coding challenge

Introduction
This challenge will be used to assess your general programming competence. You will be assessed on various factors such as coding style, design and efficiency of the solution.

You may use any external libraries or code which is openly available online. Your solution should use tools that you can reasonably expect us to have or have easy access to.

Challenge description
The challenge has four parts to it. The solutions can be implemented independently of each other, however you will be asked to use the result from one solution as part of the evaluation of another.

1. Finding Collisions in Hashes
Develop a brute force hash iterator function, which takes the following input:

- An ASCII encoded string (the salt)
- An integer (the number of prefixed zeros)

Example: abc 5

See here for an explanation of hash functions.

Your function will take the salt and find the first MD5 hash in which the hexadecimal representation is prefixed with the number of zeros specified, and use that to determine one of the characters in the output. Your function will repeat this step until you have all the characters i.e. a complete solution.

Example:

- For “abc 5” you would calculate the MD5 hash of abc1, then abc2, then abc3...
- A hash indicates the next character in the password if its hexadecimal representation starts with the number of zeros specified (in this case, five zeroes).
- When you find a hash which represents the next character in the password, the character in the hash after the predefined 0s (e.g the 6th character for “abc 5”) shows the index in which to put that character.
- In order to find the character to put in that indexed position, follow this process:
    - Take the number of attempted tries, modulo the length of the hash (32 bits)
    - The character you should take is then the character at this index position in the hash
- For example, on the 313913 iteration of “machine-learning 4” you hash “machine-learning313913” and get 00007f207750dd2de4a844e2c**1**a4129e.

    - This has 4 zeros at the beginning, and so is the output you need to find the next character
    - The character is at 313913 % 32 = the 25th position, therefore select “1”, highlighted in bold above
    - The 5th character of the hash is 7, so the 7th character in the answer is “1”
- Therefore, 1 goes in position 7: .....1..
- Your answer should always be 10 characters long. Use only the first result for each position, and ignore invalid positions. E.g ignore a hash of 00000g…
- Iteratively write these characters to console and when you have finished write to a file.
- Some example inputs and outputs to help you debugging
    - **Input**: 'machine-learning', 4 **Output:** f320e001d1
    - **Input** 'artificial-intelligence', 5 **Output:** 610d370320
    - **Input** ‘code-quality’,3 **Output:** 09e97089ae

2. Parsing Function

Develop a parsing function which, given a directory of files, will output an ASCII string displaying a graph marking a set of coordinates.

Your input will be a string pointing to a directory location containing a list of .txt files. These files will contain a list of x,y coordinates in the format “x4y1”.

The function will parse all the files in the specified directory.

- Each file contains numbers in the following form: “x4y1, x1y2, x5y2”. The first digit is the x coordinate, the second is the y coordinate. The origin is at 0, 0 in the top left corner, and the coordinates are always positive integers.
- Each file can contain any number of coordinates.
- Read all files in the specified directory. Once these are all read, then write to a single new file with ‘x’ if that point is specified, and ‘.’ if it isn’t.

For example:

Input: x4y1, x1y2, x5y2

Output:

```
......
....x.
.x...x
```

3. Distance between two points

We will use the output of the previous challenge as a map for a maze. We can imagine that the ‘.’ characters form paths and the ‘x’ characters are walls. Thus a person walking through the maze can only travel along the ‘.’ characters in the map.

You should take a map, a start x and y coordinate and an end x and y coordinate.

If you haven't done something like this before it might be helpful to read pathfinding on wikipedia for suggestions of algorithms to use.

Develop a function which takes the map, and the two points as input, and which outputs the shortest path from start to end. You can only move directly up, down, left, or right (no diagonal moves), and you can't go outside the map. For some maps there may be multiple shortest paths, so just output 1 valid path.

Your output should show the original map, with the path you have taken represented by the character O printed on top.

For example, if you have the following map, where S is the start and E is the end

```
....x
..x..
S.x.E
..x..
..x..
```

You could give the solution

```
0000x
0.x00
S.x.E
..x..
..x..
```

4: Integration

Finally we want you to integrate all the code from these challenges and write a function that takes a number of zeros, a set of start/end coordinates and a directory, and outputs a map with the path you have taken with the character O, along with the output of the hash function in part 1 run on the path's coordinate string with the predetermined number of zeros.

As an example, let's assume your start coordinate is (00,02) and your end coordinate is (05,02). If you had x2y0, x2y1, x2y2, x5y4 and x0y3 as values in your directory you would get a map of

```
..x...
..x...
S.x..E
x.....
......
```

Therefore you should output a file which could be in any the following forms

```
..x...
..x...
S0x..E
x00000
.....x

..x...
..x...
S0x.0E
x0000.
.....x

..x...
..x...
S0x00E
x000..
.....x
```

You should take the path with the **lowest** sum of coordinate values (i.e. add up all the x and y values of every coordinate on the path), and input that path written as x1y1x2y2x3y3… to the hash function you generated in part 1. For example, the lowest sum of coordinate values in the above example is the final path shown, and so you would enter 000201020103020303030204020502 as the input to your hash function.

What you should submit

Please submit your solution in the form of an archive (zip, tar etc.).

You should submit all of the code which you have written, including files needed to help us easily build and run your solution.