CS2034B / DH2144B

Data Analytics: Principles and Tools

Western
UNIVERSITY · CANADA

Week 7

Programming Part 2

Programming Part 2

A Note on Explicit vs. Implicit Variable Declarations

- By default VBA uses **Implicit** variable declarations.
- This means you can technically use a variable without declaring it or giving it a type. Example:

```
Function add(num1 As Integer, num2 As Integer) As Integer
    Sum = num1 + num2
    add = Sum
End Function
```

Not declared.
VBA essentially guesses the type.

A Note on Explicit vs. Implicit Variable Declarations

- By default VBA uses **Implicit** variable declarations.
- This means you can technically use a variable without declaring it or giving it a type. Example:

```
Function add(num1 As Integer, num2 As Integer) As Integer
    Sum = num1 + num2
    add = Sum
End Function
```

Still works,
=add(1,2) will output 3 in a cell.

A Note on Explicit vs. Implicit Variable Declarations

- Can make it easy to make mistakes
- Bad example:

```
Function add(num1 As Integer, num2 As Integer) As Integer
    Sum = num1 + num2
    add = Smu
End Function
```

Type-o.

But VBA will not give us an error.

Rather =add(1,2) will just output 0.

Makes it very hard to debug your code.

A Note on Explicit vs. Implicit Variable Declarations

- We can prevent this and make VBA use Explicit declarations by adding the following to the top of our modules:

```
Option Explicit
```

- Forces VBA to give an error if a variable is used but not declared.
- Strongly recommend to use `Option Explicit` in your code.

Conditional Statements

If – Then - Else

- Allow us to make a decision based on the outcome of a logical statement (Boolean logic).
- Conditional statements are best used when we have multiple paths to follow.
- Use when you want to perform parts of code only sometimes.
- **Example:** Output the string “Odd” if a number is odd, otherwise output the string “Even” if a number is even.

Conditional Statements

If – Then - Else

```
If condition_1 Then
   statements_1
ElseIf condition_2 Then
   statements_2
…
ElseIf condition_n Then
   statements_n
Else
   statements_to_do_otherwise
End If
```

Boolean expression that results in TRUE or FASE

Code to run only if *condition_1* is TRUE

Conditional Statements

If – Then - Else

```
If condition_1 Then
   statements_1
ElseIf condition_2 Then
   statements_2
…
ElseIf condition_n Then
   statements_n
Else
   statements_to_do_otherwise
End If
```

This condition is only considered if the one before it was FALSE

Code to run only if *condition_1* was FALSE and *condition_2* is TRUE

Conditional Statements

If – Then - Else

```
If condition_1 Then
   statements_1
ElseIf condition_2 Then
   statements_2
…
ElseIf condition_n Then
   statements_n
Else
   statements_to_do_otherwise
End If
```

This condition is only considered if all other conditions are FALSE

Code to run only if *conditions_1* to *conditions_n-1* are FALSE and *condition_n* is TRUE

Conditional Statements

If – Then - Else

```
If condition_1 Then
   statements_1
ElseIf condition_2 Then
   statements_2
…
ElseIf condition_n Then
   statements_n
Else
   statements_to_do_otherwise
End If
```

Else statement with no condition is matched if none of the If or ElseIf lines above were ran (had true conditions)

Code run if no other condition was matched (returned TRUE).

Conditional Statements

If – Then - Else

```
If condition_1 Then
   statements_1
ElseIf condition_2 Then
   statements_2
…
ElseIf condition_n Then
   statements_n
Else
   statements_to_do_otherwise
End If
```

End If keyword to tell VBA this is the end of the conditional statement.

Conditional Statements

If – Then - Else

- Do not have to use all parts at once (ElseIF, Else).
- The following are valid:

```
If condition Then
   statements
End If
```

```
If condition Then
     statements
Else
     other statements
End If
```

```
If condition Then
     statements
ElseIf condition Then
     other statements
End If
```

Conditional Statements

IF Example #1:

Write a function to return the string “Odd” if a number is odd, otherwise output the string “Even” if a number is even.

```
Function oddOrEven(number As Integer) As String
    If number Mod 2 = 0 Then
        oddOrEven = "Even"
    Else
        oddOrEven = "Odd"
    End If
End Function
```

Conditional Statements

IF Example #2:

Write a function that takes in the current hour as an Integer (in 24 hour format) and returns “Good Morning” if it is between 5:00 and 12:00 (inclusive), “Good Afternoon” if it is between 12:00 and 17:00 (exclusive) and “Good Evening” otherwise.

Conditional Statements

IF Example #2:

Write a function that takes in the current **hour as an Integer** (in 24 hour format) and **returns "Good Morning"** if it is between 5:00 and 12:00 (inclusive), "Good Afternoon" if it is between 12:00 and 17:00 (exclusive) and "Good Evening" otherwise.

Conditional Statements

IF Example #2:

Write a function that takes in the current **hour as an Integer** (in 24 hour format) and **returns "Good Morning"** if it is between 5:00 and 12:00 (inclusive), "Good Afternoon" if it is between 12:00 and 17:00 (exclusive) and "Good Evening" otherwise.

```
Function good(hour As Integer) As String

End Function
```

Conditional Statements

IF Example #2:

Write a function that takes in the current hour as an Integer (in 24 hour format) and returns **"Good Morning" if it is between 5:00 and 12:00 (inclusive)**, "Good Afternoon" if it is between 12:00 and 17:00 (exclusive) and "Good Evening" otherwise.

```
Function good(hour As Integer) As String
   If hour >= 5 And hour <= 12 Then
         good = "Good Morning"



   End If
End Function
```

Conditional Statements

IF Example #2:

Write a function that takes in the current hour as an Integer (in 24 hour format) and returns "Good Morning" if it is between 5:00 and 12:00 (inclusive), **"Good Afternoon" if it is between 12:00 and 17:00 (exclusive)** and "Good Evening" otherwise.

```
Function good(hour As Integer) As String
   If hour >= 5 And hour <= 12 Then
         good = "Good Morning"
   ElseIf hour > 12 And hour < 17 Then
         good = "Good Afternoon"

   End If
End Function
```

Conditional Statements

IF Example #2:

Write a function that takes in the current hour as an Integer (in 24 hour format) and returns “Good Morning” if it is between 5:00 and 12:00 (inclusive), “Good Afternoon” if it is between 12:00 and 17:00 (exclusive) and **“Good Evening” otherwise.**

```
Function good(hour As Integer) As String
   If hour >= 5 And hour <= 12 Then
         good = "Good Morning“
   ElseIf hour > 12 And hour < 17 Then
         good = "Good Afternoon"
   Else
         good = "Good Evening"
   End If
End Function
```

Conditional Statements

Exit a Function Early

- What if the hour is not valid? For example, 27 or -1.
- We could make our if statement more complicated to account for the possibility,
- Or we could do some error checking and exit the function early if the input is not valid.
- We can exit a function early using the following command:

```
Exit Function
```

Conditional Statements

IF Example #2: Updated

Update Example #2 to return “Bad Hour!” if the hour is invalid (less than 0 or greater than 23).

Method 1 (Expand IF Statement)

```
Function good(hour As Integer) As String
    If hour < 0 Or hour > 23 Then
         good = "Bad Hour!"
    ElseIf hour >= 5 And hour <= 12 Then
        good = "Good Morning“
    ElseIf hour > 12 And hour < 17 Then
         good = "Good Afternoon"
    Else
         good = "Good Evening"
    End If
End Function
```

Conditional Statements

IF Example #2: Updated

Update Example #2 to return “Bad Hour!” if the hour is invalid (less than 0 or greater than 23).

Method 2 (Exit Function)

```
Function good(hour As Integer) As String
    If hour < 0 Or hour > 23 Then
        good = "Bad Hour!"
        Exit Function
    End If

    If hour >= 5 And hour <= 12 Then
         good = "Good Morning"
    ElseIf hour > 12 And hour < 17 Then
         good = "Good Afternoon"
    Else
         good = "Good Evening"
    End If
End Function
```

Conditional Statements

IF Example #2: Updated

Update Example #2 to return “Bad Hour!” if the hour is invalid (less than 0 or greater than 23).

Method 2 (Exit Function)

```
Function good(hour As Integer) As String
    If hour < 0 Or hour > 23 Then
        good = "Bad Hour!"
        Exit Function
    End If

    If hour >= 5 And hour <= 12 Then
         good = "Good Morning"
    ElseIf hour > 12 And hour < 17 Then
         good = "Good Afternoon"
    Else
         good = "Good Evening"
    End If
End Function
```

At this point the function exits (no more lines are run) and the last value of good is returned.

Conditional Statements

IF Example #3: Real World Example

Create a function named total that takes as input an integer quantity and a double cost that represents the cost per item. The function should use the table below to return the appropriate total with the discount applied.

Quantity	Discount
0 – 24	10%
25 – 49	15%
50 – 74	20%
>= 75	25%

Example Call and Return:

`=total(26, 1.50)` **33.15**

Conditional Statements

IF Example #3: Real World Example

First we need to make the function header.

```
Function total(quantity As Integer, cost As Double) As Double
```

```
End Function
```

Conditional Statements

IF Example #3: Real World Example

We need somewhere to store the total and discount. Declare double variables to hold these values.

```
Function total(quantity As Integer, cost As Double) As Double
    Dim subtotal As Double, discount As Double
    subtotal = quantity * cost



End Function
```

Conditional Statements

IF Example #3: Real World Example

Use If statements to find the discount.

```
Function total(quantity As Integer, cost As Double) As Double
    Dim subtotal As Double, discount As Double
    subtotal = quantity * cost

    If quantity < 25 Then
        discount = 0.1 * subtotal
    ElseIf quantity >= 25 And quantity <= 49 Then
        discount = 0.15 * subtotal
    ElseIf quantity >= 50 And quantity <= 74 Then
        discount = 0.2 * subtotal
    Else
        discount = 0.25 * subtotal
    End If

End Function
```

Conditional Statements

IF Example #3: Real World Example

Calculate and return the result.

```
Function total(quantity As Integer, cost As Double) As Double
    Dim subtotal As Double, discount As Double
    subtotal = quantity * cost

    If quantity < 25 Then
        discount = 0.1 * subtotal
    ElseIf quantity >= 25 And quantity <= 49 Then
        discount = 0.15 * subtotal
    ElseIf quantity >= 50 And quantity <= 74 Then
        discount = 0.2 * subtotal
    Else
        discount = 0.25 * subtotal
    End If

    total = subtotal - discount
End Function
```

Output

- We have already see how we can return a result from a function, but what other ways can we display output to the user?
- Two additional options:

Message Boxes

Immediate Window

Output

- We have already see how we can return a result from a function, but what other ways can we display output to the user?
- Two additional options:

Message Boxes

Immediate Window

Output

Message Boxes

- Display text in a popup window to the user with an “OK” button.
- Program stops until user presses “OK”.
- Can be useful to warn the user about something but becomes annoying fast if multiple boxes are shown.

```
MsgBox "String To Show"
```

Output

Message Box Example:

Write a function that takes a name as a parameter and displays a message box containing the text “Hello <name>, welcome to VBA!” where <name> is the given name.

```
Function HelloName(name As String)
    MsgBox "Hello " & name & ", welcome to VBA!"
End Function
```

String Concatenation
Joins strings together

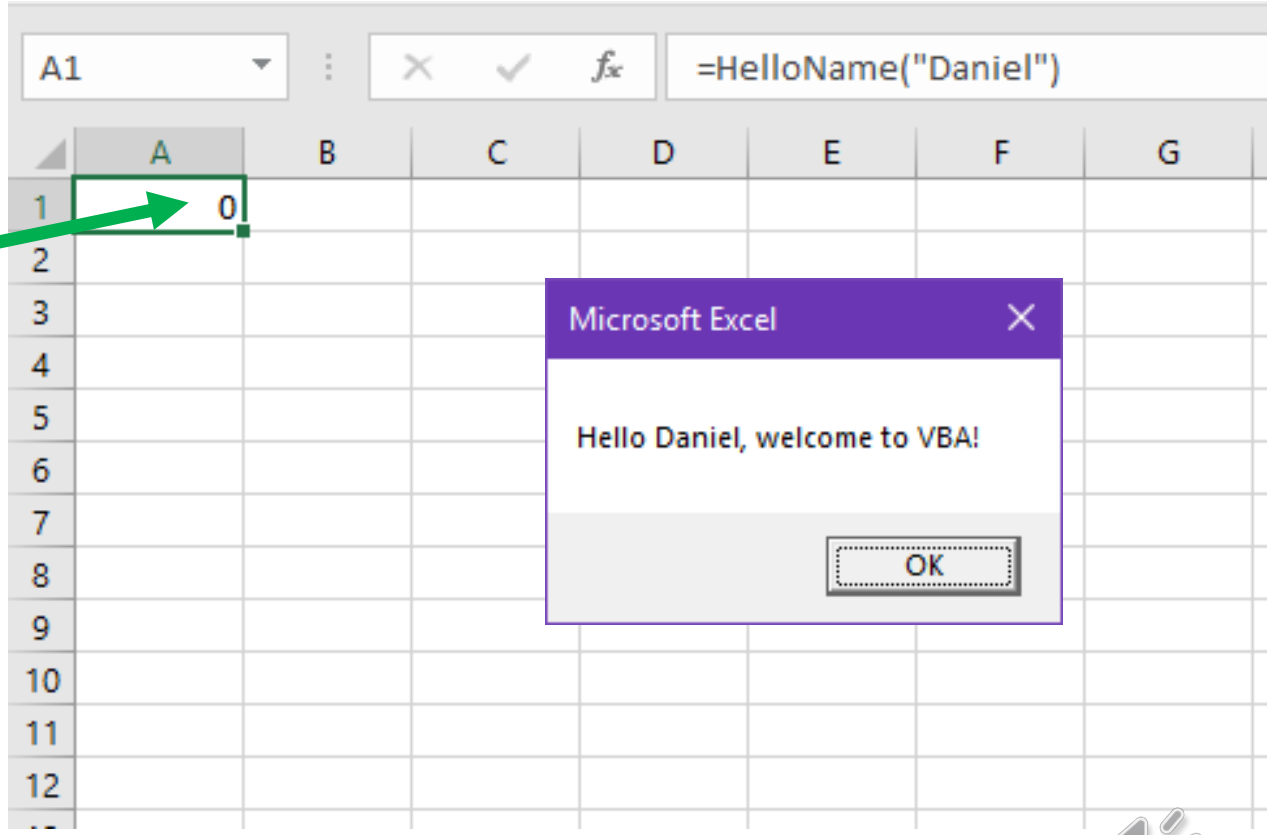
Output

Message Box Example:

Write a function that takes a name as a parameter and displays a message box containing the text “Hello <name>, welcome to VBA!” where <name> is the given name.

Function did not have return

Default is 0



Output

Immediate Window

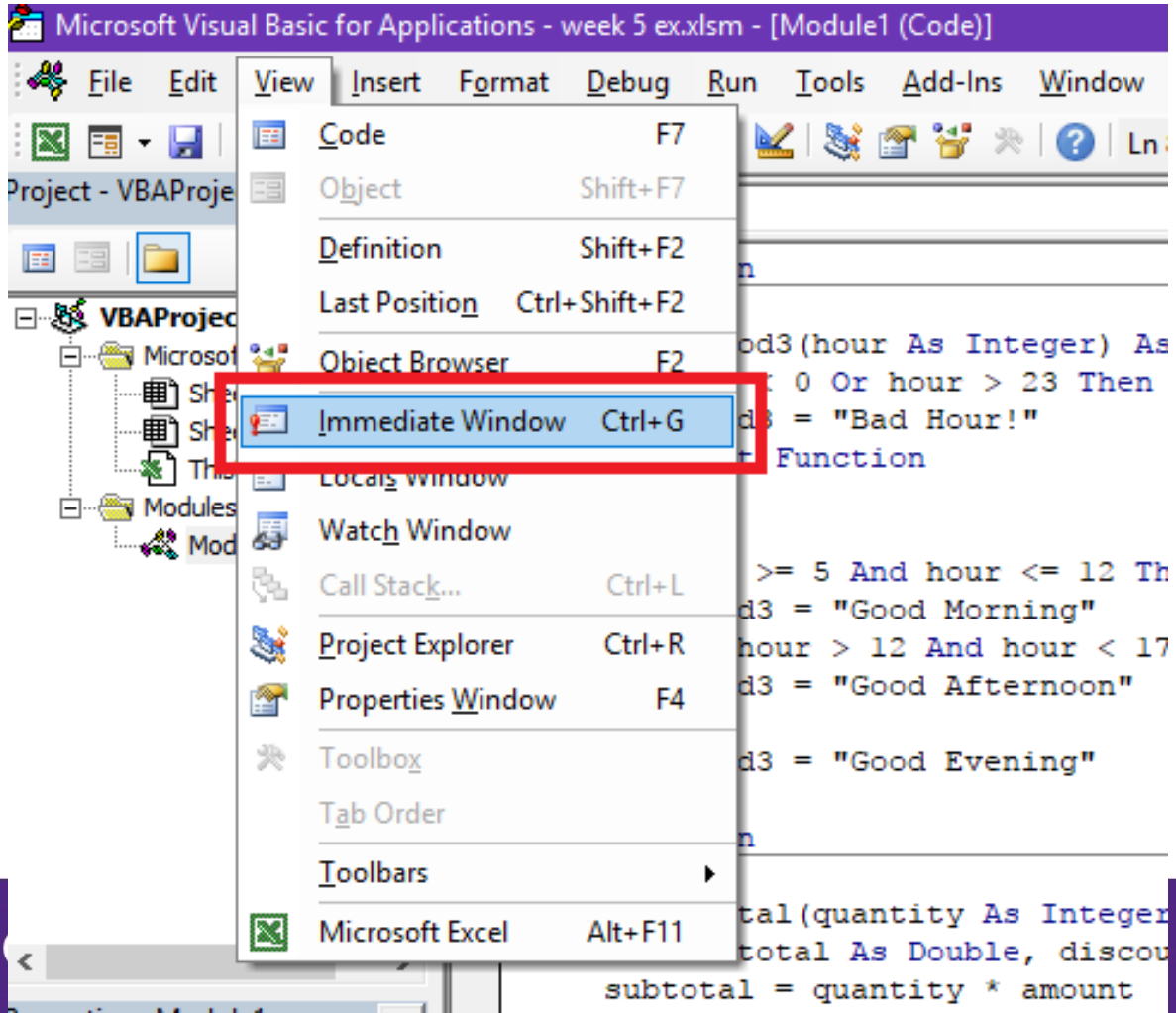
- Immediate window is a console in the VBA IDE we can print debugging messages to.
- Particularly useful while debugging code to see what the current value of a variable is or to find where your code is terminating.
- May need to expose the immediate window (not always shown by default).

```
Debug.Print "String To Show"
```

Output

Immediate Window

Expose Immediate Window in VBA IDE by pressing Ctrl-G or using the view menu:



Output

Immediate Window

Will look like the following:

Microsoft Visual Basic for Applications - week 5 ex.xlsm - [Module1 (Code)]

File Edit View Insert Format Debug Run Tools Add-Ins Window Help

Ln 9, Col 1

Project - VBAProject

VBAProject (week 5 ex.x
Microsoft Excel Objects
Sheet1 (Sheet1)
Sheet2 (Sheet2)
ThisWorkbook
Modules
Module1

Properties - Module1

Module1 Module

Alphabetic Categorized

(Name) Module1

(General) good

```
Function oddOrEven(number As Integer) As String
    If number Mod 2 = 0 Then
        oddOrEven = "Even"
    Else
        oddOrEven = "Odd"
    End If
End Function


Function good(hour As Integer) As String
    If hour >= 5 And hour <= 12 Then
        good = "Good Morning"
    ElseIf hour > 12 And hour < 17 Then
        good = "Good Afternoon"
    Else
        good = "Good Evening"
    End If
End Function
```

Immediate

Output

Immediate Window Example:

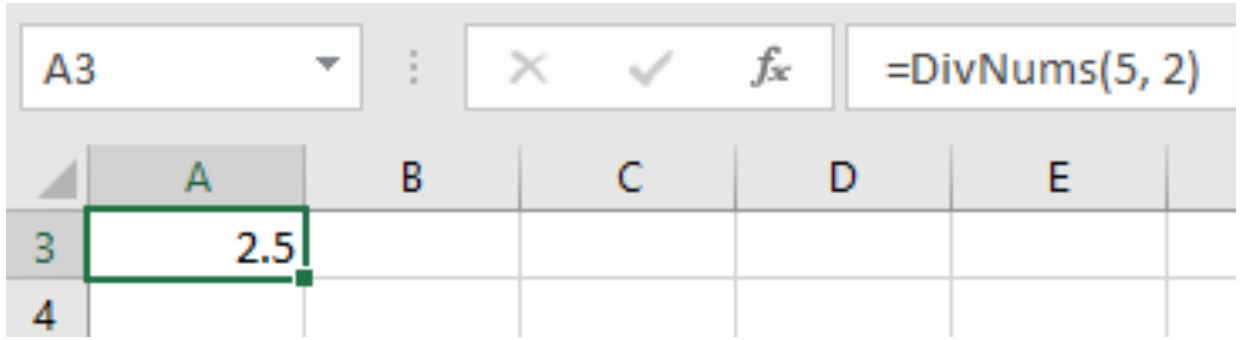
Write a function that divides two integers, use **Debug.Print** to debug your code by outputting the value of the integers and the result to the immediate window.

```
Function DivNums(n1 As Integer, n2 As Integer) As Double
    Dim result As Double
    result = n1 / n2

    Debug.Print "n1 = " & n1
    Debug.Print "n2 = " & n2
    Debug.Print "result = " & result

    DivNums = result
End Function
```

Output

Immediate Window Example:

Write a function that divides two integers, use **Debug.Print** to debug your code by outputting the value of the integers and the result to the immediate window.



Ranges and Cells

- We can access values from the worksheet directly using the Ranges and Worksheets **objects** and the Cells **property**.
- In functions these values are **read only**, we can not update the worksheet directly, only indirectly by the value the function returns.
- **Objects** are a special type of variable that contains a collection of any number of **properties** and **methods**.
- **Properties** are variables that can be any data type we have seen so far as well as other objects.
- **Methods** are functions that perform some operation on the object.

Ranges and Cells

Cells

- The Cells property is available globally, that is, it is already set up for us and we can access it in our functions without declaring it.
- `Cells(row, col)` returns the value in the active worksheet at the given row and column, where `row` and `col` are Integers.
- Example:

```
Cells(2, 2)
```

Returns the value of cell B2.

Ranges and Cells

Cells

- The Cells property is available globally, that is, it is already set up for us and we can access it in our functions without declaring it.
- `Cells(row, col)` returns the value in the active worksheet at the given row and column, where `row` and `col` are Integers.

Does not take a letter!
A is 1, B is 2, and so on.

- Example:

```
Cells(2, 2)
```

Returns the value of cell B2.

Ranges and Cells

Cells Example:

Write a function that takes a quantity of items and a cost per item as parameters and calculates the total using the tax value in cell C4.

```
Function TaxTotal(quantity As Integer, cost As Double) As Double
    Dim total As Double
    total = quantity * cost
    total = total + total * Cells(4, 3)
    TaxTotal = total
End Function
```

Ranges and Cells

Cells Example:

Write a function that takes a quantity of items and a cost per item as parameters and calculates the total using the tax value in cell C4.

```
Function TaxTotal(quantity As Integer, cost As Double) As Double
    Dim total As Double
    total = quantity * cost
    total = total + total * Cells(4, 3)
    TaxTotal = total
End Function
```

Declare variable `total` as type Double to hold the result of our calculations as we make them.

Ranges and Cells

Cells Example:

Write a function that takes a quantity of items and a cost per item as parameters and calculates the total using the tax value in cell C4.

```
Function TaxTotal(quantity As Integer, cost As Double) As Double
    Dim total As Double
    total = quantity * cost
    total = total + total * Cells(4, 3)
    TaxTotal = total
End Function
```

Find the total before tax by multiplying `quantity` by `cost` and store it in our `total` variable.

Ranges and Cells

Cells Example:

Write a function that takes a quantity of items and a cost per item as parameters and calculates the total using the tax value in cell C4.

```
Function TaxTotal(quantity As Integer, cost As Double) As Double
    Dim total As Double
    total = quantity * cost
    total = total + total * Cells(4, 3)
    TaxTotal = total
End Function
```

`Cells(4,3)` retrieves the value of the cell in the 4th row and 3rd column (cell C4). This value is multiplied by total to find the amount of tax to charge.

Ranges and Cells

Cells Example:

Write a function that takes a quantity of items and a cost per item as parameters and calculates the total using the tax value in cell C4.

```
Function TaxTotal(quantity As Integer, cost As Double) As Double
    Dim total As Double
    total = quantity * cost
    total = total + total * Cells(4, 3)
    TaxTotal = total
End Function
```

We add the amount of tax (`total * Cells(4, 3)`) to the current value of total and store the result back in the variable total (replacing its old value).

Ranges and Cells

Cells Example:

Write a function that takes a quantity of items and a cost per item as parameters and calculates the total using the tax value in cell C4.

```
Function TaxTotal(quantity As Integer, cost As Double) As Double
    Dim total As Double
    total = quantity * cost
    total = total + total * Cells(4, 3)
    TaxTotal = total
End Function
```

We return the current value of our total variable.

Ranges and Cells

Cells Example:

Write a function that takes a quantity of items and a cost per item as parameters and calculates the total using the tax value in cell C4.

E8 | =TaxTotal(D8,C8)

	A	B	C	D	E
1					
2					
3					
4		**Tax:**	0.15		
5					
6					
7		**Item**	**Cost**	**Quantity**	**Total With Tax**
8		Ballpoint Pen	1.36	25	39.1
9		Calculator	10.34	7	83.237
10		Widget	9.42	3	32.499
11		Rubber Duck	0.75	100	86.25

Ranges and Cells

Ranges

- The Range object contains a range of cells, just like a range in Excel (e.g. A2:A10).
- We can access the individual cell values using the .Cells property of the range object.
- `Range("RANGE").Cells(N)` would return the Nth cell in the range RANGE, where RANGE is an Excel range (e.g. B3:B7).
- Example:

```
Range("B3:B7").Cells(2)
```

Would return the 2nd cell in the range B3:B7, that is, cell B4.

Ranges and Cells

Ranges

- We can store Range objects in variables of type Range.
- When storing objects in variables we need to use the **Set** keyword.
- Example:

```
Dim myRange As Range
Set myRange = Range("B3:B7")
```

- We can then use this variable like a range. For example:

```
Debug.Print myRange.Cells(3)
```

Would print the value of the 3rd cell in the range (B5) to the immediate window.

Ranges and Cells

Ranges

- We can find the size of a range using the Count property.
- Example:

```
Range("B3:B7").Count
```

Returns the size of the range. In this case 5, as there are 5 cells in the range B3:B7.

Ranges and Cells

Range Example:

Write a function that takes an Integer N and returns the value of the Nth cell in range B1:G1. If N is outside of the range, return -1.

```
Function Nth(n As Integer) As Double
    Dim rng As Range
    Set rng = Range("B1:G1")

    If n < 1 Or n > rng.Count Then
        Nth = -1
    Else
        Nth = rng.Cells(n)
    End If
End Function
```

Ranges and Cells

Range Example:

Write a function that takes an Integer N and returns the value of the Nth cell in range B1:G1. If N is outside of the range, return -1.

```
Function Nth(n As Integer) As Double
    Dim rng As Range
    Set rng = Range("B1:G1")

    If n < 1 Or n > rng.Count Then
        Nth = -1
    Else
        Nth = rng.Cells(n)
    End If
End Function
```

Create a Range variable named rng to store our Range.

Ranges and Cells

Range Example:

Write a function that takes an Integer N and returns the value of the Nth cell in range B1:G1. If N is outside of the range, return -1.

```
Function Nth(n As Integer) As Double
    Dim rng As Range
    Set rng = Range("B1:G1")

    If n < 1 Or n > rng.Count Then
        Nth = -1
    Else
        Nth = rng.Cells(n)
    End If
End Function
```

Save the Range object for B1:G1 in our variable. We need to use the Set keyword as Range is an object.

Ranges and Cells

Range Example:

Write a function that takes an Integer N and returns the value of the Nth cell in range B1:G1. If N is outside of the range, return -1.

```
Function Nth(n As Integer) As Double
    Dim rng As Range
    Set rng = Range("B1:G1")

    If n < 1 Or n > rng.Count Then
        Nth = -1
    Else
        Nth = rng.Cells(n)
    End If
End Function
```

rng.Count returns the size of the range. If n is larger then this, we know it is outside of the range.

Ranges start 1 so if n is less than this, it is also outside of the range.

Ranges and Cells

Ranges As Function Parameters

- The Ranges can be given to functions as parameters just like in the built in Excel functions SUM and AVERAGE (e.g. =SUM(A1:A5) or =AVERAGE(B4:D4)).
- To take a range as a parameter we set its type as Range. For example:

```
Function MyAverage(rng As Range)
```

- We can use the parameter like we would a Range variable. For Example:

```
rng.Cells(3)
```

Would return the value of the 3rd cell in the range given as an argument to this function.

Ranges and Cells

Range Example #2:

Write a function that takes a Range of size 4 and takes the average of all values in that Range.

```
Function MyAverage(rng As Range)
    Dim sum As Double

    sum = 0
    sum = sum + rng.Cells(1)
    sum = sum + rng.Cells(2)
    sum = sum + rng.Cells(3)
    sum = sum + rng.Cells(4)

    MyAverage = sum / 4
End Function
```

Ranges and Cells

Range Example #2:

Write a function that takes a Range of size 4 and takes the average of all values in that Range.

```
Function MyAverage(rng As Range)
    Dim sum As Double

    sum = 0
    sum = sum + rng.Cells(1)
    sum = sum + rng.Cells(2)
    sum = sum + rng.Cells(3)
    sum = sum + rng.Cells(4)

    MyAverage = sum / 4
End Function
```

Note that no return type was given in the function header. If we leave it out VBA defaults to type Variant, a special type that can hold any data type.

Not recommended to do this.

Ranges and Cells

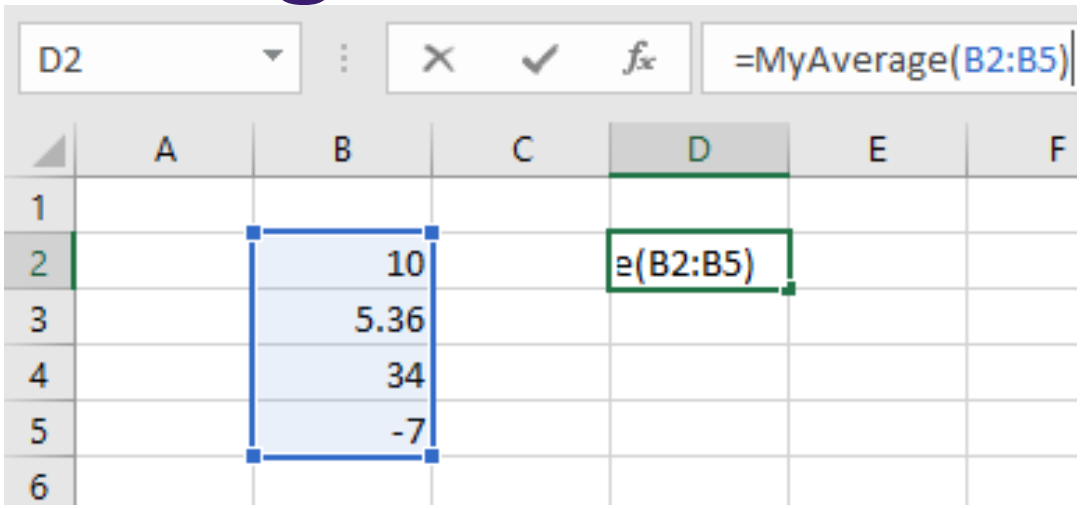
Range Example #2:

Write a function that takes a Range of size 4 and takes the average of all values in that Range.

```
Function MyAverage(rng As Range)
    Dim sum As Double

    sum = 0
    sum = sum + rng.Cells(1)
    sum = sum + rng.Cells(2)
    sum = sum + rng.Cells(3)
    sum = sum + rng.Cells(4)

    MyAverage = sum / 4
End Function
```

We don't need to declare anything for the Range or set its value as we are receiving it as an argument from Excel.

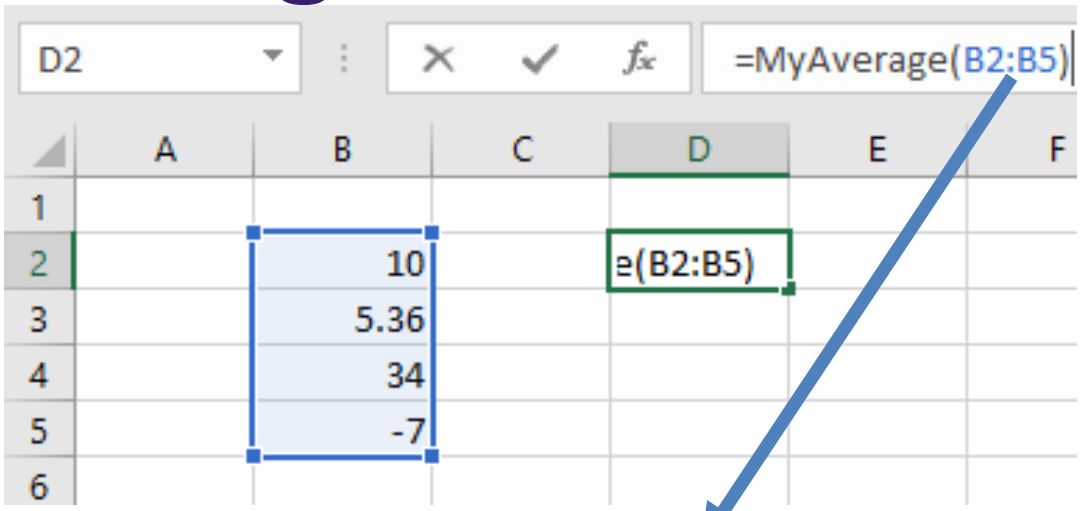
Ranges and Cells



Variable	Value
MyAverage	
rng	
sum	

```
Function MyAverage(rng As Range)
    Dim sum As Double

    sum = 0
    sum = sum + rng.Cells(1)
    sum = sum + rng.Cells(2)
    sum = sum + rng.Cells(3)
    sum = sum + rng.Cells(4)

    MyAverage = sum / 4
End Function
```

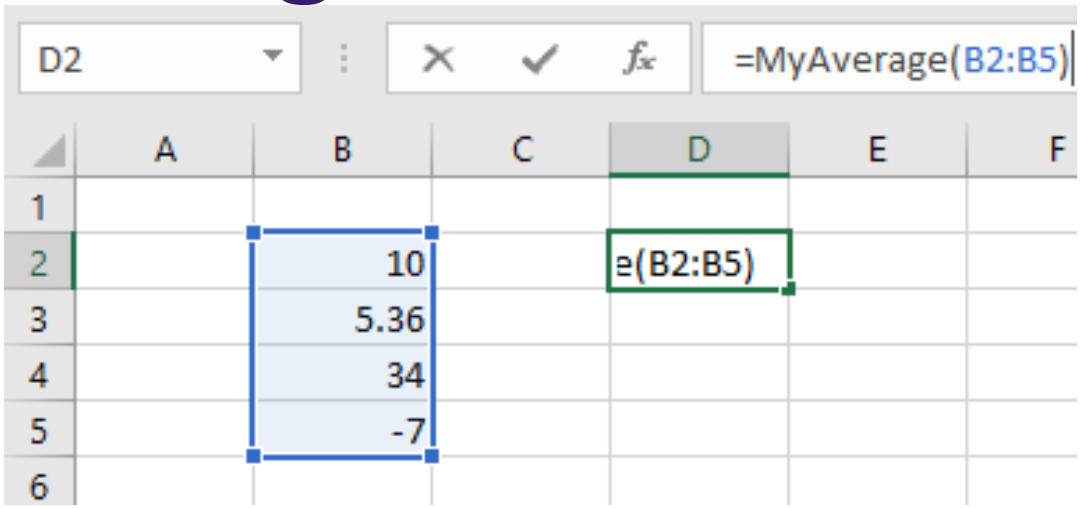
Ranges and Cells



Variable	Value
MyAverage	
rng	Range("B2:B5")
sum	

```
Function MyAverage(rng As Range)
    Dim sum As Double

    sum = 0
    sum = sum + rng.Cells(1)
    sum = sum + rng.Cells(2)
    sum = sum + rng.Cells(3)
    sum = sum + rng.Cells(4)

    MyAverage = sum / 4
End Function
```

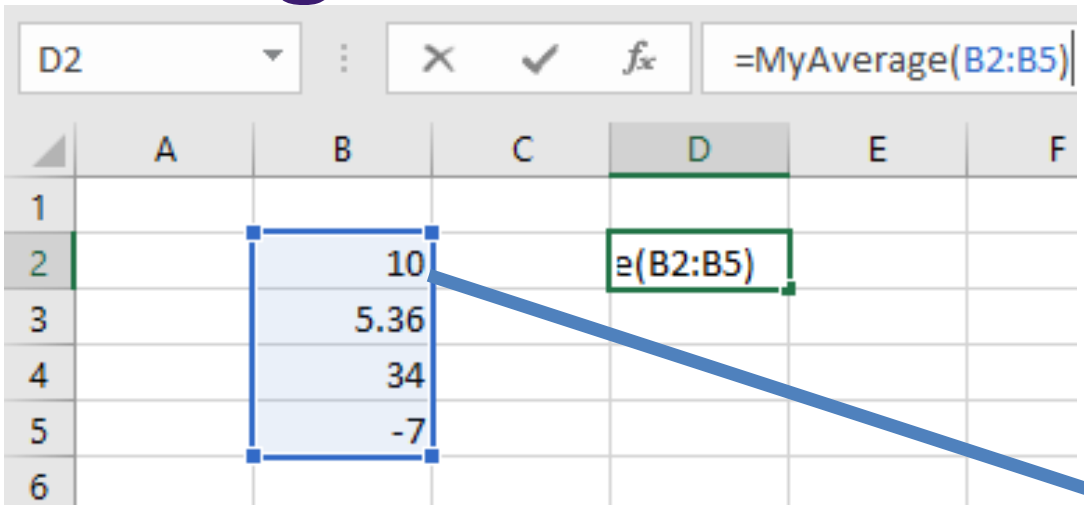
Ranges and Cells



Variable	Value
MyAverage	
rng	Range("B2:B5")
sum	0

```
Function MyAverage(rng As Range)
    Dim sum As Double

    sum = 0
    sum = sum + rng.Cells(1)
    sum = sum + rng.Cells(2)
    sum = sum + rng.Cells(3)
    sum = sum + rng.Cells(4)

    MyAverage = sum / 4
End Function
```

Ranges and Cells

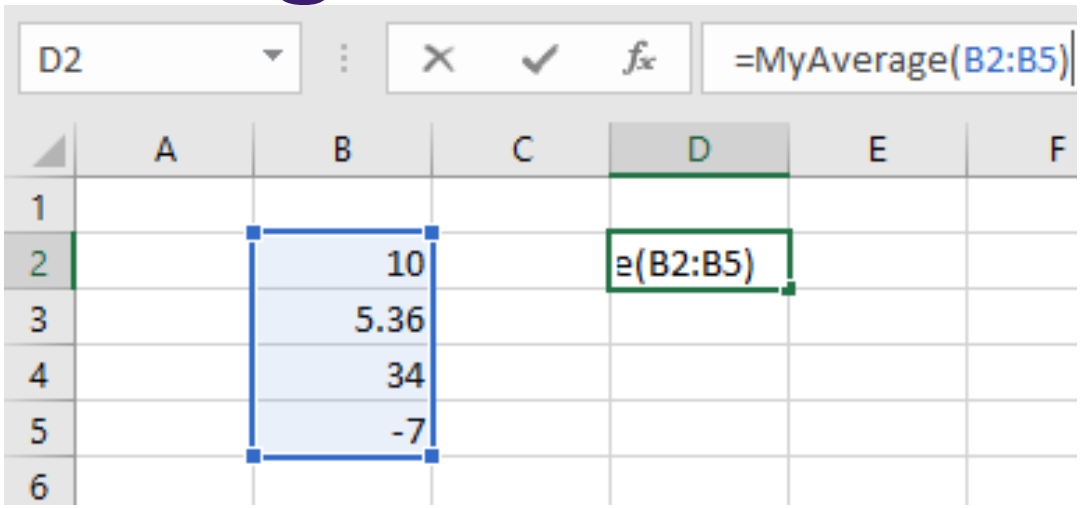


Variable	Value
MyAverage	
rng	Range("B2:B5")
sum	0

```
Function MyAverage(rng As Range)
    Dim sum As Double

    sum = 0
    sum = sum + rng.Cells(1)
    sum = sum + rng.Cells(2)
    sum = sum + rng.Cells(3)
    sum = sum + rng.Cells(4)

    MyAverage = sum / 4
End Function
```

```
sum = sum + rng.Cells(1)
```

Ranges and Cells

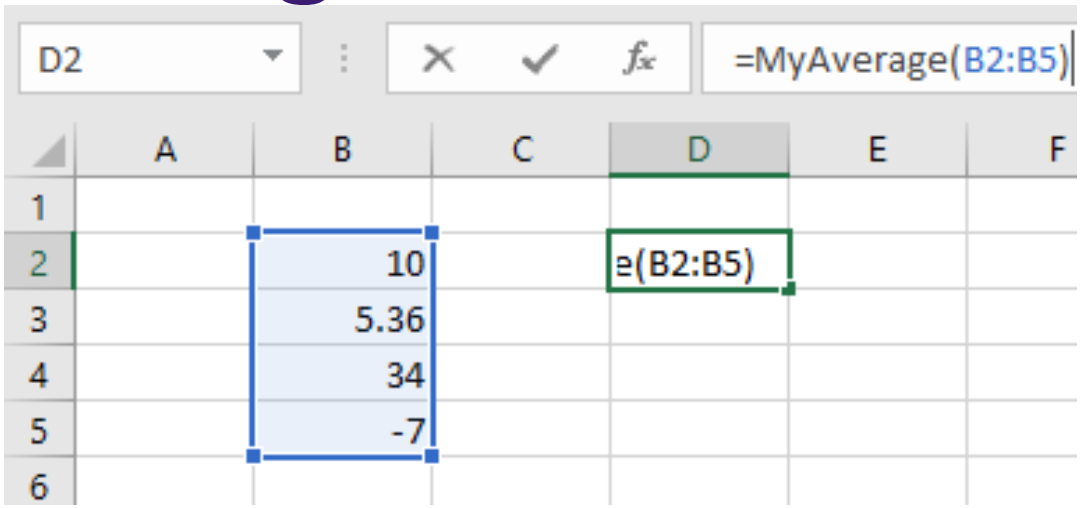


Variable	Value
MyAverage	
rng	Range("B2:B5")
sum	0

```
Function MyAverage(rng As Range)
    Dim sum As Double

    sum = 0
    sum = sum + rng.Cells(1)
    sum = sum + rng.Cells(2)
    sum = sum + rng.Cells(3)
    sum = sum + rng.Cells(4)

    MyAverage = sum / 4
End Function
```

```
sum = sum + 10
```

Ranges and Cells

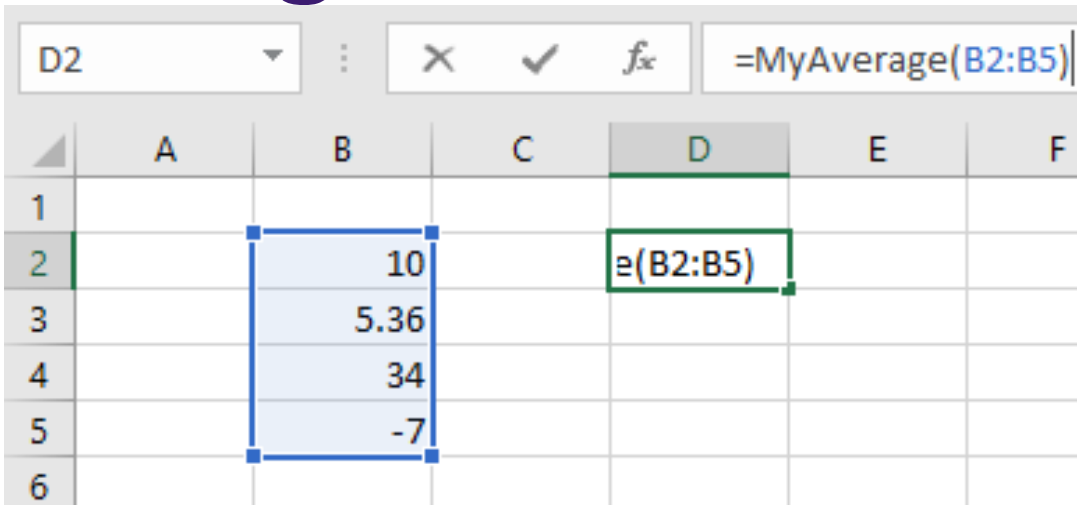


Variable	Value
MyAverage	
rng	Range("B2:B5")
sum	0

```
Function MyAverage(rng As Range)
    Dim sum As Double

    sum = 0
    sum = sum + rng.Cells(1)
    sum = sum + rng.Cells(2)
    sum = sum + rng.Cells(3)
    sum = sum + rng.Cells(4)

    MyAverage = sum / 4
End Function
```

```
sum = 0 + 10
```

Ranges and Cells

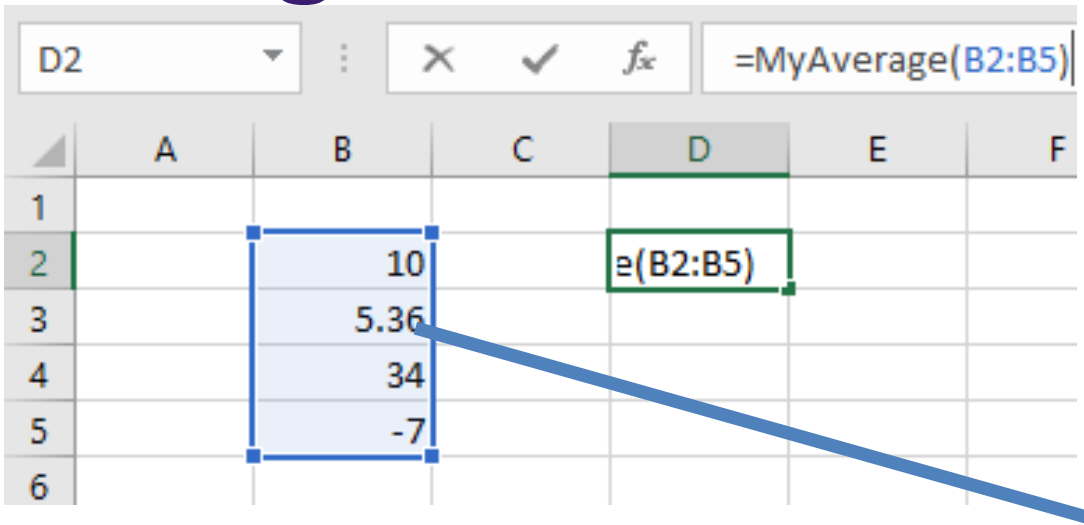


Variable	Value
MyAverage	
rng	Range("B2:B5")
sum	10

```
Function MyAverage(rng As Range)
    Dim sum As Double

    sum = 0
    sum = sum + rng.Cells(1)
    sum = sum + rng.Cells(2)
    sum = sum + rng.Cells(3)
    sum = sum + rng.Cells(4)

    MyAverage = sum / 4
End Function
```

sum = 10

Ranges and Cells

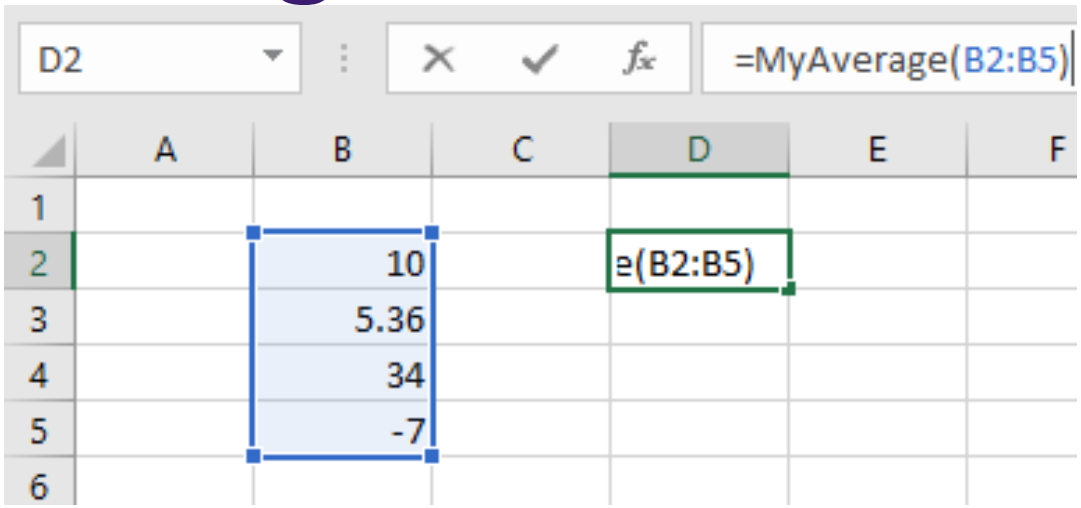


Variable	Value
MyAverage	
rng	Range("B2:B5")
sum	10

```
Function MyAverage(rng As Range)
    Dim sum As Double

    sum = 0
    sum = sum + rng.Cells(1)
    sum = sum + rng.Cells(2)
    sum = sum + rng.Cells(3)
    sum = sum + rng.Cells(4)

    MyAverage = sum / 4
End Function
```

```
sum = sum + rng.Cells(2)
```

Ranges and Cells

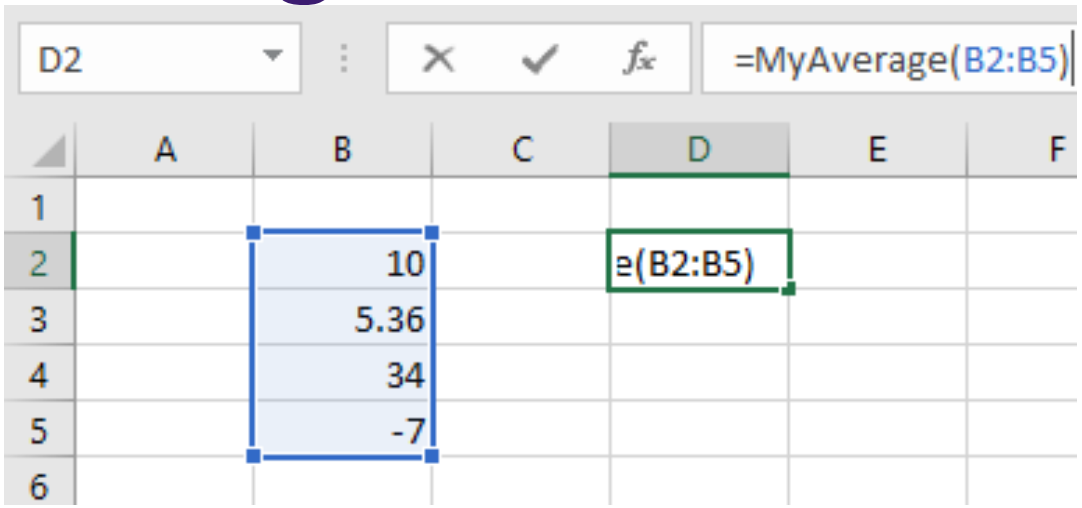


Variable	Value
MyAverage	
rng	Range("B2:B5")
sum	10

```
Function MyAverage(rng As Range)
    Dim sum As Double

    sum = 0
    sum = sum + rng.Cells(1)
    sum = sum + rng.Cells(2)
    sum = sum + rng.Cells(3)
    sum = sum + rng.Cells(4)

    MyAverage = sum / 4
End Function
```

```
sum = sum + 5.36
```

Ranges and Cells

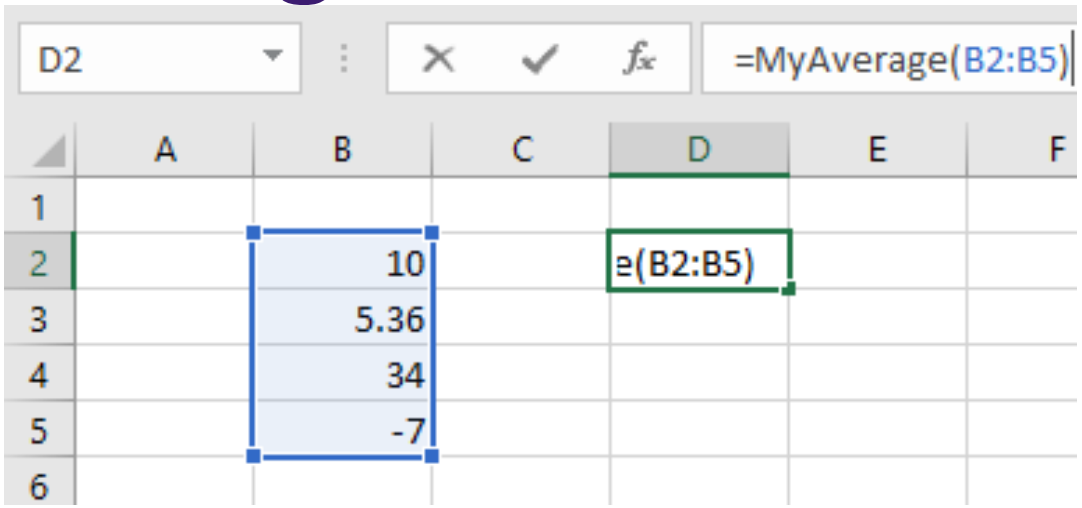


Variable	Value
MyAverage	
rng	Range("B2:B5")
sum	10

```
Function MyAverage(rng As Range)
    Dim sum As Double

    sum = 0
    sum = sum + rng.Cells(1)
    sum = sum + rng.Cells(2)
    sum = sum + rng.Cells(3)
    sum = sum + rng.Cells(4)

    MyAverage = sum / 4
End Function
```

```
sum = 10 + 5.36
```

Ranges and Cells

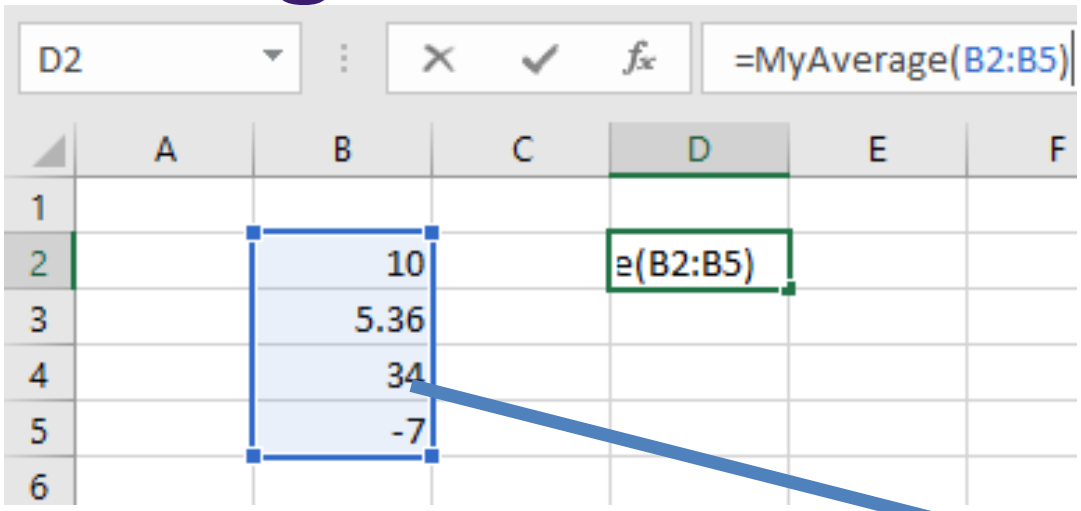


Variable	Value
MyAverage	
rng	Range("B2:B5")
sum	15.36

```
Function MyAverage(rng As Range)
    Dim sum As Double

    sum = 0
    sum = sum + rng.Cells(1)
    sum = sum + rng.Cells(2)
    sum = sum + rng.Cells(3)
    sum = sum + rng.Cells(4)

    MyAverage = sum / 4
End Function
```

sum = 15.36

Ranges and Cells

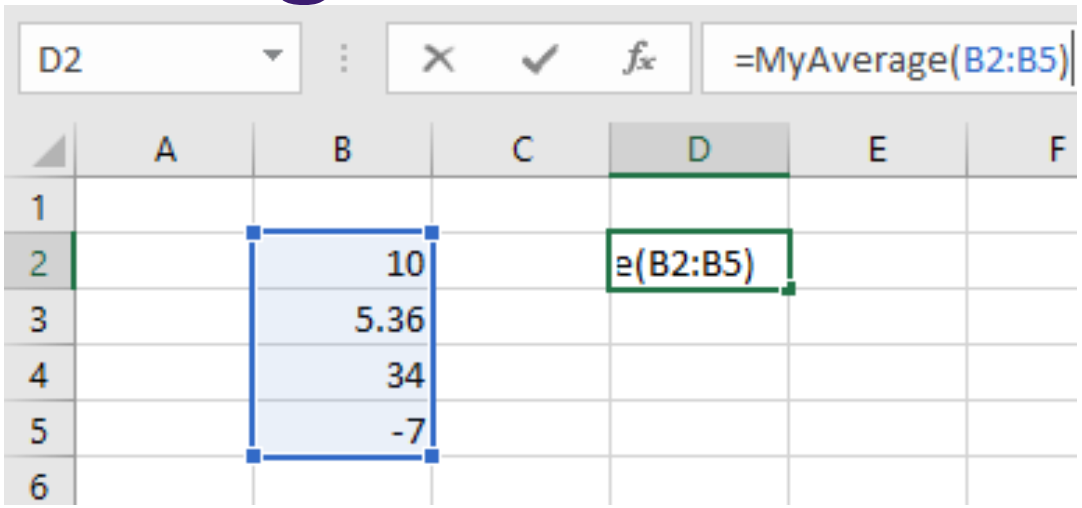


Variable	Value
MyAverage	
rng	Range("B2:B5")
sum	15.36

```
Function MyAverage(rng As Range)
    Dim sum As Double

    sum = 0
    sum = sum + rng.Cells(1)
    sum = sum + rng.Cells(2)
    sum = sum + rng.Cells(3)
    sum = sum + rng.Cells(4)

    MyAverage = sum / 4
End Function
```

```
sum = sum + rng.Cells(3)
```

Ranges and Cells

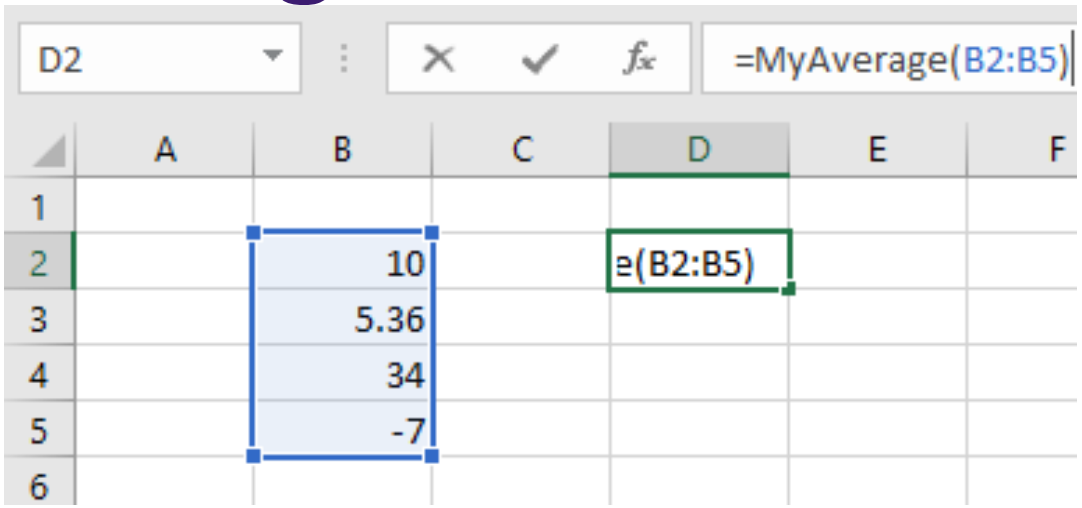


Variable	Value
MyAverage	
rng	Range("B2:B5")
sum	15.36

```
Function MyAverage(rng As Range)
    Dim sum As Double

    sum = 0
    sum = sum + rng.Cells(1)
    sum = sum + rng.Cells(2)
    sum = sum + rng.Cells(3)
    sum = sum + rng.Cells(4)

    MyAverage = sum / 4
End Function
```

```
sum = 15.36 + 34
```

Ranges and Cells

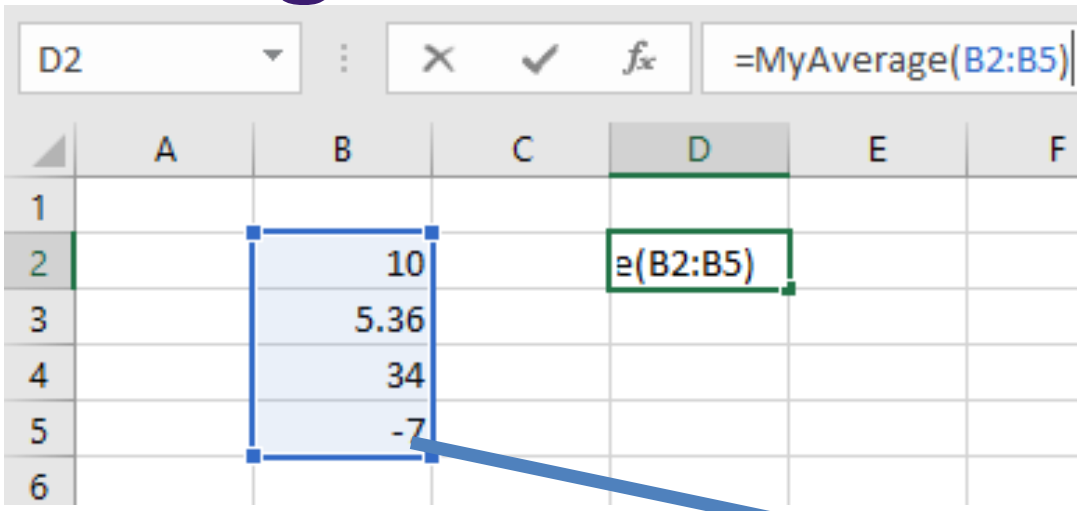


Variable	Value
MyAverage	
rng	Range("B2:B5")
sum	49.36

```
Function MyAverage(rng As Range)
    Dim sum As Double

    sum = 0
    sum = sum + rng.Cells(1)
    sum = sum + rng.Cells(2)
    sum = sum + rng.Cells(3)
    sum = sum + rng.Cells(4)

    MyAverage = sum / 4
End Function
```

sum = 49.36

Ranges and Cells

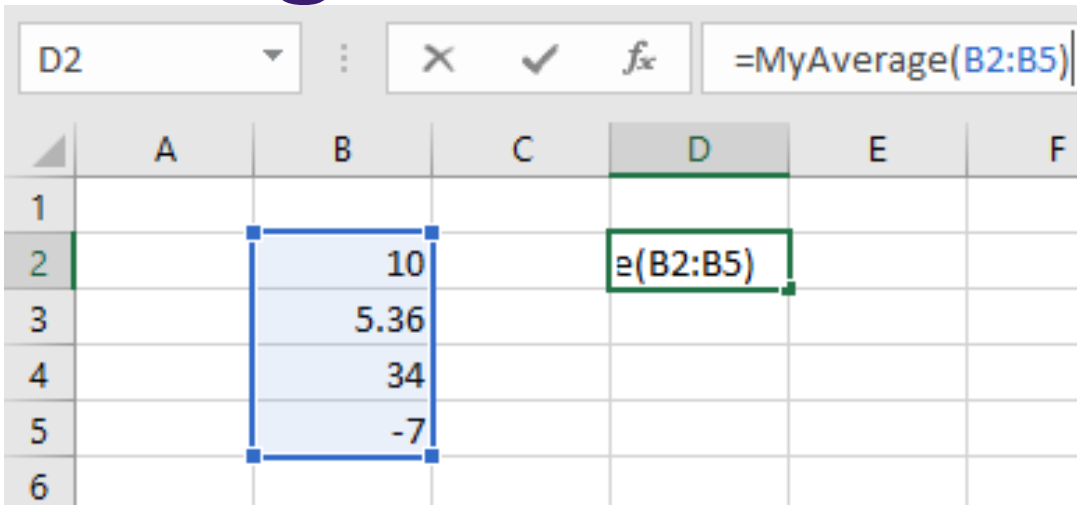


Variable	Value
MyAverage	
rng	Range("B2:B5")
sum	49.36

```
Function MyAverage(rng As Range)
    Dim sum As Double

    sum = 0
    sum = sum + rng.Cells(1)
    sum = sum + rng.Cells(2)
    sum = sum + rng.Cells(3)
    sum = sum + rng.Cells(4)

    MyAverage = sum / 4
End Function
```

```
sum = sum + rng.Cells(4)
```

Ranges and Cells

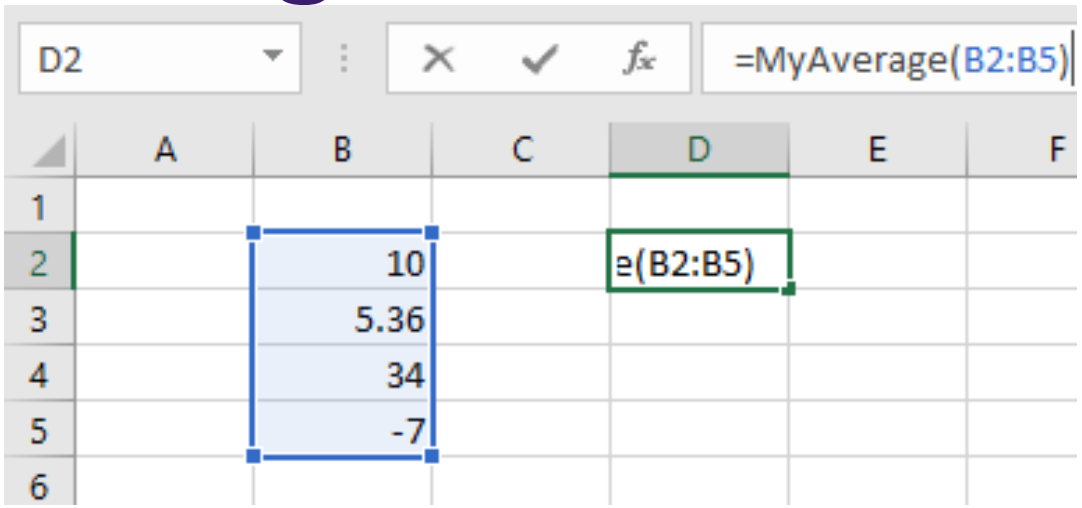


Variable	Value
MyAverage	
rng	Range("B2:B5")
sum	49.36

```
Function MyAverage(rng As Range)
    Dim sum As Double

    sum = 0
    sum = sum + rng.Cells(1)
    sum = sum + rng.Cells(2)
    sum = sum + rng.Cells(3)
    sum = sum + rng.Cells(4)

    MyAverage = sum / 4
End Function
```

```
sum = 49.36 + -7
```

Ranges and Cells

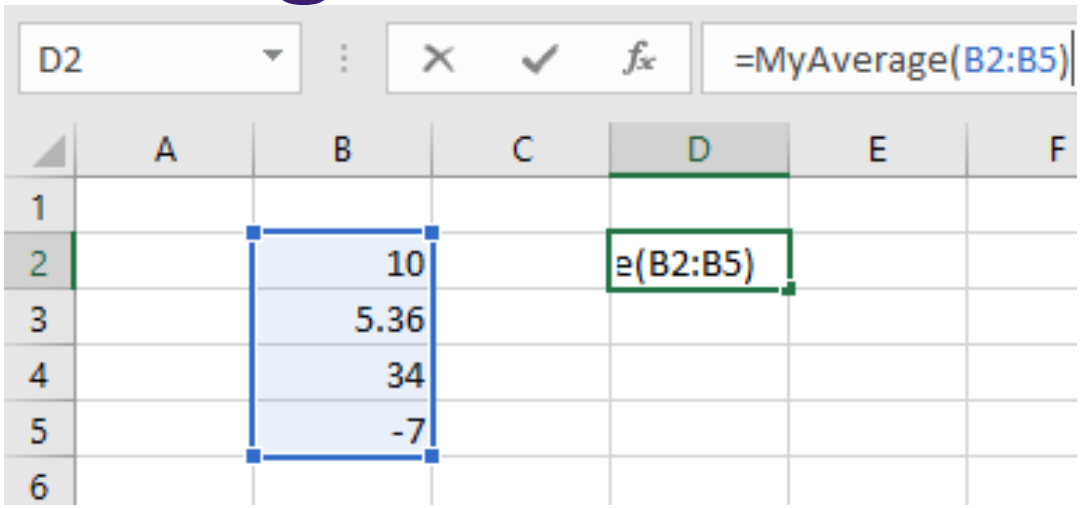


Variable	Value
MyAverage	
rng	Range("B2:B5")
sum	42.36

```
Function MyAverage(rng As Range)
    Dim sum As Double

    sum = 0
    sum = sum + rng.Cells(1)
    sum = sum + rng.Cells(2)
    sum = sum + rng.Cells(3)
    sum = sum + rng.Cells(4)

    MyAverage = sum / 4
End Function
```

sum = 42.36

Ranges and Cells

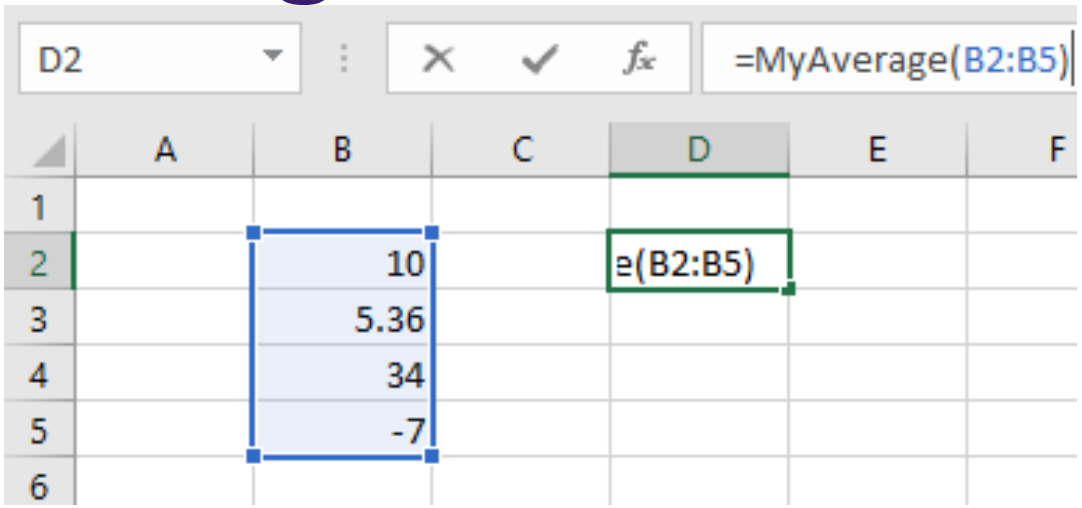


Variable	Value
MyAverage	
rng	Range("B2:B5")
sum	42.36

```
Function MyAverage(rng As Range)
    Dim sum As Double

    sum = 0
    sum = sum + rng.Cells(1)
    sum = sum + rng.Cells(2)
    sum = sum + rng.Cells(3)
    sum = sum + rng.Cells(4)

    MyAverage = sum / 4
End Function
```

```
MyAverage = sum / 4
```

Ranges and Cells

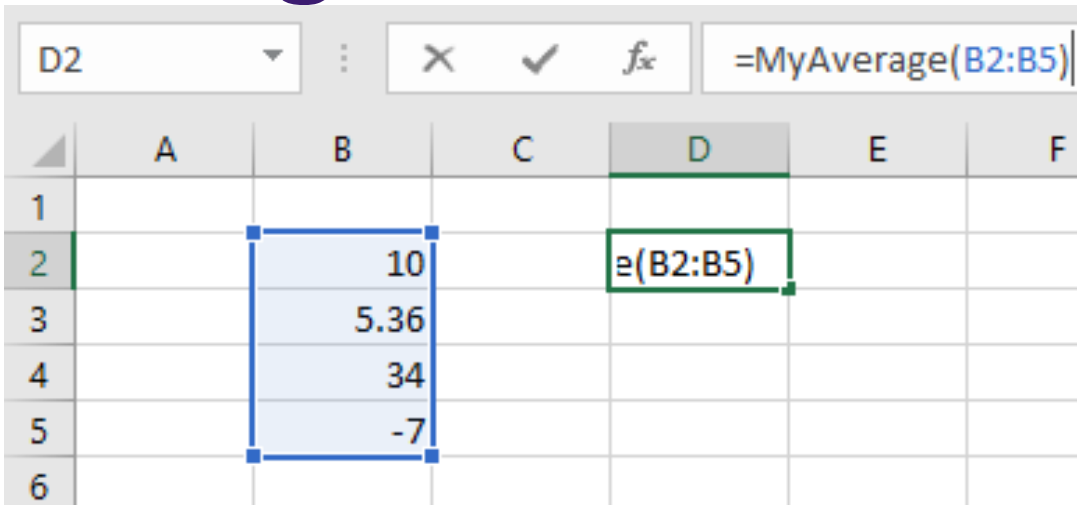


Variable	Value
MyAverage	
rng	Range("B2:B5")
sum	42.36

```
Function MyAverage(rng As Range)
    Dim sum As Double

    sum = 0
    sum = sum + rng.Cells(1)
    sum = sum + rng.Cells(2)
    sum = sum + rng.Cells(3)
    sum = sum + rng.Cells(4)

    MyAverage = sum / 4
End Function
```

```
MyAverage = 42.36 / 4
```

Ranges and Cells

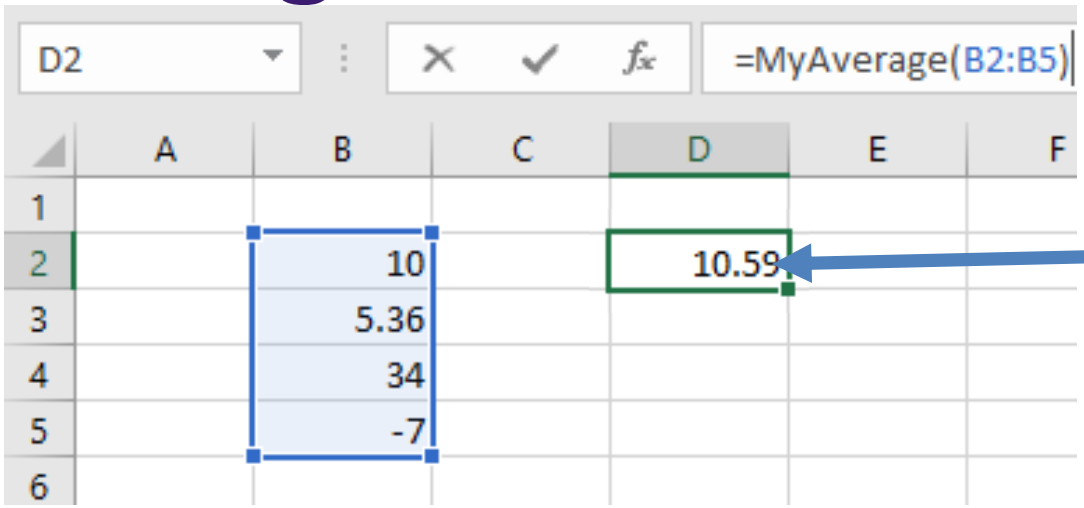


Variable	Value
MyAverage	10.59
rng	Range("B2:B5")
sum	42.36

```
Function MyAverage(rng As Range)
    Dim sum As Double

    sum = 0
    sum = sum + rng.Cells(1)
    sum = sum + rng.Cells(2)
    sum = sum + rng.Cells(3)
    sum = sum + rng.Cells(4)

    MyAverage = sum / 4
End Function
```

MyAverage = 10.59

Ranges and Cells



Variable	Value
MyAverage	10.59
rng	Range("B2:B5")
sum	42.36

```
Function MyAverage(rng As Range)
    Dim sum As Double

    sum = 0
    sum = sum + rng.Cells(1)
    sum = sum + rng.Cells(2)
    sum = sum + rng.Cells(3)
    sum = sum + rng.Cells(4)

    MyAverage = sum / 4
End Function
```

Loop Statements

- How could we make this function work for a Range of any size?
- How can we avoid typing a similar line like:

```
sum = sum + rng.Cells(1)
```

  over and over?
- **Answer:** Loops
- Loops allow us to repeat a section of code multiple times.
- Allow us to reuse code and deal with cases where we might not know how many times we need to run an operation ahead of time.

Loop Statements

For Loop

For loops run the statements they contain a set number of times well keeping track of the iteration they are on in a counter variable.

```
FOR counter = start TO end [Step increment]
            ...statements...
NEXT counter
```

Loop Statements

For Loop

Variable to use as a counter

Added to by set increment each iteration of the loop

```
FOR counter = start TO end [Step increment]
            ...statements...
NEXT counter
```

Loop Statements

For Loop

The value to start the counter at

```
FOR counter = start TO end [Step increment]
            ...statements...
NEXT counter
```

Loop Statements

For Loop

When the counter is larger than this value the loop will stop running

```
FOR counter = start TO end [Step increment]
            ...statements...
NEXT counter
```

Loop Statements

For Loop

```
FOR counter = start TO end [Step increment]
            ...statements...
NEXT counter
```

The statements to run in the loop

Loop Statements

For Loop

```
FOR counter = start TO end [Step increment]
            ...statements...
NEXT counter
```

How much to increment the counter by each iteration of the loop

This is an optional part of the syntax, by default the counter is incremented by 1.

Loop Statements

Simple Examples

```
Dim i As Integer
For i = 1 To 5
    Debug.Print i
Next i
```

```
Dim i As Integer
For i = 5 To 1 Step -1
    Debug.Print i
Next i
```

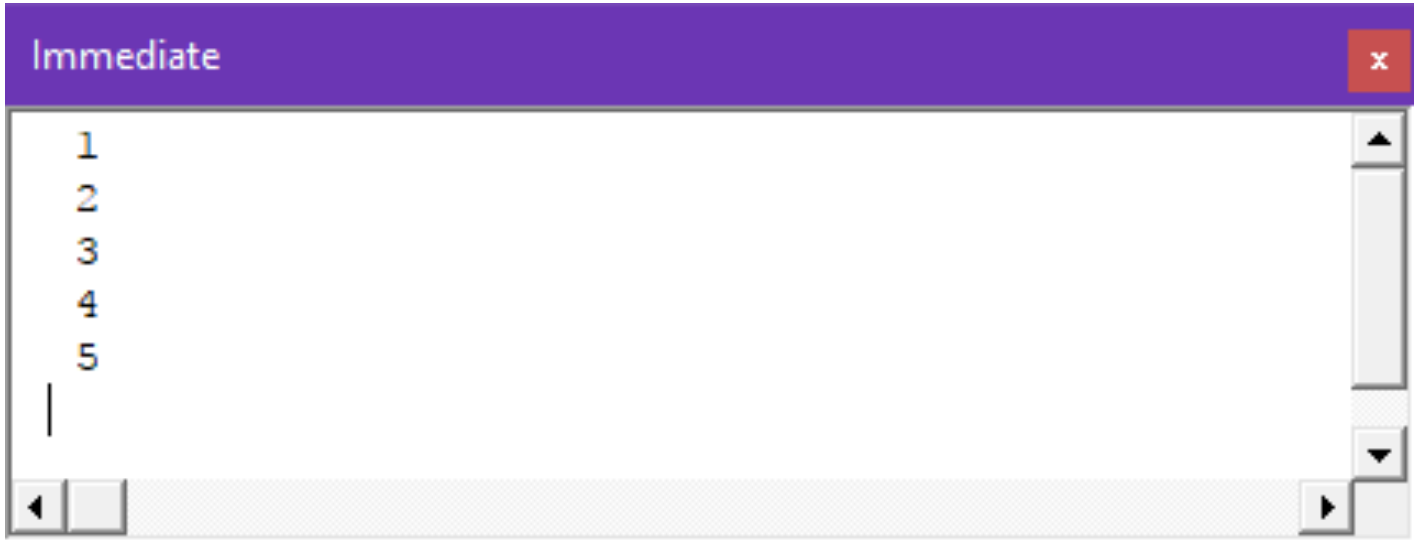
```
Dim i As Integer
For i = 1 To 12 Step 3
    Debug.Print i
Next i
```

```
Dim d As Double
For d = 0.5 To 1 Step 0.1
    Debug.Print d
Next d
```

Loop Statements

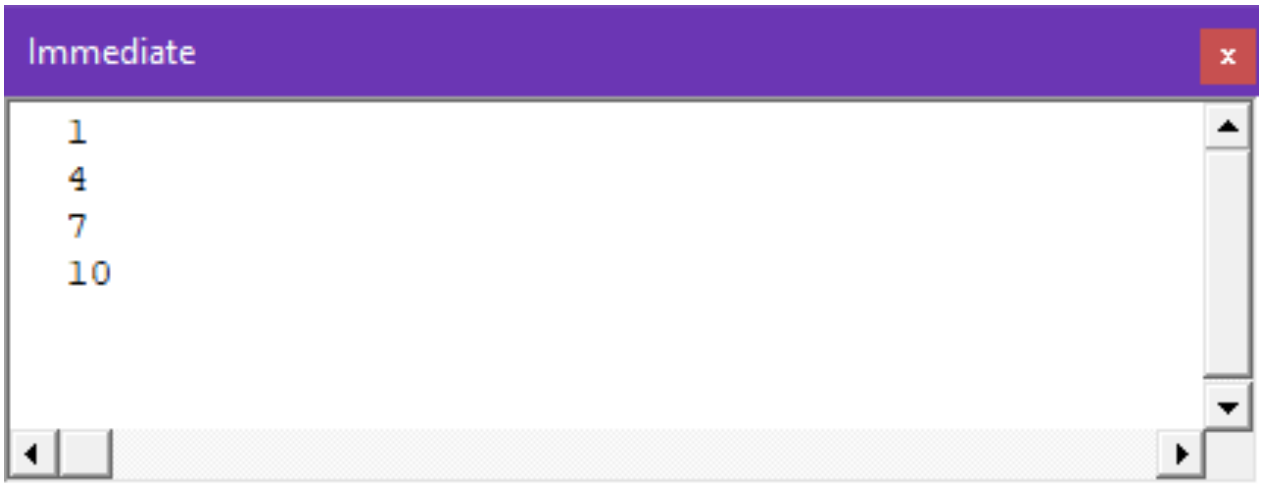
Simple Examples

```
Dim i As Integer
For i = 1 To 5
    Debug.Print i
Next i
```



Loop Statements

Simple Examples

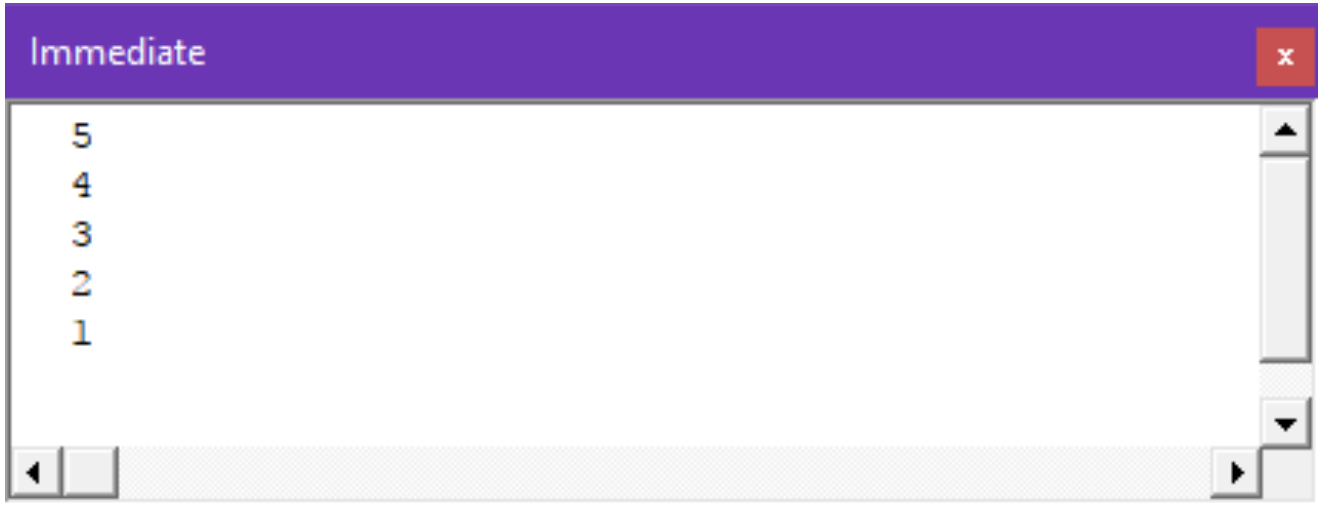



```
Dim i As Integer
For i = 1 To 12 Step 3
    Debug.Print i
Next i
```

Loop Statements

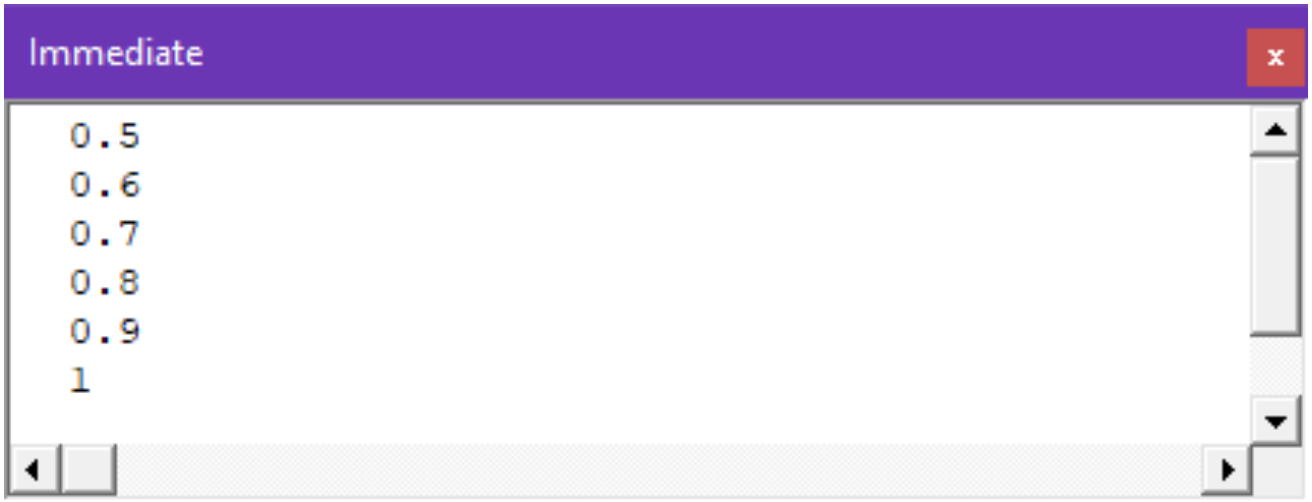
Simple Examples

```
Dim i As Integer
For i = 5 To 1 Step -1
    Debug.Print i
Next i
```



Loop Statements

Simple Examples




```
Dim d As Double
For d = 0.5 To 1 Step 0.1
    Debug.Print d
Next d
```

Loop Statements

For Loop

- How can we use a for loop to update our MyAverage function to work with any size of Range?
- We know the first cell in a Range is always 1, we could start our counter at 1.
- We know we can find the size of a Range with Range.Count, we could use this to tell how many times our loop should run (once for each cell).
- We could use the index (counter) of the loop with Range.Cells to get the current cell in the range.

Loop Statements

For Loop Example: Update MyAverage

```
Function MyAverage2(rng As Range)
    Dim sum As Double



End Function
```

Loop Statements

For Loop Example: Update MyAverage

```
Function MyAverage2(rng As Range)
    Dim sum As Double
    Dim i As Integer
```

Counter for our loop.

```
End Function
```

Loop Statements

For Loop Example: Update MyAverage

```
Function MyAverage2(rng As Range)
    Dim sum As Double
    Dim i As Integer
    sum = 0
    For i = 1
```

Start counter (index) at 1 as Ranges start at 1.

```
End Function
```

Loop Statements

For Loop Example: Update MyAverage

```
Function MyAverage2(rng As Range)
    Dim sum As Double
    Dim i As Integer
    sum = 0
    For i = 1 To rng.Count


End Function
```

rng.Count returns the number of cells in the range. We want to run our loop once for each cell to add it to our sum.

Loop Statements

For Loop Example: Update MyAverage

```
Function MyAverage2(rng As Range)
    Dim sum As Double
    Dim i As Integer
    sum = 0
    For i = 1 To rng.Count
        sum = sum + rng.Cells(i)

End Function
```

We can access the value of the ith cell using the Cells property of the range. As the value of i is incremented each loop, this will always return the value of the next cell.

Loop Statements

For Loop Example: Update MyAverage

```
Function MyAverage2(rng As Range)
    Dim sum As Double
    Dim i As Integer
    sum = 0
    For i = 1 To rng.Count
        sum = sum + rng.Cells(i)
    Next i
```

End our loop.

```
End Function
```

Loop Statements

For Loop Example: Update MyAverage

```
Function MyAverage2(rng As Range)
    Dim sum As Double
    Dim i As Integer
    sum = 0
    For i = 1 To rng.Count
        sum = sum + rng.Cells(i)
    Next i

    MyAverage2 = sum / rng.Count

End Function
```

Calculate the average using the value of sum and rng.Count. We use rng.Count again here to get the number of cells in the range.

Comments

- Lines and text that are ignored by VBA.
- Allow us to add documentation and notes to our code.
- Also useful for debugging and temporarily preventing line from being run without deleting it.
- Comments start with a single quote:

```
' My comment text
```

Comments

Example 1:

```
' This module contains my functions for finding
' the largest and smallest number

' This function takes two integers and returns the largest one
Function FindMax(n1 As Integer, n2 As Integer)

    ' If n1 is less than n2, we know it is the max and
    ' we should return it.
    If n1 > n2 Then
        FindMax = n1 ' Return n1

    ' Otherwise n1 is less than n2 or it is equal.
    ' If they are equal it does not mater which we return!
    Else
        FindMax = n2 ' Return n2
    End If
End Function
```

Comments

Example 1:

```
' This module contains my functions for finding
' the largest and smallest number

' This function takes two in                    e
Function FindMax(n1 As Integ

    ' If n1 is less than n2,
    ' we should return it.
    If n1 > n2 Then
        FindMax = n1 ' Return n1

    ' Otherwise n1 is less than n2 or it is equal.
    ' If they are equal it does not mater which we return!
    Else
        FindMax = n2 ' Return n2
    End If
End Function
```

If a comment spans multiple lines, it must have a ' at the start of each line.

Comments

Example 1:

```
' This module contains my functions for finding
' the largest and smallest number

' This function takes two in                    e
Function FindMax(n1 As Integ

    ' If n1 is less than n2, we know it is the max and
    ' we should return it.
    If n1 > n2 Then
        FindMax = n1 ' Return n1

    ' Otherwise n1 is less than n2 or it is equal.
    ' If they are equal it does not mater which we return!
    Else
        FindMax = n2 ' Return n2
    End If
End Function
```

Comments can be on their own line or after a command

Comments

Example 1:

```
' This module contains my functions for finding
' the largest and smallest number

' This function takes two integers and returns the largest one
Function FindMax(n1 As Integer, n2 As Integer)

    ' If n1 is less than n2
    ' we should return it.
    If n1 > n2 Then
        FindMax = n1 ' Return n1

    ' Otherwise n1 is less than n2 or it is equal.
    ' If they are equal it does not mater which we return!
    Else
        FindMax = n2 ' Return n2
    End If
End Function
```

Each module should contain a comment at the top describing what the module contains.

Comments

Example 1:

```
' This module contains my functions for finding
' the largest and smallest number

' This function takes two integers and returns the largest one
Function FindMax(n1 As Integer, n2 As Integer)

    ' If n1 is less than n2, we know it is the max and
    ' we should return it.
    If n1 > n2 Then
        FindMax = n1 ' Retu

    ' Otherwise n1 is less
    ' If they are equal it
    Else
        FindMax = n2 ' Return n2
    End If
End Function
```

Each function or subroutine should have a comment that describes what the function does, what arguments it takes and what the function returns.

Comments

Example 1:

```
' This module contains my functions for finding
' the largest and smallest number

' This function takes two integers and returns the largest one
Function FindMax(n1 As Integer, n2 As Integer)

    ' If n1 is less than n2, we know it is the max and
    ' we should return it.
    If n1 > n2 Then
        FindMax = n1 ' Return n1

    ' Otherwise n1 is less than n2 or it is equal.
    ' If they are equal it does not mater which we return!
    Else
        FindMax = n2 ' Return n2
    End If
End Function
```

Any lines that may not be clear to the reader should have a comment to explain them.

Comments

Example 1:

You are required to document your code using comments for assignments but not exams.

```
' This module contains
' the largest and smal

' This function takes two integers and returns the largest one
Function FindMax(n1 As Integer, n2 As Integer)

    ' If n1 is less than n2, we know it is the max and
    ' we should return it.
    If n1 > n2 Then
        FindMax = n1 ' Return n1

    ' Otherwise n1 is less than n2 or it is equal.
    ' If they are equal it does not mater which we return!
    Else
        FindMax = n2 ' Return n2
    End If
End Function
```

Comments

Example 2:

If we want to test what our function would do if we temporarily removed a line, we could use a comment.

```
Function MyAverage(rng As Range)
    Dim i As Integer, sum As Double
    sum = 0

    For i = 1 To rng.Count
        sum = sum + rng.Cells(i)
    Next i

    'sum = sum / rng.Count

    MyAverage = sum
End Function
```

Comments

Example 2:

```
Function MyAverage(rng As Range)
    Dim i As Integer, sum As Double
    sum = 0

    For i = 1 To r
        sum = sum
    Next i

    'sum = sum / rng.Count

    MyAverage = sum
End Function
```

VBA now ignores this line and the function would return the sum of the range (as opposed to the average).