In-Memory Cache:

Description:

Babbar Bhaiya got a new client for his web-services company. Clients wants to have a efficiently designed In-Memory Cache system, which he/she can invoke with the driver class. Help babbar bhaiya to deliver this task.

Requirements:

- create a user-defined size cache and with Eviction policy as “LRU”
- support fetch/Insert/Delete/Clear functionality in the system.

Other Details:

Do not use any database or NoSQL store, use in-memory store for now.
Do not create any UI for the application.
Write a driver class for demo purposes. Which will execute all the commands at one place in the code and test cases.
Please prioritize code compilation, execution and completion.
Please do not access the internet for anything EXCEPT syntax.
You are free to use the language of your choice.
All work should be your own. If found otherwise, you may be disqualified.

Expectations:

Code should be demo-able (very important)
Complete coding within the duration of 90 minutes.
Code should be modular, with Object Oriented design.
Maintain good separation of concerns.
Code should be extensible. It should be easy to add/remove functionality without rewriting the entire codebase.
Code should handle edge cases properly and fail gracefully.
Code should be readable.
Follow good coding practices: Use intuitive variable names, function names, class names etc. Indent code properly. Once the code is complete please zip the source code and upload it to: xxxyyyzzz.com

Foodkart:

Description:

Flapkart is starting a new online food ordering service. In this Service, users can order food from a restaurant which is serviceable in their area and the restaurant will deliver it.

Features:

1. Restaurants can only serve one specialized dish.
2. Restaurants can serve in multiple areas.
3. At a time, users can order from one restaurant, and the quantity of food can be more than one.
4. Users should be able to rate any restaurant with or without comment.
5. Rating of a restaurant is the average rating given by all customers.

Requirements:

1. Register a User:
    a. register_user(user_details)
    b. user_details: name, gender, phoneNumber(unique) and pincode.
2. Users should be able to login, and all the operations will happen in the context of that user. If another user logged in, the previous user will automatically be logged out.
    a. login_user(user_id): this should set the context for all the next operation to be done by this user.
3. Register a restaurant in context of login user:
    a. Register_restaurant(resturant_name, list of serviceable pin-codes, food item name, food item price, initial quantity).
4. Restaurant owners should be able to increase the quantity of the food item.
    a. update_quantity(restaurant name, quantity to Add)
5. Users should be able to rate(1(Lowest)-5(Highest)) any restaurant with or without comment.
    a. rate_restaurant(restaurant name, rating, comment)
6. User should be able to get list of all serviceable restaurant, food item name and price in descending order: show_restaurant(rating/price)
    a. Based on rating
    b. Based on Price
    NOTE: A restaurant is serviceable when it delivers to the user's pincode and has non-zero quantity of food item.
7. Place an order from any restaurant with any allowed quantity.
    a. place_order(restaurant name, quantity)

Bonus:

1. Order History of User: For a given user you should be able to fetch order history.

Other Details:

1. Do not use any database or NoSQL store, use in-memory store for now.
2. Do not create any UI for the application.

3. Write a driver class for demo purposes. Which will execute all the commands at one place in the code and test cases.
4. Please prioritize code compilation, execution and completion.
5. Please do not access the internet for anything EXCEPT syntax.
6. You are free to use the language of your choice.
7. All work should be your own. If found otherwise, you may be disqualified.

Expectations:

1. Code should be demoable (very important)
2. Complete coding within the duration of 90 minutes.
3. Code should be modular, with Object Oriented design. Maintain good separation of concerns.
4. Code should be extensible. It should be easy to add/remove functionality without rewriting the entire codebase.
5. Code should handle edge cases properly and fail gracefully.
6. Code should be readable. Follow good coding practices:
7. Use intuitive variable names, function names, class names etc.
8. Indent code properly.
9. Once the code is complete please zip the source code and upload it to: https://docs.google.com/

Sample Test Case:

All the inputs here are just indicating the high level inputs that function should accept. You are free to model entities as per your choice.

register_user(“Pralove”, “M”, “phoneNumber-1”, “HSR”)
register_user(“Nitesh”, “M”, “phoneNumber-2”, “BTM”)
register_user(“Vatsal”, “M”, “phoneNumber-3”, “BTM”)

login_user(“phoneNumber-1”)

register_restaurant(“Food Court-1”, “BTM/HSR”, “NI Thali”, 100, 5)
NOTE: we will have 2 delimiters in input : ',' to specify separate fields & '/' to identify different pincodes.
register_restaurant(“Food Court-2”, “BTM”, “Burger”, 120, 3)

login_user(“phoneNumber-2”)
register_restaurant(“Food Court-3”, “HSR”, “SI Thali”, 150, 1)
login_user(“phoneNumber-3”)
show_restaurant(“price”)

Output : Food Court-2, Burger
Food Court-1, NI Thali

place_order(“Food Court-1”, 2)
Output: Order Placed Successfully.

place_order(““Food Court-2”, 7)
Output : Cannot place order

create_review(“Food Court-2”, 3, “Good Food”)
create_review(“Food Court-1”, 5, “Nice Food”)

show_restaurant(“rating”)
Output : Food Court-1, NI Thali
Food Court-2, Burger

login_user(“phoneNumber-1”)
update_quantity(“Food Court-2”, 5)
Output: Food Court-2, BTM, Burger - 8

update_location(“Food Court-2”, “BTM/HSR”)
Output: Food Court-2, “BTM/HSR”, Burger - 8

Online Dating Application:

Problem Statement:

Create an online dating application. Every Active user account will have location, age and gender info. The App should show users their potential matches in order of relevance. The ordering of relevance will be following:

1. Gender : Opposite gender -> high priority.
2. Proximity: Nearer matches should be given more priority. Use following formula for computing distance between two locations (*dist((ax, y), (a, b)) = sqrt((x - a)^2 + (y - b)^2)).
3. Age: Less age difference -> high priority.

Operations:

A user can perform these operations in this application:

1. Create Account: A person can create an account with interest and profile details.
2. Potential Match: Provides all the potential match of a user in relevance order.
3. Like: User can like a potential match user.
4. Show Matches: Showing the users which match against a user. A match happens when both the users have liked each other.
5. Show All Matches: Showing system view by displaying all the matches in the system.

6. Ignore: User can ignore a potential match user.
7. Delete Account: If a user deletes account, then all matches and likes will be removed.

Use case:

1. If a user A likes user B, the data should be stored for further processing.
2. All the matches(case where 2 users have liked each other) in the system should be shown.

Guidelines:

- Input can be read from a file or STDIN or coded in a driver method.
- Output can be written to a file or STDOUT.
- Feel free to store all interim/output data in-memory.
- Restrict internet usage to looking up syntax
- You are free to use the language of your choice.
- Save your code/project by your name and email it. Your program will be executed on another machine. So, explicitly specify dependencies, if any, in your email.

Expectations:

- Code should be demo-able (very important). Code should be functionally correct and complete.

  - At the end of this interview round, an interviewer will provide multiple inputs to your program for which it is expected to work
- Code should handle edge cases properly and fail gracefully. Add suitable exception handling, wherever applicable.
- Code should have good object-oriented design.

- Code should be readable, modular, testable and extensible. Use intuitive names for your variables, methods and classes.
  - It should be easy to add/remove functionality without rewriting a lot of code.
  - Do not write monolithic code.
- Don't use any databases.

1. Input format:
   a. create_account(user_name, x_coordinate, y_coordinate, age, gender)
   b. delete_account(user_name)
   c. potential_match(user_name)
   d. like(user_name, user_name)
   e. ignore(user_name, user_name)

f. show_matches(user_name)
g. show_all_matches()

News Feed :

Your aim is to write a console program that can simulate a social network.

Social Network Platform has following capabilities:

- [signup] signup a user on the platform
- [login] login a user into the platform

Users have the following capabilities (command to be used in [-]):

- [post] A user can post a feed item.
- [follow] Users can follow other users.
- [reply] A user can comment on another user's feed item.
- [upvote/downvote] Upvote or downvote posts.
- [shownewsfeed] Any user can read his news feed. News items are sorted based on the following:
  - Followed users : posts by followed users appear first.
  - Score (= upvotes - downvotes): higher the better.
  - Number of comments: higher the better.
  - Timestamp: more recent the better.

Bonus:

- Allow users to comment on a comment and upvote
- Display time in language like 2m ago, 1 hr ago etc.

Input and output

The input should be taken in the form of statements. The statements should consist of commands and arguments. Commands can be words like ‘signup’, ‘newsfeed’, ‘upvote’ etc. Arguments can be separated from commands by special characters something like ‘~’. As a result a statement to reply to post 005 can look something like

```
reply~005~this is the reply text
```

You may change the input output format without changing the functionality to suit

your needs.

Sample:

Input is in bold, output is not in bold.You need to print the output of *shownewsfeed* for a user, whenever he logs in.

```
signup~lucious
signup~albus
signup~tom
login~tom
post~I am going to be the darkest dark wizard of all time
post~I am lord Voldemort btw 3:)
login~lucious

      id: 002
      (0 upvotes, 0 downvotes)
      tom
      I am lord Voldemort btw 3:)
      1994-04-19 10:11 PM

      id: 001
      (0 upvotes, 0 downvotes)
      tom
      I am going to be the darkest dark wizard of all time
      1994-04-19 10:10 PM

upvote~001
follow~tom
reply~001~I am with you dark lord!
login~albus

      id: 001
      (1 upvote, 0 downvotes)
      tom
      I am going to be the darkest dark wizard of all time
      1994-04-19 10:10 PM
            id: 003
            Lucious
            I am with you dark lord!
```

```
      1994-04-19 10:15 PM

id: 002
(0 upvotes, 0 downvotes)
tom
I am lord Voldemort btw 3:)
1994-04-19 10:11 PM



post~Happiness can be found, even in the darkest of times, if one only
remembers to turn on the light
follow~tom
downvote~001
downvote~002
reply~002~LOL!
shownewsfeed

      id: 001
      (1 upvote, 1 downvotes)
      tom
      I am going to be the darkest dark wizard of all time
      1994-04-19 10:10 PM
            id: 003
            lucius
            I am with you dark lord!
            1994-04-19 10:15 PM

      id: 002
      (0 upvotes, 1 downvotes)
      tom
      I am lord Voldemort btw 3:)
      1994-04-19 10:11 PM
            id: 005
            albus
            LOL!
            1994-04-19 10:30 PM

      id: 004
      (0 upvotes, 0 downvotes)
      albus
      Happiness can be found, even in the darkest of times, if one only
```

```
remembers to turn on the light
1994-04-19 10:28 PM
```

Guidelines:

- Input can be read from a file or STDIN or coded in a driver method.
- Output can be written to a file or STDOUT.
- Feel free to store all interim/output data in-memory.
- Restrict internet usage to looking up syntax
- You are free to use the language of your choice.
- Save your code/project by your name and email it. Your program will be executed on another machine. So, explicitly specify dependencies, if any, in your email.

Expectations:

- Code should be demo-able (very important). Code should be functionally correct and complete.

  ○ At the end of this interview round, an interviewer will provide multiple inputs to your program for which it is expected to work
- Code should handle edge cases properly and fail gracefully. Add suitable exception handling, wherever applicable.
- Code should have good object-oriented design.
- Code should be readable, modular, testable and extensible. Use intuitive names for your variables, methods and classes.
  - ○ It should be easy to add/remove functionality without rewriting a lot of code.
  - ○ Do not write monolithic code.
- Don’t use any databases.