Python Coding Test

Please create a python application that satisfies the requirements below. The source code should be a git repository with a `.gitignore` file. Use virtualenv and create your virtualenv directory `venv`, but if you have an additional IDE files or folders created please add them to your `.gitignore` or specifically do not add them into the git repo. When complete, you **must generate** a `requirements.pip` using pip. If you are not familiar with virtualenv and pip, you can find several guides online.

Please use virtualenv and create your virtualenv directory venv. The .gitignore file included with these instructions will ignore this directory, but if you have an additional IDE files or folders created please add them to your .gitignore or specifically do not add them into the git repo. When complete, you must generate a requirements.pip using pip. If you are not familiar with virtualenv and pip, you can find several guides online.

When you start, you **must create** the file called `started` that contains the current time at which you started. Please use **RFC-3339 format with seconds level precision**. Then commit this file only with the commit message of `Starting Challenge`

When you have completed the problem and commited your work to your local repository, zip the resulting directory and return the zipped solution to us by email. If the zip file is too large for email, you may host it using any storage solution you prefer (Dropbox, Google, Microsoft, ...) and send us the link.

Overview

You are to write a program to implement a grocery store / cashier line simulation. This program should read input from a file, and print the resulting score to the console. The program should be a console-only program. The program should take a single command line parameter: the name of the input file.

Customers in a grocery store arrive at a set of registers to check out. Each register is run by a cashier who serves customers on a first-come, first-served basis. The goal of the problem is to determine when all customers are done checking out.

One of the most important criteria for a successful solution is that it correctly implements the problem to be solved. This will be determined by code inspection, acceptance testing using the 5 examples demonstrated below, as well as additional test cases not provided on this page.

We will also be considering your overall approach to the problem and your programming style. This assignment is an opportunity to show off how you can use your object-oriented analysis and design skills to produce an elegant, readable, and maintainable solution. Please do NOT submit a multi-threaded or real-time solution. Please DO use what you would consider best practice in developing your solution.

Deliverables (all of the following)

The following should be available in your resulting solution zip file:

1. Source Code
   - The source code should be in Python, using version 2.7 (or any minor revision thereafter)
   - Unit tests (unittest or nose preferred)
2. Any other supporting code or other binaries that you used to develop the solution (even if it is not required to run the solution). If a resource is easily accessible on the web you can just provide a link to it rather than including the files.

Problem Details

1. The number of registers is specified by the problem input file. Registers are numbered 1, 2, 3, ..., n for n registers.
2. Time (`t`) is measured in minutes.
3. The grocery store always has a single cashier in training. This cashier is always assigned to the highest numbered register.
4. Regular registers take one minute to process each customer's item. The register staffed by the cashier in training takes two minutes for each item. So a customer with `n` items at a regular register takes `n` minutes to check out. However, if the customer ends up at the last register, it will take `2n` minutes to check out.
5. The simulation starts at `t=0`. At that time all registers are empty (i.e. no customers are in line).
6. Two types of customers arrive at the registers:
   - Customer Type A always chooses the register with the shortest line (least number of customers in line). If two or more registers have the shortest line, Customer Type A will choose the register with the lowest register number (e.g. register #1 would be chosen over register #3).
   - Customer Type B looks at the last customer in each line, and always chooses to be behind the customer with the fewest number of items left to check out, regardless of how many other customers are in the line or how many items they have. Customer Type B will always choose an empty line before a line with any customers in it.
7. Customers just finishing checking out do not count as being in line (for either kind of customer).
8. If two or more customers arrive at the same time, those with fewer items choose registers before those with more items. If the customers have the same number of items, then type A customers choose before type B customers.

Input Format

The first line of the input file is a single integer specifying the number of registers.

Each additional line is a whitespace-separated list of values. Each list specifies the customer type, customer arrival time (in minutes), and how many items that customer has respectively.

Review the below examples for sample input and output.

Example #1

For the following input file:

```
1
A 1 2
A 2 1
```

The following highlights occur:

- t=0 : Simulation starts with one register, which is a training register.
- t=1 : Customer #1 (type A) arrives with 2 items and goes to register #1, which starts servicing him.
- t=2 : Customer #2 (type A) arrives with 1 item and goes to register #1, behind Customer #1.
- t=3 : (Customer #1 now has one item left, since the first item took two minutes).
- t=5 : Customer #1 leaves and register #1 starts servicing Customer #2.
- t=7 : Customer #2 leaves.

Here is the expected command output:

```
$ python grocery.py input.txt
Finished at: t=7 minutes
```

Example #2

For the following input file:

```
2
A 1 5
B 2 1
A 3 5
B 5 3
A 8 2
```

The following highlights occur:

- t=0 : Simulation starts with two registers; #2 is a training register.
- t=1 : Customer #1 (type A) arrives with 5 items and goes to register #1, which starts servicing him.
- t=2 : Customer #2 (type B) arrives with 1 item and goes to register #2 which starts servicing her.
- t=3 : Customer #3 (type A) arrives with 5 items. Since he is type A, he goes to register #1 (lowest number of two with equal number of customers), behind Customer #1.
- t=4 : Customer #2 leaves from register #2 (which took two minutes). (At this point, register #1 has 7 items total: 2 for Customer #1 and all 5 for Customer #3.)
- t=5 : Customer #4 (type B) arrives with 3 items. Since register #1 has customers with 6 items, and register #2 is empty, she goes to register #2 which starts servicing her.
- t=6 : Customer #1 leaves and register #1 starts servicing Customer #3.
- t=8 : Customer #5 (type A) arrives with 2 items. Since both registers have one person, she goes to register #1 behind Customer #3.
- t=11 : Customer #3 leaves and register #1 starts servicing Customer #5.
- t=11 : Customer #4 leaves from register #2.

- t=13 : Customer #5 leaves from register #1.

Here is the expected command output:

```
$ python grocery.py input.txt
Finished at: t=13 minutes
```

Example #3

For the following input file:

```
2
A 1 2
A 1 2
A 2 1
A 3 2
```

Here is the expected command output:

```
$ python grocery.py input.txt
Finished at: t=6 minutes
```

This example illustrates the requirement that departing customers are not counted in line.

Example #4

For the following input file:

```
2
A 1 2
A 1 3
A 2 1
A 2 1
```

Here is the expected command output:

```
$ python grocery.py input.txt
Finished at: t=9 minutes
```

This example illustrates the requirement that customers with fewer items choose lines sooner.

Example #5

For the following input file:

```
2
A 1 3
A 1 5
A 3 1
B 4 1
A 4 1
```

Here is the expected command output:

```
$ python grocery.py input.txt
Finished at: t=11 minutes
```

This example illustrates the requirement that customers of type A choose before customers of type B.