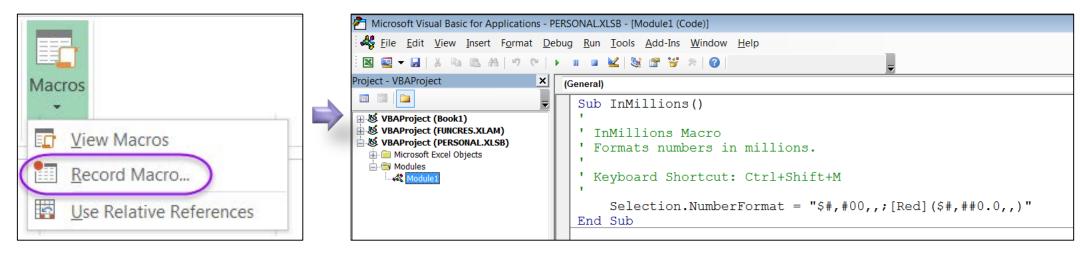
## **Excel Macro Record and VBA Editor**

Presented by Wayne Wilmeth

## What Is a Macro?

- Automates Repetitive Tasks
- Written in Visual Basic for Applications (VBA) Code



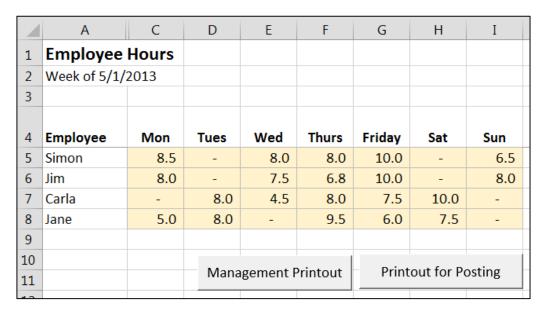
Macro Recorder

VBA Editor (Alt + F11)

# **Executing Macros**



**Keyboard Shortcuts** 



**Macro Buttons** 



Menu Buttons

# Macro Storage Locations

 Personal Macro Workbook (Personal.xlsb)

Runs in Any File



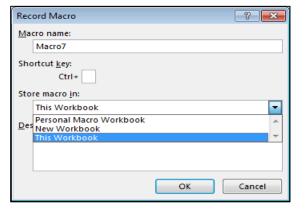
(Necessary when activation is in the menu)

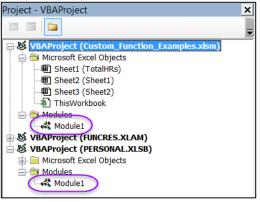
This Workbook/ New Workbook

Runs in File it is Saved to Only



(Useful when you will be emailing the file)





## The Macro Recorder

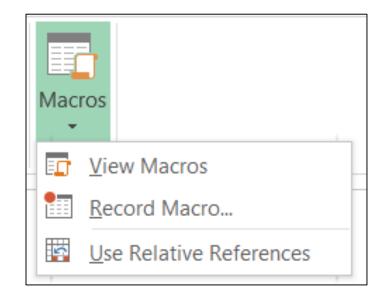
#### Advantage

Creates VBA Code for You

#### Disadvantages

- Does Not Always Record Actions as Desired
- Cannot Record Conditionally
- Cannot Loop

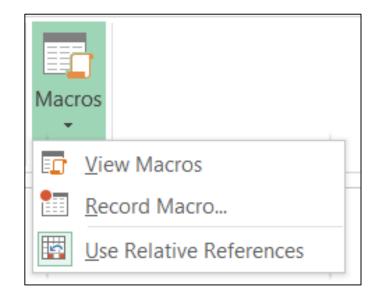
## Recorder: Absolute or Relative?



#### **Absolute**

Plays back in same cells originally recorded in.

OR

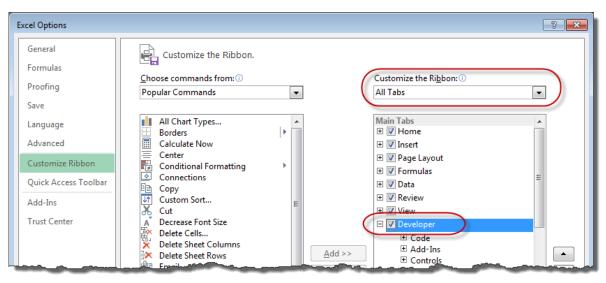


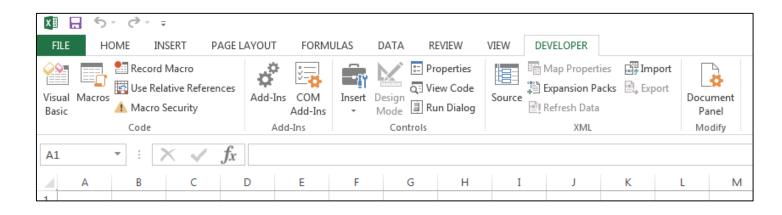
#### Relative

Plays back in any cell.

# Displaying the "Developer" Tab

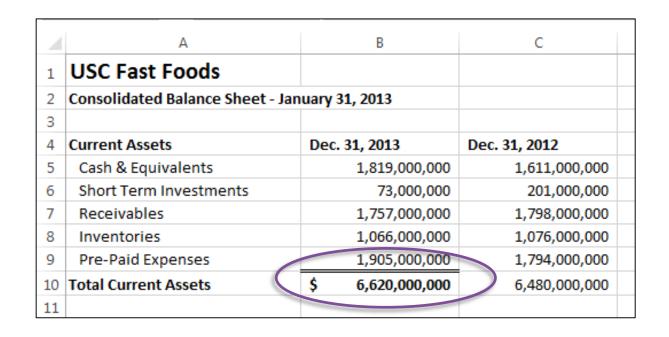
Right click a tab then "Customize the Ribbon".

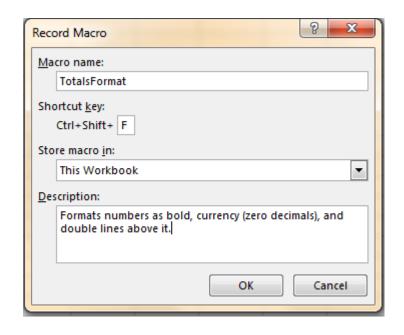




## **Exercise 1: Relative Recording**

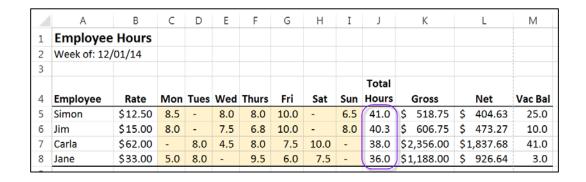
Format Cells with Bold, Double Upperline, and Currency with no numbers to the right of the decimal





## **Exercise 2: Absolute Recording**

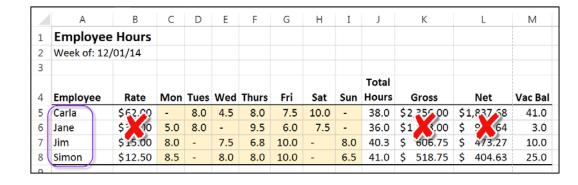
#### Print Setup Macro



#### PrintFullSheet Macro

This macro will print as follows:

- Set Print Area to A1:M8
- Sorted by Total Hours
- Landscape Orientation
- Gridlines Printed



#### PrintPartialSheet Macro

This macro will print as follows:

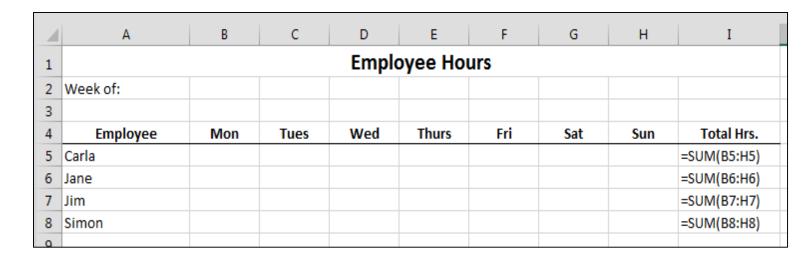
- Set Print Area to A1:M8
- Hide the columns: Rate, Gross, and Net.
- Sorted by Employee name
- Portrait Orientation
- Gridlines Printed

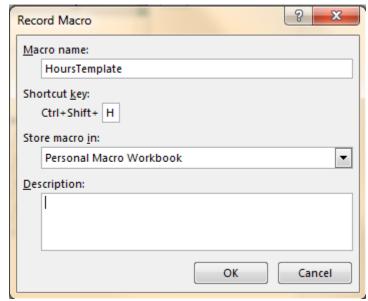
Management Printout

**Employee Printout** 

## **Exercise 3: Absolute Recording**

Building a Template (If you don't click a tab it will work on any sheet)

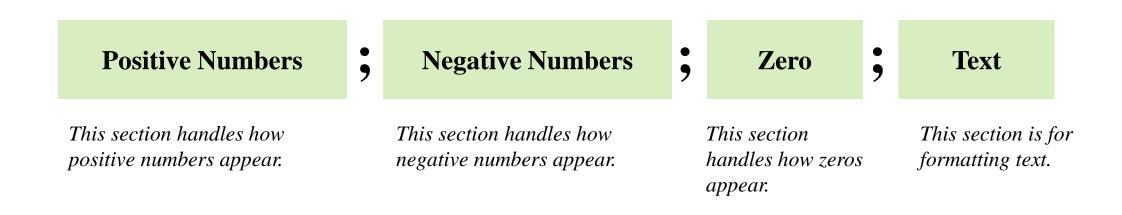




TIP: To insert a new sheet and build the template, type this as the first line of code in your sub procedure: **Sheets.Add** 

## **Exercise 4: Relative Recording**

Custom Format to Display Numbers in Millions



Note: You don't have to use each section. If you use just the first section, both positive and negative numbers are affected. If you skip a section you must still type in its semi-colon.

## **Exercise 4: Relative Recording**

#### Custom Format to Display Numbers in Millions

This comma after the first 0 or # means we wish to use thousands separators.

. Use a decimal point.

"Scales the number to display in millions. ; separates the section for positive and negative values. - in the negative section places a minus in front of the number.

Places a \$ to the left.

\$#,###.0,;[Red]-\$#,###.0,,

# is a placeholder and will not display insignificant digits. For example:

 $6.50 \rightarrow \#.\#\# \rightarrow 6.5$ 

### will display three significant digits between the decimal point and the thousands separator.

**0** is a placeholder and will display insignificant digits. For example:

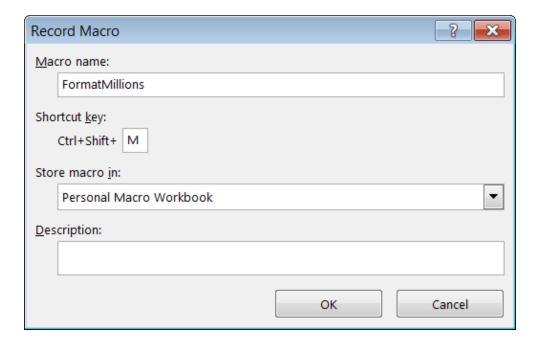
 $6.50 \Rightarrow 0.00 \Rightarrow 6.50$ 

[Red] in the negative section indicates that negative numbers should appear in red.

## **Exercise 4: Relative Recording**

Custom Format to Display Numbers in Millions

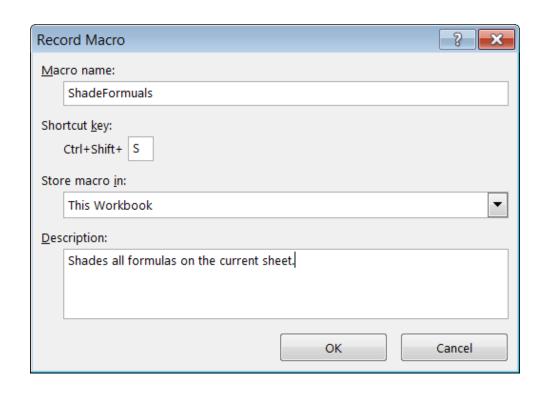
\$#,###.0,,;[Red]-\$#,###.0,,

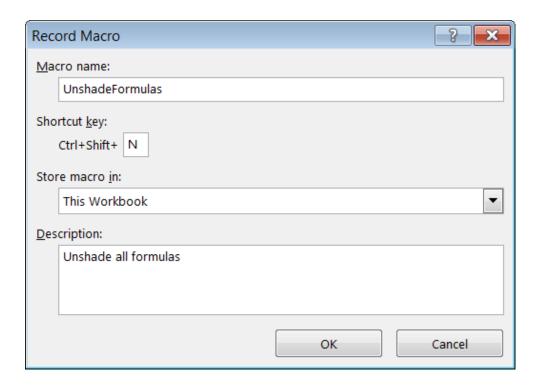


usc Marshall

## **Exercise 5: Relative Recording**

Macros to Shade Formulas & Unshade Formulas





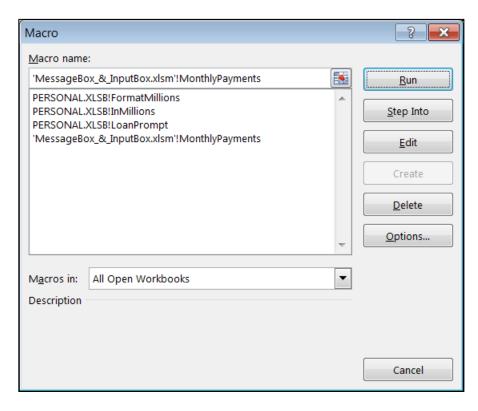
Next: Use VBA to Make a Shade/Unshade Toggle Shortcut

## Introduction to VBA

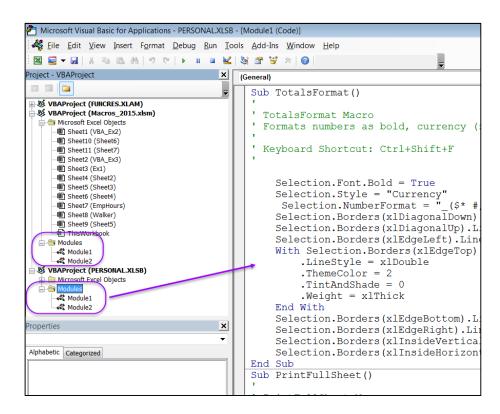
(Visual Basic for Applications)



## **Useful VBA Commands**



Alt + F8
Run a Macro in Excel



**Alt + F11** 

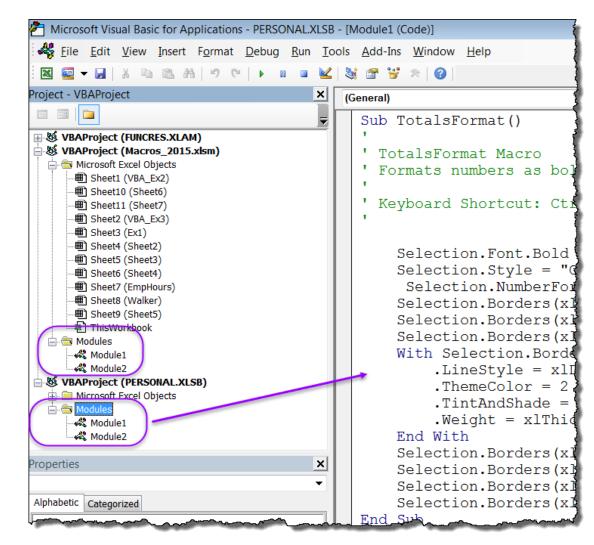
Toggle Between VBA Editor and Excel

# The VBA Environment (Alt + F11)

VBA stored in **Modules**.

Each file can contain multiple modules

A Module can contain multiple Procedures (i.e. macros) or you can separate them into different modules.



## VBA Sub Procedures (i.e. Macros)

(General) **Segins with Sub and end with End Sub Sub** ProcedureName() Sub ShadeFormuals() ShadeFormuals Macro Comments are Shades all formulas on the current sheet. notes to yourself and begin with an Keyboard Shortcut: Ctrl+Shift+S apostrophe. Selection. Special Cells (xlCellTypeFormulas, 23). Select With Selection. Interior .Pattern = xlSolid .PatternColorIndex = xlAutomatic .ThemeColor = xlThemeColorAccent4 .TintAndShade = 0.799981688894314.PatternTintAndShade = 0 End With ActiveCell.Offset(-3, -7).Range("A1").Select End Sub **End Sub** 

## **VBA** Exercise 1: If Then Else Statement

#### Toggle Formula shading On/Off Based on .Pattern

This selects formulas cells.

"With" to "End With" applies multiple properties to the interior of the selected cells.

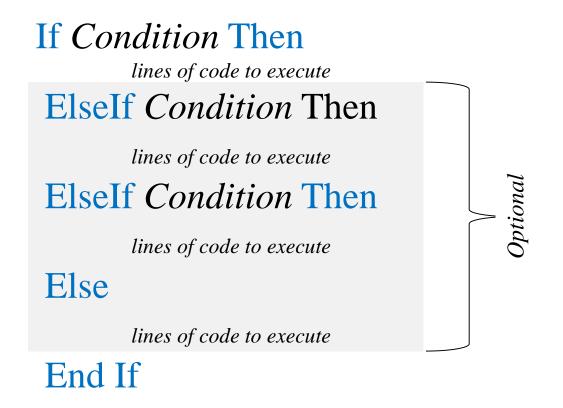
We will apply
either set of
property based
on .Pattern =
XISolid or XINone

```
(General)
 Sub ShadeFormuals()
  ' ShadeFormuals Macro
   Selection. Special Cells (xlCellTypeFormulas, 23). Select
     With Selection. Interior
          .Pattern = xlSolid
          .PatternColorIndex = xlAutomatic
          .ThemeColor = xlThemeColorAccent4
          .TintAndShade = 0.799981688894314
          .PatternTintAndShade = 0
     End With
     ActiveCell.Offset(-3, -7).Range("A1").Select
 End Sub
 Sub UnshadeFormulas()
  ' UnshadeFormulas Macro
   Selection.SpecialCells(xlCellTypeFormulas, 23).Select
     With Selection.Interior
          .Pattern = xlNone
          .TintAndShade = 0
          .PatternTintAndShade = 0
     End With
     ActiveCell.Offset(-3, -7).Range("A1").Select
 End Sub
```

These are the properties being set.

This specifies the active cell. It can be shorted to Range("A1"). Select

## Structure of a VBA If Then...Else Statement



# VBA Exercise 1: Toggle Formula Shading

```
Sub ShadeToggle()
'Toggles formula shading on and off
Selection.SpecialCells(xlCellTypeFormulas, 23).Select
With Selection Interior
   If .Pattern = xlNone Then
          .Pattern = xlSolid
          .ThemeColor = xlThemeColorAccent4
          .TintAndShade = 0.599993896298105
   ElseIf .Pattern = xlSolid Then
          .Pattern = xlNone
          .TintAndShade = 0
          .PatternTintAndShade = 0
   End If
End With
Range("A1").Select
End Sub
```

usc Marshall

#### **VBA Exercise 2:**

Divide a Single Cell by One Million

Sub MillsDivide()

'Divides a single cell by one million.

ActiveCell.Value = ActiveCell.Value / 1000000

End Sub

## VBA Exercise 3: Divide a Range by 1 Million

For Each ... Next Loop and Variables

Allows you to perform actions on each cell in a range of cells one at a time.



#### For Each item In collection

Lines of code

Next item

- item This is a variable name you are assigning each individual item in the collection of objects. It can be any name. In this example we used "item".
- collection This is the name of the object to be affected. For example, "Selection".
- **Next** This causes the next item in the collection to be evaluated.

#### **VBA** Exercise 3:

Divide a <u>Range</u> of Cells by One Million

```
Sub MillsDivideRange()'Divides a range of cells by one million.
```

Dim Item As Range

For Each Item In Selection
Item.Value = Item.Value / 1000000
Next Item

End Sub

Note: This works but has issues if the range contains a formula, blank, or text.

#### VBA Exercise 3: Issues Resolved

**Sub** MillsDivideRange()

'Divides a range of cells by one million.

Dim Item As Range

For Each Item In Selection

If Not IsEmpty(Item) And Not Item. HasFormula And IsNumeric(Item) Then Item. Value = Item. Value / 1000000

Next Item

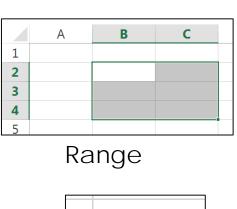
End Sub

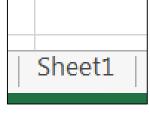
#### Notes:

- If your If Statement is all on one line you don't need "End If".
- Formulas and Blanks are considered Numeric entries but text is not.

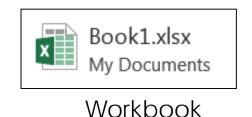
Objects, Collections, Hierarchy, Properties, Methods & Arguments

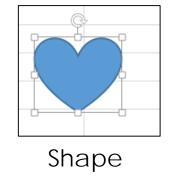
Object
Oriented
Programming
Language

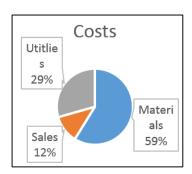




Worksheet







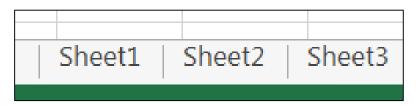
Chart

Plus Over One Hundred more!

Objects, Collections, Hierarchy, Properties, Methods & Arguments

#### **Collections**

Group of Objects of the Same Type. A Collection is also an Object and is Plural.



For example, a workbook contains multiple worksheet objects. The collection object of these is worksheets

Syntax: CollectionName("Object Name")

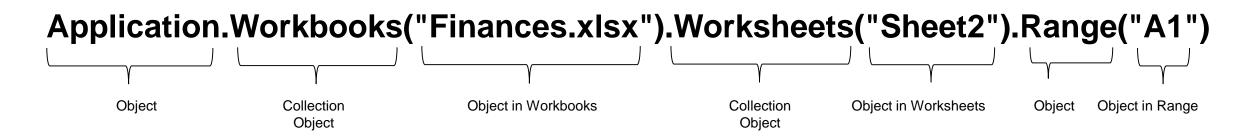
Example: Worksheets("Sheet1") Refers to Sheet1 in the workbook.

Objects, Collections, Hierarchy, Properties, Methods & Arguments

## Hierarchy

Objects can
Contain other
Objects.

To specify an object, you must follow the path through the parent objects to get to the object you wish to affect. In this example, A1 is on Sheet2 in the file called "Finances.xlsx". Hierarchical levels are separated by periods.



Note that just **Range("A1")** would refer to A1 on the current sheet in any file. You don't have to go through the entire Hierarchy unless you need to be specific.

Objects, Collections, Hierarchy, Properties, Methods & Arguments

Objects have **Properties**.

Example: a range of cells has a color, width or can contain a value. You can do two things with properties:

- Determine the Current Properties of an Object.
- Set the Properties of Most Objects.

Range("A1").Value

Determines the value in cell A1.

Range("A1"). Value = 100

→ Sets the value in cell A1 to 100.

Objects, Collections, Hierarchy, Properties, Methods & Arguments

Objects have Methods.

A Method is an Action you can Perform with the Object.

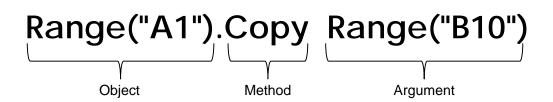
Range("A1").ClearContents

→ Erases the contents of cell A1 using the "ClearContents" method.

Note: Methods may seem similar to Properties in that with both you can change something but with properties, you are merely setting a property inherit in that object whereas with methods, you are doing something with the object.

Objects, Collections, Hierarchy, Properties, Methods & Arguments

Most Methods have Arguments as do Some Properties. Arguments further Define what an Method does.



The Copy method copies the contents of cell A1. Its optional paste argument then pastes it to cell B10 (i.e. Range("B10").

Note: Method arguments follow the name of the argument separated by a space. If using multiple arguments separate them with commas.

#### Object Model Map

http://msdn.microsoft.com/en-us/library/office/aa209782(v=office.10).aspx

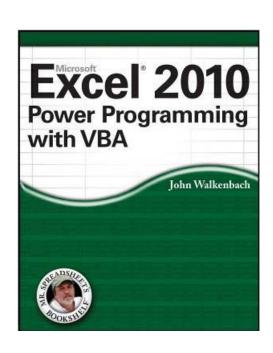
```
Application
  Workbooks (Workbook)
   Worksheets (Worksheet)
   Charts (Chart)
   DocumentProperties (DocumentProperty)
   VBProject
   CustomViews (CustomView)
   CommandBars (CommandBar)
   HTML Project
```

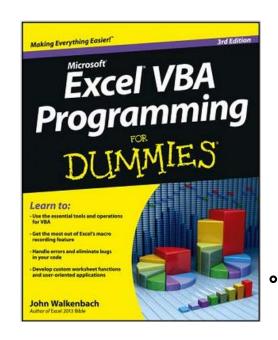
## VBA Object Model

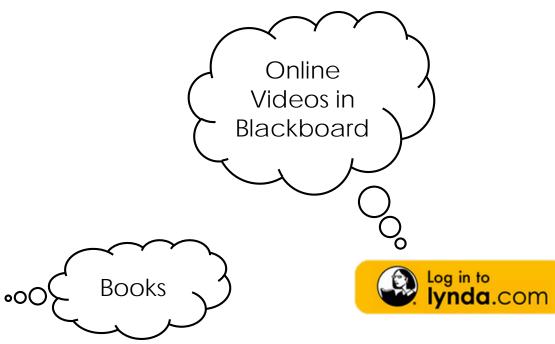
More Leaning Resources



http://www.excel-spreadsheet.com/vba/objecthierarchy.htm



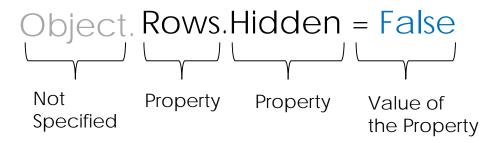




### **VBA Exercise 4: Unhide All Columns & Rows**

This will Unhide all Columns and Rows in the Current Sheet

Sub UnhideAll()
Rows.Hidden = False
Columns.Hidden = False
End Sub

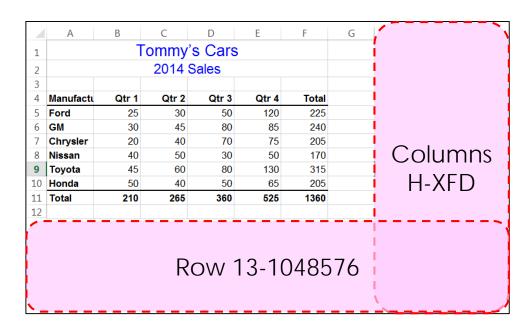


We did not specify a specific object so the code above affects the current sheet. To unhide columns and rows on a specific sheet, you would specify the object: Worksheets("Sheet4").Rows.Hidden = False

#### **VBA** Exercise 5

Hide Specific Columns & Rows (Columns H-XFD & Rows 13-1048576)

Sub HideColRows3()



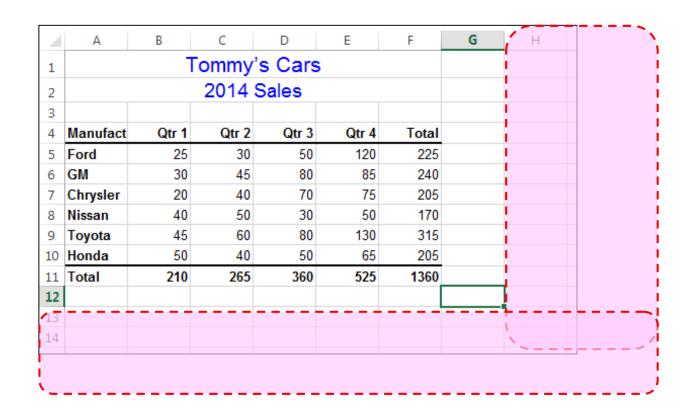
Range("H1", "XFD1").EntireColumn.Hidden = True Range("A13", "A1048576").EntireRow.Hidden = True

End Sub

Note: Because we are not hiding all columns and rows, we can't use "Rows" or "Columns" as in Exercise 4, we must use the "EntireRow" & "EntireColumn" properties.

## **VBA** Exercise 6

Goal: Hide all rows below the currently selected cell and all columns to its left.



We need to specify the first column to be hidden.

For example, this would hide column H if the currently selected cell were G7.

	1	2	3	4	5	6	7	8
	Α	В	С	D	E	F	G	Н
1	Tommy's Cars							
2	2014 Sales							: 1
3								i ¦
4	Manufact	Qtr 1	Qtr 2	Qtr 3	Qtr 4	Total		! i
5	Ford	25	30	50	120	225		1 !
6	GM	30	45	80	85	240		!
7	Chrysler	20	40	70	75	205		
8	Nissan	40	50	30	50	170		
9	Toyota	45	60	80	130	315		i
10	Honda	50	40	50	65	205		i !
11	Total	210	265	360	525	1360		! !
12								
13								
14								<u> </u>

#### ActiveCell.Column

(i.e. 7)

Returns the current column (G) as an index number (7).

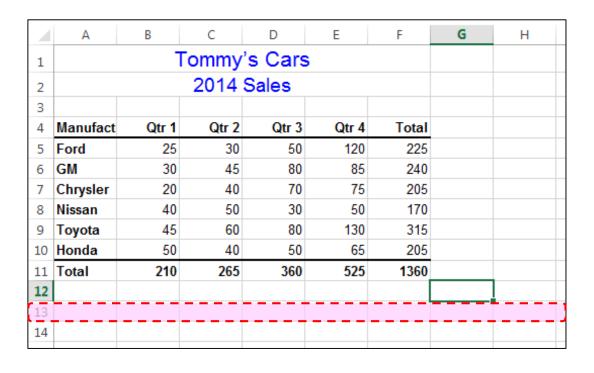
ActiveCell.Column + 1

(i.e. 7+1)

Returns one column to the right of the current column as an index number (8) (i.e. Column H).

We need to specify the first row to be hidden.

This would hide row 13 if the currently selected cell were G7.



#### ActiveCell.Row

Returns the current row (12).

ActiveCell.Row + 1 (i.e. 12+1)

Returns one row below the current row (13).

Assigning Variables to the Current Column & Row

Sub HideColRows2()

StartCol = ActiveCell.Column + 1 StartRow = ActiveCell.Row + 1

**End Sub** 

We will assigning the variable "StartCol" to the first column to be hidden and StartRow to the first row to be hidden.

Range() Requires Cell Addresses: Range("StartingCell", "EndingCell")

Earlier we used this code to hide specific columns and rows.

```
Sub HideColRows3()
Range("H1", "XFD1").EntireColumn.Hidden = True
Range("A13", "A1048576").EntireRow.Hidden = True
End Sub
```

Placing our variables here is correct but they are returning numbers and Range() requires cell addresses so this won't work!

```
Sub HideColRows2()
Range(StartCol, "XFD1").EntireColumn.Hidden = True
Range(StartRow, "A1048576").EntireRow.Hidden = True
End Sub
```

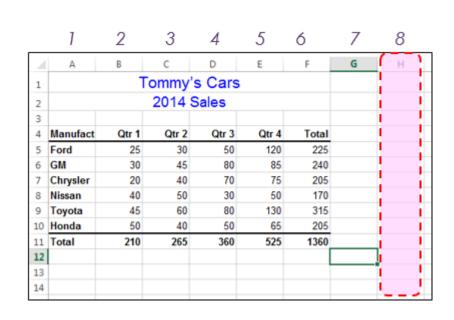


Fortunately, we can use the Cells() property to convert an number to a cell address.

## VBA Exercise 6 – Hiding the Columns

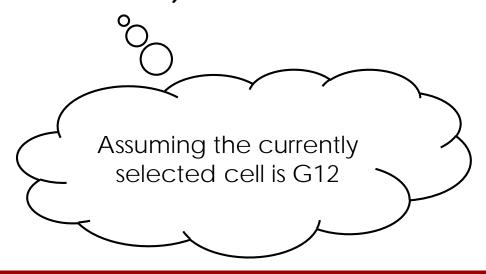
Cells() Property to Convert Numbers to Cell Addresses

Syntax: Cells(Row Index Number, Column Index Number)

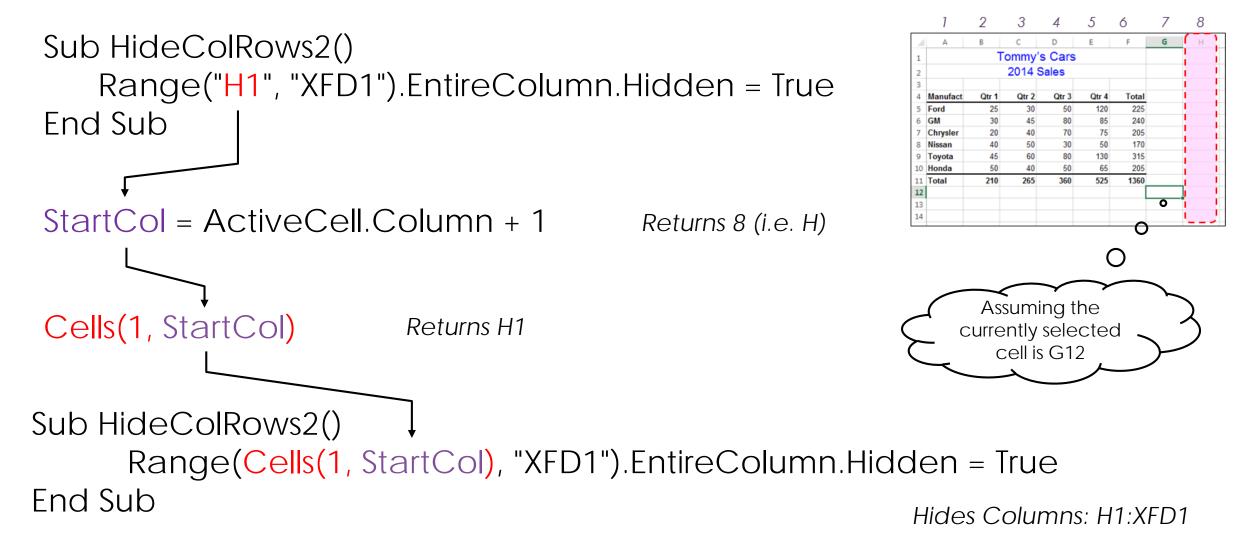




Cells(1, StartCol) Returns H



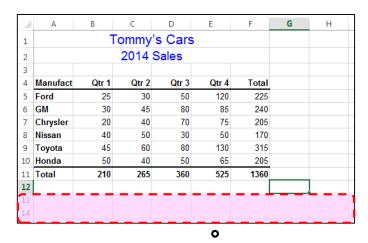
## VBA Exercise 6 - Hiding the Columns

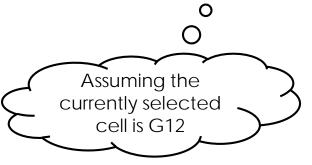


usc Marshall

## VBA Exercise 6 – Hiding the Rows

Sub HideColRows2() Range("A13", "A1048576").EntireRow.Hidden = True End Sub StartRow = ActiveCell.Row + 1 Returns 13 (i.e. 12+1) Cells(StartRow, 1) Returns A13 Sub HideColRows2()





Range(Cells(StartRow,1), "A1048576").EntireRow.Hidden = True

End Sub

Hides Rows: A13:A1048576

## **VBA Exercise 6 - Complete**

Hide Columns & Rows Beyond the Current Cell

Sub HideColRows2()

StartCol = ActiveCell.Column + 1 StartRow = ActiveCell.Row + 1

Range(Cells(1, StartCol), "XFD1").EntireColumn.Hidden = True Range(Cells(StartRow, 1), "A1048576").EntireRow.Hidden = True

**Fnd Sub** 

### **VBA** Exercise 7 – Unknown Maximums

Hide Columns & Rows Beyond the Current Cell

Range(Cells(1, StartCol), "XFD1").EntireColumn.Hidden = True Range(Cells(StartRow, 1), "A1048576").EntireRow.Hidden = True

Sub HideColRows()

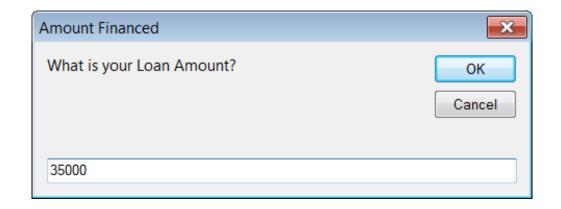
StartCol = ActiveCell.Column + 1 StartRow = ActiveCell.Row + 1

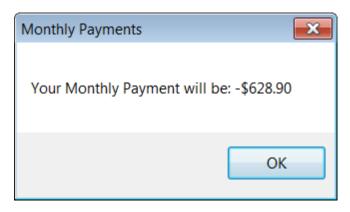


End Sub

## Exercise 8: InputBox & MsgBox

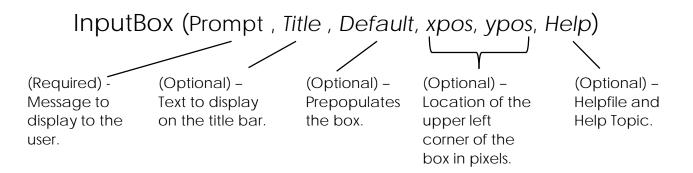
Asking for Input and Giving Output using Pop-ups

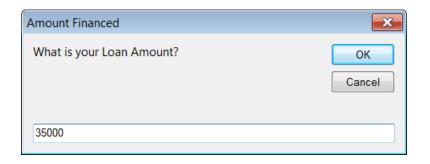




## **Exercise 8: InputBox**

Asking for Input





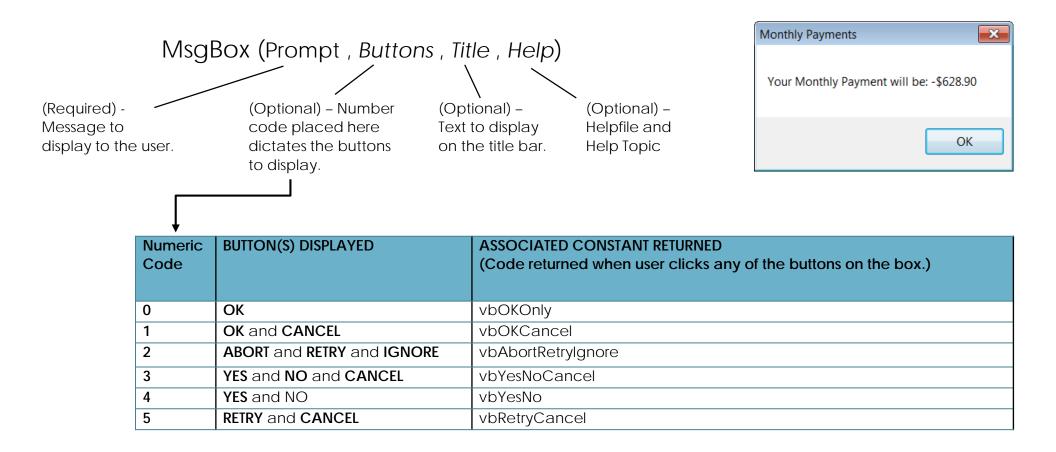
Sub MonthlyPayments()

A = InputBox("What is your Loan Amount?", "Amount Financed")

End Sub

## Exercise 8: MsgBox

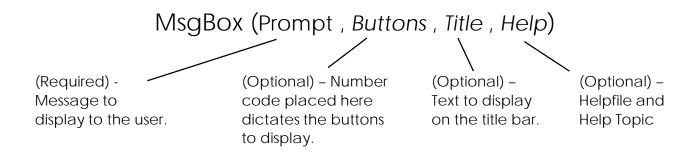
Message in a Pop-up Box

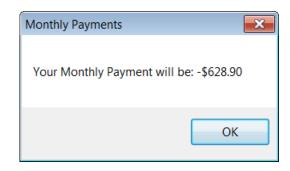


usc Marshall

## Exercise 8: MsgBox

Message in a Pop-up Box



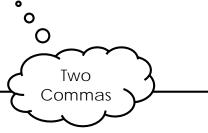


#### Sub MonthlyPayments()

A = InputBox("What is your Loan Amount?", "Amount Financed")

S = MsgBox("You can't afford a loan of " & A ,,"Amount Financed")

**End Sub** 

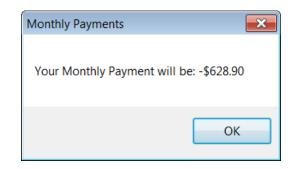


### **Exercise 8: WorksheetFunction.Pmt**

Returning the Payment Amount with Pmt()

Pmt(Rate, Periods, LoanAmt)

Rate for a single Total number of compounding periods.



#### Sub MonthlyPayments()

A = InputBox("What is your Loan Amount?", "Amount Financed")

P = WorksheetFunction.Pmt(0.03 / 12, 60, A)

S = MsgBox("Your Monthly Payment will be: " & P , , "Amount Financed")

#### **End Sub**

Note: You must prefix an Excel function with "WorksheetFunction."

## **Exercise 8: Fixing Issues**

Clicking "Cancel" Returns an Error and Numbers Not Formatted

```
Sub MonthlyPayments()
```

#### End Sub

Note this also works: P = Format(WorksheetFunction.Pmt(0.03 / 12, 60, A), "\$#,###.00")

## **Exercise 8: Fixing Issues**

What if the user does not type a number?

```
Sub MonthlyPayments()
  A = InputBox("What is your Loan Amount?", "Amount Financed")
  If A = "" Then Exit Sub
  If IsNumeric(A) Then
        P = WorksheetFunction.Pmt(0.03 / 12, 60, A)
        PF = Format(P, "$#,###.00")
       S = MsgBox("Your Monthly Payment will be: " & PF , , "Amount Financed" )
   Else
        S = MsgBox("You must enter a numeric value", , "Entry Error")
        Call MonthlyPayments
   End If
                                            Tip: To write the value of "P" to a cell:
                                            Worksheets("Sheet1").Range("A1").Value = P
End Sub
```

## Custom Functions with VBA

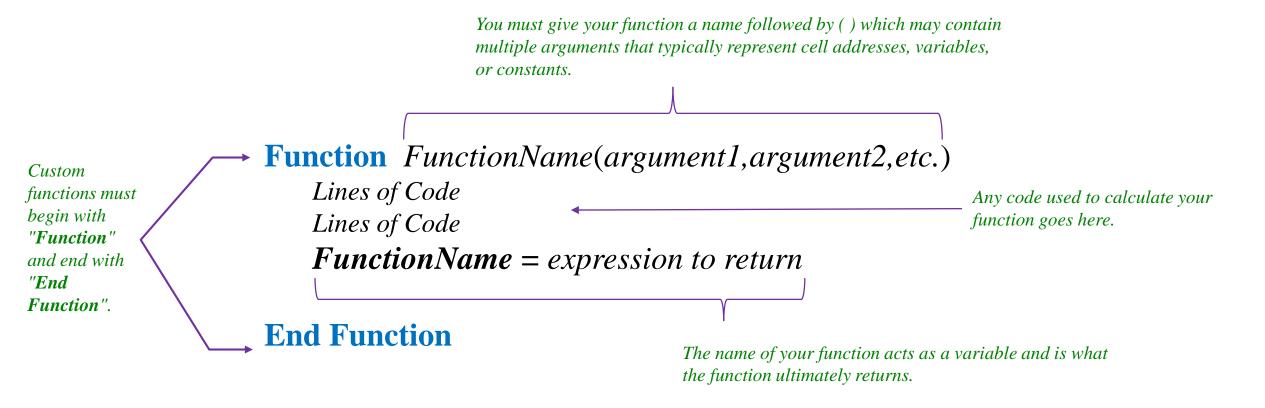
**Function** FunctionName() VBA Code

**End Function** 



### **About Custom Functions in VBA**

**Basic Structure** 



usc Marshall

Area of a Rectangle

Function Area(L,W)

Area = L \* W

**End Function** 

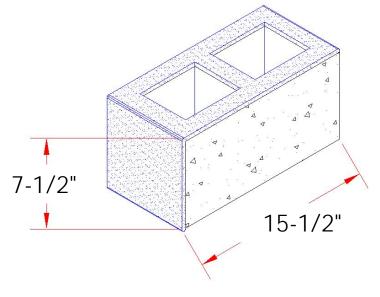
Number of Blocks Needed in a Block Wall

Function Blocks(FootLength, FootHeight)



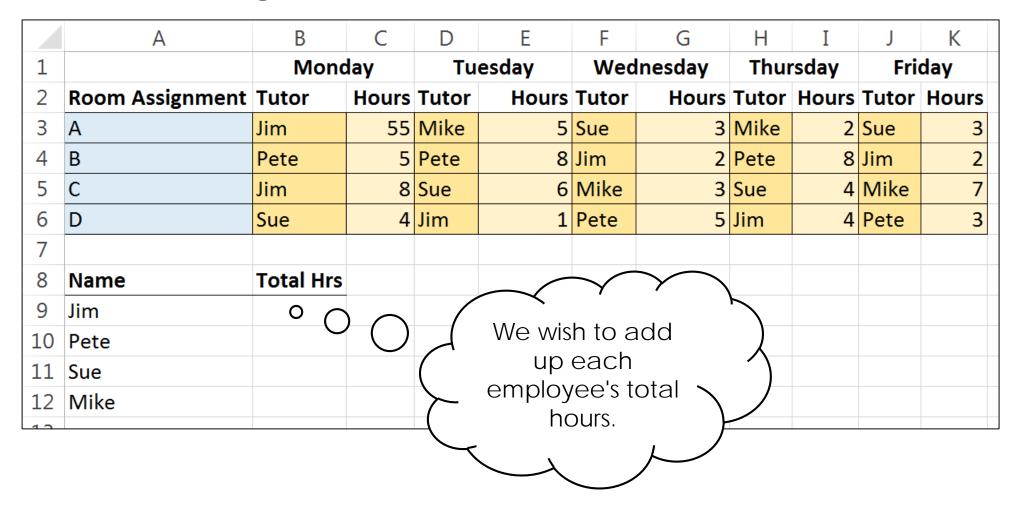
L = FootLength \* 0.75 H = FootHeight \* 1.5 Blocks = L \* H

**End Function** 



Assumes a standard block size of 15-1/2 Inches Long by 7-1/2 Inches High

Sum Cells to Right of Criteria



Offset(Y Position, X Position)

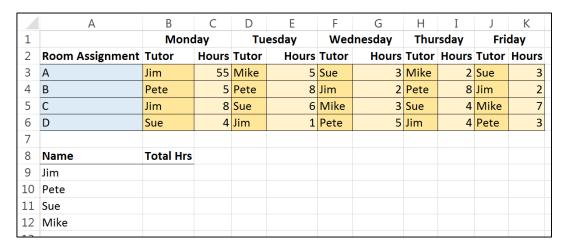
Function SumC(Rng, Crit)

For Each Item In Rng

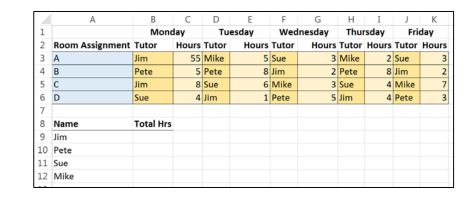
If Item. Value = Crit Then SumC = Item. Offset(0, 1). Value + SumC

Next Item

**End Function** 



Declaring Variables





Function SumC(Rng As Range, Crit As Range)
Dim Item as Range

For Each Item In Rng

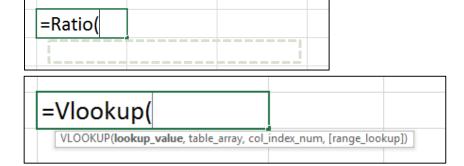
If Item.Value = Crit Then SumC = Item.Offset(0, 1).Value + SumC

Next Item

**End Function** 

# Help with Custom Functions

No Smart Tags for Custom Functions
Unfortunately, VBA does not provide a
method of creating Smart Tags.

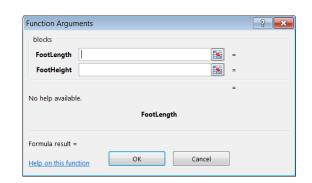


#### **Suggestion: Use Descriptive Arguments**

- a) Type Function Name: =Blocks(
- b) Then Press Control + Shift + A

OR

- a) Type Function Name: =Blocks(
- b) Click  $f_x$



=Blocks(FootLength, FootHeight)

## Using Custom Functions in Other Files

#### Three Methods

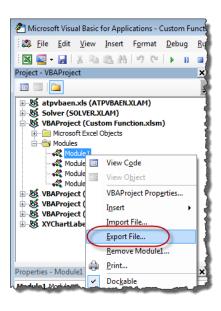
#### **Specify the File Name**

Specify the name of the file as a prefix to the function name. For example:

=MyFunctions.xlsm!Area()

#### **Copy the Module**

Export the module containing the custom functions then import it into another file in the VBA Editor.



#### Save File as an Add-in

Save the Excel file containing your custom functions as an Add-in. Then load the Add-in.



