HR: Please send this questionnaire to the candidate 2/3 days before the interviewer and interviewee can discuss the solutions in a conference call. If working candidates request it to be sent on Friday to discuss solutions on immediately following Monday/Tuesday, so that they can work on weekend. Any exceptions requested by the candidate with sufficient reasoning is acceptable (for e.g. if he/she requests more than 3 days)

General: The solutions to the below questions will be discussed on a conference call through a desktop sharing session. The interviewee will display the execution of code on his/her machine during the conference call.

Interviewer: The interviewer may choose to ask other similar questions during the interview process

Interviewee: The following exercises are intended to help us get a feel for your coding habits and strategies. You may search online and find solutions to the below. For each of the questions below, check-in your code to Github (where relevant) and share the link with us before the interview. You will have to share your desktop while explaining the solution

You may take additional time beyond the interview date, however let us know at least 2 days in advance "how" much more time you need. All Code exercises have to be done at once and submitted. Please create a git tag on the master branch at time of completion of each phase and submit the exercise with a link to that tag.

Code Exercise 1:

In the software language of your choice, and given standard algebraic notation of a chess board (see below), write code that will:

Accept two parameters:

1. **Type** of chess piece (Queen, Rook, Knight)
2. **Current position** on a chess board (for example: d2)

Return:

A list of all the potential board positions the given piece could advance to, with one move, from the given position, with the assumption there are no other pieces on the board.

Rules:

- You do not have to implement the solution for every piece type, but the solution must implement at least the following: Queen, Rook and Knight.
- You may not use any external/non-core libraries: use only primitives and built-ins for the chosen language.
- Please provide test coverage for your work.

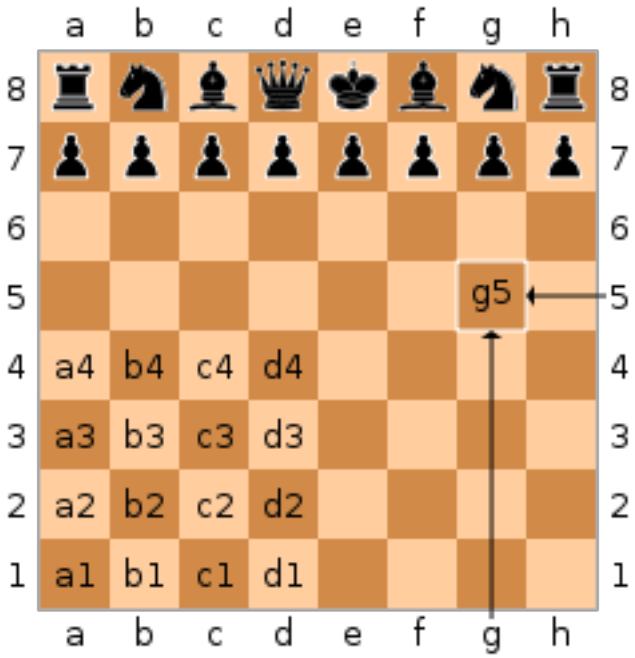
Example:

If the code is passed: "knight, d2"

```
$ chessercise.py -piece KNIGHT -position d2
```

The output should be: "b1, f1, b3, f3,c4, e4"

Algebraic Notation Legend:



Code Exercise 2:

Building on the elements from Exercise 1: **Implement a "Target" mode with --*target* parameter.**

- Randomly place 8 (opposing) pieces onto the board tiles.
- Determine the physically **most distant tile** from **Current position**. Calculate and output the ***minimum set of moves*** which the given **piece Type** could take to the **most distant tile** given that:
- Opposing pieces do not move.
- Opposing pieces may be "captured" along the way by moving to the occupied tile.
- Capturing an opposing piece marks the end of a "move".
- Provide test coverage.

Code Exercise 3:

Building on the elements from the previous Exercises: **Implement a “Collector” mode with *--collect* parameter.**

- Randomly place 8 (opposing) pieces onto the board tiles.
- Calculate and output the ***minimum set of moves*** which the given **pieceType** could take to **captureall** opposing pieces.
- Provide test coverage.