Number-Guessing Game
(Computer As Guessing Role)

OVERVIEW

In this assignment, you are going to design and develop an interactive number-guessing game. One participant (as host) is asked to pick a fixed length number (as secret number) and another participant (as game-player) is to guess the chosen number repeatedly. In the beginning of each game, a secret number is first picked by the host with no repeated digits, then repeatedly prompt the "game-player" for a "guess" of the secret number, without repeated digits. In return after each guess, the game-player will be given some feedbacks about the guess. Based on the feedback, the game-player continues to guess until the secret number is correctly guessed or the maximum number of attempts is reached.

The feedback provided to the game-player by the host contains 2 pieces of information as follows:

1. How many of the digits in the "guess" also appear in the "secret number" (as Correct)
2. How many of the digits from "Correct" appear in the same position as secret number's (as Exact)

For instances, say the chosen secret number is "2468", the following table shows the feedback as a result of each guess provided by the game-player:

Guess	Correct	Exact	Remark
1357	0	0	None of the digits (1,3,5 & 7) are from 2468
2890	2	1	Only digits 2 & 8 appear in 2468 and digit 2 is in the exact position
8294	3	0	Only digits 2, 4 & 8 appear in 2468 and none of 3 digits appear in the exact position
2486	4	2	All digits from "guess" also from 2468 and only digits 2 & 4 appear in the exact position
2468	4	4	Guess matches secret number

You are asked to design and develop a program to be the guessing role. In other words, the computer program as "game player" makes the guess and you as "host" provides the computer program with the feedback interactively.

SCOPE

1. You are free to implement your program for 3-digit or 4-digit guessing, in python.
2. In the beginning of each game, indicate to the host the number of digits your program is designed to guess.
3. Inform the host to choose a "secret number" with no repeated digits and have it written down somewhere.
4. Prompt "host" to start the game
5. Your program as "game-player" starts to guess, after each guess, prompt "host" for feedback as described in the "Overview" session:
    a. How many correct digits?
    b. How many exact digits? (skip this question if Correct is 0)
6. Your program is required to verify the responses returned by the "host". If needed, repeatedly prompt "host" to re-enter the correct information, such examples as:
    a. Non-numeric responses
    b. Value of "Exact" is greater than that of "Correct"
    c. Value of either "Correct" or "Exact" should be less than the length of the secret number (3 or 4)
    d. Inconsistent responses
7. Feedback History
    a. Keep a list of history of all guesses including corresponding feedback, and then display the entire list after each guess in chronological order
8. Termination
    a. the current guess matches the secret number
        i. In this case, display a congratulation message
    b. the number of guess attempts reaches the limit
        i. Display a game-over message
    c. the host enters "x" on any of the 2 feedback (Correct or Exact)
9. Termination Prompt
    a. After termination, prompt "host" to quit or to continue a new game.
10. Game Startup
    a. Show a welcome message to briefly describe what your program does
    b. Then prompt "host" for the secret number prior to starting the game
11. For 3-digit number implementation, your program MJST successfully guessed the secret number within 20 attempts.
12. For 4-digit number implementation, your program MJST successfully guessed the secret number within 40 attempts.

STARTUP OPTIONS

Not applicable

SKILLS

In this assignment, you will be trained on the use of the followings:

- Use input() to prompt user for information
- Use standard objects (strings, numbers & lists)
- Control statements to interact with users
- Variable Scope
- String formatting (method style)
- Functions for program structure and decomposition

DELIVERABLES

1. Design documentation (A1_School_StudentID_Design.doc)
2. Program source code (A1_School_StudentID_Source.py)
3. Output (A1_School_StudentID_Output.doc)

Zip all files above in a single file (A1_School_StudentID.zip) and submit the zip file by due date to the corresponding assignment folder under “Assignment (submission)”

For instances, a SME student with student ID “119010001”:

- A1_SME_119010001.zip:
    - A1_SME_119010001_Design.doc
    - A1_SME_119010001_Source.py
    - A1_SME_119010001_Output.doc

5% will be deducted if any files are incorrectly named!!!

DESIGN DOCUMENTATION

For the design document provide write-up for the following 2 chapters:

1. Design:
    a. Describe your strategy for guessing
    b. Describe the core data structures used for tracking the guess history and feedback
    c. Describe the structure of the program (functions, variables and program flow)
2. Test Strategy:
    a. How to verify that your program works as per "Scope" – create a test plan
    b. Describe different plans for different scenarios

TIPS & HINTS

- Format the feedback for each guess using format() (number guessed, correct : xx, position: xx)
- Use a list to keep all previous guesses & feedback as strings
- Beware of variable scope as you might keep a few variables as global such as guess results and guess count.
- Refer to python website for program styles and naming convention (PEP 8)
- Try a guessing algorithm (solution) with a simple strategy in 2 parts. The first part is to determine all the correct digits (not concerning the position); then the second part simply takes the digits form part I and reorders them to match the secret number.
- Split up the digits in logical groups and make first guess attempt for each group. Based on different feedback, identify the digit(s) that is part of the secret number or not. Use the result discovered to sequentially determine the remaining digits one by one. Once all required digits are discovered, simply shuffle the digits until a correct guess is achieved.
- Gradually optimize your guessing solution with results from "part I" to make "part II" more efficient.

SAMPLE OUTPUT – SUCCESSFUL GUESS

SECRET NUMBER = 487

Guess 1/15: 123
How many correct digits? 0

Guess 1/15: 123 Correct=0 Exact=0

Guess 2/15: 490
How many correct digits? 1
How many exact digits? 1

Guess 1/15: 123 Correct=0 Exact=0
Guess 2/15: 490 Correct=1 Exact=1

Guess 3/15: 478
How many correct digits? 3
How many exact digits? 1

Guess 1/15: 123 Correct=0 Exact=0
Guess 2/15: 490 Correct=1 Exact=1
Guess 3/15: 478 Correct=3 Exact=1

Guess 4/15: 487
How many correct digits? 3
How many exact digits? 3

Congratulations!! Start a new game (y/n)?

SAMPLE OUTPUT – INCORRECT FEEDBACK

SECRET NUMBER = 487

Guess 1/15: 123
How many correct digits? 4
Incorrect feedback, must be < 4
How many correct digits?

SECRET NUMBER = 487

Guess 1/15: 123
How many correct digits? 1
How many exact digits? 2
Incorrect feedback, exact digit must be <= 1!
How many exact digits?

SAMPLE OUTPUT – EXCEEDING GUESS LIMIT

SECRET NUMBER = 487

Guess 1/15: 123
How many correct digits? 0

Guess 1/15: 123 Correct=0 Exact=0

Guess 2/15: 490
How many correct digits? 1
How many exact digits? 1

Guess 1/15: 123 Correct=0 Exact=0
Guess 2/15: 490 Correct=1 Exact=1

Guess 3/15: 491
How many correct digits? 1
How many exact digits? 1

Guess 1/15: 123 Correct=0 Exact=0
Guess 2/15: 490 Correct=1 Exact=1
Guess 3/15: 491 Correct=1 Exact=1
Guess 4/15: 431
How many correct digits? 1
How many exact digits? 1

.
.
.
.
.
.

Guess 15/15: 413

Game Over!! Exceeded max. attempts! Start a new game (y/n)?

MARKING CRITERIA

- Coding Styles – layout, comments, white spaces, naming convention, variables, indentation.
- Documentation – Design + Test Plan
- Program Correctness – logic, program structure, functions with appropriate parameters
- User Interaction – how informative and accurate information is exchanged between game player and host.
- Readability counts – programs that are well structured and easy-to-follow using functions to breakdown complex problems into smaller cleaner generalized functions are preferred over a function embracing a complex logic with nested conditions and sub-functions! In other words, a design with clean architecture with high readability is the predilection for the course objectives over efficiency.
- KISS approach – Keep It Simple and Straightforward.
- Balance approach – you are not required to come up a very optimized solution. However, take a balance between readability and efficiency with good use of program constructs.

CHALLENGES

Implement a guessing strategy for 4-digit secret number and YET keep program structure simple and easy to follow.

DUE DATE

March 16th, 2018