MANUAL SUPER PY (Program for Supermarket)

Inhoud

Introduction

Superpy is a program by means of which the administration of a supermarket can be kept. The program can accept several kinds of cli (command line interface) commands. A cli command is simply a string of several arguments by means of which the program main.py will carry out a specific instruction.

Elaboration of CLI-commands

As mentioned in the introduction the program allows for a number of CLI commands. These will be discussed underneath mentioning examples of these. Note that any item or value never uses "",'' or =. It always starts of by py(thon) main.py e.g. py main.py --h to access the help command or

```
py main.py buy --product-name orange --price 0.68 --expiration-date 2021-06-10
```

for a buying instruction. Please be concise in formatting your instruction because the instructions are sensitive for hyphens, colons, equal signs(=) and basically any other reading sign/character. Also stick to the order in which the arguments make up for the instruction. Numbers which have . in common life should be written as such (e.g. 1.00 for prices)
The instructions are :

1.--advance-time x. Running the program with these arguments shifts the reference day forward by x days. Note that x should be a positive number because we only go forward in time.

2. buy --product-name xxxxxx --price xxx.yy --expiration-date yyyy-mm-dd.
   As you can see there are 6 individual parts in this instruction among which
   buy : this is the main part of the instruction which identifies its purpose
         buy
   --product-name : tag identifying the product name. Should be copied exactly
                    this way.
   xxxxxxxx   : any product/item sold by the supermarket such as apple,
                    orange, milk, bread etc. Please use the single form of the
                    article. Avoid using e.g. apples, oranges etc.
   --price   : the price in eur for which the article has been bought by
                    the supermarket.
   xxx.yy   : the number for the price of the article
   --expiration-date: tag identifying the expiration date. Should be copied

exactly this way.
yyyy-mm-dd : value for the expiration date (y: year, m: month, d: day)

Note that the buying date is actually set by the system for this instruction and equals the reference date registered in the referred_date.txt file

3. sell --product-name xxxxxx --price xxx.yy
   sell : main part of the instruction specifying the instruction type
   --product-name : tag identifying the product name. Should be kept identical
   xxxxxx : specification of the product name i.e. orange, apple, banana, coffee, bread etc. Like with the buy instruction please don't use plural format like apples, oranges, bananas etc.
   --price :identification of the selling price. Keep this description identical as specified here.
   xxx.yy :specification of selling price in eur. Use a similar format meaning x as a certain number of digits before the decimal point and yy as two digits after the decimal point e.g. 1.98
4. report. For reporting three different categories of data are used. All reporting is displayed by means of tables in the systems console window (either MS Dos or the Linux system's interface). The categories are inventory (how much is in stock), revenue (total amount sold), profit (total amount sold - total amount bought). Note that expiry of product doesn't play a part in reporting. It only affects the amount sold. For profit sub tables are created for sells and purchases.
   The formats of the instructions are :
   report inventory --now
   report inventory --yesterday
   report inventory --date yyyy-mm-dd
   report profit --today
   report profit --yesterday
   report profit --date yyyy-mm
   report revenue --today
   report revenue --yesterday
   report revenue --date yyyy-mm

   Both the --now and --today values refer to what's taken place so far today. The program can after all be run on a variety of time values whereas there may be more purchases or sells made throughout the day. Notice the inventory can be monitored on any other specific day whereas the revenue and profit can be computed for any other month. Notice the profit and revenue are computed for all products but not separately
5. stats. Stats are computed for all products separately, hence a product name makes up a part of the instruction. Another contrast compared to reporting is stats requires a start and end date. The output for the stats instruction is a bar chart except for profit in which case it's a line chart. In each case the display has the date on

```
           the x axis. The significance of the y axis depends on the kind of
           report which is made. The instruction for stats is specified as
           follows :
stats --product-name xxxxx --start-date yyyy-mm-dd --end-date yyyy-mm-dd zzzzzzzzz

           stats : principle indication of the report category. Stats stands
                   for statistics. Statistics means the output will be a
                   graphical display on the basis of which conclusions can be
                   drawn

   --product-name : identification of the product name. Please keep this
                    unchanged.
            xxxx  : specification of the product name e.g. apple, orange,
                    banana, milk, bread etc. Please report these products in
                    single format. Avoid writing apples, oranges etc.
     --start-date : identification of the start of the period for which the
                    graph will be made.
       yyyy-mm-dd : value for the start date which should be marked as y
                    (year), m(month), d(day)
       --end-date : identification of the end of the period. Please keep this
                    unchanged.
       yyyy-mm-dd : value for the end date of which the format should be y
                    (year) m(month) d(day)

           zzzzz  : this field can have one of the following 5 values. Each
                    refers to the specific product.
                    --number :fluctuation of sold numbers on different dates
                    --buy-price :fluctuation of the average daily price for
                      which the goods are bought
                    --sell-price :fluctuation of the average price for which
                      the goods are sold
                    --revenue : fluctuation of the daily revenue
                    --profit  : fluctuation of the daily profit
```

Examples and testing

Let me start of by staying it's always a question which character one would like to use for the decimals and what the format of dates should be. In this case the format of the arguments is specified in the English/American way so that's been left unchanged. However since I'm Dutch I prefer the ,. and the date format dd-mm-yyyy. So these have been implemented in the output. When setting up a project like this one this is actually more work since ,s need to be replaced with . when you want to read date from files and vice versa when you're creating them. Within Europe in most countries the , is used to indicate decimal places and dd-mm-yyyy or dd-mmm-yyyy for dates. Notice I took the trouble to implement this in the output.

MANUAL SUPER PY (Program for Supermarket)

In line with the preceding paragraph, this paragraph contains a list of examples. The examples are clarified in such a way ideas for testing the functionality are given as well.

In order to start testing it’s recommendable to set the date somewhere in the past e.g. 01-06-2021. This can be done by computing the number of days which have been passed since then and simply set by the command
py(thon) main.py --advance-time -x

This reference day is registered in the file referred_date.txt. The test should be carried out in such a way that from that day on the day is only shifted forwards. Reporting can of course be done with reference to a month or a date from the past.

Logically the next step is filling the inventory by means of buy instructions though for testing purposes it’s possible to try to sell products which have never been in stock or alternatively products that aren’t in stock because the products which have previously been bought are all sold or expired.

```
py main.py buy --product-name orange --price 0.68 --expiration-date 2021-06-10
Ok
py main.py buy --product-name pear --price 0.70 --expiration-date 2021-06-14
Ok
py main.py buy --product-name coconut --price 2.50 --expiration-date 2021-06-05
Ok
py main.py buy --product-name melon --price 0.55 --expiration-date 2021-06-20
Ok
py main.py buy --product-name milk --price 0.95 --expiration-date 2021-06-15
Ok
py main.py buy --product-name yoghurt --price 0.88 --expiration-date 2022-06-20
Ok
```

When you've done this, check the spreadsheet bought.csv. Explication of this file is as follows :
column 1 : Unique purchase id
column 2 : Product name
column 3 : Purchase date (equals reference date registered in referred_date.txt)
column 4 : Price for which the item was bought
column 5 : Expiration date
column 6 : either False when the item hasn't been sold or the date on which the item was sold

After having done this sell a couple of items using the following instructions:

```
py main.py sell --product-name orange --price 0.75
Ok
py main.py sell --product-name pear --price 0.80
Ok
```

MANUAL SUPER PY (Program for Supermarket)

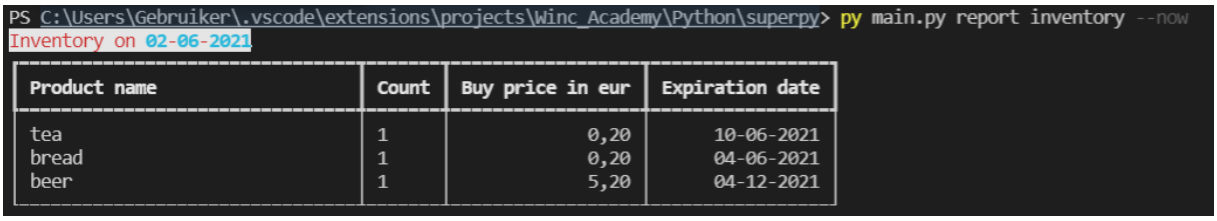
```
py main.py sell --product-name coconut --price 2.80
Ok
py main.py sell –product-name gold –-price 1000
ERROR. Not in stock
```

Check the selling date really gets filled in the bought.csv file according to the instructions you gave. After each selling instruction the file sold.csv gets appended. The explication of the columns in this file is as follows :
Column 1. Unique selling id
Column 2. Related buying id
Column 3. Product name
Column 4. Selling date in dd-mm-yyyy
Column 5. Price for which the product got sold
Column 6. Price for which the product got bought

Reporting

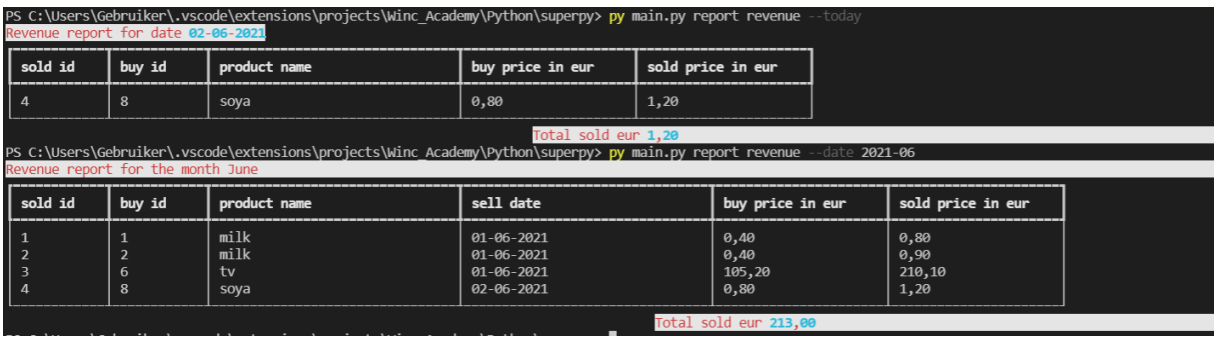
Now after the first buying instruction has been carried out, the inventory should be raised. In order to check the inventory use one of the following CLI's :

```
py main.py report inventory –-now
py main.py report inventory --yesterday
py main.py report inventory --date 2021-06-01 (or an alternate date)
```



Also immediately the revenue can be checked. In order to do this use one of the following instructions :

```
py main.py report revenue --today
py main.py report revenue --yesterday
py main.py report revenue --date 2021-06  (notice in this case the month is used where as for the inventory the date is used with a day)
```



When the revenu is listed for a month period a column for the selling date is added.
It's also possible to check the profit.

```
py main.py report profit --today
```

Manual SUPER PY made by Jochum Sloot

`py main.py report profit --yesterday`
`py main.py report profit --date 2021-06` (notice whichever option is used, the calculation of the profit always entails the calculation of the revenue as well as the computation of purchases and loss due to decay/expired products). Technically it means there's no duplication of code.

In case there are no sales or purchases taking place on a certain day or in a certain period, rather than showing an empty table, a coinciding remark will be given such as there are no sales for the month/date e.g. January/03-01-2021 (the slash indicating one or the other but not both).

Statistics

In all cases the x-axis refers to the date. When the numbers of days is restricted. Matplotlib plots the time as well. At first glance this is confusing. No effort has been put into it to clarify this. Overall the impression with statistics for the supermarket if the number of days is not restricted to a number which is smaller than 10. Matplotlib does computations on the representation of data. That was one reason why I didn't bother to try to convert the price/profit/revenue data here. These data are unfortunately still presented with a . as decimal point though it should be possible to present these data with a colon as decimal point as well (representation could be independent of computation).