Thinking Through the Steps to Solve a Problem

by Sophia

WHAT'S COVERED

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

Table of Contents

1. First Steps

It is important to understand what problem you would like to solve before designing a program to solve it. Think about the goal, or purpose of the program, and what users should be able to expect when they run it.

EXAMPLE When running a program to convert temperature from Celsius to Fahrenheit, the user should expect that if 0 degrees Celsius is entered, the result should be 32 degrees Fahrenheit.

HINT

The steps should make sense to the end user. You will get a chance to go through some examples of these necessary steps as we progress in the course.

Once the goal and purpose of the program are defined, the next steps involve breaking the problem down into the smallest possible individual steps that are needed to perform necessary tasks. It is important to not leave out any instructions. It is also important to not add extra instructions that are not needed for the program.

For the program to work correctly, the logic and syntax must be correct. Specific syntax of Java will be discussed later in the course. For this purpose, you can think of syntax as specific grammar rules for the language.

HINT

Programming syntax will be indicated by monospace font throughout the course to help differentiate programming language from written English.

For the next few steps, focus on planning the logic of a program. In order to write a program that works properly, the correct logic, the correct program instructions, must be in the correct order. If certain steps are not performed in the right order, there could be several potential issues, where specific steps would not make sense in the order they are written.

> **IN CONTEXT**
> 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? Would it make sense to put soap on your face after you're done drying your hands? Translated to a Java program, the steps must have a starting point. The rest of the steps must come in the order they are intended to be done, starting at the starting point.

Once the steps are defined and are in the right order, the next step is to consider items that could be stored as variables. A **variable** is a term that you'll encounter quite often in programming. It is defined as a name that is used for a location in memory that can store a certain type of value. For now, when thinking about "types" of data, think about whole number values like 5, values with decimals like 3.141, and strings of characters like "hello". That value may change at various points throughout the program. This is okay as long as it is the same type of data.

Using the temperature conversion example, the temperature is a variable because the input from the user is necessary 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 the variables are defined, the next step is to review the logic beyond the sequence of steps. There may be scenarios where conditions, or branching of decisions exists, based on user input or other variables. For example, if the user is hungry, one path of statements could be taken. If they are not hungry, another path of statements could be taken. There could also be other elements to consider that include certain sequences of statements that need to run more than once. In these cases, the program should make use of loops to handle the repetition. Or as an alternative, it could break down those statements into reusable sub-parts of a program.

TERM TO KNOW

Variable
A named memory location for a certain type of value, that can hold various distinct values, at different points in time as the program runs.

---

2. Real-World Example

The initial steps of determining the goal and purpose of writing a program can seem overwhelming, when considering all of the steps involved. It is critical to break things down into smaller steps to grasp all that is involved! There are many examples around us that demonstrate the breaking down of problems, determining how the steps are related, and how they are linked.

Someone making a glass of orange juice might perform these steps:

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

When evaluating the steps to make orange juice, it appears that vegetable oil was added to the orange juice. That would not work so well. Additionally, the steps that were established are also out of sequence. Some steps are missing, and it is obvious that some steps are intended for some other procedure, not for making orange juice. The logical order of the statements is important. A computer program can only execute the steps that it is instructed to. And, it can only execute the steps in the specific order it was instructed to. It is critical to reorder the steps to achieve the desired result.
The steps and order of steps should be entered as follows:

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 certain steps that could be swapped around in cases where the order doesn't matter. In this case, the first five steps that involve gathering items to complete the task could be listed in any order. As another example, consider daily tasks that you do. Assess them and consider how they could be broken down into substeps in a consistent manner that even a robot could follow them exactly as they are.

---

3. Is It Raining or Snowing?

The previous example of steps in a program were fairly simple to break down. As you can imagine, there are

other programs that involve more complicated sequences of steps. These complex scenarios require consideration of more factors. For example, Sophie walks to work Monday through Friday. She checks the weather forecast. If it is supposed to rain during the day, she brings an umbrella. If it is snowing, she will bring her winter jacket. On the weekends, she needs to take the car keys because she drives to visit her family on Saturday and visits her friends on Sunday.

Any scenario can be represented in code, as long as it can be broken down into specific steps. Start by breaking things down into various logical parts of the process.

↱ EXAMPLE

1. Check the day of the week and store in `dayOfWeek`.
2. If `dayOfWeek` is "Saturday" or "Sunday", then set `itemToBring` to "car keys".
3. Otherwise (if `dayOfWeek` is a different day), then check the weather forecast and store in `weatherForecast`.
4. If `weatherForecast` is equal to "rain", set `itemToBring` to "umbrella".
5. If `weatherForecast` is equal to "snow", set `itemToBring` to "winter jacket".

First, note that the selection criteria for what Sophie should bring with her is based on the day of the week. If it is Saturday or Sunday, Sophie will only need the car keys. Next, if it is Monday through Friday, Sophie also needs to check the weather to determine if she needs an umbrella or a winter jacket.

☆ BIG IDEA

Sophie's problem—what to bring with her each day—relies on conditions like day of the week and weather. These conditions are where decisions need to be made.

CONCEPT TO KNOW

When considering the logic sample above, you will notice that it is not written in a programming language. However, considering that it may be translated into code at some point, it is important to write down the logic, as it will be useful regardless of what programming language is used. An example of this can be found on the first line. We do not need to know how to actually check the day of the week at this point. In a given programming language, this step could be one line of code or a dozen lines of code. It is more important to understand the set of logical steps, not just the condition. This example will be discussed a bit further in an upcoming tutorial.

---

4. Coding Example

Now that you are aware of the roles of the basic elements of a process, it is important that you consider how to use a computer program to perform a desired task. 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, it is important to think about the tasks (logical steps) related to input, processing, and output. Rather than get lost in syntax, it can be written in **pseudocode** or English-like statements that describe the steps in the process.

STEP BY STEP

1. Start the program by asking the user to enter a number. If the program does not do this, the user would not know what should be done next:

```
output “Enter in the first number: ”
```

2. Take the user’s input and store that in a variable. It can simply be named `firstNumber`, since that name reflects what it is supposed to contain. This is an important part to consider when defining the variable, so that later on in the code, it can be referenced easily. If the variable is named `X` or `Y`, in a larger program, it will be hard to keep track without constantly looking back to the start of the code when it was first defined. Notice as well that there are no spaces in the variable name. This is important, as variables cannot contain any embedded spaces in most programming languages.

```
input firstNumber
```

HINT

Remember that a variable is a named memory location that has the ability to hold various distinct values of a particular type. For example, variables can be whole numbers, numbers with decimals, strings of characters, and many other values at different points in time in the program. For example, the first time the program is run, the value can be set to 1, and the next time it could be set to 1000. If it is necessary to change the value later on in the program, this could be done as well. This is why a variable is intentionally referred to as a “variable” instead of a “constant value.”

3. Prompt the user for the second number. Remember that the prompt should always quote the literal string. A **literal string** is simply a series of characters that are combined together. A **string** is a data type that can store a literal string as a value.

```
output “Enter in the second number: ”
```

4. The user's answer should be stored in the variable secondNumber.

```
input secondNumber
```

5. Processing the sum of the variable firstNumber and the variable secondNumber is stored in a new variable called total.

```
total = firstNumber + secondNumber
```

6. Output a string of text followed by 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:

First, the computer would display the following output on the screen:

Enter in the first number:

Then, the computer would wait until the user typed in the value:

↱ EXAMPLE
50

The value that was entered by the user 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:

EXAMPLE

33

This value would be entered in and stored in the variable `secondNumber`*. The next line would not output anything to the screen because it is a processing line that does the calculation. It takes the value of* `firstNumber` *and adds it to the* `secondNumber` *and stores the sum 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

CONCEPT TO KNOW

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". To concatenate our literal string and the value of our variable would result in "The total is: 83". There are specific features in Java that allow values to be concatenated together. We will cover those in upcoming tutorials.

TERMS TO KNOW

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

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

Literal String
A series of characters that are combined together.

SUMMARY

In this lesson, you learned that the **first step** of solving a problem is to break down the problem into the smallest possible steps needed to perform the necessary tasks. You reviewed a **real-world example** of breaking tasks down using the process to make orange juice. You then considered the use of conditions like, **is it raining or snowing**? You learned in this scenario the situations where decisions can be added to the process. Finally, you looked at **coding examples** to reflect on breaking problems down into their individual steps.

Source: This content and supplemental material has been adapted from Java, Java, Java: Object-Oriented Problem Solving. Source **cs.trincoll.edu/~ram/jjj/jjj-os-20170625.pdf**

It has also been adapted from “Python for Everybody” By Dr. Charles R. Severance. Source **py4e.com/html3/**

TERMS TO KNOW

Literal String

A series of characters that are combined together

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 as different points in time in the program.