Forming the Algorithm

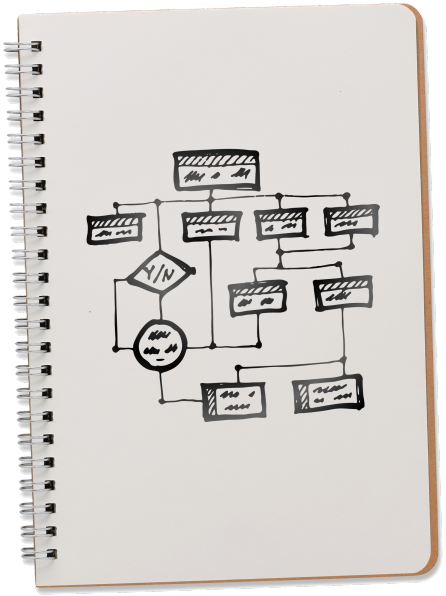
by Sophia

WHAT'S COVERED

In this lesson, you will learn how to plan for algorithms to ensure that all of your coding is incorporated. Specifically, this lesson covers:

1. Planning for Algorithms

There are some common algorithms that can be used or added to your code since they represent common ways to approach a problem. Rather than reinvent the wheel each time, using these algorithms can be useful to avoid issues in your code. Let's look at some of the different repeating patterns that could be put in place.



We've already started previously with some key patterns around the program. Although this is a great starting point, we do have a lot more to expand on as well. We need to be able to think about how the entire program works as we currently only have the logic around a single attempt of the game.

EXAMPLE

Set the rolled value to roll first dice + roll second dice
If the rolled value is equal to 2, 3 or 12
    Set player status to lose
Else If the rolled value is equal to 7 or 11
    Set player status to win

```
Else
    Set point value to rolled value
    While the player status is not set, run the following
        Set the rolled value to roll first dice + roll second dice
        If the rolled value is equal to 7
            Set player status to lose
        Else If rolled value is equal to point value
            Set player status to win
```

There is more to the program that we need to consider beyond the logic of a single play. If you remember, there's the specific output that our friend wanted in the program that would be used to track the process, including:

- Total number of wins
- Total number of losses
- Average number of rolls per win
- Average number of rolls per loss

We know already that our friend wanted to be able to track the data across multiple iterations of the game. This would need to be added to the existing logic. We could use a function that we'll call "play" in which we'll just have the following pseudocode put into it:

↷ EXAMPLE

```
Function play()
    Set the rolled value to roll first dice + roll second dice
    If the rolled value is equal to 2, 3 or 12
        Set player status to lose
    Else If the rolled value is equal to 7 or 11
        Set player status to win
    Else
        Set point value to rolled value
        While the player status is not set, run the following
            Set the rolled value to roll first dice + roll second dice
            If the rolled value is equal to 7
                Set player status to lose
            Else If rolled value is equal to point value
                Set player status to win
```

Now, to help track statistics for multiple plays of the game, including wins, losses, number of rolls per win, and number of rolls per loss, we would need to place the logic into a loop. We would wrap some of that logic into a larger program by first defining the variables to handle each of those items.

```
Set Wins = 0
Set Losses = 0
```

Set Total of Win Rolls = 0
Set Total of Loss Rolls = 0
Set Games Played to 0

Next, we can define the loop of how the game would be played. The logic of most of the program would be in the play function but the added code just fits into the key criteria.

EXAMPLE

Input Times to Play from user
Loop until Games Played = Times to Play
    Add 1 to Games Played
    Call function play()
    Get If the player won or lost
    Get the number of rolls
    If the player won
        Add 1 to Wins
        Add the rolls values to the Total of Win Rolls
    Else If player lost
        Add 1 to Losses
        Add the rolls values to the Total of Loss Rolls

Once we're done with defining the looping of the game, we have to calculate the final values. For example, the average number of rolls per win requires us to calculate Total of Win Rolls divided by the Wins. Likewise, the average number of rolls per loss requires the same calculation:

Average Number of Rolls Per Win = Total of Win Rolls / Wins
Average Number of Rolls Per Loss = Total of Loss Rolls / Loss

In addition, we want to indicate the calculation for the winning percentage, which would be by taking the number of Wins and dividing it by the Games Played:

Winning Percentage = Wins / Games Played

Putting this together now, we have a few more parts but note that this is just a starting process for the design of the program. It's not meant to include all of the functional elements, but rather it is more of a guide for the algorithms that the program should take.

TRY IT

Directions: Before moving on, see if you can pseudocode out what steps you believe this program will need. Once you are done, see how close you are to the code below.

↷ EXAMPLE

```
Function play()
    Set the rolled value to roll first dice + roll second dice
    If the rolled value is equal to 2, 3 or 12
        Set player status to lose
    Else If the rolled value is equal to 7 or 11
        Set player status to win
    Else
        Set point value to rolled value
        While the player status is not set, run the following
        Set the rolled value to roll first dice + roll second dice
        If the rolled value is equal to 7
            Set player status to lose
        Else If rolled value is equal to point value
            Set player status to win

Main Program
    Set Wins = 0
    Set Losses = 0
    Set Total of Win Rolls = 0
    Set Total of Loss Rolls = 0
    Set Games Played to 0

    Input Times to Play from user
    Loop until Games Played = Times to Play
        Add 1 to Games Played
        Call function play()
        Get If the player won or lost
        Get the number of rolls
        If the player won
            Add 1 to Wins
            Add the rolls value to the Total of Win Rolls
        Else If player lost
            Add 1 to Losses
            Add the rolls value to the Total of Loss Rolls

    Average Number of Rolls Per Win = Total of Win Rolls / Wins
    Average Number of Rolls Per Loss = Total of Loss Rolls / Loss
    Output results
```

And finally, we add the winning percentage:
Winning Percentage = Wins / Games Played

2. Adding Our Third Journal Entry

KEY CONCEPT

As you start building your logic, you want to have as much of the algorithm detailed out as possible to make the conversion to code more easily performed. Remember that the pseudocode is meant to describe the application at a high level. The pseudocode should, at a high level, be as optimized as you can have it to be before the conversion to the actual programming language.

Bad Example of Journal Entry for Part 3

A bad entry would not fully break down and identify the patterns to add conditional statements, loops, or functions where necessary. A bad entry could be more based on a single run of the program, similar to what we created as part of the previous lesson, like the following:

EXAMPLE

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.

The problem with this type of code is that it has not been fully broken down and uses specific examples.

HINT

The reason why is because this pseudocode is using specific values vs. variables. It doesn't use conditional statements or loops to optimize the code. It doesn't improve on the prior entry by adding to or improving on those patterns.

Good Example of Journal Entry for Part 3

The following would be considered as a good entry for the journal:

EXAMPLE

```
Function play()
    Set the rolled value to roll first dice + roll second dice
    If the rolled value is equal to 2, 3 or 12
        Set player status to lose
    Else If the rolled value is equal to 7 or 11
        Set player status to win
    Else
        Set point value to rolled value
        While the player status is not set, run the following
            Set the rolled value to roll first dice + roll second dice
            If the rolled value is equal to 7
                Set player status to lose
            Else If rolled value is equal to point value
                Set player status to win

Main Program
    Set Wins = 0
    Set Losses = 0
    Set Total of Win Rolls = 0
    Set Total of Loss Rolls = 0
    Set Games Played to 0

    Input Times to Play from user
    Loop until Games Played = Times to Play
        Add 1 to Games Played
        Call function play()
        Get If the player won or lost
        Get the number of rolls
        If the player won
            Add 1 to Wins
            Add the rolls values to the Total of Win Rolls
        Else If player lost
            Add 1 to Losses
            Add the rolls values to the Total of Loss Rolls
    Average Number of Rolls Per Win = Total of Win Rolls / Wins
    Average Number of Rolls Per Loss = Total of Loss Rolls / Loss
    Winning Percentage = Wins / Games Played
    Output results
```

If we preview the Example Python Journal Submission document, we see this was added as the entry to Part 3. Although you could have a lot more code to break things out like defining the output of the results, this gives us enough information to move on.

---

3. Guided Brainstorming

As you start building your program, you'll need to start thinking about the bigger picture and the different aspects. With the pseudocode, you are only worrying about the logic, but you'll want to think about where the functionality needs to be enhanced and expanded on. Each program will have some unique criteria or a menu structure. Think about where the exceptions may be. For example, are you working with files? If so, how do you ensure that the program can handle issues with the files not being there? These are things to keep in the back of your mind as you design the code. Can you walk through the program from start to finish with all the examples? If so, you're ready to move on to start designing your code.

If you have trouble here, think about the algorithms, and try to break things down into small parts. From here, you should be able to identify the variables that need to be used and what needs to be calculated afterwards. With the example that we also looked at in the prior lesson, this was well defined already:

↪ EXAMPLE

```
Ask user to enter in "water, coffee or tea"
Store input into drink
Store drink in outputString
If drink is equal to water
    Ask user to enter in hot or cold
    Store input in heat
    If heat equal to hot
        Add hot to outputString
    Else If heat equal to cold
        Add cold to outputString
        Ask user to enter in ice or not
        If ice is yes
            Add ice to outputString
        Else
            Add no ice to output String
Else If drink is equal to coffee
    Ask user to enter in decaf or not
    Store input in decaf
    If decaf equal to Yes
        Add decaf to outputString
    Ask user to enter in Milk, cream or nonet
    Store input in milkCream
```

```
    If milkCream equal to milk
        Add milk to outputString
    Else If milkCream equal to cream
        Add cream to outputString
    Ask user to enter in sugar or not
    Store input in sugar
    If sugar equal to Yes
        Add sugar to outputString
Else If drink is equal to tea
    Ask user to enter in teaType
    Store input in teaType
    If teaType equal to green
        Add green to outputString
    Else If teaType equal to black
        Add black to outputString
print outputString
```

In this case, we wouldn't need to change anything unless we wanted to add in a menu structure or allow multiple drinks per order. Those things are elements that you want to think about in your own program.

TRY IT

Directions: At this point, you should have a good idea of what your program should look like. Take the time to think about the big picture of the program and go beyond just the core logic of the program. Also, look at what the completed program should run like. Review the example of a good entry for Part 3 in the Example Python Journal Submission document and add your entry for Part 3 to your Python Journal.

SUMMARY

In this lesson, we continued **planning for the algorithm** using simple pseudocode and the demonstration program. Based on what our friend wishes for this program, we included some additional logic to store some statistical information. We created a single attempt of the gameplay and placed it into a play function. Then, we added the functionality to create the statistics and allow for multiple game plays by calling the play function we created. We had an opportunity to see a good example for our **third journal entry**. In the **Guided Brainstorming** section, we identified another example of good pseudocode formation.

Best of luck in your learning!