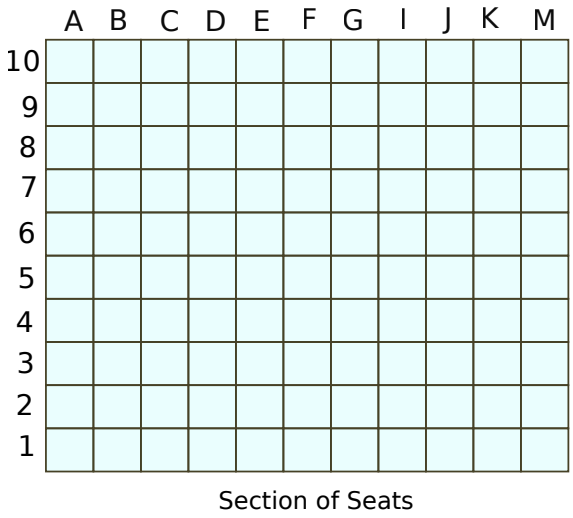

Figure 1: The "blank" `tty` output presented to user once the game commences.

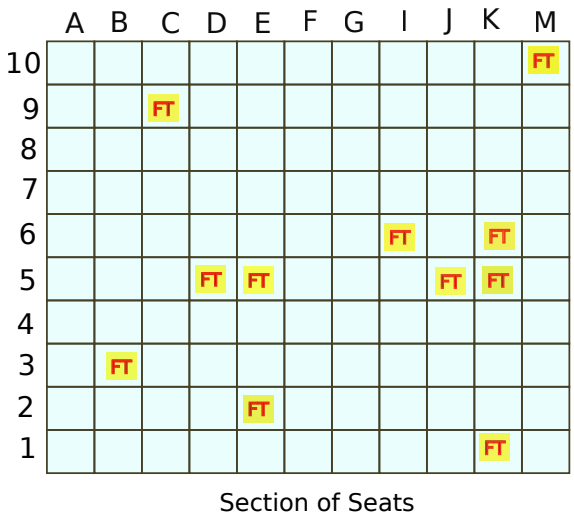

Figure 2: Fake-ticket (marked FT) holders found in seats `C9, M10, I6, K6, B3,` etc.

This will give you practice in working with *lists*, *decision structures*, *iterations* and *loops* in `Python3`. Moreover, the creation of your ftident program is a good exercise in *decomposing* a larger problem into smaller and presumably more manageable parts.

Fake-Tickets Game: An Overview:
Your fake-tickets identification program seeks to help a stadium manager ascertain visitors seated in a specific section who have entered the venue using not-genuine tickets. ftident is essentially a 1-player guessing game played on a 2-dimensional board. Here, the ftident user takes on to the role of the stadium manager checking seated spectators while the program operates "behind the scenes" and maintains full knowledge of the fans carrying either a genuine or a fake ticket.

Acting as ticket controller, the user may selectively check a specific fan *at a time* to ascertain the validity of the spectator's ticket. In this regard, the controller challenges individuals from the section one-at-a-time to "present" their ticket so she can immediately determine the type of the ticket the fan carries.

The goal of the game is for the ftident user to identify a high number of illegitimate spectators set to be 2/3 of actual fake tickets holders seated in the section.

There are essentially 3 components that make up the program:

- **The Board**: to create a functional ftident you will have to work with a grid board consisting of cells. Individual cells are identified by letters and numbers as depicted in Figure 1. Your program ftident initially "places" a number of randomly fake-ticket holders in cells as Figure 2 shows. Nevertheless, the image presented to the user is that of Figure 1; the fake-ticket holders are "hidden" from the user. As the game commences, all cells are presumed to be filled by spectators.

- **The Player**: Undertaking the role of ticket-controller, the ftident user can check individual fans by issuing a specific seat location. For example while the player looks at Figure 1, she may want to check position `C9` and so she writes the seat-coordinates on ftident's prompt. In this way, the controller identifies a spectator with a fake-ticket. Similarly, if she check `M3`

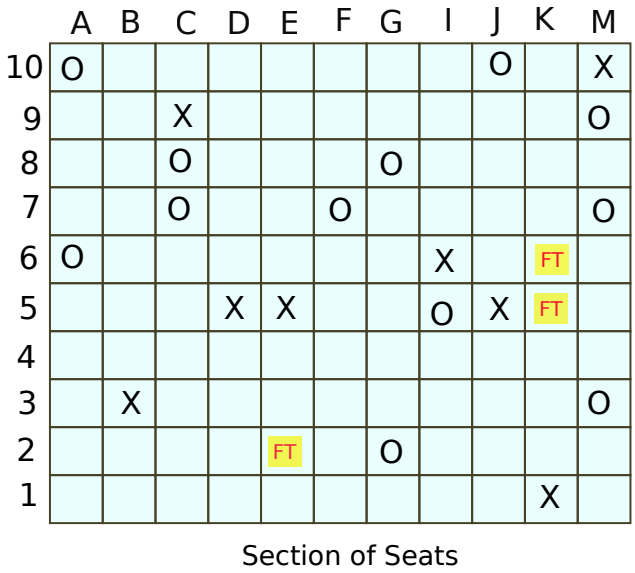

Figure 3: After 20 attempts in the game: 8 of the fake tickets have been identified.

seat, he will find that no problem exists with the person seated there.

- **The Game**: at the start of the game, the computer secretly places multiple (hidden) fake-ticket holders anywhere on the seat section (or grid). Fake-ticket holders can be seated alone or in groups. There is always a maximum number `NUMOFFAKES` of fans holding fake tickets; you can assume that `NUMOFFAKES` is (much) smaller than the number of spectators in the section.

  The location of all fake-ticket holders are initially concealed and the player of the program is called to find such seats by checking individual positions.

  If the ftident user calls a seat and the corresponding spectator has a legitimate ticket the location is marked as '`O`". Otherwise, the fan holds a fake-ticket, his position is *immediately* revealed and its respective seat on the grid is marked as "`X`"; this is also known as a **hit** and the spectator is expected to leave the game.

  The ftident user has a maximum number of attempts (`NUMATTEMPTS`) that can she can use to examine specific ticket holders. The objective of the ftident user is to come close to the existing total number of fake-tickets (`NUMOFFAKES`).

  When **at least 2/3 of the actual fake tickets** are identified by the user-player, the game is won as the manager has done a good job in finding out non-legitimate attendees. Otherwise, the player fails to identify sufficient number of illegitimate spectators.

Figures 3 shows the state-of-affairs for the game once the user has expended 20 moves and has found 8 fake-ticket holders in this specific section of seats. The game has been won as more than 2/3 of the initially 11 FTs dispersed on the grid have been identified. The remaining "hidden" illegitimate ticket holders who still enjoy the game are seated in `K6, K5` and `E2`.

ftident: Implementation

The implementation of this game mainly consists of 2 phases: 1) initialization phase and 2) game phase. In this section, you will find pertinent details for these 2 phases.

Initialization Phase:

- At the start of the game, an (empty) board is created, where columns are identified by a letter and rows by a number as Figure 1 depicts. The coordinates of the game can be set by constants that are part of your code; these constants can be set as the following snippet of code depicts:

```
XSIZE = 12       # horizontal size of grid - num of seats per row
YSIZE = 10       # vertical size of grid - num of rows
NUMOFFAKES = 25  # number of fake tickets to be dispersed to grid in groups
NUMATTEMPTS = 30 # number of attempts to be expended by the user
```

  The above definition of constants results into a section with 120 seats. Apparently, the values of the 2 constants `XSIZE` and `YSIZE` can change between executions of ftident furnishing diverse grid sizes in different game sessions.

- Your program then *randomly* places `NUMOFFAKES` fake-tickets holders throughout the grid. Such spectators can be seated individually or in groups. The seat section is always deemed full to capacity when the work of the ftident user or controller commences.

- In order to make random selection use the `randint()` function of `Python3 random` module. As the location of fake-ticket holders are initially concealed from the user, you have to find a solution to store the cells that are occupied by such spectators without revealing them to the user-player. In this regard, you *may use another board* that holds the locations of all fake-tickets. This auxiliary and hidden board is not accessible to the user and it only exists to verify a genuine (ok) or a fake ticket (hit).

- Once created, the board should be printed on the `tty` so that the player can get an idea on how it looks. Also the user should be provided with a number of checks (`NUMATTEMPTS`) she can do; this number is a portion of the total number of seats in the section. Every time the user makes a move, the number in question is automatically decremented and the remaining moves are reported on the game's display.

- Do not "hard-code" board dimensions, number and sizes of groups of fake-holders in your program statements. All these should emanate from constants that are set before the game starts. This would allow you to easily change these values without modifying multiple lines of code in your program. To keep the game logic simple, we restrict our implementation to the following:

  – The minimum board dimension is 6x4 and the maximum is 14x12. Your code does not have to incorporate smaller/larger dimensions.

  – The number of attempts a user still has available for her play as well as the time elapsed thus far has to be displayed just below the grid before the user makes her next move.

Game Phase:
After creating the board and placing the "hidden" fake-ticket holders, your game-phase implementation should consider the following:

1. Valid/Invalid User Input:

   – The player is asked to make a guess by entering a coordinate (location) within the board dimensions. The coordinate consists of the column (a letter) followed by the row (a digit).

- Make sure that the input clearly describes the format of the expected input. For example, valid coordinates are `B4, C1, A6, F0`, etc. Note that the column letter is case-sensitive.
- In case the player enters an invalid input or an invalid coordinate, the program informs the player about this and asks for a new coordinate. Examples for invalid input are `AB, 1A, A 6, 32, A;5, B,4,` etc. as well as coordinates beyond the board dimensions.

2. Processing Valid User Input:
   - After a valid user input, the program places the guess on the board, where a hit or a fake-ticket is identified is denoted as an "`X`" and a miss as a "`O`".
   - The program must maintain a score indicating the *number of guesses* at every iteration. At the end, if all allotted user-attempts have been expended the program terminates.
   - After each turn, the program must check if the game is won, i.e., more than 2/3 of the existing fake-tickets have been identified. If so, the program must inform the user that the game is over, display number of attempts carried out, and terminate. Otherwise, the board should be printed reflecting the updated cells and ask for another input.
   - Before printing the updated board, the screen should be cleared by using the following commands (add lines 2 and 7 to appropriate locations in your source code):

```
1 import random
2 import os
3
4 # your code comes here
5
6 os.system("clear")  # clears the screen
```

Grading Scheme

Description	Points (/100)
Initialization of the board and usage of constants variables	08
Printing of the board and board labels, as depicted in Figure 1	08
Randomly placing fake-tickets	10
Checking for valid input format (letter followed by number)	10
Checking for invalid input (repetitive guesses of same cell, out of board, etc.)	10
Updating the board cells based on user input	10
Detecting end of the game	08
Displaying the correct number of moves remaining after every move	06
Displaying a correct score after a win	10
Terminate the game after a win is detected	05
Use of effective comments in your program	06
Readability of code (style, variable naming, etc.)	09