1. Message Scramble

In this exercise, you will implement a simple encryption algorithm, one that rearranges the letters in a message. While this algorithm will make the message difficult to read, it provides little actual security and should never be used for sensitive information.

a. Given two messages, A and B, which have the same length, we can create a new message by taking the first character from A, then the first character from B, then the second character from A, then the second character from B, and so on. We'll call this the interleave of A and B.

For example, if A is the text "abcde" and B is the text "12345" the interleave of A and B is "a1b2c3d4e5".

Write a function, interleave(), that takes two strings of the same length and returns the interleave of the two.

b. To make a message even harder to read, we can perform several interleaves in a row. Assume that the length of a message is a power of 2. We define the scramble of the message recursively as follows:

1. The scramble of a single character is just that character.
2. The scramble of a longer message is found by taking the scramble of the first half of the message and the scramble of the second half of the message, and interleaving them.

For example, the scramble("12") should compute the scramble of "1", which is "1", interleaved with the scramble of "2", which is "2". The result is simply "12".

The scramble of "1234" is the interleave of the scramble of "12", which is "12", and the scramble of "34", which is similarly "34". The result is "1324".

The scramble of “12345678” can be similarly computed as “15372648".

To compute the scramble of a message with a length that is not a power of 2, first add periods (“.”) to the end until its length is a power of 2, then proceed as above.

Using this rule, the scramble of “hello” can be computed as 'hol.e.l.' The scramble of “Madam I'm Adam” can be computed as "MmmadAI.a mad'."

Write a scramble function that implements this algorithm.

Submission
Submit your program as “scramble.py”. Your program needs to read a list of input messages from “input.txt” (each line contains a message) and then generate an “output.txt” file, in which scrambled messages are listed accordingly (each line contains a scrambled message). We have attached examples of input and output files for you to test your codes before submission. Since your programs will be tested using an auto-grader system, your code must replicate the given output.txt file exactly. In addition, please follow best coding practices to make your code easy to understand.

2. Winning Strategies

Many simple games are so well understood that players can play perfectly, always choosing the best move. At that point, the outcome often becomes predetermined. The most familiar example may be tic-tac-toe. In this game, either player can force the game into a draw, no matter what moves the other player makes. In other games, such as connect 4 and chopsticks, one of the players actually has a way to win, no matter how well the opponent plays.

Mathematically, we define a winning strategy as a plan a player can use that always ends in a win. To write a winning strategy for one of these games, we can’t just fixed list a sequence of moves. The moves have to

change depending on what the opponent plays. For example, a winning strategy for tic-tac-toe has to specify a next move for every possible state of the tic-tac-toe board. You can think of it as a function from the state of the game to a next move.

Perhaps the easiest way to think about winning strategies is recursively. Let's call a state of the game hot if the current player to move has a winning strategy to finish the game - this is the state that you want to be in when it's your move. Otherwise, let's call it cold. Clearly, if a player can win in one move, the state of the game is hot. If the current player loses immediately for any available move, the state of the game is cold.

If the game is not about to end, the best strategy is to play not to lose. That is, the state of the game is hot if the current player has a move that puts the game in a cold state for the next player. Otherwise the state of the game is cold - whatever move the current player makes, the state of the game will be hot for the next player.

For this problem, let's consider a two-player version of the decreasing number game. The game begins with a number N - this is the state of the game. Two players take turns decreasing the number. Each player has two moves available: they can subtract 1 from the number, or divide it in half (not integer division - use floats for this!). The player that decreases the number below 1 is the loser.

Write a script to prompt the user for the starting state of the game, N, and compute whether that state is hot or cold. Here are some examples:

Any game state that is greater than or equal to 1 but less than 2 is cold, since the current player will immediately decrease the number below 1.

Game state 2 is hot, since no matter what move the current player makes, the next game state will be 1, which is cold (and the next player will lose automatically).

Game state 3 is hot, since the current player can divide it by 2 to reach game state 1.5, which we know is cold. Notice that the current player has to play the right move. If the player subtracted 1, she would give her opponent a game state of 2, which is hot, in which case the opponent wins. What crucial is that the current player has one move that puts the game in a cold state.

Game state 4 is cold, since the current player can put the game in state 2 or state 3, but both of these are hot.

Hint: Write a recursive function, is_hot, to check whether a state of the game is hot. For a base case, check whether the state is less than 2.

Submission

Submit your program as "strategy.py". Your program needs to read a list of input numbers representing game states from "input.txt" (each line contains an integer) and then generate an "output.txt" file, in which game states are listed accordingly (each line is "hot" or "cold"). We have attached examples of input and output files for you to test your code before submission. Since your programs will be corrected using an auto-grader system, your code is required to replicate the output.txt file exactly. In addition, please follow best coding practices to make your code easy to understand.

Optional extra: Write a program that actually plays the decreasing number game against the user, and follows a winning strategy whenever one is available. In other words, when it is the computer's turn to move, it selects a move that places the game in a cold state, if such a move is available.