CHAPTER 3

Introducing Formula Concepts

Q: What does every newborn spreadsheet need?
A: Formula

Spreadsheet formulas hold a unique place in advanced Excel development. Most of us are familiar with formulas as a means to produce results more quickly than with manual calculation. For example, if we want to find the arithmetic sum of a range, does it make sense to pull out the Burroughs Adding Machine and punch in each item one by one? No. The very nature of a spreadsheet provides a built-in means to manipulate its elements.

Most of us are used to this type of manipulation with formulas; that is, we use formulas as a means to find and return results. Spreadsheet formulas, when used for Excel development, however, do much more. They form the infrastructure upon which much of our work is based.

Throughout this book we will be working with formulas. Some of these formulas will be very complex. When you first start, they may appear daunting. However, practice makes perfect, and experience is your greatest teacher. The more you use them, the more you develop a formula literacy. What may have appeared hard to read at first glance should become easier. But more important than knowing the formulas themselves is understanding the concepts behind what drives them.

And, of course, Excel includes a few tools and features to help you understand your formulas. Let’s go through a few of them you can start using now.

Formula Help

In this section, I’ll talk about making the most of your formula experience. The following tips should make your life easier, especially when working with complex formulas.

F2 to See the Formula of a Select Cell

Chances are you’re already pretty familiar with F2. But for the uninitiated, pressing the F2 key on a cell containing a formula will highlight the portions of a spreadsheet upon which the formula depends. If you’re trying to evaluate a formula, F2 is a good first start to your investigation.

F9 for On-Demand and Piecewise Calculation

F9 is the shortcut key to tell Excel to recalculate. If you type =RANDBETWEEN(1, 2) in an empty cell on an Excel worksheet and then press F9 continuously, you will see that cell update to 1 or 2 at random. (In addition, if you have any other volatile formulas, those will update too).

F9 can also provide a piecewise, or partial, calculation of a long formula. Take the seemingly complex formula shown in Listing 3-1.
CHAPTER 3 — INTRODUCING FORMULA CONCEPTS

Listing 3-1. An Example of a Long, Complex Formula

=IF(SUMPRODUCT(A1:A3*(B1:B3>2))>7, CONCATENATE(A2 & L3), IFERROR(C6, "An error occurred."))

Let's say you want to evaluate only a part of this formula, specifically the highlighted portion of the same formula but now in Excel's formula bar (Figure 3-1).

Figure 3-1. You can select a portion of the formula to be evaluated immediately

In fact, you can tell Excel to evaluate just that easily. If you highlight the portion as I've done in Figure 3-1, you can press F9 to see what it evaluates to (see Figure 3-2).

Figure 3-2. Pressing F9 on the highlighted portion evaluates the highlighted portion immediately

You now see this portion evaluates to False. In the formula bar, Excel just rewrites this portion of highlighted text to read “FALSE.” And you can do this to any portion of the formula. If you click outside the formula bar or press the escape key, the formula will return to its original, unevaluated text. F9 then, when used with formulas, is the ultimate on-demand approach for quick formula evaluation.

Evaluate Formula Button

The Evaluate Formula button allows you to step through an entire formula. Here's how it works. First, click the cell you're interested in investigating. Then, click the Formulas tab on the ribbon. Go to Evaluate Formulas in the Formula Auditing group. Take a look at Figure 3-3.

Figure 3-3. The Evaluate Formula button

A dialog box similar to the one shown in Figure 3-4 should appear. The underlined portion is the current expression to be evaluated. If available, you can go deeper into the formula by pressing the Step In button. You can Step Out if that level of granularity is no longer need. For formulas that resolve to an error, the Evaluate Formula tool can be very helpful to understand the conditions right before the error. I find Evaluate Formula an indispensable part of my Excel Development toolkit.
Excel Formula Concepts

In this section, I’ll talk about formula concepts you’ll be using throughout the rest of this book. To begin, Excel formulas are made up of four main types:

- Functions, such as `AVERAGE()`, `SUM()`, `IF()`
- Constants and literals, such as number, string, and Boolean values like 2, 100, 1E7, “Hello world”, and `FALSE`
- References, such as `A1` or `A$1$:A$20$
- Operators, such as `+`, `-`, `/`, `>`, `:`

You’re probably already familiar with several of these types. Obviously, functions make up a huge part of formula use. Constants that are numbers are also probably familiar. However, did you know that Boolean values like `TRUE` and `FALSE` are also constants? Finally, you’ve probably used references and operations many times by now, but did you know the colon (:) that forms the range `A$1$:A$20$` is also an operator?

Operators, in Depth

This section will discuss Excel operators. You’re probably familiar with Excel’s arithmetic operators, plus (+), minus (-), times (*), and divide (/). But besides arithmetic operators, Excel has a text and three reference operators.

Excel’s text operator is the ampersand (&), which stands in for the `CONCATENATE` function. For instance, the formulas `=A1&B1` and `=CONCATENATE(A1,B1)` do the exact same thing. You’ve probably also used Excel’s reference operators many times, the colon (:) in particular, without thinking of them as operators. Excel’s two other reference operators are the comma (,) and space ( ) characters. Table 3-1 talks about what they do.
In the next few sections, I’ll go through examples of what you can do with these reference operators.

The Range Operator (:)  

In this section, I discuss the range operator. The range operator (:) is one of the most used operators in Excel. It’s an operator in every sense of the word in that it acts upon two different ranges (which are the operands, if you want to get technical) and returns a contiguous range. What’s so great about the range operator is that you can actually combine functions, like

\[
= \text{A1:INDEX(A:A, COUNTA(A:A))}
\]

and

\[
= \text{B1:OFFSET(B:B, COUNTA(B:B), 0)}
\]

So let’s take a look at an example that shows the power of the range operator.

### EXAMPLE: DYNAMICALLY SIZED RANGES

Using the range operator, you can create dynamically sized ranges. This means you can create a range that can grow and shrink as the list they represent is added to or subtracted from. Both the `INDEX` and `OFFSET` formulas can help you with this mechanism. In this example, they both work about the same way.

Consider the range in Figure 3-5.

![Figure 3-5. A sample set of data upon which you will create a dynamically sized range](image)
If I want a count of all my favorite colors in this example (in real life, I have only one favorite color, and it’s black), I can use the COUNTA function on the range A2 to A8. But what if I want to add to the list? In that case, I must reapply my formula to accommodate the next color in cell A9. Alternatively, I can just say something like A2:A1000, where the second range is an arbitrarily large number. Neither the former’s formula reapplication nor the latter’s arbitrarily high number are very good fixes.

The best solution is to use a dynamically sized range. To do this with the INDEX formula, you can write =$A$2:INDEX($A:$A,COUNTA($A:$A)) like in Figure 3-6.

**Figure 3-6.** A demonstration of the formula that will ultimately help you create a dynamically sized range

Here’s how it works. You supply the entire column range A:A to the INDEX formula. In the row argument of the INDEX formula, you’re interested in the last row of content in the column range of A:A. COUNTA, which counts every filled cell in the range supplied to it, will return an 8, since the last row of content is the eighth row down. When you use INDEX, you’re probably used to its returning values. If you hadn’t added that A1 at the beginning of the formula, the INDEX function by itself would have simply returned the word “Violate.” But behind the scenes, Excel is actually returning a reference to the cell containing “Violate,” not just its value. So, effectively, Excel is returns A8, which becomes A1:A8 in the formula.

When you press Enter, you’ll probably see the formula return the value Red. This is because it’s returning the top of the range. If you continue to drag the formula down, you’ll see that it returns the other cells in the range too (if it doesn’t, select the entire range and press Ctrl+Shift+Enter). But to really use dynamically sized ranges to your advantage, you can assign them to a named range as I’ve done in Figure 3-7. Make sure when you do it the cell references are absolute.

**Figure 3-7.** Creating a new named range out of the formula
You can then use that named range elsewhere on your spreadsheet. For example, in cell C8 in Figure 3-8, I’ve used the formula =COUNTA(myNamedRange). As you can see, I’ve added to my list, and the count has updated automatically. Just imagine using these dynamically sized ranges in charts, dropdowns, and formulas! You’ll get to do that in the next chapter.

![Figure 3-8. Using the Name Range elsewhere](image)

You can do the same with OFFSET, using this formula:

=\$A\$2:OFFSET(\$A\$1,COUNTA(\$A:\$A),0)

Experiment a little and see if you can figure this one out. Remember, if you need help, use the formula help suggestions from the beginning of the chapter.

A final note is in order. There’s also some argument on whether INDEX is faster than OFFSET, since OFFSET is a volatile function (that means it will recalculate every time the sheet recalculates) and INDEX is not. In general, I prefer INDEX for this reason.

The Union Operator (,)

The union operator (,) is also likely familiar to you. The formula =SUM(A1:A10,C1:C5) employs the union operator to combine the two disparate ranges into one range upon which to take the sum. Unlike the range operator, which forms a contiguous range between two cells, the union operator essentially turns the two noncontiguous ranges into one long range. Think of it like this:

\( (A1:A10,C1:C5) = \{ A1:A10 \cup C1:C5 \} \)

In this next section, I’ll talk about how you can use the union operation to your advantage.
EXAMPLE: PULLING RANK

Let’s say you wanted to find where a certain number ranks within a series of numbers, when they’re ordered. For example, if you have an unsorted series of numbers (8, 4, 6, 1, and 2), you can use Excel’s RANK function to find where the number 6 resides in a descending list of these numbers.

In Figure 3-9, I have the formula =RANK(D2,$A$2:$A$6) in cell D2.

Figure 3-9. A demonstration of finding the rank of a given number within an unsorted list

RANK will automatically turn the range in the given series in descending order (by default, descending is selected; however, this can be changed in RANK’s third, optional parameter). The rank of the number 6 then is 2, as shown in Figure 3-10.

6 8 4 1 2

Six is highlighted and is in the second place in the region.

Figure 3-10. A visual representation of how this example works

This function only works when the input number (in D2 above) is a number in the set of the five given numbers. But what if you want to find where the number 4.4 resides in the ordered series? The formula, left as is, will return an NA( ) error if D2 is set to 4.4. To get around this, you need to add the input number to the set of numbers. You can do this with the union operator, like so:

=RANK(D2,($A$2:$A$6,D2))

If D2 = 4.4, the series ($A$2:$A$6,D2) becomes 8, 6, 4.4, 4, 1, 2, which returns the number 3. Consider how this formula might be useful. If you have a list of times, dates, or temperatures and want to return certain information when an input value is between two boundaries, you can do that with this formula.

The Intersection Operator ( )

The intersection operator ( ), demonstrated as one space, returns one or more cells from overlapping ranges. Figure 3-11 shows that the intersection of range D2:D6 and B4:F4 is 3. You can verify that both of the ranges intersect, or overlap, at cell D4.
INTRODUCING FORMULA CONCEPTS

You'll learn a creative use for the intersection operator in this next example.

**INTERSECTING REGIONS AND MONTHS**

Let's say you have a table of units sold by month and region, like in Figure 3-12.

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>G</th>
<th>H</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>North</td>
<td>Jan</td>
<td>Feb</td>
<td>Mar</td>
<td>Apr</td>
<td>May</td>
<td>Jun</td>
</tr>
<tr>
<td>3</td>
<td>South</td>
<td>326</td>
<td>880</td>
<td>42</td>
<td>59</td>
<td>745</td>
<td>621</td>
</tr>
<tr>
<td>4</td>
<td>East</td>
<td>974</td>
<td>830</td>
<td>414</td>
<td>462</td>
<td>670</td>
<td>551</td>
</tr>
<tr>
<td>5</td>
<td>West</td>
<td>201</td>
<td>747</td>
<td>388</td>
<td>748</td>
<td>163</td>
<td>135</td>
</tr>
</tbody>
</table>

*Figure 3-12. A sample set of regional and monthly data*

To save time, you've had a macro assign columns B through H to be the named ranges Jan, Feb, Mar... etc. You've done the same thing for each region, assigning the row ranges to North, South, East, and West.

Then, if you're interested in the sum total of units sold in the East region on January and March, you can use the formula =SUM(East Jan:March), as shown in Figure 3-13.

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>G</th>
<th>H</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>North</td>
<td>Jan</td>
<td>Feb</td>
<td>Mar</td>
<td>Apr</td>
<td>May</td>
<td>Jun</td>
</tr>
<tr>
<td>3</td>
<td>South</td>
<td>326</td>
<td>880</td>
<td>42</td>
<td>59</td>
<td>745</td>
<td>621</td>
</tr>
<tr>
<td>4</td>
<td>East</td>
<td>974</td>
<td>830</td>
<td>414</td>
<td>462</td>
<td>670</td>
<td>551</td>
</tr>
<tr>
<td>5</td>
<td>West</td>
<td>201</td>
<td>747</td>
<td>388</td>
<td>748</td>
<td>163</td>
<td>135</td>
</tr>
</tbody>
</table>

*Figure 3-13. An application of the union operator on sample regional and monthly data*
The formula returns 1366, which is the sum of 201, 747, and 388. If you want to see the performance for the eastern region for just the months of January and March but not February, you can use the following formula:

*=SUM(East Jan + East March)*

If you’re particularly mathematically minded, and hopefully you will be somewhat by the end of the next chapter, you can simplify this formula like so:

*=SUM(East (Jan, March))*

Note that *East Jan + East March = East (Jan, March)*, which parallels the Distributive Law of algebra.

I’ll go into this in a little more detail later in the next chapter.

**When to Use Conditional Expressions**

In this section, you’re going to dive deeper into conditional expressions. If you’ve used IF, then you’ve used a conditional expression before. Conditional expressions are all about testing things. For example, in the formula

*=IF(AB>2, "Yes", "No")*, the first argument, *AB>2*, is the conditional expression. Any expression that uses the logic operators, =, <, >, etc., is a conditional expression.

So you want to test the value of a cell and return a result if it passes a test or another result if it fails.

**Quick: which function should you use?**

Was your answer IF? If it was, then you’re not alone. The IF function feels like a natural choice, especially because the first parameter of the IF function calls for a logical expression. But there are also some instances where IF isn’t the best choice. The Excel MVP, Daniel Ferry, has gone so far as to argue that the IF function is the most overused function of all. And, as this chapter will demonstrate, there’s good reason to believe this.

**Deceptively Simple Nested IF Statements**

One supposed advantage to using the IF function is the ability to make use of nesting conditions. For example, if I have multiple compounding conditions, I can place IF statements inside the value_if_true and value_if_false parameters (Listing 3-2). In my experience, however, IF statements are nested far more often than they need to be.

**Listing 3-2. A Prototype of the IF Function**

*IF(logical_test, value_if_true, value_if_false)*

Even I have to admit that nested IF statements are unavoidable. But I like to save them for formulas that exhibit natural branching conditions. Consider

*=IF(ProjectStatus = "Stopped", IF(Err_Code=1, "Halted by internal error.","Unknown error."),"Project has NOT finished.")*

I would argue this is a good example of the problem with using nested IF statements. Its inherent logic naturally represents a branching condition (see Figure 3-14).
Sometimes it’s not always so clear whether the problems represent a compound branching condition. A good rule of thumb is to start from one of the possible results and work backwards. Ask yourself: does the result naturally follow from the test condition? In other words: does this result make sense given the conditions?

Confused? I hear you. Well, let’s consider the following example from Microsoft’s very own help guide, shown in Listing 3-3.

**Listing 3-3.** An Example of Nested IFs from Microsoft’s Excel Help

```
```

This formula returns a letter grade based on a student’s raw grade stored in A2. It’s a good example of a problem that makes for a poor branching condition. The grade you receive isn’t the result of *not receiving another grade*. (I know you’re scratching your head here but bear with me for a moment). Your letter grade is the result of where your score falls within one of five different numerical boundaries. If anything, this is a lookup problem. You could easily employ the RANK function example from above or use the MATCH function. But if you were to frame this problem organically, the reason a student receives an F is not because they didn’t receive a D, C, B, or A. The IF function above turns this lookup problem into a branching condition problem when it needn’t be.

Another common example involves using states as numbers. Consider the formula in Listing 3-4.

**Listing 3-4.** Another Example Using IFs That Isn’t a Branching Condition

```
=IF(A2=1, "Small",IF(A2=2,"Med", "Large"))
```

---

**Figure 3-14.** A flowchart showing the branching conditions of your IF statement
In this example, A2 holds an encoded Id or state. For an example like this, the states could be anything, but they usually form some natural ordinal scale. In the example above, the Ids map to the following results: 1=Small, 2=Medium, and 3=Large. We call these categories ordinal because they can be ordered naturally. Here again, IF is not a good choice. The problem presented is not a branching condition but rather a test of scale. Indeed, for formulas like these, the CHOOSE function is a much better choice.

CHOOSE Wisely

In this section, I’ll go through how to use CHOOSE, and why for some situations it makes for a better choice than IF. CHOOSE is much like IF; but it can more naturally deal with ordinal data. Listing 3-5 includes the prototype for CHOOSE.

Listing 3-5. CHOOSE() Prototype

CHOOSE(index_num, value1, value2,...)

CHOOSE analyzes the argument supplied to the index_num parameter and returns the value at the given index number. In the example above, when index_num is 1, value1 is returned; when index_num is 2, value2 is returned, and so forth.

In the previous instance, you could simply write =CHOOSE(A2, "Small", "Med", "Large"). This appears to be more closely align with the way this example is naturally formulated. Because of this, CHOOSE makes the data arrangement more easy to read and understand at first glance. Compare the two arrangements:

IF arrangement

=IF(A2=1, "Small",IF(A2=2,"Med", "Large")).

CHOOSE arrangement

=CHOOSE(A2, "Small", "Med", "Large")

GENERATING RANDOM DATA WITH CHOOSE()

CHOOSE is also great for generating random categorical or nominal data. This type of random data generation is particularly useful to create test data for your dashboard backend database. All it takes is the addition of the RANDBETWEEN function. Say you have categorical data of Big, Medium, and Little. You could generate data with the following formula:

=CHOOSE(RANDBETWEEN(1,3), "Big", "Medium", "Little")

Why This Discussion Is Important

Like the IF statement, CHOOSE can be useful for elements that appear on your next spreadsheet dashboard, decision support tool, or application.

A nested IF condition will attempt to evaluate every condition until a true value results or terminates to the end of the nest. CHOOSE makes one evaluation and goes to the specified index. On its face, CHOOSE would seem superior for scenarios in which a nested condition isn’t necessary. Fewer evaluations means fewer instructions for Excel to complete. In previous versions of Excel and on older machines, conserving machine processing by using optimal formula structures really did seem to make a difference. However, now that we’ve entered the age of multithreaded processors, I must admit the performance differences have become less noticeable.
So then why have I made the distinction? Well, using the formula that best matches what you’re trying to accomplish just makes sense. In addition, and perhaps more importantly, when you come back to your formula later after having been away from your spreadsheet for a while, a formula that better matches your test conditions will ultimately be easier to once again comprehend, especially if it’s complex in nature.

Ok, you’re not convinced. I wasn’t at first, either. In the end, there may not be a noticeable difference between using IF or CHOOSE, I admit. But in the previous chapter I turned conventional coding on its head. And I’ll keep doing so throughout this book.

And if you’re tempted to keep using IF, read on. Chances are you’ll find it at least one example in which IF isn’t necessary.

**Introduction to Boolean Concepts**

In this section, I’ll talk about concepts surrounding Boolean expressions. For the unfamiliar, Boolean formulas use a type of mathematical logic called Boolean algebra and they’re the natural result of conditional expressions.

The most important feature of a Boolean expression is that it always returns one of two mutually exclusive values: either it returns TRUE, or it returns FALSE. Excel, however, brings another important twist to the TRUE/FALSE dynamic. Sometimes TRUE can also mean the number one, and FALSE can also mean the number zero. Let’s take a look in the following example.

**FILTERING ODD OR EVEN VALUES**

Booleans are great for filtering. Take a look at Figure 3-15. In this example, I’ve created a mechanism to only show either odd or even values in the accompanying chart.

*Figure 3-15. Booleans used for filtering*
I provide the user a dropdown box to select between either showing odd values or even values. On the left, I’ve included a table that helps evaluate what the final chart will show. Figure 3-16 shows this table in more detail.

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Raw Value</td>
<td>Odd/Even</td>
<td>Boolean Filter</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Raw Value</td>
<td>Odd/Even</td>
<td>TRUE</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>Odd</td>
<td>TRUE</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>Even</td>
<td>FALSE</td>
</tr>
<tr>
<td>5</td>
<td>3</td>
<td>Odd</td>
<td>TRUE</td>
</tr>
<tr>
<td>6</td>
<td>4</td>
<td>Even</td>
<td>FALSE</td>
</tr>
<tr>
<td>7</td>
<td>5</td>
<td>Odd</td>
<td>TRUE</td>
</tr>
<tr>
<td>8</td>
<td>6</td>
<td>Even</td>
<td>FALSE</td>
</tr>
</tbody>
</table>

**Figure 3-16. The table that allows for chart filtering**

In column B, I use the following formula:

\[=\text{CHOOSE(MOD(A3,2)+1,"Even","Odd")}\]

So let’s break this down.

Nested inside the `CHOOSE` conditional is the `MOD( )` formula. `MOD` performs modulo division, which is a technical way of saying it performs division like a third grader. Remember when you first started learning how to divide, 3 divided by 2 would equal 1 remainder 1? Well, modulo division performs this same operation but only returns the remainder part. In the case of `MOD(A3,2)` you’re simply testing whether the list of numbers given in column A is odd or even. As you might recall, when even numbers are divided by two, there is never a remainder (think of it as a reminder of zero); for odd numbers there’s always a remainder of one.

What you run into is that you’re using the `CHOOSE( )` formula to tell Excel whether to return the word “Odd,” or to return the word “Even.” `CHOOSE( )`, however, can’t take in numbers that are less than one, and so far, it’s possible this could return a zero. So, my solution is to add the one at the end. So going back to the original `CHOOSE` formula,

\[=\text{CHOOSE(MOD(A3,2)+1,"Even","Odd")}\]

...you can see how all the parts fit together.

Moving on to Column C (Figure 3-17), you’re simply testing if the contents in Column are equal to the contents of your dropdown.

**Figure 3-17. Testing whether the contents of the boolean filter are equal to the dropdown**
This is achieved by writing the following Boolean formula from cell C3:

\[ (B3=\$L\$2) \]

The parentheses surround the test condition telling Excel to either return a TRUE or FALSE value. When there’s only one test case, the parentheses are optional. However, it’s good practice to keep parentheses anyway, keeping in line with the idea presented above that you should match your formulas to manifest the conditions you’re developing. And, specifically, note that the following two formulas are not equal:

\[ (B3=\$L\$2)+1 \quad \neq \quad =B3=\$L\$2+1 \]

Finally, in Column D you multiply columns A and C (Figure 3-18). When the number in Column A is multiplied by a TRUE value, it’s the same as multiplying it by the number one. When multiplied by a FALSE value, it’s the as multiplying it by zero. The chart is linked to column D so the outcomes in column D are automatically updated on the chart.

![Figure 3-18. The Final Value column of your table](image)

I have to admit: CHOOSE wasn’t the best function for the example above. By all accounts, if you were thinking I should have used IF instead, you wouldn’t have been off base. The values of "Even" and "Odd" aren’t ordinal. Numbers are either only even or odd. And I’m usually of the belief that the more natural the function mirrors the problem, the easier it is to comprehend. What makes the example above such a good IF problem is because the Boolean dynamic, that TRUE/FALSE = 1/0, goes both ways. Recall in your test for an even or odd value, the MOD function was returning either a zero or a one. You could have written =CHOOSE(MOD(A3,2)+1,"Even","Odd") as =IF(MOD(A3,2),"Odd","Even") which is reasonably easier to read, and it’s probably easier to comprehend when you come back to it later.

**Condensing Your Work**

What makes =IF(MOD(A3,2),"Odd","Even") so readable is because there are no nested conditions. Once you add more conditions, it becomes much harder to comprehend at first glance. And, when you represent information on your spreadsheet, you’ll sometimes have to condense formulas from different cells into one to save space. In the example above, if you want to condense your work, you can do something like this in column D:

\[ =IF(MOD(A3,2),IF(\$L\$2="Odd",A3,0),IF(\$L\$2="Even",A3,0)) \]
But now the IF function is longer and harder to understand. Maybe it’s time you dispense with the IF function altogether. But how can you recreate the same conditions without using IF? Well, you can use the exclusive-or function, XOR, like this:

\[=\text{XOR}(\$L2=“Even”, \text{MOD}(A3, 2)) \times A3\]

---

**Note** XOR is available only in Excel 2013.

---

### The Legend of XOR()-oh

Technically, XOR is not pronounced “zore,” but rather as “ex-or,” which as you’ve likely figured is shorthand for exclusive-or. So what the heck does XOR do? Well it’s a type of truth-testing conditional function. You’re probably somewhat familiar with Excel’s cousin truth functions, AND and OR.

Let’s review them first. AND tests if all the supplied conditional expressions are TRUE. If they are, AND returns TRUE. If one condition is not true, as in FALSE, AND returns FALSE. OR tests if *only one* argument is TRUE and returns TRUE when at least one conditional expression evaluates to TRUE. If all arguments passed to OR evaluate to FALSE, OR returns FALSE. Table 3-2 shows the outcomes for AND and OR formulas when supplied with only two arguments, x and y.

**Table 3-2. A Truth Table for AND and OR Functions**

<table>
<thead>
<tr>
<th>X</th>
<th>Y</th>
<th>=AND(x,y)</th>
<th>=OR(x,y)</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE</td>
<td>TRUE</td>
<td>TRUE</td>
<td>TRUE</td>
</tr>
<tr>
<td>TRUE</td>
<td>FALSE</td>
<td>FALSE</td>
<td>TRUE</td>
</tr>
<tr>
<td>FALSE</td>
<td>TRUE</td>
<td>FALSE</td>
<td>TRUE</td>
</tr>
<tr>
<td>FALSE</td>
<td>FALSE</td>
<td>FALSE</td>
<td>FALSE</td>
</tr>
</tbody>
</table>

XOR adds an extra constraint: *only one of the arguments can contain a value of TRUE*. That’s what makes it so exclusive. It’s like a club where everyone is invited but only one person is allowed to come in—and that person is you, you lucky dog! You can think of OR as being all inclusive because it does not constrain the amount of TRUE values required to return TRUE. It’s like a club that everyone can get into (but then everyone leaves because I decide to show up). The truth table is for XOR is shown in Table 3-3.

**Table 3-3. The Truth Table for XOR**

<table>
<thead>
<tr>
<th>X</th>
<th>Y</th>
<th>=XOR(x,y)</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE</td>
<td>TRUE</td>
<td>FALSE</td>
</tr>
<tr>
<td>TRUE</td>
<td>FALSE</td>
<td>TRUE</td>
</tr>
<tr>
<td>FALSE</td>
<td>TRUE</td>
<td>TRUE</td>
</tr>
<tr>
<td>FALSE</td>
<td>FALSE</td>
<td>FALSE</td>
</tr>
</tbody>
</table>
Going back to your condensed formula, let’s see how XOR() works by examining this formula:

\[ \text{=XOR($L$2="Even", MOD(A3,2))\times A3.} \]

Recall, MOD(A3,2) will return a one when A3 is odd and a zero when A3 is even. In the example above, you’re always testing if the dropdown has “Even” selected. So, let’s say A3 equals an odd value, like the number 3. Listing 3-6 shows a step-by-step evaluation when L2 is even. Listing 3-7 shows a step-by-step evaluation when L2 is odd.

Listing 3-6. Formula Evaluation When L2 Is Even

If $L$2 = "Even" then
\[ \text{=XOR($L$2="Even", MOD(A3,2))\times A3} \]
\[ \text{=XOR(TRUE, 1)\times A3} \]
\[ \text{=FALSE\times A3} \]
\[ \text{=0 \times 3} \]
\[ \text{=0} \]

Listing 3-7. Formula Evaluation When L2 Is Odd

If $L$2 = "Odd" then
\[ \text{=XOR($L$2="Even", MOD(A3,2))\times A3} \]
\[ \text{=XOR(FALSE, 1)\times A3} \]
\[ \text{=TRUE\times A3} \]
\[ \text{=1 \times 3} \]
\[ \text{=3} \]

So, think about this way: you’re actually interested in the inverse relationship between your two conditions. If $L$2 has “Even” selected, for the value in A3 to show, it must also be even. For even values, MOD(A3,2) will return a zero (which is the opposite result of the test $L$2 = "Even"). If $L$2 has “Odd” selected, the first argument will return FALSE, but MOD(A3,2) will actually return a one.

Do We Really Need IF?

For this section, I’ll combine everything you’ve learned so far to answer the question: do we really need IF? The fact is, many problems that feel like they need IF probably don’t need it. Let’s go through a few quick examples.

Need to test if a cell is blank so you can return a blank instead of a zero?

Use: --REPT(A2, LEN(A2)>1)

Instead of: IF(LEN(A2) > 1, A2, "")

Note: "--" is shorthand to convert a string into a number.

Need to return a certain range based on a dropdown select?

Just add the numbers 1, 2, 3, and 4 to the beginning of your dropdown items (see Figure 3-19).
CHAPTER 3  INTRODUCING FORMULA CONCEPTS

Figure 3-19. Adding numbers to the dropdown items can help you quickly ascertain which item was selected without using an IF statement

Use: =CHOOSE(--LEFT(A2, 1), NorthRange, EastRange, SouthRange, WestRange)
Instead of: IF(A2 = "North", NorthRange, IF(A2 = "East", EastRange,
                      IF(A2 = "South", SouthRange, WestRange))

Want to know what grade you got?

Figure 3-20 shows a grade letter calculator.

Figure 3-20. A grade calculator that uses INDEX and MATCH instead of nested IFs

Use: =INDEX($B$4:$B$8, MATCH(B1,$A$4:$A$8,1))
Instead of: = IF(B1>89,“A”,IF(B1>79,”B”, IF(B1>69,”C”,IF(B1>59,”D”,“F”))))

Need to return a -1 whenever a test condition is zero; otherwise return the value?

This example uses Figure 3-21 as an example.

Figure 3-21. You can use Boolean functions instead of IF

Use: =-NOT(A1) + A1
Instead of: =IF(A1=0, A1, 0)
The Last Word

I realize some of the material in this chapter might be new for you. And perhaps you’re not yet ready to turn your back on IF. Fair enough; although don’t expect me to use it much from here on out! The point of this chapter is to get your mind to think differently about certain problems. IF is a common convention, but the popular choice isn’t always the best. This chapter introduced you to formula concepts you’ve used many times before but might not have realized what they were or what they meant. Empowered with new knowledge, I’m confident you’ll be able to think about formulas differently.

The best formulas fit somewhere on a spectrum of performance, readability, and design simplicity. If the formula you’re using to model your problem feels like a good fit, chances are—it is. I firmly believe that formulas that are a natural fit to a problem give you that “intuitively pleasing” feeling when you look at them. If this chapter has you thinking how you might do some of your own formulas differently, then my work is done here (well, except for the other eight chapters coming your way).