Unit 15 Exercises – Problem-solving & planning

Aims and objectives

- To consolidate previous Python concepts
- To practice problem-solving and planning

The Problem: Simulating an ATM

The object of the exercise is to write a program that (somewhat) simulates the operation of an ATM (Automated Teller Machine). Part of the exercise involves developing a plan for the program, before starting to write code.

To do this exercise you will need to know about *pickling*, which was covered in Unit 12.

Bank account details are held in a file *accounts.pck*, which contains the pickled form of a dictionary with a certain structure which will be explained later. When the program is started, a dictionary storing bank account details is loaded from this file. The program then expects an account number to be entered (simulating the reading of a bank card), followed by a PIN. If the PIN is correct, a menu of options is offered, consisting of:

- w – withdraw money
- d – deposit money (of course, this option would normally only be available to a human clerk inside the bank)
- b – display current balance
- m – produce a mini-statement (details of the last 6 transactions)
- c – cancel request

When the transaction is complete, the program again expects an account number. This process continues until the account number 0 is entered, at which point the dictionary of account details is dumped to a new file called *new_accounts.pck*, and the program terminates. (It would be more realistic to dump the account details back to the original file, but this would make program development more problematic – the file might become corrupted because of programming errors.)

The effect of choosing each of the options is as follows:

- w: the program expects to input a floating point number, representing the amount to be withdrawn. If this does not exceed the balance in the account, this amount is subtracted from the balance, and the list of the last 6 transactions is updated, otherwise an error message is displayed.
- d: the program expects to input a floating point number, representing the amount to be deposited. This amount is added to the balance, and the list of the last 6 transactions is updated.
- b: the current balance in the account is displayed.
- m: a "mini-statement", consisting of the last 6 transactions, together with the current balance, is displayed.
- c: the transaction is cancelled (i.e., nothing is done to this account).

Data Structures

The main data structure is a dictionary with account numbers (represented by strings) as keys, and account details as values. **You must use this structure, otherwise you will not be able to use the data in *accounts.pck*.**

Details of an account are stored in a dictionary with the following keys and values.

- "pin": the PIN, as a string
- "balance": the account balance, as a floating point number
- "transactions": a list of the last 6 transactions

One transaction is stored as a dictionary with the following keys and values:

- "date": the date and time of the transaction, as a string
- "nature": a string, "w" for a withdrawal or "d" for a deposit
- "amount": the amount of the withdrawal or deposit, as a floating point number

What you are given

When you set up the exercise from the lab files on Moodle, you will obtain:

- a file *accounts.pck*
- a file *date.py*, which you can import as a module; it provides the function *getDate()* which obtains the current date and time and formats them into a string
- a file *display.py*, which you can use to display the account details from a file such as *accounts.pck*; this is useful for testing
- a file *plan.txt*, which contains a top-level plan and some refinement
- a file *skeleton.py*, which contains an outline of a Python program corresponding to the top-level plan

Task 1 – Refining the top-level plan; no mini-statements

For this task, ignore the option to produce a mini-statement; therefore you do not need to do anything with the list of recent transactions.

In the file *plan.txt*, develop further refinements of the plan, to the point where you feel confident about converting the plan into Python code.

Task 2 – Implementing the program without mini-statements

Copy *skeleton.py* to *task2.py*. Using the skeleton and your refined plan, implement the program, again ignoring the possibility of mini-statements. It is advisable to develop this program incrementally, testing at each stage. A possible strategy is as follows.

1. Write code to load the dictionary of account details from *accounts.pck* and dump it to *new_accounts.pck*. You will need to import the *pickle* module. Use *display.py* to check that this works; this will also let you see the details of the accounts in *accounts.pck*, which will be useful for testing later.
2. Implement the main loop of the program (i.e., terminating when account number 0 is entered) but without checking the account number or processing any options.
3. Write code to check that the account number is valid, and to input and check the PIN.

4. Write code to input an option and process it, at first just outputting confirmation of which option was chosen.
5. Implement the "display balance" option and test it.
6. Implement the "deposit" and "withdraw" options, and test them.

For simplicity, assume that data is always entered in the correct format.

Task 3 – Mini-statements

Copy *task2.py* to *task3.py*. Extend *task3.py* so that the option of producing a mini-statement is implemented. This will require further planning and perhaps refinement of your plan.

Task 4 (optional) – Adding a GUI

Implement a GUI for your program. This will require significant restructuring, although the core of the implementation of each option will remain the same.

Sample input/output

Text typed by the user is shown in bold.

```
Enter account number: 12345677
Invalid account number

Enter account number: 12345678
Enter PIN: 4312
Incorrect PIN

Enter account number: 12345678
Enter PIN: 4321
Options:  w: withdrawal
          d: deposit
          b: display balance
          m: mini-statement
          c: cancel request
Choose option: w 100.00
Insufficient funds in account

Enter account number: 98765432
Enter PIN: 2222
Options:  w: withdrawal
          d: deposit
          b: display balance
          m: mini-statement
          c: cancel request
Choose option: b
Current balance: 0.00

Enter account number: 86421357
Enter PIN: 1357
Options:  w: withdrawal
          d: deposit
          b: display balance
          m: mini-statement
          c: cancel request
Choose option: w 10.00
10.00 withdrawn

Enter account number: 98765432
Enter PIN: 2222
Options:  w: withdrawal
          d: deposit
```

```
          b: display balance
          m: mini-statement
          c: cancel request
Choose option: d 50.75
50.75 deposited

Enter account number: 12345678
Enter PIN: 4321
Options:  w: withdrawal
          d: deposit
          b: display balance
          m: mini-statement
          c: cancel request
Choose option: m
Account 12345678 mini-statement
27-02-2007, 14:02:58 deposit    34.56
27-02-2007, 14:03:17 withdrawal 10.00
27-02-2007, 14:03:33 withdrawal 12.45
27-02-2007, 15:35:45 deposit    12.00
27-02-2007, 15:35:53 withdrawal 5.76
27-02-2007, 15:36:02 deposit    23.00

Current balance:  62.69

Enter account number: 12345678
Enter PIN: 4321
Options:  w: withdrawal
          d: deposit
          b: display balance
          m: mini-statement
          c: cancel request
Choose option: c
Request cancelled

Enter account number: 0
```