**Project 1: Number Guessing Game**

It’s time to put everything you’ve learned so far to the test! You have seen the example of the completed project, now it’s your turn to make it. As always, there is a problem we’re trying to solve. Here it is!

**The Problem**

Your goal is to create a game. Specifically, it is a game where there is a random number (that the computer will generate) from 1 to 100 and players can keep guessing until they get it right. After every guess, the program will let the player know if their guesses are too high or too low. At the end, the program will tell them how many guesses they took to get it right.

Here’s an example of the final code working correctly:

|  |
| --- |
|  |

Hopefully, just by seeing the problem, you’re already starting to come up with a way to create a program for it. In case you’re not, we’re going to come up with a general solution to this problem without writing any code. Once we’re sure that it will work, we will then turn that into code. To do this, we will split this problem into 4 parts: Initial Framework, Pseudocode, Code Translation, and Debugging.

For the initial framework, we will make a very general outline of how a computer would go about solving this problem. We will be asking basic questions about what goes into solving the problem, such as if loops or if statements will be needed, how many of each, and things like that.

For the Pseudocode, we will be creating a set of instructions that anyone can translate into code, using no specific programming language, but words that any programmer could use to make the program. This part is more about getting the general idea down about the step-by-step process that goes into solving the problem and less about using language-specific syntax (specifically Python).

For the Code Translation part, we will then worry about translating the pseudocode we made into Python and that there are no syntax errors.

For the Debugging part, we will run the code thoroughly to make sure that there are no errors at all, and we will be considering all the different ways the program could break and whether or not it is worth it to implement a fix for those bugs.

**The Schedule and Grading**

Work on this project will be split up into two parts. We will be working on parts 1 and 2 during the first week and will be working on the last 2 parts the week after. This means that your homework will be to get the first two parts done by the second week and then after that, the homework will be to finish the project.

|  |  |  |
| --- | --- | --- |
| **Date** | **Project Parts** | **Homework** |
| June 6 | Parts 1 and 2 | Finish parts 1 and 2 |
| June 13 | Parts 3 and 4 | Finish parts 3 and 4 |

Part 1 will be a series of questions that you need to answer. All in all, each part will be worth 25% of the total project grade. Part 1 will be the only part with questions to answer. Part 2 will just require you to submit the pseudocode, which will be graded on how well it works. Part 3 will have you submitting the code. I will match your python code to your pseudocode to see if it matches your submitted code and that it has no Compilation Errors. For Part 4, you will be graded on how well your code runs and also on its style/variable names.

**Part 1: Initial Framework**

As I have stated before, this part is all about getting the general concept down about how to do this problem. Therefore, you should reread the problem and then answer the following questions (you will be able to answer these questions in the Google Drive document called “Project 1 Answers” in your homework folder):

1. Is there any part of the solution that will need to be run multiple times? In other words, will we be needing a loop anywhere in the code?
2. Hopefully you said yes to the previous question. In that case, will we be needing a for loop or a while loop? For loops are used when we want to run the loop a specific number of times and while loops are used when we don’t know precisely how many times to run the loop. In this problem, we have to depend on the user of the program to know when to end the loop (the program stops when they answer the right number). Will we know exactly when they will enter the correct number?
3. What will we need to do every time the loop runs? In other words, at every step in the loop, what do we need to do? Your answer should include asking the user for their guess and what to do with their guess.
4. Under what condition(s) should the loop stop?
5. Is there any part of the solution that will need to do different things depending on different situations? In other words, do we need any if statements?
6. Hopefully you said yes to the previous question as well. What are those different cases? What do you need to do in each of those cases? The answer should involve having the program reacting to the player’s guess.
7. What should your program keep track of? There should be 3 things. These are what you’ll be making variables to store, and they should be things you want to use or change every time the loop runs. Come up with appropriate variable names for each of these 3 things.
8. So, you probably know now that there is one big loop with if statements inside it. However, there should be code that comes before the loop. Generally, what should it be? There should be 2 things to do before the loop. (Hint: one of the things has to do with what we typically do with most variables at the top of the functions we have written)
9. After the loop is done, do we need to do anything else? If so, what are the things we need to do?
10. Besides generating a random number, is there any built-in function you will need to use? If so, what are they? Refer to lesson 5 if you don’t remember all the built-in functions that we have talked about. The solution I made had 3 different ones, one of which was used 3 times. Don’t feel like you have to use exactly 3 built-in functions, as you might be able to find a solution that uses less or more than 3.
11. Finally, write down a very general step-by-step process of the different parts of the code that you’re going to eventually write. At the end, provided there aren’t too many changes, these will be what you put in the comments when you stylize your code and label each parts. Try to be more general than specific. For example, don’t say “create and set variable1 to 0. Create and set variable2 to 0.” Instead, say “initialize variable1 and variable2” together in the same line. For another example, don’t say “If the guess was smaller than the actual number, print ‘some random string’”, instead say “tell user about their guess being too small” or something like that.

**Part 2: Pseudocode**

Now that you’ve come up with what’s known as *pseudocode*. This is a process many programmers go through when realizing a program. Pseudocode is basically writing down the code without worrying about any language-specific syntax (you don’t have to worry about if the pseudocode you’re writing will make sense to a computer). Instead, you want to write down generally what each line of code will do. To do this, you may write lines that look like code, but aren’t actually. There is no specific format to this, but any programmer should be able to read the pseudocode and translate it into a program using whatever language they want, so while you don’t have to be very particular when it comes to syntax, you should be specific when it comes to what each line *represents*.

Here’s an example of pseudocode for how to calculate a number’s *factorial*. This is the product of (a.k.a. the result of multiplying) all the integers from 1 to that number. In addition, 0! (0 factorial, the factorial symbol is an exclamation mark) is 1.

|  |
| --- |
| 1 function factorial(integer n):  2 initialize variable total to 1  3 for integer variable i from 1 to n+1:  4 set variable total to total\*i  5 return total |

You can see that this is not code. However, each line in the pseudocode can be translated to Python code, or if you knew other languages (like C++ or Java), they could be translated into those languages. In addition, we can follow along with the pseudocode to see if it will work. That way, we don’t have to worry about syntax and still know that our solution will work. How do we do that without running the code? We do it by hand. It may seem tedious, but it really helps iron out any problems with the code.

For example, we could pretend like we are running factorial(3). Here’s the step-by-step process that the computer will do to run factorial(3):

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Step Number** | **Line Number** | **Description of What Happens** | **Variable total** | **Variable i** |
| 1 | 1 | The function is called with n = 3. | None | None |
| 2 | 2 | Make variable total and set it to 1. | 1 | None |
| 3 | 3 | Start a for loop. In this loop, a variable i is created and represents how many times the loop has iterated (a.k.a. looped). It starts at 1 and stops when it is equal to n+1 (which is 4). | 1 | 1 |
| 4 | 4 | Set the variable total equal to itself times i, or 1\*1. | 1 | 1 |
| 5 | 3 | Increase i by 1. It is not yet equal to 4, so continue the loop. | 1 | 2 |
| 6 | 4 | Set the variable total equal to itself times i, or 1\*2. | 2 | 2 |
| 7 | 3 | Increase i by 1. It is not yet equal to 4, so continue the loop. | 2 | 3 |
| 8 | 4 | Set the variable total equal to itself times i, or 2\*3. | 6 | 3 |
| 9 | 3 | Increase i by 1. Now it is equal to 4, so stop the loop. | 6 | 4 |
| 10 | 5 | Return the total. | 6 | 4 |

Since 3! is 1\*2\*3, or 6, and 6 was returned, we can see that the code works the way it’s supposed to. Now it’s your turn to turn your answer from question 11 in part 1 into pseudocode. When your code needs to generate a random number, just write “generate random integer from 1 to 100”. Be sure to still designate strings with quotations. If your code needs to use a built-in function, don’t be afraid to write that down too. For example, if you need to use the type() function (you shouldn’t need to, but this is an example) on a variable named var1, write “call function type(var1)”.

Since this is pseudocode, there aren’t really rules to this other than this one: each line of your code should be able to be written in one line of code in Python (or whatever language you plan on writing in, provided you were going to write it in something other than Python). I can’t stress this enough, but pseudocode is not about whether if you’re using correct syntax, but if you know how the program works. Thus, it doesn’t really matter if you write “initialize variable var1 to 0” or “var1 = 0”, despite the latter option being valid Python code and the former just being English. Both are equally valid in pseudocode.

Go do it now (in the “Project 1 Answers” Google Drive document). Don’t worry about keeping track of line numbers like I did in the example. Once you’re done, get a pencil and paper and test running the code multiple times. For the time being, choose a random number when you get to the part where the program is supposed to generate a random number. When asking the player for a guess, pretend like you don’t know what the answer is and choose another random (but logical) number and play from there. Keep track of each variable and what line of code you’re on. One tip is to type a symbol you don’t use anywhere else in the pseudocode (i.e., &, $, or @) before the line you’re on to keep track of it. Once you’re done with the line, delete the symbol and put it on the next line. Use your paper for keeping track of variables. Every time a variable gets updated, erase whatever value is there and replace it with whatever the new value is. I’m not going to grade this part or even check to make sure you did this, so you technically don’t have to do this part. HOWEVER, I will be checking to make sure your pseudocode works, and this is a good way to make sure that it does.

Aside from this project, if you find that pseudocode helps (which it does) and you want to use it for future problems in this class, definitely feel free to do that! Aside from projects, I probably won’t force you to use it, so make sure that you know that it is an option for checking that your code works *before* putting it into IDLE.

**Part 3: Translating the Pseudocode**

There’s not really much to say for this part. Make a new IDLE file and put your python-ified pseudocode in there. Run the code a couple of times to make sure that it works. When you’re done, submit it to your homework folder on Google Drive. Call the file “P1.3\_<name>.py”, replacing <name> with your first name.

To generate a random number from 1 to 100, you first need to put the following line of code at the very top of your program:

from random import randint

Then, when you need to generate the random number, you only need to put randint(1,100). You can even save that to a variable by doing:

variable\_name = randint(1,100)

Just make sure to replace variable\_name with the name of the variable you want to create/use.

**Part 4: Debugging and Style**

This is probably the most tedious part of any code: the debugging. Make a copy of the file you used in part 3 and name it “P1.4\_<name>.py”, replacing <name> with your first name.

To start off with, can you think of any way the program might fail to run? The biggest way would involve the fact that you’re calling the input function (what happens if the user guesses something that isn’t a number?). Can you think of other ways the program might fail to run or might run in an unintended way? Are any of these problems worth implementing a fix for? There is no right answer to this. If you decide that fixing the bug needs to be fixed, then do so. If not, make sure you have a good reason as to why you deliberately chose to not fix the bug (such as: “The program asks the user to enter a *number*. If they enter something that is not a number, that is their fault that the program crashes, not the fault of the program or the programmer.”).

If you decide that non-integers should be allowed to be entered, and the program should tell them not to do that, here’s the code for a function that takes in 1 variable and returns True if the variable can be converted into an integer:

def isInteger(o):

try:

int(o)

return True

catch:

return False

You’ll need to use that function to tell if what the user has guessed is an integer or not. Just run the isInteger() function with the guess as the input (i.e., isInteger(guess) if the variable you stored the guess in is called guess). So if you’re implementing this fix, copy and paste that function definition at the top of your program (below the from random import randint line). If you decide to do this and get it to work, you’ll get 5 extra percent of extra credit for this project.

If you wrote your code correctly, it should still work fine if a player were to guess a number below 1 or above 100. Should your program still tell them that their guess was too low (if they guessed a number below 1) or that it was too high (if they guessed a number above 100), or should it tell them that they guessed a number out of the bounds of the game? Regardless of whether you think it’s necessary or not, I want you to change your code to handle this case. If a user guesses a number below 1 or above 100, tell them that their answer is out-of-bounds and remind them of what the bounds are (1 and 100).

Next is what’s known as *bounds-checking*. Because programmers often use both < and <= as well as > and >=, it is quite often that they mix them up. Thus, it can lead to what’s known as *one-off* errors where the program reacts in a way that it shouldn’t when inputting numbers that are one off from the boundary. Run the code and check that guessing 0 and 101 are correctly identified as out-of-bounds guesses and that 1 and 100 are correctly identified as valid guesses. In addition, temporarily change your code so that instead of generating a random number, it always chooses 1 (replace randint(1,100) with 1). Then run the code to make sure that there are no errors. Then do the same with 100 instead of 1. Once both work, you can change it back to generating a random number.

Now that your debugging should be (hopefully) over with, now it’s time to add some style to your code. Here’s a checklist of things to do:

* Put empty lines in between each part of your code and comments above each part explaining what each part does.
* Are there any specific lines of code that may be hard to understand from a quick glance? If so, put a comment after it on the same line explaining what it does. If the comment would make the line go on too long (past col 80 or so), put and empty line above the line and put the comment there.
* Are there any lines of code that are too long? If so, consider using variables to split up a line of code, doing part of a calculation on one line and saving the result to a variable and using that variable in the next line to continue the calculation. If you don’t want to do that, you could also use the \ character as explained in lesson 10 to split up lines of code.
* Last thing to do, add a couple of empty lines at the very top of the program (below the from random import randint line) and put one big comment there that explains what the entire program does. Also be sure to mention if there are any invalid inputs and generally what the program does.
* Below the comment described in the bullet point above, put another empty line and another comment with your name and another line/comment with today’s date.

Now that everything is completely done, make sure that the file is named “P1.4\_<name>”, replacing <name> with your first name. Submit this file to your Homework folder in the Google Drive.