Programming 1: VBA and Python

Juan F. Imbet *Ph.D.*

Paris Dauphine University

M203 (M1)

- **Session 1** (03/09) Introduction to Programming and VBA, basis, variables and operators.
- **Session 2** (10/09) Control flow, modules and functions.
- **Session 3** (20/09) Object Oriented Programming in VBA
- **Session 4** (01/10) Programming with Solver
- **Session 5** (08/10) Introduction to Python: Getting Started and Introduction to Programming.
- **Session 6** (15/10) Intermediate Concepts.
- **Session 7** (22/10) Linear Algebra and Optimization.
- **Session 8** (29/10) Data Frames
- **Session 9** (12/11) Object Oriented Programming.
- **Session 10** (19/11) API and Web Scraping + Project Explanation.

Grading

- 90% Final Project
- 10% Participation

Note: As of M203 policies, attendance is mandatory.

Visual Basic for Applications (VBA)

- VBA is a programming language that is used to automate tasks in Microsoft Office applications.
- VBA is a subset of Visual Basic (VB), focusing on **macros** and **automation**.
- You can use VBA inside of Outlook, Word, Excel, Access, and PowerPoint.
- We will focus on Excel VBA as it is the most common use case.
- Although some other languages are becoming more and more popular, VBA is still widely used in the industry, and this is unlikely to change in the near future.

What happens in the back?

- Code written in VBA is **compiled** to Microsoft P-Code (Pseudo Code), a proprietary intermediate language.
- This P-Code is then executed by the host application (Excel, Word, etc.).
- It is not the most efficient language, but it is very flexible and easy to use.

Characteristics of VBA

- General Purpose
- Interpreted
- Event-Driven
- Object-Oriented
- High Level

General Purpose

- VBA is a general-purpose programming language, meaning that it can be used to solve a wide range of problems. It is only limited by the use of Office applications.
- Automated Financial Models and Dashbords.
- Custom ERP Systems
- Interactive Games
- In 2003 a flight simulator was created in Excel using VBA.

Interpreted

- VBA is an interpreted language, meaning that the code is executed line by line. This is different from compiled languages like C++ or Java, where the code is compiled into machine code before execution.

Event-Driven

- VBA is an event-driven language, meaning that code is executed in response to events. For example, you can write code that is executed when a user clicks a button or opens a workbook.

Object-Oriented

- VBA is an object-oriented language, meaning that it uses objects to represent data and functionality. Objects can be manipulated using methods and properties.

High Level

- VBA is a high-level language, meaning that it is closer to human language than machine language. This makes it easier to read and write code (more on this when we talk about Python).

SETTING UP YOUR ENVIRONMENT

You need

- Microsoft Excel
- VS Code (we will not use the standard VBA editor)
- VBA extension for VS Code
- Python (Anaconda Distribution)
- `xlwings` library for Python that allows you to interact with Excel.

Install `xlwings`

```
pip install xlwings
pip install watchgod
```

Open an excel file and save it as a macro-enabled workbook, call it `hello_world.xlsm`.

VBA in VS Code

- Excel files are really zip files with a different extension.
- Enable Trust Access to the VBA Project Object Model:
    - Open Excel.
    - Go to File > Options.
    - In the Excel Options window, select Trust Center on the left.
    - Click on the Trust Center Settings button.
    - In the Trust Center, select Macro Settings.
    - Check the box that says Trust access to the VBA project object model.
    - Click OK to close the Trust Center and then OK to close the Excel Options.
- Click on the View tab and then click on the Macros button, create a new macro called `my_first_macro()`.

`xlwings`

- Directly in vs code, open a prompt and type

```
xlwings vba edit -f hello_world.xlsm
```

```
xlwings version: 0.29.1

This will affect the following workbook/folder:

* hello_world.xlsm
* C:\Users\jfimb\Dropbox\jfimbett.github.io\teaching\vba_python

Proceed? [Y/n] y
NOTE: Deleting a VBA module here will also delete it in the VBA editor!
Watching for changes in hello_world.xlsm (silent mode)...(Hit Ctrl-C to stop)
```

- Do not close excel, keep the macro editor open.
- In the file explorer you should see a file called Module1.bas
- Edit the file and write the following code:

```
Attribute VB_Name = "Module1"
Sub my_first_macro()
' Display a message box
    MsgBox "Hello, world!"
End Sub
```

- The first line does not appear in the VBA Editor, but VS Code identifies it.
- You still need to run the macro from the Excel file.

Let's start coding!

Types

Create a new macro to explote the types in VBA.

```
Attribute VB_Name = "Module2"
Sub VariableTypesExamplesToTable()
...
End Sub
```

Code in VBA most times has to be enclosed in a `Sub` or `Function` block. `Sub` stands for subroutine and `Function` is used to return a value.

```vba
' Boolean - True or False
Dim isComplete As Boolean
isComplete = True

' Byte - Integer from 0 to 255
Dim byteValue As Byte
byteValue = 255

' Integer - Integer from -32,768 to 32,767
Dim smallNumber As Integer
smallNumber = 12345

' Long - Integer from -2,147,483,648 to 2,147,483,647
Dim largeNumber As Long
largeNumber = 1234567890
```

```
' Single - Single-precision floating-point (approximately -3.4E38 to 3.4E38)
Dim singlePrecisionNumber As Single
singlePrecisionNumber = 1234.56

' Double - Double-precision floating-point (approximately -1.7E308 to 1.7E308)
Dim doublePrecisionNumber As Double
doublePrecisionNumber = 1234567.89

' Currency - Fixed-point with 4 decimal places
' (approximately -922,337,203,685,477.5808 to 922,337,203,685,477.5807)
Dim currencyValue As Currency
currencyValue = 12345.6789

' Decimal - Floating-point number (exact values, used for financial calculations)
Dim decimalValue As Variant
decimalValue = CDec(12345678.1234)
```

```
' Date - Date and time
Dim currentDate As Date
currentDate = Now

' String - Text
Dim name As String
name = "John Doe"

' Variant - Can hold any type of data, default data type if not specified
Dim unknownType As Variant
unknownType = "Can hold any type"

' Object - Can refer to any object
Dim ws As Worksheet
Set ws = ThisWorkbook.Sheets.Add(After:=ThisWorkbook.Sheets(ThisWorkbook.Sheets.Count))
ws.Name = "Variable Types"
```

```vba
' Range - Special type for referring to Excel ranges
Dim rng As Range
Set rng = ws.Range("A1:A10")

' Array - A collection of variables of the same type
Dim numbersArray(1 To 5) As Integer
numbersArray(1) = 10
numbersArray(2) = 20
numbersArray(3) = 30
numbersArray(4) = 40
numbersArray(5) = 50

' Object - Example with a custom object
Dim dict As Object
Set dict = CreateObject("Scripting.Dictionary")
dict.Add "Key1", "Value1"
```

Display the information in a table

```
' Preparing headers for the table
ws.Cells(1, 1).Value = "Variable Type"
ws.Cells(1, 2).Value = "Variable Name"
ws.Cells(1, 3).Value = "Value"
```

Print the information in the table

```
' Filling in the table with data
ws.Cells(2, 1).Value = "Boolean"
ws.Cells(2, 2).Value = "isComplete"
ws.Cells(2, 3).Value = isComplete

ws.Cells(3, 1).Value = "Byte"
ws.Cells(3, 2).Value = "byteValue"
ws.Cells(3, 3).Value = byteValue

ws.Cells(4, 1).Value = "Integer"
ws.Cells(4, 2).Value = "smallNumber"
ws.Cells(4, 3).Value = smallNumber

ws.Cells(5, 1).Value = "Long"
ws.Cells(5, 2).Value = "largeNumber"
ws.Cells(5, 3).Value = largeNumber
```

```
ws.Cells(6, 1).Value = "Single"
ws.Cells(6, 2).Value = "singlePrecisionNumber"
ws.Cells(6, 3).Value = singlePrecisionNumber

ws.Cells(7, 1).Value = "Double"
ws.Cells(7, 2).Value = "doublePrecisionNumber"
ws.Cells(7, 3).Value = doublePrecisionNumber

ws.Cells(8, 1).Value = "Currency"
ws.Cells(8, 2).Value = "currencyValue"
ws.Cells(8, 3).Value = currencyValue

ws.Cells(9, 1).Value = "Decimal"
ws.Cells(9, 2).Value = "decimalValue"
ws.Cells(9, 3).Value = decimalValue
```

```
ws.Cells(10, 1).Value = "Date"
ws.Cells(10, 2).Value = "currentDate"
ws.Cells(10, 3).Value = currentDate

ws.Cells(11, 1).Value = "String"
ws.Cells(11, 2).Value = "name"
ws.Cells(11, 3).Value = name

ws.Cells(12, 1).Value = "Variant"
ws.Cells(12, 2).Value = "unknownType"
ws.Cells(12, 3).Value = unknownType

ws.Cells(13, 1).Value = "Array (Element 1)"
ws.Cells(13, 2).Value = "numbersArray(1)"
ws.Cells(13, 3).Value = numbersArray(1)

ws.Cells(14, 1).Value = "Object (Dictionary)"
ws.Cells(14, 2).Value = "dict(""Key1"")"
ws.Cells(14, 3).Value = dict("Key1")
```

Formatting Ranges (more on this later)

```
' Formatting the table
With ws.Range("A1:C14")
    .Font.Bold = True
    .Borders.LineStyle = xlContinuous
    .Columns.AutoFit
End With
```

Variable Type	Variable Name	Value
Boolean	isComplete	TRUE
Byte	byteValue	255
Integer	smallNumber	12345
Long	largeNumber	1234567890
Single	singlePrecisionNumber	1234.560059
Double	doublePrecisionNumber	1234567.89
Currency	currencyValue	£12,345.68
Decimal	decimalValue	12345678.12
Date	currentDate	01/09/2024 18:39

Communicating with the workbook

- You can use the `MsgBox` function to display a message box to the user.
- You can use the `InputBox` function to get input from the user.
- You can use the `Cells` property of a `Worksheet` object to read from and write to cells in a worksheet.
- You can access an Excel function using the `Application` object.

Simple messages

```
Sub SimpleMessages()
    MsgBox "This is a simple message box"
    MsgBox "This is a simple message box with a title", vbInformation, "Title"
    MsgBox "This is a simple message box with a title and a Yes/No button", vbYesNo, "Title"
End Sub
```

What did the user click?

```
Sub SimpleMessagesUserClick()
    Dim response As VbMsgBoxResult
    response = MsgBox("Do you want to continue?", vbYesNo, "Continue?")
    If response = vbYes Then
        MsgBox "You clicked Yes"
    Else
        MsgBox "You clicked No"
    End If
End Sub
```

Retrieve information from the user

```
Sub RetrieveInformation()
    Dim name As String
    name = InputBox("What is your name?", "Name")
    MsgBox "Hello, " & name
End Sub
```

Recall that the `&` operator is used to concatenate strings.

Retrieve information from an Excel cell

```
Sub RetrieveInformationFromCell()
    Dim value As Variant
    value = ThisWorkbook.Sheets("Sheet2").Range("A1").Value
    MsgBox "The value in cell A1 is " & value
End Sub
```

Write single values to a cell

```
Sub WriteToCell()
    ThisWorkbook.Sheets("Sheet1").Range("A1").Value = "Hello, world!"
End Sub
```

Write an array to a range

```
Sub WriteArrayToRange()
    Dim values As Variant
    values = Array("One", "Two", "Three", "Four", "Five")
    ThisWorkbook.Sheets("Sheet1").Range("A1:A5").Value = Application.Transpose(values)
End Sub
```

Use Excel functions

```
Sub UseExcelFunction()
    ' Initialize an array with values
    Dim valuesArray As Variant
    valuesArray = Array(1, 2, 3, 4, 5)

    ' Loop through the range and assign the array values to the cells
    Dim i As Integer
    For i = 1 To 5
        ThisWorkbook.Sheets("Sheet1").Cells(i, 1).Value = valuesArray(i - 1)
    Next i

    ' Calculate the sum of the range
    Dim result As Variant
    result = Application.WorksheetFunction.Sum(ThisWorkbook.Sheets("Sheet1").Range("A1:A5"))

    ' Display the result in a message box
    MsgBox "The sum of the values in A1:A5 is " & result
End Sub
```

Control Flow

- Control flow statements allow you to control the flow of execution in your code.
- You can use control flow statements to make decisions, loop through code, and exit code early.
- The most common control flow statements are `If`, `ElseIf`, `Else`, `For`, `Do`, and `While`.

If Statements

- The `If` statement allows you to execute code conditionally, the `If` statement is terminated with an `End If` statement, the command `Then` states what happens if the condition is met.

```
Sub IfStatement()
    Dim value As Integer
    value = 10

    If value > 5 Then
        MsgBox "The value is greater than 5"
    End If
End Sub
```

If Else Statements

- The `Else` statement allows you to execute code if the main condition is not met.

```
Sub IfElseStatement()
    Dim value As Integer
    value = 3

    If value > 5 Then
        MsgBox "The value is greater than 5"
    Else
        MsgBox "The value is less than or equal to 5"
    End If
End Sub
```

Else If Statements

- The `ElseIf` statement allows you to check multiple conditions.

```
Sub ElseIfStatement()
    Dim value As Integer
    value = 5

    If value > 5 Then
        MsgBox "The value is greater than 5"
    ElseIf value < 5 Then
        MsgBox "The value is less than 5"
    Else
        MsgBox "The value is equal to 5"
    End If
End Sub
```

For Loops

- The `For` loop allows you to execute code a specific number of times. It is useful when you know the number of iterations in advance, as well as when you deal with arrays.

```
Sub ForLoop()
    Dim N As Integer
    N = 10
    Dim myArray() As Integer
    ReDim myArray(N)
    Dim i As Integer
    For i = 0 To N - 1
        myArray(i) = i  + 1
    Next i
    Dim sum As Integer
    sum = 0
    For i = 0 To N -1
        sum = sum + myArray(i)
    Next i
    MsgBox "The sum of the numbers from 1 to " & N & " is " & sum
End Sub
```

While Loops

- The `While` loop allows you to execute code while a condition is true. It is useful when you do not know the number of iterations in advance.

```
Sub WhileLoop()
    Dim N As Integer
    N = 10
    Dim myArray() As Integer
    ReDim myArray(N)
    Dim i As Integer
    i = 0
    While i < N
        myArray(i) = i + 1
        i = i + 1
    Wend
    Dim sum As Integer
    sum = 0
    i = 0
    While i < N
        sum = sum + myArray(i)
        i = i + 1
    Wend
    MsgBox "The sum of the numbers from 1 to " & N & " is " & sum
End Sub
```

Stop the execution of a loop in advance

- You can use the `Exit` statement to stop the execution of a loop in advance.
- You can specify the environment you want to exit from, for example, `Exit For` or `Exit While`, `Exit Function`, `Exit Sub`.
- Example, find the first even number in an array.

```
Sub ExitLoop()
    Dim N As Integer
    N = 10
    Dim myArray() As Integer
    ReDim myArray(N)
    Dim i As Integer
    For i = 0 To N - 1
        myArray(i) = i + 1
        If myArray(i) Mod 2 = 0 Then
            MsgBox "The first even number is " & myArray(i)
            Exit For
        End If
    Next i
End Sub
```

Stop the execution of a function or a subroutine

- You can use the `Exit` statement to stop the execution of a function or a subroutine in advance.
- Determine if a number is prime.

Functions

- Functions are used to return a value.
- The `Function` statement is used to define a function.
- The `End Function` statement is used to terminate a function.
- The `Function` statement is followed by the name of the function and a list of parameters in parentheses.
- The `Function` statement can also include a return type.

Return the sum of two numbers

```
Function AddNumbers(x As Double, y As Double) As Double
    AddNumbers = x + y
End Function
```

- The variable that holds the return value has the same name as the function.