If the ID exists in the file, go to the next step.
3. Update the row related to the ID in the file “list.csv” with the provided new information, and return a string “updated”.

“Report” function

The module ‘reportCustomer’ is executed as follows:

1. Check whether the provided customer ID exists in the file “list.csv”. If the ID does not exist in the file, the module will return a string “not exist”. If the ID exists in the file, go to the next step.
2. Calculate the final expense using the following formula:

   Final expense = expense for the mouse + expense for the bag + expense for the lamp
3. Determine the profit evaluation based on the following scheme:

Expense	800-1000	500-799	< 500
Profit Evaluation	Loss	Low Profit	High Profit

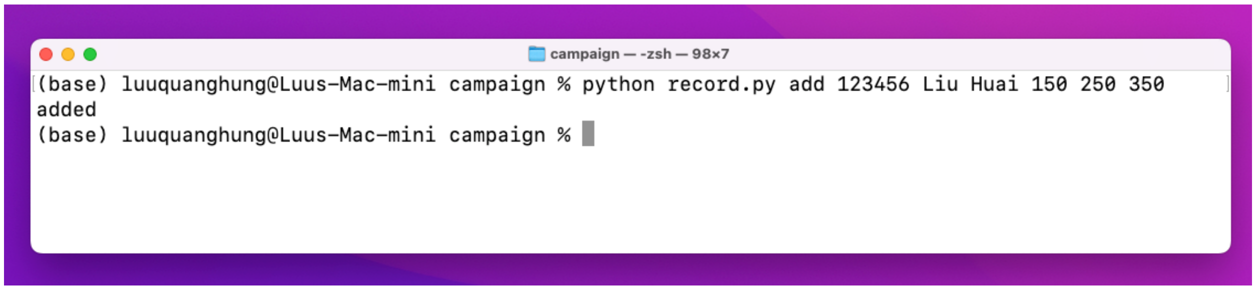
4. The module will return both the final expense (as an integer) and the evaluation (as a string).

Execution scenarios of *record.py*

The following shows some typical scenarios when running the program *record.py*.

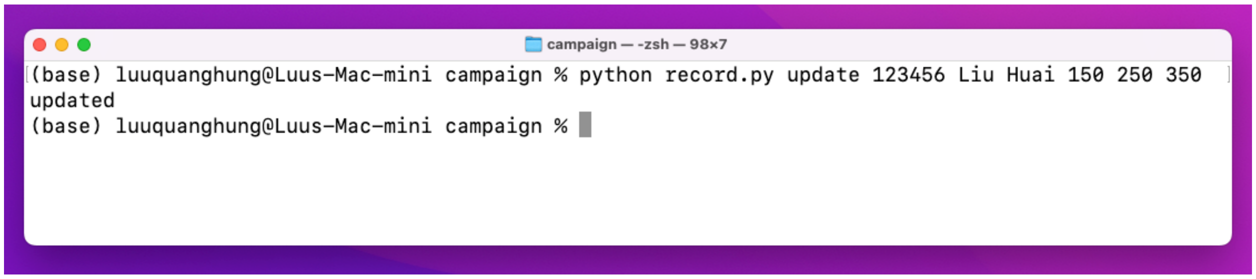
Normal cases

- For correctly running “Add” function, seven parameters must be provided in the command line, including the function name ‘add’ and the six items in Table 1. For example,



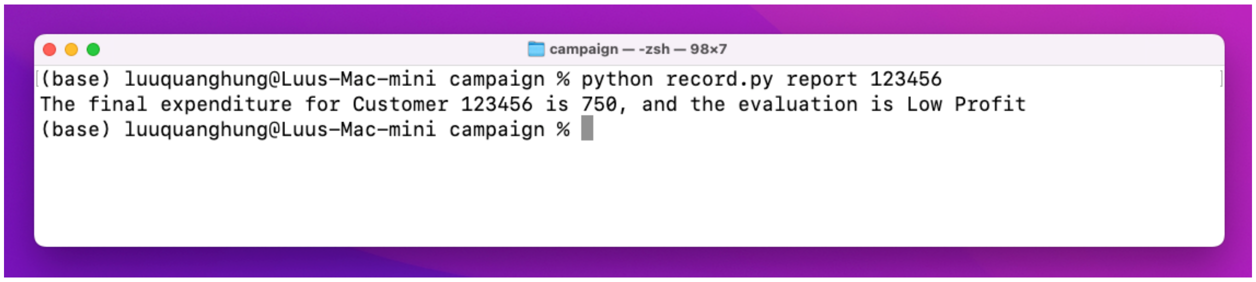
```
(base) luuquanghung@Luus-Mac-mini campaign % python record.py add 123456 Liu Huai 150 250 350
added
(base) luuquanghung@Luus-Mac-mini campaign %
```

- For correctly running “Update” function, seven parameters must be provided in the command line, including the function name ‘update’ and the six items in Table 1. For example,



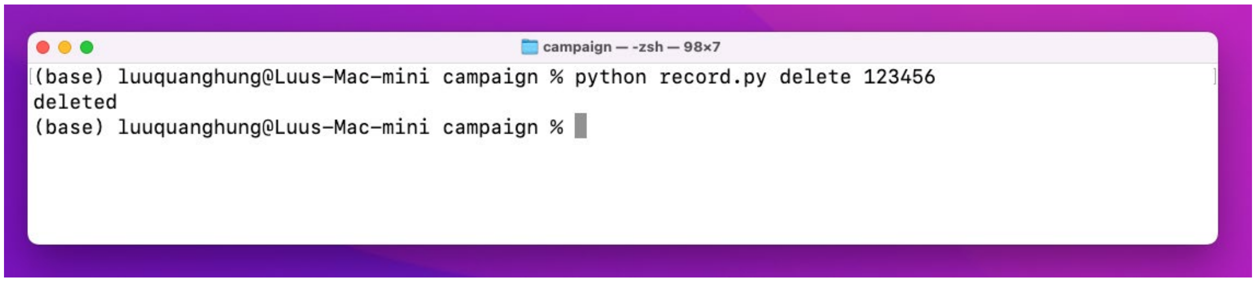
```
(base) luuquanghung@Luus-Mac-mini campaign % python record.py update 123456 Liu Huai 150 250 350
updated
(base) luuquanghung@Luus-Mac-mini campaign %
```

- For correctly running “Report” function, two parameters must be provided in the command line, including the function name ‘report’ and the customer ID. For example,



```
(base) luuquanghung@Luus-Mac-mini campaign % python record.py report 123456
The final expenditure for Customer 123456 is 750, and the evaluation is Low Profit
(base) luuquanghung@Luus-Mac-mini campaign %
```

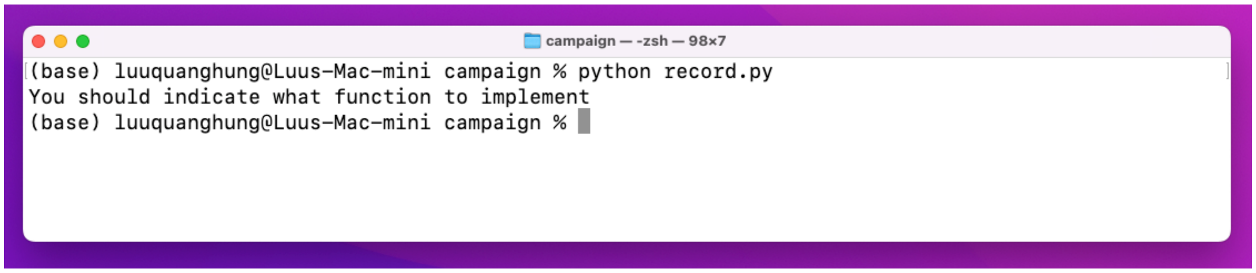
- For correctly running “Delete” function, two parameters must be provided in the command line, including the function name ‘delete’ and the customer ID. For example,



```
(base) luuquanghung@Luus-Mac-mini campaign % python record.py delete 123456
deleted
(base) luuquanghung@Luus-Mac-mini campaign %
```

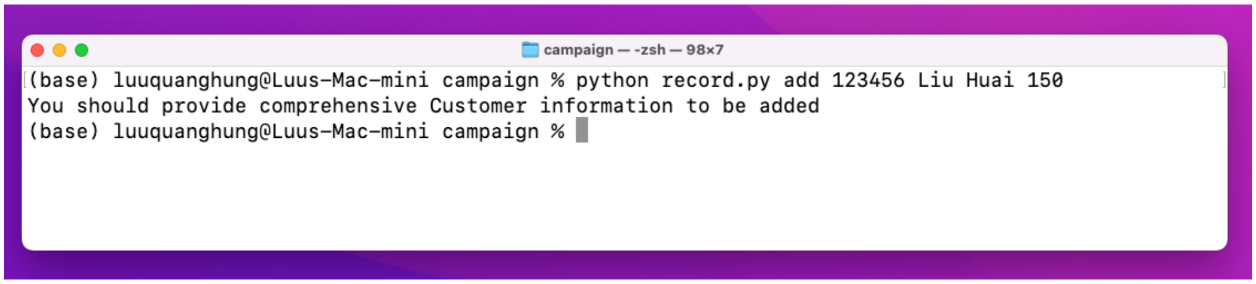
Exceptional cases

- If no parameter is provided when running the program, it will exit with a warning message. For example,



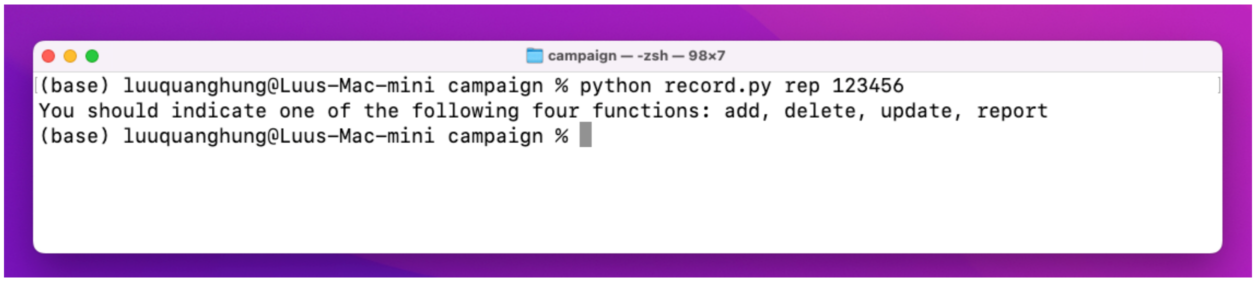
```
(base) luuquanghung@Luus-Mac-mini campaign % python record.py
You should indicate what function to implement
(base) luuquanghung@Luus-Mac-mini campaign %
```

- The program will also exit with a warning message if the number of provided parameters is not correct. For example,



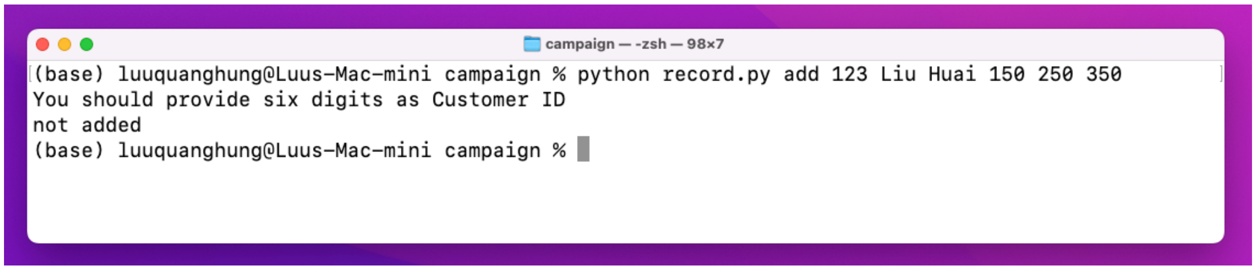
```
(base) luuquanghung@Luus-Mac-mini campaign % python record.py add 123456 Liu Huai 150
You should provide comprehensive Customer information to be added
(base) luuquanghung@Luus-Mac-mini campaign %
```

- The program will also exit with a warning message if the function name is not correct. For example,



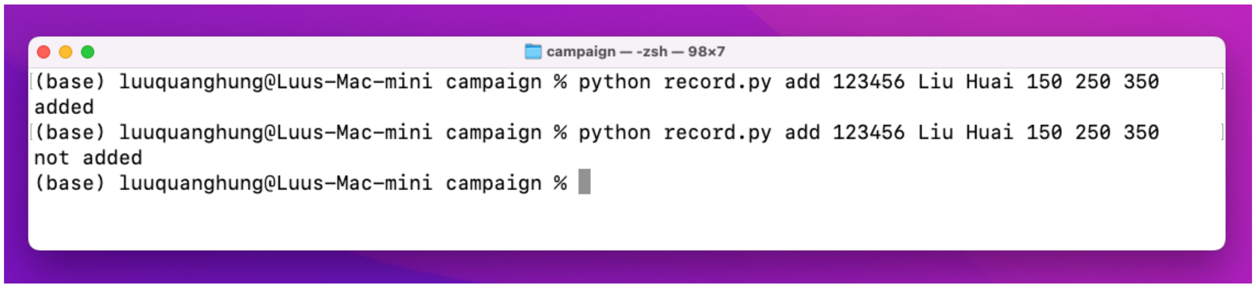
```
(base) luuquanghung@Luus-Mac-mini campaign % python record.py rep 123456
You should indicate one of the following four functions: add, delete, update, report
(base) luuquanghung@Luus-Mac-mini campaign %
```

- If invalid information is provided for “add” and “update” functions, the information cannot be added and updated, respectively. For example,



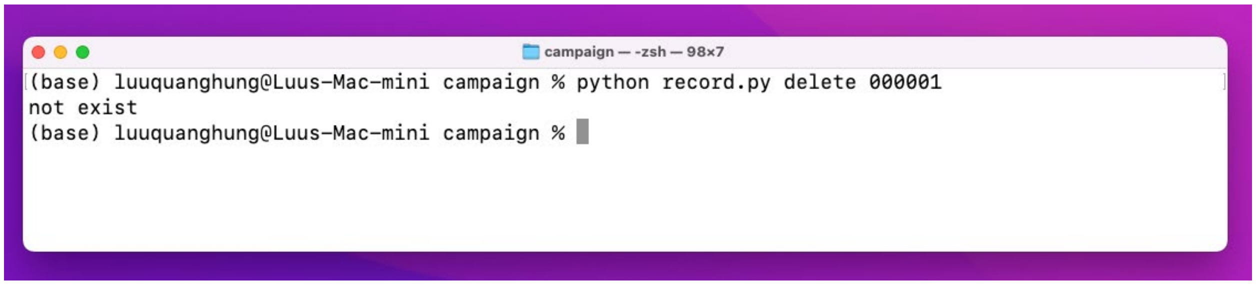
```
(base) luuquanghung@Luus-Mac-mini campaign % python record.py add 123 Liu Huai 150 250 350
You should provide six digits as Customer ID
not added
(base) luuquanghung@Luus-Mac-mini campaign %
```

- For the “add” function, if the given ID already exists, the information cannot be added. For example,



```
(base) luuquanghung@Luus-Mac-mini campaign % python record.py add 123456 Liu Huai 150 250 350
added
(base) luuquanghung@Luus-Mac-mini campaign % python record.py add 123456 Liu Huai 150 250 350
not added
(base) luuquanghung@Luus-Mac-mini campaign %
```

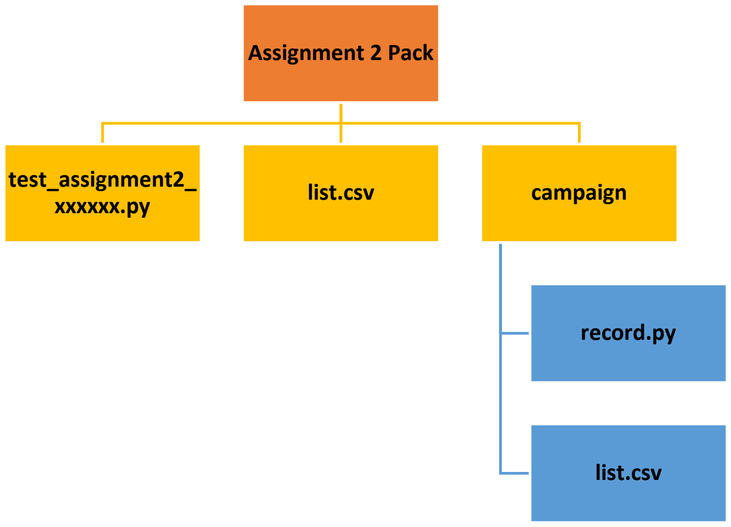
- For the “delete”, “update”, and “report” function, if the given ID does not exist, the corresponding operation cannot be undertaken. For example,



```
(base) luuquanghung@Luus-Mac-mini campaign % python record.py delete 000001
not exist
(base) luuquanghung@Luus-Mac-mini campaign %
```

Appendix B: Provided python/pytest code

You are given a file "assignment2pack.zip" for this assignment.



After unzipping the provided "assignment2pack.zip", you will find the following files:

- A template pytest script 'test_assignment2_xxxxxx.py', which contains four sample test cases. If you want to generate your own script based on this template, remember to replace "xxxxxx" in the file name by your customer ID.
  Note the **sequence** of these four test cases, which is "test_add", "test_update", "test_report", and finally "test_delete". Such a sequence was deliberately organised (as a test implementation activity) to guarantee the satisfaction of pre-conditions of some test cases. For example, only after adding the customer information with ID '123456' via the test case "test_add", can we successfully update its information via the test case "test_update".
- The "list.csv" file, which should be used as part of the input for testing, especially when you run the pytest script 'test_assignment2_xxxxxx.py'.
- The source code of the program 'record.py' in the folder "customerList".
- The "list.csv" file (exactly same as the other one) in the folder "customerList". You may use this file when you want to directly run the 'record.py' program in the folder "customerList".