

# Data 301 Data Analytics

## Microsoft Excel VBA

### Excel Part 3 of 3

**Dr. Irene Vrbik**

University of British Columbia Okanagan  
irene.vrbik@ubc.ca

# What is VBA?

Excel Visual Basic for Applications (VBA) is a programming language developed by Microsoft.

It allows users to build their own functions, automate tasks in several Microsoft applications, including Microsoft Office, and develop customized code.

The language has been part of almost all versions of Office for over 20 years.

VBA allows for expanding the capabilities of Excel and provides a level of customization, eg. adding user-interface elements (buttons, lists) to your spreadsheet.

# Why Microsoft Excel Visual Basic for Applications?

Microsoft Excel VBA allows for automating tasks in Excel and provides a full programming environment for data analysis.

- ▶ We can use VBA to automate tedious processes in Excel, eg. formatting a month report, creating headers to look a certain way, ...

Excel VBA is commonly used in high finance (eg. investment banking) and frequency trading applications for creating and validating financial models.

Using Excel VBA will be our first practice with programming and allow us to explore fundamental programming concepts of commands, variables, decisions, repetition, objects, and events.

# Macros

A *macro* (short for macroinstruction) is a recorded set of actions that is saved so that they can be easily executed again.

If you do the same set of actions repetitively, then creating a macro allows for doing all those actions with one command.

Macros are accessible either under the **View** or the **Developer** tab in the Ribbon (I usual navigate to the latter).

Macros are converted into VBA *programs*; that is, a sequence of instructions that a computer can interpret and execute.

# Developer Tab

The Developer tab contains icons for performing VBA and macro development.

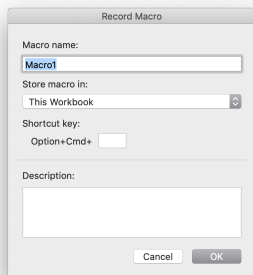
This tab is disabled by default.

To add the Development tab on a PC, go to **File, Options, Customize Ribbon** and make sure it is checked beside Developer.

For a Mac, go to **Excel, Preferences, View**. Under the *In Ribbon, Show* heading, select the checkbox marked "Developer Tab"

# Recording a Macro

To record a macro, go to the **View** tab or **Developer** tab and select the “Record Macro” button



Once selected, pop-up window will appear prompting you to enter a Macro name and (optional) keyboard shortcut.

# Recording a Macro

Macro names cannot contain spaces or begin with a number.

When possible, you should try to give a meaningful/concise macro name that provides some insight into what it does.

It is recommended to use **Ctrl** + **Shift** + <Key> (PC) / **Option** + **Cmd** + <Key> (Mac) for a Shortcut key so that you do not override built-in shortcuts.

Macs will give you a warning when you attempt to override an existing shortcut.

A macro can be created without assigning it a shortcut key.

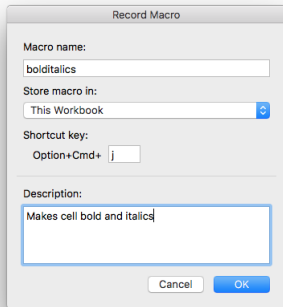
# Recording a Macro

- ▶ To record your macro's actions, fill out the fields in the pop-up window, click OK then perform the actions you wish to record.



- ▶ Once you start, the **Record Macro** will become **Stop Recording** , a **Stop Recording** button
- ▶ Save for cursor movements, Excel will record your every action until you select the **Stop Recording** button.
- ▶ Next time we would like to perform these actions, rather than going through all the motions, we can simply run this recorded macro.





- ▶ As a simple example, we could create a macro that bold and italicizes a cell.
  - ▶ Macro name: bolditalics
  - ▶ Shortcut: Option + Cmd + j
  - ▶ Makes cells bold and italics
- ▶ While the macro name is pretty self-explanatory, it is usually a good idea to provide a description about what your macro does.

# Recording Macros

By default, macros are created using absolute references

- ▶ To use Relative references, we need to turn on the "Use Relative References" button, before we record.<sup>1</sup>
- ▶ If you want your macro to run on the active cell, make sure we don't move out of the active cell once we start to record our macro.

While the Macro Recorder makes creating macros very easy, it has it's limitations:

- ▶ eg. cannot handle "loops" (more on loops later)
- ▶ generates more code than is necessary (which can slow down your process).

---

<sup>1</sup>feature **not available for Macs**

# Using a Macro

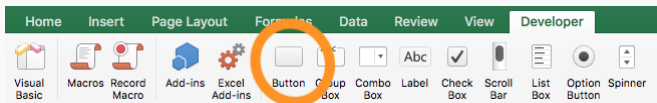
There are a number of ways you can use your macro:

1. With the shortcut key if defined
2. Under the **View** tab, Select **View Macros** (or **Macros** under the **Developer** tab) then highlight the macro name and click **Run**.
3. Assign a macro to a button or on the toolbar.

# Macro Buttons

To assign a macro button:

1. Select the **Developer** tab and click



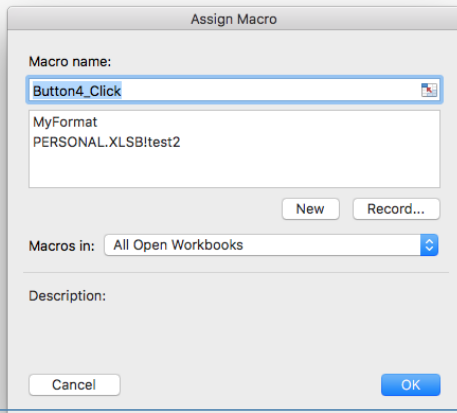
2. To prompt the Assign Macro popup window, click where you would like this button to appear on your worksheet.

**Side-note:** You can also assign macros to shapes (Insert "Shapes"), by right-clicking on it, and selecting **Assign Macro**.

# Macro Buttons

## 3. Then, either:

- ▶ select the name of an existing macro from those listed in the pop-up window,
- ▶ press the **Record...** button to record a new macro, or
- ▶ press **New** to open the VBA editor (more on this later).



# Try It: Macros

## Question

Create a macro called MyFormat that does the following tasks:

1. Use a shortcut of **Ctrl** + **Shift** + **b** (PC)/**Opt** + **Cmd** + **b** (Mac).
2. Bolds the cell and makes the font Courier 20
3. Sets the cell background to orange.
4. Centers the text in the cell.
5. Add it to a button in the shape of a star that says OC20

Try-out your macro using 1) the shortcut key, 2) the macro dialog, and 3) the macro button.

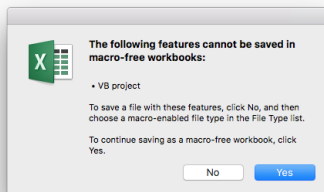
## Warning

- ▶ There is no easy way to undo a macro update.
- ▶ That is to say, the safety net that comes in the form of the **Ctrl** / **Cmd** (Windows/Mac) + **Z** keyboard combination will not allow you to undo any changes that your macro made to your document (in fact, running a macro completely erases the Undo list)
- ▶ While there is no intrinsic way of preserving the Undo list, you can save your workbook immediately before running the macro. If you want to undo the effects, simply close the workbook without saving and re-open your preserved pre-macro version.

## Saving macros

In order to save the workbook with the macros we've created while it was open, we need use a "macro-enabled" file format.

When you go to save a workbook containing macros, you will receive the following popup



- ▶ 'Yes' will save it as as macro-free workbook (\*.xlsx file)
- ▶ Select 'No' to save it as as macro-enabled workbook (\*.xlsm file)



# Saving macros

Saving a macros to a given workbook will only allow you to use it with that file.

We could alternatively save it to a *Personal Workbook* allowing them to be used in multiple workbooks.

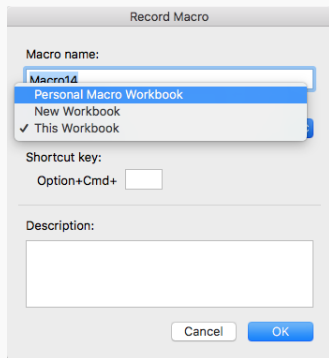
Your Personal Workbook is a hidden workbook saved under the name **personal.xlsb**

This file loads every time you open up Excel.

Hence saving a macro to **personal.xlsb** will allow you to use that macro on any workbook that you open on your computer

# Saving macros

To save it to your personal workbook, select the "Personal Macro Workbook" option in the drop-down menu for "Store Macro in"



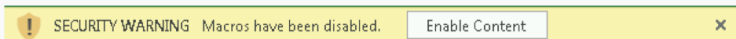
# Macro Security

Since macros can execute any code, they have been a target for virus writers.

Understanding the source of the Excel spreadsheet that contains macros is important when deciding to run them or not.

Excel has *macro security settings* to allow you to enable or disable running macros.

Spreadsheets with macros often will generate a warning when opening them:



# Macro Security Settings

The default security is to *Disable all macros with notification*. This prevents macros from running but displays a warning allowing you to enable them.

One of the biggest issues with macros is security. **Make sure you are only using macros from a trusted source.**

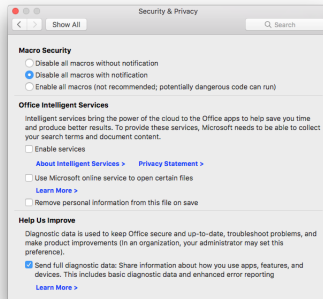
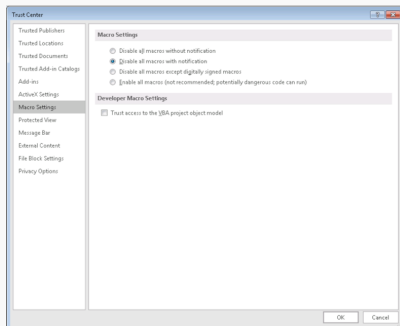
To change these defaults (which I don't recommend):

**Windows** click "Macro Security" located below the "Record Macro" button on the **Developer** tab

**Mac** Navigate to **Excel** in the toolbar → **Preference** "Security & Privacy"

Select a desired option in Macro Settings category, under the *Macro Security* header.

# Macro Security Settings



# Macros: Implementation

Macros are converted to Visual Basic (VB) code.

You can edit macro code and create your own code.

Under the **Developer** tab, select the **Macros** button then click the **Edit** button in the pop-up window.

The code will then open in you *Visual Basic Editor (VBE)*

**Tip:** It is best practice to break long tasks into, smaller relevant macros rather than one big macro.

**Side-note:** Macros can be written for and any other Office application that supports VBA.

# Recording a Macro

Here is the VBA code that was produced during our bolditalics macro recording:

```
1 Sub bolditalics()  
2 '  
3 ' bolditalics Macro  
4 ' bold and italicize text  
5 '  
6 ' Keyboard Shortcut: Ctrl+j  
7 '  
8     Selection.Font.Bold = True  
9     Selection.Font.Italic = True  
10 End Sub
```

# Comments

- ▶ The apostrophe indicate the starting of a *comment*.
- ▶ *Comments* are ignored by the computer and are used by the programmer to document and explain the code.
- ▶ On the previous slide, our comments provide the name of our macro, a description of what it does and the keyboard shortcut information.
- ▶ If we were to remove those commented lines (ie. lines 2–7), the macro would still run exactly the same; however, a reader of the VBA code would have a harder time determining what this macro does.



# Integrated Development Environment

An integrated development environment (IDE) enables programmers to consolidate the different aspects of writing a computer program.

More generally, a IDE provides a convenient way for you to write, compile, and run your code all in one place (as opposed to having to open many applications and windows).

An IDE typically contains a code editor, a compiler or interpreter, and a debugger, accessed through a graphical user interface (GUI).

- ▶ The user writes and edits source code in the code editor.
- ▶ The compiler translates the source code into a readable language that is executable for a computer.
- ▶ And the debugger tests the software to solve any issues or bugs.

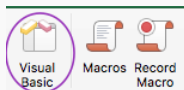
# Visual Basic Editor

Visual Basic Editor (VBE) allows editing visual basic code and is a complete IDE.

- ▶ That is to say, users can create and edit macros as well as other Visual Basic code with the editor.

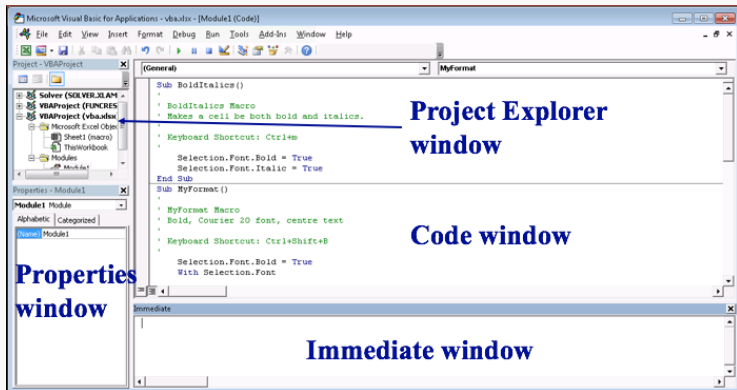
To open the VBE either:

- ▶ Edit a macro as explained on [this](#) slide
- ▶ Under the **Developer** tab click the **Visual Basic** button



- ▶ or use the keyboard shortcut: **Alt** + **F11** .

# Visual Basic Editor



If you don't see the Immediate window, you can add by going to **View > Immediate window** (while in VBE mode).

# Visual Basic Editor

**Project (Explorer) window** This window shows the tree diagram with every workbook and its worksheets that are currently opened organizes code files (modules). You can expand and collapse them by clicking the plus and minus signs.

**Code window** shows source code for editing. By default, this windows is empty. Here, you can put your VBA code. In order to open this window double-click one of the objects in the Project window.





**Immediate window** allows for execution of commands and getting immediate results.

**Properties window** display properties and values of currently selected object.

# Visual Basic Editor - Toolbar

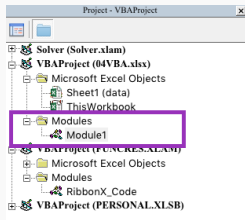
In the Toolbar, you can find buttons to which you can have quick and easy access.

For example:

- ▶ running a macro (  ) ,
- ▶ stopping a macro (  ),
- ▶ switching back to Excel (  ) ,
- ▶ saving the macro (  )
- ▶ and more ...

# Modules

- ▶ A VBA project is a collection of so-called *modules*
- ▶ You can think of a VBA module as a text document containing your VBA code.



- ▶ All of your code can go into a single module; however, you may choose to save your code across multiple modules for organizing large projects.
- ▶ For more information see [here](#)

**Exercise:** Open the VB Editor in a window beside your active workbook. Press the record macro button and see how the VBA code produce before you eyes!

# Immediate Window

- ▶ The VBA Immediate Window is a window from which you can get immediate answers about your Excel files, and quickly execute code.
- ▶ **Tip from Excelcampus** If you would like to “undock” the immediate window from the VB editor:
  1. Left-click and hold on the top bar of the immediate window.
  2. Drag it out of the VB Editor window. The immediate window becomes a free floating window that you can put on top of Excel.
  3. To re-dock it, double-click on the top bar of the immediate window.
- ▶ For instance, you can use the Immediate Window for performing simple calculations. For example type `?765*39` into the Immediate Window and press **ENTER**. The solution 29835 should appear directly below.

# Visual Basic Editor: Immediate Window

The `?` (or alternatively `PRINT` or `Debug.Print`<sup>2</sup>) evaluates/displays specified information to the immediate window.

- ▶ You may use `msgbox` to display them in a pop-up window.
- ▶ `?Worksheets.Count` prints the number of worksheets in our workbook
- ▶ `? Range("A2").Value` prints the value currently stored in **A2**
- ▶ ...
- ▶ See more examples in the **ImmediateWindowExamples** file uploaded to Canvas

---

<sup>2</sup>use within a VBA program; see Example 4 [here](#)



# Visual Basic Editor: Immediate Window

We can also ...

- ▶ format cells. Eg, center the text in the "active" cell (ie. the one currently highlighted in a green border):

```
ActiveCell.HorizontalAlignment = xlCenter
```

- ▶ reassign the value in cell **A2** to 10 using:

```
Range("A2").Value = 10
```

- ▶ run a macro by calling it by name. For example, the following will apply the corresponding macro to the active cell.

```
MyFormat
```

## Visual Basic Editor: Immediate Window

We can also use VBA code for doing things we would normally do with our mouse and keyboard. For example:

```
Worksheets("forms").Activate  
Workbooks("04VBA.xlsx").Worksheets("forms").Activate
```

will change the active workbook to forms in our **04VBA.xlsx** workbook (if you don't specify the workbook, it will default to the currently active workbook). We can activate worksheets either by name or number:



```
Worksheets(1).Activate  
Worksheets("data").Activate
```

Alternatively we can also call the sheet by the Excel object name given in the Project window:

```
Sheet1.Activate
```

## Visual Basic Editor: Immediate Window

We can select certain cells in a similar manner to how we would in an excel formula. For example

- ▶ `Range("A6").Activate` is equivalent to clicking on cell **A6** with our mouse. We could also use `Cells(6,1).Activate`
- ▶ `Range("A1:A4").Select` is equivalent to selecting (ie. via clicking a dragging) to corresponding range of cells.
- ▶ `Range("A1, C3, E7").Select` is equivalent to activating (ie. via clicking whilst holding  /  (Windows/Mac)) non-contiguous cells.

N.B. Many cells can be selected, but only one object (eg. cell, workbook, row) may be the activated at any given time.

# Try It: Immediate Window

## Question:

Try do these actions using the immediate window:

1. Print "Hey There!"
2. Calculate the answer of  $765 * 39$ .
3. Count the number of worksheets.
4. Switch the active worksheet to *Sheet2*.
5. Switch the active worksheet to *Sheet1* and select cells **A1:A4**.
6. Switch the active cell to **A6** and print its value.
7. Change the value in the current cell (ActiveCell) to 10.
8. Use the command `msgbox` to display value in current cell (ActiveCell).

# Subs

Most of the macros you write in VBA are Sub procedures.

A *Sub procedure* (also know as a subroutine or “sub”) is set of *commands* to perform a certain task.

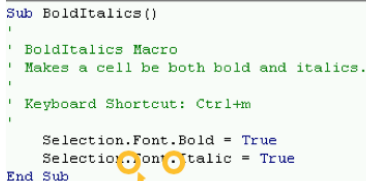
In the code window, text is colour coded for ease of readability:

```
Sub macro_name()  
,  
,  
, comments are in green  
, place description here  
,  
,  
, Keyboard Shortcut: if applicable  
,  
  
    Command 1  
    Command 2  
  
    ...  
    Command n (errors in red)  
End Sub
```

# Macro Code in Visual Basic Editor

The general syntax for a subroutine:

## Subroutine with name and no arguments



```
Sub BoldItalics()  
'  
' BoldItalics Macro  
' Makes a cell be both bold and italics.  
'  
' Keyboard Shortcut: Ctrl+M  
'  
    Selection.Font.Bold = True  
    Selection.Font.Italic = True  
End Sub
```

Comments start with '

Every statement is on its own line.

Dot notation to separate "items" (objects, methods, properties).

# WITH Statement in Visual Basic Code

```
Sub MyFormat()  
|  
' MyFormat Macro  
' Bold, Courier 20 font, centre text  
,  
' Keyboard Shortcut: Ctrl+Shift+B  
,  
  
Selection.Font.Bold = True  
With Selection.Font  
    .Name = "Courier New"  
    .Size = 20  
    .Strikethrough = False  
    .Superscript = False  
    .Subscript = False  
    .OutlineFont = False  
    .Shadow = False  
    .Underline = xlUnderlineStyleNone  
    .ThemeColor = xlThemeColorLight1  
    .TintAndShade = 0  
    .ThemeFont = xlThemeFontNone  
End With  
With Selection.Interior  
    .Pattern = xlSolid  
    .PatternColorIndex = xlAutomatic  
    .ThemeColor = xlThemeColorAccent2  
    .TintAndShade = 0  
    .PatternTintAndShade = 0  
End With
```

WITH syntax simplifies typing same object many times.

These lines all apply to Selection.Font.

# Advanced: Object-Oriented Programming

*Object-oriented programming* structures code as objects, classes, methods, and properties. This organization makes it easier to understand and construct large programs.

An *object* is an instance of a class that has its own properties and methods that define what the object is and what it can do.

A *class* is a generic template (blueprint) for creating an object.

A *property* is an attribute or feature of an object.

A *method* is a set of statements that performs an action.



## Advanced: Object-Oriented Programming

All *objects* of a *class* have the same *methods* and *properties* (although the property values can be different).

Consider this the analogy:

- ▶ Class: Laptops
- ▶ Object: MacBookAir
- ▶ Property: size
- ▶ Value: MacBookAir.size = 11inch
- ▶ Method: MacBookAir.PowerOn

All laptops will have the same methods that can be performed on them and properties (eg. MicroSoftSurface2.size = 10.6inch, MicroSoftSurface2.PowerOn)

# Advanced: Object-Oriented Programming

Methods may (or may not) come with extra settings called *arguments/parameters*. These parameters describe *how* an action is carried out. eg.

```
MacBookAir.Login User:=Irene
```

Consider real VBA code:

```
Worksheets("data").Copy After:=Worksheets("forms")  
Range("A1").Copy Range("A2")  
Application.ActiveDocument.SaveAs ("NewName.docx")  
ActiveWorkbook.SaveAs ("NewName.xlsm")
```

blue = object, green = method, orange = argument/parameter

# Excel Objects

Excel structures everything as a hierarchy of objects, and commands are done by running a method of an object.

An object may contain other objects as well as methods and properties. (eg. `ActiveCell` and `ActiveCell.Font` are both objects)

A dot "." is used as a separator between objects and subobjects, methods, and properties.

Examples:

- ▶ `Application` (Top-level object)
- ▶ `Workbook` – individual Excel file
- ▶ `Worksheet` - sheet in a workbook

```
Application.ActiveWorkbook.Worksheets("macro").Range("A1").Value
```

# Collections

- ▶ *Collections* are variables that store multiple data items.
- ▶ Data items can either be indexed (selected) by name or number.
  - ▶ example: `Worksheets("macro")`, `Worksheets(2)`

`Worksheets` is a collection as there may be multiple worksheets in the workbook.
- ▶ Collections are indexed starting with 1.
- ▶ Another example is 'Rows' which is a collection object containing all the rows of a Worksheet.

```
Worksheets("Sheet1").Rows(3).Delete
```

# Advanced: Object-Oriented Programming

Building on our first analogy:

- ▶ Class: Laptops
- ▶ Object: MacBookAir
  - ▶ MacBookAir
- ▶ Sub-object: hard drive,
  - ▶ MacBookAir.harddrive
- ▶ Property: size
- ▶ Values for example:
  - ▶ Laptops.MacBookAir.size=10in
  - ▶ Laptops.MacBookAir.harddrive.size = 500GB
- ▶ Method (action):
  - ▶ MacBookAir.Login

All objects of class Laptop should follow the same template. For example, they should all have a size property, a harddrive sub-object, etc.

# Advanced: Object-Oriented Programming

Now lets consider actual VBA code:

- ▶ Class: Range
- ▶ Object: `ActiveCell, Range("A1")`
- ▶ Sub-object: `ActiveCell.Font`
- ▶ Property: `ActiveCell.Font.Name`
- ▶ Value: `ActiveCell.Font.Name = "Times New Roman"`
- ▶ Method: `Range("A1").Select`

# Analogy

Here's an analogy that may help

- ▶ Object: nouns
  - ▶ man
- ▶ Property: adjectives (something that describe the object)
  - ▶ eg. height, usage `man.height`
- ▶ Method: verbs (action words)
  - ▶ eg. run, usage `man.run`
- ▶ Arguments/parameters: adverbs
  - ▶ eg. quickly, usage `man.run quickly`

While this doesn't fit in with the Grammar analogy, ...

- ▶ Class: Mammals
- ▶ Value: eg. `usage man.height = "5 foot 11"`

### Example

How many of the following statements are TRUE?

- ▶ A macro can be created without assigning it a shortcut key.
- ▶ A macro will record cursor movements.
- ▶ Macros saved in your Personal Macro Workbook can only be used in the a single workbook.

A) 0

B) 1

C) 2

D) 3



## Answer

How many of the following statements are TRUE?

- ▶ A macro can be created without assigning it a shortcut key.
- ▶ A macro will record cursor movements.
- ▶ Macros saved in your Personal Macro Workbook can only be used in the a single workbook.

A) 0

B) 1

C) 2

D) 3

## Answer

How many of the following statements are TRUE?

- ▶ A macro can be created without assigning it a shortcut key.
- ▶ A macro will record cursor movements.
- ▶ Macros saved in your Personal Macro Workbook can only be used in the a single workbook.

A) 0

B) 1

C) 2

D) 3

## Answer

How many of the following statements are TRUE?

- ▶ A macro can be created without assigning it a shortcut key.
- ▶ A macro will record cursor movements.
- ▶ Macros saved in your Personal Macro Workbook can only be used in the a single workbook.

A) 0

B) 1

C) 2

D) 3

## Answer

How many of the following statements are TRUE?

- ▶ A macro can be created without assigning it a shortcut key.
- ▶ A macro will record cursor movements.
- ▶ Macros saved in your Personal Macro Workbook can only be used in the a single workbook.

A) 0

B) 1

C) 2

D) 3

### Example

How many of the following statements are TRUE?

1. A method can have no parameters.
2. Two objects of the same class will have the same values for any given property.
3. Workbook is the top-level object in Excel.

A) 0

B) 1

C) 2

D) 3

## Answer

How many of the following statements are TRUE?

1. A method can have no parameters.
2. Two objects of the same class will have the same values for any given property.
3. Workbook is the top-level object in Excel.

A) 0

B) 1

C) 2

D) 3

## Answer

How many of the following statements are TRUE?

1. A method can have no parameters.
2. Two objects of the same class will have the same values for any given property.
3. Workbook is the top-level object in Excel.

A) 0

B) 1

C) 2

D) 3

## Answer

How many of the following statements are TRUE?

1. A method can have no parameters.
2. Two objects of the same class will have the same values for any given property.
3. Workbook is the top-level object in Excel.

A) 0

B) 1

C) 2

D) 3



## Answer

How many of the following statements are TRUE?

1. A method can have no parameters.
2. Two objects of the same class will have the same values for any given property.
3. Workbook is the top-level object in Excel.

A) 0

B) 1

C) 2

D) 3

# Challenge Try it: Create a Macro in VBE

## Question

Copy the MyFormat macro and edit to produce a new macro called RedUnderline that:

- ▶ Convert to plain text; that is, if the cell was bold or italics before, resets to not have bold or italics.
- ▶ Underlines the text in the cell.
- ▶ Makes the cell background red.

See hints on next page

## Challenge Try it: Create a Macro in VBE

### Hints:

- ▶ Underline property in Excel is `Font.Underline` and can set to constant `xlUnderlineStyleSingle`.
- ▶ Can change background colour with `Interior.Color` and set to `RGB(redValue, greenValue, blueValue)` where the colour values are numbers from 0 to 255. See more on this [here](#).
- ▶ Alternatively you could use the `ColorIndex` values. See more [here](#)

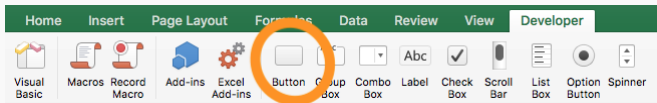
### Exercise:

Assign the MyFormat and RedUnderline macro to a single subroutine called Button2\_Click. Assign this new macro to a button in your excel file.

**Hint:** Within VBA, you can call macros by name. See instructions on the next page.

# Macro Buttons

1. Select the **Developer** tab and click



2. To prompt the Assign Macro popup window, click where you would like this button to appear on your worksheet.
3. Select **New** to open the VBA editor and Type the name of the macros you wish to assign in the body of the Sub.

```
Sub ButtonClick()  
    MyFormat  
    RedUnderline  
End Sub
```

# Creating Excel Functions

As mentioned previously, recording macros are restrictive in that they can't perform loops.

If we want to loop a procedure we can/need to code it up ourselves.

In addition, we will learn how we can create our own *functions* which we refer to as User Defined Functions (UDF)

But first things first, ...

# Introduction to Programming

An *algorithm* is a precise sequence of steps to produce a result. A program is an encoding of an algorithm in a *language* to solve a particular problem.

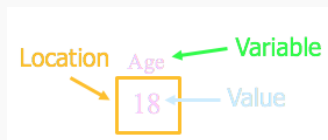
There are numerous languages that programmers can use to specify instructions. Each language has its different features, benefits, and usefulness.

We will start with Excel VBA but also study Python and R.

The goal is to understand fundamental programming concepts that apply to all languages.

# Variables

A *variable* is a name that refers to a location that stores a data value.



**IMPORTANT:** The *value* at a location can change using initialization or assignment.

*Assignment* using an sets the value of a variable.

Example:

```
► num = 10  
  num = Range("A1").Value
```



# Excel Variables

Variables increase code efficiency and readability.

Every variable in Excel has a *name* and a *data type*.

Data types include: Boolean (TRUE or FALSE), Currency, Date, Double, Integer, Long, Object, String, Variant (any type)

- ▶ See Part 2 of [this \(click me\)](#) tutorial for more on this.

Example:

```
Dim num As Integer
```

The **Dim** keyword *declares* the variable (named `num`) as an integer.

**Sidenote:** A declaration may be optional or required, depending on the programming language.

# Excel Variables

There are number of different declaration keywords we can use. for example:

**Const** for setting constants

**Static** for setting constants

**Public** for using across macros (default declaration for subs)

**Private** variable is limited to the current module

**Public Const** for setting constants across macros

...

For example, to declare a constant in VBA:

```
Const MaxCount = 5000
```

# Excel Variables - Strings

We can also declare *strings* i.e. a series of characters.

```
Dim MyString As String
```

```
MyString = "This is an example of the String"
```

We can declare and assign strings all in one step:

```
Dim MyString As String = "ABCDE"
```

Notice that a Char type is restricted to be a single character. For example:

```
Dim MyString As String = "ABCDE"
```

```
Dim myChar Char
```

```
' The value of myChar is "D".
```

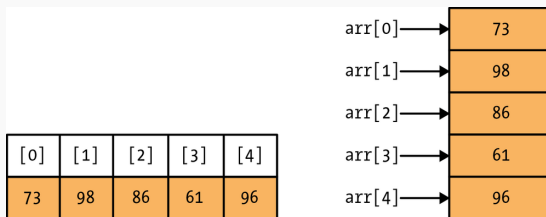
```
myChar = myString.Chars(3)
```

See more about this topic [here](#).

## Excel Variables - Arrays

We can also define *arrays* which are data structure consisting of a collection of elements (values or variables).

Each element is identified by an *index* or *key*.



**Figure:** Source: [O'Reilly Media](#)

**Warning:** Depending on the programming language, sometimes this index will begin from 0, other times it will begin from 1 (in VBA we are able choose either).

## Excel Variables - Arrays

*Arrays* are declared the same way as other variables, the only difference is that now, we need to specify its length.

By default, arrays are indexed starting from 0. Hence

```
Dim Class_List(150) As String
```

would be recognized as an array of 150 variables, which are indexed from 0 to 149.

```
Class_List(0) = "Irene Vrbik"
```

```
Class_List(1) = "Peter Gregory"
```

If you want your index to start from 1 rather than 0, use:

```
Dim MidtermGrades (1 To 150) As Integer
```

```
MidtermGrades(1) = 89
```

```
MidtermGrades(2) = 91
```

For more on this subject click [here](#).

## Option Explicit Option

Although VBA does not require us to declare variables, it is good coding practice to do so.

By default, all undeclared variable in Excel will have the Variant type (which can hold Dates, Floating Point Numbers or Strings of Characters)

If you want to prevent default declarations, you can always set the "Option Explicit" option for force you to explicitly declare each variable. Read more about it [here](#).

# Variables Question

## Question

How many of the following statements are TRUE?

1. A variable name cannot change during a program.
2. A variable value cannot change during a program.
3. A collection is a variable that can store multiple data items.

A) 0

B) 1

C) 2

D) 3

# Variables Question

## Question

How many of the following statements are TRUE?

1. A variable name cannot change during a program.
2. A variable value cannot change during a program.
3. A collection is a variable that can store multiple data items.

A) 0

B) 1

C) 2

D) 3



# Variables Question

## Question

How many of the following statements are TRUE?

1. A variable name cannot change during a program.
2. A variable value cannot change during a program.
3. A collection is a variable that can store multiple data items.

A) 0

B) 1

C) 2

D) 3

# Variables Question

## Question

How many of the following statements are TRUE?

1. A variable name cannot change during a program.
2. A variable value cannot change during a program.
3. A collection is a variable that can store multiple data items.

A) 0

B) 1

C) 2

D) 3

# Variables Question

## Question

How many of the following statements are TRUE?

1. A variable name cannot change during a program.
2. A variable value cannot change during a program.
3. A collection is a variable that can store multiple data items.

A) 0

B) 1

C) 2

D) 3

# Variables Question

## Question

How many of the following statements are TRUE?

1. A value in a collection can be retrieved by name or by index starting from 0.
2. In Excel, variables can be declared using Dim.
3. In Excel, variables are declared with a data type.

A) 0

B) 1

C) 2

D) 3

# Variables Question

## Question

How many of the following statements are TRUE?

1. A value in a collection can be retrieved by name or by index starting from 0.
2. In Excel, variables can be declared using Dim.
3. In Excel, variables are declared with a data type.

A) 0

B) 1

C) 2

D) 3

# Variables Question

## Question

How many of the following statements are TRUE?

1. A value in a collection can be retrieved by name or by index starting from 0.
2. In Excel, variables can be declared using Dim.
3. In Excel, variables are declared with a data type.

A) 0

B) 1

C) 2

D) 3

# Variables Question

## Question

How many of the following statements are TRUE?

1. A value in a collection can be retrieved by name or by index starting from 0.
2. In Excel, variables can be declared using Dim.
3. In Excel, variables are declared with a data type.

A) 0

B) 1

C) 2

D) 3

# Variables Question

## Question

How many of the following statements are TRUE?

1. A value in a collection can be retrieved by name or by index starting from 0.
2. In Excel, variables can be declared using Dim.
3. In Excel, variables are declared with a data type.

A) 0

B) 1

C) 2

D) 3



# Decisions

Decisions allow the program to perform different actions in certain conditions.

Logical operators: AND, OR, NOT

Example decision syntax:

- ▶ If condition Then  
    statement  
End If
  
- ▶ If condition Then  
    statement if true  
Else  
    statement if false  
End If

# Question: Decisions

## Example

What is the output of the following code:

```
Sub TryIf()  
    Dim num As Integer  
    num = 20  
    If num > 10 Then  
        Debug.Print num * 5  
    Else  
        Debug.Print num * 2  
    End If  
End Sub
```

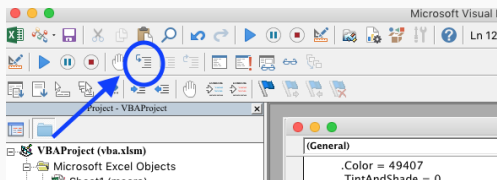
A) 100

B) 40

C) 20

D) error

We can verify this result by running the TryIf sub found in the **04VBA.xlsm** file. To step through your code in , navigate to the TryIf() function and press **F8** on a windows or press this button:



Alternatively you could type

TryIf

into the Immediate window and press **ENTER** .

# Question: Decisions

## Question

What is the output of the following code:

```
Sub TryIf()  
    Dim num As Integer  
    num = 20  
    If num > 10 Then  
        Debug.Print num * 5  
    Else  
        Debug.Print num * 2  
    End If  
End Sub
```

A) 100

B) 40

C) 20

D) error

# Try It: Decisions

## Question

Create a method called EchoDecision that asks user a Yes and No question and outputs a message either "Yes" or "No" depending on what they chose.

```
Sub EchoDecision()  
    Dim answer As Integer  
    answer = MsgBox("Pick Yes or No!", vbYesNo)  
    ' answer will either be vbYes or vbNo  
    ' Use debug.print to output "Yes" or "No"  
End Sub
```

MsgBox function in VBA displays a message in a window and waits for click on a button.

Debug.Print is telling VBA to print that information in the Immediate Window.

# Loops and Iteration

A *loop* repeats a set of statements multiple times until some condition is satisfied.

Each time a loop is executed is called an *iteration*.

A *for loop* repeats statements a given number of times.

Example:

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

The above will print out the numbers 1 to 5 *inclusively*.

# Question: loops

## Question

How many numbers are printed with this loop?

```
Sub TestFor()  
    Dim i As Integer  
  
    For i = 0 To 10  
        Debug.Print i  
    Next i  
End Sub
```

A) 11

B) 10

C) 0

D) error

# Question: loops

## Question

How many numbers are printed with this loop?

```
Sub TestFor()  
    Dim i As Integer  
  
    For i = 0 To 10  
        Debug.Print i  
    Next i  
End Sub
```

A) 11

B) 10

C) 0

D) error



# UDF General Syntax

A user-defined function is your own Excel function that can be used in cell formulas like built-in functions.

A UDF must return a number, string, array, or Boolean. We declare the data type of the returned value directly after any argument.

Notice the data types of each argument using the As keyword.

The general syntax for creating a UDF is:

```
Function FunctionName(arg1 As DataType, ...) As DataType
    'comments
    [Statements]
End Function
```

# User-Defined Functions (UDFs)

Example: UDF `doubleIt` will double the input argument.

```
' UDF expect a number to double
Function doubleIt(num As Double) As Double
  ' To return a value, assign the value to a
  ' variable with the same name as the function
  doubleIt = num * 2
End Function
```

**N.B** The value returned must be assigned to a variable with the same name as the UDF (this variable is implicitly declared in the first line).

- ▶ That is, `doubleIt` stores the value returned by our function `doubleIt`.
- ▶ N.B. If the function data type is not declared, it is assumed to be `Variant`

## Sub vs. UDFs

Unlike a Sub, a UDF cannot change the Excel environment including the current cells or other cells (e.g. change formatting).

Unlike a UDF, a Sub procedure does not return a result, nor can they be typed directly into a Worksheet in Excel

- ▶ eg. the function SUM() returns a result of summing a group of numbers
- ▶ the Sub might carry out actions like formatting a set of cells

Most of the macros you write in VBA are Sub procedures.

Both VBA Sub procedures and UDF procedures can take arguments but they are not essential.

## UDF Example: Sum Cells by Background Colour

The function `SumColor` works like the `SUM()` function, only now, it will only consider the cells having a certain background colour.

This function takes two arguments.

1. The first specifies where in the table are we looking for values to add (i.e. a range of cells).
2. The second argument specifies the background colour.

Colours can be index by integers. See [this](#) reference for more on colour indexing. For example:

- ▶ Black = 1, White = 2, Red = 3, . . . , Yellowy Orange = 44, Light Orange = 45, Another Orange = 46

---

```
' Sums all the cells with the same color
Function SumColor(RangeToSum As Range, ColorID As Integer) As Long
    Dim ColorCell As Range
    Dim result As Long

    ' Loop through each cell in the range.
    For Each ColorCell In RangeToSum
        If ColorCell.Interior.ColorIndex = ColorID Then
            result = result + ColorCell.Value
        End If
    Next ColorCell

    SumColor = result
End Function
```

Let's use the function `SumColor` to find the sum of all the cells that have been formatted in orange by the macro `MyFormat`.

We will need to know the colour index for the colour used in this macro for the input for `ColorID`.

To obtain that, we can either go to the VBA code, go to the colour indexing [link](#) and look for orange (which might be hard to distinguish by eye), or obtain it from the Immediate Window by typing

```
?Range("A6").Interior.ColorIndex
```

assuming cell **A6** has been formatted using `MyFormat`.

The screenshot shows an Excel spreadsheet with the following data:

	A	B	C	D	E
1				<b>UDFs:</b>	
2	Number	Letter		Double It	Sum Colour Count
3	1	A			
4	2	B	H132Fun42!		
5	3	D			
6	4	E			
7	5	F			
8					
9	Color	ColorIndex	Sum of Cells with an background colour = Color		
10	Orange	46	4 =SumColor(A3:A7,B10)		
11	Red	3	9 =SumColor(A3:A7,B11)		
12					
13					

The status bar at the bottom indicates the active cell is D12, and the zoom level is 171%.

### Question:

Create a UDF called CountNumbers that will return a count of the number of digits (ie 0 to 9) in a string. eg. applying this function to the string Data301 should return 3 (since this course code contains 3 numbers).

To build this function use:

- ▶ `Len()` to count the length of the string, and
- ▶ `Mid()` to extract a given number of characters from the middle of a supplied text string. For example, `=MID("apple",2,3)` returns "ppl" (start at index 2, extract a string of length 3).
- ▶ `IsNumeric()` is a VBA function that returns TRUE if the expression is a valid number.

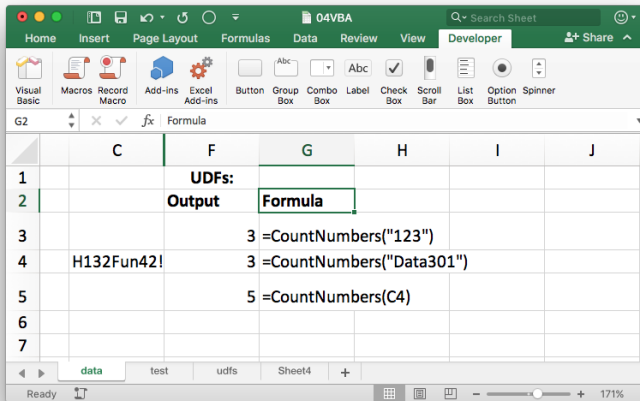


# VBA functions vs Excel Functions

- ▶ VBA functions can be used in any program that supports VBA (including Microsoft Word and Access). List of them [here](#).
- ▶ Worksheet functions are specific to Excel and can be used using `Application.WorksheetFunction.[function name]` in VBA code or simply by calling them by name (with any necessary arguments) in a cell, eg. `MID("example", 1, 2)`

Read more about the differences between VBA functions and Excel Functions [here](#). Note that:

- ▶ `IsNumber()` is a worksheet function to check if a value is *stored* as a number, whereas `IsNumeric` checks if a value can be *converted* to a number.
- ▶ `Mid()` and `len()` are both Worksheet functions (WS) *and* VBA functions (VBA)



# Conclusion

*Microsoft Excel VBA* allows for automating tasks in Excel and provides a full programming environment for data analysis.

*Macros* record a set of actions so they can be easily executed again.  
**Be aware of security risks when using macros.**

The *Visual Basic Editor (VBE)* is a complete integrated development environment for editing macros and *user-defined functions*.controls that dynamically respond to events.

Excel VBA uses *object-oriented programming* that structures code as object, classes, methods, and properties. A developer can control and automate everything with Excel using VBA.

# Objectives

- ▶ List some reasons to use Excel VBA
- ▶ Define macro and explain the benefit of using macros
- ▶ Be able to record and execute a macro
- ▶ Explain the security issues with macros and how Excel deals with them
- ▶ List and explain the use of the four main windows of the Visual Basic Editor
- ▶ Explain the role of the object browser
- ▶ Explain and use the WITH statement syntax
- ▶ Be able to write simple macros using the VBE
- ▶ Define: algorithm, program, language
- ▶ Define: object-oriented programming, object, class, property, method
- ▶ Understand and use dot-notation
- ▶ Use the Range object to select a group of cells

# Objectives (2)

- ▶ Define: variable, value, location
- ▶ Create and use Excel variables
- ▶ Explain how a collection is different from a typical variable
- ▶ Use If/Then/Else syntax to make decisions
- ▶ Use For loop for repetition
- ▶ Create user-defined functions and use them in formulas
- ▶ List some typical user interface controls
- ▶ Understand that Excel allows for forms and controls to be added to a worksheet which respond to events