Peer Assessments (https://class.coursera.org/interactivepython-004/human_grading/)
/ Mini-project # 2 - "Guess the number" game"

Help (https://class.coursera.org/interactivepython-004/help/peergrading?
url=https%3A%2F%2Fclass.coursera.org%2Finteractivepython-
004%2Fhuman_grading%2Fview%2Fcourses%2F972072%2Fassessments%2F29%2Fsubmissions)

due in 5day 22h

Submission Phase

1. Do assignment ☐ (/interactivepython-004/human_grading/view/courses/972072/assessments/29/submissions)

Evaluation Phase

2. Evaluate peers 🔒 (/interactivepython-004/human_grading/view/courses/972072/assessments/29/peerGradingSets)

3. Self-evaluate 🔒 (/interactivepython-004/human_grading/view/courses/972072/assessments/29/selfGradingSets)

Results Phase

4. See results 🔒 (/interactivepython-004/human_grading/view/courses/972072/assessments/29/results/mine)

☐ In accordance with the Honor Code, I certify that my answers here are my own work, and that I have appropriately acknowledged all external sources (if any) that were used in this work.

Save draft   Submit for grading

A reminder about the Honor Code

For previous mini-projects, we have had instances of students submitting solutions that have been copied from the web. Remember, if you can find code on the web for one of the mini-projects, we can also find that code. Submitting copied code violates the Honor Code (https://class.coursera.org/interactivepython-003/wiki/view?page=honorcode) for this class as well as Coursera's Terms of Service. Please write your own code and refrain from copying. If, during peer evaluation, you suspect a submitted mini-project includes copied code, please evaluate as usual and email the assignment details to iipphonorcode@online.rice.edu (mailto:iipphonorcode@online.rice.edu). We will investigate and handle as appropriate.

Mini-project description — "Guess the number" game

One of the simplest two-player games is "Guess the number". The first player thinks of a secret number in some known range while the second player attempts to guess the number. After each guess, the first player answers either "Higher", "Lower" or "Correct!" depending on whether the secret number is higher, lower or equal to the guess. In this project, you will build a simple interactive program in Python where the computer will take the role of the first player while you play as the second player.

You will interact with your program using an input field and several buttons. For this project, we will ignore the canvas and print the computer's responses in the console. Building an initial version of your project that prints information in the console is a development strategy that you should use in later projects as well. Focusing on getting the logic of the program correct before trying to make it display the information in some "nice" way on the canvas usually saves lots of time since debugging logic errors in graphical output can be tricky.

Mini-project development process

We have provided a basic template for this mini-project **here** (http://www.codeskulptor.org/#examples-guess_the_number_template.py). Our suggested development strategy for the basic version of "Guess the number" is:

1. Decide on a set of global variables that contain the state of the game. For example, one obvious choice is the secret number that has been generated by the program. You will need other global variables, especially to accommodate later extensions to the basic game.

2. Figure out how to generate a random secret number in a given range, `low` to `high`. When discussing ranges, we will follow the standard Python convention of including the low end of the range and excluding the high end of the range, which can be expressed mathematically as `[low, high)`. So, `[0, 3)` means all of the numbers starting at 0 up to, but not including 3. In other words 0, 1, and 2. We suggest using the range `[0, 100)` in your first implementation. **Hint:** look at the functions in the `random` module to figure out how to easily select such a random number. We suggest testing this in a separate CodeSkulptor tab before adding code to your project.

3. Figure out how to create an input text box using the `simplegui` module. You will use this input to get the guess from the user. For all variants of the game, this input field should always be active (in other words, a game should always be in progress). Again, test in a separate CodeSkulptor tab before adding code to your project.

4. Write the event handler `input_guess(guess)` that takes the input `guess`, compares it to the secret number and prints out the appropriate response. Remember that `guess` is a string so you will need to convert it into a number before testing it against the secret number. **Hint:** We have showed you how to convert strings to numbers in the lectures.

5. Test your code by playing multiple games of "Guess the number" with a fixed range. At this point, you will need to re-run your program between each game (using the CodeSkulptor "Run" button).

6. Fill in your `new_game()` function so the generation of the secret number is now done inside this function. That is, calling `new_game()` should compute a random secret number and assign it to a global variable. You can now call the function `new_game()` in the body of your code right before you start your frame.

From this minimal working version of "Guess the number", the rest of this project consists of adding extra functionality to your project. There are two improvements that you will need to make to get full credit:

1. Using function(s) in the `simplegui` module, add buttons to restart the game so that you don't need to repeatedly click "Run" in CodeSkulptor to play multiple games. You should add two buttons: "Range: 0 - 100" and "Range: 0 - 1000" that allow the player to choose different ranges for the secret number. Using either of these buttons should restart the game and print out an appropriate message. They should work at any time during the game. In our implementation, the event handler for each button set the desired range for the secret number (as a global variable) and then call `new_game` to reset the secret number in the desired range.

As you play "Guess the number", you might notice that a good strategy is to maintain an interval that consists of the highest guess that is "Lower" than the secret number and the lowest guess that is "Higher" than the secret number. A good choice for the next guess is the number that is the average of these two numbers. The answer for this new guess then allows you to figure a new interval that contains the secret number and that is half as large. For example, if the secret number is in the range `[0, 100)`, it is a good idea to guess `50`. If the answer is "Higher", the secret number must be in the range `[51, 100)`. It is then a good idea to guess `75` and so on. This technique of successively narrowing the range corresponds to a well-known computer algorithm known as <u>binary search</u> <u>(http://en.wikipedia.org/wiki/Binary_search_algorithm)</u>.

2. Your final addition to "Guess the number" will be to restrict the player to a limited number of guesses. After each guess, your program should include in its output the number of remaining guesses. Once the player has used up those guesses, they lose, the game prints out an appropriate message, and a new game immediately starts.

   Since the strategy above for playing "Guess the number" approximately halves the range of possible secret numbers after each guess, any secret number in the range `[low, high)` can always be found in at most `n` guesses where `n` is the smallest integer such that `2 ** n >= high - low + 1`. For the range `[0, 100)`, `n` is seven. For the range `[0, 1000)`, `n` is ten. In our implementation, the function `new_game()` set the number of allowed guess to seven when the range is `[0, 100)` or to ten when the range is `[0, 1000)`. For more of a challenge, you may compute `n` from `low` and `high` using the functions `math.log` and `math.ceil` in the `math` module.

When your program starts, the game should immediately begin in range `[0, 100)`. When the game ends (because the player either wins or runs out of guesses), a new game with the same range as the last one should immediately begin by calling `new_game()`. Whenever the player clicks one of the range buttons, the current game should stop and a new game with the selected range should begin.

Grading rubric — 11 pts total (scaled to 100 pts)

Your peers will assess your mini-project according to the rubric given below. To guide you in determining whether your project satisfies each item in the rubric, please consult the video that demonstrates our implementation of "Guess the number". Small deviations from the textual output of our implementation are fine. You should avoid potentially confusing deviations (such as printing "Too high" or "Too low" instead of "Lower" and "Higher"). Whether moderate deviations satisfy an item of the grading rubric is at your peers' discretion during their assessment.

Here is a break down of the scoring:

- 1 pt — The game starts immediately when the "Run" button in CodeSkulptor is pressed.

- 1 pt — A game is always in progress. Finishing one game immediately starts another in the same range.

- 1 pt — The game reads `guess` from the input field and correctly prints it out.

- 3 pts — The game correctly plays "Guess the number" with the range `[0, 100)` and prints understandable output messages to the console. Play three complete games: 1 pt for each correct game.

- 2 pts — The game includes two buttons that allow the user to select the range `[0, 100)` or the range `[0, 1000)` for the secret number. These buttons correctly change the range and print an appropriate message. (1 pt per button.)

- 2 pts — The game restricts the player to a finite number of guesses and correctly terminates the game when these guesses are exhausted. Award 1 pt if the number of remaining guesses is printed, but the game does not terminate correctly.

- 1 pt — The game varies the number of allowed guesses based on the range of the secret number — seven guesses for range `[0, 100)`, ten guesses for range `[0, 1000)`.

To help aid you in gauging what a full credit project might look like, the video lecture on the "Guess the number" project includes a demonstration of our implementation of this project. You do not need to validate that the input number is in the correct range. (For this game, that responsibility should fall on the player.)

In the submission phase, cut and paste the URL for your cloud-saved mini-project into the box below. Click the Honor Code box and hit the "Submit for grading" button when you are ready to submit your mini-project. (You may submit as many times as you like before the deadline so we suggest that you use "Submit" instead of "Save".) **IMPORTANT: Please use the "Review your work" link that appears at the top of the subsequent submission page to verify that you submitted a working link for the final version of your mini-project.**

☐ In accordance with the Honor Code, I certify that my answers here are my own work, and that I have appropriately acknowledged all external sources (if any) that were used in this work.

Save draft   Submit for grading