Assignment 3

Instructions

- Unless specified otherwise,
  - you can use anything in the Python Standard Library;
  - don't use any third-party library in your code for this assignment;
  - you can assume all user inputs are always valid and require no validation.
- The underlined blue texts in the Sample Runs section of each question refers to the user inputs. It does NOT mean that your program needs to underline them.
- Please follow closely the format of the sample output for each question. Your program should produce **exactly** the same output as the sample (same text, symbols, letter case, spacing, etc.). The output for each question should end with a single newline character, which has been provided by the `print()` function by default.
- Please follow exactly the specified name, parameter list and return value(s) for the functions to be written if a question has stated (for easier grading). You may define additional functions for specific tasks or further task decomposition in any question if you see fit.
- Name your script files as instructed in each question (case-sensitive). Using other names may affect our grading and may result in slight deduction.
- Your source files will be tested in script mode rather than interactive mode.
- You are highly encouraged to annotate your functions to provide type information for parameters and return values, and include suitable comments in your code to improve the readability.

Question 1 (20%)

Question 2 (20%)

Question 2 – The Math Millionaire Game (60%)

Write a Python script `millionaire.py` to simulate the international TV game show *"Who Wants to Be a Millionaire?"* in the math flavor and a simplified manner.

In case you are not familiar with this game show, you may look at this Wikipedia page. There was a Hong Kong variant of the game show franchise, which was once a popular TV program during 2001-2005. The main goal of the game is to win HK$1 million by correctly answering 15 multiple-choice questions with 4 answer choices and increasing difficulty. Answering each question correctly can win a certain amount of money. The player can choose to walk away with the money earned thus far or to continue tackling more difficult questions to increase their prize fund at the risk of losing what they earned so far. The player has a set of *lifelines* that they may use only once to help them with a question. There are also two "safety nets" at certain levels, passing which will guarantee a certain prize fund to take away if the player gets the current question wrong.

Table 1: The game's payout structure

Question number	Question value (HK$)
15	1000000
14	500000
13	250000
12	150000
11	80000
10	60000#
9	40000
8	30000
7	20000
6	10000
5	8000#
4	4000
3	3000
2	2000
1	1000

In this assignment, we will adopt the payout structure used in the Hong Kong version of the game (Table 1), in which level 5 and level 10 (highlighted in pink and marked with #) are taken as the safety nets. That means if the player answers Question 5 right, they will earn at least HK$8,000 for sure. Answering Question 10 correctly will guarantee a prize of HK$60,000.

To simplify the implementation, you are required to implement only one lifeline in this game, which is adopted from the US version – *Jump the Question*. This lifeline allows the player to skip the current question and move on to the next one, but not to earn the money of the question they skipped. The lifeline can be used once only and cannot be used in the final question (Question 15). Note that when used in a "safety net" question, say skipping Question 5, the player moves on to Question 6 but this hasn't yet guaranteed the take-away of HK$8,000. If the player answers Question 6 wrong, they will leave the game empty-handed (HK$ 0). On the contrary, skipping Question 5 and getting Question 6 correct, the safety net of getting HK$8,000 will still be effective for subsequent levels.

Program Specification

In the program output, each question begins with a header line whose format is:
`Question n ($m):`
where `n` is the question number and `m` is the question value.

Refer to Table 1 for the corresponding prize value for each question. Examples:
- `Question 1 ($1000):`
- `Question 15 ($1000000):`

For safety net questions, there is an extra symbol # shown to remind the player.
- `Question 5 ($8000#):`
- `Question 10 ($60000#):`

All the questions in this game are simple arithmetic questions of the following structure:

x op y,

where op is one of the binary operators below:
- + : addition
- - : subtraction
- * : multiplication
- / : integer division

The two operands x and y in each question are random integers in the range from 1 to 100, inclusive. The choice of operator for a question is also picked randomly.

Each question will be displayed in the following format:

x op y = ?

For example,

20 + 15 = ?

Each question is followed by its answer list whose format is illustrated in the example below:

A. 14 B. 27 C. 35 D. 27 E. Jump F. Withdraw

The line shows 4 numeric answer choices plus two more options for using the "Jump the Question" lifeline and withdrawing from the game. Note: one of the 4 numeric choices is the correct answer. The other three answers are again random integers in the range from 1 to 100, inclusive. They could be duplicated values (e.g., both choices B and D are 27), but they must be different from the correct answer (35 in this case). The correct answer is randomly positioned in the 4-choice answer list. A single space follows each dot symbol; two spaces exist between choices. For clarity, we revise the example with the space characters shown as ␣ as follows:

A.␣14␣␣B.␣27␣␣C.␣35␣␣D.␣27␣␣E.␣Jump␣␣F.␣Withdraw

You do ***not*** have to validate the user input. We assume that ***all inputs are always valid options***. We will not test your program against invalid inputs, and the behavior of your program in such cases could deviate from that of our sample program.

Program Flow

The following suggests a program flow for your implementation. Your actual program design and variable naming may differ from what we provide here as long as they are correct.

- Prompt the user to enter an integer to seed the random number generator.
- Initialize the prize value for the 1st question, amount of money earned, and any other variables.
- Write the main loop for asking the user 15 questions, which carries out the following steps:

1. For each question, generate two random integers in [1, 100] for the operands `x` and `y`. Generate another random integer in [0, 3] or use the `random.choice()` function to pick one of the operators (+, -, *, /) to form the question: `x op y`.

2. Compute the solution `soln` (int) to the formed question `x op y`.

3. Print the question header. Remember to print # for safety net questions 5 and 10.

4. Print a line showing the question, e.g.,
   ```
   20 + 15 = ?
   ```

5. Print the answer choice line, e.g.,
   ```
   A. 14   B. 27   C. 35   D. 27   E. Jump   F. Withdraw
   ```

   In this step, generate 3 random integers in [1, 100] to be the wrong answers that mix with the correct answer. Use extra looping to ensure their values are different from the correct value `soln`. To randomly place the correct answer in the output, you may generate another random integer in [0, 3] or use the `random.choice()` function to decide which of A, B, C and D should be associated with `soln`. Save the correct choice into a variable, namely `correct` (useful for later answer checking). Refer to the pseudocode in Figure 1 for more details of how to implement this part. The option "E. Jump" should be shown only if the lifeline was not yet used.

6. Prompt the user to enter an option (character 'A', 'B', 'C', 'D', 'E' or 'F') for the question. Save it into a variable. Note that your program should accept both uppercase and lowercase, so 'a' or 'A' means the same choice. Entering 'E' while the option "E. Jump" has disappeared (i.e., the lifeline has been used) is taken as a wrong answer leading to the end of the game.

7. Set the prize value for the next question according to Table 1.

8. If the input equals 'F' (i.e., withdraw), set money earned to be the withdrawal amount saved in step 12; break the main loop to end the game.

9. If (1) the lifeline was not used, and (2) the question level is not 15 (final question), and (3) the input equals 'E', then mark the lifeline as used and continue the next iteration, thus bypassing the current question and its reward.

10. If user input does not equal `correct`, print “Wrong!” and break the main loop.

11. *(Reaching here means "correctly answered".)* Print "Right!" and save the current question value into a variable denoting the withdrawal amount.

12. Handle safety net cases (Question 5 and Question 10). Set money earned to the guaranteed value accordingly.

When the game ends, print two lines to indicate game-over and the amount of money earned, e.g.,

```
Game over!
You got $10000!
```

If the user wins $1000000, then print an extra line of congratulation:

```
Congrats! You are a millionaire!
```

As you can see above, there are several random integers to be generated in this program. In order to produce exactly the same output (same questions and answer lists) as our sample program does, the random integers must be generated in the same order used in our sample program. Please follow the pseudocode in Figure 1 for how to generate the questions and answer lists. It is important to follow our order of generating x, y, op, c, r's and the order of listing +, -, *, / and A, B, C, D in your program. If you reorder some of them, say generating y before x, then your program output will differ from our sample, and your program will fail to pass most test cases in our grading.

```
for / while:  # loop through 15 questions
   x = random int in [1, 100]         # 1st operand
   y = random int in [1, 100]         # 2nd operand
   op = random choice in [+, -, *, /]  # operator
   c = random choice in [A, B, C, D]   # pos. of correct answer
   soln = evaluate x op y

   for choice in [A, B, C, D]:
      print choice.
      if choice == c:
         print soln
         correct = choice
      else:
         do
            r = random int in [1, 100]  # a wrong answer
         while (r == soln);  # avoid duplicate with correct answer
         print r
      }
      ...
   }
}
```

Figure 1. Pseudocode for generating the questions and answer lists

Sample Runs

Sample Executable