2.3.2 Loop Over All Tickers

To run analyses on all of the stocks, we need to create a program flow that loops through all of the tickers.

One way to accomplish our task—running an analysis of all stocks—is to copy the code from the Daqo analysis and paste it over and over, changing the ticker and the line to output each time. However, computers are better at repeating themselves than humans are, so we'll let VBA handle that part.

NOTE

> In general, when it comes to programming, follow the advice of the acronym **DRY: Don't Repeat Yourself**. If you're reusing a piece of code over and over again, there's probably a better way to do it. After all, if you have to go back and edit your code, it's better to do it in just one place and then be done with it. When there are multiple places where you need to make the same edit, it's easy to miss one—which could result in a bug that's difficult to troubleshoot.

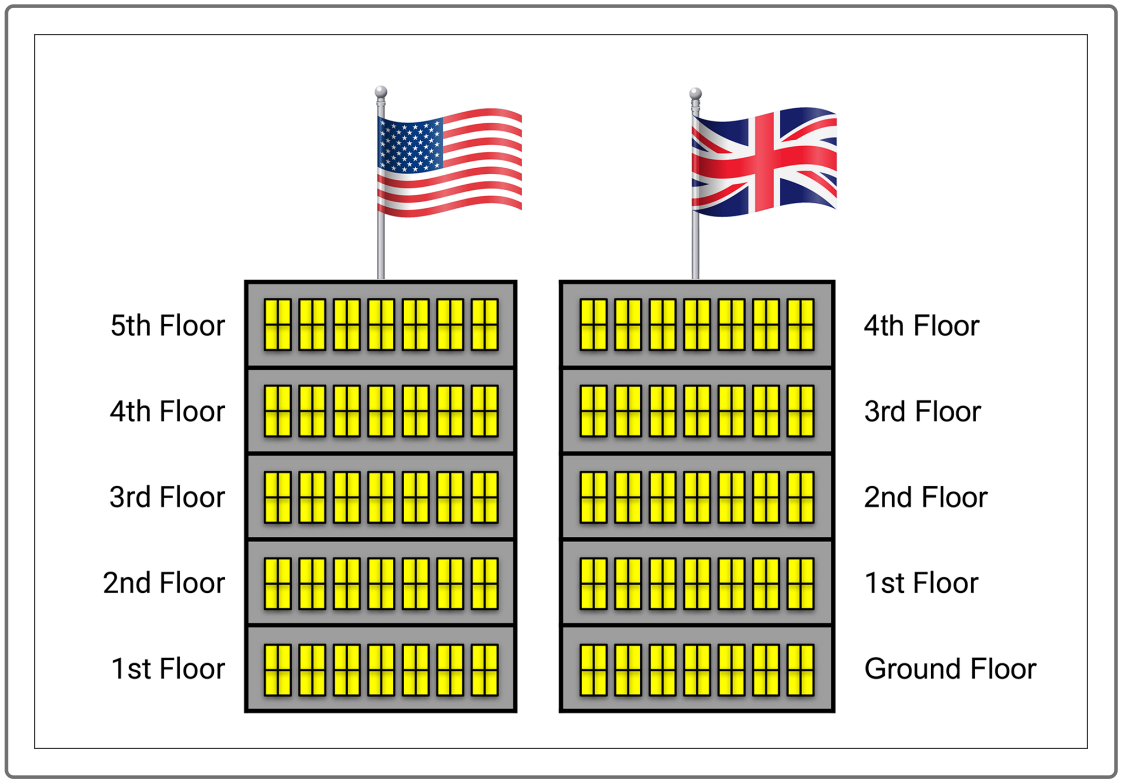
Arrays in Programming

Instead of repeating our Daqo analysis code over and over and changing the bits that are stock-specific, we need to create a list of tickers and have VBA handle reusing the code. Luckily, we can do this with a `for` loop and an **array**.

IMPORTANT

> **Arrays** are another fundamental building block in programming languages. In some languages, arrays are known as **lists** because they work like a shopping list. Arrays hold an arbitrary number of variables of the same type. Instead of giving each variable a different name, we give the array a name and then access the individual variables by their index.

An **index** of a variable is its position in the array. Almost all programming languages start their index at zero, which means that a variable at index 1 will be the second element in the list. To conceptualize this, think about how floors of buildings in the U.S. are numbered compared to buildings in Great Britain. In the U.S., the floor level of a building is usually referred to as the "first floor," and the floor above that the "second floor," and so on. In Great Britain, however, the floor level is called the "ground floor," and the floor above it is considered the "first floor." In VBA, we could say indexes follow the British model.



Retake

Arrays in VBA

In VBA, arrays are initialized with the `Dim` keyword, but with a couple of key changes:

- Insert a number in parentheses after the array name that represents the number of elements in the array.
- Specify the type of variable for each element in the array.

For example, the code to create an array to hold 12 tickers would be:

```
Dim tickers(11) As String
```

To access the elements in the array, we'll put the index in parentheses after the array name `tickers`. Don't forget that we're starting at zero. This means `tickers(0)` will be the first element in the list and `tickers(11)` will be the 12th and final element of the array. When we reference a single element in the array, we treat it like any other variable: we assign it a value, check if it equals another variable, and so on.

Let's assign each of the tickers to an element in the array. That code should look like this:

```
tickers(0) = "AY"
tickers(1) = "CSIQ"
tickers(2) = "DQ"
tickers(3) = "ENPH"
tickers(4) = "FSLR"
tickers(5) = "HASI"
tickers(6) = "JKS"
tickers(7) = "RUN"
tickers(8) = "SEDG"
tickers(9) = "SPWR"
tickers(10) = "TERP"
tickers(11) = "VSLR"
```

Nested Loops

Now we're going to loop through the array. For each element in the array, we'll run the same analysis we did for DQ. This means we'll be running a `for` loop inside of another `for` loop! Loops inside loops are called **nested loops**.

In theory, we can nest as many for loops as we want—but that means it can quickly become difficult to keep track of which loop you're working in. This is another scenario where clean code formatting comes into play. Remember whitespace? This is a great time for it. We'll add one more indentation in the code each time we begin a new inner loop. This way, when we're writing our code, we'll know which loop the code belongs to.

Additionally, we need to make a new iterator variable for our inner loop. A simple nested `for` loop with proper indenting will look something like this:

```
For i = 1 To 10

    'a line of code here will run 10 times

    For j = 1 To 20

        'a line of code here will run 200 times

    Next j

Next i
```

Note that we must end the inner loop before we can end the outer loop; otherwise, VBA will get confused, give up, and throw a compile error.

To loop through all of the stocks, let's use the index `i` to access each element in the array. The code will look like this:

```
For i = 0 To 11

    ticker = tickers(i)

    'Do stuff with ticker

Next i
```

Let's get some practice using nested `for` loops.

SKILL DRILL

1. Create a nested `for` loop that puts a 1 into the cells of all columns A through J, and rows 1 through 10 (e.g., cells A1 - A10, B1 - B10, and so forth).

2. Modify the nested `for` loop to fill in a variable number of cells and rows (Hint: use variables).
3. Modify the nested `for` loop again to put the sum of the row number and column number into each cell.
4. Finally, using the same looping mechanism, find a method (through Stack Overflow or another site discussed previously) used to clear the content of the cells you have been editing in the previous steps.

ADD/COMMIT/PUSH

Save `green_stocks.xlsm` and push the changes to the "stocks-analysis" repository in GitHub.