Rock-Paper-Scissors

Stage 1/5: Unfair computer

Description

Rock paper scissors is a popular hand game. Two players simultaneously form one of three shapes with their hands, and then, depending on the shapes, one player wins — rock beats scissors, paper wins over rock, scissors defeats paper. The game is well-known for fair win-conditions between equal options.

Let's start with an unfair version! :)

Write a program that reads input that states which option users have chosen. After this, your program must make users lose! That is, your program should always select an option that defeats the one picked by users. Once it finds this option, output the line `Sorry, but the computer chose <option>`, where `<option>` is the name of the option that the program's picked depending on the user's input.For example, if users enter `rock`, the program should print `Sorry, but the computer chose paper` and so on.

Pay attention to the format of the output. It should follow the one in the example, including capital letters and punctuation. Do not print additional strings.

Just think about it: in this stage, the computer has to win every time.

- if users choose `paper`, the computer chooses `scissors` (the computer wins);
- if users choose `scissors`, the computer chooses `rock` (the computer wins);
- if users choose `rock`, the computer chooses `paper` (the computer wins).

Objectives

Your program should:

- Take input from the user;
- Find an option that wins over the user's option;
- Output the line: `Sorry, but the computer chose <option>`.

Examples

The greater-than symbol followed by a space (`> `) represents the user input. Note that it's not part of the input.

Example 1:

```
> scissors
Sorry, but the computer chose rock
```

Example 2:

```
> paper
Sorry, but the computer chose scissors
```

Stage 2/5: Equalizing chances

Description

Well, now let's do something more tangible. Nobody wants to play the game where you always lose. We can use the power of the `Random` class to make the game a bit more challenging.

Write a program that reads input from users, chooses a random option, and then says who won: a user or the computer. There are a few examples below to provide the output for any outcome (`<option>` is the option chosen by your program):

- Loss: `Sorry, but the computer chose <option>`;
- Draw: `There is a draw (<option>)`;
- Win: `Well done. The computer chose <option> and failed`;

Objectives

Your program should:

- Read the user input that includes an option;
- Choose a random option;
- Compare the options and determine the winner;
- Output one of the lines above depending on the result of the game.

Examples

The greater-than symbol followed by a space (`> `) represents the user input. Note that it's not part of the input.

Example 1:

```
> rock
Well done. The computer chose scissors and failed
```

Example 2:

```
> scissors
There is a draw (scissors)
```

Example 3:

```
> paper
Sorry, but the computer chose scissors
```

Stage 3/5: Endless game

Description

We came up with a really cool idea in the previous stage. But the game is really short. Nobody plays a single shot of rock paper scissors. We need to find a way to run the game forever. Not literally, though — let's implement a way to stop your program.

Improve your program so that it will take an unlimited number of inputs until users enter `!exit`. After entering `!exit`, the program should print `Bye!` and terminate. Also, let's try to handle invalid inputs: your program should be ready to handle typos in user input, or when there's a mishmash instead of a normal command. So, if the input doesn't correspond to any known command (an option or `!exit`), your program should output the following line: `Invalid input`.

Objectives

Your program should:

- Take input from users;
- If the input contains `!exit`, output `Bye!` and stop the game;

- If the input is the name of the option, then pick a random option and output a line with the result of the game in the following format (`<option>` is the name of the option chosen by the program):
  - Loss: `Sorry, but the computer chose <option>`
  - Draw: `There is a draw (<option>)`
  - Win: `Well done. The computer chose <option> and failed`
- If the input corresponds to anything else, output `Invalid input`;
- Repeat it all over again.

Example

The greater-than symbol followed by a space (`> `) represents the user input. Note that it's not part of the input.

Example 1:

```
> rock
Well done. The computer chose scissors and failed
> paper
Well done. The computer chose rock and failed
> paper
There is a draw (paper)
> scissors
Sorry, but the computer chose rock
> rokc
Invalid input
> xit!
Invalid input
> !exit
Bye!
```

Stage 4/5: Scoring the game

Description

People love to see their results as they're advancing to their goals. So, let's learn how to show the scoreboard!

When the game starts, users must be able to enter their names. After that, the program should greet users and read a file named *rating.txt*.

In order to execute the test program correctly, name the file *rating.txt* and save it in the current directory, without subdirectories. In tests

this file will be created, pre-filled with data. Make sure that you read
the data correctly when you start the game.

It is the file
that contains the current scores for different players. You can see the
file format below: lines containing usernames and their scores, divided
by a single space:

```
Tim 350
Jane 200
Alex 400
```

Take the current user score from the file and
use it as a basis for counting the score during the game. For example,
if a user entered `Tim`, then their score at the start of the game is `350`. If a user inputs
a name that is not on the list, the program should count the score from `0`.

No need to write anything to the *rating.txt* file.

Print the user score with the `!rating` command. For example, if your rating is `0`,
then the program should print:

```
Your rating: 0
```

Add `50` points for every draw, `100` for every win, and `0` for losing.

Objectives

Your program should:

- Output a line `Enter your name:`. Users enter their names on the same line (not the one following the output!);
- Read input with the name and output a new line: `Hello, <name>`
- Read *rating.txt* and check whether it contains an entry with the current username. If yes,
  use the score specified in the file as a starting point. If not, start
  the score from `0`.
- Play the game by the rules defined in the previous stages and read the user input;
- If the input is `!exit`, output `Bye!` and stop the game;

- If the input is the name of the option, then pick a random option and output a line with the result in the following format (`<option>` is the name of the option chosen by the program):
  - Loss: `Sorry, but the computer chose <option>`
  - Draw: `There is a draw (<option>)`
  - Win: `Well done. The computer chose <option> and failed`
- For each draw, add `50` points to the score. For each win, add `100` points. In case a user loses, don't change the score;
- If the input corresponds to anything else, output `Invalid input`;
- Restart the game.

Examples

The greater-than symbol followed by a space (`> `) represents the user input. Note that it's not part of the input.

Example 1:

```
Enter your name: > Tim
Hello, Tim
> !rating
Your rating: 350
> rock
Sorry, but the computer chose paper
> paper
Well done. The computer chose rock and failed
> scissors
There is a draw (scissors)
> !rating
Your rating: 500
> !exit
Bye!
```

Example 2:

```
Enter your name: > Chuck
Hello, Chuck
> scissors
There is a draw (scissors)
> rock
Well done. The computer chose scissors and failed
> paper
Well done. The computer chose rock and failed
> !rating
Your rating: 250
```

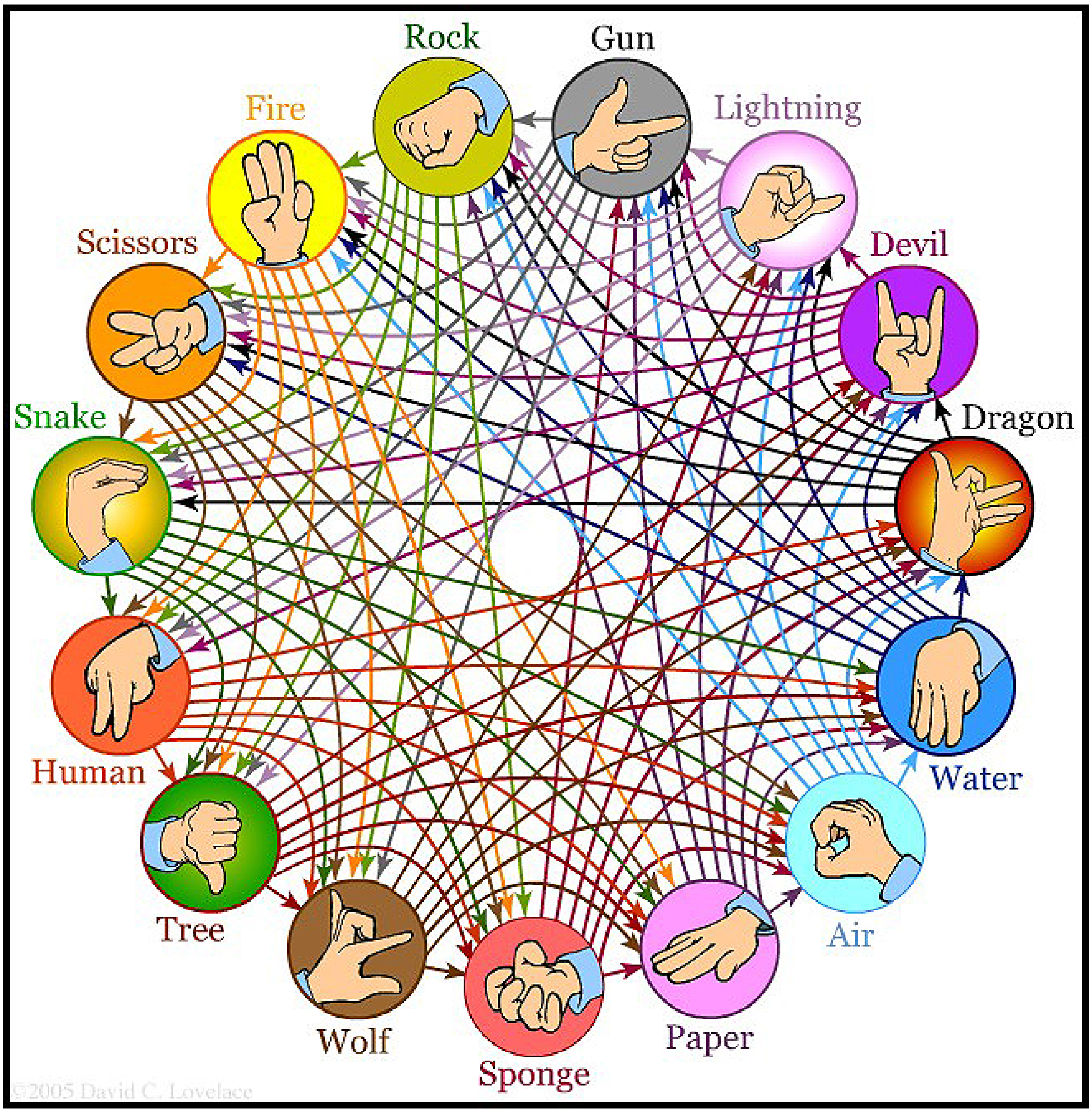
```
> !exit
Bye!
```

Stage 5/5: More options

Description

How about new game rules? The original game has a fairly small choice of options.

The extended version of the game makes it hard to draw. Now, your program should accept alternative lists of options, like `Rock`, `Paper`, `Scissors`, `Lizard`, `Spock`, and so on. You can take the following options (don't take their relations into account; we'll speak about them further on):



In this stage, before the game starts, users can choose the options. After entering the name, they should provide a list of the options separated by a comma. For example:

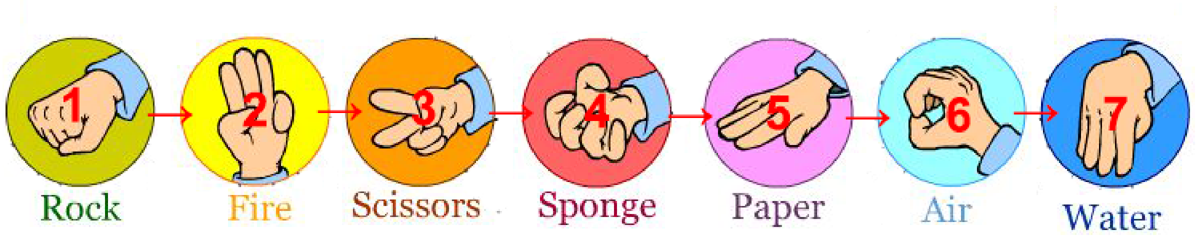
```
rock,paper,scissors,lizard,spock
```

If users input an empty line, start the game with default options: `rock`, `paper`, and `scissors`.

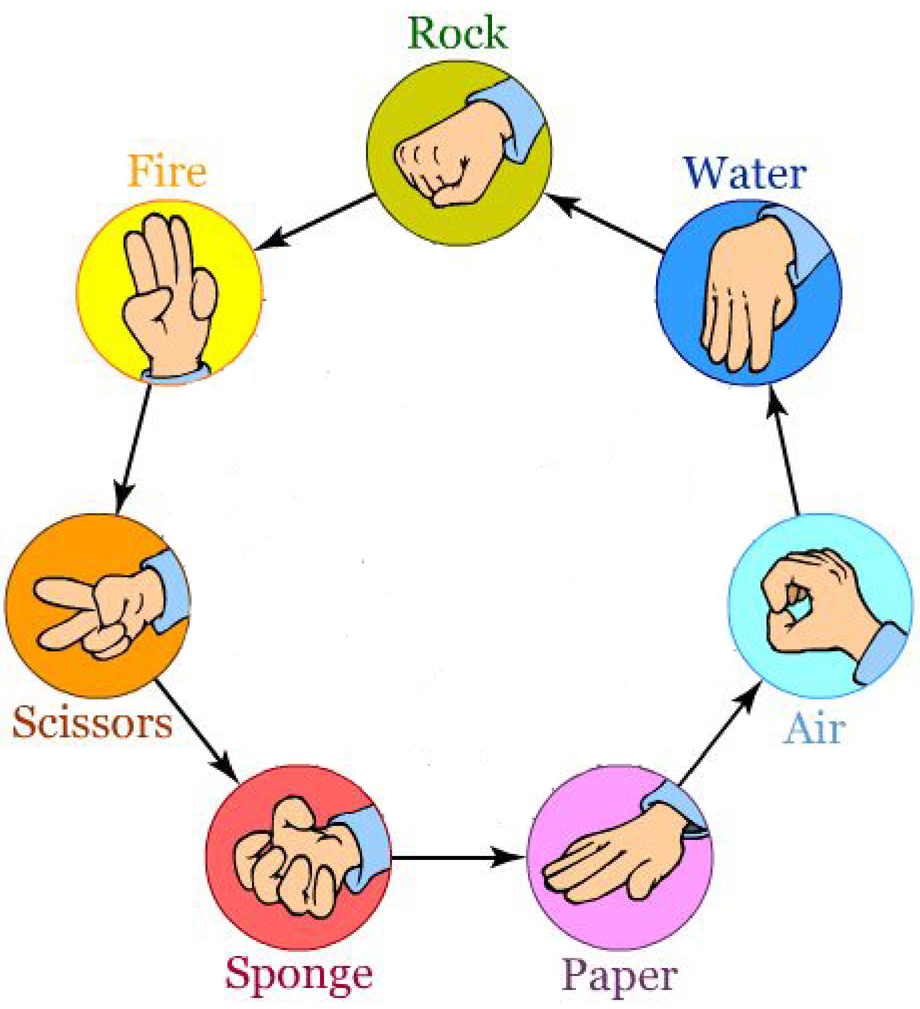
Once the game options are defined, output `Okay, let's start`.

Regardless of the chosen options, your program, obviously, should identify which option beats which. You can use the following algorithm.

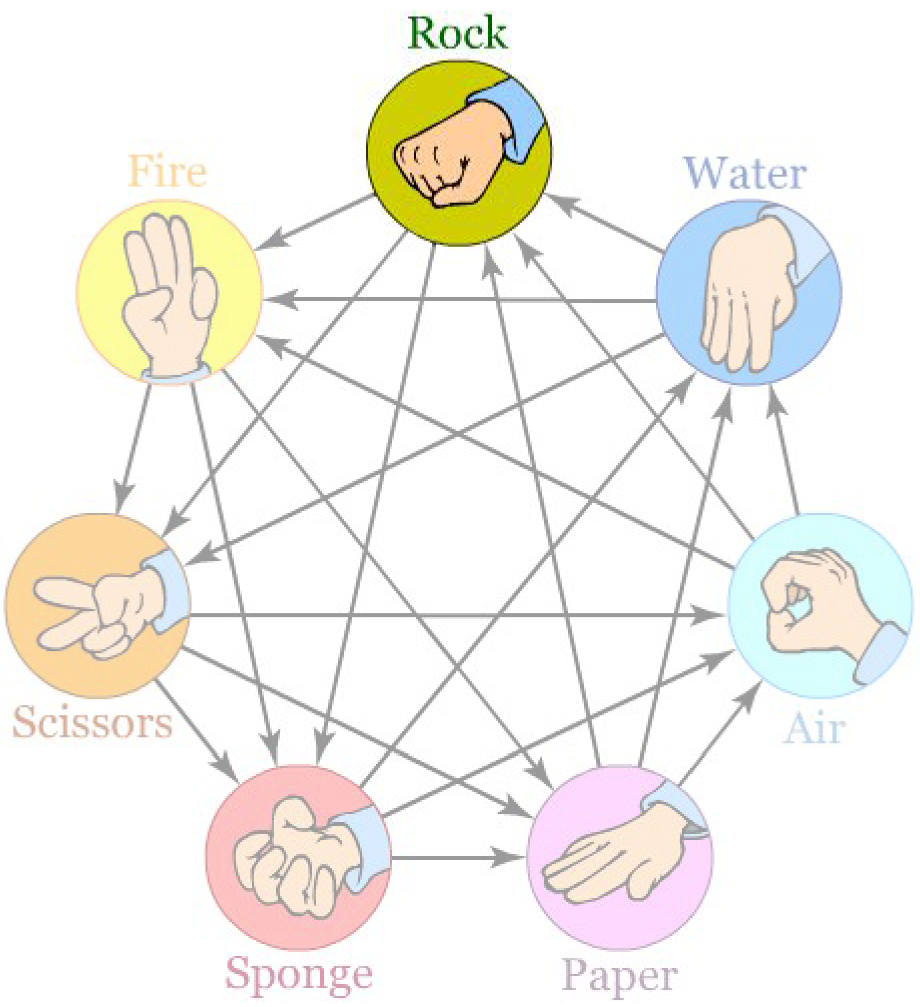
Let's imagine that the following options are involved in the game "Rock Fire Scissors Sponge Paper Air Water". Order is important.



Let us represent this line as a closed circle:



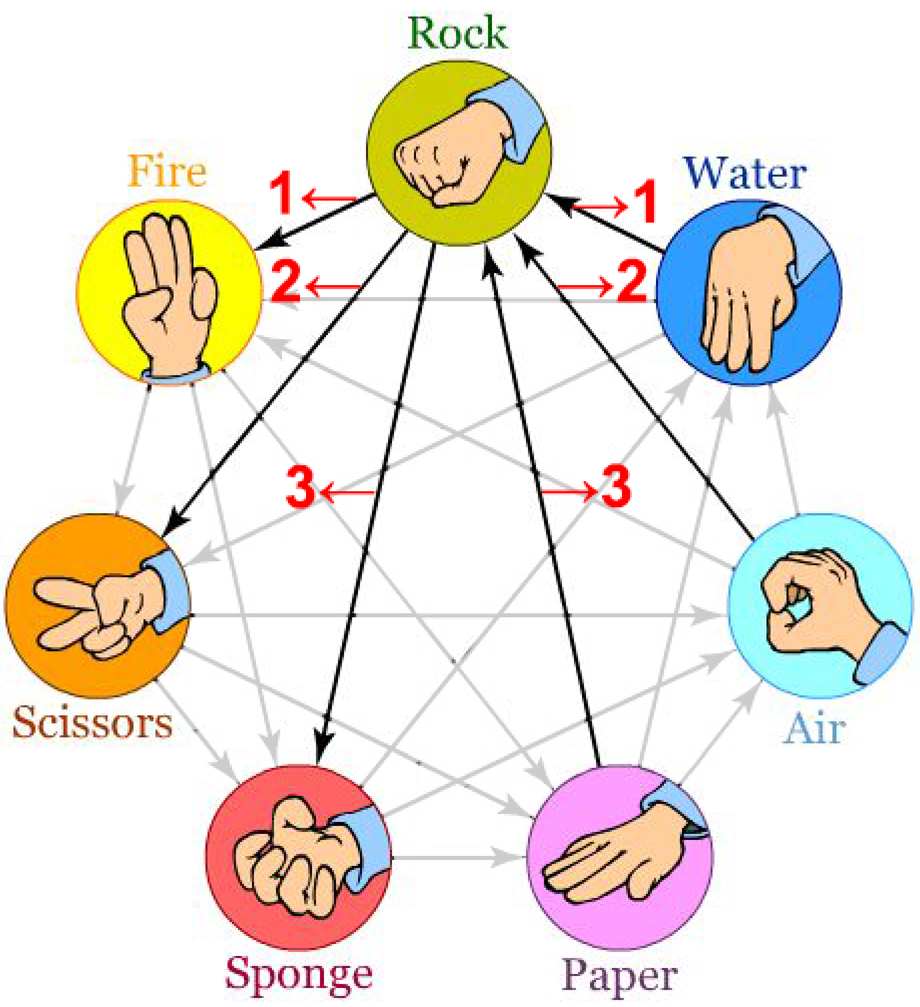
First, every option produces a draw when opposed to itself.



If Rock opposed Rock - it's a draw.

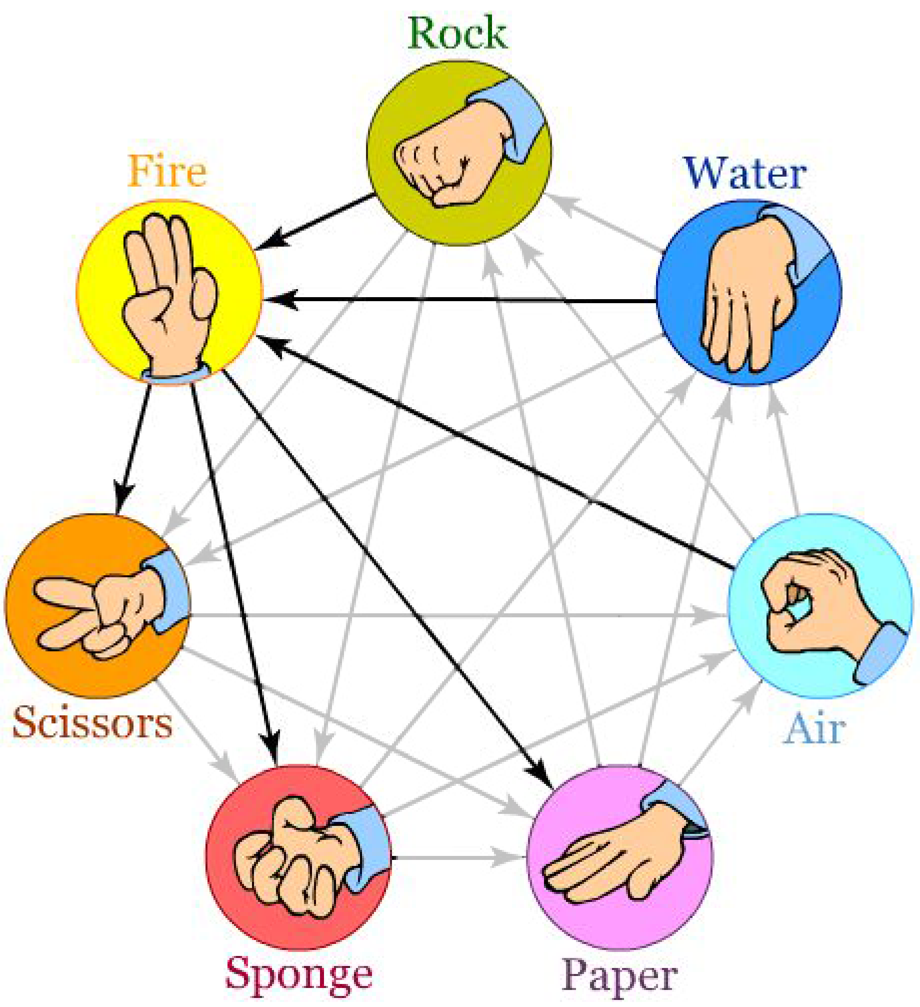
Secondly, every option beats half of the other options and is defeated by another half.

For "Rock":



The first half of the options after Rock
are Fire Scissors and Sponge (Rock beat it). Another half (after first
half) - Paper, Air and Water are defeated Rock.

For "Fire":



The first half of the options after Fire
are Scissors, Sponge and Paper (Fire beat it). Another half (after first
half) - Air, Water and Rock are defeated Fire.

And so on

How to determine which options are stronger or weaker? Take the list
of options provided by the user and pick the option that you want to
know the relationships of. Take all other options from the user's list.
Add them to the list of options that precede the chosen option. Now, you
have another list of options that don't include the user's option with a
different order of elements inside. First are the options that follow
the chosen one in the original list; then, there are the ones that
precede it. So, in this "new" list, the first half of the options defeat
the "chosen" option, and the second half is beaten by it.

Once again, never mind the "links" between the options in the picture above. They
are only for illustration purposes

For example, the user's list of options is `rock,paper,scissors,lizard,spock`. You want to know what options are weaker than `lizard`. By looking at the list `spock,rock,paper,scissors` you realize that `spock` and `rock` beat `lizard`. `Paper` and `scissors` are defeated by it. For spock, it'll be almost the same, but it'll get beaten by `rock` and `paper`, and prevail over `scissors` and `lizard`.

Of course, this is not the most efficient way to determine which option prevails over which. You are welcome to try to develop other methods of tackling this problem.

Objectives

Your program should:

- Output a line `Enter your name: `. Users enter their names on the same line (not the one following the output);
- Read the input with the username and output `Hello, <name>`;
- Read *rating.txt* and check whether it contains an entry with the current username. If yes, use the score specified in the file as a starting point. If not, start the score from `0`;
- Read the input with the list of options for the game (options are separated by comma). If the input is an empty line, play with the default options;
- Output a line `Okay, let's start`;
- Play the game by the rules defined in the previous stages and read the user's input;
- If the input is `!exit`, output `Bye!` and stop the game;
- If the input is the name of the option, then pick a random option and output a line with the result of the game in the following format (`<option>` is the name of the option chosen by the program):
    - Loss: `Sorry, but the computer chose <option>`
    - Draw: `There is a draw (<option>)`
    - Win: `Well done. The computer chose <option> and failed`
- For each draw, add `50` points to the score. For each user's win, add `100` to their score. In case of a loss, don't change the score;
- If input corresponds to anything else, output `Invalid input`;

- Restart the game (with the same options defined before the start of the game).

Examples

The greater-than symbol followed by a space (`> `) represents the user input. Note that it's not part of the input.

Example 1:

```
Enter your name: > Tim
Hello, Tim
> rock,gun,lightning,devil,dragon,water,air,paper,sponge,wolf,tree,human,snake,scissor
s,fire
Okay, let's start
> rock
Sorry, but the computer chose air
> !rating
Your rating: 0
> rock
Well done. The computer chose sponge and failed
> !rating
Your rating: 100
> !exit
Bye!
```

Example 2:

```
Enter your name: > Tim
Hello, Tim
>
Okay, let's start
> rock
Well done. The computer chose scissors and failed
> paper
Well done. The computer chose rock and failed
> paper
There is a draw (paper)
> scissors
Sorry, but the computer chose rock
> !exit
Bye!
```