Working An Example

by Sophia

WHAT'S COVERED

In this lesson, you will learn about working through an example. Specifically, this lesson covers:

1. Starting With Input/Output

In the prior lesson, we explored your idea for a project. Breaking down the individual overall steps can be challenging. What can be helpful here is to first determine what the output should be. To help with that process, let's look at our initial example and build from there. As we start to work through this example, try to think about how you'd start to break down your program. It can help to start back in Unit 1's Programming Mindset lesson and review how we looked at the input, processing, and output. Then, as part of the Thinking Through Examples lesson, we looked at real-world examples like making orange juice. You'll remember how we needed to break it down into specific steps.

THINK ABOUT IT

The main logic of the program is what we're aiming toward here. This part does not require you to break down the problem into specific programming steps. This part determines what steps we should include and how we should order them. As a developer, you won't worry about the syntax of the language to be used during this part. The goal is to be able to use any programming language to create the output that we want to create. The input aspects should be straightforward since you want to simply determine how the input is coming in. This could be user inputs from the screen/console or actual data from files.

In our demonstration program, the only input from the user would be the number of times a game is played, which simplifies this, but there may be other items that you need to consider such as a menu and reading from a file, along with many other items.

BRAINSTORM

Let's start with the idea and the questions and answers that we had in the prior lesson to use as an initial guide of what the output should be. Let's focus purely on the questions that are related to the output.

- If they played multiple games, what information would they need to track?
  - Ideally, I would like to be able to enter a number of games to play. In part, that would be based on my budget for the day. In knowing the number of games to play, I would like to know how many rolls on average it took to win and how many it took to lose. In addition, I'd like to know what my winning percentage would be.
- What information would they like to store?
  - On a daily basis for each set of games, I'd like the statistics about the games I played stored in a file. The file can just keep track of each time.
- What kind of output would they want?
  - I would like to see on screen the number of times the game was played, how many times out of those games I won, how many times I lost, the average number of rolls per win, the average number of rolls per loss, and the winning percentage.
- Where would they like the information tracked?
  - In this case, I would like the information being tracked to be placed in a file for every time I played the game.

There are some key criteria that we can pull from here. We know that the output to the file should consist of the following:

- Date and time
- Total number of wins
- Total number of losses
- Average number of rolls per win
- Average number of rolls per loss
- Winning percentage

Now, what our friend/user wasn't specific about was the formatting of the output. We could follow up on this with our user to identify if there's a specific format or if they just wanted the information presented as-is. Some may want it in a specific table format while others won't care how it is formatted. This could be a sample output of what it could look like:

EXAMPLE

date and time = 25/03/2022 01:34:03
The total number of wins is 2
The total number of losses is 3
The average number of rolls per win is 1.5

The average number of rolls per loss is 5.0
The winning percentage is 0.4

Once we have an idea of what it could look like, we can use that to build a foundation for what the output may look like and work backward towards some of the variables that we would need to store the output:

EXAMPLE
Output the date and time
Output the total number of wins
Output the total number of losses
Output the average number of rolls per win
Output the average number of rolls per loss
Output the winning percentage

THINK ABOUT IT

When it comes to your program, think about what the output should look like. Would the output be done throughout the program or would it be done all at the end? Would the output be done to the screen or to a file or perhaps both?

2. Moving to Processing

The next part we have to think about is the processing within the program. There's a lot in a program that you may have to think about for this part but let's focus on the core of the processing. This is a crucial element and depending on your program, you may have a core part of the processing but also potentially have more supplemental parts as well. Ideally, you want to focus on one part at a time. The logic should be nailed down to what the core of the program is. Let's go back to our example program, pull the key questions, and build off of that.

- How would the game of casino craps be played?
  - With the game of craps, the game is made with players putting bets on the first roll of the sum of two dice, also known as the come-out roll. In this round, the players must decide whether the dice will land on 7 or 11 (pass bet) or 2, 3, or 12 (don't pass bet). The round ends if the total dice value is 7 or 11 (known as a natural), or 2, 3, or 12 (craps). If the total of the dice value rolled is a 4, 5, 6, 8, 9, or 10, then that number becomes the point and starts the next stage of the game. The shooter then rolls the dice again and repeats until the shooter rolls a 7 or the point number, which ends the craps game. If a 7 is rolled, they "sevens out" and lose. If they rolled the initial point score, they win the round.
- What would the rules be for the game?
  - Player rolls 2 dice the first time.
    - If 2, 3, or 12 is rolled on the first time, the player loses.

    - If 7 or 11 is rolled on the first time, the player wins.
    - If any other number is rolled, that number becomes the point value or initial sum.
- If the player has not won or lost, roll again.
    - If the sum of the roll of the two dice is equal to 7, the player loses.
    - If the sum of the roll of the two dice is equal to the initial sum, the player wins.
    - If the sum of the two dice is anything else, reroll again.

Interestingly, the way that the second question was answered helps us determine some of the functionality already. One of the best ways to start to break down the processing for this game is by taking the responses and determining a complete run of a program step by step. First, let's go through one example of a game in which the game is determined on the first roll, in the situation where the player wins or loses on the first roll:

↪ EXAMPLE

Player loses a game on the first roll.

1. Roll two six-sided random dice for the first time, getting a 1 and a 2.
2. Add the values on the top of the two dice, getting 3.
3. The player loses the game.

And:

↪ EXAMPLE

Player wins a game on the first roll.

1. Roll two six-sided random dice for the first time, getting a 3 and a 4.
2. Add the values on the top of the two dice, getting 7.
3. The player wins the game.

These can cover the first instance of the game but potential challenges can occur when the player has not rolled a 2, 3, 7, 11, or 12 that allows them to win or lose on the first roll. Let's go through an example of a full game in which the player isn't determined to win or lose the game on the first roll:

↪ EXAMPLE

Player loses on a few rolls of the dice.

1. Roll two six-sided random dice for the first time, getting a 1 and a 4.
2. Add the values on the top of the two dice, getting 5.
3. The value is not 2, 3, 7, 11, or 12, so the game continues.
4. Roll the two dice again, getting a 3 and a 6.
5. Add the values on the top of the two dice to get 9.
6. That value is not equal to either 7 or 5, so the game continues.
7. Roll the two dice again, getting a 4 and a 3.
8. Add the values on the top of the two dice to get 7.
9. The player loses the game.

Now we have a full set of steps where the dice rolling had to occur three total times. You should see from this example that there's some consistency that we can start to bring together now. You'll notice that each of these steps is clear and has a specific order. As you work through your own program, you'll want to define each item step by step as they'll help you work through those repeating patterns. If you only submitted the first example with a winner after the first roll, you'd miss out on the other examples to expand the game in a realistic fashion.

3. Adding Our Second Journal Entry

THINK ABOUT IT

At this point, we are ready to add our second journal entry for the Touchstone. After breaking a few examples/scenarios down step by step (sample runs of the craps game), we started to see some patterns that we can apply in our next lesson. We want to include steps for a few outcomes to ensure we identify all possible outcomes.

Again, we will start off with a not-so-good example of a journal entry first.

Bad Example of Journal Entry for Part 2

EXAMPLE

1. Roll two six-sided random dice for the first time, getting a 3 and a 4.
2. Add the values on the top of the two dice, getting 7.
3. The player wins the game.

This is a correct step-by-step example; however, it does not go through more outcomes that are possible.
Good Example of Journal Entry for Part 2

EXAMPLE
Here are some step by steps for a few possible examples.

Scenario 1: Player loses on the first roll:

1. Roll two six-sided random dice for the first time, getting a 1 and a 2.
2. Add the values on the top of the two dice, getting 3.
3. The player loses the game.

Scenario 2: Player wins on the first roll:

1. Roll two six-sided random dice for the first time, getting a 3 and a 4.
2. Add the values on the top of the two dice, getting 7.
3. The player wins the game.

Scenario 3: Player loses on a few rolls of the dice:

1. Roll two six-sided random dice for the first time, getting a 1 and a 4.
2. Add the values on the top of the two dice, getting 5.
3. The value is not 2, 3, 7, 11, or 12, so the game continues.
4. Roll the two dice again, getting a 3 and a 6.
5. Add the values on the top of the two dice to get 9.
6. That value is not equal to either 7 or 5, so the game continues.
7. Roll the two dice again, getting a 4 and a 3.
8. Add the values on the top of the two dice to get 7.
9. The player loses the game.

If we preview the Example Python Journal Submission document, we see this was added as the entry to Part 2. As you can see, the good example does include all distinct examples of possible outcomes. The more you can break out all possible outcomes, the easier it is to identify any patterns that you can carry over to Part 3 of the Python Journal, which is creating pseudocode for patterns and algorithms.

---

4. Guided Brainstorming

Given the coding project that you are trying to solve, it's a good idea to try and think about how the ordering would work. If you remember back to our Drink Order program at the end of Unit 1, we started to break down more of the specifics of the program by looking at the menu structure of what was possible in a selection for drinks.

↷ EXAMPLE

Water

- Hot
- Cold
  - Ice/No Ice

Coffee

- Decaffeinated or Not?
- Milk or Cream or None
- Sugar or None

Tea

- Green
- Black

We can use this as part of defining how the program works to help with the step-by-step breakdown that we are looking for in this lesson. In looking at these options, the user should be first prompted with the choice of water, coffee, and tea. Based on each of those selections, there are additional prompts that are unique to them. This

will show how the program is broken down with the individual steps that would be included. In this case, we have three main key items as the first input.

EXAMPLE

1. User selects water
2. User selects hot water
3. Output water, hot

This is great to start but it doesn't help us see some of the other possibilities, like what happens with the user selecting cold water, as there's an extra input.

EXAMPLE

1. User selects water
2. User selects cold water
3. User selects ice
4. Output water, cold, ice

It's better that we have both of those. At this point, although this seems like it would cover most of the task already, this isn't the case as you want to ensure that you've gone through each of the steps of the possibilities in as much detail as possible.

EXAMPLE

1. User selects coffee
2. User selects decaffeinated
3. User selects milk
4. User selects sugar
5. Output coffee, decaffeinated, milk, sugar

That covers another set for the coffee. There weren't any scenarios where there was a separate option like we had with the cold water. We have one more option to go:

EXAMPLE

1. User selects tea
2. User selects green tea
3. Output tea, green

Now this covers all of the possibilities and would be a great journal entry with all four of those examples. Addressing them individually would not be enough since this wouldn't cover the key differences between each option.

TRY IT

Directions: Now would be a good time for you to create these different examples/scenarios and think about each step in a specific order for your program. Think about each question and step one at a time to ensure that you've covered each key example. Ask yourself if you've covered the main types of examples or scenarios that

would be needed. Review the example of a good entry for Part 2 in the Example Python Journal Submission document and add your journal entry for Part 2 to your Python Journal.

SUMMARY

In this lesson, we learned that breaking down the individual overall steps to a future program can be challenging. What can be helpful is to first determine what the **input and output** should be. Once that is identified, we can **move to the expected processing** of the program and look at some possible examples that we can break down and detail into steps. This can help us identify patterns early in the design process. We had an opportunity to see some specific steps for a few examples of the demonstration program and **added that as our second journal entry**. Finally, we used an old program from Unit 1 in the **Guided Brainstorming** section to reemphasize the need to break down the steps of a program to include all aspects of expected outcomes.

Best of luck in your learning!