Submission Phase

1. Do assignment ✓ (/interactivepython1-010/human_grading/view/courses/976300/assessments/29/submissions)

Evaluation Phase

2. Evaluate peers ✓ (/interactivepython1-010/human_grading/view/courses/976300/assessments/29/peerGradingSets)
3. Self-evaluate ✓ (/interactivepython1-010/human_grading/view/courses/976300/assessments/29/selfGradingSets)

Results Phase

4. See results ✓ (/interactivepython1-010/human_grading/view/courses/976300/assessments/29/results/mine)

Your effective grade is 9

Your unadjusted grade is 9, which was calculated based on a combination of the grade you received from your peers and the grade you gave yourself.

See below for details.

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 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.

When discussing ranges in this mini-project, we will follow the standard Python convention of including the low end of the range and excluding the high end of the range. Mathematically, we will express ranges via the notation `[low, high)`. The square bracket of the left of the range indicates that the corresponding bound should be included. The left parenthesis on the right of the range indicates that corresponding bound should be excluded. For example, the range `[0, 3)` consists of numbers starting at 0 up to, but not including 3. In other words 0, 1, and 2.

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 below. Remember to run your program after each step to ensure that you implemented that step correctly.

1. Add code to the program template that creates a frame with an input field whose handler has the name `input_guess`. You will use this input field to enter guesses.
2. Add code to the event handler `input_guess(guess)` that takes the input string `guess`, converts it to an integer, and prints out a message of the form `"Guess was 37"` (or whatever the guess actually was). Hint: We have shown you how to convert strings to numbers in the lectures.
3. Add code to the function `new_game` that initializes a global variable `secret_number` to be a random number in the range `[0, 100)`. (Note that this range should not include 100 as a possible secret number.) Remember to include a `global` statement. Hint: Look at the functions in the `random` module to figure out how to easily select such a random number. Observe that the call to `new_game()` at the bottom of the template ensures that `secret_number` is always initialized when the program starts running.
4. Add code to `input_guess` that compares the entered number to `secret_number` and prints out an appropriate message such as `"Higher"`, `"Lower"`, or `"Correct"`.
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). You are also welcome to use this testing template http://www.codeskulptor.org/#examples-gtn_testing_template.py (http://www.codeskulptor.org/#examples-gtn_testing_template.py). The bottom of the template contains a sequence of calls to `input_guess()` for three games of "Guess the number". Uncomment each sequence of calls and check whether the output in the console matches that provided in the comments below. Note that your output doesn't have to be identical, just of a similar form.

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 with the labels `"Range is [0,100)"` and `"Range is [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 binary search (http://en.wikipedia.org/wiki/Binary_search_algorithm).

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.

For more helpful tips on implementing this mini-project, please visit the Code Clinic tips (https://class.coursera.org/interactivepython1-010/wiki/gtn_tips) page for this mini-project.

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.

http://www.codeskulptor.org/#user41_JBFDXiAS3U_5.py
(http://www.codeskulptor.org/#user41_JBFDXiAS3U_5.py)

Evaluation/feedback on the above work

Note: this section can only be filled out during the evaluation phase.

Grading Instructions

In this assessment, you will be asked to grade five of your peer's mini-projects. In return, five of your peers will grade your project. Since you've worked hard on your mini-project and would like your peers to do a good job of assessing your project, please take your time and do a good job of assessing your peer's mini-projects in return. Finally, you will be asked to self-assess your own project after assessing your peer's mini-projects. Before beginning your assessment, please review the video lecture on "Guess the number" to see a demonstration of a correct, full-credit implementation of the mini-project.

For each item below, run your peer's program and observe its behavior. If the program demonstrates the correct associated functionality, give it full credit for that item. If the program demonstrates partial, inconsistent, or incorrect functionality, assign the program partial credit based on your testing and our comments. When assigning partial or no credit, please add a short written comment that describes the issues you observed. While this takes extra effort, please remember how frustrating it would be to receive partial or no credit on part of your mini-project with no accompanying explanation.

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

Score from your peers: 1

Score from yourself: 1

Optional comments.

peer 1 → frame.start() initializing is missing no canvas black background present

peer 2 → *[This area was left blank by the evaluator.]*

peer 3 → *[This area was left blank by the evaluator.]*

peer 4 → *[This area was left blank by the evaluator.]*

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

Score from your peers: 1

Score from yourself: 1

Optional comments.

peer 1 → *[This area was left blank by the evaluator.]*

peer 2 → *[This area was left blank by the evaluator.]*

peer 3 → *[This area was left blank by the evaluator.]*

peer 4 → *[This area was left blank by the evaluator.]*

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

Score from your peers: 1

Score from yourself: 1

Optional comments.

peer 1 → *[This area was left blank by the evaluator.]*

peer 2 → *[This area was left blank by the evaluator.]*

peer 3 → *[This area was left blank by the evaluator.]*

peer 4 → *[This area was left blank by the evaluator.]*

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

Score from your peers: 1

Score from yourself: 3

Optional comments.

peer 1 → The games do not work properly once count starts and 7 guesses it does not tell you the secret number. Games on show Higher or Lower not both.

peer 2 → Higher and Lower messages are backwards; otherwise, the basics of the game are correct. The indicator "Higher!" means the player should guess a higher number than they just did.

peer 3 → game prints wrong higher/lower hints for player, thus it is not possible to win this game without printing secret_number

peer 4 → *[This area was left blank by the evaluator.]*

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.)

Score from your peers: 2

Score from yourself: 2

Optional comments.

peer 1 → *[This area was left blank by the evaluator.]*

peer 2 → *[This area was left blank by the evaluator.]*

peer 3 → *[This area was left blank by the evaluator.]*

peer 4 → *[This area was left blank by the evaluator.]*

2 pts — The game prints out the number of guesses remaining 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.

Score from your peers: 2

Score from yourself: 2

Optional comments.

peer 1 → *[This area was left blank by the evaluator.]*

peer 2 → *[This area was left blank by the evaluator.]*

peer 3 → *[This area was left blank by the evaluator.]*

peer 4 → *[This area was left blank by the evaluator.]*

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

Score from your peers: 1

Score from yourself: 1

Optional comments.

peer 1 → Entered guess = 0 and was told it was lower indicates program is not working properly

peer 2 → *[This area was left blank by the evaluator.]*

peer 3 → *[This area was left blank by the evaluator.]*

peer 4 → *[This area was left blank by the evaluator.]*

Overall evaluation/feedback

Note: this section can only be filled out during the evaluation phase.

Please provide feedback to your classmate for the mini-project you're grading. In particular, focus on areas where you did not assign the mini-project full credit. Remember that a sentence or two explaining your rationale will be appreciated.

peer 1 → Good effort still need some work to get guess function working properly

peer 2 → Good job

peer 3 → Good job overall. You just mixed printings for lower/higher stuff

peer 4 → Good job