Thinking Through Examples

by Sophia

WHAT'S COVERED

In this lesson, you will review various problems and determine how to break them down into discrete steps. Specifically, this lesson covers:

1. First Steps

BEFORE YOU START

Before we get started with any new problem that we want to design a program for, we need to understand what the problem is. For example, we need to determine what the goal or purpose of the program is. Try to think about what the program is supposed to do and what users should be able to expect when they run a particular program. For example, if we run a program to convert temperature from Celsius to Fahrenheit, we should expect that if we pass in a temperature value of 0 (which is 0 Celsius), we should get a result of 32 Fahrenheit.

Remember this formula from your high school days?
Celsius to Fahrenheit: °F = 9/5 * (°C) + 32

Once we define the goal and purpose of the program, we need to be able to then start to break down the problem into the smallest possible individual steps that are needed to perform the necessary tasks. We should not leave out any instructions or add in extra instructions that are not needed for the program.

For us to have a program work correctly, we have to ensure that the logic and syntax are correct. We'll get into the specific syntax of Python later in the course, but remember that syntax is the specific grammar rules for the language. For now, we will focus on planning the logic of a program. For us to write a program that works properly, the proper logic with the correct program instructions must be in the right order. If certain steps are not performed in the right order, we'll run into a lot of potential issues.

EXAMPLE

For example, consider the steps it takes for you to wash your face. Would it make sense to dry your face prior to putting water on your face? Or would it make sense to put on soap after you're done drying your hands? Thinking about the steps in a program is done in a similar manner. We need to have the specific steps defined as a starting point.

Once we've defined those steps in the right order, we can start to think about what items could be stored as variables. A **variable** is a term that you'll hear about quite often in programming. It is defined as a name that is used for a location in memory where the value that's stored in that location can hold different values at various points throughout the program. Using our example of the temperature conversion, we may have the temperature as a variable, as we would want to get the input from the user to make the conversion in the code. In a larger program, there could be many other variables that could be used within the program. Any potential user input or a result of a calculation could be a variable that could be defined and used.

Once we have those variables defined, we'll need to then start to look at the logic beyond the sequence of steps. There may be scenarios where we have conditions or branching of decisions based on user input or other variables. For example, if you ask the user if they are hungry, there could be one path if they are hungry and another path of statements if they are not. We could also have other elements to think about such as certain statements being possible to run more than once. In these cases, we may make use of repetition or break down those statements into sub-parts of a program.

TERM TO KNOW

Variable

A named memory location that has the ability to hold various distinct values at different points in time in the program.

2. Real-World Example

Analyzing problems can potentially seem overwhelming when you think about all of the steps involved. So, let's break things down into smaller steps to start demonstrating what's involved! There are many different examples that exist around us that can make use of the process of breaking down problems and understanding how the steps are intertwined.

EXAMPLE

Let's have someone make a glass of orange juice with these steps:

1. Get oranges.
2. Get knife.
3. Get cutting board.
4. Get juicer.
5. Turn on juicer.
6. Add bread.

7. Place halved oranges in juicer.
8. Turn off juicer.
9. Get glass.
10. Place glass under juicer.
11. Cut oranges in halves on cutting board.

First of all, it looks like we added bread into the orange juice; that wouldn't work so well. Other steps are out of sequence, some are missing, and some are obviously for other procedures outside of making orange juice. The logical order of the statements is very important, as a computer program can only execute the steps that you tell it to in that specific order. We would probably need to change our steps to:

1. Get cutting board.
2. Get juicer.
3. Get glass.
4. Get oranges.
5. Get knife.
6. Cut oranges in halves on cutting board.
7. Place glass under juicer.
8. Turn on juicer.
9. Place halved oranges in juicer.
10. Turn off juicer.

THINK ABOUT IT

There are some steps where the order doesn't matter and we could swap them around, such as the first five steps in which we are getting items to complete the task. Think of some of your daily tasks that you do and how you would break down the steps so that a robot could follow them exactly as is.

3. Is It Raining or Snowing?

Let's take a look at a bit more complex scenario where we'll have to incorporate more areas of consideration. Say we have Sophie, who walks to work from Monday to Friday. She checks the weather forecast. If it's raining during the day, she'll bring a rain jacket. If it's snowing, she'll bring her winter jacket. On the weekends, she needs to bring the car keys, as she drives to visit her family on Saturday and visit her friends on Sunday.

To start, we'll need to break things down into various parts of the program.

↪ EXAMPLE
First, we need to make note that the selection (as a start) is based on the day of the week. If it's Saturday or Sunday, Sophie just needs the car keys:

- dayOfWeek = check the day of the week
- if dayOfWeek is equal to Saturday or Sunday then
    - itemToBring = "car keys"
- else if dayOfWeek is equal to Monday, Tuesday, Wednesday, Thursday, or Friday then
    - weatherForecast = check weather forecast for the day
        - if weatherForecast is equal to rain
            - itemToBring = "rain jacket"
        - if weatherForecast is equal to snow
            - itemToBring = "winter jacket"

We're incorporating a few new features with the use of conditions here, which is a big part of programming. We'll get into that level of detail later on, but the idea is that we can represent any scenario in code as long as we can break it down into specific steps.

The logic sample above is not in any programming language. But this type of approach to writing down the logic is useful regardless of what programming language is used. For example, on the first line, we don't need to know how to check the day of the week. Rather, we just state that this is what we're intending to do. Within a programming language, it could be one line of code or a dozen lines of code, but we care more about the logic at this point instead. We'll dive into this example a bit further in the upcoming lesson.

---

4. Coding Example

We discussed the basic elements of input, processing and output, so let's consider how we can use a computer program to do something that we want. Consider a program that asks the user to enter in two numbers, calculates the sum, and then outputs the results to the user. In order to break this down, we want to think about what we're trying to do in logical steps with input, processing, and output. Rather than get lost in syntax, we can write it in **pseudocode** or English-like statements. We want to start by first asking the user to enter in a number; otherwise, the user wouldn't know what they should be doing:

```
output "Enter in the first number:"
```

The next step is to take the user's input and store that into a variable. We'll simply name it as `firstNumber`, as that reflects what it is supposed to contain. This is an important part to consider. You want to define the variable so that later on in the code, you know what it references. If you name it as X or Y, in a larger program, it'll be hard to keep track without constantly referencing back to the start of the code when it was first defined. Notice as well that we don't have any spaces in the variable. This is important too, because in programming languages, variables cannot contain any embedded spaces.

Remember that a variable is a named memory location that has the ability to hold various distinct values at different points in time in the program. For example, the first time the program is run, we may have it set to 1. The next time, we could have it set to 1000. If we needed to change the value later on in the program, we could do so as well.

```
input firstNumber
```

The next step is to prompt the user for the second number. Take note that we should always quote the literal string. A **literal string** is simply a series of characters that are combined together and enclosed in quotes. For example, "Enter in the second number" is a literal string. A **string** is a data type that can store a literal string as a value. You will learn a lot more about strings in a future lesson.

```
output "Enter in the second number:"
```

Similar to the first number, we'll store it in the variable `secondNumber`.

```
input secondNumber
```

The next step is the processing step, where we will take the sum of the `firstNumber` and the `secondNumber` and store it in a new variable called `total`.

```
total = firstNumber + secondNumber
```

The last step will output a string of text and then the calculated total from the prior step. What's unique about this statement is that it combines a literal string with the value of the variable as part of the output.

```
output "The total is:" + total
```

Combining this all together, it would look like the following:

```
output "Enter in the first number:"
input firstNumber
output "Enter in the second number:"
input secondNumber
total = firstNumber + secondNumber
output "The total is:" + total
```

If this were a program that was being executed (run), it would look like the following:

STEP BY STEP

Enter in the first number:
Then, the computer would wait until the user typed in the value:
50
This value that was entered in would be stored in the variable `firstNumber`. Then, the computer would output the following to the screen:
Enter in the second number:
Then, the computer would wait until the user typed in the value:
33
This value would be entered in and stored in the variable `secondNumber`. The next line wouldn't output anything to the screen. Rather, it is a processing line that does the calculation. It takes the value of `firstNumber` and adds `secondNumber` to it to be stored in the variable `total`. After the line is run, the variable `total` has the value of 83 as 50+33 = 83. There are no changes to the `firstNumber` or `secondNumber`.

Lastly, the last line would take the literal string and concatenate it with the value of the variable `total`, resulting in the following output:

The total is: 83

KEY CONCEPT

You just read the term concatenate (concatenation) and you will hear this term throughout this course. Concatenate means to combine words or numbers. For example, to concatenate the numbers 1234 and 5678 would result in 12345678. To concatenate the words "snow" and "ball" would result in "snowball." There are specific features in Python that allow values to be concatenated together. We will cover those in upcoming lessons.

TERMS TO KNOW

Pseudocode
English-like statements that describe the steps in a program or algorithm.

String
A data type that can store a literal string as a value.

Literal String
A series of characters that are combined together and enclosed in quotes; for example, "Enter in the second number" is a literal string.

SUMMARY

In this lesson, we learned that the **first steps** of solving a problem are to break down the problem into the smallest possible steps needed to perform the necessary tasks. We used a **real-world example** of making orange juice to see this breakdown. Then, we added the use of conditions in the **"Is it raining or snowing?"** scenario where decisions can be added to the process. Finally, we also looked at **coding examples** to understand how to break down problems into individual steps.

Best of luck in your learning!

TERMS TO KNOW

Literal String
A series of characters that are combined together and enclosed in quotes; for example, "Enter in the second number" is a literal string.

Pseudocode
English-like statements that describe the steps in a program or algorithm.

String
A data type that can store a literal string as a value.

Variable
A named memory location that has the ability to hold various distinct values at different points in time in the program.