Potential enhancements and refinements

The solution meets user requirements, allowing the user to play the game and guess the random number correctly. The user input has been validated to ensure that the solution is robust. The validation includes the use of a try: catch when checking the format of the data input. This prevents the program from crashing if the input is not a number, so the user has an opportunity to re-enter the number, rather than restarting the program.

The code has comments that explain the functions and is structured logically, so making an efficient solution that can be easily maintained.

The main problem encountered was how the user input was handled and there are some alternatives that can be considered. In my solution I originally used the input() function. When I tested the solution this caused a problem when validating the input.

The python function input() reads the keyboard input and makes a decision on how to format the data based on what the user has typed. For example if the user types in a number such as 1234 then input() will set the data variable to an integer (so in this case user_input is an integer if 1234 is entered).

```
while user_input != "exit":
    print("Please enter a 4 digit number or exit")
    user_input = input("number: ")
```

The advantage of using input() is that python makes a decision without needing code to test if the number is an integer or a string. Code is shorter and less complex. When comparing the randomly generated number in my example the code will be comparing “like with like”, as the random number returns an integer. This could have performance benefits as there is no need for extra instructions to the processor to convert data types.

This would be fine if I was comparing the random and user input numbers as a whole. However I needed to compare them digit by digit and an integer data type does not allow this. There are a few ways to do this in python but all involve converting the user input into a string and then mapping into a list. This would make the code more complex and less efficient as data would have to be evaluated twice and potentially converted twice.

Therefore I used the raw_input function that always assumes a string input. This could then be tested once to see if it was an integer, for validation. It could then be compared digit by digit to the randomly generated number that had been formatted as a list originally.

An Improvement to the functionality of the solution could be considered. For example it could be possible to save a game half way through, so that the user could resume guessing the number. This would require a way of saving the random number and number of guesses into some data store.

A history of games could also be created, and a way of entering user names and playing against another user, to compare guesses.

This was not required as part of this solution however.