Computer Science 1
Train Yard — Lab 8

RIT CS 10/29/2020

October 29-30, 2020

1 Problem

Your friend, Sheldon, is looking to start a delivery company, but he only wants to use trains. He has asked you to help him create an application that simulates the running of the trains and lets him know when trains will make it to certain stations.

With your new found knowledge of linked lists, you have all the confidence in the world that you can tackle this problem. With the train being linear, and dynamically expandable, it is a perfect use of a linked list, albeit a slightly customized version of the one we created in the lecture.

When you run the application, the user will be able to do any of the following, in somewhat of a flexible order, though some items need to be done before others for what will become obvious reasons. Here are the basic ideas for the application's command set.

- Set the engine speed for the train.
- Add a station – a place to drop off cars – to the route.
- Show all stations on the route (entered so far) with their relative distances.
- Add a train car to a location in the train based on its destination.
- Show all cars attached so far to the train.
- Start the simulation, which will start the train running, dropping off cars at stations until all train cars have been delivered.

We will be using miles and hours to talk about distance, time, and speed.

2 Problem Solving (15%)

You will be working in small teams, as determined by your instructor, to answer the following questions.

1. Given the following towns that comprise the train route
   - Richmond, VA: mile 0 (home)
   - Washington DC: mile 109
   - Philadelphia: mile 251
   - New York City: mile 341
   - Boston: mile 550

   Assuming a half-hour layover in each city, excluding the home city, and a train averaging 60 miles/hour, how long will the entire trip take?
2. The following cars are added to the train.
   (a) Politicians headed to DC
   (b) Ben Franklin headed to Philadelphia (his own train car!)
   (c) Tea destined for Boston
   (d) Bagels destined for New York City
   (e) Bears destined for Boston
   (f) Frozen shaved steak destined for Philadelphia
   (g) Pretzels destined for New York City

   What order should the cars be if, at each city, all the cars whose contents are destined for that city are at the end of the train for easy removal? Make a list of the cars' contents, front to back.
3. What data structures would you choose (lists, tuples, sets, dictionaries, classes, linked lists, and combinations thereof) to represent the information in the previous questions?
   (a) a station
   (b) the entire route of stations
   (c) a train car
   (d) the entire train

   Some of these things may be very simple. It is not certain that you will want to write a dataclass for each of them. *However you must use a linked list somewhere in your final code.*
4. In lecture you saw the following list functions.
   - convert to string
   - append a value
   - insert an item at a (numerical) location
   - remove an item of a given value

   Would any of the above be useful in adding new cars to the train? If so, state which one and why. If not, describe the new operation that you think should be written.

If you have time before the problem-solving (PS) session is over, start reading Section 3.1 to help you prepare for the post-PS work.

3 Post-Problem-Solving Implementation Work (10%)

There is a lot of material here, but it is rather repetitive. It is also a great way to set you up for the work you will do in the main part of the lab assignment.

Create your main file. Call it `train_run.py`.

In it, declare the class types you think you will use in the program.

You are free to import and use any files from the lecture notes to get the classes, and functions, you need.

Then read the specification in the following section.

3.1 Command List

Here are the commands allowed in your program. A lot of details are left out of this list, because you don't have to think about them for the Post-PSS work. It is sufficient to understand that each line contains a command, followed sometimes by arguments, and that everything is space-separated.

- `add_station` *station distance*
  Add a new stopping point named *station* to the train's route.
- `set_speed` *speed*
  Specify the speed for the train.
- `add_car` *content station*
  Add a new car containing *content* destined for *station*.
- `show_route`
  Print all the stations on the route.
- `show_train`
  Print a list of the cars currently on the train and the train speed.
- `start`
  Given the current route and train of cars, visit all the stations where the train has cars destined for them.
- `help`
  Print a list of possible commands.
- `quit`
  Terminate the program.

3.2 Command Handler Code

Write a `main` function. Fill it in with code that does the following.

1. displays a starting message
2. initializes any data structures, if needed
3. calls a `process_commands` function
   - The function will need some data structures passed to it.
4. displays a closing message.

Write the `process_commands` function. It should go into a loop, reading a line of input until the `quit` command is entered. The input consists of commands, read from *standard input*, which means that each line can be acquired by use of Python's `input` function. One good implementation strategy is to use `split`. You can extract the first word | the command | and use it to determine which command function to call. (See below.)

Print the command and the list of arguments you extracted.

For each command, write a function that will eventually handle that command. (You may prefer to put these functions in a separate file, if you like that organization.) Name the functions after the command names.

All the functions should have the same parameter list:

- the data structure(s) that will eventually be needed to carry out these commands, and
- a list of command argument strings that come with the command.
  (Make a *slice* from the result of the earlier `split` command.)

Each function returns a `bool` value to indicate whether or not the command's arguments were OK. For now, just "stub" them by simply returning True.

Inside the loop, if the value returned is `False` (which for now can't happen), print the error message

```
Illegal use or form for this command.
```

After that you may optionally show the proper format for the command. There should also be a final case that executes if the first word from the input does not match any legal command. Include code that prints the error message

```
Illegal command name.
```

After an error do not terminate the program; continue reading more commands until the `quit` command is entered.

3.3 Sample Output from Post-PSS Work

```
Welcome to the train yard.
> help
Command is help arguments are []
> addStation Rochester 0
Command is addStation arguments are ['Rochester', '0']
Illegal command name.
> add_station Rochester 0
Command is add_station arguments are ['Rochester', '0']
> add_station RIT 9
Command is add_station arguments are ['RIT', '9']
> quit
Command is quit arguments are []
Train yard simulation ending.
```

3.4 First Submission

Zip all the code files you have created into a zip file called **postPSS08.zip**. If you used any course Python files, include them as well in the zip file. Submit it to the mycourses **Assignments** drop box.

4 Full Implementation(75%)

Implement all the commands for the simulation. Details of the commands are given below.

4.1 Input

The input consists of commands, read from *standard input*.

- `add_station` *station distance*
  Add a new stopping point named *station* to the train's route.
  - *station* is a string without any spaces.
  - *distance* is a floating-point number. Its value is the distance from the home station.
  - ***NOTE!*** stations are entered in order from closest to farthest! (a simplification)
  - The home station at mile 0 must be specified first.
  - *distance* must not be negative.
  - Two stations at the same *distance* is not allowed.
- `set_speed` *speed*
  Specify the speed for the train.
  - *speed* is a positive floating-point number.
- `add_car` *content station*
  Add a new car containing *content* destined for *station*.
  - *content* is a string without any spaces.
  - *station* is a string without any spaces.
  - *station* must have already been defined in an `add_station` command.
  - Car positions are based on destination station's distance from the home station. For easy simulated removal, the cars destined for the closest stations are further back on the train. However, for easy manipulation in the program, cars destined for the closest stations ought to be put towards the head of the list structure.
  - No cars may be destined for the station at mile 0.
- `show_route`
  Print all the stations in the route with this format.
  `DC --- 142 --> Philadelphia --- 90 --> NYC --- 209 --> Boston`
  (*station* `--` *relative-distance* `-->` *station* `--` ...)
  - The state of the system does not change at all.
  - Remember – the stations are entered in order.
- `show_train`
  Print the train speed and a list of the cars currently on the train, *starting at the end of the list*, i.e., the front of the train.
  - The state of the system does not change at all.
  - Engine speed format: "`engine(` *speed* `)`"
  - Train car list format *is up to you!* Just be reasonable.

- `start`
  Given the current route and train of cars, visit all the stations where the train has cars destined for them.
  - Keep track of time based on route distances, engine speed, and the fixed layover time per station.
    - Allow a fixed layover time of *a half hour* at each station to decouple the cars from the end of the train. *No time is taken at stations where no cars need to be dropped off.*
  - The train is literally empty upon completion of this command; only the engine's speed is retained.
  - Whenever `start` is executed, the train always starts from its home station. There is no simulated return trip.
  - Details of the output format can be gleaned from the examples later in this document.
- `help`
  Print a list of commands as follows.
  ```
  add_car <content> <station>
  set_speed <speed>
  add_station <station> <distance>
  show_route
  show_train
  start
  help
  quit
  ```
- `quit`
  Terminate the program, unconditionally.
  - Print the concluding message "`Train yard simulation ending.`"

4.2 Sample Output

Use the following sample run as a guide. Some line breaks were added for the `show_route` and `show_train` commands so that they would fit within the margins.

The `show_train` output does not have to match what you see below.

```
Welcome to the train yard.
List of commands
================
add_car <content> <station>
set_speed <speed>
add_station <station> <distance>
show_route
show_train
start
help
quit


> add_station Richmond 0

> add_station DC 109

> add_station Philadelphia 251

> show_route
Richmond --- 109.0 --> DC --- 142.0 --> Philadelphia

> add_car politicians DC

> add_car benFranklin Philadelphia

> show_train
engine( 0.0 ) [ TrainCar(contents='politicians', destination='DC'),
TrainCar(contents='benFranklin', destination='Philadelphia') ]

> add_station NYC 341

> add_station Boston 550

> add_car bears Boston

> add_car shavedSteak Philadelphia

> add_car bagels NYC
> add_car tea Boston

> add_car pretzels NYC

> show_route
Richmond --- 109.0 --> DC --- 142.0 --> Philadelphia --- 90.0 --> NYC ---
209.0 --> Boston

> set_speed 60

> show_train
engine( 60.0 ) [ TrainCar(contents='politicians', destination='DC'),
TrainCar(contents='shavedSteak', destination='Philadelphia'),
TrainCar(contents='benFranklin', destination='Philadelphia'),
TrainCar(contents='pretzels', destination='NYC'),
TrainCar(contents='bagels', destination='NYC'),
TrainCar(contents='tea', destination='Boston'),
TrainCar(contents='bears', destination='Boston') ]

> start
Moving on to DC
0.50 hours taken to separate cars.
This segment took 1.82 hours to travel.
Unloading politicians in DC
Moving on to Philadelphia
0.50 hours taken to separate cars.
This segment took 2.37 hours to travel.
Unloading shavedSteak in Philadelphia
Unloading benFranklin in Philadelphia
Moving on to NYC
0.50 hours taken to separate cars.
This segment took 1.50 hours to travel.
Unloading pretzels in NYC
Unloading bagels in NYC
Moving on to Boston
0.50 hours taken to separate cars.
This segment took 3.48 hours to travel.
Unloading tea in Boston
Unloading bears in Boston
Total time for trip was 11.17 hours.

> quit
Train yard simulation ending.
```

4.3 Additional Requirements

You must use a mutable linked list based on lecture code for the train cars. Other data structure choices are up to you.

The input errors you must check for are listed here.

- The command entered is legal.
- The correct number of arguments are given for the command.
- The destination in `add_car` is known.
- The home station is the first one specified.
- The home station and train speed have been set when `start` command is entered.
  The following checks are optional.
- positive speed
- positive distance

Error message formats are not specified. Just make sure they are clear.

4.4 Design and Implementation Tips

As stated in Section 4.1, you will find it easier to work with a linked list of cars if you stipulate that the front of the list is actually the back end of the train. This is because removing items from the front of a linked list is less complex.

Because new stations can be added at any time, each station should probably contain its distance from the home station.

To print a floating-point number `x` with two decimal places, use the expression “`("%0.2f" % x)`”. The result is a string. More information on this formatting technique can be found at
`https://docs.python.org/3/library/stdtypes.html#old-string-formatting`

You will probably want to enter the same sequence of commands many times while you’re testing. There is a way in PyCharm to put what you would have typed into a text file.

1. Make a copy of your run configuration.
2. Look for a check box labeled “redirect input from” and check it.
3. Enter the name of your command text file in the text box next to it.
4. Name this new configuration appropriately.
5. Save the configuration. (Apply/OK)

If you are executing from a shell or command window, it’s easier. Just type

```
python train_run.py < test_file.text
```

(or “`python3`”, on UNIX^{TM}-based platforms)

5 Samples

You can get sample test files and their output from
`https://www.cs.rit.edu/~csci141/Labs/08/lab08-tests.zip`.

6 Grading

Here is the grade point breakdown for this lab.

6.1 Problem-Solving Session

Adequate participation is worth 15% of your grade.

6.2 Post-Problem-Solving Work

Total: 10%

- 3% Contains some data type declarations
- 3% A `main` function that runs without crashing
- 4% `process_commands` function works as specified

6.3 Main Lab Assignment

Total: 75%

- 5% Initialization (everything before reading the first command)
- 10% Command dispatch functionality (`process_commands`)
- 15% add_station and show_route
  structure of data, output format
- 20% set_speed and add_car
  ability to set the train's speed and add the train cars
- 5% show_train format
- 15% start
  The train traverses the route, dropping off cars at stations.
- 3% `help` display accuracy
- 2% `quit` function

- Style: up to 10% deduction
- No use anywhere of linked list structures: up to 20% deduction
- Not following submission instructions: 10% deduction

Overall, error-checking is 5% of the individual grade components above.

7 Submission

Zip **train_run.py** and all of your other source (**.py**) files, including ones from the course that you used, into a zip file called **lab08.zip**. Submit **lab08.zip** to the myCourses dropbox.