Ciências ULisboa

Fundamentos de Programação

Project – Assignment 2

2023/2024

The goal of the FP Project, divided in three assignments, is that students practice the subjects that they learn in classes.

In addition to topics already covered in the first assignment, in this second assignment students will practice dictionaries, comprehensions, and functions implemeting more complex algorithms.

What is your task in this 2nd assignment?

Your mission, should you choose to accept it, is to develop a Python program that allows a single user to play a game where he must fill several bottles, each with the same "liquid" symbol. He does this by transferring "liquid" between bottles, step by step. He can only add "liquid" to a given bottle that has the same symbol as its top.

You are given a skeleton of the program in file `FP2A00.py`, where the implementation of all functions is missing. A brief description of each function follows:

- `askForExpertise()`: asks the user what level of expertise he chooses, and returns the integer value (between `MAX_EXPERT` and `LESS_EXPERT`);
- `buildGameBottles(expertise)`: given an integer `expertise` representing the user's expertise, builds and returns a structure containing the information of each and every bottle in the game – there will exist `NR_BOTTLES` bottles in total, each with a maximum capacity of `CAPACITY`; these bottles are to be (partially) filled with various symbols in `SYMBOLS`, and each bottle is named a letter from `LETTERS` (see **NOTE 1** for rules about initial filling the game bottles);
- `showBottles(bottles, nrErrors)`: given a structure `bottles` containing the information about the game bottles, and an integer `nrErrors` representing the number of errors the user has already commited, writes in the standard output a representation of the bottles in the game (see the examples in file *OutputExamples.zip*);
- `askForPlay()`: asks the user for the letters that identify the source and destination bottles for the next "liquid" transfer; returns the two letters;
- `moveIsPossible(source, destin, bottles)`: given two letters `source` and `destin` identifying the source and destination bottles, and the `bottles` structure containing the information about the game bottles, returns *True* if it is possible to transfer "liquid" from bottle `source` to bottle `destin`; returns *False*

otherwise, that is, if the top symbol of those bottles is not the same, or if the destination bottle has no space available;

- `doMove(source, destin, bottles)`: given two letters `source` and `destin` identifying the source and destination bottles, and the `bottles` structure containing the information about the game bottles, transfers as many symbols from the source to the destination bottle as possible;
- `full(aBottle)`: given structure `aBottle` containing the information about a bottle, returns *True* if the bottle is full with equal symbols; returns *False* otherwise;
- `allBottlesFull(fullBots, expertise)`: given two integers representing the current number of full bottles in the game, and the user's expertise, returns *True* if the total number of bottles that the user is supposed to fill are already full; returns *False* otherwise.

Other functions can be created, in order to better structure your code and to control the complexity of your tasks.

NOTE 1: The final goal of the game is to have each bottle full of "homogeneous liquid" (without mixings) or empty. In order to be possible to transfer liquid between bottles during the game, the bottles cannot be all full. The more empty space there is in bottles, the easier it is to do liquid transfers. So, in the beginning of the game, the program must decide how many bottles will be completely full at the end of the game.

The expertise level of the user will determine the total number of bottles that will be completely full at the end of the game, and, hence, the total number of empty spaces that must exist in the bottles.

Let us consider a game with a total number of bottles of 10, each with a capacity of 8:

- Several examples follow:
  - A player with an expertise level of 5 (the least expert), will play a game where 5 (= 10 – 5) bottles will have to be full at the end of the game, and 5 will be empty;
  - A player with an expertise level of 3 will play a game where 7 (= 10 – 3) bottles will have to be full at the end of the game, and 3 will be empty;
  - A player with an expertise level of 1 (the maximum level), will play a game where 9 (= 10 – 1) bottles will have to be full at the end of the game, and 1 will be empty;
- If N is the number of bottles that will have to be full at the end of the game, then the game will only use N different symbols (because each bottle will have to be totally full with the same symbol at the end of the game);
- The quantity to use of each of those N symbols must be 8 (which is the capacity of each bottle). Then, the total number of symbols we have to distribute initially by the 10 bottles will be 8 x N;
- This means that the total number of empty spaces in the game bottles must be 8 x (10-N).

The strategy to be used to distribute these 8 x N symbols by the 10 bottles in the game may vary, and it is students that decide how to do it. Two examples:

- Divide the 8 x N symbols by the 10 bottles, filling the remaining positions in each bottle with spaces;
- Distribute the 8 x N símbolos by the 10 bottles in a random way – in this way, among the 10 bottles there will be full bottles, empty bottles and partially full bottles (more challenging).

A TIP: In order to create a list with the 8 x N symbols you can:

1. Create a list comprehension containing 8 of each of the first N symbols of the `SYMBOLS` constant (the constant defined in program `FP2A00.py` that was supplied to you);
2. Shuffle the elements of that list using the function `shuffle` of the random module.

You are also given, in file `OutputExamples.zip`, several output examples of the execution of the `FP2A00` program, for several values of the user's *expert level*. These outputs are just to help you understand what is the desired format for the output representation of the game.

What should students deliver?

You must deliver a `.py` file similar to the one you were given – `FP2A00.py` – containing all missing function bodies AND documentation. The file name should be `FP2AXX.py` where `XX` is your group number.

You should include a header at the begining of your file, containing the group number and the names and numbers of its members.

You should present your code well aligned and legible.

To deliver: A zip file containing the `.py` file with your program.

The name of the *zip* file you deliver should also have the format `FPxx.zip` (where `xx` is the number of your group). Only one of the members of the group should deliver the file.

How to deliver?

Using the FP Moodle page. At 11:55 PM of November the 22nd, the files uploaded to Moodle will be collected.

IMPORTANT NOTE: The file you deliver is assignable only and exclusively to the members of your group; Any sign of plagiarism will be investigated and it may lead to course failure of group elements and subsequent disciplinary process.