2.2 Lesson Plan - Vexing VBA

Overview

In today's class, students will learn to use basic for-loops and nested for-loops in conjunction with conditionals to complete common coding challenges in VBA.

Class Objectives

- Students will understand the basic syntax of a VBA for loop
- Students will understand how to utilize for-loops in conjunction with conditionals to direct logic flow
- Students will understand the value of a nested for-loop and gain basic proficiency in their use
- Students will refine their fundamental coding skills (syntax recollection, pattern recognition, problem decomposition, and debugging)

Instructor Notes

- Today is a fun class, albeit a challenging one. In this class, students will complete a series of programming challenges in VBA. As with many classes to come, this one is exercise heavy. Today's class should feel lively as students struggle through challenges and experience many "light bulb" moments. Do your part to facilitate this atmosphere by encouraging continual dialogue within groups.

- Because today's class is heavy on exercises, your time behind the podium should feel minimal. Instead, wander the class continually and bring your "teaching" to their seats. This approach will allow you to tailor your teaching to the needs of specific students.

- As with the previous class, it is important to stress that today's exercises are very much paradigmatic of fundamental programming across languages. At times, you may hear students express frustration at VBA. In these moments, do your part to re-frame their thinking. Inform them that the real challenge isn't from VBA, but rather from the increased complexity that comes when fundamental building blocks are layered atop one another.

- Have your TAs refer to the Time Tracker to stay on track.

Sample Class Video (Highly Recommended)

- To view an example class lecture visit (Note video may not reflect latest lesson plan): Class Video

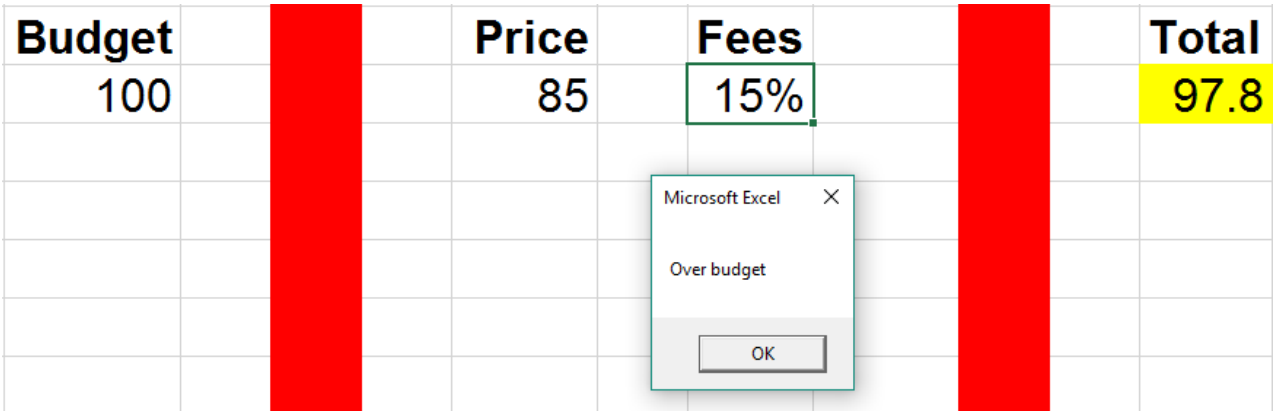
1. Instructor Do: Welcome Students (0:05)

- Spend a few moments to welcome students. Inform them that in today's class, we'll be "ripping the band-aid." They'll be spending the vast majority of class focused on exercises that will build their programming skills.

- If you'd like feel free to use the Vexing_VBA.pptx included in the lesson plan to tee off class with some warm-ups.

2. Partners Do: Warm-up Activity (Budget Checker) (0:17)

- After your welcome, direct students to the Warm-Up Activity. This activity involves them creating a script to calculate and correct a price inclusive of fees.



- Show students how the solved code should work, then have your TAs slack out the instructions and unsolved file:
  - **Files:**
    - BudgetChecker_Unsolved
  - **Instructions:**
    - Create a VBA Script to complete the budget checker.
    - There are three parts to this problem.
      - Part 1: Calculate the total after fees and enter the value in the "Total" cell.
      - Part II: Create a Message Box for the user to designate whether the amount including fees is within or over budget.
      - Part III (Challenge): If the total is over budget, correct the price such that it fits within the max of the user's budget. Be sure to round down! (Example: If the user's budget is 100 and the fees are 15%, the max price should 86)
  - **Hints:**
    - Break up the problem into smaller steps.
    - Look at old code!
    - You got this!

3. Instructor Do: Review Warm-up (Budget Checker) (0:10)

- Once time is complete, review the solution of the previous exercise. As you are doing so, encourage students to explain what is happening in the code for you.
- In your discussion of Part I, be sure to point out the following:
  - First, we created a variable called `total` (as a double) to hold our full cost.
  - Next, we combined the price and (1 + tax) to calculate the full cost.
  - Finally, we entered the final cost into the total cell.

```
Sub BudgetChecker()

    ' Part I: Calculate Total
    ' ------------------------------------------------------------

    ' Retrieve the Price and Fees from the cells
    dim total as double

    ' Use these values to calculate the total
    total = Range("F3").value * (1 + Range("H3").Value)

    ' Enter the total into the appropriate cell
    Range("L3").value  = total
```

- In your discussion of Part II, be sure to point out the following:
  - First, we created a variable called `budget` to hold our budget amount.
  - We then compared the value of our budget against the total.
  - If the budget was greater than the total, we printed under budget. If greater, we printed over budget.

```
    ' Part II: Over Budget Alert
    ' ------------------------------------------------------------

    ' Create a variable to store budget
    dim budget as double
    budget = Range("C3").value

    ' If our total is under budget...
    if budget > total then

        msgbox("Under budget")

    ' If our total is over budget...
    else

        msgbox("Over budget")
```

- In your discussion of Part III, be sure to point out the following:
  - First we needed to do some basic algebra to determine our formula for the new price. In this case new price was equal to the budget divided by (1 + tax).
  - We then stored the new price into a variable and used it with the tax to calculate the new total.
  - Finally, we changed the price and the total in the appropriate cells.

```
    ' Part III: Price Correction
    ' ------------------------------------------------------------


    ' Calculate what the alternative price should be
    dim newPrice as double
    newPrice = budget / (1 + Range("H3").value)

    ' Change the price
    Range("F3").value = newPrice

    ' Change the new total
    Range("L3").value = newPrice * (1 + Range("H3").value)
```

- Ask if there are any questions before providing students with the solution and proceeding to the next section.

4. Instructor Do: For Loop (0:07)

- Next, you'll be introducing for-loops. Open the exercise inside 02_Ins_ForLoops and walk students through the code and run the VBA script. This code, quite simply inserts a series of numbers across rows and columns.

- In explaining the code be sure to note the parts of a VBA for loop:

  - `For i = 1 to 20` specifies the range to loop through.

  - Subsequent uses of `i` change with the loop

  - `Next i` iterates to the next value of `i`.

- Spend a few extra moments in your discussion to have students guess how the spreadsheet will look before running the code. In particular, challenge them to understand why `Cells(i, 1)` creates entries across rows and `Cells(1, i)` create entries across columns.

```
' Loop through from numbers 1 through 20
For i = 1 To 20

    ' Iterate through the rows placing a value of 1 throughout
    Cells(i, 1).Value = 1

    ' Iterate through the columns placing a value of 5 throughout
    Cells(1, i).Value = 5

    ' Places increasing values based upon the variable "i" in B2 to B21
    Cells(i + 1, 2).Value = i + 1

' Call the next iteration
Next i
```

- Once you feel confident in their understanding, provide them with your code.

5. Students Do: Chicken Nugget Loop (0:10)

- Next, proceed with the next student exercise. In this example, students create a basic VBA script that prints "I will eat `i` Chicken Nuggets," where the value of `i` changes within the for loop.

- Show students what happens after the code runs, before sending them the instructions.
- **Instructions:**
  - Create a `For` loop that will produce the following example. (Note: The lines signify new cells.)

A1	B1	C1
I will eat	11	Chicken Nuggets
I will eat	12	Chicken Nuggets
I will eat	13	Chicken Nuggets
I will eat	14	Chicken Nuggets
I will eat	15	Chicken Nuggets
I will eat	16	Chicken Nuggets
I will eat	17	Chicken Nuggets
I will eat	18	Chicken Nuggets
I will eat	19	Chicken Nuggets
I will eat	20	Chicken Nuggets

  - *Bonus*
    - If you finish early, talk to your neighbor about why you may want to use a For loop over the "range" function.

6. Instructor Do: Review Chicken Nuggets Loop (0:05)

- Once the timer is done, review the exercise with students.
- As you are doing so, be sure to point out the following:
  - We created a for loop that iterates from 1 through 10.
  - We set the value of `(i, 1)` and `(i, 3)` to be fixed value of "I will eat " and "Chicken Nuggets"
  - We set the value of `(i, 2)` to be `i +10` . This forces the loop to print 11 through 20.
  - Lastly, we use `Next i` to signal we are done with the loop and onto the next one.

```
' Loop through first 10 rows
For i = 1 To 10

    ' Set values in column 1 to "I will eat"
    Cells(i, 1).Value = "I will eat "

    ' Set values in column 2 to the sum of the counter + 10
    Cells(i, 2).Value = i + 10

    ' Set values in column 3 to "Chicken Nuggets"
    Cells(i, 3).Value = "Chicken Nuggets"

' Call the next iteration
Next i
```

- Ask if there are any questions before proceeding to the next example.

7. Instructor Do: Loop Conditionals (0:10)

- Next, introduce one of the most important concepts of this week: Looped Conditionals. As a proficient developer, this will be a simple concept to you, but do not underestimate how critical it is for students to grasp.
- Start by opening the exercises on modulus contained within 04-Ins_LoopConditionals and briefly review the concept modulus to calculate the remainder. (Note that in VBA modulus is denoted by `mod`).
- Then open the exercise on looped conditionals inside 04-Ins_LoopConditionals and walk students through the code. Have them guess what it will do, before running the code.
- Then explain it line by line.
    - Start by pointing out the basic for loop structure.
    - Then introduce the concept of the modulus to determine remainder.
    - Point out that we are using if-else statements to route the flow of logic depending on whether `i` is even or odd.
    - Point out that we need to include the `end if` and also the `Next i` to close each respective block.

```
' Create a for loop from 1 to 10
For i = 1 To 10

    ' Use the modulus function to determine if a number is divisible by 2 (even number)
    If Cells(i, 1).Value Mod 2 = 0 Then

        ' Enter "Even Row" the adjacent cell
        Cells(i, 2).Value = "Even Row"

    ' If the number is not divisible by 2 (odd number)
    Else

        ' Enter "Even Row" the adjacent cell
        Cells(i, 2).Value = "Odd Row"

    ' Close the If/Else Statement
    End If

Next i
```

- Check if there are any questions before slacking out the solution.

8. Students Do: Fizz Buzz (0:20)

- Then proceed to the hallmark logic problem in coding: FizzBuzz. Inform students that this next exercise is a classic problem in technical interviews -- across all programming languages. Run the code once for them so they can see how it works.

Number	Fizz Buzz?
1	
2	
3	Fizz
4	
5	Buzz
6	Fizz
7	
8	
9	Fizz
10	Buzz
11	
12	Fizz
13	
14	
15	Fizzbuzz
16	
17	
18	Fizz
19	
20	Buzz

- Explain at a high-level, the rules of the exercise:
    - If a number is divisible by just 3, then the code should print Fizz.
    - If a number is divisible by just 5, then the code should print Buzz.
    - If a number is divisible by both 3 and 5, then the code should print FizzBuzz.
- Then send students the instructions to the exercise:
    - **Files:**
        - Fizzbuzz_Unsolved.xlsm
    - **Instructions:**
        - Create a VBA Script that populates the second column with the word "Fizz", "Buzz", or "Fizzbuzz" based on the value in the first column.
            - If the value in column 1 is a multiple of both 3 and 5, print "Fizzbuzz" in column 2.
            - If the value in column 1 is a multiple of just 3, print "Fizz" in column 2.
            - If the value in column 1 is a multiple of just 5, print "Buzz" in column 2.

9. Instructor Do: Review Fizz Buzz (0:15)

- Then walk students through the solution. As you do so, be sure to explain:
    - That we started the exercise by creating a basic for loop.
    - We then created a variable to track the value of the number in column 1.
    - We then created a series of if-then statements. We started these by checking for numbers that are both divisible by 3 and 5. It is important to start here, because if-then statements move from least specific to most specific. (i.e. If a number is divisible by 3 and 5, it is also divisible by 3. We have to make sure our code handles the more specific scenario first). Let them know that this isn't an obvious solution, but rather something that emerges as you approach your code.
    - Each of our if-then statements triggers a change to `Cells(i, 2)` (column 2).

```
' Loop through the values in Column 1
For i = 2 to 100

    num = Cells(i, 1).value

    ' Check if the number is divisible by 3 and 5....
    If (Cells(i, 1).value mod 3 = 0 AND Cells(i, 1).value mod 5 = 0) Then

        ' If so print Fizzbuzz
        Cells(i, 2).value = "Fizzbuzz"

    ' Check if the number is divisible by just 3...
    Elseif (Cells(i, 1).value mod 3 = 0) Then

        ' If so print "Fizz"
        Cells(i, 2).value = "Fizz"

    ' Check if the number is divisible by just 5...
    Elseif (Cells(i, 1).value mod 5 = 0) Then

        ' If so print "Buzz"
        Cells(i, 2).value = "Buzz"

    End If

Next i
```

- See if there are any questions before proceeding to break.

10. BREAK (0:10)

11. Partners Do: Lotto Search (0:20)

- As students return, introduce the next exercise. Warn them that it's a challenging one, but they will likely have fun with it. In this exercise, students are given a series of lotto tickets. Their task is to create a VBA script that finds these lotto winners in the list of all ticket purchases. For the bonus, they must additionally find the first instance in which any runner up appears in the list.

- Show them what the codes by running it in VBA. Point out that the winners in the Winner Table matches what you'd find if you did a `Ctrl+F` to search for the same numbers.

first_name	last_name	Lotto Number		Winner-First Name	Winner -Last Name	Winning Number
Mandi	Attewell	9573620	First	Brigid	Redwing	3957481
Claiborne	Gerardi	5751241	Second	Graeme	Readings	5865187
Marne	Berthod	9345566	Third	Valentine	Tegler	2817729
Violetta	Gillivrie	1015769	Runner Up	Micheil	Maciunas	1841402
Aldridge	Dahlman	5634498				
Yard	Stobbart	4524651				
Aloysius	Vardon	1695287				

- Then deliver the instructions:
    - **Instructions:**
        - You are in charge of finding our winners for a local lotto drawing.

        - The results are, in order:
            - First: 3957481
            - Second: 5865187
            - Third: 2817729
        - Create a script that will return those lucky winners and print them on the sheet.
            - For each winner include the following pieces of information:
                - First name
                - Last name
                - The winning number
            - They should be placed in winning order of First, Second, Third.
            - There should also be a message box that congratulates the first place winner.
    - **Bonus:**
        - There may just be one other winner! The below numbers are Wild Lotto Balls. Whichever comes up first in the list will be the fourth place (runner-up) winner. (Note: You must find the *first* runner up to appear in the list.)
            - 2275339
            - 5868182
            - 1841402

    Hints:

    - Remember to utilize variables to keep your code clean.
    - For the bonus, you may need to use `Exit For`

12. Instructor Do: Review Lotto Search (0:10)

- Once the time is complete, walk students through the solution:
    - We began by creating a series of variables to hold our ticket numbers and winner information. (Note: Because the length of the tickets is so long, we needed to use `double` or `long`. Explain to students that these are alternative formats that allow for longer numbers. These different variable types emerge for space saving considerations).

```
' Create variables to hold winners. (Use "Long"
Dim first_place As Long
Dim second_place As Long
Dim third_place As Long
Dim runner1 As Long
Dim runner2 As Long
Dim runner3 As Long

' Establish the winning ticket numbers
first_place = 3957481
second_place = 5865187
third_place = 2817729

' Establish the runner-up numbers
runner1 = 2275339
runner2 = 5868182
runner3 = 1841402
```

- Proceed to point out that we then created a for loop to scan through each of the rows. All the while, our code is searching for instances when the value in Column 3 (`Cells(i, 3)`) matches the value of our first, second, or third place winners. If there is a match we copy the winner's first name, last name, and ticket information and place them into the winners table (`Cells(2,6) - Cells(4,8)`).

```
' Loop through each of the lotto tickets
For i = 1 To 1001

    ' Check if the lotto number matches the first place winner...
    If Cells(i, 3).Value = first_place Then

        ' If so, create a message box specifying the first place win
        MsgBox " Congratulations " + Cells(i, 1).Value

        ' Retrieve the values associated with the winner and enter them into the winner's box.
        Cells(2, 6).Value = Cells(i, 1).Value
        Cells(2, 7).Value = Cells(i, 2).Value
        Cells(2, 8).Value = first_place

    ' Check if the lotto number matches the second place winner...
    ElseIf Cells(i, 3).Value = second_place Then

        ' Retrieve the values associated with the winner and enter them into the winner's box.
        Cells(3, 6).Value = Cells(i, 1).Value
        Cells(3, 7).Value = Cells(i, 2).Value
        Cells(3, 8).Value = second_place
```

- Lastly, with regards to the bonus, we needed to create a second *separate* for loop. This was necessary, because our current for loop will replace our Runner Up winner with the last instance and not the first. To avoid this, we needed to create a for-loop with an `Exit for` . This code would exit the loop the moment the first runner up is found, allowing us to avoid replacing our runner-up.

```
' Loop through the lotto tickets a second time to find the first instance of a "runner-up" winner

For i = 1 to 1001

    ' BONUS: Check for runner ups with an OR operator
    If Cells(i, 3).Value = runner1 Or Cells(i, 3).Value = runner2 Or Cells(i, 3).Value = runner3 Then

        ' Retrieve the values associated with the winner and enter them into the winner's box.
        runner_up = Cells(i, 3).Value
        Cells(5, 6).Value = Cells(i, 1).Value
        Cells(5, 7).Value = Cells(i, 2).Value
        Cells(5, 8).Value = runner_up

        ' If first match is found, exit the for loop
        Exit for

    End If

Next i
```

- Send students the final version of the code. Stress that this exercise introduced some tricky concepts, but encourage them to repeat this exercise at home.

13. Instructor Do: Nested For Loops (0:10)

- Then, proceed to the final instructor demo: Nested For Loops. Like Looped Conditionals, this is an extremely important concept and one that leaves students easily confused. Encourage them to focus in on this section.
- Open the exercise inside 07-Ins_NestedForLoops and walk students through the spreadsheet and code. Point out that in this example, we are looking to loop through both the rows and columns. Try to highlight cells as you try to simulate the actions of each loop. Begin at the top left, move across the columns, before proceeding to the next row.
- Encourage students to re-do this process themselves to one another before moving on to the next exercise.
-

```
Dim TargetStudent As String

' Loop through the rows
For i = 1 To 3

    ' Loop through the columns
    For j = 1 To 5

        ' Print the Student Name
        MsgBox ("Row: " & i & " Column: " & j & " | " & Cells(i, j).Value)

    Next j

Next i
```

14. Students Do: Hornets Nest (0:25)

- Finally, introduce the last exercise of the day: Hornets Nest. This is a fun exercise, but another challenging one. In essence the problem of this exercise is as follows:
  - You have been given a spreadsheet that's been infested by Hornets.
  - In Part I, you are responsible for counting the number of Hornets in the spreadsheet.
  - In Part II, you are responsible for replacing all instances of the word "Hornets" with the word "Bugs".

    - In Part III, you are given a set number of "Bugs" and "Bees". Each time you replace a Hornet with a Bug or Bee, you are deducting from your supply. You must replace as many hornets as you can and indicate whether all the Hornets have been replaced.

- Run the code to show students what the code does. (Note: Do not save over your file as the macro will overwrite the contents.)

Bugs	Bees	Bugs	Hornets	Bugs	Hornets	Bees						**Bugs Available**						**Bees Available**
Bees	Hornets	Bugs	Bugs	Bees	Bugs	Bees						10						5
Hornets	Bees	Bugs	Hornets	Bugs	Hornets	Bugs												
Bugs	Bugs	Hornets	Hornets	Bugs	Bugs	Bugs												
Hornets	Bugs	Bugs	Bugs	Hornets	Hornets	Bugs												
Bugs	Hornets	Hornets	Bugs	Bugs	Bugs	Bees												

- Then send the instructions and initial file:

    - **Files:**

        - HornetsNest_Unsolved

    - **Instructions:**

        - Create a VBA script to handle the growing Hornet infestation in your spreadsheet.

        - There are three parts to this problem:

            - Part I: Count the number of Hornets found and display the number to your user in the form of a message box.

            - Part II: Modify the script such that it changes the word Hornets to "Bugs".

            - Part III: Modify the script a third time, this time keeping in mind that you have a limited number of Bugs and Bees. Use the full set of Bugs and Bees you have available to replace the Hornets. If you run out of Bugs or Bees provide the user with the message: "Oh no! We still have hornets..."

    - **Hints**

        - You may want to create a backup of your spreadsheet as your macro will write over the contents.

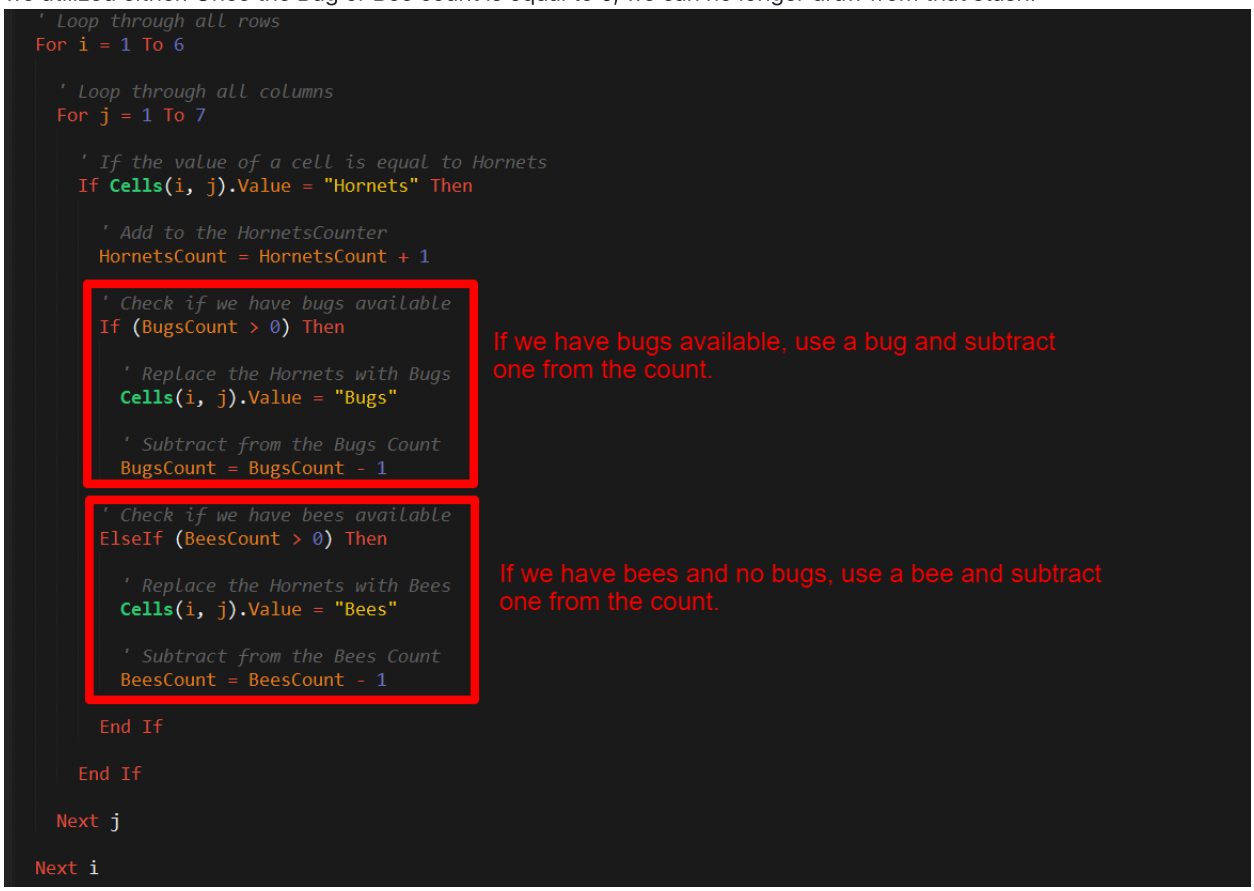
15. Instructor Do: Review Hornets Nest (0:10)

- Once the timer completes, review the solution with the group.

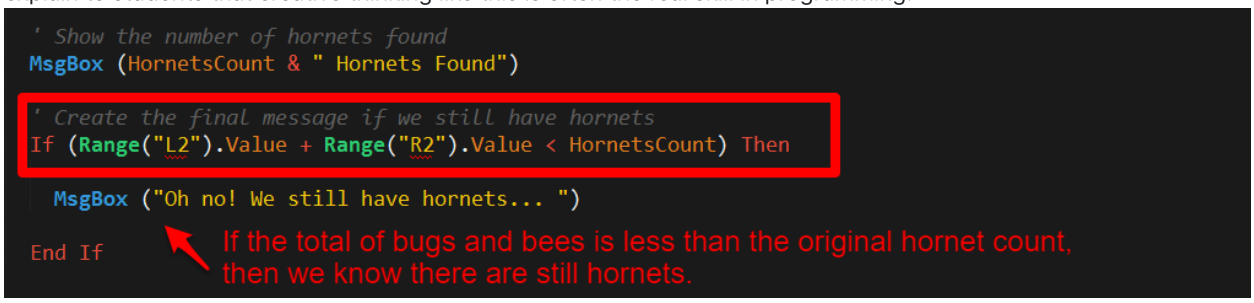
- In discussing Part 1, point out that we utilized a simple nested for loop to search for the term "Hornets" in each of the cells. In Part 2, we subsequently change the value of these cells to be "Bugs".

```
Dim HornetsCount as Integer

' Set the initial value for the HornetsCount to 0
HornetsCount = 0

' Loop through all rows
For i = 1 to 6

  ' Loop through all columns
  For j = 1 to 7

    ' If the value of a cell is equal to Hornets
    If Cells(i, j).Value = "Hornets" Then

      ' Add to the HornetsCounter
      HornetsCount = HornetsCount + 1

      ' Replace the Hornets with Bugs or Bees
      Cells(i, j).Value = "Bugs"

    End If

  Next j

Next i
```

- In Part 3, we had the added challenge of continually tracking our Bug and Bee count. The easiest way to approach this problem was to draw from our Bug stash first and then once depleted to draw from our Bee stash. In essence, this code works by storing our initial bug and bee count, then continually subtracting one from these variables each time we utilized either. Once the Bug or Bee count is equal to 0, we can no longer draw from that stash.



- For the last requirement of Part 3, we deftly avoided the for loop completely. Instead, we concluded if hornets were unaccounted for by comparing the sum of bugs and bees to the initial number of hornets. If there were more hornets than bugs and bees to start, then we know there are still hornets after our replacement efforts. Take a moment to explain to students that creative thinking like this is often the real skill in programming.



16. Instructor Do: Intro HW / Close Class (0:05)

- Spend the last few moments of class to introduce the next homework assignment. You may want to briefly explain the objective of the assignment and how it looks once complete.

Copyright