Random
Problem ID: a13p03random

In the final part, you change the game such that the user neither chooses directions nor chooses to pull a lever. Instead, the program itself makes these decisions by using random functionality. The question "`Play again (y/n)`" should still be answered by the user, not randomly.

At the start of the program, the user is asked to give a seed to initialize the random number generator. After that, the program itself plays the game! When victory has been achieved, the program prints out how many moves were needed to finish the game, including both valid and invalid moves.

Have a look at this page, as an introduction to the random module in Python. You only need to use `random.seed()` and `random.choice()`.

When selecting a random choice for pulling a lever, use: `random.choice([YES, NO])` and for choosing a direction, use: `random.choice([NORTH, EAST, SOUTH, WEST])`

The `random.seed()` method can accept a string as a parameter, as well as an integer, which has a vastly different result, so take care to convert the input to `int`. Hint: Take a look at the attached starter code.

For those who are curious

In case you're wondering why we need to set `random.seed(seed)`, here's a bit of an explanation, which hopefully helps to illuminate the concept:

Randomness in computers is a bit... difficult. There are no dice inside the computer, everything is represented in binary and determined by algorithms, so there isn't really any true randomness. There are ways to try to imitate randomness, such as grabbing values from outside sources that appear somewhat random, like checking the value of some clock at the time a random number is asked for. But such methods have their limitations, and for example, if a lot of random numbers are required in a short interval of time, clock values might be very similar.

So what we do instead is using so-called "pseudo random number generators" (prng). They are essentially a sequence of numbers that appear to be very random (though in reality they are anything but). The values of the sequence are calculated using techniques from number theory, involving large prime numbers, and division with remainders (modulus). Each value of the sequence is calculated based on the previous value of the sequence, using some particular formula. So, if you know the formula, and you know the initial value of the sequence, you could calculate the rest of the sequence, making the sequence entirely predictable, and not at all random.

You can think of the "seed" as the initial value of the sequence. The pseudo random number generator of python (`random`) uses a specific formula, so if we also use a specific seed, then that determines the sequence entirely. We can supply a specific seed into the prng by calling `random.seed()` and giving it the seed value as a parameter.

Normally, we don't want to fix the seed in this way, as it removes the randomness, but in case of the automatic tests on Gradescope, we need to remove the randomness so that we can compare the output to judge (automatically) if your code is behaving correctly.

Input

The first line l_0 of the input will contain an integer s, the seed for the random number generator.

Then follows a sequence $l = (l_1, \ldots, l_n)$ of n lines, where each line l_i should be interpreted as containing *an answer to the question whether the player wants to play again*. The last line will correspond to the player declining another round, *while all the other lines will contain an affirmative answer*.

In the tests, n will be restricted to $n \leq 10$. The last line l_n will always be "`N`" or "`n`", while lines $l_1, \ldots, l_{n-1}$ will each be "`Y`" or "`y`". Also, $0 \leq s < 10\,000$.

Note that since the sequence of moves is up to the random number generator, we can not guarantee a maximum on the number of moves it will take for each game to finish, but they will finish eventually.

Output

In short, the output should be as in the previous problem, with two exceptions:

1. Now, after each direction prompt and lever prompt, the choice selected by the random number generator should appear after the prompt. See the samples below.

2. Each victory report should now include the number of moves made (or attempted) throughout the player's journey from the starting location to the goal tile during that particular run of the game, like so:

   - "`Victory!  Total coins {`C`}.  Moves {`M`}.`", without quotations,

   where C is the total amount of coins the player received, as before, and M is the total moves made, including both valid moves and invalid attempts. In other words, using the terminology from the previous problems, M is the total number of direction blocks in that particular game block of the output, and C is the total number of lever blocks containing an income report.

Note that the output in the samples is too long to fit on the pages below, and gets truncated, so you can't see all of it here unfortunately. But hopefully you see enough to get the idea. You can see more by submitting on Gradescope, unless you pass the sample test cases, in which case you probably don't need to see more of the output anyway.

Sample Input 1	Sample Output 1
10 n	Input seed: You can travel: (N)orth. Direction: n Pull a lever (y/n): n You can travel: (N)orth or (E)ast or (S)outh. Direction: w Not a valid direction! You can travel: (N)orth or (E)ast or (S)outh. Direction: n You can travel: (E)ast or (S)outh. Direction: e Pull a lever (y/n): n You can travel: (E)ast or (W)est. Direction: w You can travel: (E)ast or (S)outh. Direction: s Pull a lever (y/n): y You received 1 coin, your total is now 1. You can travel: (N)orth or (E)ast or (S)outh. Direction: n You can travel: (E)ast or (S)outh. Direction: w Not a valid direction! You can travel: (E)ast or (S)outh. Direction: s Pull a lever (y/n): y You received 1 coin, your total is now 2. You can travel: (N)orth or (E)ast or (S)outh. Direction: e Pull a lever (y/n): n You can travel: (S)outh or (W)est. Direction: n Not a valid direction! You can travel: (S)outh or (W)est. Direction: w Pull a lever (y/n): y You received 1 coin, your total is now 3. You can travel: (N)orth or (E)ast or (S)outh. Direction: s You can travel: (N)orth. Direction: w Not a valid direction! You can travel: (N)orth. Direction: w Not a valid direction! You can travel: (N)orth. Direction: s Not a valid direction! You can travel: (N)orth. Direction: s Not a valid direction! You can travel: (N)orth. Direction: w Not a valid direction! You can travel: (N)orth. Direction: e Not a valid direction! You can travel: (N)orth. Direction: s Not a valid direction!

Sample Input 2	Sample Output 2
10 y y y y n	Input seed: You can travel: (N)orth. Direction: n Pull a lever (y/n): n You can travel: (N)orth or (E)ast or (S)outh. Direction: w Not a valid direction! You can travel: (N)orth or (E)ast or (S)outh. Direction: n You can travel: (E)ast or (S)outh. Direction: e Pull a lever (y/n): n You can travel: (E)ast or (W)est. Direction: w You can travel: (E)ast or (S)outh. Direction: s Pull a lever (y/n): y You received 1 coin, your total is now 1. You can travel: (N)orth or (E)ast or (S)outh. Direction: n You can travel: (E)ast or (S)outh. Direction: w Not a valid direction! You can travel: (E)ast or (S)outh. Direction: s Pull a lever (y/n): y You received 1 coin, your total is now 2. You can travel: (N)orth or (E)ast or (S)outh. Direction: e Pull a lever (y/n): n You can travel: (S)outh or (W)est. Direction: n Not a valid direction! You can travel: (S)outh or (W)est. Direction: w Pull a lever (y/n): y You received 1 coin, your total is now 3. You can travel: (N)orth or (E)ast or (S)outh. Direction: s You can travel: (N)orth. Direction: w Not a valid direction! You can travel: (N)orth. Direction: w Not a valid direction! You can travel: (N)orth. Direction: s Not a valid direction! You can travel: (N)orth. Direction: s Not a valid direction! You can travel: (N)orth. Direction: w Not a valid direction! You can travel: (N)orth. Direction: e Not a valid direction! You can travel: (N)orth. Direction: s Not a valid direction!