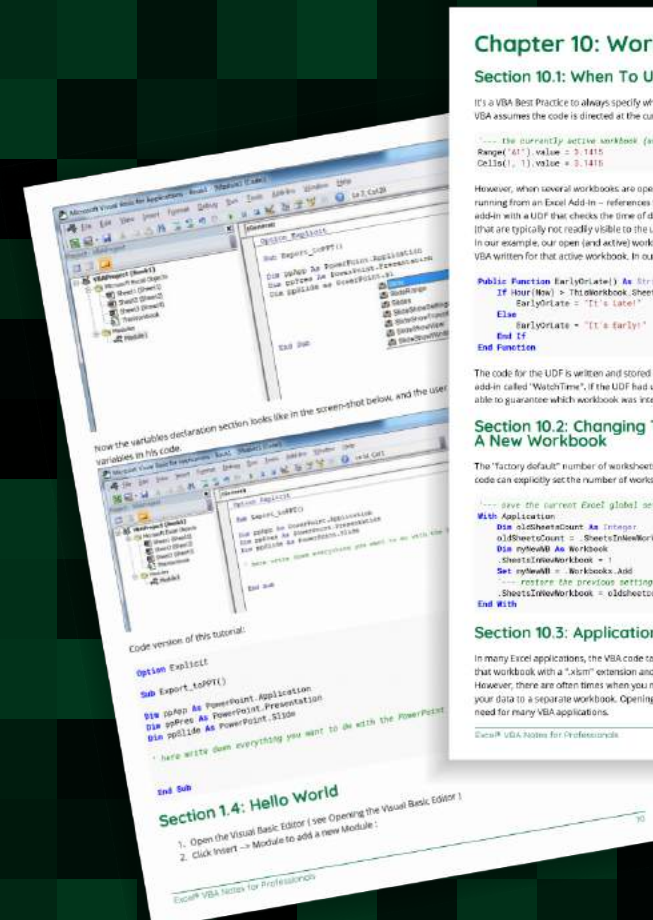


# Excel® VBA

## Notes for Professionals

专业人士笔记Excel® VBA  
专业人士笔记



100多页  
专业提示和技巧

100+ pages  
of professional hints and tricks

# 目录

关于	1
第1章：Excel VBA入门	2
第1.1节：打开Visual Basic编辑器（VBE）	3
第1.2节：声明变量	5
第1.3节：添加新的对象库引用	6
第1.4节：你好，世界	10
第1.5节：Excel对象模型入门	12
第2章：数组	16
第2.1节：动态数组（数组调整大小和动态处理）	16
第2.2节：填充数组（添加值）	16
第2.3节：锯齿状数组（数组的数组）	17
第2.4节：检查数组是否已初始化（是否包含元素）	17
第2.5节：动态数组【数组声明，调整大小】	17
第3章：条件语句	19
第3.1节：If语句	19
第4章：区域和单元格	21
第4.1节：引用单个单元格的方法	21
第4.2节：创建区域	21
第4.3节：偏移属性	23
第4.4节：将单元格引用保存到变量中	23
第4.5节：如何转置区域（水平到垂直及反之）	23
第5章：命名区域	25
第5.1节：定义命名区域	25
第5.2节：在VBA中使用命名区域	25
第5.3节：使用名称管理器管理命名区域	26
第5.4节：命名范围数组	28
第6章：合并单元格/范围	29
第6.1节：使用合并单元格/范围前请三思	29
第7章：定位范围内的重复值	30
第7.1节：在范围内查找重复项	30
第8章：用户自定义函数（UDF）	32
第8.1节：允许整列引用且无惩罚	32
第8.2节：计算范围内的唯一值	33
第8.3节：用户自定义函数（UDF）- 你好，世界	33
第9章：使用VBA进行条件格式设置	36
第9.1节：FormatConditions.Add	36
第9.2节：移除条件格式	37
第9.3节：FormatConditions.AddUniqueValues	37
第9.4节：FormatConditions.AddTop10	38
第9.5节：FormatConditions.AddAboveAverage	38
第9.6节：FormatConditions.AddIconSetCondition	38
第10章：工作簿	41
第10.1节：何时使用ActiveWorkbook和ThisWorkbook	41
第10.2节：更改新工作簿中默认的工作表数量	41
第10.3节：应用工作簿	41
第10.4节：打开（新）工作簿，即使它已经打开	42

# Contents

About	1
Chapter 1: Getting started with Excel VBA	2
Section 1.1: Opening the Visual Basic Editor (VBE)	3
Section 1.2: Declaring Variables	5
Section 1.3: Adding a new Object Library Reference	6
Section 1.4: Hello World	10
Section 1.5: Getting Started with the Excel Object Model	12
Chapter 2: Arrays	16
Section 2.1: Dynamic Arrays (Array Resizing and Dynamic Handling)	16
Section 2.2: Populating arrays (adding values)	16
Section 2.3: Jagged Arrays (Arrays of Arrays)	17
Section 2.4: Check if Array is Initialized (If it contains elements or not)	17
Section 2.5: Dynamic Arrays [Array Declaration, Resizing]	17
Chapter 3: Conditional statements	19
Section 3.1: The If statement	19
Chapter 4: Ranges and Cells	21
Section 4.1: Ways to refer to a single cell	21
Section 4.2: Creating a Range	21
Section 4.3: Offset Property	23
Section 4.4: Saving a reference to a cell in a variable	23
Section 4.5: How to Transpose Ranges (Horizontal to Vertical & vice versa)	23
Chapter 5: Named Ranges	25
Section 5.1: Define A Named Range	25
Section 5.2: Using Named Ranges in VBA	25
Section 5.3: Manage Named Range(s) using Name Manager	26
Section 5.4: Named Range Arrays	28
Chapter 6: Merged Cells / Ranges	29
Section 6.1: Think twice before using Merged Cells/Ranges	29
Chapter 7: Locating duplicate values in a range	30
Section 7.1: Find duplicates in a range	30
Chapter 8: User Defined Functions (UDFs)	32
Section 8.1: Allow full column references without penalty	32
Section 8.2: Count Unique values in Range	33
Section 8.3: UDF - Hello World	33
Chapter 9: Conditional formatting using VBA	36
Section 9.1: FormatConditions.Add	36
Section 9.2: Remove conditional format	37
Section 9.3: FormatConditions.AddUniqueValues	37
Section 9.4: FormatConditions.AddTop10	38
Section 9.5: FormatConditions.AddAboveAverage	38
Section 9.6: FormatConditions.AddIconSetCondition	38
Chapter 10: Workbooks	41
Section 10.1: When To Use ActiveWorkbook and ThisWorkbook	41
Section 10.2: Changing The Default Number of Worksheets In A New Workbook	41
Section 10.3: Application Workbooks	41
Section 10.4: Opening A (New) Workbook, Even If It's Already Open	42

第10.5节：保存工作簿而不提示用户	43
<b>第11章：在VBA中使用Excel表格</b>	44
第11.1节：实例化ListObject	44
第11.2节：操作列表行 / 列	44
第11.3节：将Excel表转换为普通区域	44
<b>第12章：遍历活动工作簿中的所有工作表</b>	45
第12.1节：获取活动工作簿中所有工作表名称	45
第12.2节：遍历文件夹中所有文件的所有工作表	45
<b>第13章：使用工作表对象而非工作表页对象</b>	47
第13.1节：打印第一个对象的名称	47
<b>第14章：查找工作表中最后使用的行或列的方法</b>	48
第14.1节：查找列中最后一个非空单元格	48
第14.2节：查找工作表中最后一个非空行	48
第14.3节：查找工作表中最后一个非空列	49
第14.4节：查找行中最后一个非空单元格	50
第14.5节：获取范围内最后一个单元格的行号	50
第14.6节：使用命名范围查找最后一行	50
第14.7节：范围.CurrentRegion中的最后一个单元格	51
第14.8节：查找工作表中最后一个非空单元格 - 性能（数组）	51
<b>第15章：在活动工作表中使用组合框创建下拉菜单</b>	54
第15.1节：示例2：未包含的选项	54
第15.2节：吉米·亨德里克斯菜单	55
<b>第16章：文件系统对象</b>	57
第16.1节：文件、文件夹、驱动器是否存在	57
第16.2节：基本文件操作	57
第16.3节：基本文件夹操作	58
第16.4节：其他操作	58
<b>第17章：数据透视表</b>	60
第17.1节：向数据透视表添加字段	60
第17.2节：创建数据透视表	60
第17.3节：数据透视表范围	63
第17.4节：格式化数据透视表数据	63
<b>第18章：绑定</b>	64
第18.1节：早期绑定与晚期绑定	64
<b>第19章：自动筛选；用途与最佳实践</b>	66
第19.1节：智能筛选！	66
<b>第20章：应用程序对象</b>	70
第20.1节：简单的应用程序对象示例：显示Excel和VBE版本	70
第20.2节：简单的应用程序对象示例：最小化Excel窗口	70
<b>第21章：图表与制图</b>	71
第21.1节：使用区域和固定名称创建图表	71
第21.2节：创建空白图表	72
第21.3节：通过修改SERIES公式创建图表	73
第21.4节：将图表排列成网格	75
<b>第22章：CustomDocumentProperties的实际应用</b>	79
第22.1节：组织新的发票号码	79
<b>第23章：通过VBA集成PowerPoint</b>	82
第23.1节：基础知识：从VBA启动PowerPoint	82

Section 10.5: Saving A Workbook Without Asking The User	43
<b>Chapter 11: Working with Excel Tables in VBA</b>	44
Section 11.1: Instantiating a ListObject	44
Section 11.2: Working with ListRows / ListColumns	44
Section 11.3: Converting an Excel Table to a normal range	44
<b>Chapter 12: Loop through all Sheets in Active Workbook</b>	45
Section 12.1: Retrieve all Worksheets Names in Active Workbook	45
Section 12.2: Loop Through all Sheets in all Files in a Folder	45
<b>Chapter 13: Use Worksheet object and not Sheet object</b>	47
Section 13.1: Print the name of the first object	47
<b>Chapter 14: Methods for Finding the Last Used Row or Column in a Worksheet</b>	48
Section 14.1: Find the Last Non-Empty Cell in a Column	48
Section 14.2: Find the Last Non-Empty Row in Worksheet	48
Section 14.3: Find the Last Non-Empty Column in Worksheet	49
Section 14.4: Find the Last Non-Empty Cell in a Row	50
Section 14.5: Get the row of the last cell in a range	50
Section 14.6: Find Last Row Using Named Range	50
Section 14.7: Last cell in Range.CurrentRegion	51
Section 14.8: Find the Last Non-Empty Cell in Worksheet - Performance (Array)	51
<b>Chapter 15: Creating a drop-down menu in the Active Worksheet with a Combo Box</b>	54
Section 15.1: Example 2: Options Not Included	54
Section 15.2: Jimi Hendrix Menu	55
<b>Chapter 16: File System Object</b>	57
Section 16.1: File, folder, drive exists	57
Section 16.2: Basic file operations	57
Section 16.3: Basic folder operations	58
Section 16.4: Other operations	58
<b>Chapter 17: Pivot Tables</b>	60
Section 17.1: Adding Fields to a Pivot Table	60
Section 17.2: Creating a Pivot Table	60
Section 17.3: Pivot Table Ranges	63
Section 17.4: Formatting the Pivot Table Data	63
<b>Chapter 18: Binding</b>	64
Section 18.1: Early Binding vs Late Binding	64
<b>Chapter 19: autofilter ; Uses and best practices</b>	66
Section 19.1: Smartfilter!	66
<b>Chapter 20: Application object</b>	70
Section 20.1: Simple Application Object example: Display Excel and VBE Version	70
Section 20.2: Simple Application Object example: Minimize the Excel window	70
<b>Chapter 21: Charts and Charting</b>	71
Section 21.1: Creating a Chart with Ranges and a Fixed Name	71
Section 21.2: Creating an empty Chart	72
Section 21.3: Create a Chart by Modifying the SERIES formula	73
Section 21.4: Arranging Charts into a Grid	75
<b>Chapter 22: CustomDocumentProperties in practice</b>	79
Section 22.1: Organizing new invoice numbers	79
<b>Chapter 23: PowerPoint Integration Through VBA</b>	82
Section 23.1: The Basics: Launching PowerPoint from VBA	82



<b>第24章：如何录制宏</b>	83
第24.1节：如何录制宏	83
<b>第25章：Excel VBA中的SQL——最佳实践</b>	85
第25.1节：如何在VBA中使用ADODB.Connection？	85
<b>第26章：Excel-VBA优化</b>	87
第26.1节：通过扩展调试优化错误查找	87
第26.2节：禁用工作表更新	88
第26.3节：行删除——性能	88
第26.4节：在执行大型宏之前禁用所有Excel功能	89
第26.5节：执行时间检查	90
第26.6节：使用With块	91
<b>第27章：VBA安全性</b>	93
第27.1节：为VBA设置密码保护	93
<b>第28章：调试与故障排除</b>	94
第28.1节：立即窗口	94
第28.2节：使用计时器查找性能瓶颈	95
第28.3节：调试器局部变量窗口	95
第28.4节：Debug.Print	96
第28.5节：停止	97
第28.6节：向代码中添加断点	97
<b>第29章：VBA最佳实践</b>	98
第29.1节：始终使用“Option Explicit”	98
第29.2节：使用数组而非范围	100
第29.3节：在宏执行期间切换属性	101
第29.4节：使用可用的VB常量	102
第29.5节：避免使用SELECT或ACTIVATE	103
第29.6节：始终定义并设置对所有工作簿和工作表的引用	105
第29.7节：使用描述性变量命名	105
第29.8节：记录你的工作	106
第29.9节：错误处理	107
第29.10节：永远不要假设工作表	109
第29.11节：避免将属性或方法的名称重新用作变量名	109
第29.12节：避免在Excel中使用ActiveCell或ActiveSheet	110
第29.13节：WorksheetFunction对象的执行速度快于等效的用户自定义函数（UDF）	111
<b>第30章：Excel VBA技巧与窍门</b>	113
第30.1节：使用xlVeryHidden工作表	113
第30.2节：使用带分隔符的字符串代替动态数组	114
第30.3节：工作表的.Name、.Index或.CodeName属性	114
第30.4节：Excel图形的双击事件	116
第30.5节：打开文件对话框 - 多文件	117
<b>第31章：常见错误</b>	118
第31.1节：限定引用	118
第31.2节：在循环中删除行或列	119
第31.3节：ActiveWorkbook 与 ThisWorkbook	119
第31.4节：单文档界面与多文档界面	120
<b>鸣谢</b>	122
<b>你可能也喜欢</b>	124

<b>Chapter 24: How to record a Macro</b>	83
Section 24.1: How to record a Macro	83
<b>Chapter 25: SQL in Excel VBA - Best Practices</b>	85
Section 25.1: How to use ADODB.Connection in VBA?	85
<b>Chapter 26: Excel-VBA Optimization</b>	87
Section 26.1: Optimizing Error Search by Extended Debugging	87
Section 26.2: Disabling Worksheet Updating	88
Section 26.3: Row Deletion - Performance	88
Section 26.4: Disabling All Excel Functionality Before executing large macros	89
Section 26.5: Checking time of execution	90
Section 26.6: Using With blocks	91
<b>Chapter 27: VBA Security</b>	93
Section 27.1: Password Protect your VBA	93
<b>Chapter 28: Debugging and Troubleshooting</b>	94
Section 28.1: Immediate Window	94
Section 28.2: Use Timer to Find Bottlenecks in Performance	95
Section 28.3: Debugger Locals Window	95
Section 28.4: Debug.Print	96
Section 28.5: Stop	97
Section 28.6: Adding a Breakpoint to your code	97
<b>Chapter 29: VBA Best Practices</b>	98
Section 29.1: ALWAYS Use "Option Explicit"	98
Section 29.2: Work with Arrays, Not With Ranges	100
Section 29.3: Switch off properties during macro execution	101
Section 29.4: Use VB constants when available	102
Section 29.5: Avoid using SELECT or ACTIVATE	103
Section 29.6: Always define and set references to all Workbooks and Sheets	105
Section 29.7: Use descriptive variable naming	105
Section 29.8: Document Your Work	106
Section 29.9: Error Handling	107
Section 29.10: Never Assume The Worksheet	109
Section 29.11: Avoid re-purposing the names of Properties or Methods as your variables	109
Section 29.12: Avoid using ActiveCell or ActiveSheet in Excel	110
Section 29.13: WorksheetFunction object executes faster than a UDF equivalent	111
<b>Chapter 30: Excel VBA Tips and Tricks</b>	113
Section 30.1: Using xlVeryHidden Sheets	113
Section 30.2: Using Strings with Delimiters in Place of Dynamic Arrays	114
Section 30.3: Worksheet .Name, .Index or .CodeName	114
Section 30.4: Double Click Event for Excel Shapes	116
Section 30.5: Open File Dialog - Multiple Files	117
<b>Chapter 31: Common Mistakes</b>	118
Section 31.1: Qualifying References	118
Section 31.2: Deleting rows or columns in a loop	119
Section 31.3: ActiveWorkbook vs. ThisWorkbook	119
Section 31.4: Single Document Interface Versus Multiple Document Interfaces	120
<b>Credits</b>	122
<b>You may also like</b>	124

请随意免费分享此PDF，  
本书最新版本可从以下网址下载：  
<https://goalkicker.com/ExcelVBABook>

这本Excel® VBA 专业笔记是从Stack Overflow Documentation汇编而成，内容由Stack Overflow的优秀人士撰写。文本内容以知识共享署名-相同方式共享许可发布，详见本书末尾对各章节贡献者的致谢。除非另有说明，图片版权归各自所有者所有。

这是一本非官方的免费书籍，旨在用于教育目的，与官方的 Excel® VBA 组织或公司以及 Stack Overflow 无关。所有商标和注册商标均为其各自公司所有者的财产。

本书中提供的信息不保证正确或准确，使用风险自负。

请将反馈和更正发送至[web@petercv.com](mailto:web@petercv.com)

Please feel free to share this PDF with anyone for free,  
latest version of this book can be downloaded from:  
<https://goalkicker.com/ExcelVBABook>

This Excel® VBA Notes for Professionals book is compiled from Stack Overflow Documentation, the content is written by the beautiful people at Stack Overflow. Text content is released under Creative Commons BY-SA, see credits at the end of this book whom contributed to the various chapters. Images may be copyright of their respective owners unless otherwise specified

This is an unofficial free book created for educational purposes and is not affiliated with official Excel® VBA group(s) or company(s) nor Stack Overflow. All trademarks and registered trademarks are the property of their respective company owners

The information presented in this book is not guaranteed to be correct nor accurate, use at your own risk

Please send feedback and corrections to [web@petercv.com](mailto:web@petercv.com)

# 第1章：Excel VBA入门

微软Excel包含一种全面的宏编程语言，称为VBA。该编程语言为您提供了至少三种额外的资源：

1. 使用宏通过代码自动驱动Excel。在大多数情况下，用户通过界面操作Excel所能做的任何事情，都可以通过编写Excel VBA代码来实现。
2. 创建新的自定义工作表函数。
3. 使Excel与其他应用程序交互，如Microsoft Word、PowerPoint、Internet Explorer、记事本等。

VBA代表“应用程序可视化基础”（Visual Basic for Applications）。它是经典Visual Basic编程语言的定制版本，自1990年代中期以来一直支持微软Excel的宏功能。

### 重要

请确保在excel-vba标签内创建的任何示例或主题都是具体的且与Microsoft Excel中VBA的使用相关的。任何提供的通用VBA语言主题或示例应被拒绝，以防止重复劳动。

- 相关主题示例：

- ✓ 创建和操作工作表对象
- ✓ WorksheetFunction 类及其相应方法
- ✓ 使用 xlDirection 枚举来导航区域

- 非相关主题示例：

- ✗ 如何创建“for each”循环
- ✗ MsgBox 类及如何显示消息
- ✗ 在VBA中使用WinAPI

### VB

#### 版本 发布日期

VB6	1998-10-01
VB7	2001-06-06
WIN32	1998-10-01
WIN64	2001-06-06
MAC	1998-10-01

### Excel

#### 版本 发布日期

16	<a href="#">2016-01-01</a>
15	2013-01-01
14	2010-01-01
12	2007-01-01
11	2003-01-01
10	2001-01-01
9	1999-01-01

# Chapter 1: Getting started with Excel VBA

Microsoft Excel includes a comprehensive macro programming language called VBA. This programming language provides you with at least three additional resources:

1. Automatically drive Excel from code using Macros. For the most part, anything that the user can do by manipulating Excel from the user interface can be done by writing code in Excel VBA.
2. Create new, custom worksheet functions.
3. Interact Excel with other applications such as Microsoft Word, PowerPoint, Internet Explorer, Notepad, etc.

VBA stands for Visual Basic for Applications. It is a custom version of the venerable Visual Basic programming language that has powered Microsoft Excel's macros since the mid-1990s.

### IMPORTANT

Please ensure any examples or topics created within the excel-vba tag are **specific** and **relevant** to the use of VBA with Microsoft Excel. Any suggested topics or examples provided that are generic to the VBA language should be declined in order to prevent duplication of efforts.

- on-topic examples:

- ✓ *Creating and interacting with worksheet objects*
- ✓ *The WorksheetFunction class and respective methods*
- ✓ *Using the xlDirection enumeration to navigate a range*

- off-topic examples:

- ✗ *How to create a 'for each' loop*
- ✗ *MsgBox class and how to display a message*
- ✗ *Using WinAPI in VBA*

### VB

#### Version Release Date

VB6	1998-10-01
VB7	2001-06-06
WIN32	1998-10-01
WIN64	2001-06-06
MAC	1998-10-01

### Excel

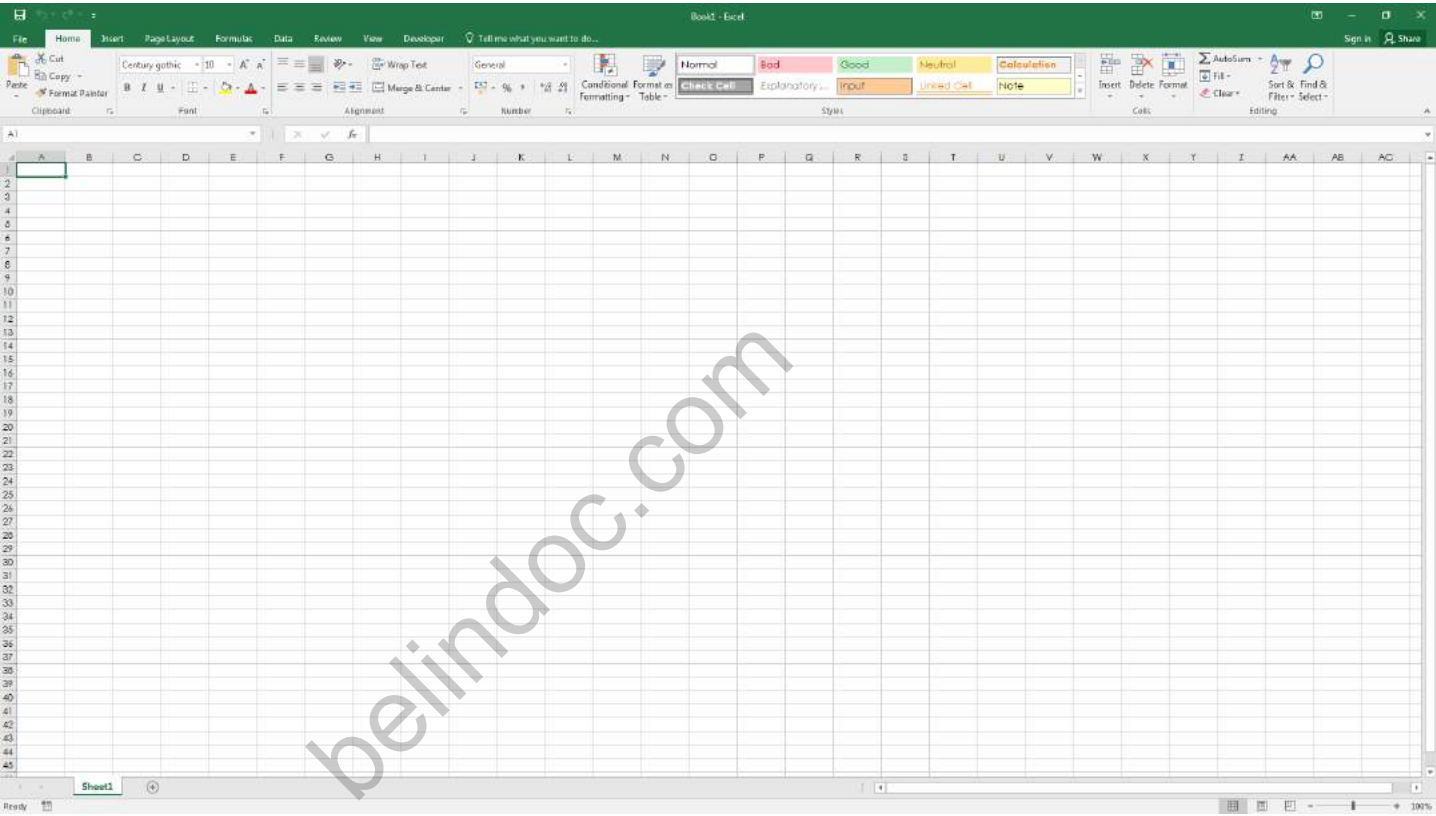
#### Version Release Date

16	<a href="#">2016-01-01</a>
15	2013-01-01
14	2010-01-01
12	2007-01-01
11	2003-01-01
10	2001-01-01
9	1999-01-01

- 8 1997-01-01
- 7 1995-01-01
- 5 1993-01-01
- 2 1987-01-01

第1.1节：打开Visual Basic编辑器（VBE）

步骤1：打开工作簿



步骤2 选项A：按下 **Alt** + **F11**

这是打开VBE的标准快捷键。

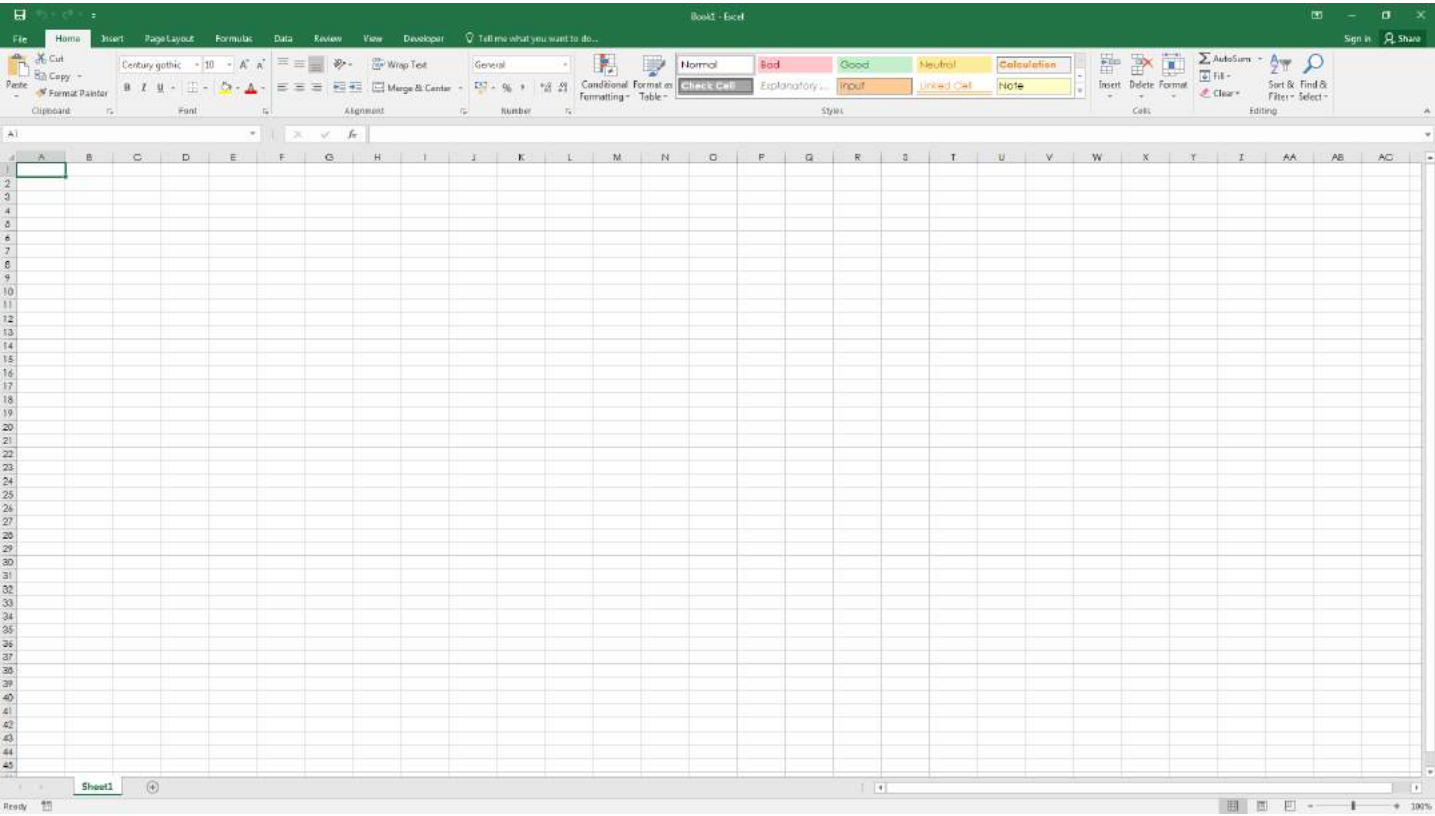
步骤2 选项B：开发工具选项卡 --> 查看代码

首先，必须将开发工具选项卡添加到功能区。进入文件 -> 选项 -> 自定义功能区，然后勾选开发工具的复选框。

- 8 1997-01-01
- 7 1995-01-01
- 5 1993-01-01
- 2 1987-01-01

Section 1.1: Opening the Visual Basic Editor (VBE)

Step 1: Open a Workbook



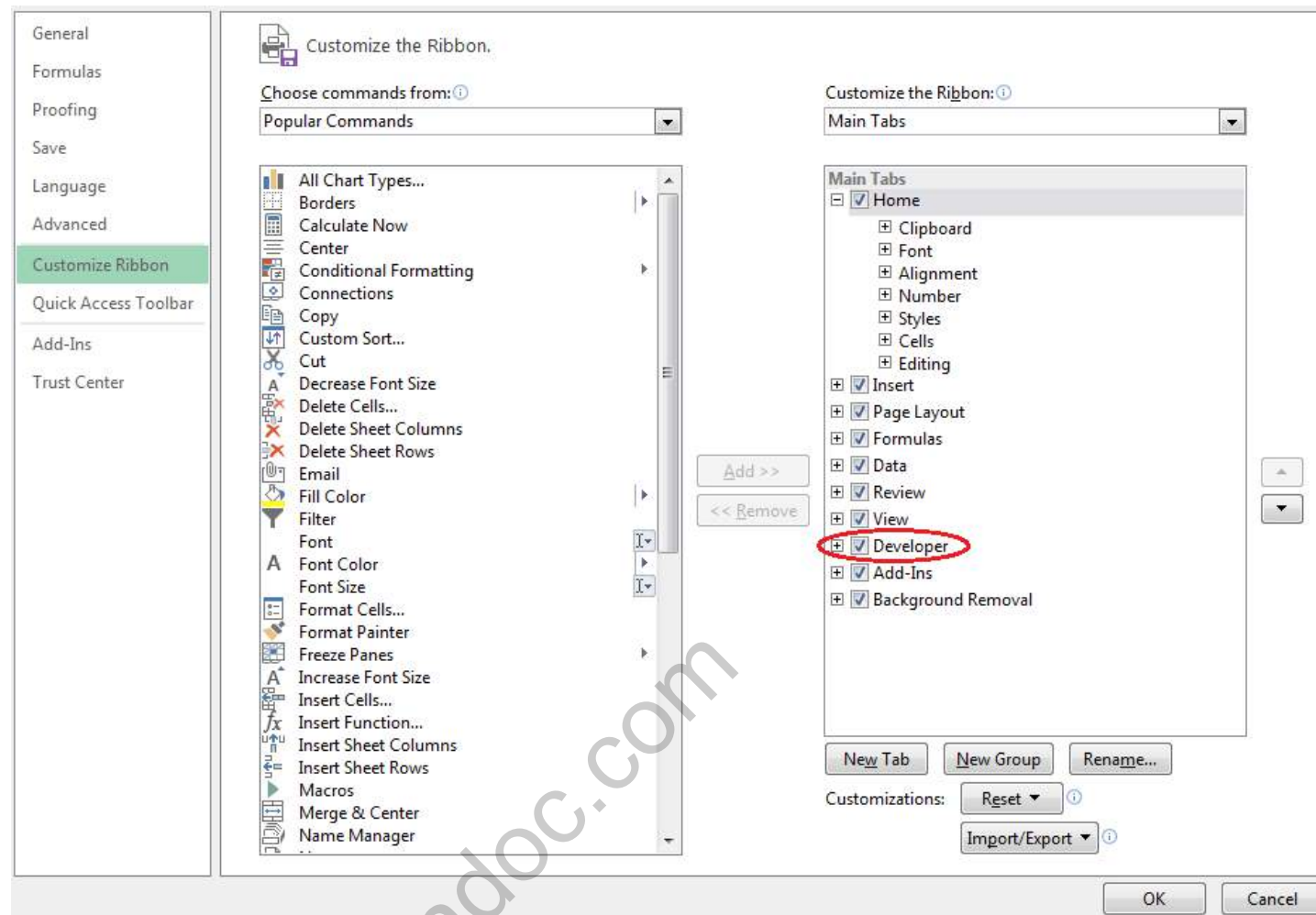
Step 2 Option A: Press **Alt** + **F11**

This is the standard shortcut to open the VBE.

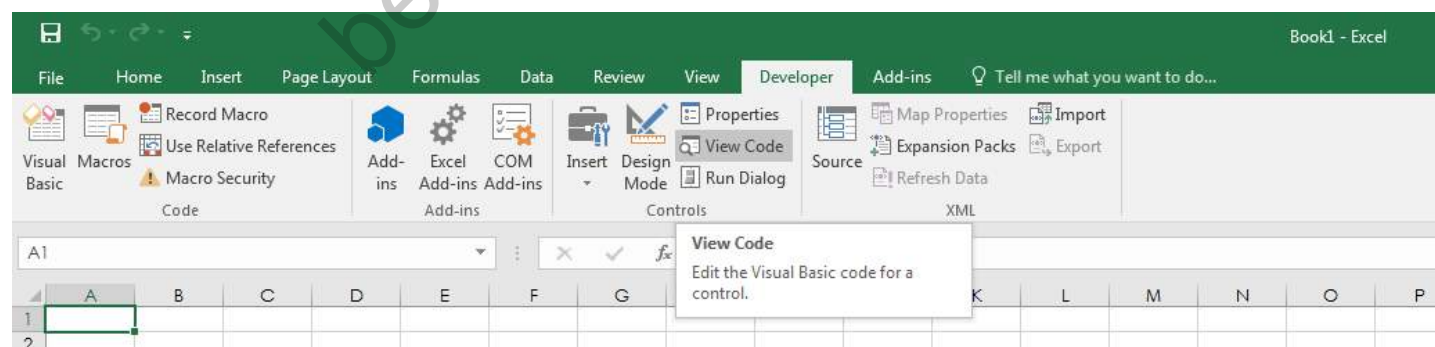
Step 2 Option B: Developer Tab --> View Code

First, the Developer Tab must be added to the ribbon. Go to File -> Options -> Customize Ribbon, then check the box for developer.



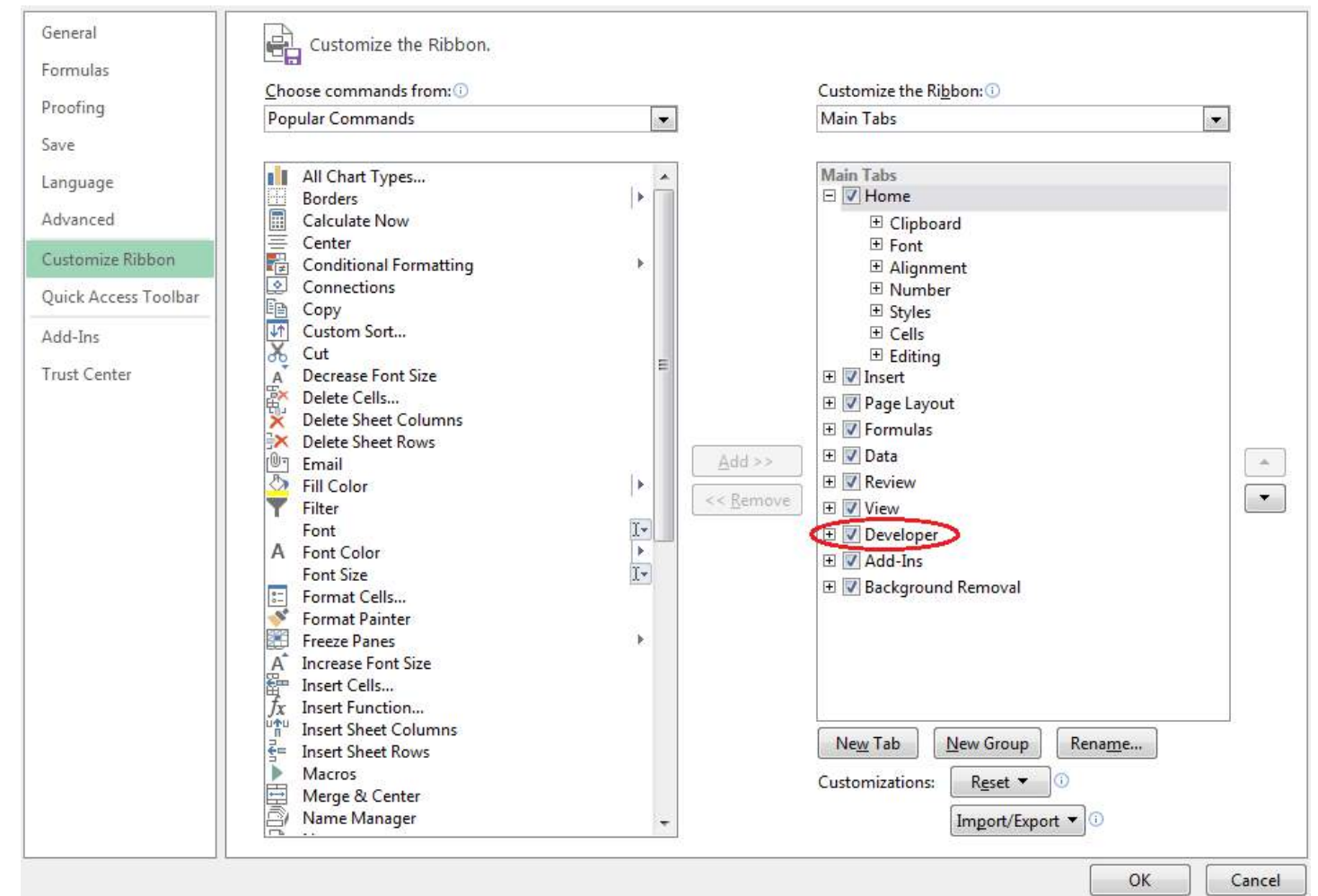


然后，进入开发工具选项卡，点击“查看代码”或“Visual Basic”

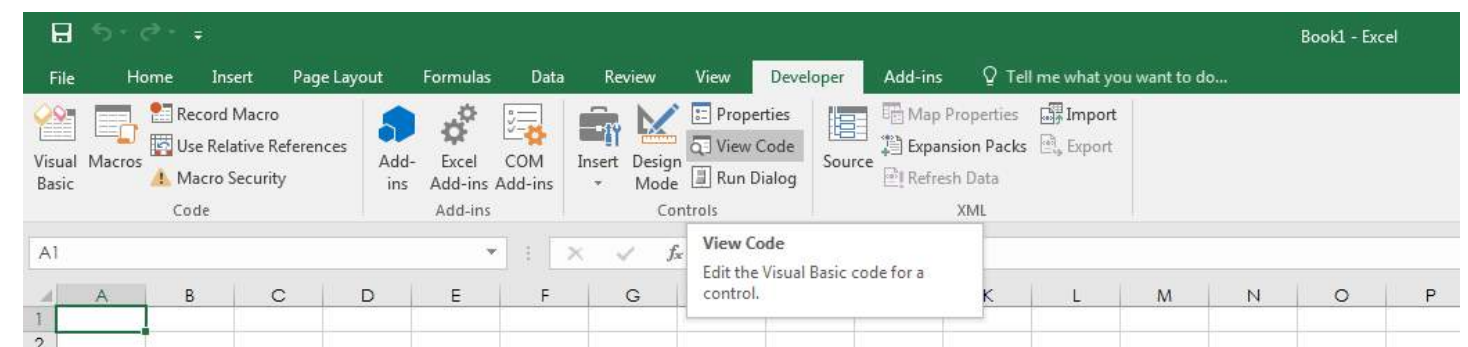


步骤2 选项C：视图选项卡 > 宏 > 点击编辑按钮以打开现有宏

这三种选项都会打开Visual Basic编辑器（VBE）：



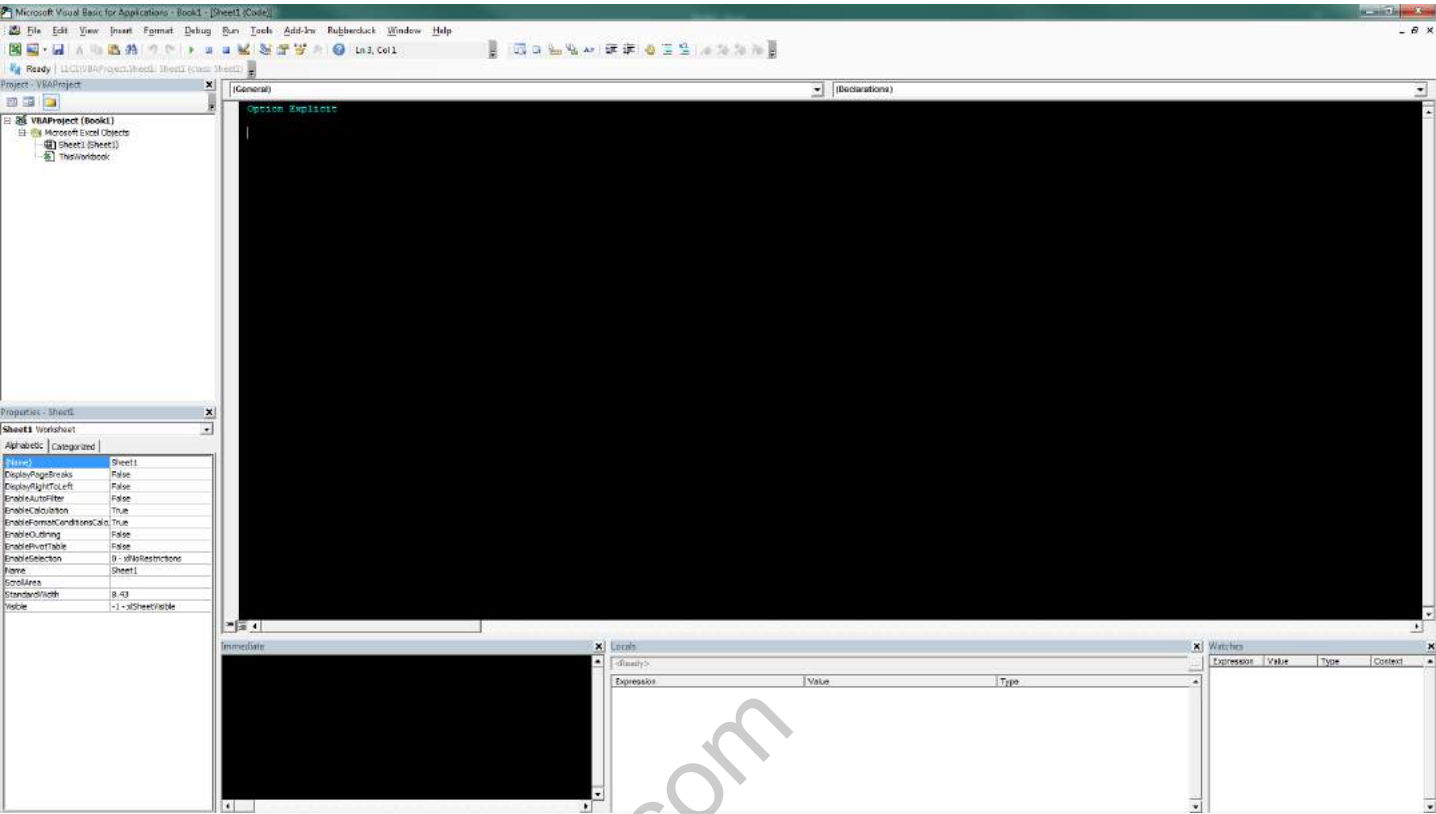
Then, go to the developer tab and click "View Code" or "Visual Basic"



Step 2 Option C: View tab > Macros > Click Edit button to open an Existing Macro

All three of these options will open the Visual Basic Editor (VBE):





## 第1.2节：声明变量

在VBA中显式声明变量，使用Dim语句，后跟变量名和类型。如果变量未声明使用，或未指定类型，则会被赋予类型Variant。

在模块的第一行使用Option Explicit语句，强制所有变量在使用前必须声明（参见始终使用“Option Explicit”）。

强烈建议始终使用Option Explicit，因为它有助于防止拼写错误，并确保变量/对象保持其预期类型。

```
Option Explicit

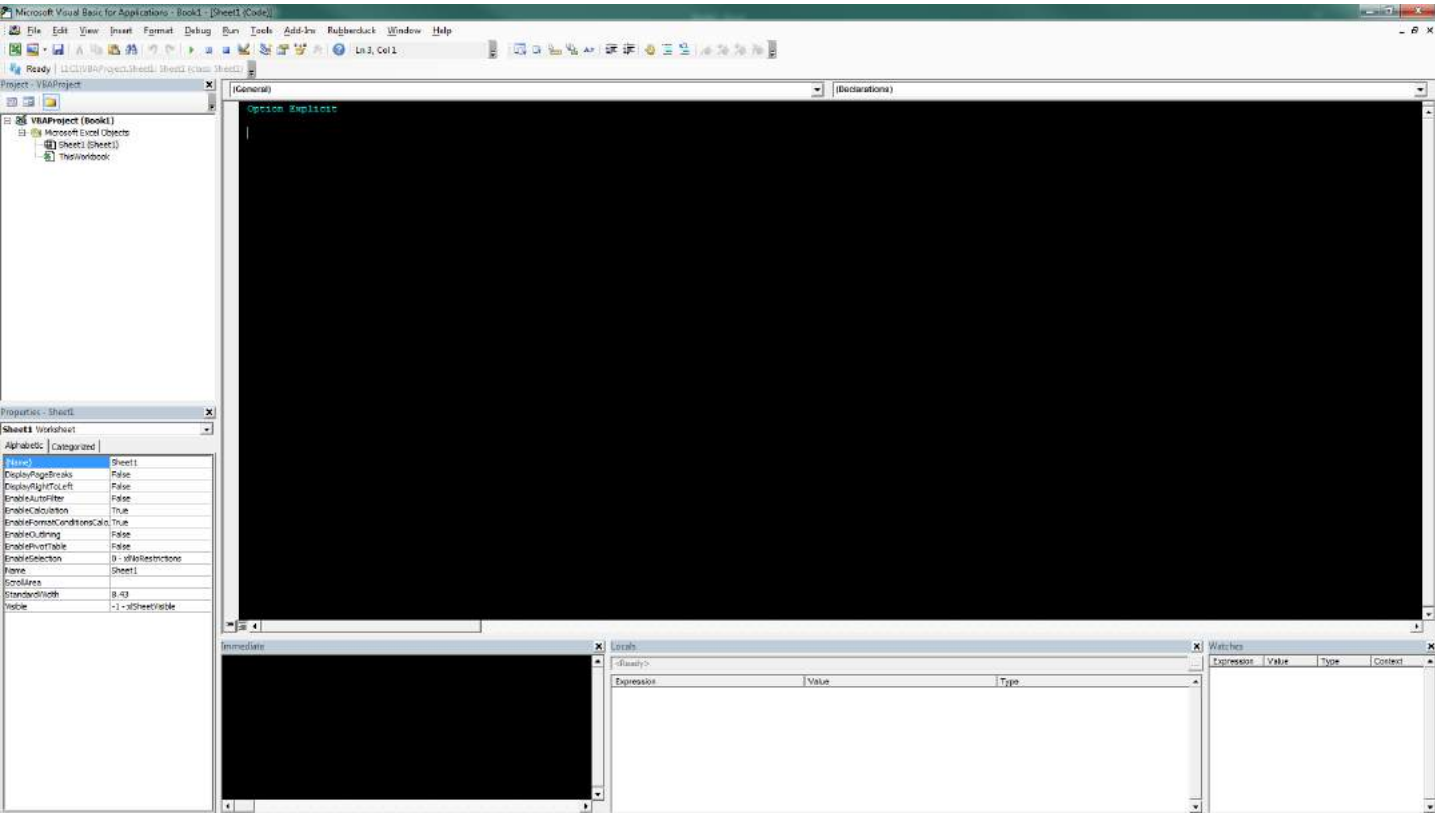
Sub 示例()
    Dim a As Integer
    a = 2
    Debug.Print a
    '输出：2

    Dim b As Long
    b = a + 2
    Debug.Print b
    '输出：4

    Dim c As String
    c = "Hello, world!"
    Debug.Print c
    '输出：你好，世界！
End Sub
```

多个变量可以在一行中使用逗号作为分隔符声明，但每种类型必须单独声明，否则它们将默认为 Variant 类型。

```
Dim Str As String, IntOne, IntTwo As Integer, Lng As Long
```



## Section 1.2: Declaring Variables

To explicitly declare variables in VBA, use the **Dim** statement, followed by the variable name and type. If a variable is used without being declared, or if no type is specified, it will be assigned the type Variant.

Use the **Option Explicit** statement on first line of a module to force all variables to be declared before usage (see ALWAYS Use "Option Explicit" ).

Always using **Option Explicit** is highly recommended because it helps prevent typo/spelling errors and ensures variables/objects will stay their intended type.

```
Option Explicit

Sub Example()
    Dim a As Integer
    a = 2
    Debug.Print a
    'Outputs: 2

    Dim b As Long
    b = a + 2
    Debug.Print b
    'Outputs: 4

    Dim c As String
    c = "Hello, world!"
    Debug.Print c
    'Outputs: Hello, world!
End Sub
```

Multiple variables can be declared on a single line using commas as delimiters, but **each type must be declared individually**, or they will default to the Variant type.

```
Dim Str As String, IntOne, IntTwo As Integer, Lng As Long
```

```
Debug.Print TypeName(Str)      '输出: String
Debug.Print TypeName(IntOne)   '输出: Variant <--- !!!
Debug.Print TypeName(IntTwo)   '输出: Integer
Debug.Print TypeName(Lng)      '输出: Long
```

变量也可以使用数据类型字符后缀 (\$ % & ! # @) 声明，但这些用法越来越不被推荐。

```
Dim this$ '字符串
Dim this% '整数
Dim this& '长整数
Dim this! '单精度浮点数
Dim this# '双精度浮点数
Dim this@ '货币类型
```

声明变量的其他方式有：

- Static 例如: Static CounterVariable as Integer

当你使用 Static 语句代替 Dim 语句时，声明的变量将在多次调用之间保留其值。

- Public 如：Public CounterVariable 作为 Integer

公共变量可以在项目中的任何过程使用。如果公共变量在标准模块或类模块中声明，它也可以在引用声明该公共变量的项目中的任何项目中使用。

- Private 如：Private CounterVariable 作为 Integer

私有变量只能被同一模块中的过程使用。

来源及更多信息：

[MSDN-声明变量](#)

[类型字符 \(Visual Basic\)](#)

### 第1.3节：添加新的对象库引用

本过程描述如何添加对象库引用，以及之后如何声明与新库类对象相关的新变量。

下面的示例展示了如何将PowerPoint库添加到现有的VB项目中。如图所示，目前PowerPoint对象库不可用。

```
Debug.Print TypeName(Str)      'Output: String
Debug.Print TypeName(IntOne)   'Output: Variant <--- !!!
Debug.Print TypeName(IntTwo)   'Output: Integer
Debug.Print TypeName(Lng)      'Output: Long
```

Variables can also be declared using Data Type Character suffixes (\$ % & ! # @), however using these are increasingly discouraged.

```
Dim this$ 'String
Dim this% 'Integer
Dim this& 'Long
Dim this! 'Single
Dim this# 'Double
Dim this@ 'Currency
```

Other ways of declaring variables are:

- Static like: Static CounterVariable as Integer

When you use the Static statement instead of a Dim statement, the declared variable will retain its value between calls.

- Public like: Public CounterVariable as Integer

Public variables can be used in any procedures in the project. If a public variable is declared in a standard module or a class module, it can also be used in any projects that reference the project where the public variable is declared.

- Private like: Private CounterVariable as Integer

Private variables can be used only by procedures in the same module.

Source and more info:

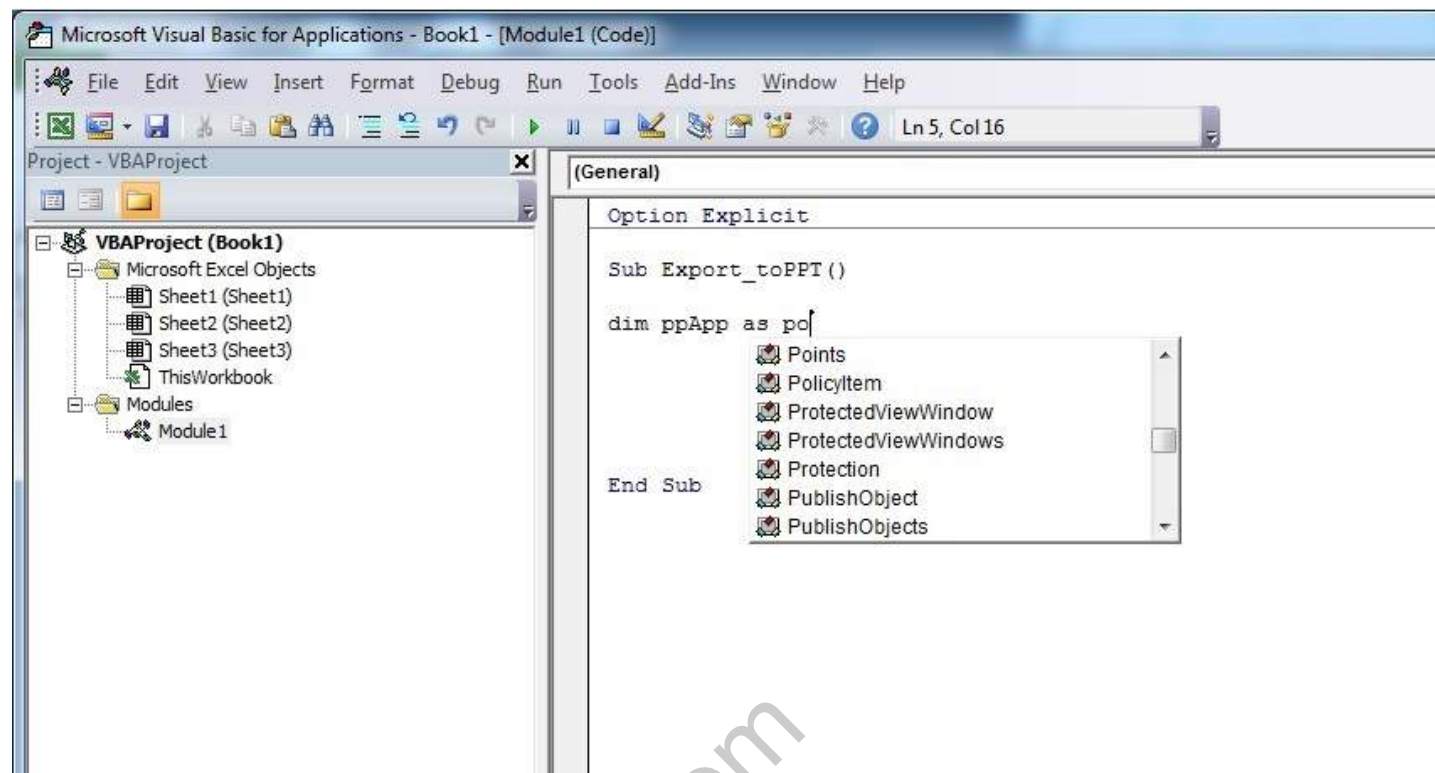
[MSDN-Declaring Variables](#)

[Type Characters \(Visual Basic\)](#)

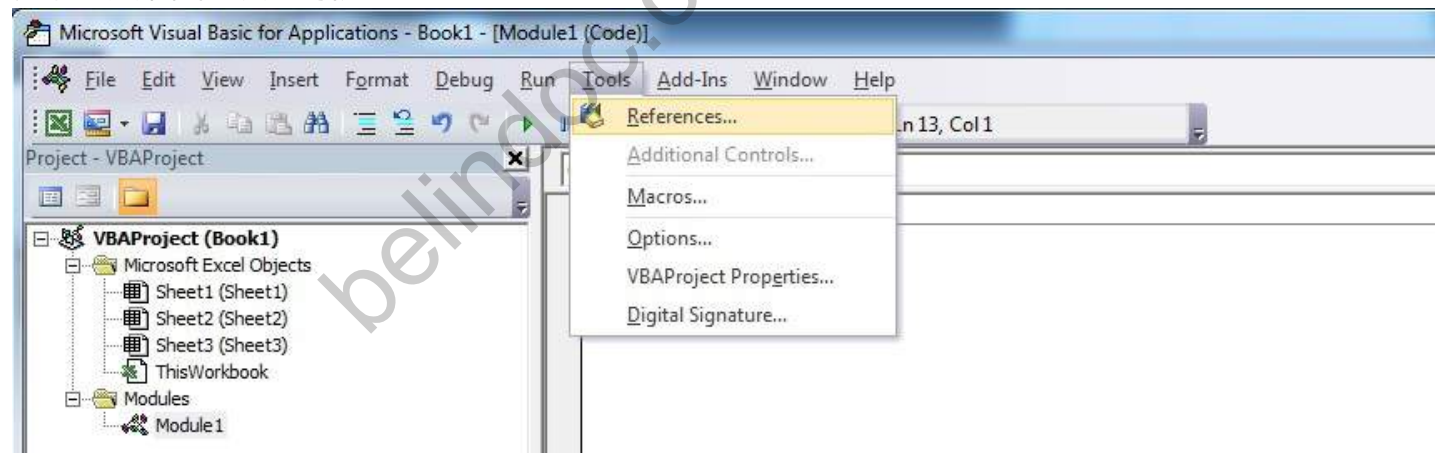
### Section 1.3: Adding a new Object Library Reference

The procedure describes how to add an Object library reference, and afterwards how to declare new variables with reference to the new library class objects.

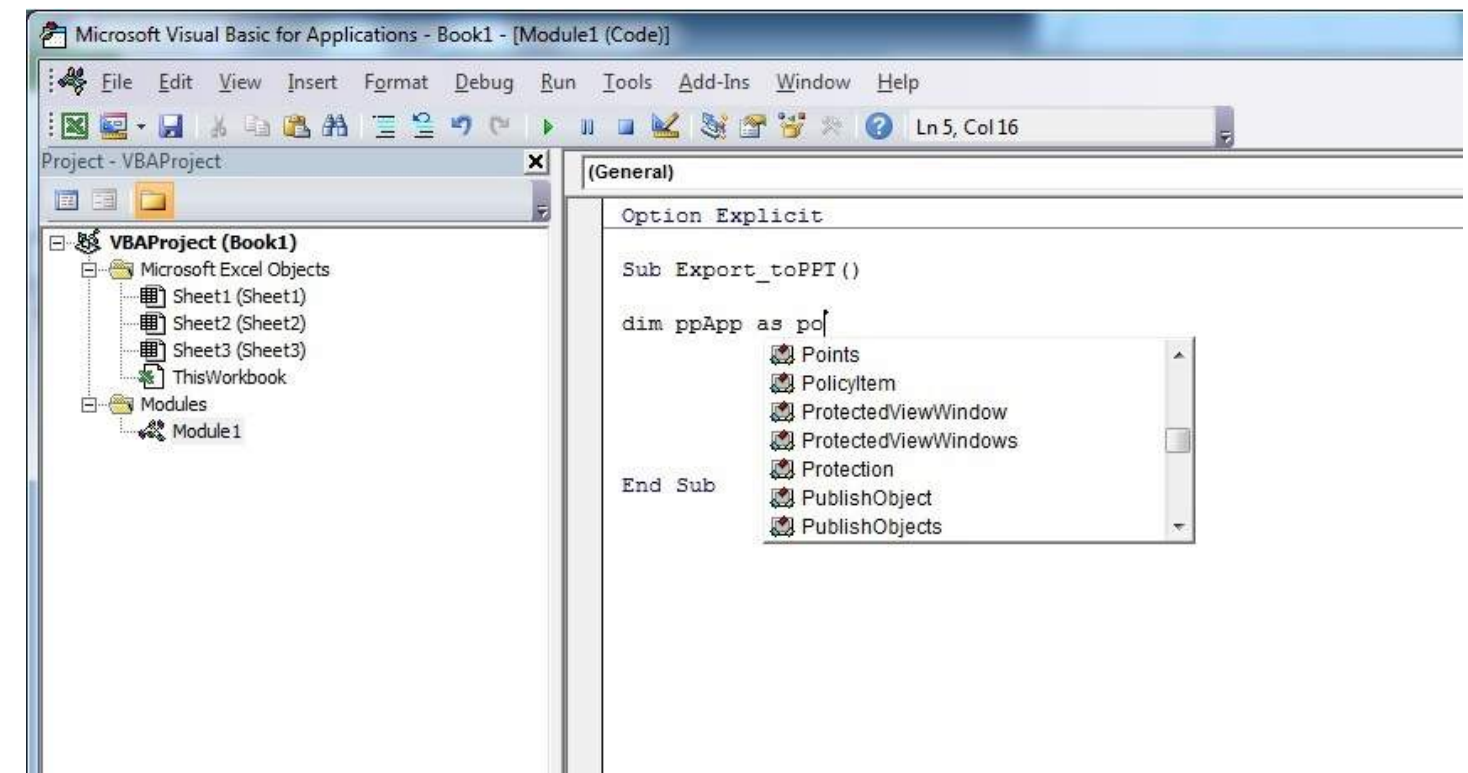
The example below shows how to add the *PowerPoint* library to the existing VB Project. As can be seen, currently the PowerPoint Object library is not available.



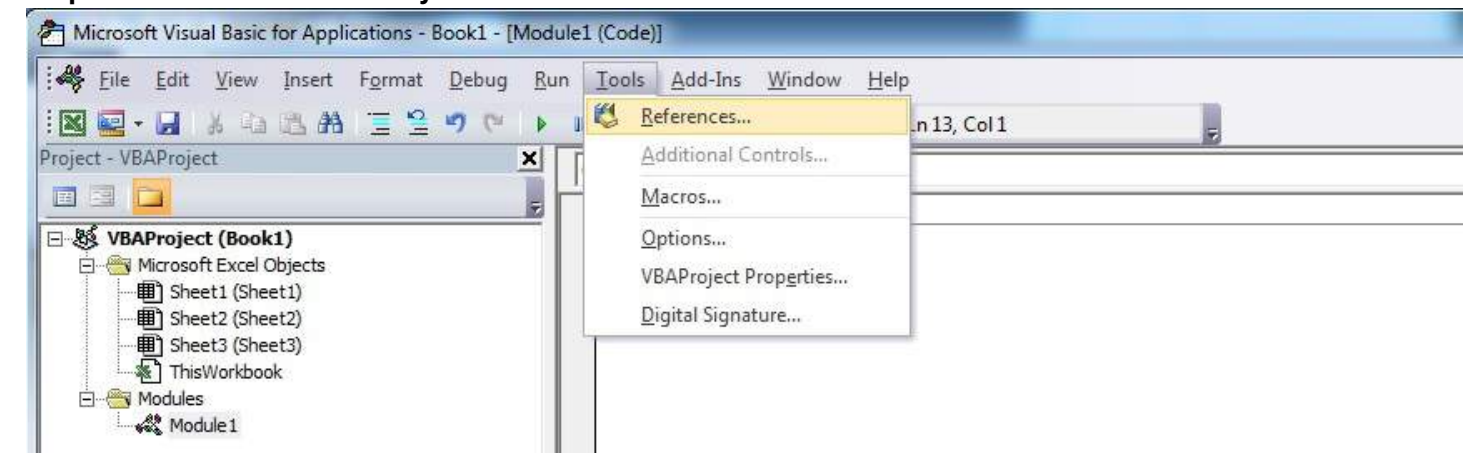
步骤 1：选择菜单工具--> 引用...



步骤 2：选择您想要添加的引用。在此示例中，我们向下滚动以找到“**Microsoft PowerPoint 14.0 对象库**”，然后按“确定”。

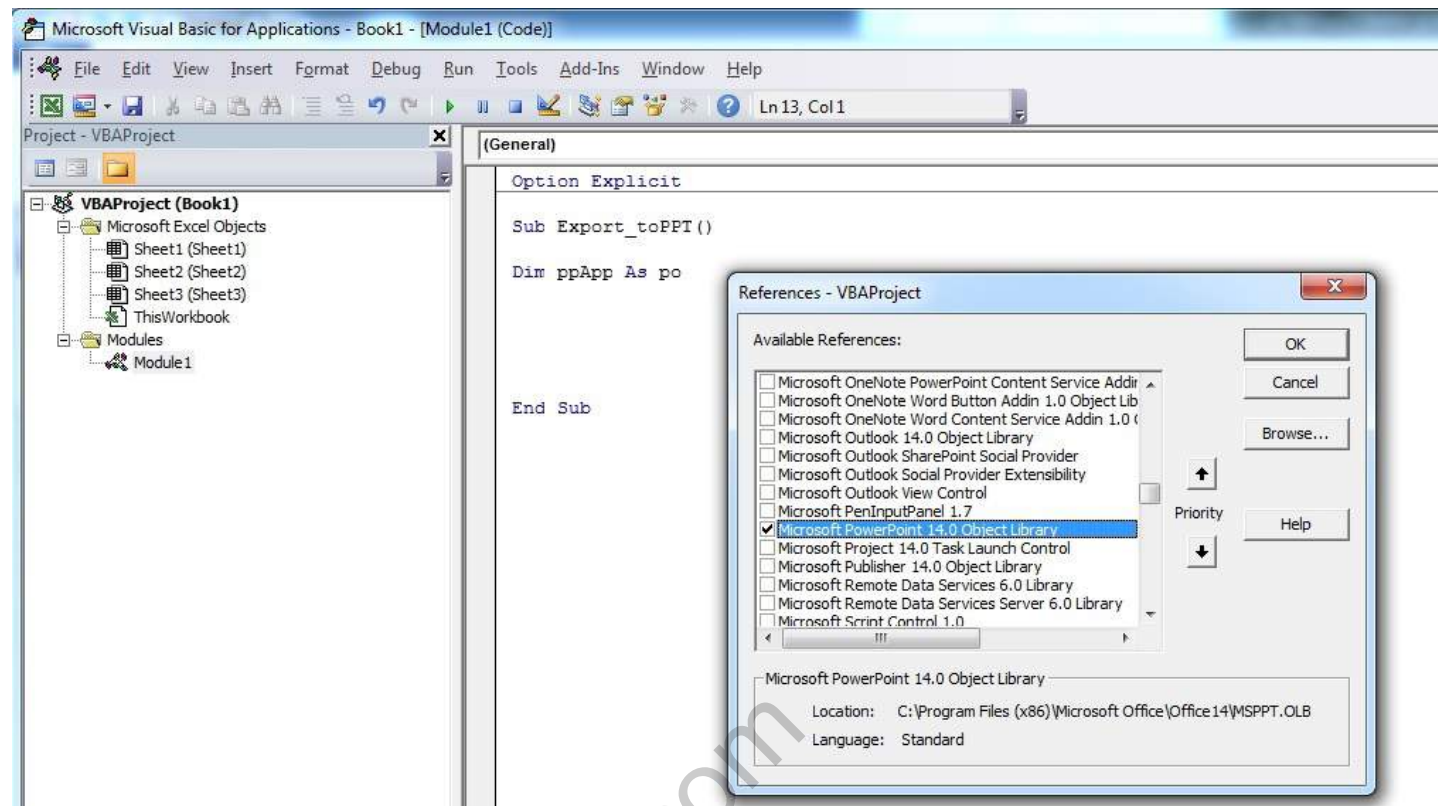


Step 1: Select Menu **Tools --> References...**



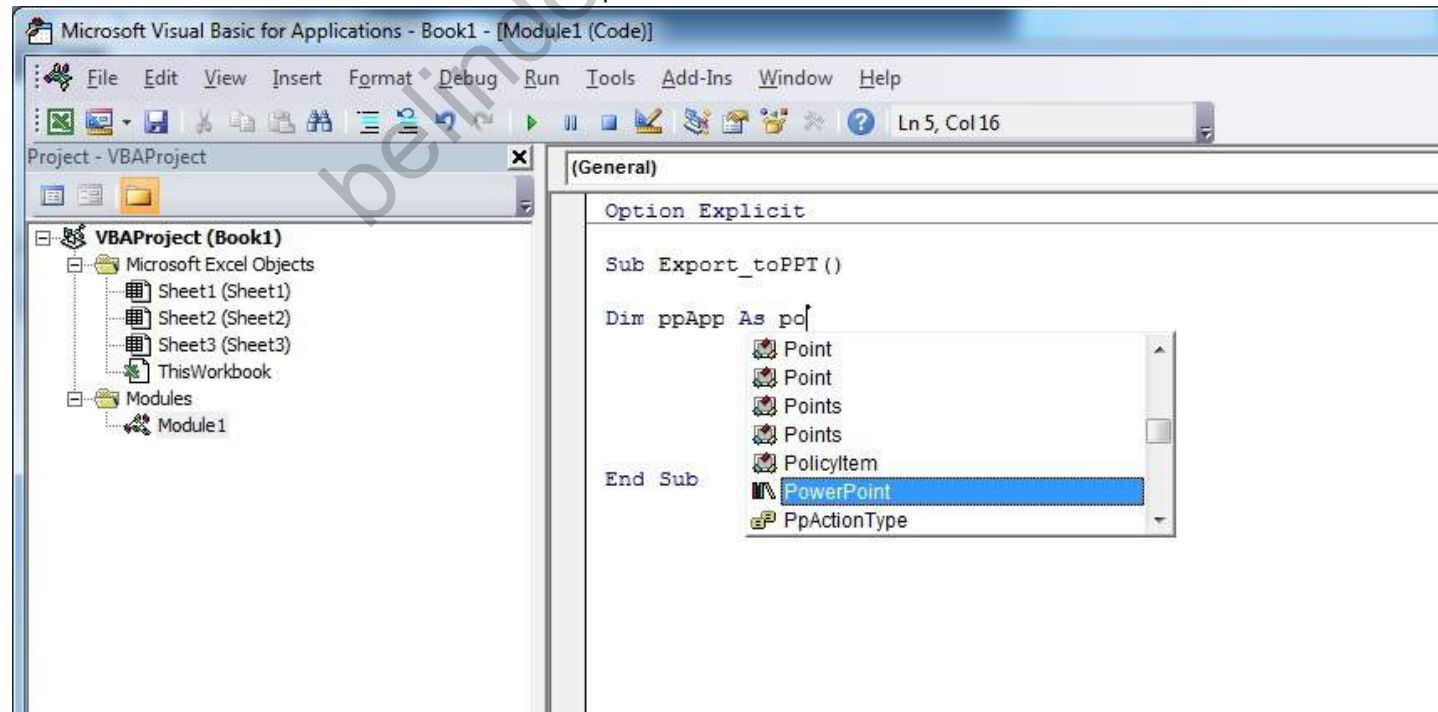
Step 2: Select the Reference you want to add. This example we scroll down to find “**Microsoft PowerPoint 14.0 Object Library**”, and then press “OK”.



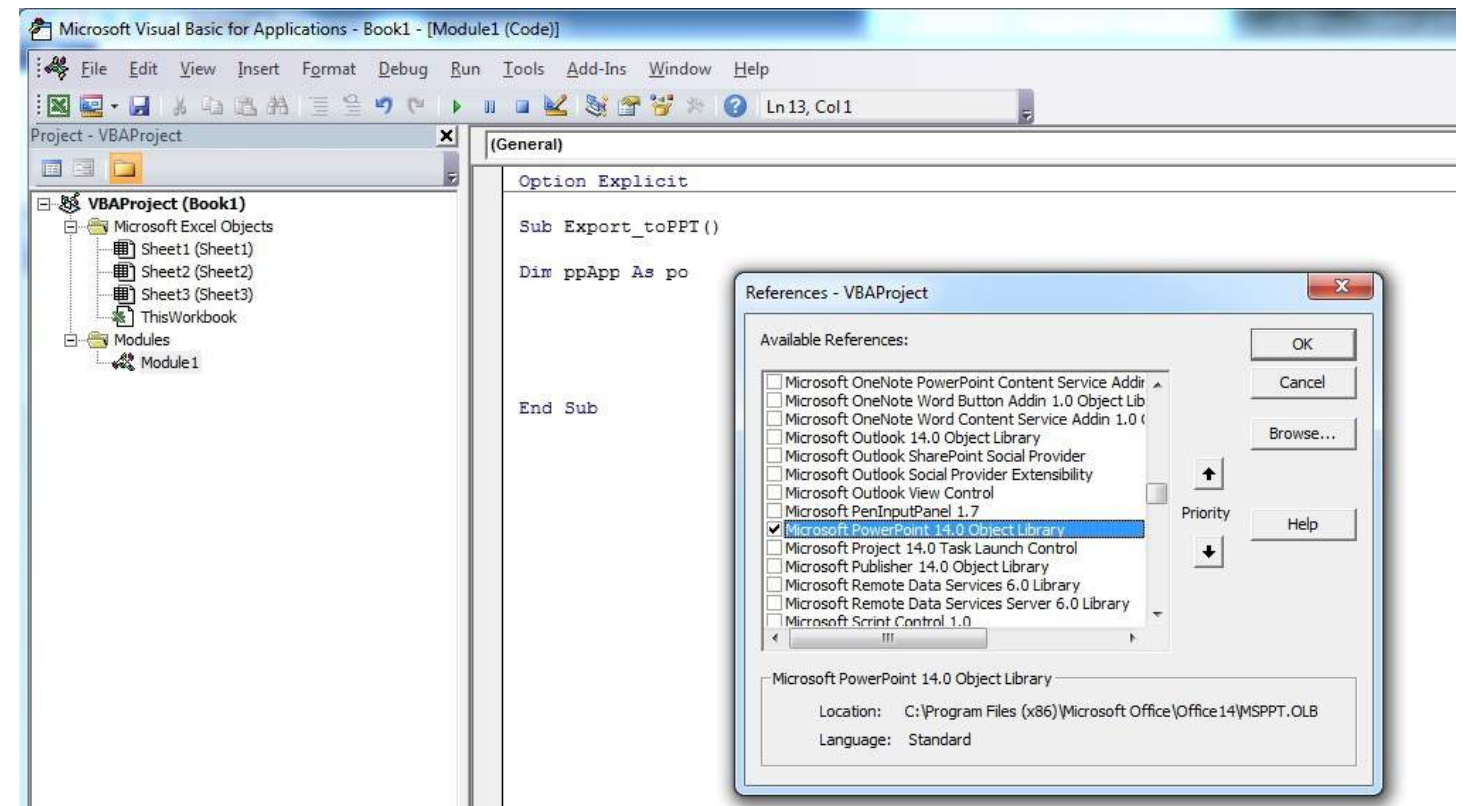


注意：PowerPoint 14.0 表示电脑上安装的是 Office 2010 版本。

步骤 3：在 VB 编辑器中，一旦你同时按下Ctrl+Space，就会出现 PowerPoint 的自动完成选项。

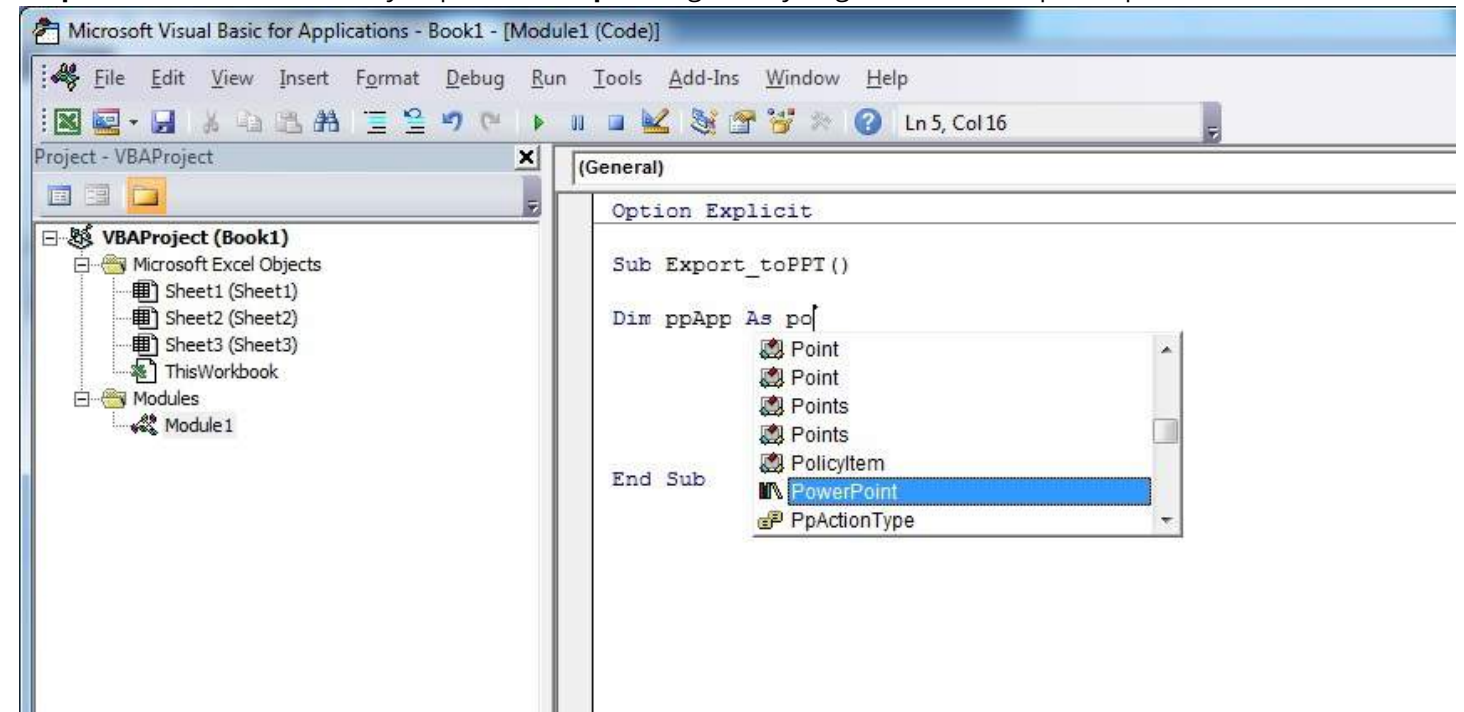


选择PowerPoint并按下.后，会出现另一个菜单，显示与 PowerPoint 对象库相关的所有对象选项。此示例展示了如何选择 PowerPoint 的对象Application。

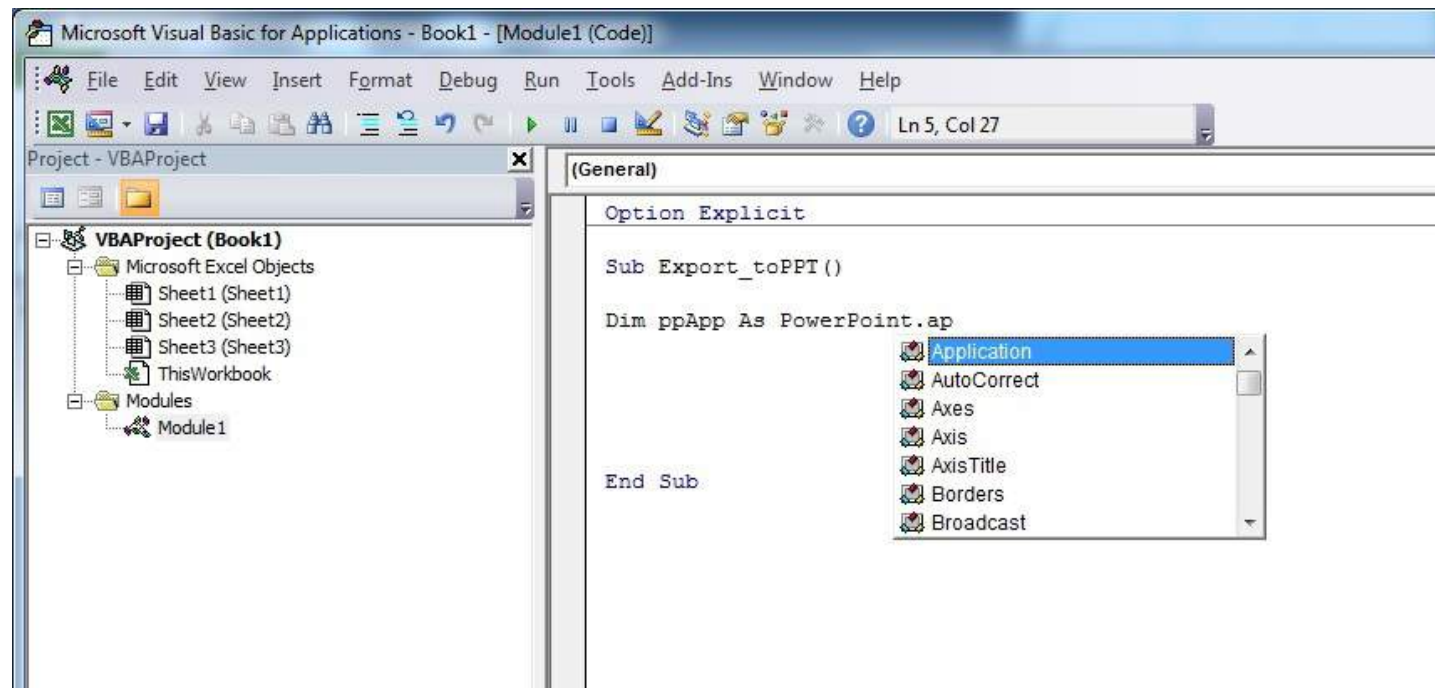


Note: PowerPoint 14.0 means that Office 2010 version is installed on the PC.

**Step 3:** in the VB Editor, once you press **Ctrl+Space** together, you get the autocomplete option of PowerPoint.

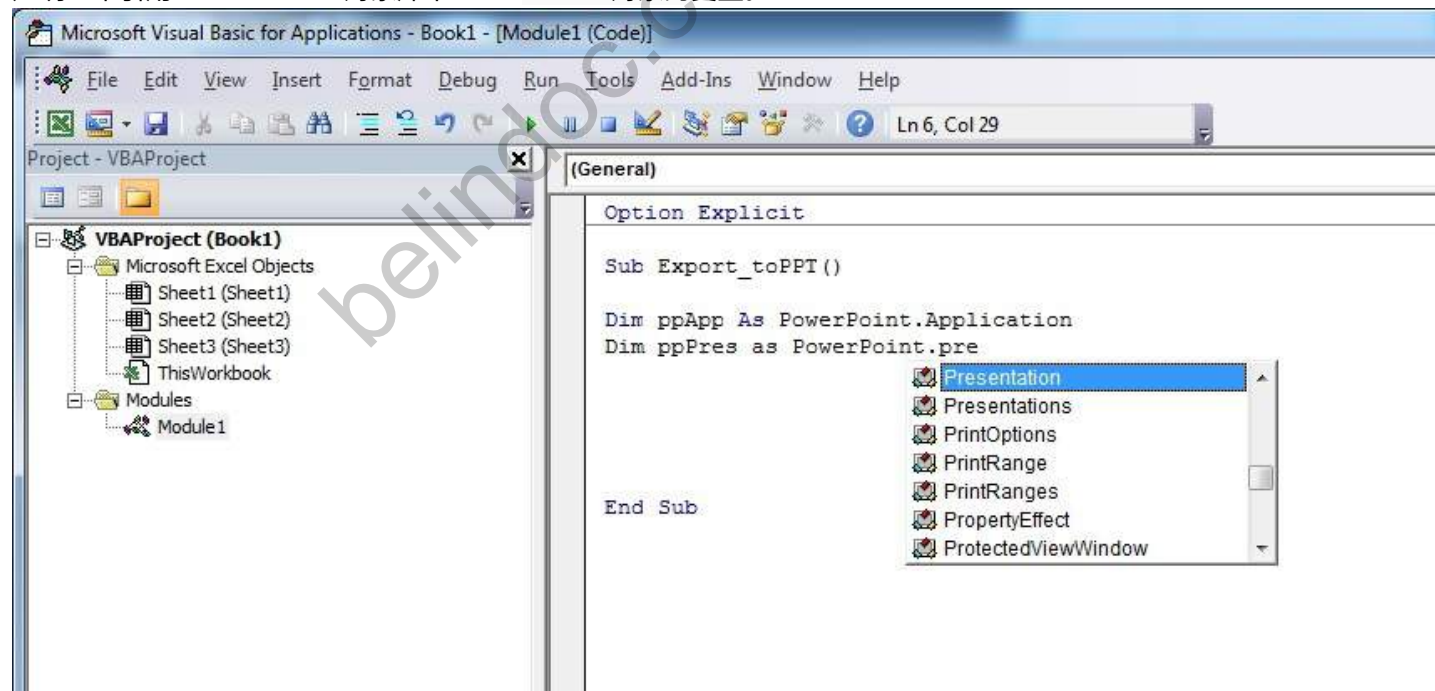


After selecting PowerPoint and pressing ., another menu appears with all objects options related to the PowerPoint Object Library. This example shows how to select the PowerPoint's object Application.

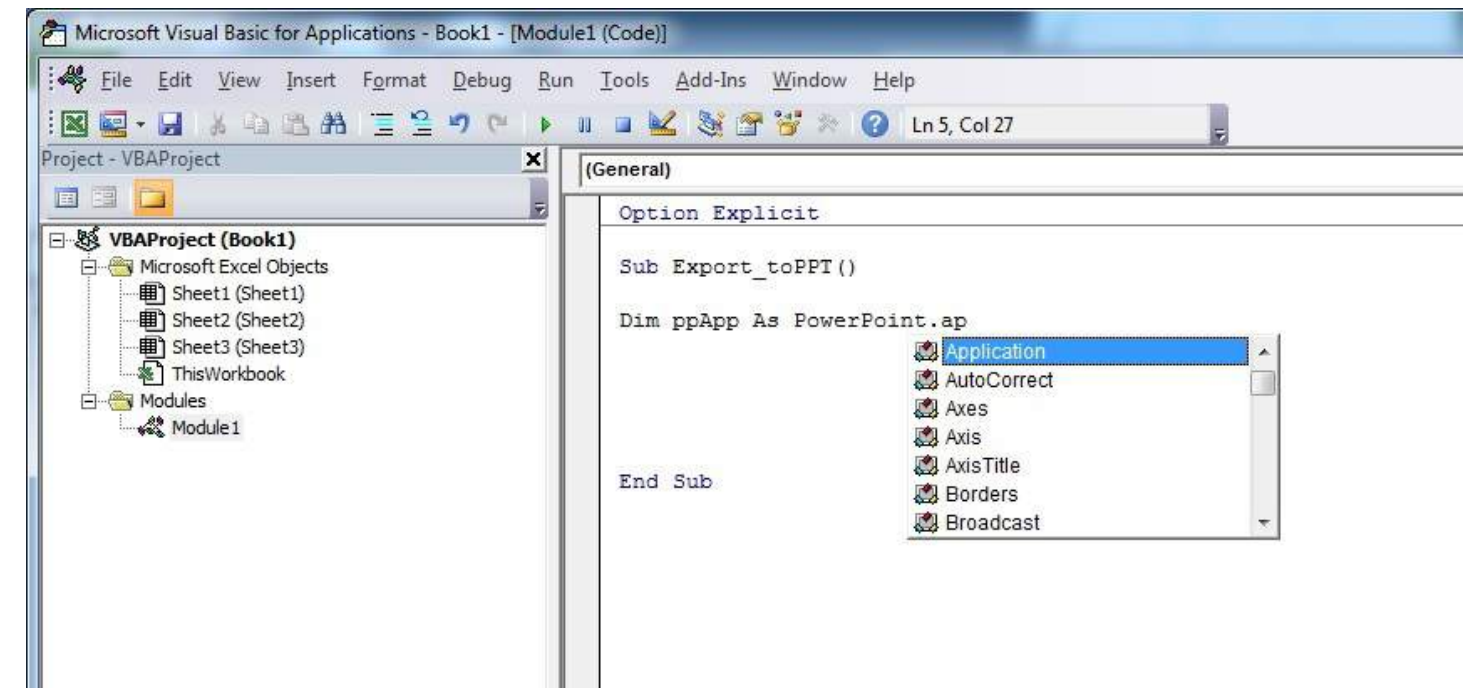


步骤 4：现在用户可以使用 PowerPoint 对象库声明更多变量。

声明一个引用 PowerPoint 对象库中Presentation对象的变量。

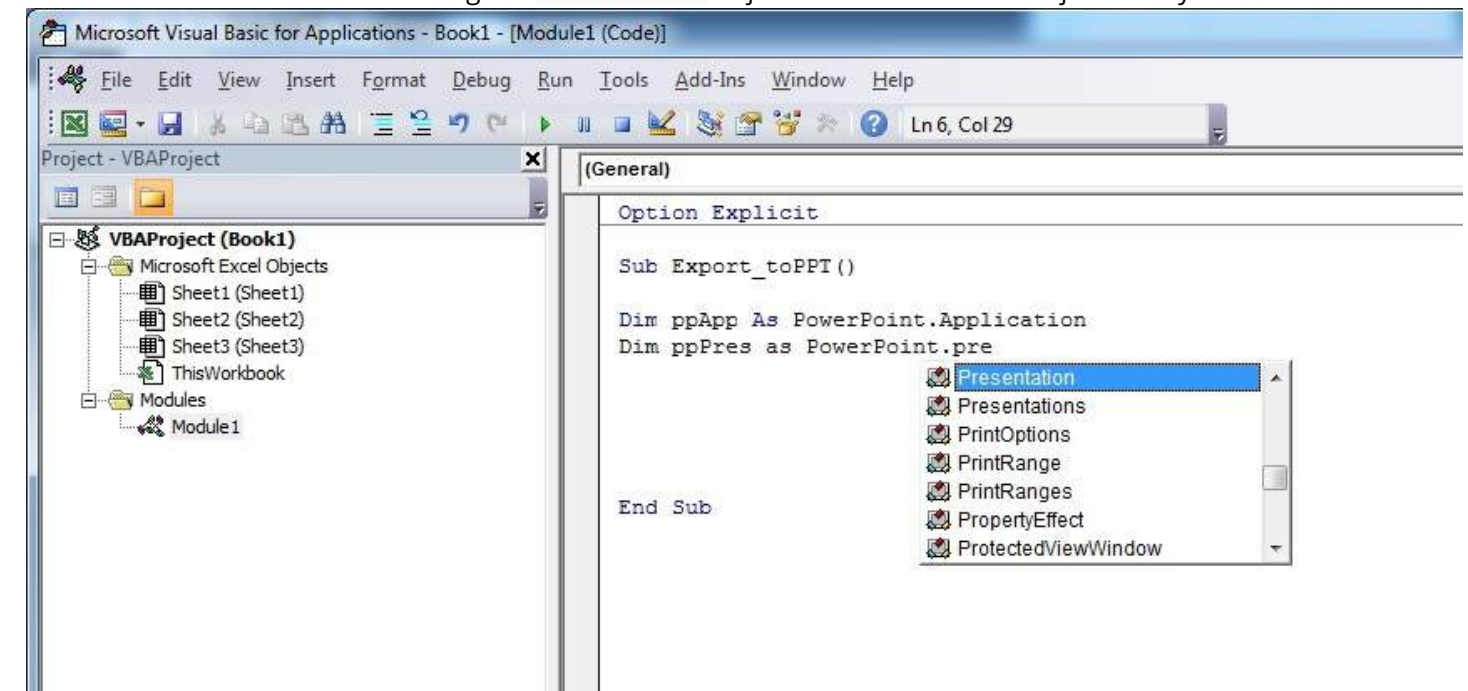


声明另一个引用 PowerPoint 对象库中Slide对象的变量。

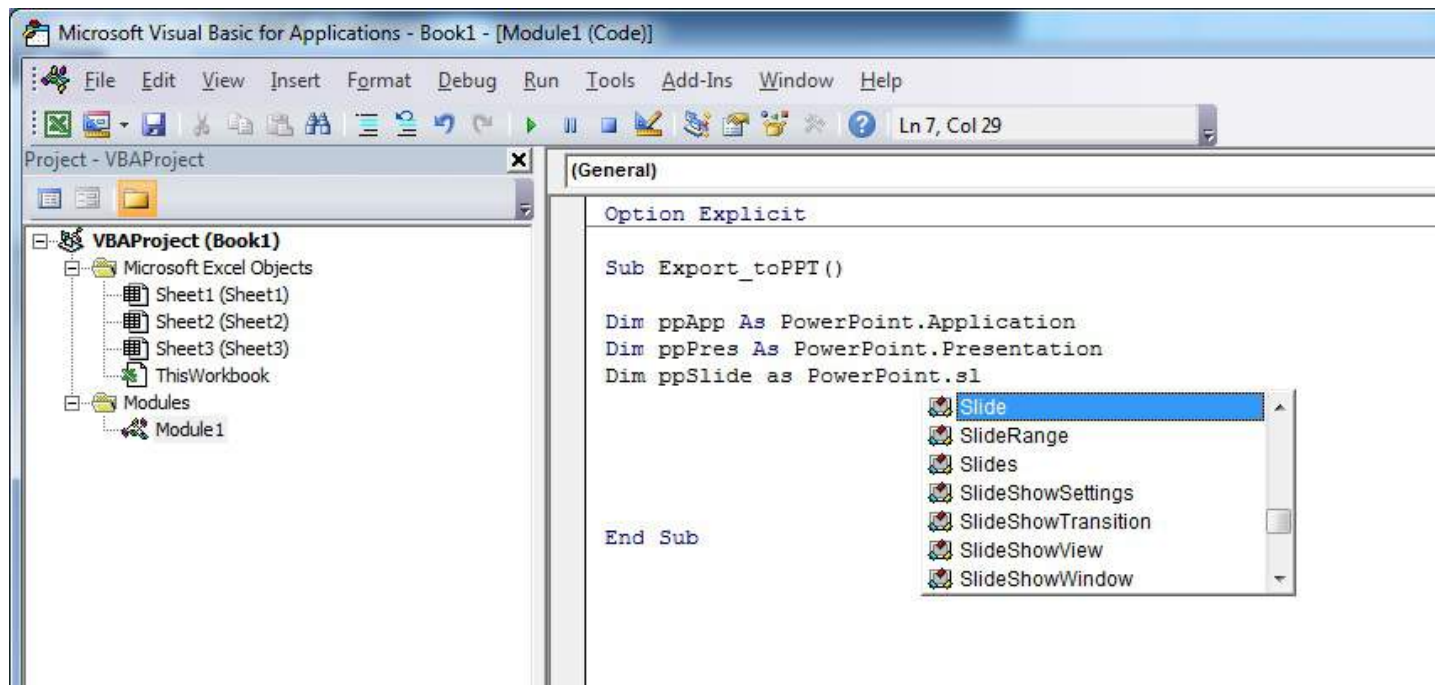


Step 4: Now the user can declare more variables using the PowerPoint object library.

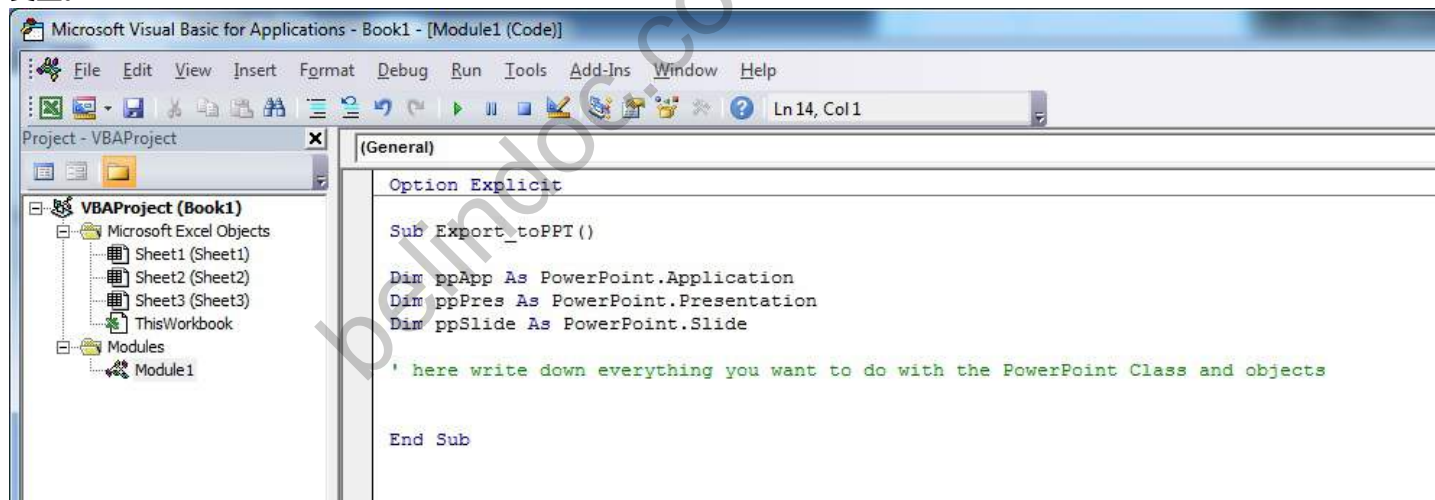
Declare a variable that is referencing the Presentation object of the PowerPoint object library.



Declare another variable that is referencing the Slide object of the PowerPoint object library.



现在变量声明部分如下面的截图所示，用户可以开始在代码中使用这些变量。



本教程的代码版本：

```
Option Explicit

Sub Export_toPPT()

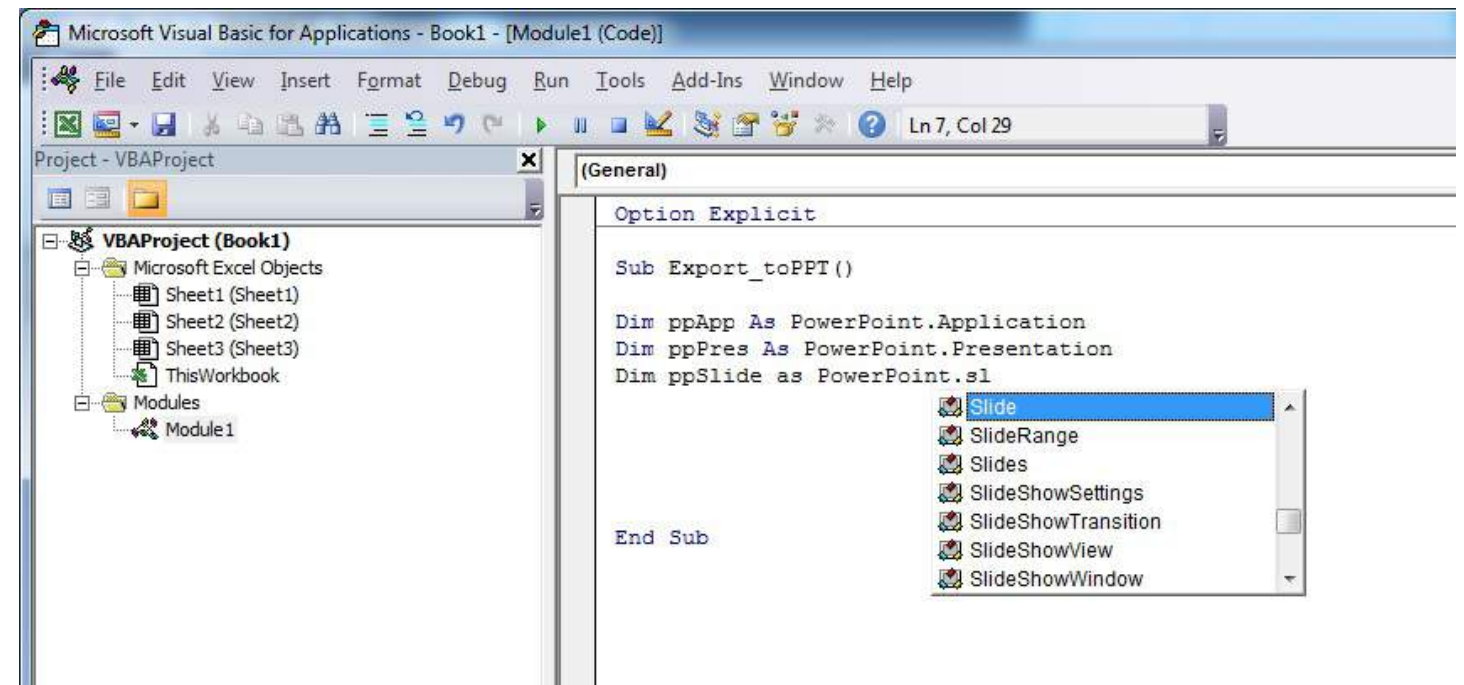
Dim ppApp As PowerPoint.Application
Dim ppPres As PowerPoint.Presentation
Dim ppSlide As PowerPoint.Slide

' 在这里写下你想用 PowerPoint 类和对象做的所有操作

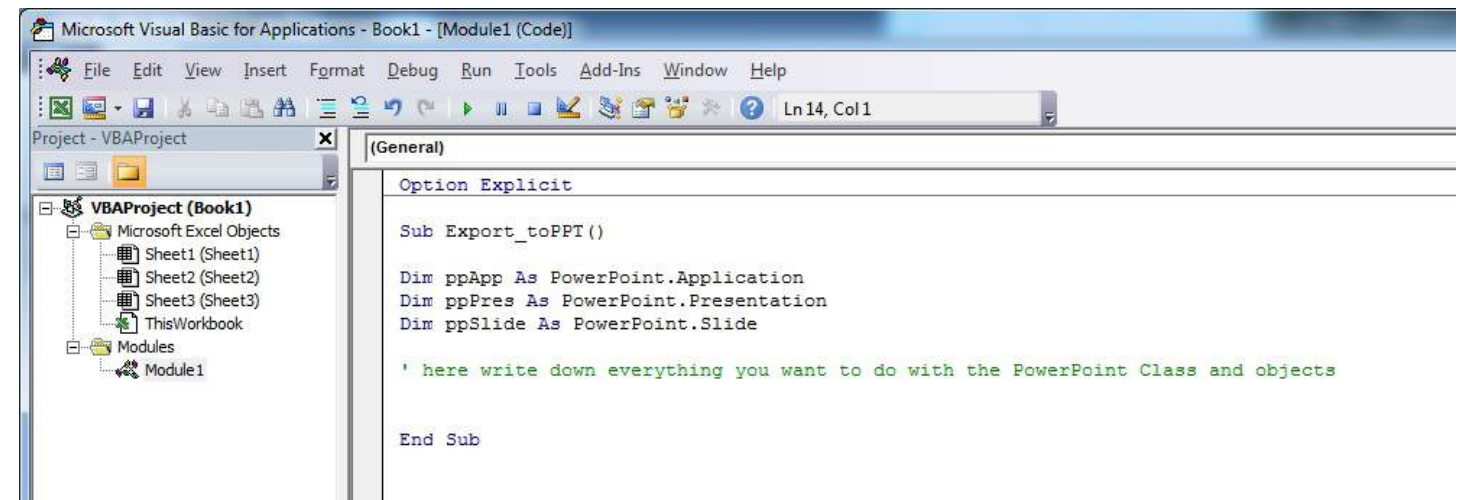
End Sub
```

## 第1.4节：你好，世界

1. 打开 Visual Basic 编辑器（参见打开 Visual Basic 编辑器）
2. 点击 插入 --> 模块 以添加一个新模块：



Now the variables declaration section looks like in the screen-shot below, and the user can start using these variables in his code.



Code version of this tutorial:

```
Option Explicit

Sub Export_toPPT()

Dim ppApp As PowerPoint.Application
Dim ppPres As PowerPoint.Presentation
Dim ppSlide As PowerPoint.Slide

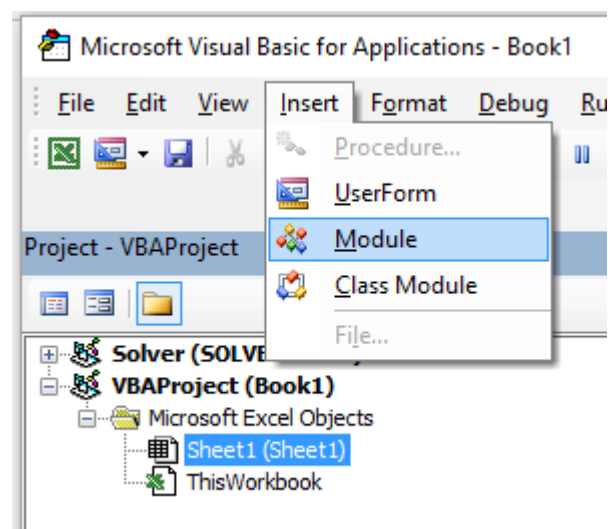
' here write down everything you want to do with the PowerPoint Class and objects

End Sub
```

## Section 1.4: Hello World

1. Open the Visual Basic Editor ( see Opening the Visual Basic Editor )
2. Click Insert --> Module to add a new Module :

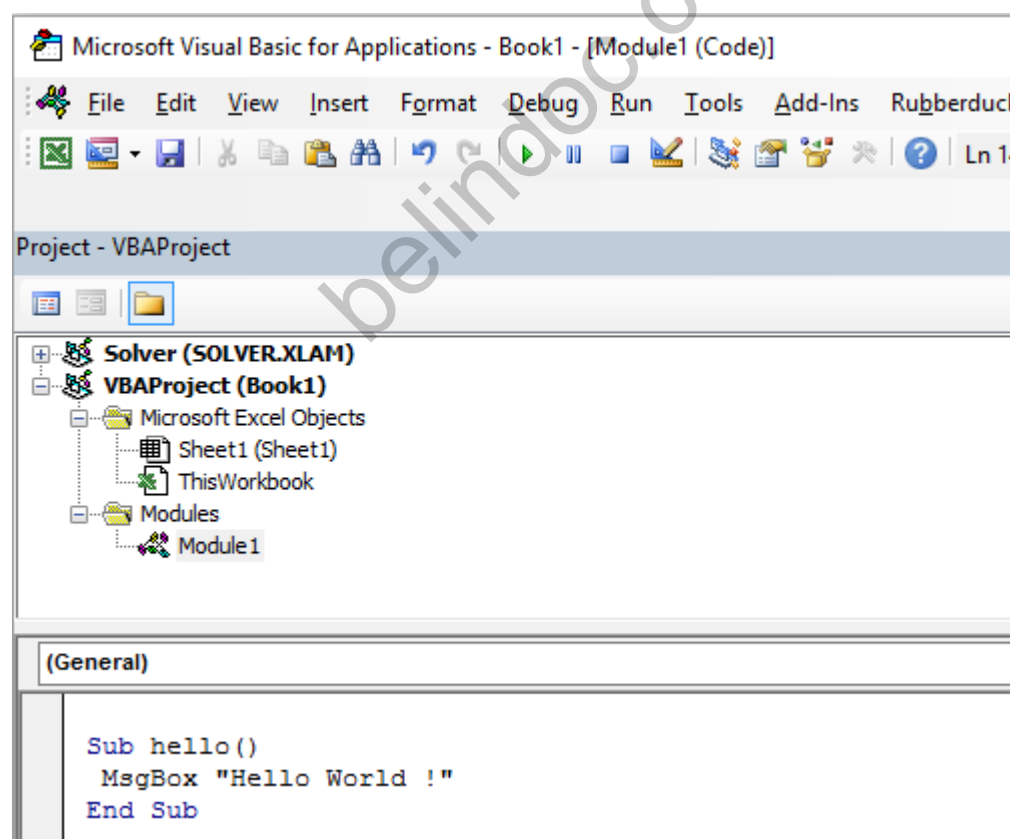




3. 将以下代码复制并粘贴到新模块中：

```
Sub hello()
MsgBox "你好，世界！"
End Sub
```

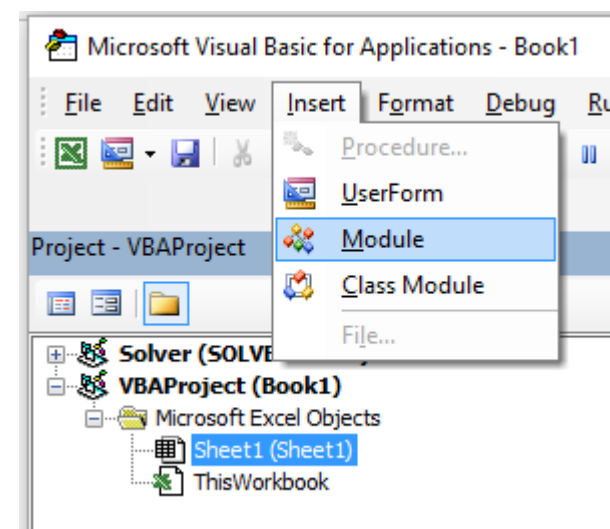
要获得：



4. 点击 Visual Basic 工具栏中的绿色“播放”箭头（或按 F5）运行程序：



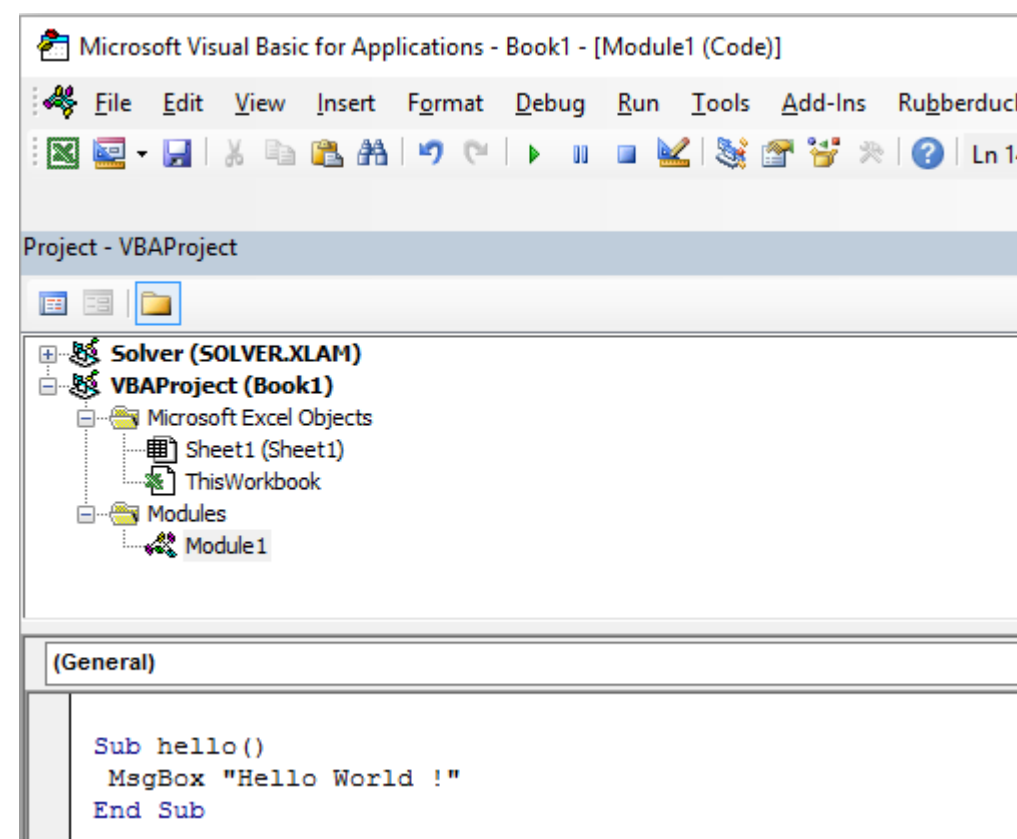
5. 选择新创建的子程序 "hello" 并点击 运行：



3. Copy and Paste the following code in the new module：

```
Sub hello()
MsgBox "Hello World !"
End Sub
```

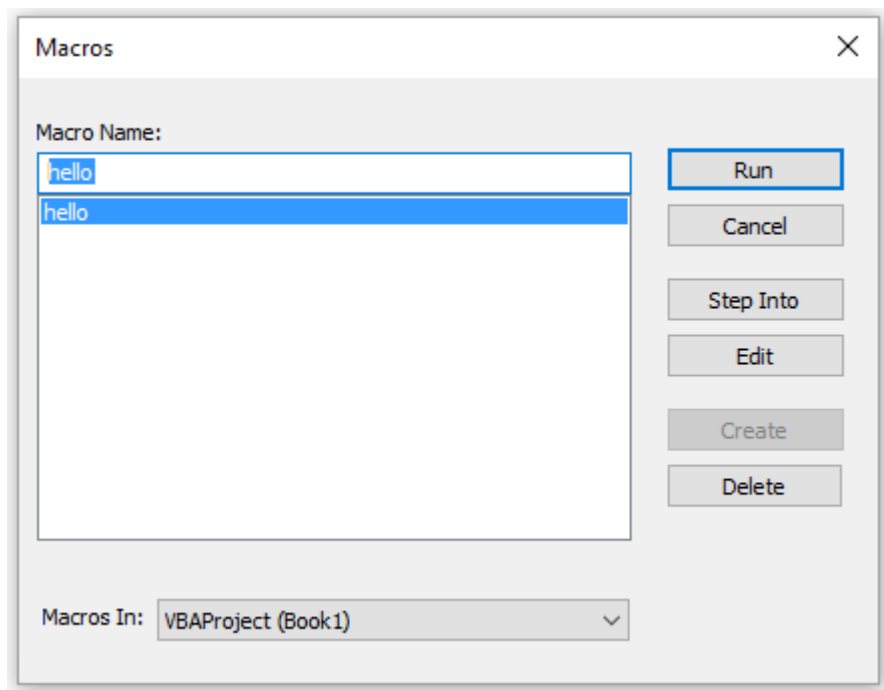
To obtain：



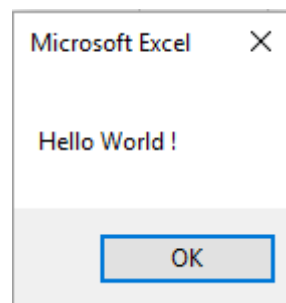
4. Click on the green “play” arrow (or press F5) in the Visual Basic toolbar to run the program:



5. Select the new created sub "hello" and click Run：



6.完成后，您应该看到以下窗口：

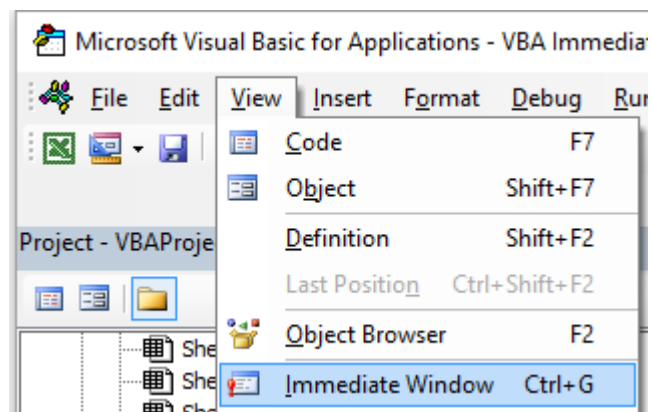


## 第1.5节：开始使用Excel对象模型

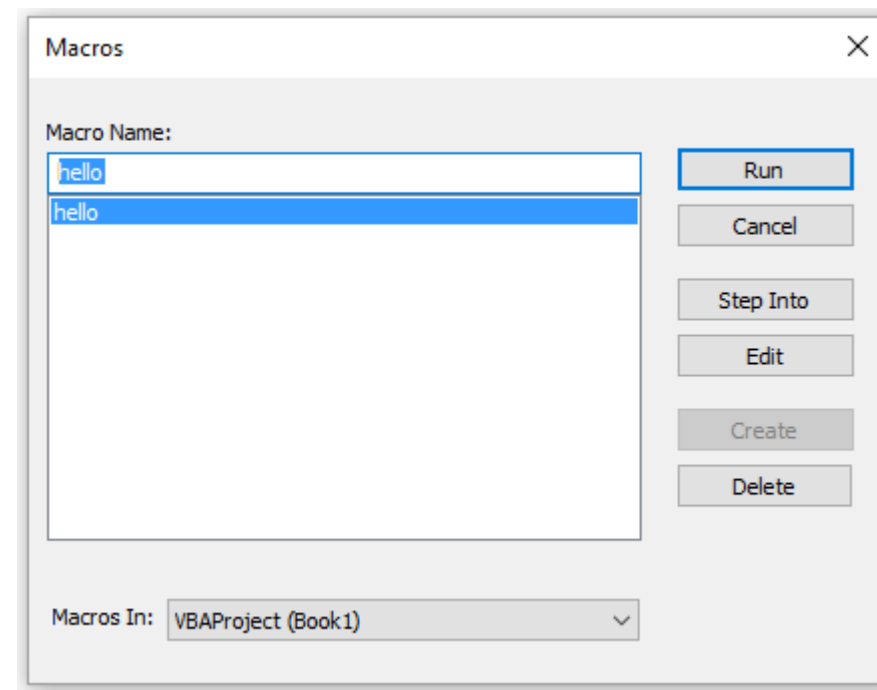
本示例旨在为初学者提供对Excel对象模型的温和介绍。

- 1.打开Visual Basic编辑器（VBE）
2. 点击视图 --> 立即窗口以打开立即窗口（或

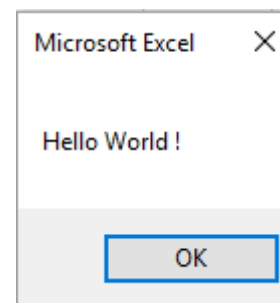
`ctrl` + `G`



3.您应该在VBE底部看到以下立即窗口：



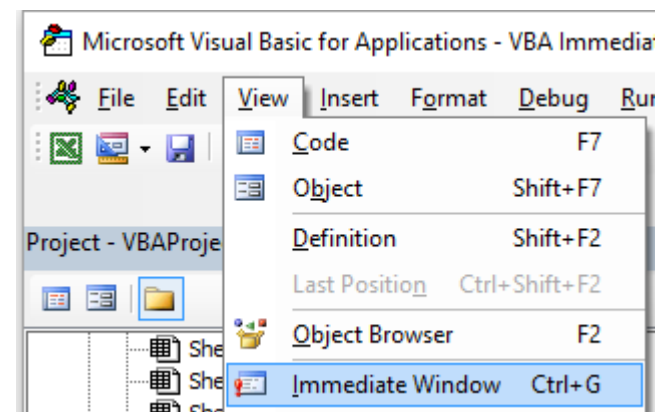
6. Done, your should see the following window:



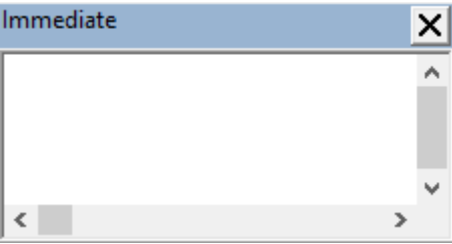
## Section 1.5: Getting Started with the Excel Object Model

This example intend to be a gentle introduction to the Excel Object Model **for beginners**.

1. Open the Visual Basic Editor (VBE)
2. Click View --> Immediate Window to open the Immediate Window (or `ctrl` + `G`):



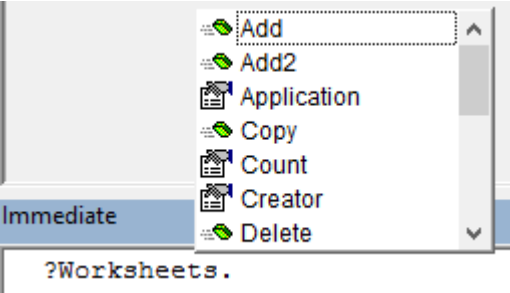
3. You should see the following Immediate Window at the bottom on VBE:



此窗口允许您直接测试一些VBA代码。那我们开始吧，在此控制台中输入：

```
?Worksheets.
```

VBE具有智能感知功能，应该会像下图一样弹出提示框：



在列表中选择.Count，或直接输入.Count，得到：

```
?Worksheets.Count
```

- 4. 然后按回车。表达式被计算，应该返回1。这表示当前工作簿中存在的工作表数量。  
问号 (?) 是Debug.Print的别名。

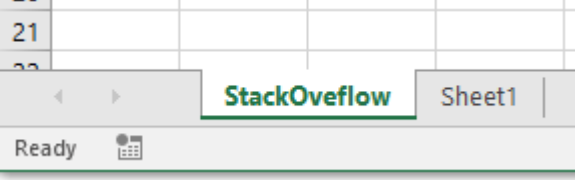
Worksheets是一个**对象**，Count是一个**方法**。Excel有多个对象（工作簿、工作表、区域、图表等），每个对象包含特定的方法和属性。您可以在[Excel VBA 参考文档](#)中找到完整的对象列表。Worksheets对象在此处[展示](#)。

此Excel VBA参考文档应成为您关于Excel对象模型的主要信息来源。

- 5. 现在让我们尝试另一个表达式，输入（不带?字符）：

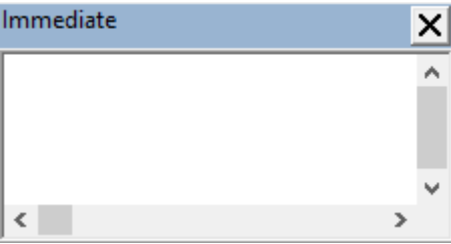
```
Worksheets.Add().Name = "StackOverflow"
```

- 6. 按回车键。这将创建一个名为StackOverflow的新工作表：



要理解这个表达式，你需要阅读前面提到的Excel参考中的Add函数。你会发现以下内容：

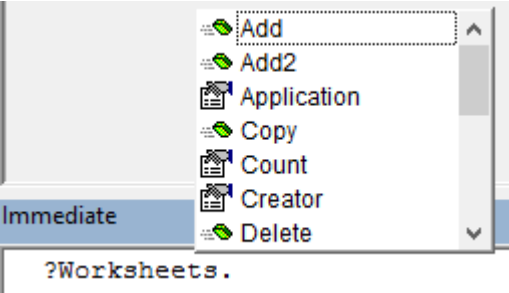
Add：创建一个新的工作表、图表或宏表。  
该新的工作表将成为活动工作表。  
返回值：一个对象值，表示新的工作表、图表，



This window allow you to directly test some VBA code. So let's start, type in this console :

```
?Worksheets.
```

VBE has intellisense and then it should open a tooltip as in the following figure :



Select .Count in the list or directly type .Cout to obtain :

```
?Worksheets.Count
```

- 4. Then press Enter. The expression is evaluated and it should returns 1. This indicates the number of Worksheet currently present in the workbook. The question mark (?) is an alias for Debug.Print.

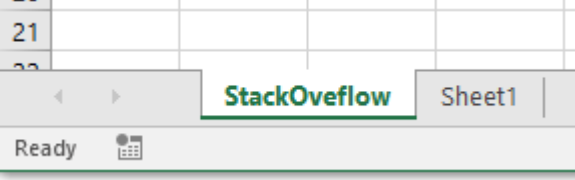
Worksheets is an **Object** and Count is a **Method**. Excel has several Object (Workbook, Worksheet, Range, Chart ..) and each of one contains specific methods and properties. You can find the complete list of Object in the [Excel VBA reference](#). Worksheets Object is presented [here](#) .

This Excel VBA reference should become your primary source of information regarding the Excel Object Model.

- 5. Now let's try another expression, type (without the ? character):

```
Worksheets.Add().Name = "StackOveflow"
```

- 6. Press Enter. This should create a new worksheet called StackOverf1ow.:



To understand this expression you need to read the Add function in the aforementioned Excel reference. You will find the following:

Add：Creates a **new** worksheet, chart, **or** macro sheet.  
The **new** worksheet becomes the active sheet.  
**Return** Value: An **Object** value that represents the **new** worksheet, chart,



或宏表。

因此，Worksheets.Add()会创建一个新工作表并返回它。Worksheet（无s）本身是文档中可以找到的一个对象，Name是其属性之一（见此处）。其定义为：

Worksheet.Name属性：返回或设置一个字符串值，表示该对象的名称。

因此，通过调查不同对象的定义，我们能够理解这段代码Worksheets.Add().Name = "StackOveflow"。

Add() 创建并添加一个新的工作表，并返回一个引用，然后我们将其名称属性设置为"StackOverflow"

现在让我们更正式地说明，Excel 包含多个对象。这些对象可能由一个或多个相同类别的 Excel 对象集合组成。工作表（WorkSheets）就是一个工作表对象（Worksheet object）的集合。每个对象都有一些属性和方法，程序员可以与之交互。

Excel 对象模型指的是 Excel 的对象层级结构

所有对象的顶层是Application对象，它代表 Excel 实例本身。VBA 编程需要对该层级结构有良好的理解，因为我们总是需要一个对象的引用才能调用方法或设置/获取属性。

（非常简化的）Excel 对象模型可以表示为，

Application  
    Workbooks  
        Workbook  
        Worksheets  
            Worksheet  
                Range

下面展示了工作表对象的更详细版本（如 Excel 2007 中所示），

or macro sheet.

So the Worksheets.Add() create a new worksheet and return it. Worksheet(**without s**) is itself a Object that [can be found](#) in the documentation and Name is one of its **property** (see [here](#)). It is defined as :

Worksheet.Name **Property**: Returns or sets a String value that represents the object name.

So, by investigating the different objects definitions we are able to understand this code Worksheets.Add().Name = "StackOveflow".

Add() creates and add a new worksheet and return a **reference** to it, then we set its Name **property** to "StackOverflow"

Now let's be more formal, Excel contains several Objects. These Objects may be composed of one or several collection(s) of Excel objects of the same class. It is the case for WorkSheets which is a collection of Worksheet object. Each Object has some properties and methods that the programmer can interact with.

The Excel Object model refers to the Excel **object hierarchy**

At the top of all objects is the Application object, it represents the Excel instance itself. Programming in VBA requires a good understanding of this hierarchy because we always need a reference to an object to be able to call a Method or to Set/Get a property.

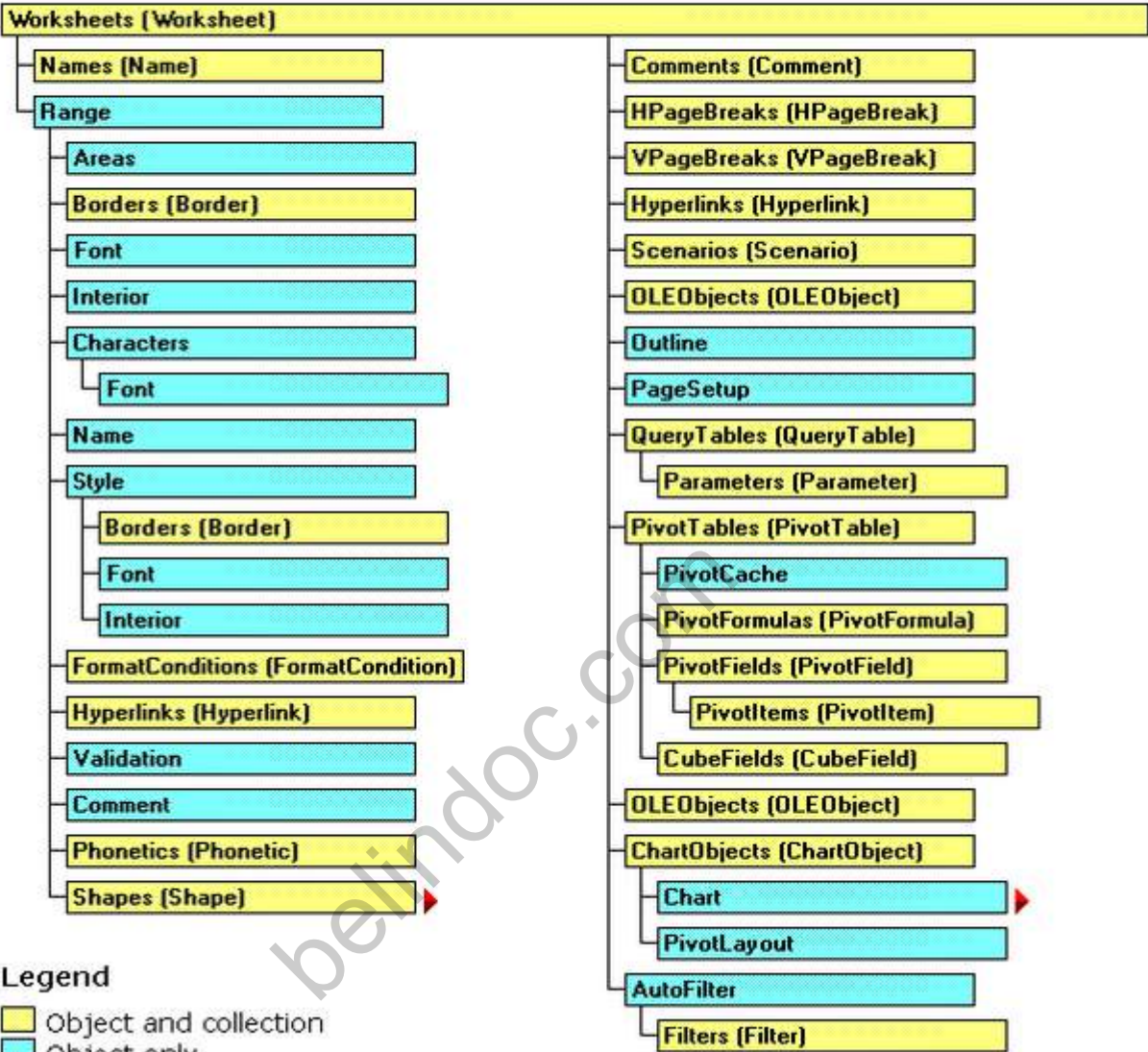
The (very simplified) Excel Object Model can be represented as,

Application  
    Workbooks  
        Workbook  
        Worksheets  
            Worksheet  
                Range

A more detail version for the Worksheet Object (as it is in Excel 2007) is shown below,

Microsoft Excel Objects (Worksheet)

See Also

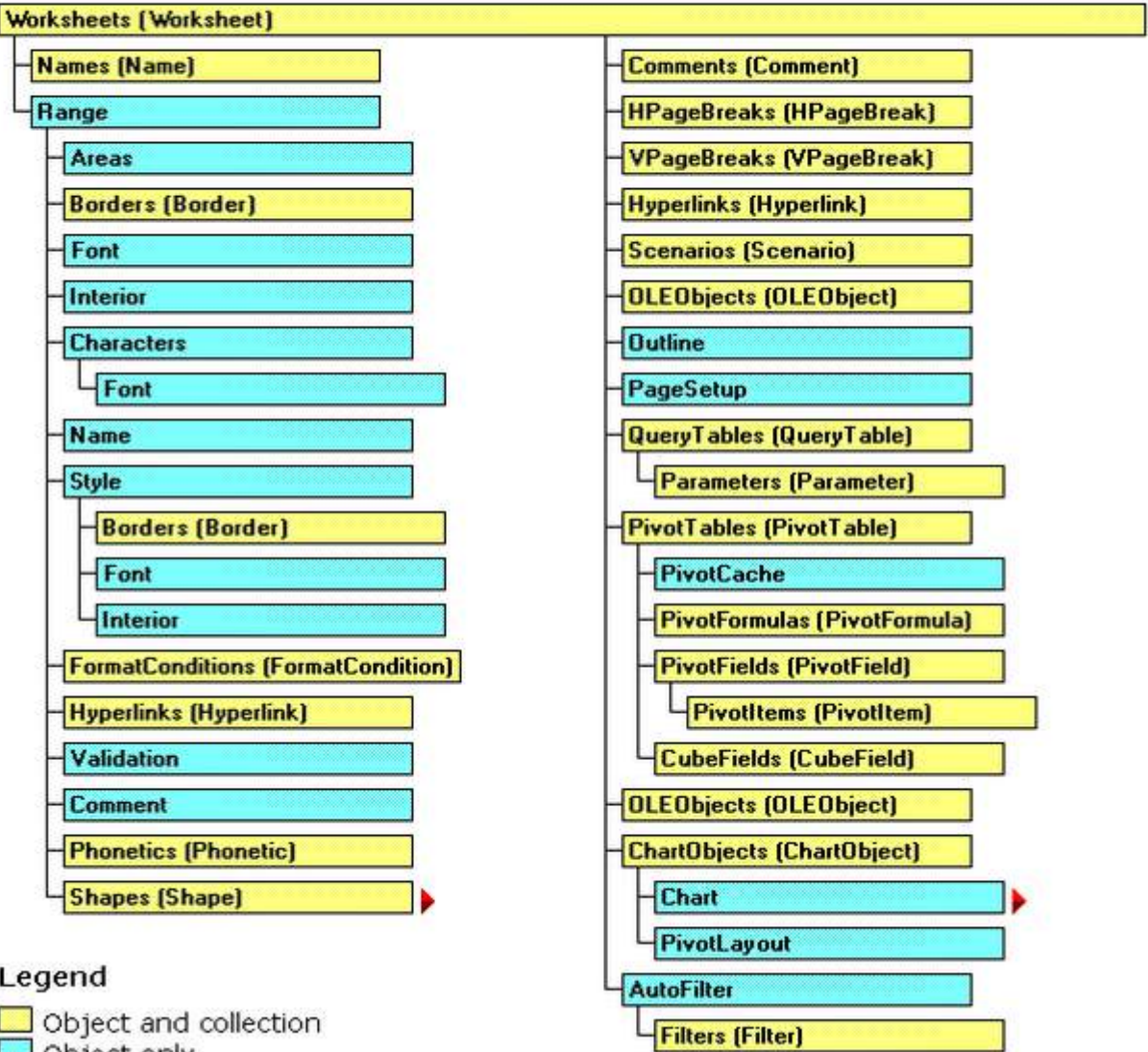


完整的 Excel 对象模型可以在这里找到。

最后，某些对象可能具有事件（例如：Workbook.WindowActivate），这些事件也是 Excel 对象模型的一部分。

Microsoft Excel Objects (Worksheet)

See Also



The full Excel Object Model can be found [here](#).

Finally some objects may have events (ex: Workbook.WindowActivate) that are also part of the Excel Object Model.

# 第二章：数组

## 第2.1节：动态数组（数组调整大小和动态处理）

由于内容不专属于Excel-VBA，本示例已移至VBA文档。

链接：动态数组（数组调整大小和动态处理）

## 第2.2节：填充数组（添加值）

有多种方法可以填充数组。

### 直接填充

```
'一维数组  
Dim arrayDirect1D(2) As String  
arrayDirect(0) = "A"  
arrayDirect(1) = "B"  
arrayDirect(2) = "C"
```

```
'多维数组（此处为3维）  
Dim arrayDirectMulti(1, 1, 2)  
arrayDirectMulti(0, 0, 0) = "A"  
arrayDirectMulti(0, 0, 1) = "B"  
arrayDirectMulti(0, 0, 2) = "C"  
arrayDirectMulti(0, 1, 0) = "D"  
.....
```

### 使用 Array() 函数

```
'仅限一维  
Dim array1D As Variant '必须是 Variant 类型  
array1D = Array(1, 2, "A")  
'-> array1D(0) = 1, array1D(1) = 2, array1D(2) = "A"
```

### 从区域获取

```
Dim arrayRange As Variant '必须是 Variant 类型  
  
'将区域放入数组总是创建二维数组（即使只有1行或1列）  
'索引从1开始而非0，第一维是行，第二维是列  
arrayRange = Range("A1:C10").Value  
'-> arrayRange(1,1) = A1 单元格的值  
'-> arrayRange(1,2) = B1 单元格的值  
'-> arrayRange(5,3) = C5 单元格的值  
'....  
  
'你可以通过使用工作表函数 Index 和 Transpose  
'从区域（行或列）获取一维数组：  
  
'将区域中的一行转换为一维数组：  
arrayRange = Application.WorksheetFunction.Index(Range("A1:C10").Value, 3, 0)  
将范围的第3行转换为一维数组  
'-> arrayRange(1) = A3单元格的值, arrayRange(2) = B3单元格的值, arrayRange(3) = C3单元格的值  
  
'将一列转换为一维数组：  
'限制为列中65536行，原因：.Transpose的限制
```

# Chapter 2: Arrays

## Section 2.1: Dynamic Arrays (Array Resizing and Dynamic Handling)

Due to not being Excel-VBA exclusive contents this Example has been moved to VBA documentation.

Link: Dynamic Arrays (Array Resizing and Dynamic Handling)

## Section 2.2: Populating arrays (adding values)

There are multiple ways to populate an array.

### Directly

```
'one-dimensional  
Dim arrayDirect1D(2) As String  
arrayDirect(0) = "A"  
arrayDirect(1) = "B"  
arrayDirect(2) = "C"
```

```
'multi-dimensional (in this case 3D)  
Dim arrayDirectMulti(1, 1, 2)  
arrayDirectMulti(0, 0, 0) = "A"  
arrayDirectMulti(0, 0, 1) = "B"  
arrayDirectMulti(0, 0, 2) = "C"  
arrayDirectMulti(0, 1, 0) = "D"  
'....
```

### Using Array() function

```
'one-dimensional only  
Dim array1D As Variant 'has to be type variant  
array1D = Array(1, 2, "A")  
'-> array1D(0) = 1, array1D(1) = 2, array1D(2) = "A"
```

### From range

```
Dim arrayRange As Variant 'has to be type variant  
  
'putting ranges in an array always creates a 2D array (even if only 1 row or column)  
'starting at 1 and not 0, first dimension is the row and the second the column  
arrayRange = Range("A1:C10").Value  
'-> arrayRange(1,1) = value in A1  
'-> arrayRange(1,2) = value in B1  
'-> arrayRange(5,3) = value in C5  
'....  
  
'Yoo can get an one-dimensional array from a range (row or column)  
'by using the worksheet functions index and transpose:  
  
'one row from range into 1D-Array:  
arrayRange = Application.WorksheetFunction.Index(Range("A1:C10").Value, 3, 0)  
'-> row 3 of range into 1D-Array  
'-> arrayRange(1) = value in A3, arrayRange(2) = value in B3, arrayRange(3) = value in C3  
  
'one column into 1D-Array:  
'limited to 65536 rows in the column, reason: limit of .Transpose
```



```
arrayRange = Application.WorksheetFunction.Index( _  
Application.WorksheetFunction.Transpose(Range("A1:C10").Value), 2, 0)  
'-> 将范围的第2列转换为一维数组  
'-> arrayRange(1) = B1单元格的值, arrayRange(2) = B2单元格的值, arrayRange(3) = B3单元格的值  
'...  
  
'通过使用Evaluate() - 简写为[] - 你可以同时将  
范围转换为数组并更改其值。  
'这相当于工作表中的数组公式：  
arrayRange = [(A1:C10*3)]  
arrayRange = [(A1:C10&"_test")]  
arrayRange = [(A1:B10*C1:C10)]  
.....
```

使用Evaluate()的二维数组

```
Dim array2D As Variant  
'[] 是 evaluate() 的简写  
'使用 evaluate 定义的数组起始索引为 1 而非 0  
array2D = [{"1A","1B","1C";"2A","2B","3B"}]  
'-> array2D(1,1) = "1A", array2D(1,2) = "1B", array2D(2,1) = "2A" ...  
  
'如果你想用字符串来填充二维数组：  
Dim strValues As String  
strValues = "{"&"1A"&","&"1B"&","&"1C"&";"&"2A"&","&"2B"&","&"2C"&"}"  
array2D = Evaluate(strValues)
```

使用 Split() 函数

```
Dim arraySplit As Variant '必须是 Variant 类型  
arraySplit = Split("a,b,c", ",")  
'-> arraySplit(0) = "a", arraySplit(1) = "b", arraySplit(2) = "c"
```

第 2.3 节：锯齿数组（数组的数组）

由于内容不专属于Excel-VBA，本示例已移至VBA文档。

链接：锯齿数组（数组的数组）

第2.4节：检查数组是否已初始化（是否包含元素）

一个常见的问题可能是尝试遍历没有任何值的数组。例如：

```
Dim myArray() As Integer  
For i = 0 To UBound(myArray) '将导致“下标越界”错误
```

为避免此问题，并检查数组是否包含元素，请使用以下一行代码：

```
If Not Not myArray Then MsgBox UBound(myArray) Else MsgBox "myArray 未初始化"
```

第2.5节：动态数组【数组声明，调整大小】

```
Sub Array_clarity()  
  
Dim arr() As Variant '创建一个空数组  
Dim x As Long  
Dim y As Long
```

```
arrayRange = Application.WorksheetFunction.Index( _  
Application.WorksheetFunction.Transpose(Range("A1:C10").Value), 2, 0)  
'-> column 2 of range into 1D-Array  
'-> arrayRange(1) = value in B1, arrayRange(2) = value in B2, arrayRange(3) = value in B3  
'...  
  
'By using Evaluate() - shorthand [] - you can transfer the  
'range to an array and change the values at the same time.  
'This is equivalent to an array formula in the sheet:  
arrayRange = [(A1:C10*3)]  
arrayRange = [(A1:C10&"_test")]  
arrayRange = [(A1:B10*C1:C10)]  
'...
```

2D with Evaluate()

```
Dim array2D As Variant  
'[] ist a shorthand for evaluate()  
'Arrays defined with evaluate start at 1 not 0  
array2D = [{"1A","1B","1C";"2A","2B","3B"}]  
'-> array2D(1,1) = "1A", array2D(1,2) = "1B", array2D(2,1) = "2A" ...  
  
'if you want to use a string to fill the 2D-Array:  
Dim strValues As String  
strValues = "{"&"1A"&","&"1B"&","&"1C"&";"&"2A"&","&"2B"&","&"2C"&"}"  
array2D = Evaluate(strValues)
```

Using Split() function

```
Dim arraySplit As Variant 'has to be type variant  
arraySplit = Split("a,b,c", ",")  
'-> arraySplit(0) = "a", arraySplit(1) = "b", arraySplit(2) = "c"
```

Section 2.3: Jagged Arrays (Arrays of Arrays)

Due to not being Excel-VBA exclusive contents this Example has been moved to VBA documentation.

Link: Jagged Arrays (Arrays of Arrays)

Section 2.4: Check if Array is Initialized (If it contains elements or not)

A common problem might be trying to iterate over Array which has no values in it. For example:

```
Dim myArray() As Integer  
For i = 0 To UBound(myArray) 'Will result in a "Subscript Out of Range" error
```

To avoid this issue, and to check if an Array contains elements, use this *oneliner*:

```
If Not Not myArray Then MsgBox UBound(myArray) Else MsgBox "myArray not initialised"
```

Section 2.5: Dynamic Arrays [Array Declaration, Resizing]

```
Sub Array_clarity()  
  
Dim arr() As Variant 'creates an empty array  
Dim x As Long  
Dim y As Long
```

```
x = Range("A1", Range("A1").End(xlDown)).Cells.Count
y = Range("A1", Range("A1").End(xlToRight)).Cells.Count
```

```
ReDim arr(0 To x, 0 To y) '固定数组大小
```

```
For x = LBound(arr, 1) To UBound(arr, 1)
    For y = LBound(arr, 2) To UBound(arr, 2)
        arr(x, y) = Range("A1").Offset(x, y) '将 activesheet 中 Range("A1:E10") 的值存储到 x 和 y 变量中
```

```
    Next
Next
```

```
'根据声明放置到同一工作表上：
Range("A14").Resize(UBound(arr, 1), UBound(arr, 2)).Value = arr
```

```
End Sub
```

```
x = Range("A1", Range("A1").End(xlDown)).Cells.Count
y = Range("A1", Range("A1").End(xlToRight)).Cells.Count
```

```
ReDim arr(0 To x, 0 To y) 'fixing the size of the array
```

```
For x = LBound(arr, 1) To UBound(arr, 1)
    For y = LBound(arr, 2) To UBound(arr, 2)
        arr(x, y) = Range("A1").Offset(x, y) 'storing the value of Range("A1:E10") from activesheet
        in x and y variables
    Next
Next
```

```
'Put it on the same sheet according to the declaration:
Range("A14").Resize(UBound(arr, 1), UBound(arr, 2)).Value = arr
```

```
End Sub
```

# 第3章：条件语句

## 第3.1节：If语句

If 控制语句允许根据条件（布尔）语句的评估执行不同的代码。条件语句是指其结果为 True 或 False 的语句，例如 `x > 2`。

实现If语句时可以使用三种模式，具体如下所述。请注意，If条件判断后总是紧跟着一个Then。

### 1. 评估一个If条件语句，并在其为True时执行某操作

#### 单行If语句

这是使用If的最简方式，当仅需在True评估时执行一条语句时非常有用。使用此语法时，所有代码必须写在同一行。行尾不要包含End If。

```
If [某条件为 True] Then [执行某操作]
```

#### If代码块

如果需要在True评估时执行多行代码，可以使用If代码块。

```
If [某条件为 True] Then
    [执行多条操作]
End If
```

注意，如果使用多行If代码块，则必须有对应的End If。

### 2. 评估一个条件If语句，若为True执行一件事，若为False

#### 单行If, Else语句

当在True评估时执行一条语句，而在False评估时执行另一条语句时，可以使用此语法。使用此语法时要小心，因为读者往往不容易看出存在Else语句。使用此语法时，所有代码必须写在同一行。行尾不要包含End If。

```
If[某条件为True]Then[执行某操作]Else[执行其他操作]
```

#### If, Else代码块

使用If, Else代码块可以使代码更清晰，或者当在True或False评估下需要执行多行代码时使用。

```
If [某条件为 True] Then
    [执行多条操作]
Else
    [执行其他操作]
End If
```

注意，如果使用多行If代码块，则必须有对应的End If。

# Chapter 3: Conditional statements

## Section 3.1: The If statement

The If control statement allows different code to be executed depending upon the evaluation of a conditional (Boolean) statement. A conditional statement is one that evaluates to either **True** or **False**, e.g. `x > 2`.

There are three patterns that can be used when implementing an If statement, which are described below. Note that an If conditional evaluation is always followed by a **Then**.

### 1. Evaluating one If conditional statement and doing something if it is True

#### Single line If statement

This is the shortest way to use an If and it is useful when only one statement needs to be carried out upon a **True** evaluation. When using this syntax, all of the code must be on a single line. Do not include an **End If** at the end of the line.

```
If [Some condition is True] Then [Do something]
```

#### If block

If multiple lines of code need to be executed upon a **True** evaluation, an If block may be used.

```
If [Some condition is True] Then
    [Do some things]
End If
```

Note that, if a multi-line If block is used, a corresponding **End If** is required.

### 2. Evaluating one conditional If statement, doing one thing if it is True and doing something else if it is False

#### Single line If, Else statement

This may be used if one statement is to be carried out upon a **True** evaluation and a different statement is to be carried out on a **False** evaluation. Be careful using this syntax, as it is often less clear to readers that there is an **Else** statement. When using this syntax, all of the code must be on a single line. Do not include an **End If** at the end of the line.

```
If [Some condition is True] Then [Do something] Else [Do something else]
```

#### If, Else block

Use an If, **Else** block to add clarity to your code, or if multiple lines of code need to be executed under either a **True** or a **False** evaluation.

```
If [Some condition is True] Then
    [Do some things]
Else
    [Do some other things]
End If
```

Note that, if a multi-line If block is used, a corresponding **End If** is required.

3. 评估多个条件语句，当前面的语句均为False时，对每个条件执行不同操作

此模式是If最通用的用法，适用于多个不重叠的条件需要不同处理的情况。与前两种模式不同，即使每个条件只执行一行代码，也需要使用If代码块。

If, ElseIf, ..., Else 代码块

与其一个接一个地创建许多If块，不如使用ElseIf来评估额外的条件。只有当前面任何If的评估结果为False时，才会评估ElseIf。

```
如果[某个条件为True]则
    [执行某些操作]
ElseIf[另一个条件为True]则
    [执行一些不同的操作]
Else'以上所有条件均评估为False
    [执行其他操作]
End If
```

在If和End If之间可以根据需要包含任意数量的ElseIf控制语句。使用ElseIf时不要求必须有Else控制语句（尽管推荐使用），但如果包含，必须是End If之前的最后一个控制语句。

3. Evaluating many conditional statements, when preceding statements are all False, and doing something different for each one

This pattern is the most general use of If and would be used when there are many non-overlapping conditions that require different treatment. Unlike the first two patterns, this case requires the use of an If block, even if only one line of code will be executed for each condition.

If, ElseIf, ..., Else block

Instead of having to create many If blocks one below another, an ElseIf may be used evaluate an extra condition. The ElseIf is only evaluated if any preceding If evaluation is False.

```
If [Some condition is True] Then
    [Do some thing(s)]
ElseIf [Some other condition is True] Then
    [Do some different thing(s)]
Else 'Everything above has evaluated to False
    [Do some other thing(s)]
End If
```

As many ElseIf control statements may be included between an If and an End If as required. An Else control statement is not required when using ElseIf (although it is recommended), but if it is included, it must be the final control statement before the End If.



# 第4章：区域和单元格

## 第4.1节：引用单个单元格的方法

引用当前Excel工作表上单个单元格的最简单方法是将其A1形式的引用用方括号括起来：

```
[a3] = "Hello!"
```

请注意，方括号只是对Application对象的Evaluate方法的方便语法糖，因此从技术上讲，这与以下代码是相同的：

```
Application.Evaluate("a3") = "Hello!"
```

你也可以调用Cells方法，该方法接受行和列参数并返回单元格引用。

```
Cells(3, 1).Formula = "=A1+A2"
```

请记住，每当你从VBA向Excel传递行和列时，行总是第一个，随后是列，这很容易让人困惑，因为这与常见的A1表示法相反，后者是列先出现。

在这两个例子中，我们都没有指定工作表，因此Excel将使用活动工作表（用户界面中位于前面的工作表）。你可以显式指定活动工作表：

```
ActiveSheet.Cells(3, 1).Formula = "=SUM(A1:A2)"
```

或者你可以提供特定工作表的名称：

```
Sheets("Sheet2").Cells(3, 1).Formula = "=SUM(A1:A2)"
```

有多种方法可以从一个区域定位到另一个区域。例如，Rows方法可以用来获取任何区域的单独行，Cells方法可以用来获取行或列的单个单元格，所以下面的代码指的是单元格C1：

```
ActiveSheet.Rows(1).Cells(3).Formula = "hi!"
```

## 第4.2节：创建范围

范围（Range）不能像字符串那样创建或赋值：

```
Sub RangeTest()  
    Dim s As String  
        Dim r As Range '特定类型的对象，具有 Address、WrapText、AutoFill 等成员  
  
    ' 这是给字符串赋值的方式：  
    s = "Hello World!"  
  
    ' 但我们不能这样给范围赋值：  
    r = Range("A1") '//运行错误：91 对象变量或 With 块变量未设置//  
  
    ' 我们必须使用对象方式，使用关键字 Set：  
    设置 r = Range("A1")  
End Sub
```

# Chapter 4: Ranges and Cells

## Section 4.1: Ways to refer to a single cell

The simplest way to refer to a single cell on the current Excel worksheet is simply to enclose the A1 form of its reference in square brackets:

```
[a3] = "Hello!"
```

Note that square brackets are just convenient [syntactic sugar](#) for the Evaluate method of the Application object, so technically, this is identical to the following code:

```
Application.Evaluate("a3") = "Hello!"
```

You could also call the Cells method which takes a row and a column and returns a cell reference.

```
Cells(3, 1).Formula = "=A1+A2"
```

Remember that whenever you pass a row and a column to Excel from VBA, the row is always first, followed by the column, which is confusing because it is the opposite of the common A1 notation where the column appears first.

In both of these examples, we did not specify a worksheet, so Excel will use the active sheet (the sheet that is in front in the user interface). You can specify the active sheet explicitly:

```
ActiveSheet.Cells(3, 1).Formula = "=SUM(A1:A2)"
```

Or you can provide the name of a particular sheet:

```
Sheets("Sheet2").Cells(3, 1).Formula = "=SUM(A1:A2)"
```

There are a wide variety of methods that can be used to get from one range to another. For example, the Rows method can be used to get to the individual rows of any range, and the Cells method can be used to get to individual cells of a row or column, so the following code refers to cell C1:

```
ActiveSheet.Rows(1).Cells(3).Formula = "hi!"
```

## Section 4.2: Creating a Range

A [Range](#) cannot be created or populated the same way a string would:

```
Sub RangeTest()  
    Dim s As String  
    Dim r As Range 'Specific Type of Object, with members like Address, WrapText, AutoFill, etc.  
  
    ' This is how we fill a String:  
    s = "Hello World!"  
  
    ' But we cannot do this for a Range:  
    r = Range("A1") '//Run. Err.: 91 Object variable or With block variable not set//  
  
    ' We have to use the Object approach, using keyword Set:  
    Set r = Range("A1")  
End Sub
```

通常认为最好对引用进行限定，因此从现在开始我们将在此处采用相同的方法。  
更多关于创建对象变量（例如 Range）请参见 MSDN。更多关于Set 语句请参见 MSDN。

创建相同 Range 的方法有多种：

```
子程序 SetRangeVariable()  
    声明 ws 作为 工作表  
    声明 r 作为 范围  
  
    设置 ws = ThisWorkbook.Worksheets(1) ' 该代码所在工作簿中的第一个工作表  
  
    ' 以下均等效：  
    设置 r = ws.Range("A2")  
    设置 r = ws.Range("A" & 2)  
    设置 r = ws.Cells(2, 1) ' 第2行第1列的单元格  
    设置 r = ws.[A2] ' Range 的简写表示法。  
    设置 r = Range("NamedRangeInA2") '如果单元格A2被命名为NamedRangeInA2。注意，这与工作表无关。  
  
    设置 r = ws.Range("A1").Offset(1, 0) ' 单元格距离A1向下1行，向右0列  
    设置 r = ws.Range("A1").Cells(2,1) ' 类似于Offset。你可以“超出”原始范围。  
  
    设置 r = ws.Range("A1:A5").Cells(2) ' 较大范围内的第二个单元格。  
    设置 r = ws.Range("A1:A5").Item(2) ' 较大范围内的第二个单元格。  
    设置 r = ws.Range("A1:A5")(2) ' 较大范围内的第二个单元格。  
End Sub
```

注意示例中Cells(2, 1)等同于Range("A2")。这是因为Cells返回的是Range对象。  
一些参考资料：[Chip Pearson-Cells Within Ranges](#)；[MSDN-Range Object](#)；[John Walkenback-Referring To Ranges In Your VBA Code。](#)

还要注意，在任何声明范围时使用数字且数字本身不在引号内的情况，如Range("A" & 2)，你可以将该数字替换为包含整数/长整数的变量。例如：

```
子程序 RangeIteration()  
    定义 wb 作为 工作簿, ws 作为 工作表  
    定义 r 作为 范围  
  
    设置 wb = ThisWorkbook  
    设置 ws = wb.Worksheets(1)  
  
    循环 i = 1 到 10  
        设置 r = ws.Range("A" & i)  
        ' 当 i = 1 时, 结果为 Range("A1")  
        ' 当 i = 2 时, 结果为 Range("A2")  
        ' 依此类推。  
        证明：  
        Debug.Print r.Address  
    Next i  
End Sub
```

如果使用双重循环，Cells 更好：

```
Sub RangeIteration2()  
    Dim wb As Workbook, ws As Worksheet  
    Dim r As Range  
  
    Set wb = ThisWorkbook  
    Set ws = ws.Worksheets(1)
```

It is considered best practice to qualify your references, so from now on we will use the same approach here.  
More about [Creating Object Variables \(e.g. Range\) on MSDN](#) . More about [Set Statement on MSDN](#).

There are different ways to create the same Range:

```
Sub SetRangeVariable()  
    Dim ws As Worksheet  
    Dim r As Range  
  
    Set ws = ThisWorkbook.Worksheets(1) ' The first Worksheet in Workbook with this code in it  
  
    ' These are all equivalent:  
    Set r = ws.Range("A2")  
    Set r = ws.Range("A" & 2)  
    Set r = ws.Cells(2, 1) ' The cell in row number 2, column number 1  
    Set r = ws.[A2] 'Shorthand notation of Range.  
    Set r = Range("NamedRangeInA2") 'If the cell A2 is named NamedRangeInA2. Note, that this is Sheet independent.  
    Set r = ws.Range("A1").Offset(1, 0) ' The cell that is 1 row and 0 columns away from A1  
    Set r = ws.Range("A1").Cells(2,1) ' Similar to Offset. You can "go outside" the original Range.  
  
    Set r = ws.Range("A1:A5").Cells(2) 'Second cell in bigger Range.  
    Set r = ws.Range("A1:A5").Item(2) 'Second cell in bigger Range.  
    Set r = ws.Range("A1:A5")(2) 'Second cell in bigger Range.  
End Sub
```

Note in the example that Cells(2, 1) is equivalent to Range("A2"). This is because Cells returns a Range object.  
Some sources: [Chip Pearson-Cells Within Ranges](#); [MSDN-Range Object](#); [John Walkenback-Referring To Ranges In Your VBA Code](#).

Also note that in any instance where a number is used in the declaration of the range, and the number itself is outside of quotation marks, such as Range("A" & 2), you can swap that number for a variable that contains an integer/long. For example:

```
Sub RangeIteration()  
    Dim wb As Workbook, ws As Worksheet  
    Dim r As Range  
  
    Set wb = ThisWorkbook  
    Set ws = ws.Worksheets(1)  
  
    For i = 1 To 10  
        Set r = ws.Range("A" & i)  
        ' When i = 1, the result will be Range("A1")  
        ' When i = 2, the result will be Range("A2")  
        ' etc.  
        ' Proof:  
        Debug.Print r.Address  
    Next i  
End Sub
```

If you are using double loops, Cells is better:

```
Sub RangeIteration2()  
    Dim wb As Workbook, ws As Worksheet  
    Dim r As Range  
  
    Set wb = ThisWorkbook  
    Set ws = ws.Worksheets(1)
```

```
For i = 1 To 10
    For j = 1 To 10
        Set r = ws.Cells(i, j)
        ' 当 i = 1 且 j = 1 时, 结果是 Range("A1")
        ' 当 i = 2 且 j = 1 时, 结果是 Range("A2")
        ' 当 i = 1 且 j = 2 时, 结果是 Range("B1")
        ' 等等。
        ' 证明:
        Debug.Print r.Address
    下一 j
下一 i
End Sub
```

第4.3节：偏移属性

- Offset(行, 列) - 用于从当前单元格静态引用另一个点的操作符。通常用于循环中。应理解行部分的正数向右移动，负数向左移动。列部分的正数向下移动，负数向上移动。

即

```
Private Sub this()
ThisWorkbook.Sheets("Sheet1").Range("A1").Offset(1, 1).Select
    ThisWorkbook.Sheets("Sheet1").Range("A1").Offset(1, 1).Value = "New Value"
    ActiveCell.Offset(-1, -1).Value = ActiveCell.Value
    ActiveCell.Value = vbNullString
End Sub
```

此代码选择了B2，在那里放入一个新字符串，然后将该字符串移回A1，随后清空B2。

第4.4节：将单元格引用保存到变量中

要将单元格引用保存到变量中，必须使用Set语法，例如：

```
Dim R 作为 范围
设置 R = ActiveSheet.Cells(3, 1)
```

稍后...

```
R.Font.Color = RGB(255, 0, 0)
```

为什么需要Set关键字？Set告诉Visual Basic，等号右边的值表示的是一个对象。

第4.5节：如何转置范围（水平到垂直及反之）

```
子程序 TransposeRangeValues()
    Dim TmpArray() 作为 变体, FromRange 作为 范围, ToRange 作为 范围

    设置 FromRange = Sheets("Sheet1").Range("a1:a12")           'Worksheets(1).Range("a1:p1")
    设置 ToRange = ThisWorkbook.Sheets("Sheet1").Range("a1")
    'ThisWorkbook.Sheets("Sheet1").Range("a1")

    TmpArray = Application.Transpose(FromRange.Value)
    FromRange.Clear
```

```
For i = 1 To 10
    For j = 1 To 10
        Set r = ws.Cells(i, j)
        ' When i = 1 and j = 1, the result will be Range("A1")
        ' When i = 2 and j = 1, the result will be Range("A2")
        ' When i = 1 and j = 2, the result will be Range("B1")
        ' etc.
        ' Proof:
        Debug.Print r.Address
    Next j
Next i
End Sub
```

Section 4.3: Offset Property

- **Offset(Rows, Columns)** - The operator used to statically reference another point from the current cell. Often used in loops. It should be understood that positive numbers in the rows section moves right, whereas negatives move left. With the columns section positives move down and negatives move up.

i.e

```
Private Sub this()
    ThisWorkbook.Sheets("Sheet1").Range("A1").Offset(1, 1).Select
    ThisWorkbook.Sheets("Sheet1").Range("A1").Offset(1, 1).Value = "New Value"
    ActiveCell.Offset(-1, -1).Value = ActiveCell.Value
    ActiveCell.Value = vbNullString
End Sub
```

This code selects B2, puts a new string there, then moves that string back to A1 afterwards clearing out B2.

Section 4.4: Saving a reference to a cell in a variable

To save a reference to a cell in a variable, you must use the Set syntax, for example:

```
Dim R as Range
Set R = ActiveSheet.Cells(3, 1)
```

later...

```
R.Font.Color = RGB(255, 0, 0)
```

Why is the Set keyword required? Set tells Visual Basic that the value on the right hand side of the = is meant to be an object.

Section 4.5: How to Transpose Ranges (Horizontal to Vertical & vice versa)

```
Sub TransposeRangeValues()
    Dim TmpArray() As Variant, FromRange as Range, ToRange as Range

    set FromRange = Sheets("Sheet1").Range("a1:a12")           'Worksheets(1).Range("a1:p1")
    set ToRange = ThisWorkbook.Sheets("Sheet1").Range("a1")
    'ThisWorkbook.Sheets("Sheet1").Range("a1")

    TmpArray = Application.Transpose(FromRange.Value)
    FromRange.Clear
```

```
    ToRange.Resize(FromRange.Columns.Count, FromRange.Rows.Count).Value2 = TmpArray
End Sub
```

注意：复制/选择性粘贴也有“粘贴转置”选项，该选项会更新转置单元格中的公式。

belindoc.com

```
    ToRange.Resize(FromRange.Columns.Count, FromRange.Rows.Count).Value2 = TmpArray
End Sub
```

Note: Copy/PasteSpecial also has a Paste Transpose option which updates the transposed cells' formulas as well.



# 第5章：命名区域

主题应包含与Excel中命名区域相关的具体信息，包括创建、修改、删除和访问已定义命名区域的方法。

## 第5.1节：定义命名区域

使用命名区域可以描述单元格内容的含义，并使用该定义名称代替实际的单元格地址。

例如，公式=A5\*B5可以替换为=Width\*Height，使公式更易于阅读和理解。

要定义新的命名区域，选择要命名的单元格，然后在公式栏旁边的名称框中输入新名称。

	A1					
	A	B	C	D	E	F
1						
2						
3						
4						
5	15	20				
6						
7						

注意：命名区域默认具有全局作用域，意味着它们可以在工作簿的任何位置访问。旧版本的Excel允许重复名称，因此必须注意防止全局作用域的名称重复，否则结果将不可预测。可使用“公式”选项卡中的名称管理器更改作用域。

## 第5.2节：在VBA中使用命名范围

创建名为'MyRange'的新命名范围，分配给单元格A1

```
ThisWorkbook.Names.Add Name:="MyRange", _
    RefersTo:=Worksheets("Sheet1").Range("A1")
```

按名称删除已定义的命名范围

```
ThisWorkbook.Names("MyRange").Delete
```

通过名称访问命名范围

```
Dim rng As Range
Set rng = ThisWorkbook.Worksheets("Sheet1").Range("MyRange")
Call MsgBox("宽度 = " & rng.Value)
```

# Chapter 5: Named Ranges

Topic should include information specifically related to named ranges in Excel including methods for creating, modifying, deleting, and accessing defined named ranges.

## Section 5.1: Define A Named Range

Using named ranges allows you to describe the meaning of a cell(s) contents and use this defined name in place of an actual cell address.

For example, formula =A5\*B5 can be replaced with =Width\*Height to make the formula much easier to read and understand.

To define a new named range, select cell or cells to name and then type new name into the Name Box next to the formula bar.

	A1					
	A	B	C	D	E	F
1						
2						
3						
4						
5	15	20				
6						
7						

Note: Named Ranges default to global scope meaning that they can be accessed from anywhere within the workbook. Older versions of Excel allow for duplicate names so care must be taken to prevent duplicate names of global scope otherwise results will be unpredictable. Use Name Manager from Formulas tab to change scope.

## Section 5.2: Using Named Ranges in VBA

Create new named range called 'MyRange' assigned to cell A1

```
ThisWorkbook.Names.Add Name:="MyRange", _
    RefersTo:=Worksheets("Sheet1").Range("A1")
```

Delete defined named range by name

```
ThisWorkbook.Names("MyRange").Delete
```

Access Named Range by name

```
Dim rng As Range
Set rng = ThisWorkbook.Worksheets("Sheet1").Range("MyRange")
Call MsgBox("Width = " & rng.Value)
```

通过快捷方式访问命名范围

就像访问其他范围一样，命名范围也可以通过快捷方式直接访问，无需创建Range对象。上面代码片段中的三行代码可以用一行代码替代：

```
Call MsgBox("Width = " & [MyRange])
```

注意：Range 的默认属性是其值，因此[MyRange]与[MyRange].Value是相同的

你也可以对范围调用方法。以下代码选择了MyRange：

```
[MyRange].Select
```

注意：有一个警告是快捷表示法不能用于在VBA库中其他地方使用的单词。例如，名为Width的范围不能通过[Width]访问，但如果通过ThisWorkbook.Worksheets("Sheet1").Range("Width")访问，则可以正常使用

第5.3节：使用名称管理器管理命名范围

公式选项卡 > 已定义名称组 > 名称管理器按钮

名称管理器允许你：

- 1. 创建或更改名称
- 2. 创建或更改单元格引用
- 3. 创建或更改作用域
- 4. 删除现有命名范围

Access a Named Range with a Shortcut

Just like any other range, named ranges can be accessed directly with through a shortcut notation that does not require a Range object to be created. The three lines from the code excerpt above can be replaced by a single line:

```
Call MsgBox("Width = " & [MyRange])
```

Note: The default property for a Range is its Value, so [MyRange] is the same as [MyRange].Value

You can also call methods on the range. The following selects MyRange:

```
[MyRange].Select
```

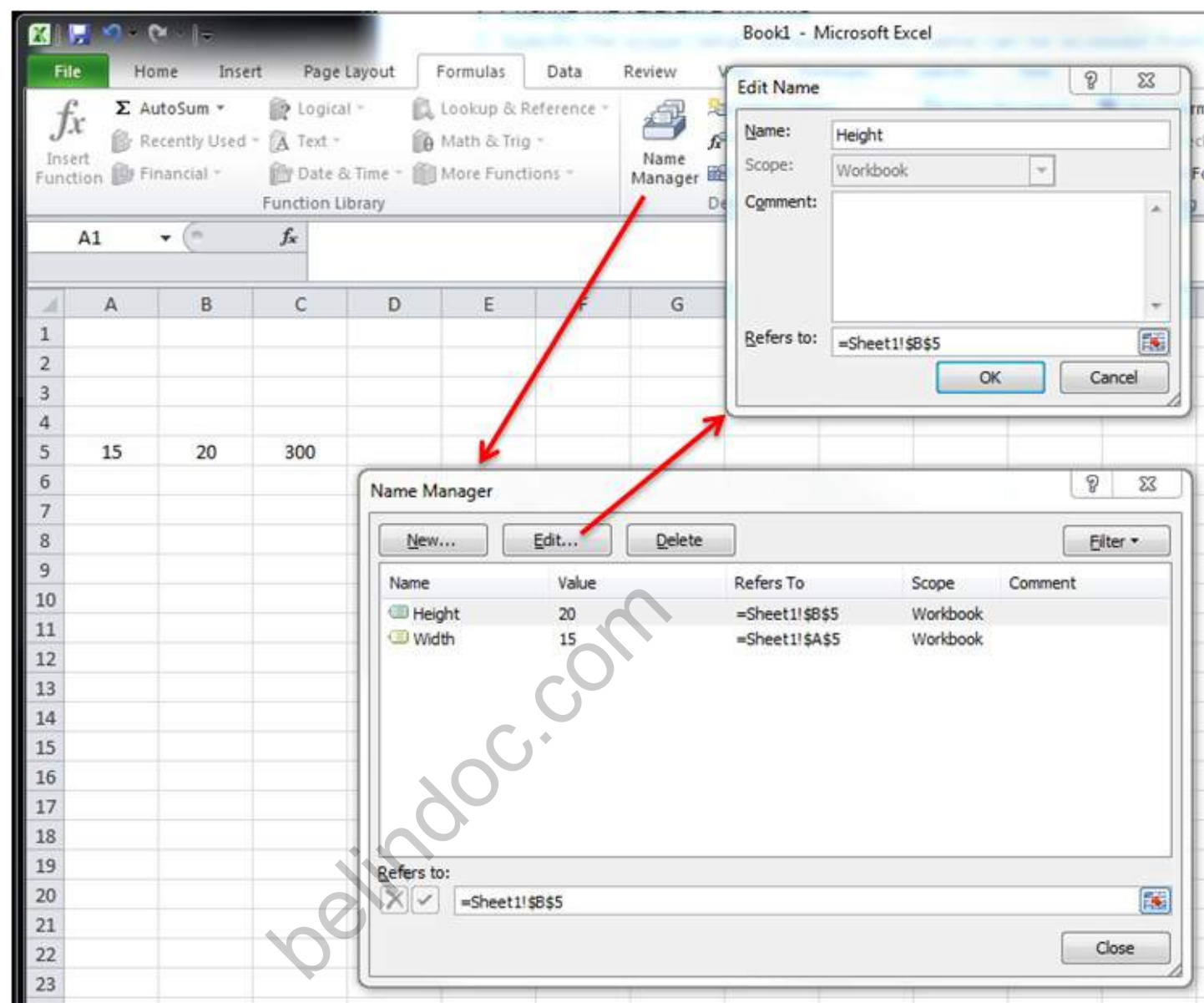
Note: One caveat is that the shortcut notation does not work with words that are used elsewhere in the VBA library. For example, a range named Width would not be accessible as [Width] but would work as expected if accessed through ThisWorkbook.Worksheets("Sheet1").Range("Width")

Section 5.3: Manage Named Range(s) using Name Manager

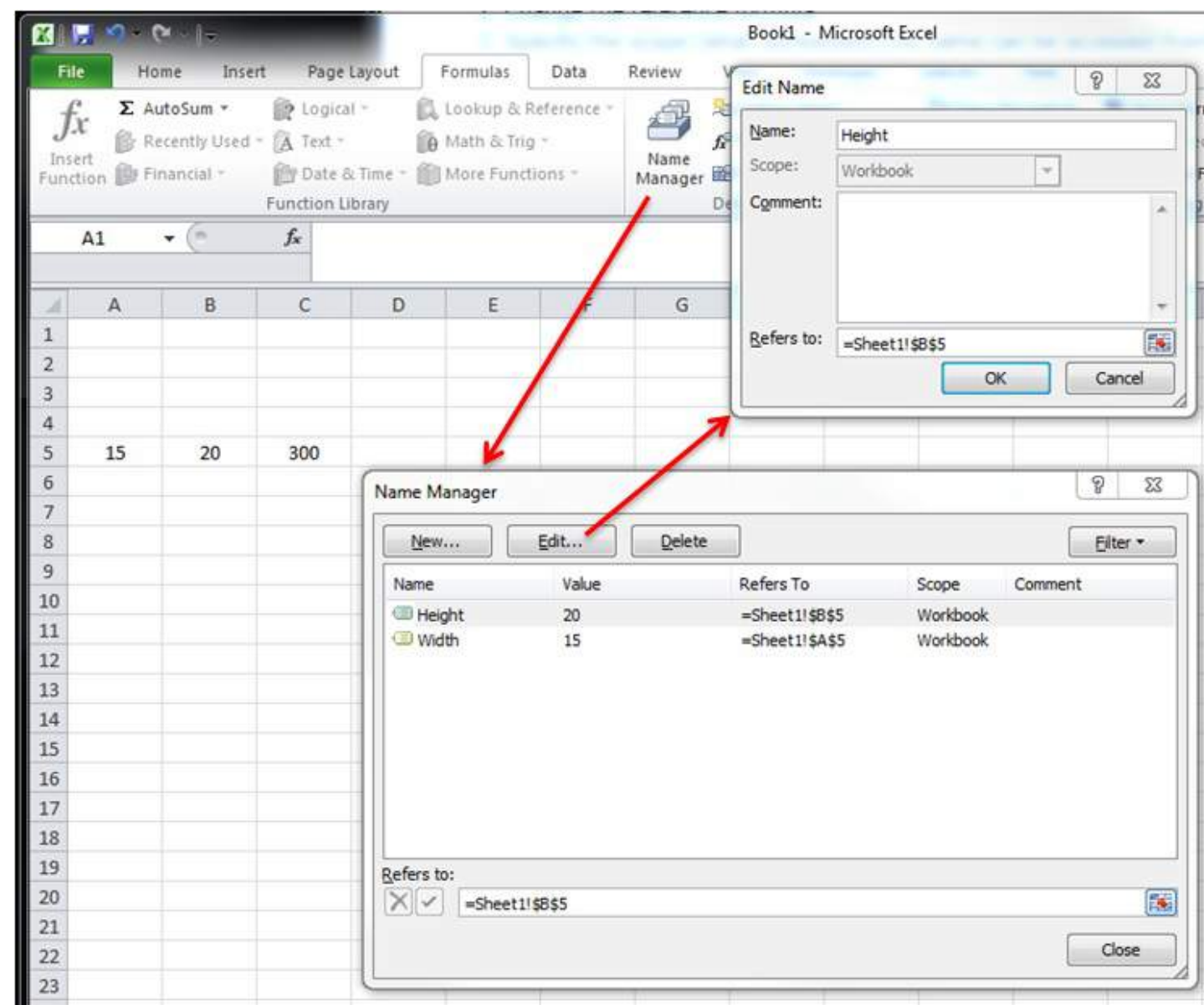
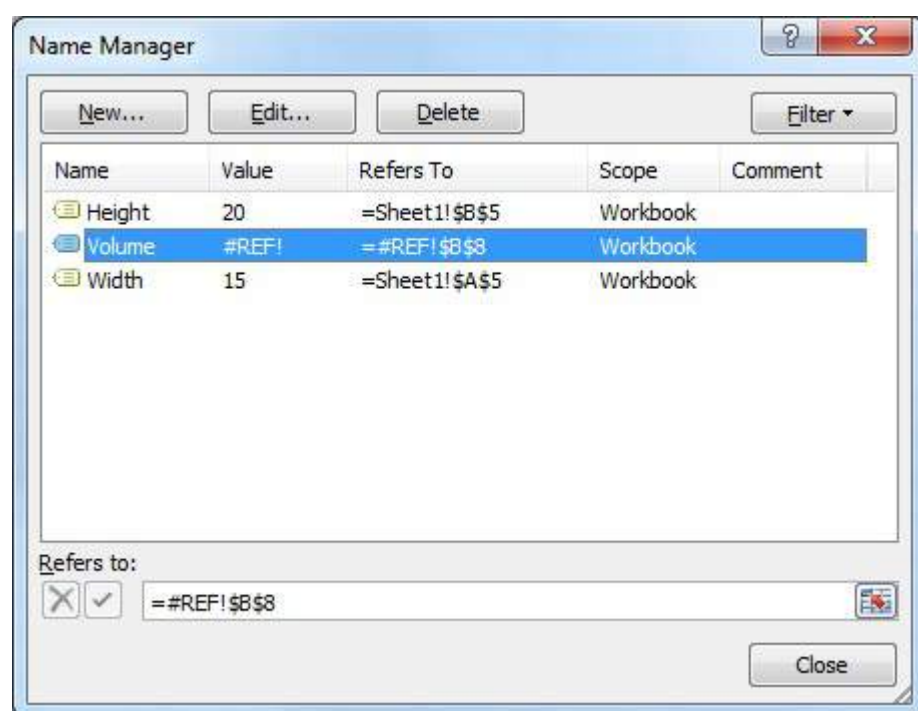
Formulas tab > Defined Names group > Name Manager button

Named Manager allows you to:

- 1. Create or change name
- 2. Create or change cell reference
- 3. Create or change scope
- 4. Delete existing named range



名称管理器提供了一个有用的快速查看断开链接的功能。

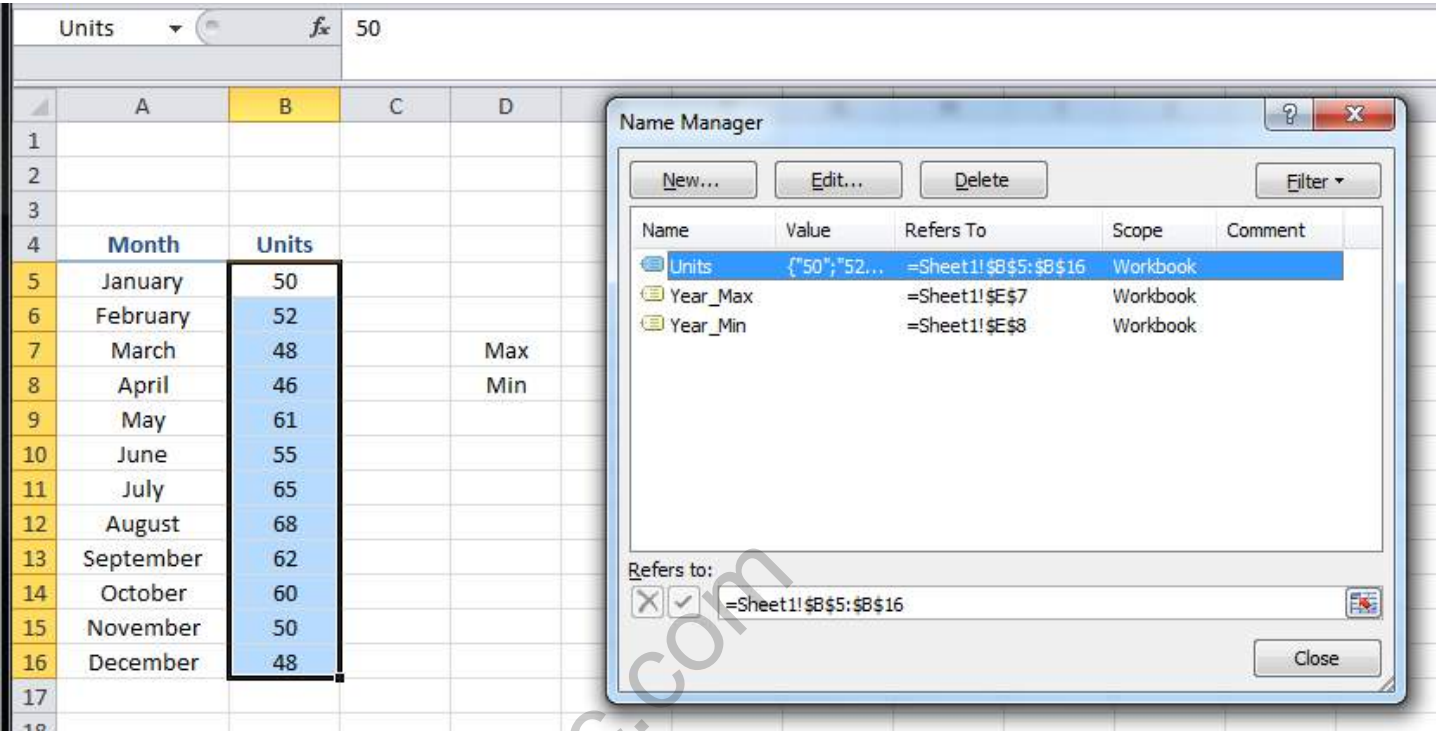


Named Manager provides a useful quick look for broken links.



第5.4节：命名范围数组

示例表



代码

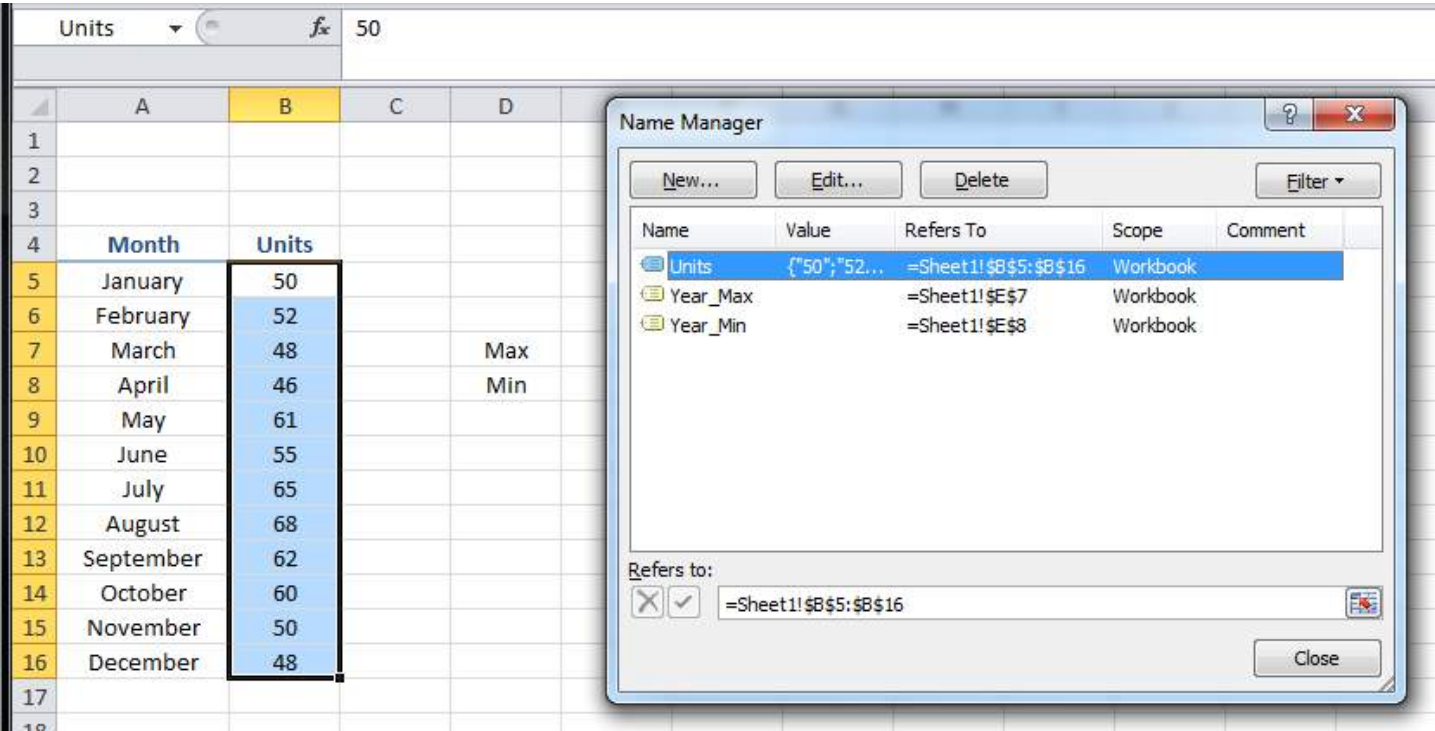
```
子程序 示例()  
    声明 wks 为 工作表  
    设置 wks = ThisWorkbook.Worksheets("Sheet1")  
  
    声明 units 为 范围  
    设置 units = ThisWorkbook.Names("Units").RefersToRange  
  
    Worksheets("Sheet1").Range("Year_Max").Value = WorksheetFunction.Max(units)  
    Worksheets("Sheet1").Range("Year_Min").Value = WorksheetFunction.Min(units)  
End Sub
```

结果

Month	Units			
January	50			
February	52			
March	48	Max		68
April	46	Min		46
May	61			
June	55			
July	65			
August	68			
September	62			
October	60			
November	50			
December	48			

Section 5.4: Named Range Arrays

Example sheet



Code

```
Sub Example()  
    Dim wks As Worksheet  
    Set wks = ThisWorkbook.Worksheets("Sheet1")  
  
    Dim units As Range  
    Set units = ThisWorkbook.Names("Units").RefersToRange  
  
    Worksheets("Sheet1").Range("Year_Max").Value = WorksheetFunction.Max(units)  
    Worksheets("Sheet1").Range("Year_Min").Value = WorksheetFunction.Min(units)  
End Sub
```

Result

Month	Units			
January	50			
February	52			
March	48	Max		68
April	46	Min		46
May	61			
June	55			
July	65			
August	68			
September	62			
October	60			
November	50			
December	48			



# 第6章：合并单元格/区域

## 第6.1节：使用合并单元格/区域前请三思

首先，合并单元格仅用于改善工作表的外观。

因此，只有在您的工作表和工作簿完全功能正常后，才应考虑使用合并单元格，这实际上是最后一步！

### 合并区域中的数据在哪里？

当您合并一个区域时，您只会显示一个块。

数据将位于该区域的第一个单元格，其他单元格将为空！

关于这一点的一个优点是：合并后无需填充所有单元格或整个区域，只需填写第一个单元格！；)

合并区域的其他方面总体上是负面的：

- 如果您使用查找最后一行或最后一列的方法，可能会出现错误如果您遍历行，
- 并且为了更好的可读性合并了一些区域，您将遇到空单元格，而不是合并区域显示的值

# Chapter 6: Merged Cells / Ranges

## Section 6.1: Think twice before using Merged Cells/Ranges

First of all, Merged Cells are there only to improve the look of your sheets.

So it is literally the last thing that you should do, once your sheet and workbook are totally functional!

### Where is the data in a Merged Range?

When you merge a Range, you'll only display one block.

The data will be in the very **first cell of that Range**, and the **others will be empty cells!**

One good point about it : no need to fill all the cells or the range once merged, just fill the first cell! ;)

The other aspects of this merged ranged are globally negative :

- If you use a method for finding last row or column, you'll risk some errors
- If you loop through rows and you have merged some ranges for a better readability, you'll encounter empty cells and not the value displayed by the merged range

# 第7章：定位区域中的重复值

在某些情况下，您需要评估一段数据并定位其中的重复值。对于较大的数据集，您可以采用多种方法，这些方法要么使用VBA代码，要么使用条件函数。此示例使用两个嵌套的for-next循环中的简单if-then条件，测试该区域内的每个单元格是否与区域内的任何其他单元格的值相等。

## 第7.1节：查找范围内的重复项

以下测试范围A2到A7中的重复值。 **备注：** 该示例展示了一种可能的解决方案，作为解决方案的初步方法。使用数组比使用范围更快，也可以使用集合、字典或XML方法来检查重复项。

```
Sub find_duplicates()  
' 声明变量  
Dim ws As Worksheet ' 工作表  
Dim cell As Range ' 工作表范围内的单元格  
Dim n As Integer ' 最大行号  
Dim bFound As Boolean ' 布尔标志，若找到重复项则为真  
Dim sFound As String: sFound = "|" ' 找到的重复项  
Dim s As String ' 消息字符串  
Dim s2 As String ' 部分消息字符串  
' 将工作表设置到内存  
Set ws = ThisWorkbook.Sheets("Duplications")  
  
' 遍历完全限定引用  
For Each cell In ws.Range("A2:A7")  
bFound = False: s2 = "" ' 每个单元格开始时赋空值  
' 检查该值是否首次作为重复出现，以避免进一步搜索  
如果 InStr(sFound, "|" & cell & "|") = 0 则  
  
    对于 n = cell.Row + 1 到 7 ' 迭代起点以避免重复搜索  
    如果 cell = ws.Range("A" & n).Value 则  
        如果 cell.Row <> n 则 ' 仅其他单元格，因为同一单元格不能是重复项  
            bFound = True ' 布尔标志  
            ' 在单元格 A{n} 中找到重复项  
s2 = s2 & vbNewLine & " -> A" & n & " 中的重复项"  
        End If  
    End If  
    下一步  
End If  
' 注意所有找到的重复项  
如果 bFound 则  
    ' 将值添加到所有找到的重复值列表中  
    ' (可以轻松拆分为数组以便进一步分析)  
sFound = sFound & cell & "|"  
s = s & cell.Address & " (值=" & cell & ")" & s2 & vbNewLine & vbNewLine  
End If  
    下一步  
' 带有最终结果的消息框  
MsgBox "重复值为 " & sFound & vbNewLine & vbNewLine & s, vbInformation, "找到的重复项"  
End Sub
```

根据您的需求，示例可以进行修改——例如，n 的上限可以是范围内最后一个有数据单元格的行号，或者在 True If 条件下的操作可以编辑为提取重复值到其他位置。

# Chapter 7: Locating duplicate values in a range

At certain points, you will be evaluating a range of data and you will need to locate the duplicates in it. For bigger data sets, there are a number of approaches you can take that use either VBA code or conditional functions. This example uses a simple if-then condition within two nested for-next loops to test whether each cell in the range is equal in value to any other cell in the range.

## Section 7.1: Find duplicates in a range

The following tests range A2 to A7 for duplicate values. **Remark:** This example illustrates a possible solution as a first approach to a solution. It's faster to use an array than a range and one could use collections or dictionaries or xml methods to check for duplicates.

```
Sub find_duplicates()  
' Declare variables  
Dim ws As Worksheet ' worksheet  
Dim cell As Range ' cell within worksheet range  
Dim n As Integer ' highest row number  
Dim bFound As Boolean ' boolean flag, if duplicate is found  
Dim sFound As String: sFound = "|" ' found duplicates  
Dim s As String ' message string  
Dim s2 As String ' partial message string  
' Set Sheet to memory  
Set ws = ThisWorkbook.Sheets("Duplications")  
  
' loop thru FULLY QUALIFIED REFERENCE  
For Each cell In ws.Range("A2:A7")  
bFound = False: s2 = "" ' start each cell with empty values  
' Check if first occurrence of this value as duplicate to avoid further searches  
If InStr(sFound, "|" & cell & "|") = 0 Then  
  
    For n = cell.Row + 1 To 7 ' iterate starting point to avoid REDUNDANT SEARCH  
    If cell = ws.Range("A" & n).Value Then  
        If cell.Row <> n Then ' only other cells, as same cell cannot be a duplicate  
            bFound = True ' boolean flag  
            ' found duplicates in cell A{n}  
s2 = s2 & vbNewLine & " -> duplicate in A" & n  
        End If  
    End If  
    Next  
End If  
' notice all found duplicates  
If bFound Then  
    ' add value to list of all found duplicate values  
    ' (could be easily split to an array for further analyze)  
sFound = sFound & cell & "|"  
s = s & cell.Address & " (value=" & cell & ")" & s2 & vbNewLine & vbNewLine  
End If  
Next  
' MessageBox with final result  
MsgBox "Duplicate values are " & sFound & vbNewLine & vbNewLine & s, vbInformation, "Found duplicates"  
End Sub
```

Depending on your needs, the example can be modified - for instance, the upper limit of n can be the row value of last cell with data in the range, or the action in case of a True If condition can be edited to extract the duplicate

值到其他位置。然而，例程的机制不会改变。

belindoc.com

value somewhere else. However, the mechanics of the routine would not change.

# 第8章：用户自定义函数（UDF）

## 第8.1节：允许完整列引用且不受惩罚

如果可以将完整列引用作为参数传入，在工作表上实现某些 UDF 会更容易。但是，由于编码的显式特性，任何涉及这些范围的循环可能会处理成千上万个完全为空的单元格。这会导致您的 VBA 项目（和工作簿）冻结，同时处理不必要的无效值。

遍历工作表的单元格是完成任务最慢的方法之一，但有时是不可避免的。将工作量缩减到实际所需的范围是完全合理的。

解决方案是使用 Intersect 方法将完整列或完整行引用截断为Worksheet.UsedRange 属性。以下示例将大致复制工作表的原生 SUMIF 函数，因此criteria\_range也将调整大小以适应 sum\_range，因为 sum\_range中的每个值都必须对应criteria\_range中的一个值。

用于工作表上的 UDF 的Application.Caller是其所在的单元格。该单元格的.Parent属性是工作表。这将于定义.UsedRange。

在模块代码表中：

```
Option Explicit

Function udfMySumIf(rngA As Range, rngB As Range, _
    Optional crit As Variant = "yes")
    Dim c As Long, ttl As Double

    With Application.Caller.Parent
        Set rngA = Intersect(rngA, .UsedRange)
        Set rngB = rngB.Resize(rngA.Rows.Count, rngA.Columns.Count)
    End With

    For c = 1 To rngA.Cells.Count
        If IsNumeric(rngA.Cells(c).Value2) Then
            If LCase(rngB(c).Value2) = LCase(crit) Then
                ttl = ttl + rngA.Cells(c).Value2
            End If
        End If
    Next c

    udfMySumIf = ttl
End Function
```

语法：  
=udfMySumIf(\*sum\_range\*, \*criteria\_range\*, [\*criteria\*])

# Chapter 8: User Defined Functions (UDFs)

## Section 8.1: Allow full column references without penalty

It's easier to implement some UDFs on the worksheet if full column references can be passed in as parameters. However, due to the explicit nature of coding, any loop involving these ranges may be processing hundreds of thousands of cells that are completely empty. This reduces your VBA project (and workbook) to a frozen mess while unnecessary non-values are processed.

Looping through a worksheet's cells is one of the slowest methods of accomplishing a task but sometimes it is unavoidable. Cutting the work performed down to what is actually required makes perfect sense.

The solution is to truncate the full column or full row references to the [Worksheet.UsedRange property](#) with the [Intersect method](#). The following sample will loosely replicate a worksheet's native SUMIF function so the *criteria\_range* will also be resized to suit the *sum\_range* since each value in the *sum\_range* must be accompanied by a value in the *criteria\_range*.

The [Application.Caller](#) for a UDF used on a worksheet is the cell in which it resides. The cell's [.Parent](#) property is the worksheet. This will be used to define the .UsedRange.

In a Module code sheet:

```
Option Explicit

Function udfMySumIf(rngA As Range, rngB As Range, _
    Optional crit As Variant = "yes")
    Dim c As Long, ttl As Double

    With Application.Caller.Parent
        Set rngA = Intersect(rngA, .UsedRange)
        Set rngB = rngB.Resize(rngA.Rows.Count, rngA.Columns.Count)
    End With

    For c = 1 To rngA.Cells.Count
        If IsNumeric(rngA.Cells(c).Value2) Then
            If LCase(rngB(c).Value2) = LCase(crit) Then
                ttl = ttl + rngA.Cells(c).Value2
            End If
        End If
    Next c

    udfMySumIf = ttl
End Function
```

Syntax:  
=udfMySumIf(\*sum\_range\*, \*criteria\_range\*, [\*criteria\*])



	A	B	C	D	E	F	G
1	numbers	include					
2		17 Yes					
3	L	Maybe			68		
4		17 Maybe					
5		15 Yes					
6		8 Maybe					
7	Y	No					
8		5 No					
9		18 Yes					
10	L	Maybe					
11	A	Yes					
12	J	Maybe					
13		18 Yes					
14		7 No					
15		16 Maybe					
16							
17							

虽然这是一个相当简单的示例，但它充分演示了传入两个完整的列引用（每列1,048,576行），但仅处理15行数据和条件。

微软™提供的官方MSDN文档链接，涵盖各个方法和属性。

### 第8.2节：统计范围内唯一值的数量

```
函数 countUnique(r 作为 范围) 作为 长整型
'Application.Volatile False ' 可选
设置 r = Intersect(r, r.Worksheet.UsedRange) ' 如果传入整个行或列给函数，则可选

声明 c 作为 新 集合, v
出错时继续 ' 忽略运行时错误457：“此键已与集合中的元素关联”。

遍历 v 在 r.Value ' 对于包含多个区域的范围，去掉.Value
c.Add 0, v & ""
下一步
c.Remove "" ' 可选，排除空值计数
countUnique = c.Count
End Function
```

收藏

### 第8.3节：UDF - 你好，世界

- 1. 打开Excel
- 2. 打开Visual Basic编辑器（参见打开Visual Basic编辑器）
- 3. 通过点击 插入 --> 模块 添加一个新模块：

	A	B	C	D	E	F	G
1	numbers	include					
2		17 Yes					
3	L	Maybe			68		
4		17 Maybe					
5		15 Yes					
6		8 Maybe					
7	Y	No					
8		5 No					
9		18 Yes					
10	L	Maybe					
11	A	Yes					
12	J	Maybe					
13		18 Yes					
14		7 No					
15		16 Maybe					
16							
17							

While this is a fairly simplistic example, it adequately demonstrates passing in two full column references (1,048,576 rows each) but only processing 15 rows of data and criteria.

Linked official MSDN documentation of individual methods and properties courtesy of Microsoft™.

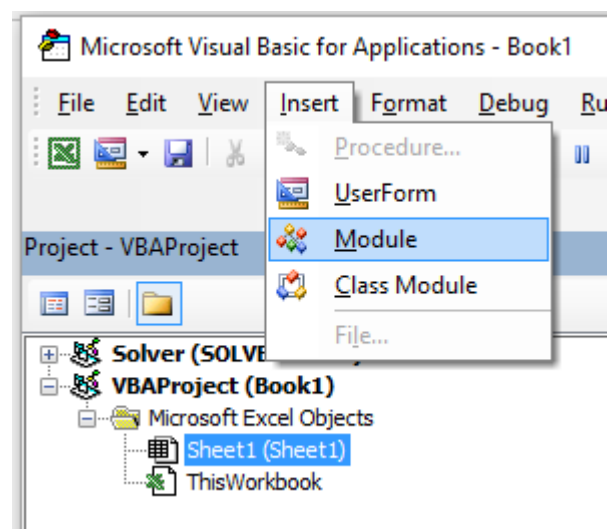
### Section 8.2: Count Unique values in Range

```
Function countUnique(r As range) As Long
'Application.Volatile False ' optional
Set r = Intersect(r, r.Worksheet.UsedRange) ' optional if you pass entire rows or columns to the function
Dim c As New Collection, v
On Error Resume Next ' to ignore the Run-time error 457: "This key is already associated with an element of this collection".
For Each v In r.Value ' remove .Value for ranges with more than one Areas
c.Add 0, v & ""
Next
c.Remove "" ' optional to exclude blank values from the count
countUnique = c.Count
End Function
```

Collections

### Section 8.3: UDF - Hello World

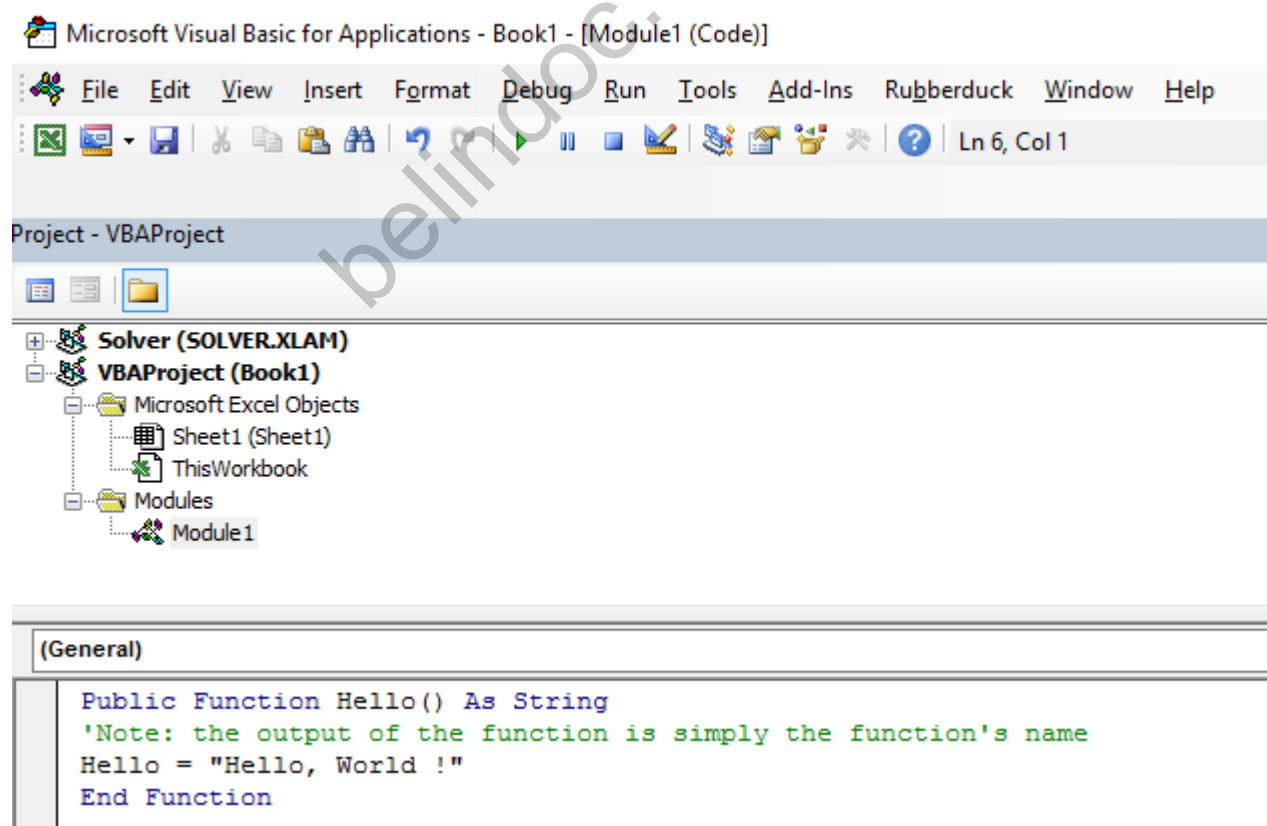
- 1. Open Excel
- 2. Open the Visual Basic Editor ( see Opening the Visual Basic Editor )
- 3. Add a new module by clicking Insert --> Module :



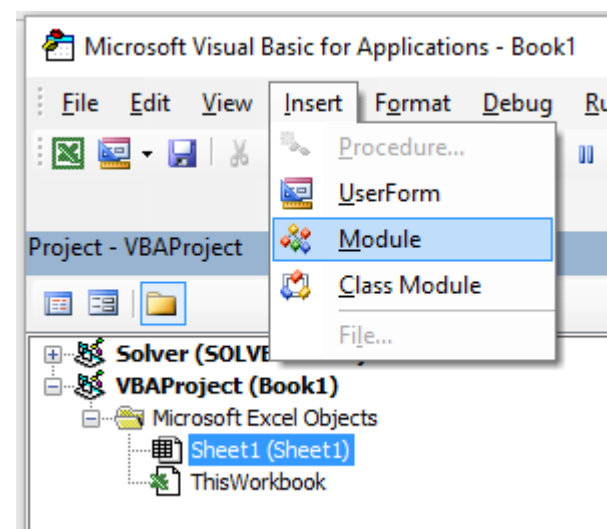
4. 在新模块中复制并粘贴以下代码：

```
Public Function Hello() As String
'注意：该函数的输出仅为函数名
Hello = "Hello, World !"
End Function
```

要获得：



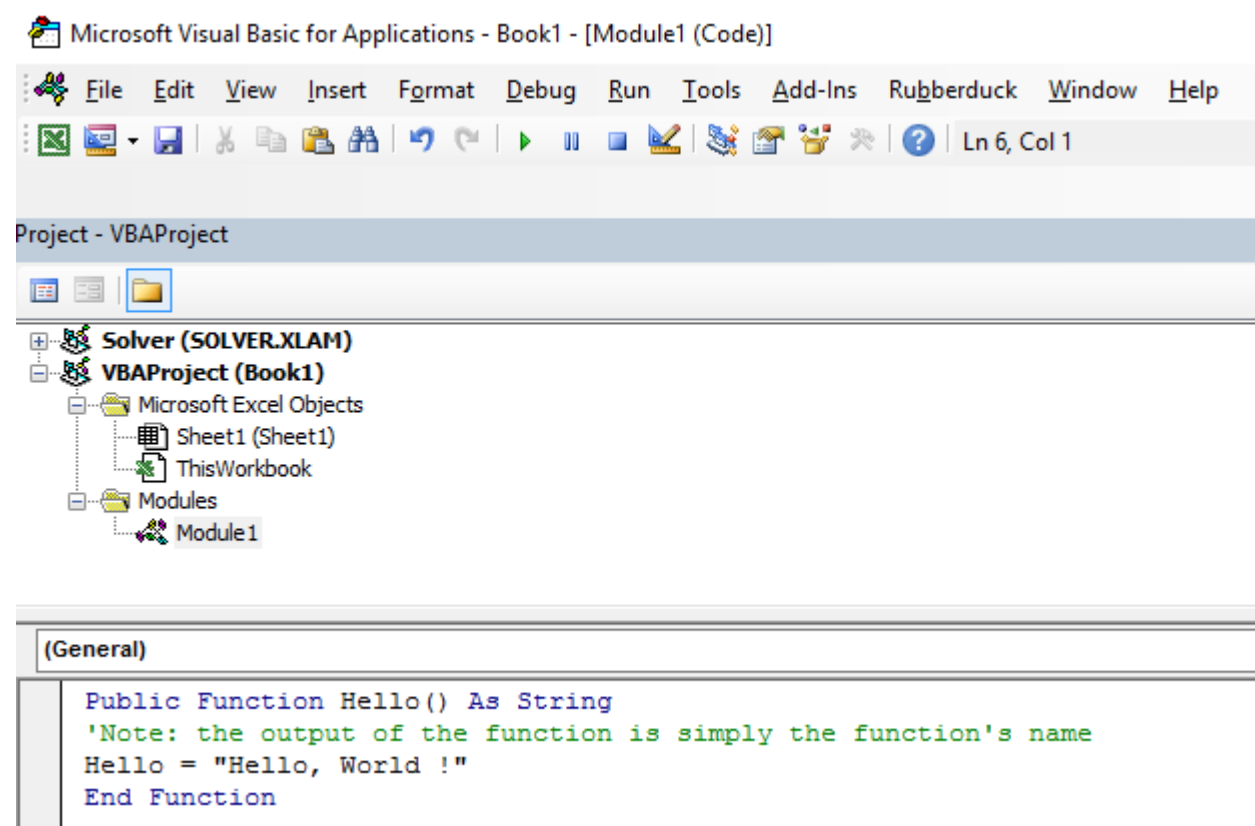
5. 返回工作簿，在单元格中输入“=Hello()”以显示“Hello World”。



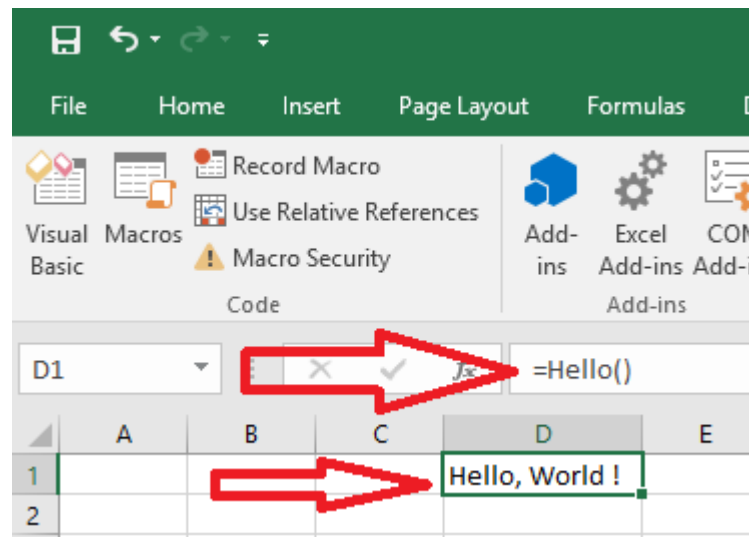
4. Copy and Paste the following code in the new module :

```
Public Function Hello() As String
'Note: the output of the function is simply the function's name
Hello = "Hello, World !"
End Function
```

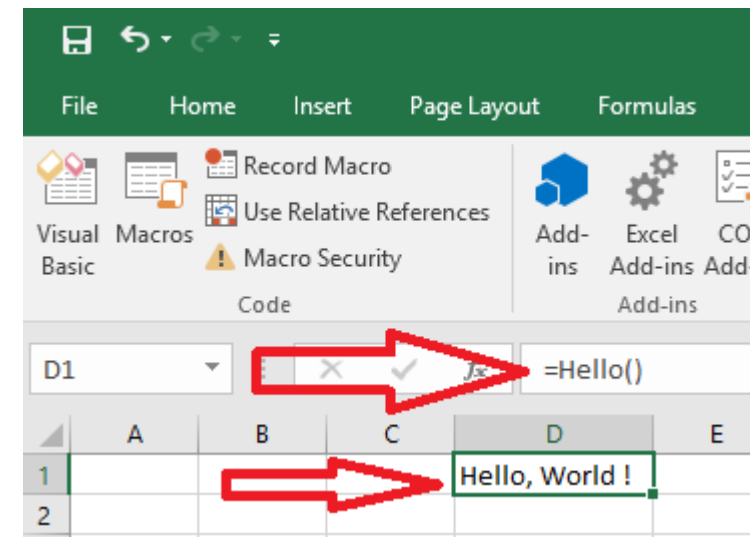
To obtain :



5. Go back to your workbook and type "=Hello()" into a cell to see the "Hello World".



belindoc.com



# 第9章：使用VBA的条件格式

## 第9.1节：FormatConditions.Add

语法：  
FormatConditions.Add(Type, Operator, Formula1, Formula2)

参数：

名称	必需 / 可选	数据类型
类型	必需	XlFormatConditionType
Operator	可选	Variant
Formula1	可选	Variant
Formula2	可选	Variant

**XlFormatConditionType 枚举：**

名称	描述
xlAboveAverageCondition	高于平均条件
xlBlanksCondition	空白条件
xlCellValue	单元格数值
xlColorScale	颜色刻度
xlDatabar	数据条
xlErrorsCondition	错误条件
xlExpression	表达式
xlIconSet	图标集
xlNoBlanksCondition	无空白条件
xlNoErrorsCondition	无错误条件
xlTextString	文本字符串
xlTimePeriod	时间段
xlTop10	前10个值
xlUniqueValues	唯一值

按单元格值格式化：

```
With Range("A1").FormatConditions.Add(xlCellValue, xlGreater, ">=100")
    With .Font
        .Bold = True
        .ColorIndex = 3
    End With
End With
```

运算符：

名称
xlBetween
xlEqual
xlGreater
xlGreaterEqual
xlLess
xlLessEqual
xlNotBetween
xlNotEqual

# Chapter 9: Conditional formatting using VBA

## Section 9.1: FormatConditions.Add

Syntax:  
FormatConditions.Add(Type, Operator, Formula1, Formula2)

Parameters:

Name	Required / Optional	Data Type
Type	Required	XlFormatConditionType
Operator	Optional	Variant
Formula1	Optional	Variant
Formula2	Optional	Variant

**XlFormatConditionType enumeration:**

Name	Description
xlAboveAverageCondition	Above average condition
xlBlanksCondition	Blanks condition
xlCellValue	Cell value
xlColorScale	Color scale
xlDatabar	Databar
xlErrorsCondition	Errors condition
xlExpression	Expression
xlIconSet	Icon set
xlNoBlanksCondition	No blanks condition
xlNoErrorsCondition	No errors condition
xlTextString	Text string
xlTimePeriod	Time period
xlTop10	Top 10 values
xlUniqueValues	Unique values

Formatting by cell value:

```
With Range("A1").FormatConditions.Add(xlCellValue, xlGreater, ">=100")
    With .Font
        .Bold = True
        .ColorIndex = 3
    End With
End With
```

Operators:

Name
xlBetween
xlEqual
xlGreater
xlGreaterEqual
xlLess
xlLessEqual
xlNotBetween
xlNotEqual



如果类型是 xlExpression, 则忽略 Operator 参数。

按文本包含进行格式设置：

```
使用 Range("a1:a10").FormatConditions.Add(xlTextString, TextOperator:=xlContains, String:="egg")
    With .Font
        .Bold = True
        .ColorIndex = 3
    End With
End With
```

运算符：

名称	描述
xlBeginsWith	以指定值开头。
xlContains	包含指定的值。
xlDoesNotContain	不包含指定的值。
xlEndsWith	以指定的值结尾

按时间段格式化

```
使用 Range("a1:a10").FormatConditions.Add(xlTimePeriod, DateOperator:=xlToday)
    With .Font
        .Bold = True
        .ColorIndex = 3
    End With
End With
```

运算符：

名称
xlYesterday
xlTomorrow
xlLast7Days
xlLastWeek
xlThisWeek
xlNextWeek
xlLastMonth
xlThisMonth
xlNextMonth

## 第9.2节：删除条件格式

删除范围内的所有条件格式：

```
Range("A1:A10").FormatConditions.Delete
```

删除工作表中的所有条件格式：

```
Cells.FormatConditions.Delete
```

## 第9.3节：FormatConditions.AddUniqueValues

突出显示重复值

```
使用 Range("E1:E100").FormatConditions.AddUniqueValues
    .DupeUnique = xlDuplicate
使用 .Font
    .Bold = True
```

If Type is xlExpression, the Operator argument is ignored.

Formatting by text contains:

```
With Range("a1:a10").FormatConditions.Add(xlTextString, TextOperator:=xlContains, String:="egg")
    With .Font
        .Bold = True
        .ColorIndex = 3
    End With
End With
```

Operators:

Name	Description
xlBeginsWith	Begins with a specified value.
xlContains	Contains a specified value.
xlDoesNotContain	Does not contain the specified value.
xlEndsWith	Endswith the specified value

Formatting by time period

```
With Range("a1:a10").FormatConditions.Add(xlTimePeriod, DateOperator:=xlToday)
    With .Font
        .Bold = True
        .ColorIndex = 3
    End With
End With
```

Operators:

Name
xlYesterday
xlTomorrow
xlLast7Days
xlLastWeek
xlThisWeek
xlNextWeek
xlLastMonth
xlThisMonth
xlNextMonth

## Section 9.2: Remove conditional format

Remove all conditional format in range:

```
Range("A1:A10").FormatConditions.Delete
```

Remove all conditional format in worksheet:

```
Cells.FormatConditions.Delete
```

## Section 9.3: FormatConditions.AddUniqueValues

Highlighting Duplicate Values

```
With Range("E1:E100").FormatConditions.AddUniqueValues
    .DupeUnique = xlDuplicate
    With .Font
        .Bold = True
    End With
End With
```

```
        .ColorIndex = 3
    结束 With
结束 With
```

突出显示唯一值

```
使用 Range("E1:E100").FormatConditions.AddUniqueValues
    使用 .Font
        .Bold = True
        .ColorIndex = 3
    结束 With
结束 With
```

第9.4节：FormatConditions.AddTop10

突出显示前5个值

```
使用 Range("E1:E100").FormatConditions.AddTop10
    .TopBottom = xlTop10Top
    .Rank = 5
    .Percent = False
    使用 .Font
        .Bold = True
        .ColorIndex = 3
    结束 With
结束 With
```

第9.5节：FormatConditions.AddAboveAverage

```
使用 Range("E1:E100").FormatConditions.AddAboveAverage
    .AboveBelow = xlAboveAverage
    With .Font
        .Bold = True
        .ColorIndex = 3
    结束 With
结束 With
```

运算符：

名称	描述
XIAboveAverage	高于平均值
XIAboveStdDev	高于标准差
XIBelowAverage	低于平均值
XIBelowStdDev	低于标准差
XIEqualAboveAverage	等于或高于平均值
XIEqualBelowAverage	等于或低于平均值

第9.6节：FormatConditions.AddIconSetCondition

```
        .ColorIndex = 3
    End With
End With
```

Highlighting Unique Values

```
With Range("E1:E100").FormatConditions.AddUniqueValues
    With .Font
        .Bold = True
        .ColorIndex = 3
    End With
End With
```

Section 9.4: FormatConditions.AddTop10

Highlighting Top 5 Values

```
With Range("E1:E100").FormatConditions.AddTop10
    .TopBottom = xlTop10Top
    .Rank = 5
    .Percent = False
    With .Font
        .Bold = True
        .ColorIndex = 3
    End With
End With
```

Section 9.5: FormatConditions.AddAboveAverage

```
With Range("E1:E100").FormatConditions.AddAboveAverage
    .AboveBelow = xlAboveAverage
    With .Font
        .Bold = True
        .ColorIndex = 3
    End With
End With
```

Operators:

Name	Description
XIAboveAverage	Above average
XIAboveStdDev	Above standard deviation
XIBelowAverage	Below average
XIBelowStdDev	Below standard deviation
XIEqualAboveAverage	Equal above average
XIEqualBelowAverage	Equal below average

Section 9.6: FormatConditions.AddIconSetCondition

	A
1	13
2	22
3	33
4	30
5	23
6	40
7	50
8	4
9	20
10	13
11	5
12	45
13	30
14	37
15	12

```
Range("a1:a10").FormatConditions.AddIconSetCondition
With Selection.FormatConditions(1)
    .ReverseOrder = False
    .ShowIconOnly = False
    .IconSet = ActiveWorkbook.IconSets(xl3Arrows)
End With

With Selection.FormatConditions(1).IconCriteria(2)
    .Type = xlConditionValuePercent
    .Value = 33
    .Operator = 7
End With

With Selection.FormatConditions(1).IconCriteria(3)
    .Type = xlConditionValuePercent
    .Value = 67
    .Operator = 7
End With
```

- 图标集：
- 名称
- xl3Arrows
  - xl3ArrowsGray
  - xl3Flags
  - xl3Signs
  - xl3Stars
  - xl3Symbols
  - xl3Symbols2
  - xl3TrafficLights1
  - xl3TrafficLights2
  - xl3Triangles
  - xl4Arrows
  - xl4ArrowsGray
  - xl4CRV
  - xl4RedToBlack
  - xl4TrafficLights
  - xl5Arrows

	A
1	13
2	22
3	33
4	30
5	23
6	40
7	50
8	4
9	20
10	13
11	5
12	45
13	30
14	37
15	12

```
Range("a1:a10").FormatConditions.AddIconSetCondition
With Selection.FormatConditions(1)
    .ReverseOrder = False
    .ShowIconOnly = False
    .IconSet = ActiveWorkbook.IconSets(xl3Arrows)
End With

With Selection.FormatConditions(1).IconCriteria(2)
    .Type = xlConditionValuePercent
    .Value = 33
    .Operator = 7
End With

With Selection.FormatConditions(1).IconCriteria(3)
    .Type = xlConditionValuePercent
    .Value = 67
    .Operator = 7
End With
```

- IconSet:
- Name
- xl3Arrows
  - xl3ArrowsGray
  - xl3Flags
  - xl3Signs
  - xl3Stars
  - xl3Symbols
  - xl3Symbols2
  - xl3TrafficLights1
  - xl3TrafficLights2
  - xl3Triangles
  - xl4Arrows
  - xl4ArrowsGray
  - xl4CRV
  - xl4RedToBlack
  - xl4TrafficLights
  - xl5Arrows

xl5ArrowsGray  
xl5Boxes  
xl5CRV  
xl5Quarters



类型：  
名称  
xlConditionValuePercent  
xlConditionValueNumber  
xlConditionValuePercentile  
xlConditionValueFormula

运算符：  
名称 值  
xlGreater 5  
xlGreaterEqual 7

数值：  
  
返回或设置条件格式中图标的阈值。

xl5ArrowsGray  
xl5Boxes  
xl5CRV  
xl5Quarters



Type:  
Name  
xlConditionValuePercent  
xlConditionValueNumber  
xlConditionValuePercentile  
xlConditionValueFormula

Operator:  
Name Value  
xlGreater 5  
xlGreaterEqual 7

Value:  
  
Returns or sets the threshold value for an icon in a conditional format.



# 第10章：工作簿

## 第10.1节：何时使用ActiveWorkbook和ThisWorkbook

VBA的最佳实践是始终明确指定VBA代码所引用的工作簿。如果省略此指定，VBA会假定代码指向当前活动的工作簿（ActiveWorkbook）。

```
'--- 当前活动的工作簿（和工作表）是默认的'  
Range("A1").value = 3.1415  
Cells(1, 1).value = 3.1415
```

然而，当同时打开多个工作簿时——尤其是当VBA代码从Excel加载项中运行时——对ActiveWorkbook的引用可能会混淆或指向错误。例如，一个带有用户自定义函数（UDF）的加载项，该函数检查当前时间并将其与存储在加载项某个工作表上的值进行比较（这些工作表通常对用户不可见），就必须明确指定引用的是哪个工作簿。在我们的示例中，我们打开（且处于活动状态）的工作簿在单元格A1中有一个公式=EarlyOrLate()，且该活动工作簿没有任何VBA代码。在我们的加载项中，我们有以下用户自定义函数（UDF）：

```
Public Function EarlyOrLate() As String  
    If Hour(Now) > ThisWorkbook.Sheets("WatchTime").Range("A1") Then  
        EarlyOrLate = "现在很晚了！"  
    Else  
        EarlyOrLate = "现在还早！"  
    End If  
End Function
```

UDF 的代码编写并存储在已安装的 Excel 加载项中。它使用存储在加载项中名为“WatchTime”的工作表上的数据。如果 UDF 使用的是ActiveWorkbook而不是ThisWorkbook，那么它将无法保证所指的工作簿是哪一个。

## 第10.2节：更改新工作簿中默认的工作表数量

新 Excel 工作簿中创建的“出厂默认”工作表数量通常设置为三个。您的 VBA 代码可以显式设置新工作簿中的工作表数量。

```
'--- 保存当前 Excel 全局设置  
With Application  
    Dim oldSheetsCount As Integer  
    oldSheetsCount = .SheetsInNewWorkbook  
    Dim myNewWB As Workbook  
    .SheetsInNewWorkbook = 1  
    Set myNewWB = .Workbooks.Add  
    '--- 恢复之前的设置  
    .SheetsInNewWorkbook = oldSheetsCount  
End With
```

## 第10.3节：应用程序工作簿

在许多 Excel 应用程序中，VBA 代码针对其所在的工作簿执行操作。您将该工作簿保存为“.xlsm”扩展名，VBA 宏仅关注其中的工作表和数据。然而，您经常需要从其他工作簿合并或汇总数据，或者将部分数据写入单独的工作簿。打开、关闭、保存、创建和删除其他工作簿是许多VBA应用程序的常见需求。

# Chapter 10: Workbooks

## Section 10.1: When To Use ActiveWorkbook and ThisWorkbook

It's a VBA Best Practice to always specify which workbook your VBA code refers. If this specification is omitted, then VBA assumes the code is directed at the currently active workbook (ActiveWorkbook).

```
'--- the currently active workbook (and worksheet) is implied  
Range("A1").value = 3.1415  
Cells(1, 1).value = 3.1415
```

However, when several workbooks are open at the same time -- particularly and especially when VBA code is running from an Excel Add-In -- references to the ActiveWorkbook may be confused or misdirected. For example, an add-in with a UDF that checks the time of day and compares it to a value stored on one of the add-in's worksheets (that are typically not readily visible to the user) will have to explicitly identify which workbook is being referenced. In our example, our open (and active) workbook has a formula in cell A1 =EarlyOrLate() and does NOT have any VBA written for that active workbook. In our add-in, we have the following User Defined Function (UDF):

```
Public Function EarlyOrLate() As String  
    If Hour(Now) > ThisWorkbook.Sheets("WatchTime").Range("A1") Then  
        EarlyOrLate = "It's Late!"  
    Else  
        EarlyOrLate = "It's Early!"  
    End If  
End Function
```

The code for the UDF is written and stored in the installed Excel add-in. It uses data stored on a worksheet in the add-in called "WatchTime". If the UDF had used ActiveWorkbook instead of ThisWorkbook, then it would never be able to guarantee which workbook was intended.

## Section 10.2: Changing The Default Number of Worksheets In A New Workbook

The "factory default" number of worksheets created in a new Excel workbook is generally set to three. Your VBA code can explicitly set the number of worksheets in a new workbook.

```
'--- save the current Excel global setting  
With Application  
    Dim oldSheetsCount As Integer  
    oldSheetsCount = .SheetsInNewWorkbook  
    Dim myNewWB As Workbook  
    .SheetsInNewWorkbook = 1  
    Set myNewWB = .Workbooks.Add  
    '--- restore the previous setting  
    .SheetsInNewWorkbook = oldsheetcount  
End With
```

## Section 10.3: Application Workbooks

In many Excel applications, the VBA code takes actions directed at the workbook in which it's contained. You save that workbook with a ".xlsm" extension and the VBA macros only focus on the worksheets and data within. However, there are often times when you need to combine or merge data from other workbooks, or write some of your data to a separate workbook. Opening, closing, saving, creating, and deleting other workbooks is a common need for many VBA applications.

在任何时候，在VBA编辑器中，你都可以通过Application对象的Workbooks属性查看并访问该Excel实例当前打开的所有工作簿。MSDN文档对此进行了带有参考的说明。

## 第10.4节：打开一个（新的）工作簿，即使它已经打开

如果你想访问一个已经打开的工作簿，那么从Workbooks集合中获取赋值是直接的：

```
dim myWB as Workbook
Set myWB = Workbooks("UsuallyFullPathnameOfWorkbook.xlsx")
```

如果你想创建一个新的工作簿，则使用Workbooks集合对象的Add方法添加一个新条目。

```
Dim myNewWB as Workbook
Set myNewWB = Workbooks.Add
```

有时你可能不确定（或不关心）所需的工作簿是否已经打开，或者该工作簿可能根本不存在。下面的示例函数展示了如何始终返回一个有效的工作簿对象。

```
Option Explicit
Function GetWorkbook(ByVal wbFilename As String) As Workbook
    '--- 返回给定文件名的工作簿对象，包括检查
    '    当工作簿已经打开、存在但未打开，或
    '    尚不存在（必须创建）时的情况
    '    *** wbFilename 必须是完整的路径名

    Dim folderFile As String
    Dim returnedWB As Workbook

    '--- 检查文件是否存在于目录位置
    folderFile = File(wbFilename)
    If folderFile = "" Then
        '--- 工作簿不存在，因此创建它
        Dim pos1 As Integer
        Dim fileExt As String
        Dim fileFormatNum As Long
        '--- 为了正确保存工作簿，我们需要根据文件扩展名推断用户意图的工作簿类型

        pos1 = InStrRev(sFullName, ".", , vbTextCompare)
        fileExt = Right(sFullName, Len(sFullName) - pos1)
        Select Case fileExt
            Case "xlsx"
                fileFormatNum = 51
            Case "xlsm"
                fileFormatNum = 52
            Case "xls"
                fileFormatNum = 56
            Case "xlsb"
                fileFormatNum = 50
            Case Else
                Err.Raise vbObjectError + 1000, "GetWorkbook function", _
                    "您请求的文件类型（文件扩展名）无法识别。" & _
                    "请使用已知的扩展名：xlsx、xlsm、xls 或 xlsb。"
        End Select
        Set returnedWB = Workbooks.Add
        Application.DisplayAlerts = False
        returnedWB.SaveAs filename:=wbFilename, FileFormat:=fileFormatNum
```

At any time in the VBA Editor, you can view and access any and all workbooks currently open by that instance of Excel by using the Workbooks property of the Application object. The [MSDN Documentation](#) explains it with references.

## Section 10.4: Opening A (New) Workbook, Even If It's Already Open

If you want to access a workbook that's already open, then getting the assignment from the Workbooks collection is straightforward:

```
dim myWB as Workbook
Set myWB = Workbooks("UsuallyFullPathnameOfWorkbook.xlsx")
```

If you want to create a new workbook, then use the Workbooks collection object to Add a new entry.

```
Dim myNewWB as Workbook
Set myNewWB = Workbooks.Add
```

There are times when you may not or (or care) if the workbook you need is open already or not, or possible does not exist. The example function shows how to always return a valid workbook object.

```
Option Explicit
Function GetWorkbook(ByVal wbFilename As String) As Workbook
    '--- returns a workbook object for the given filename, including checks
    '    for when the workbook is already open, exists but not open, or
    '    does not yet exist (and must be created)
    '    *** wbFilename must be a fully specified pathname

    Dim folderFile As String
    Dim returnedWB As Workbook

    '--- check if the file exists in the directory location
    folderFile = File(wbFilename)
    If folderFile = "" Then
        '--- the workbook doesn't exist, so create it
        Dim pos1 As Integer
        Dim fileExt As String
        Dim fileFormatNum As Long
        '--- in order to save the workbook correctly, we need to infer which workbook
        '    type the user intended from the file extension
        pos1 = InStrRev(sFullName, ".", , vbTextCompare)
        fileExt = Right(sFullName, Len(sFullName) - pos1)
        Select Case fileExt
            Case "xlsx"
                fileFormatNum = 51
            Case "xlsm"
                fileFormatNum = 52
            Case "xls"
                fileFormatNum = 56
            Case "xlsb"
                fileFormatNum = 50
            Case Else
                Err.Raise vbObjectError + 1000, "GetWorkbook function", _
                    "The file type you've requested (file extension) is not recognized." & _
                    "Please use a known extension: xlsx, xlsm, xls, or xlsb."
        End Select
        Set returnedWB = Workbooks.Add
        Application.DisplayAlerts = False
        returnedWB.SaveAs filename:=wbFilename, FileFormat:=fileFormatNum
```

```
Application.DisplayAlerts = True
    Set GetWorkbook = returnedWB
Else
    --- 工作簿已存在于目录中，因此检查它是否已经打开

    On Error Resume Next
    Set returnedWB = Workbooks(sFile)
    If returnedWB Is Nothing Then
        Set returnedWB = Workbooks.Open(sFullName)
    End If
End If
End Function
```

## 第10.5节：保存工作簿而不提示用户

使用VBA在现有工作簿中保存新数据时，通常会弹出提示，说明文件已存在。

要防止此弹出提示，必须禁止这类警告。

```
Application.DisplayAlerts = False      '禁用覆盖文件的用户提示
myWB.SaveAs FileName:="NewOrExistingFilename.xlsx"
Application.DisplayAlerts = True       '重新启用覆盖文件的用户提示
```

```
Application.DisplayAlerts = True
Set GetWorkbook = returnedWB
Else
    '--- the workbook exists in the directory, so check to see if
    '    it's already open or not
    On Error Resume Next
    Set returnedWB = Workbooks(sFile)
    If returnedWB Is Nothing Then
        Set returnedWB = Workbooks.Open(sFullName)
    End If
End If
End Function
```

## Section 10.5: Saving A Workbook Without Asking The User

Often saving new data in an existing workbook using VBA will cause a pop-up question noting that the file already exists.

To prevent this pop-up question, you have to suppress these types of alerts.

```
Application.DisplayAlerts = False      'disable user prompt to overwrite file
myWB.SaveAs FileName:="NewOrExistingFilename.xlsx"
Application.DisplayAlerts = True       're-enable user prompt to overwrite file
```

# 第11章：在VBA中使用Excel表格

本主题涉及在VBA中使用表格，假设具备Excel表格的相关知识。在VBA中，或者更准确地说，在Excel对象模型中，表格被称为ListObjects。ListObject最常用的属性包括ListRow(s)、ListColumn(s)、DataBodyRange、Range和HeaderRowRange。

## 第11.1节：实例化ListObject

```
Dim lo as ListObject
Dim MyRange as Range

Set lo = Sheet1.ListObjects(1)

'或者

Set lo = Sheet1.ListObjects("Table1")

'或者

Set lo = MyRange.ListObject
```

## 第11.2节：操作ListRows / ListColumns

```
Dim lo as ListObject
Dim lr as ListRow
Dim lc as ListColumn

Set lr = lo.ListRows.Add
设置 lr = lo.ListRows(5)

对于每个 lr 在 lo.ListRows 中
    lr.Range.ClearContents
lr.Range(1, lo.ListColumns("某列").Index).Value = 8
下一条

设置 lc = lo.ListColumns.Add
设置 lc = lo.ListColumns(4)
设置 lc = lo.ListColumns("标题 3")

对于每个 lc 在 lo.ListColumns 中
lc.DataBodyRange.ClearContents 'DataBodyRange 不包括标题行
lc.Range(1,1).Value = "新标题名称" 'Range 包括标题行
下一条
```

## 第11.3节：将Excel表转换为普通区域

```
声明 lo 作为 ListObject

设置 lo = Sheet1.ListObjects("Table1")
lo.Unlist
```

# Chapter 11: Working with Excel Tables in VBA

This topic is about working with tables in VBA, and assumes knowledge of Excel Tables. In VBA, or rather the Excel Object Model, tables are known as ListObjects. The most frequently used properties of a ListObject are ListRow(s), ListColumn(s), DataBodyRange, Range and HeaderRowRange.

## Section 11.1: Instantiating a ListObject

```
Dim lo as ListObject
Dim MyRange as Range

Set lo = Sheet1.ListObjects(1)

'or

Set lo = Sheet1.ListObjects("Table1")

'or

Set lo = MyRange.ListObject
```

## Section 11.2: Working with ListRows / ListColumns

```
Dim lo as ListObject
Dim lr as ListRow
Dim lc as ListColumn

Set lr = lo.ListRows.Add
Set lr = lo.ListRows(5)

For Each lr in lo.ListRows
    lr.Range.ClearContents
    lr.Range(1, lo.ListColumns("Some Column").Index).Value = 8
Next

Set lc = lo.ListColumns.Add
Set lc = lo.ListColumns(4)
Set lc = lo.ListColumns("Header 3")

For Each lc in lo.ListColumns
    lc.DataBodyRange.ClearContents 'DataBodyRange excludes the header row
    lc.Range(1,1).Value = "New Header Name" 'Range includes the header row
Next
```

## Section 11.3: Converting an Excel Table to a normal range

```
Dim lo as ListObject

Set lo = Sheet1.ListObjects("Table1")
lo.Unlist
```



# 第12章：循环遍历活动工作簿中的所有工作表

## 第12.1节：检索活动工作簿中所有工作表名称

```
Option Explicit

Sub LoopAllSheets()

Dim sht As Excel.Worksheet
' 声明一个字符串类型的数组，但不限定成员的最大数量
Dim sht_Name() As String
Dim i As Integer

' 获取活动工作簿中工作表的数量，并将其作为数组成员的最大数量

ReDim sht_Name(1 To ActiveWorkbook.Worksheets.count)

i = 1

' 遍历活动工作簿中的所有工作表
For Each sht In ActiveWorkbook.Worksheets
sht_Name(i) = sht.Name ' 获取每个工作表的名称并保存到数组中
i = i + 1
Next sht

End Sub
```

## 第12.2节：遍历文件夹中所有文件的所有工作表

```
Sub Theloopofloops()

Dim wbk As 工作簿
Dim Filename As 字符串
Dim path As 字符串
Dim rCell As 范围
Dim rRng As 范围
Dim ws0 As 工作表
Dim sheet As 工作表

path = "pathtofile(s)" & "\"
Filename = Dir(path & "*.xl??")
Set ws0 = ThisWorkbook.Sheets("Sheet1") '包含以防你需要区分_
当前打开的工作簿与包含代码的工作簿

Do While Len(Filename) > 0
DoEvents
Set wbk = Workbooks.Open(path & Filename, True, True)
For Each sheet In ActiveWorkbook.Worksheets '需要调整以指定
工作表。对每个工作表重复循环，因此是按工作表处理
Set rRng = sheet.Range("a1:a1000") '显然需要更改
For Each rCell In rRng.Cells
If rCell <> "" And rCell.Value <> vbNullString And rCell.Value <> 0 Then

'执行操作的代码

End If
Next rCell
Next sheet
wbk.Close
Set ws0 = ThisWorkbook.Sheets("Sheet1")
Filename = Dir(path & "*.xl??")
Loop

End Sub
```

# Chapter 12: Loop through all Sheets in Active Workbook

## Section 12.1: Retrieve all Worksheets Names in Active Workbook

```
Option Explicit

Sub LoopAllSheets()

Dim sht As Excel.Worksheet
' declare an array of type String without committing to maximum number of members
Dim sht_Name() As String
Dim i As Integer

' get the number of worksheets in Active Workbook , and put it as the maximum number of members in the array
ReDim sht_Name(1 To ActiveWorkbook.Worksheets.count)

i = 1

' loop through all worksheets in Active Workbook
For Each sht In ActiveWorkbook.Worksheets
sht_Name(i) = sht.Name ' get the name of each worksheet and save it in the array
i = i + 1
Next sht

End Sub
```

## Section 12.2: Loop Through all Sheets in all Files in a Folder

```
Sub Theloopofloops()

Dim wbk As Workbook
Dim Filename As String
Dim path As String
Dim rCell As Range
Dim rRng As Range
Dim ws0 As Worksheet
Dim sheet As Worksheet

path = "pathtofile(s)" & "\"
Filename = Dir(path & "*.xl??")
Set ws0 = ThisWorkbook.Sheets("Sheet1") 'included in case you need to differentiate_
between workbooks i.e currently opened workbook vs workbook containing code

Do While Len(Filename) > 0
DoEvents
Set wbk = Workbooks.Open(path & Filename, True, True)
For Each sheet In ActiveWorkbook.Worksheets 'this needs to be adjusted for specifying
sheets. Repeat loop for each sheet so thats on a per sheet basis
Set rRng = sheet.Range("a1:a1000") 'OBV needs to be changed
For Each rCell In rRng.Cells
If rCell <> "" And rCell.Value <> vbNullString And rCell.Value <> 0 Then

'code that does stuff

End If
Next rCell
Next sheet
wbk.Close
Set ws0 = ThisWorkbook.Sheets("Sheet1")
Filename = Dir(path & "*.xl??")
Loop

End Sub
```

```
        End If
    Next rCell
Next sheet
wbk.Close False
    Filename = Dir
Loop
End Sub
```

belindoc.com

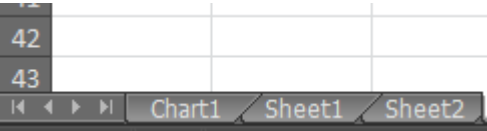
```
        End If
    Next rCell
Next sheet
wbk.Close False
    Filename = Dir
Loop
End Sub
```

# 第13章：使用工作表对象而非工作表（Sheet）对象

许多VBA用户认为Worksheets和Sheets对象是同义词。其实不是。

Sheets对象包含Worksheets和Charts。因此，如果我们的Excel工作簿中有图表，使用Sheets和Worksheets时应当小心，不要将它们视为同义词。

## 第13.1节：打印第一个对象的名称



```
Option Explicit

Sub CheckWorksheetsDiagram()

    Debug.Print Worksheets(1).Name
    Debug.Print Charts(1).Name
    Debug.Print Sheets(1).Name

End Sub
```

结果：

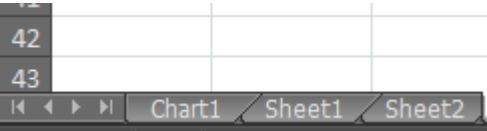
Sheet1  
Chart1  
Chart1

# Chapter 13: Use Worksheet object and not Sheet object

Plenty of VBA users consider Worksheets and Sheets objects synonyms. They are not.

Sheets object consists of both Worksheets and Charts. Thus, if we have charts in our Excel Workbook, we should be careful, not to use Sheets and Worksheets as synonyms.

## Section 13.1: Print the name of the first object



```
Option Explicit

Sub CheckWorksheetsDiagram()

    Debug.Print Worksheets(1).Name
    Debug.Print Charts(1).Name
    Debug.Print Sheets(1).Name

End Sub
```

The result:

Sheet1  
Chart1  
Chart1

# 第14章：查找工作表中最后使用的行或列的方法

## 第14.1节：查找列中最后一个非空单元格

在本例中，我们将介绍一种返回数据集中某列最后一个非空行的方法。

该方法适用于数据集中存在空白区域的情况。

但是，如果涉及合并单元格，应谨慎使用，因为End方法会在合并区域“停止”，返回合并区域的第一个单元格。

此外，隐藏行中的非空单元格将不会被考虑在内。

```
Sub 查找最后一行()  
    Dim wS As 工作表, LastRow As Long  
    Set wS = ThisWorkbook.Worksheets("Sheet1")  
  
    '这里我们查看A列  
    LastRow = wS.Cells(wS.Rows.Count, "A").End(xlUp).Row  
    Debug.Print LastRow  
End Sub
```

为了解决上述限制，以下代码行：  
LastRow = wS.Cells(wS.Rows.Count, "A").End(xlUp).Row

可以替换为：

- 1. 用于“Sheet1”的最后使用行：  
LastRow = wS.UsedRange.Row - 1 + wS.UsedRange.Rows.Count.
- 2. 查找"Sheet1"中"A"列最后一个非空单元格：

```
Dim i As Long  
For i = LastRow To 1 Step -1  
    If Not (IsEmpty(Cells(i, 1))) Then Exit For  
Next i  
LastRow = i
```

## 第14.2节：查找工作表中最后一个非空行

```
Private Sub Get_Last_Used_Row_Index()  
    Dim wS As Worksheet  
  
    Set wS = ThisWorkbook.Sheets("Sheet1")  
    Debug.Print LastRow_1(wS)  
    Debug.Print LastRow_0(wS)  
End Sub
```

您可以在以下两种选项中选择，取决于您是否想知道工作表中是否没有数据：

- 否：使用 LastRow\_1 ：您可以直接在 wS.Cells(LastRow\_1(wS),...) 中使用
- 是：使用 LastRow\_0 ：在使用函数结果之前，您需要测试返回值是否为0

# Chapter 14: Methods for Finding the Last Used Row or Column in a Worksheet

## Section 14.1: Find the Last Non-Empty Cell in a Column

In this example, we will look at a method for returning the last non-empty row in a column for a data set.

This method will work regardless of empty regions within the data set.

However **caution should be used if merged cells are involved**, as the End method will be "stopped" against a merged region, returning the first cell of the merged region.

In addition non-empty cells in **hidden rows** will not be taken into account.

```
Sub FindingLastRow()  
    Dim wS As Worksheet, LastRow As Long  
    Set wS = ThisWorkbook.Worksheets("Sheet1")  
  
    'Here we look in Column A  
    LastRow = wS.Cells(wS.Rows.Count, "A").End(xlUp).Row  
    Debug.Print LastRow  
End Sub
```

To address the limitations indicated above, the line:  
LastRow = wS.Cells(wS.Rows.Count, "A").End(xlUp).Row

may be replaced with:

- 1. for last used row of "Sheet1":  
LastRow = wS.UsedRange.Row - 1 + wS.UsedRange.Rows.Count.
- 2. for last non-empty cell of Column "A" in "Sheet1":

```
Dim i As Long  
For i = LastRow To 1 Step -1  
    If Not (IsEmpty(Cells(i, 1))) Then Exit For  
Next i  
LastRow = i
```

## Section 14.2: Find the Last Non-Empty Row in Worksheet

```
Private Sub Get_Last_Used_Row_Index()  
    Dim wS As Worksheet  
  
    Set wS = ThisWorkbook.Sheets("Sheet1")  
    Debug.Print LastRow_1(wS)  
    Debug.Print LastRow_0(wS)  
End Sub
```

You can choose between 2 options, regarding if you want to know if there is no data in the worksheet：

- NO : Use LastRow\_1 : You can use it directly within wS.Cells(LastRow\_1(wS), ...)
- YES : Use LastRow\_0 : You need to test if the result you get from the function is 0 or not before using it



```
Public Function LastRow_1(ws As Worksheet) As Double
    With ws
        If Application.WorksheetFunction.CountA(.Cells) <> 0 Then
            LastRow_1 = .Cells.Find(What:="*", _
                                   After:=.Range("A1"), _
                                   Lookat:=xlPart, _
                                   LookIn:=xlFormulas, _
                                   SearchOrder:=xlByRows, _
                                   SearchDirection:=xlPrevious, _
                                   MatchCase:=False).Row

        Else
            LastRow_1 = 1
        End If
    End With
End Function

Public Function LastRow_0(ws As Worksheet) As Double
    On Error Resume Next
    LastRow_0 = ws.Cells.Find(What:="*", _
                              After:=ws.Range("A1"), _
                              Lookat:=xlPart, _
                              LookIn:=xlFormulas, _
                              SearchOrder:=xlByRows, _
                              SearchDirection:=xlPrevious, _
                              MatchCase:=False).Row

End Function
```

### 第14.3节：查找工作表中最后一个非空列

```
Private Sub Get_Last_Used_Row_Index()
    Dim ws As Worksheet

    Set ws = ThisWorkbook.Sheets("Sheet1")
    Debug.Print LastCol_1(ws)
    Debug.Print LastCol_0(ws)
End Sub
```

您可以在以下两种选项中选择，取决于您是否想知道工作表中是否没有数据：

- 否：使用 LastCol\_1：你可以直接在 ws.Cells(...,LastCol\_1(ws)) 中使用它
- 是：使用 LastCol\_0：在使用函数结果之前，你需要测试该结果是否为0

```
Public Function LastCol_1(ws As 工作表) As Double
    With ws
        如果 Application.WorksheetFunction.CountA(.Cells) <> 0 那么
            LastCol_1 = .Cells.Find(What:="*", _
                                   After:=.Range("A1"), _
                                   Lookat:=xlPart, _
                                   LookIn:=xlFormulas, _
                                   SearchOrder:=xlByColumns, _
                                   SearchDirection:=xlPrevious, _
                                   MatchCase:=False).Column

        Else
            LastCol_1 = 1
        结束 如果
    结束 With
结束 Function
```

Err 对象的属性在函数退出时会自动重置为零。

```
Public Function LastRow_1(ws As Worksheet) As Double
    With ws
        If Application.WorksheetFunction.CountA(.Cells) <> 0 Then
            LastRow_1 = .Cells.Find(What:="*", _
                                   After:=.Range("A1"), _
                                   Lookat:=xlPart, _
                                   LookIn:=xlFormulas, _
                                   SearchOrder:=xlByRows, _
                                   SearchDirection:=xlPrevious, _
                                   MatchCase:=False).Row

        Else
            LastRow_1 = 1
        End If
    End With
End Function

Public Function LastRow_0(ws As Worksheet) As Double
    On Error Resume Next
    LastRow_0 = ws.Cells.Find(What:="*", _
                              After:=ws.Range("A1"), _
                              Lookat:=xlPart, _
                              LookIn:=xlFormulas, _
                              SearchOrder:=xlByRows, _
                              SearchDirection:=xlPrevious, _
                              MatchCase:=False).Row

End Function
```

### Section 14.3: Find the Last Non-Empty Column in Worksheet

```
Private Sub Get_Last_Used_Row_Index()
    Dim ws As Worksheet

    Set ws = ThisWorkbook.Sheets("Sheet1")
    Debug.Print LastCol_1(ws)
    Debug.Print LastCol_0(ws)
End Sub
```

You can choose between 2 options, regarding if you want to know if there is no data in the worksheet :

- NO : Use LastCol\_1 : You can use it directly within ws.Cells(...,LastCol\_1(ws))
- YES : Use LastCol\_0 : You need to test if the result you get from the function is 0 or not before using it

```
Public Function LastCol_1(ws As Worksheet) As Double
    With ws
        If Application.WorksheetFunction.CountA(.Cells) <> 0 Then
            LastCol_1 = .Cells.Find(What:="*", _
                                   After:=.Range("A1"), _
                                   Lookat:=xlPart, _
                                   LookIn:=xlFormulas, _
                                   SearchOrder:=xlByColumns, _
                                   SearchDirection:=xlPrevious, _
                                   MatchCase:=False).Column

        Else
            LastCol_1 = 1
        End If
    End With
End Function
```

The Err object's properties are automatically reset to zero upon function exit.

```
公共函数 LastCol_0(ws 作为 工作表) 作为 双精度
    出错时继续下一步
    LastCol_0 = ws.Cells.Find(What:="*", _
                               After:=ws.Range("A1"), _
                               Lookat:=xlPart, _
                               LookIn:=xlFormulas, _
                               SearchOrder:=xlByColumns, _
                               SearchDirection:=xlPrevious, _
                               MatchCase:=False).Column

End Function
```

## 第14.4节：查找行中最后一个非空单元格

在本例中，我们将介绍一种返回行中最后一个非空列的方法。

该方法适用于数据集中存在空白区域的情况。

但是，如果涉及合并单元格，应谨慎使用，因为End方法会在合并区域“停止”，返回合并区域的第一个单元格。

此外，隐藏列中的非空单元格将不被考虑。

```
Sub 查找最后一列()
    Dim 工作表 As Worksheet, 最后列 As Long
    Set 工作表 = ThisWorkbook.Worksheets("Sheet1")

    '这里我们查看第1行
    最后列 = 工作表.Cells(1, 工作表.Columns.Count).End(xlToLeft).Column
    Debug.Print 最后列
End Sub
```

## 第14.5节：获取范围内最后一个单元格的行号

```
'如果只有一个区域（非多个区域）：
With Range("A3:D20")
    Debug.Print .Cells(.Cells.CountLarge).Row
    Debug.Print .Item(.Cells.CountLarge).Row '使用 .item 也是可以的
End With '调试输出：20

'多个区域（即使只有一个区域也适用）：
Dim rngArea As Range, LastRow As Long
With Range("A3:D20, E5:I50, H20:R35")
    For Each rngArea In .Areas
        如果 rngArea(rngArea.Cells.CountLarge).Row > LastRow 那么
            LastRow = rngArea(rngArea.Cells.CountLarge).Row
        End If
    Next
    下一步
    Debug.Print LastRow '调试打印：50
结束 With
```

## 第14.6节：使用命名区域查找最后一行

如果您的工作表中有一个命名区域，且您想动态获取该动态命名区域的最后一行。本节还涵盖了命名区域不从第一行开始的情况。

```
Sub 查找最后一行()
```

```
Public Function LastCol_0(ws As Worksheet) As Double
    On Error Resume Next
    LastCol_0 = ws.Cells.Find(What:="*", _
                               After:=ws.Range("A1"), _
                               Lookat:=xlPart, _
                               LookIn:=xlFormulas, _
                               SearchOrder:=xlByColumns, _
                               SearchDirection:=xlPrevious, _
                               MatchCase:=False).Column

End Function
```

## Section 14.4: Find the Last Non-Empty Cell in a Row

In this example, we will look at a method for returning the last non-empty column in a row.

This method will work regardless of empty regions within the data set.

However **caution** should be used if **merged cells** are involved, as the **End** method will be "stopped" against a merged region, returning the first cell of the merged region.

In addition non-empty cells in **hidden columns** will not be taken into account.

```
Sub FindingLastCol()
    Dim ws As Worksheet, LastCol As Long
    Set ws = ThisWorkbook.Worksheets("Sheet1")

    'Here we look in Row 1
    LastCol = ws.Cells(1, ws.Columns.Count).End(xlToLeft).Column
    Debug.Print LastCol
End Sub
```

## Section 14.5: Get the row of the last cell in a range

```
'if only one area (not multiple areas):
With Range("A3:D20")
    Debug.Print .Cells(.Cells.CountLarge).Row
    Debug.Print .Item(.Cells.CountLarge).Row 'using .item is also possible
End With 'Debug prints: 20

'with multiple areas (also works if only one area):
Dim rngArea As Range, LastRow As Long
With Range("A3:D20, E5:I50, H20:R35")
    For Each rngArea In .Areas
        If rngArea(rngArea.Cells.CountLarge).Row > LastRow Then
            LastRow = rngArea(rngArea.Cells.CountLarge).Row
        End If
    Next
    Debug.Print LastRow 'Debug prints: 50
End With
```

## Section 14.6: Find Last Row Using Named Range

In case you have a Named Range in your Sheet, and you want to dynamically get the last row of that Dynamic Named Range. Also covers cases where the Named Range doesn't start from the first Row.

```
Sub FindingLastRow()
```

Dim sht 作为 工作表  
Dim LastRow 作为 长整型  
Dim FirstRow 作为 长整型

设置 sht = ThisWorkbook.Worksheets("form")

'使用命名区域 "MyNameRange"  
FirstRow = sht.Range("MyNameRange").Row

' 如果 "MyNameRange" 不从第1行开始  
LastRow = sht.Range("MyNameRange").Rows.count + FirstRow - 1

End Sub

更新：  
@Jeeped 指出一个潜在漏洞，针对具有非连续行的命名区域会产生意外结果。为了解决该问题，代码修订如下。  
假设：目标工作表 = form，命名区域 = MyNameRange

```
Sub 查找最后一行()  
    Dim rw As Range, rwMax As Long  
    For Each rw In Sheets("form").Range("MyNameRange").Rows  
        If rw.Row > rwMax Then rwMax = rw.Row  
    Next  
    MsgBox "工作表 'form' 中 'MyNameRange' 的最后一行: " & rwMax  
End Sub
```

### 第14.7节：Range.CurrentRegion中的最后一个单元格

Range.CurrentRegion 是由空单元格包围的矩形区域。带有公式如 ="" 或 ' 的空白单元格不被视为空白（即使使用 ISBLANK Excel函数）。

```
Dim rng As Range, lastCell As Range  
Set rng = Range("C3").CurrentRegion ' 或 Set rng = Sheet1.UsedRange.CurrentRegion  
Set lastCell = rng(rng.Rows.Count, rng.Columns.Count)
```

### 第14.8节：查找工作表中最后一个非空单元格 - 性能（数组）

- 第一个函数，使用数组，速度**快得多**
- 如果调用时没有传入可选参数，默认值为.ThisWorkbook.ActiveSheet
- 如果范围为空，默认返回Cell(1,1)，而不是Nothing

速度：

GetMaxCell（数组）：耗时：0.0000790063秒  
GetMaxCell（查找）：耗时：0.0002903480秒

使用MicroTimer测量

```
Public Function GetLastCell(Optional ByVal ws As Worksheet = Nothing) As Range  
    Dim uRng As Range, uArr As Variant, r As Long, c As Long  
    Dim ubR As Long, ubC As Long, lRow As Long
```

```
Dim sht As Worksheet  
Dim LastRow As Long  
Dim FirstRow As Long
```

Set sht = ThisWorkbook.Worksheets("form")

'Using Named Range "MyNameRange"  
FirstRow = sht.Range("MyNameRange").Row

' in case "MyNameRange" doesn't start at Row 1  
LastRow = sht.Range("MyNameRange").Rows.count + FirstRow - 1

End Sub

Update:  
A potential loophole was pointed out by @Jeeped for a a named range with non-contiguous rows as it generates unexpected result. To addresses that issue, the code is revised as below.  
Asumptions: targes sheet = form, named range = MyNameRange

```
Sub FindingLastRow()  
    Dim rw As Range, rwMax As Long  
    For Each rw In Sheets("form").Range("MyNameRange").Rows  
        If rw.Row > rwMax Then rwMax = rw.Row  
    Next  
    MsgBox "Last row of 'MyNameRange' under Sheets 'form': " & rwMax  
End Sub
```

### Section 14.7: Last cell in Range.CurrentRegion

Range.CurrentRegion is a rectangular range area surrounded by empty cells. Blank cells with formulas such as ="" or ' are not considered blank (even by the ISBLANK Excel function).

```
Dim rng As Range, lastCell As Range  
Set rng = Range("C3").CurrentRegion ' or Set rng = Sheet1.UsedRange.CurrentRegion  
Set lastCell = rng(rng.Rows.Count, rng.Columns.Count)
```

### Section 14.8: Find the Last Non-Empty Cell in Worksheet - Performance (Array)

- The first function, using an array, is **much faster**
- If called without the optional parameter, will default to .ThisWorkbook.ActiveSheet
- If the range is empty will returns Cell( 1, 1 ) as default, instead of Nothing

Speed:

GetMaxCell (Array): Duration: 0.0000790063 seconds  
GetMaxCell (Find ): Duration: 0.0002903480 seconds

.Measured with MicroTimer

```
Public Function GetLastCell(Optional ByVal ws As Worksheet = Nothing) As Range  
    Dim uRng As Range, uArr As Variant, r As Long, c As Long  
    Dim ubR As Long, ubC As Long, lRow As Long
```

```

If ws Is Nothing Then Set ws = Application.ThisWorkbook.ActiveSheet
Set uRng = ws.UsedRange
uArr = uRng
If IsEmpty(uArr) Then
    Set GetLastCell = ws.Cells(1, 1): Exit Function
End If
If Not IsArray(uArr) Then
    Set GetLastCell = ws.Cells(uRng.Row, uRng.Column): Exit Function
End If
ubR = UBound(uArr, 1): ubC = UBound(uArr, 2)
For r = ubR To 1 Step -1 '----- last row
    For c = ubC To 1 Step -1
        If Not IsError(uArr(r, c)) Then
            If Len(Trim$(uArr(r, c))) > 0 Then
                lRow = r: Exit For
            End If
        End If
    Next c
    下一步
    If lRow > 0 Then Exit For
    下一步
    If lRow = 0 Then lRow = ubR
    For c = ubC To 1 Step -1 '----- last col
        For r = lRow To 1 Step -1
            If Not IsError(uArr(r, c)) Then
                If Len(Trim$(uArr(r, c))) > 0 Then
                    Set GetLastCell = ws.Cells(lRow + uRng.Row - 1, c + uRng.Column - 1)
                    Exit Function
                End If
            End If
        Next r
    Next c
    下一步
End Function

```

'使用 Find 返回最后一个单元格 (最大行和最大列)

公共函数 GetMaxCell2(可选 ByRef rng 作为 Range = 无) 作为 Range '使用 Find

常量 NONEMPTY 作为 字符串 = "\*"

声明 lRow 作为 Range, lCol 作为 Range

如果 rng 是无 Then 设置 rng = Application.ThisWorkbook.ActiveSheet.UsedRange

如果 WorksheetFunction.CountA(rng) = 0 则

设置 GetMaxCell2 = rng.Parent.Cells(1, 1)

Else

使用 rng

设置 lRow = .Cells.Find(What:=NONEMPTY, LookIn:=xlFormulas, \_  
After:=.Cells(1, 1), \_  
SearchDirection:=xlPrevious, \_  
SearchOrder:=xlByRows)

如果 不 lRow 是无 Then

Set lCol = .Cells.Find(What:=NONEMPTY, LookIn:=xlFormulas, \_  
After:=.Cells(1, 1), \_  
SearchDirection:=xlPrevious, \_  
SearchOrder:=xlByColumns)

Set GetMaxCell2 = .Parent.Cells(lRow.Row, lCol.Column)

End If

End With

End If

```

If ws Is Nothing Then Set ws = Application.ThisWorkbook.ActiveSheet
Set uRng = ws.UsedRange
uArr = uRng
If IsEmpty(uArr) Then
    Set GetLastCell = ws.Cells(1, 1): Exit Function
End If
If Not IsArray(uArr) Then
    Set GetLastCell = ws.Cells(uRng.Row, uRng.Column): Exit Function
End If
ubR = UBound(uArr, 1): ubC = UBound(uArr, 2)
For r = ubR To 1 Step -1 '----- last row
    For c = ubC To 1 Step -1
        If Not IsError(uArr(r, c)) Then
            If Len(Trim$(uArr(r, c))) > 0 Then
                lRow = r: Exit For
            End If
        End If
    Next c
    If lRow > 0 Then Exit For
Next
If lRow = 0 Then lRow = ubR
For c = ubC To 1 Step -1 '----- last col
    For r = lRow To 1 Step -1
        If Not IsError(uArr(r, c)) Then
            If Len(Trim$(uArr(r, c))) > 0 Then
                Set GetLastCell = ws.Cells(lRow + uRng.Row - 1, c + uRng.Column - 1)
                Exit Function
            End If
        End If
    Next r
Next
End Function

```

'Returns last cell (max row & max col) using Find

Public Function GetMaxCell2(Optional ByRef rng As Range = Nothing) As Range 'Using Find

Const NONEMPTY As String = "\*"

Dim lRow As Range, lCol As Range

If rng Is Nothing Then Set rng = Application.ThisWorkbook.ActiveSheet.UsedRange

If WorksheetFunction.CountA(rng) = 0 Then

Set GetMaxCell2 = rng.Parent.Cells(1, 1)

Else

With rng

Set lRow = .Cells.Find(What:=NONEMPTY, LookIn:=xlFormulas, \_  
After:=.Cells(1, 1), \_  
SearchDirection:=xlPrevious, \_  
SearchOrder:=xlByRows)

If Not lRow Is Nothing Then

Set lCol = .Cells.Find(What:=NONEMPTY, LookIn:=xlFormulas, \_  
After:=.Cells(1, 1), \_  
SearchDirection:=xlPrevious, \_  
SearchOrder:=xlByColumns)

Set GetMaxCell2 = .Parent.Cells(lRow.Row, lCol.Column)

End If

End With

End If

End Function

.

MicroTimer:

```
Private Declare PtrSafe Function getFrequency Lib "Kernel32" Alias "QueryPerformanceFrequency"
(cyFrequency As Currency) As Long
Private Declare PtrSafe Function getTickCount Lib "Kernel32" Alias "QueryPerformanceCounter"
(cyTickCount As Currency) As Long

Function MicroTimer() As Double
    Dim cyTicks1 As Currency
    Static cyFrequency As Currency

MicroTimer = 0
    如果 cyFrequency = 0 那么 getFrequency cyFrequency          '获取频率
    getTickCount cyTicks1                                     '获取计数
    如果 cyFrequency 那么 MicroTimer = cyTicks1 / cyFrequency '返回秒数
结束函数
```

End Function

.

MicroTimer:

```
Private Declare PtrSafe Function getFrequency Lib "Kernel32" Alias "QueryPerformanceFrequency"
(cyFrequency As Currency) As Long
Private Declare PtrSafe Function getTickCount Lib "Kernel32" Alias "QueryPerformanceCounter"
(cyTickCount As Currency) As Long

Function MicroTimer() As Double
    Dim cyTicks1 As Currency
    Static cyFrequency As Currency

MicroTimer = 0
    If cyFrequency = 0 Then getFrequency cyFrequency          'Get frequency
    getTickCount cyTicks1                                     'Get ticks
    If cyFrequency Then MicroTimer = cyTicks1 / cyFrequency 'Returns Seconds
End Function
```



# 第15章：在活动工作表中使用组合框创建下拉菜单

这是一个简单的示例，演示如何通过在工作簿的活动工作表中插入一个组合框ActiveX控件来创建下拉菜单。您可以在工作表的任何激活单元格中插入五首吉米·亨德里克斯的歌曲之一，并能够相应地清除它。

## 第15.1节：示例2：未包含的选项

此示例用于指定可能未包含在可用住房及其附属设施数据库中的选项。

它建立在前面的例子基础上，但有一些不同之处：

- 1. 通过将代码合并到一个过程，两个过程对于单个组合框不再必要。
- 2. 使用 LinkedCell 属性以确保每次用户选择的正确输入
- 3. 包含备份功能以确保活动单元格位于正确的列，并基于以往经验添加错误预防代码，防止数值在填充到活动单元格时被格式化为字符串。

```
Private Sub cboNotIncl_Change()  
  
Dim n As Long  
Dim notincl_array(1 To 9) As String  
  
n = myTarget.Row  
  
    如果 n >= 3 且 n < 10000 则  
  
        如果 myTarget.Address = "$G$" & n 则  
  
            '设置未包含服务的数组元素  
notincl_array(1) = "中央空调"  
notincl_array(2) = "热水"  
notincl_array(3) = "加热器租赁"  
notincl_array(4) = "公用设施"  
notincl_array(5) = "停车"  
notincl_array(6) = "互联网"  
notincl_array(7) = "电力"  
notincl_array(8) = "电力/热水/加热器租赁"  
notincl_array(9) = "电力和公用设施"  
  
            cboNotIncl.List = notincl_array()  
  
        Else  
  
            退出子程序  
  
        End If  
  
    使用 cboNotIncl  
  
    '确保组合框移动到目标单元格  
    .Left = myTarget.Left  
    .Top = myTarget.Top
```

# Chapter 15: Creating a drop-down menu in the Active Worksheet with a Combo Box

This is a simple example demonstrating how to create a drop down menu in the Active Sheet of your workbook by inserting a Combo Box Activex object in the sheet. You'll be able to insert one of five Jimi Hendrix songs in any activated cell of the sheet and be able to clear it, accordingly.

## Section 15.1: Example 2: Options Not Included

This example is used in specifying options that might not be included in a database of available housing and its attendant amenities.

It builds on the previous example, with some differences:

- 1. Two procedures are no longer necessary for a single combo box, done by combining the code into a single procedure.
- 2. The use of the LinkedCell property to allow for the correct input of the user selection every time
- 3. The inclusion of a backup feature for ensuring the active cell is in the correct column and an error prevention code, based on previous experience, where numeric values would formatted as strings when populated to the active cell.

```
Private Sub cboNotIncl_Change()  
  
Dim n As Long  
Dim notincl_array(1 To 9) As String  
  
n = myTarget.Row  
  
    If n >= 3 And n < 10000 Then  
  
        If myTarget.Address = "$G$" & n Then  
  
            'set up the array elements for the not included services  
notincl_array(1) = "Central Air"  
notincl_array(2) = "Hot Water"  
notincl_array(3) = "Heater Rental"  
notincl_array(4) = "Utilities"  
notincl_array(5) = "Parking"  
notincl_array(6) = "Internet"  
notincl_array(7) = "Hydro"  
notincl_array(8) = "Hydro/Hot Water/Heater Rental"  
notincl_array(9) = "Hydro and Utilities"  
  
            cboNotIncl.List = notincl_array()  
  
        Else  
  
            Exit Sub  
  
        End If  
  
    With cboNotIncl  
  
        'make sure the combo box moves to the target cell  
        .Left = myTarget.Left  
        .Top = myTarget.Top
```

```
'调整单元格大小以适应组合框
myTarget.ColumnWidth = .Width * 0.18

'通过编辑一些字体属性使其看起来更美观
.Font.Size = 11
.Font.Bold = False

'用用户选择填充单元格，并备份保证其在G列

如果 myTarget.Address = "$G$" & n 则

    .LinkedCell = myTarget.Address

    '防止数值被格式化为文本时出错
    myTarget.EntireColumn.TextToColumns

End If

End With

结束 如果 '确保活动单元格仅在第3行到第1000行之间

End Sub
```

上述宏在工作表模块中每次激活单元格时通过SelectionChange事件触发：

```
Public myTarget As Range

Private Sub Worksheet_SelectionChange(ByVal Target As Range)

    Set myTarget = Target

    'switch for Not Included
    If Target.Column = 7 And Target.Cells.Count = 1 Then

        Application.Run "Module1.cboNotIncl_Change"

    End If

End Sub
```

## 第15.2节：吉米·亨德里克斯菜单

通常，代码放置在工作表的模块中。

这是 Worksheet\_SelectionChange 事件，每次在活动工作表中选择不同的单元格时都会触发。你可以从代码窗口上方的第一个下拉菜单中选择“Worksheet”，并从旁边的下拉菜单中选择“Selection\_Change”。在这种情况下，每次激活一个单元格时，代码都会重定向到组合框的代码。

```
Private Sub Worksheet_SelectionChange(ByVal Target As Range)

    ComboBox1_Change

End Sub
```

这里，专用于组合框的例程默认编码到Change事件中。里面有一个固定数组，数组中填充了所有选项。最后一个位置不是“清除”选项，该选项将用于清除单元格内容。然后将该数组传递给组合框，并传递给执行操作的例程。

```
'adjust the size of the cell to fit the combo box
myTarget.ColumnWidth = .Width * 0.18

'make it look nice by editing some of the font attributes
.Font.Size = 11
.Font.Bold = False

'populate the cell with the user choice, with a backup guarantee that it's in column G

If myTarget.Address = "$G$" & n Then

    .LinkedCell = myTarget.Address

    'prevent an error where a numerical value is formatted as text
    myTarget.EntireColumn.TextToColumns

End If

End With

End If 'ensure that the active cell is only between rows 3 and 1000

End Sub
```

The above macro is initiated every time a cell is activated with the SelectionChange event in the worksheet module:

```
Public myTarget As Range

Private Sub Worksheet_SelectionChange(ByVal Target As Range)

    Set myTarget = Target

    'switch for Not Included
    If Target.Column = 7 And Target.Cells.Count = 1 Then

        Application.Run "Module1.cboNotIncl_Change"

    End If

End Sub
```

## Section 15.2: Jimi Hendrix Menu

In general, the code is placed in the module of a sheet.

This is the Worksheet\_SelectionChange event, which fires each time a different cell is selected in the active sheet. You can select "Worksheet" from the first drop-down menu above the code window, and "Selection\_Change" from the drop down menu next to it. In this case, every time you activate a cell, the code is redirected to the Combo Box's code.

```
Private Sub Worksheet_SelectionChange(ByVal Target As Range)

    ComboBox1_Change

End Sub
```

Here, the routine dedicated to the ComboBox is coded to the Change event by default. In it, there is a fixed array, populated with all the options. Not the CLEAR option in the last position, which will be used to clear the contents of a cell. The array then is handed to to the Combo Box and passed to the routine that does the work.

```
Private Sub ComboBox1_Change()
```

```
Dim myarray(0 To 5)
myarray(0) = "嘿, 乔"
myarray(1) = "小翅膀"
myarray(2) = "巫毒小子"
myarray(3) = "紫色迷雾"
myarray(4) = "风在哭泣玛丽"
myarray(5) = "清除"
```

```
With ComboBox1
    .List = myarray()
End With
```

```
FillACell myarray()
```

```
End Sub
```

该数组被传递给填充单元格的例程，用歌曲名称或空值清空单元格。首先，一个整数变量被赋值为用户所选项的位置。然后，组合框被移动到用户激活的单元格的左上角，并调整其尺寸以使体验更流畅。

然后，活动单元格被赋予整数变量中对应位置的值，该变量跟踪用户的选择。如果用户从选项中选择“清除”，则该单元格将被清空。

整个例程会对每个选中的单元格重复执行。

```
Sub FillACell(MyArray As Variant)
```

```
Dim n As Integer
```

```
n = ComboBox1.ListIndex
```

```
ComboBox1.Left = ActiveCell.Left
ComboBox1.Top = ActiveCell.Top
Columns(ActiveCell.Column).ColumnWidth = ComboBox1.Width * 0.18
```

```
ActiveCell = MyArray(n)
```

```
If ComboBox1 = "CLEAR" Then
    Range(ActiveCell.Address) = ""
End If
```

```
End Sub
```

```
Private Sub ComboBox1_Change()
```

```
Dim myarray(0 To 5)
myarray(0) = "Hey Joe"
myarray(1) = "Little Wing"
myarray(2) = "Voodoo Child"
myarray(3) = "Purple Haze"
myarray(4) = "The Wind Cries Mary"
myarray(5) = "CLEAR"
```

```
With ComboBox1
    .List = myarray()
End With
```

```
FillACell myarray()
```

```
End Sub
```

The array is passed to the routine that fills the cells with the song name or null value to empty them. First, an integer variable is given the value of the position of the choice that the user makes. Then, the Combo Box is moved to the TOP LEFT corner of the cell the user activates and its dimensions adjusted to make the experience more fluid. The active cell is then assigned the value in the position in the integer variable, which tracks the user choice. In case the user selects CLEAR from the options, the cell is emptied.

The entire routine repeats for each selected cell.

```
Sub FillACell(MyArray As Variant)
```

```
Dim n As Integer
```

```
n = ComboBox1.ListIndex
```

```
ComboBox1.Left = ActiveCell.Left
ComboBox1.Top = ActiveCell.Top
Columns(ActiveCell.Column).ColumnWidth = ComboBox1.Width * 0.18
```

```
ActiveCell = MyArray(n)
```

```
If ComboBox1 = "CLEAR" Then
    Range(ActiveCell.Address) = ""
End If
```

```
End Sub
```

# 第16章：文件系统对象

## 第16.1节：文件、文件夹、驱动器是否存在

文件是否存在：

```
Sub FileExists()  
    Dim fso as Scripting.FileSystemObject  
    Set fso = CreateObject("Scripting.FileSystemObject")  
    If fso.FileExists("D:est.txt") = True Then  
MsgBox "文件存在。"  
    Else  
MsgBox "文件不存在。"  
    End If  
End Sub
```

文件夹存在：

```
子程序 FolderExists()  
    Dim fso 作为 Scripting.FileSystemObject  
    Set fso = CreateObject("Scripting.FileSystemObject")  
    如果 fso.FolderExists("D:estFolder") = True 那么  
        MsgBox "文件夹存在。"  
    否则  
MsgBox "文件夹不存在。"  
    End If  
End Sub
```

驱动器存在：

```
子程序 DriveExists()  
    Dim fso 作为 Scripting.FileSystemObject  
    Set fso = CreateObject("Scripting.FileSystemObject")  
    如果 fso.DriveExists("D:\") = True 那么  
MsgBox "驱动器存在。"  
    Else  
MsgBox "驱动器不存在。"  
    End If  
End Sub
```

## 第16.2节：基本文件操作

复制：

```
子程序 CopyFile()  
    声明 fso 作为 Scripting.FileSystemObject  
    设置 fso = CreateObject("Scripting.FileSystemObject")  
    fso.CopyFile "c:\Documents and Settings\Makro.txt", "c:\Documents and Settings\Macros\  
End Sub
```

移动：

```
子程序 MoveFile()  
    声明 fso 作为 Scripting.FileSystemObject  
    设置 fso = CreateObject("Scripting.FileSystemObject")  
    fso.MoveFile "c:*.txt", "c:\Documents and Settings\  
End Sub
```

删除：

```
子程序 DeleteFile()  
    声明 fso  
    Set fso = CreateObject("Scripting.FileSystemObject")  
    fso.DeleteFile "c:\Documents and Settings\Macros\Makro.txt"
```

# Chapter 16: File System Object

## Section 16.1: File, folder, drive exists

File exists:

```
Sub FileExists()  
    Dim fso as Scripting.FileSystemObject  
    Set fso = CreateObject("Scripting.FileSystemObject")  
    If fso.FileExists("D:\test.txt") = True Then  
        MsgBox "The file is exists."  
    Else  
        MsgBox "The file isn't exists."  
    End If  
End Sub
```

Folder exists:

```
Sub FolderExists()  
    Dim fso as Scripting.FileSystemObject  
    Set fso = CreateObject("Scripting.FileSystemObject")  
    If fso.FolderExists("D:\testFolder") = True Then  
        MsgBox "The folder is exists."  
    Else  
        MsgBox "The folder isn't exists."  
    End If  
End Sub
```

Drive exists:

```
Sub DriveExists()  
    Dim fso as Scripting.FileSystemObject  
    Set fso = CreateObject("Scripting.FileSystemObject")  
    If fso.DriveExists("D:\") = True Then  
        MsgBox "The drive is exists."  
    Else  
        MsgBox "The drive isn't exists."  
    End If  
End Sub
```

## Section 16.2: Basic file operations

Copy:

```
Sub CopyFile()  
    Dim fso as Scripting.FileSystemObject  
    Set fso = CreateObject("Scripting.FileSystemObject")  
    fso.CopyFile "c:\Documents and Settings\Makro.txt", "c:\Documents and Settings\Macros\  
End Sub
```

Move:

```
Sub MoveFile()  
    Dim fso as Scripting.FileSystemObject  
    Set fso = CreateObject("Scripting.FileSystemObject")  
    fso.MoveFile "c:*.txt", "c:\Documents and Settings\  
End Sub
```

Delete:

```
Sub DeleteFile()  
    Dim fso  
    Set fso = CreateObject("Scripting.FileSystemObject")  
    fso.DeleteFile "c:\Documents and Settings\Macros\Makro.txt"
```

End Sub

## 第16.3节：基本文件夹操作

创建：

```
子程序 CreateFolder()  
    Dim fso 作为 Scripting.FileSystemObject  
    Set fso = CreateObject("Scripting.FileSystemObject")  
    fso.CreateFolder "c:\Documents and Settings\NewFolder"  
End Sub
```

复制：

```
子程序 CopyFolder()  
    Dim fso 作为 Scripting.FileSystemObject  
    Set fso = CreateObject("Scripting.FileSystemObject")  
    fso.CopyFolder "C:\Documents and Settings\NewFolder", "C:\"  
End Sub
```

移动：

```
子程序 MoveFolder()  
    Dim fso 作为 Scripting.FileSystemObject  
    Set fso = CreateObject("Scripting.FileSystemObject")  
    fso.MoveFolder "C:\Documents and Settings\NewFolder", "C:\"  
End Sub
```

删除：

```
子程序 DeleteFolder()  
    Dim fso 作为 Scripting.FileSystemObject  
    Set fso = CreateObject("Scripting.FileSystemObject")  
    fso.DeleteFolder "C:\Documents and Settings\NewFolder"  
End Sub
```

## 第16.4节：其他操作

获取文件名：

```
子程序 GetFileName()  
    Dim fso 作为 Scripting.FileSystemObject  
    Set fso = CreateObject("Scripting.FileSystemObject")  
    MsgBox fso.GetFileName("c:\Documents and Settings\Makro.txt")  
End Sub
```

结果：Makro.txt

获取基本名：

```
子程序 GetBaseName()  
    Dim fso 作为 Scripting.FileSystemObject  
    Set fso = CreateObject("Scripting.FileSystemObject")  
    MsgBox fso.GetBaseName("c:\Documents and Settings\Makro.txt")  
End Sub
```

结果：Makro

获取扩展名：

```
子程序 GetExtensionName()  
    Dim fso 作为 Scripting.FileSystemObject  
    Set fso = CreateObject("Scripting.FileSystemObject")  
    MsgBox fso.GetExtensionName("c:\Documents and Settings\Makro.txt")
```

End Sub

## Section 16.3: Basic folder operations

Create:

```
Sub CreateFolder()  
    Dim fso as Scripting.FileSystemObject  
    Set fso = CreateObject("Scripting.FileSystemObject")  
    fso.CreateFolder "c:\Documents and Settings\NewFolder"  
End Sub
```

Copy:

```
Sub CopyFolder()  
    Dim fso as Scripting.FileSystemObject  
    Set fso = CreateObject("Scripting.FileSystemObject")  
    fso.CopyFolder "C:\Documents and Settings\NewFolder", "C:\"  
End Sub
```

Move:

```
Sub MoveFolder()  
    Dim fso as Scripting.FileSystemObject  
    Set fso = CreateObject("Scripting.FileSystemObject")  
    fso.MoveFolder "C:\Documents and Settings\NewFolder", "C:\"  
End Sub
```

Delete:

```
Sub DeleteFolder()  
    Dim fso as Scripting.FileSystemObject  
    Set fso = CreateObject("Scripting.FileSystemObject")  
    fso.DeleteFolder "C:\Documents and Settings\NewFolder"  
End Sub
```

## Section 16.4: Other operations

Get file name:

```
Sub GetFileName()  
    Dim fso as Scripting.FileSystemObject  
    Set fso = CreateObject("Scripting.FileSystemObject")  
    MsgBox fso.GetFileName("c:\Documents and Settings\Makro.txt")  
End Sub
```

Result: Makro.txt

Get base name:

```
Sub GetBaseName()  
    Dim fso as Scripting.FileSystemObject  
    Set fso = CreateObject("Scripting.FileSystemObject")  
    MsgBox fso.GetBaseName("c:\Documents and Settings\Makro.txt")  
End Sub
```

Result: Makro

Get extension name:

```
Sub GetExtensionName()  
    Dim fso as Scripting.FileSystemObject  
    Set fso = CreateObject("Scripting.FileSystemObject")  
    MsgBox fso.GetExtensionName("c:\Documents and Settings\Makro.txt")
```



End Sub

结果：txt

获取驱动器名称：

```
子程序 获取驱动器名称()  
    定义 fso 作为 Scripting.FileSystemObject  
    设置 fso = 创建对象("Scripting.FileSystemObject")  
    消息框 fso.GetDriveName("c:\Documents and Settings\Makro.txt")  
End Sub
```

结果：c:

End Sub

Result: txt

Get drive name:

```
Sub GetDriveName()  
    Dim fso as Scripting.FileSystemObject  
    Set fso = CreateObject("Scripting.FileSystemObject")  
    MsgBox fso.GetDriveName("c:\Documents and Settings\Makro.txt")  
End Sub
```

Result: c:

# 第17章：数据透视表

## 第17.1节：向数据透视表添加字段

向数据透视表添加字段时需要注意的两个重要事项是方向和位置。有时开发者可能会假设字段的位置，因此明确指定这些参数总是更清晰。这些操作只影响当前数据透视表，而不影响数据透视缓存。

```
定义 thisPivot 作为 数据透视表
定义 ptSheet 作为 工作表
定义 ptField 作为 数据透视字段

Set ptSheet = ThisWorkbook.Sheets("SheetNameWithPivotTable")
Set thisPivot = ptSheet.PivotTables(1)

With thisPivot
    Set ptField = .PivotFields("Gender")
    ptField.Orientation = xlRowField
    ptField.Position = 1
    Set ptField = .PivotFields("LastName")
    ptField.Orientation = xlRowField
    ptField.Position = 2
    Set ptField = .PivotFields("ShirtSize")
    ptField.Orientation = xlColumnField
    ptField.Position = 1
    Set ptField = .AddDataField(.PivotFields("Cost"), "Sum of Cost", xlSum)
    .InGridDropZones = True
    .RowAxisLayout xlTabularRow
End With
```

## 第17.2节：创建数据透视表

Excel中最强大的功能之一是使用数据透视表对数据进行排序和分析。如果你理解数据透视表与数据透视缓存之间的关系，以及如何引用和使用数据透视表的不同部分，使用VBA来创建和操作数据透视表会更容易。

在最基本的层面上，你的源数据是工作表上的一个Range数据区域。该数据区域必须以标题行作为范围中的第一行来标识数据列。数据透视表创建后，用户可以随时查看和更改源数据。然而，源数据的更改可能不会自动或立即反映在数据透视表中，因为存在一个称为数据透视缓存的中间数据存储结构，它直接连接到数据透视表本身。

# Chapter 17: Pivot Tables

## Section 17.1: Adding Fields to a Pivot Table

Two important things to note when adding fields to a Pivot Table are Orientation and Position. Sometimes a developer may assume where a field is placed, so it's always clearer to explicitly define these parameters. These actions only affect the given Pivot Table, not the Pivot Cache.

```
Dim thisPivot As PivotTable
Dim ptSheet As Worksheet
Dim ptField As PivotField

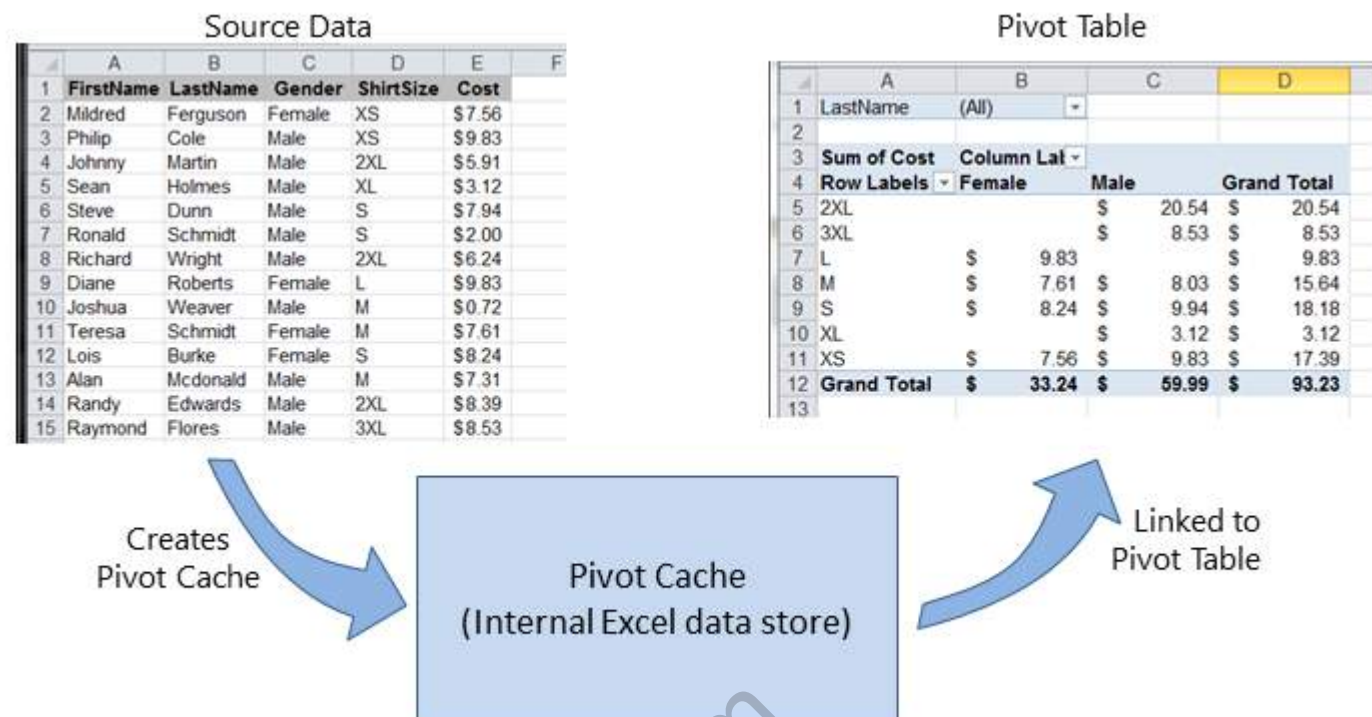
Set ptSheet = ThisWorkbook.Sheets("SheetNameWithPivotTable")
Set thisPivot = ptSheet.PivotTables(1)

With thisPivot
    Set ptField = .PivotFields("Gender")
    ptField.Orientation = xlRowField
    ptField.Position = 1
    Set ptField = .PivotFields("LastName")
    ptField.Orientation = xlRowField
    ptField.Position = 2
    Set ptField = .PivotFields("ShirtSize")
    ptField.Orientation = xlColumnField
    ptField.Position = 1
    Set ptField = .AddDataField(.PivotFields("Cost"), "Sum of Cost", xlSum)
    .InGridDropZones = True
    .RowAxisLayout xlTabularRow
End With
```

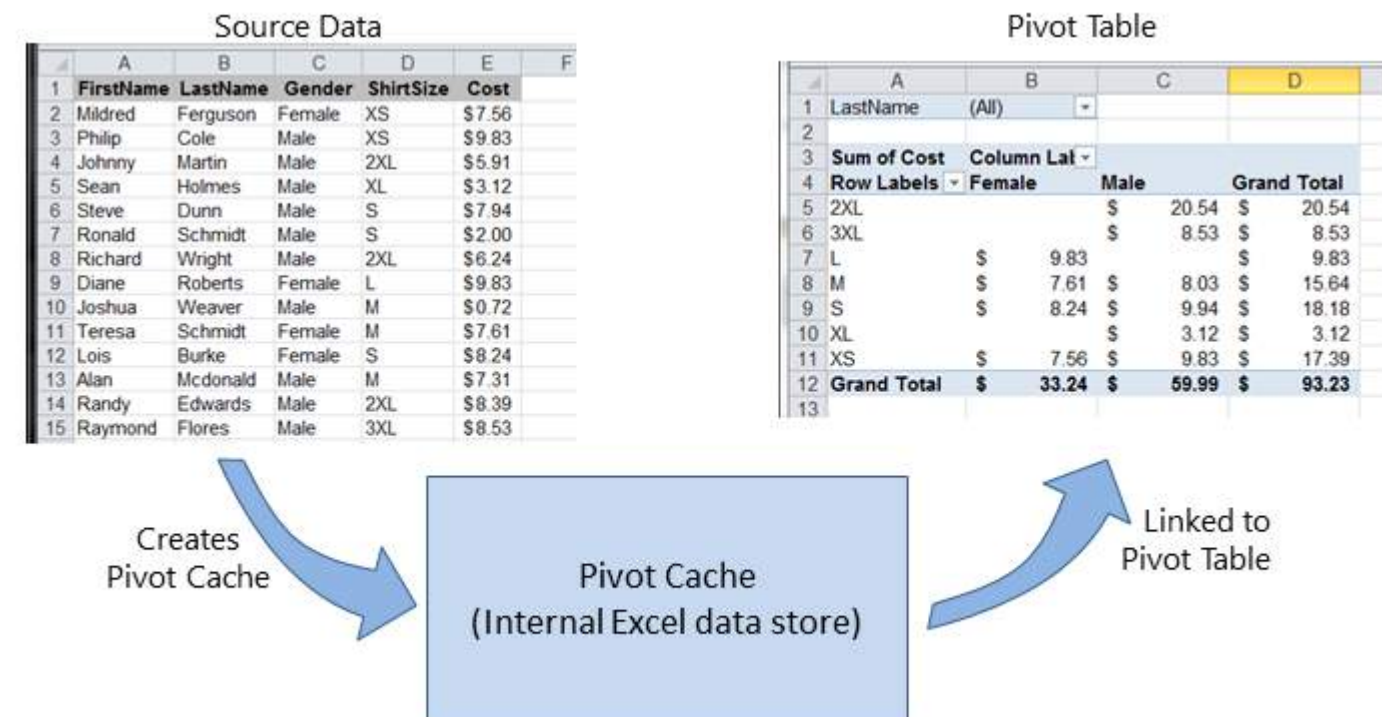
## Section 17.2: Creating a Pivot Table

One of the most powerful capabilities in Excel is the use of Pivot Tables to sort and analyze data. Using VBA to create and manipulate the Pivots is easier if you understand the relationship of Pivot Tables to Pivot Caches and how to reference and use the different parts of the Tables.

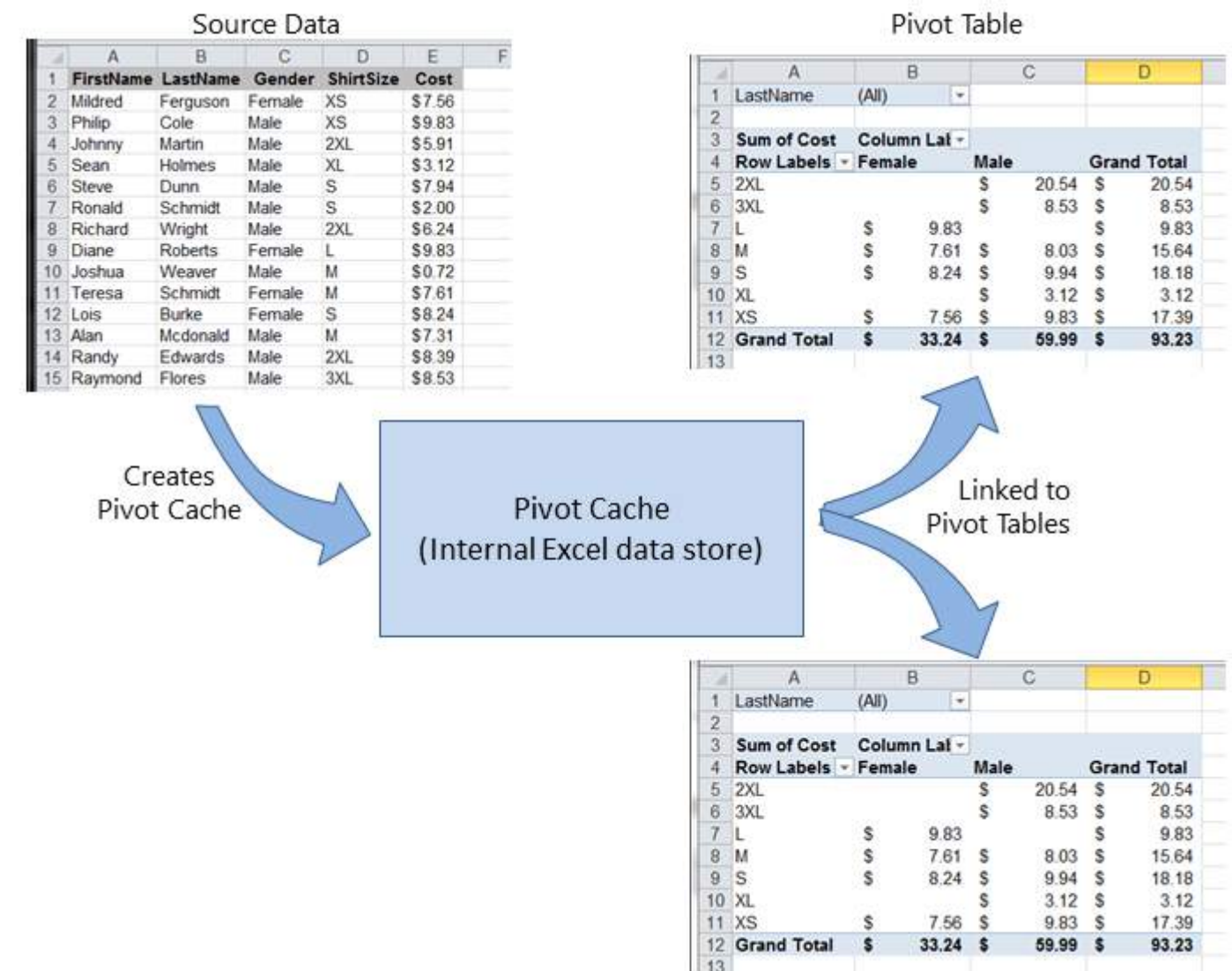
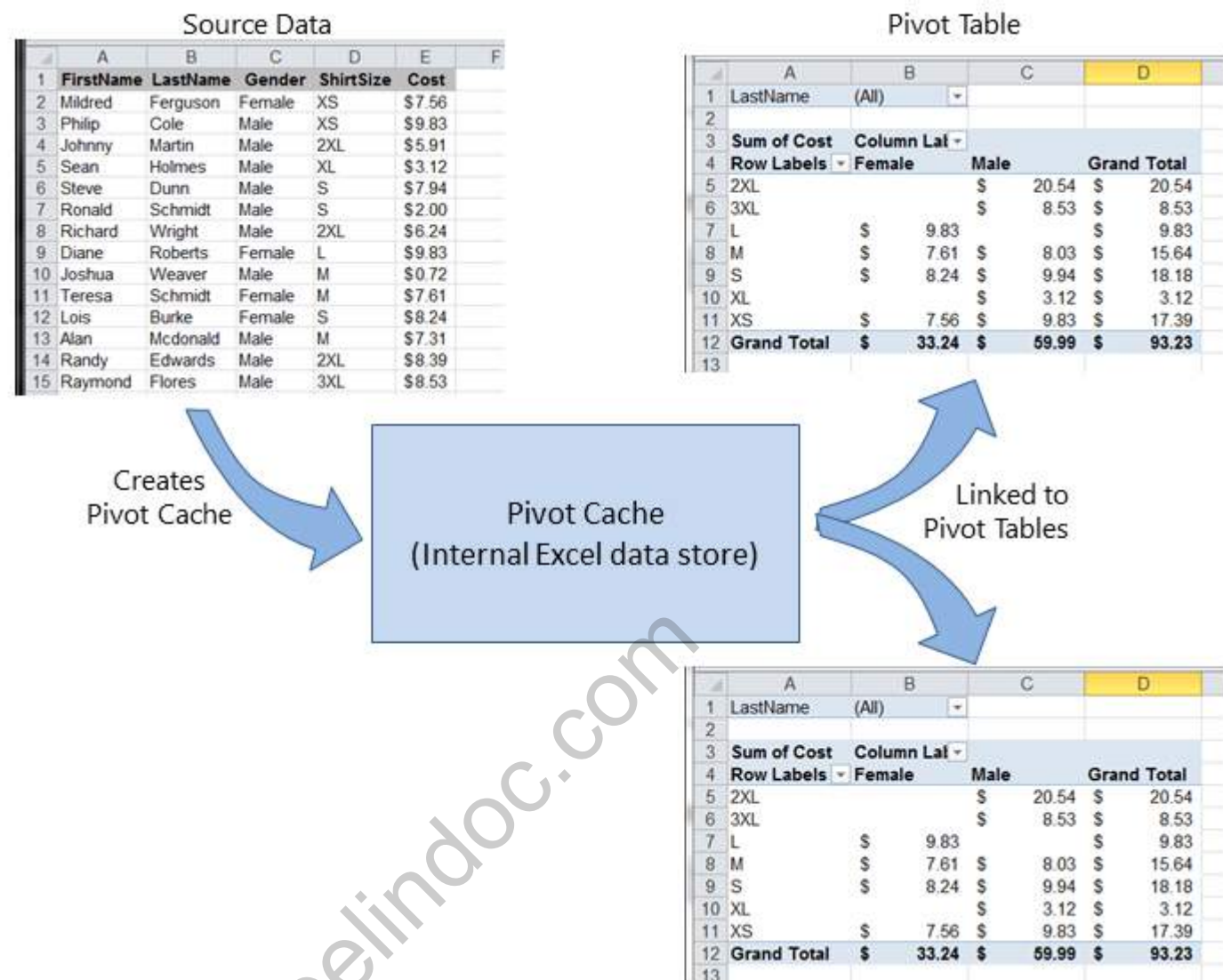
At its most basic, your source data is a Range area of data on a Worksheet. This data area **MUST** identify the data columns with a header row as the first row in the range. Once the Pivot Table is created, the user may view and change the source data at any time. However, changes may not be automatically or immediately reflected in the Pivot Table itself because there is an intermediate data storage structure called the Pivot Cache that is directly connected to the Pivot Table itself.



如果需要多个基于相同源数据的数据透视表，则可以重复使用数据透视缓存作为每个数据透视表的内部数据存储。这是一种良好的做法，因为它节省内存并减少Excel文件的存储大小。



If multiple Pivot Tables are needed, based on the same source data, the Pivot Cache may be re-used as the internal data store for each of the Pivot Tables. This is a good practice because it saves memory and reduces the size of the Excel file for storage.



例如，要基于上图所示的源数据创建数据透视表：

```
Sub test()
    Dim pt As 数据透视表
    Set pt = CreatePivotTable(ThisWorkbook.Sheets("Sheet1").Range("A1:E15"))
End Sub
```

*Function CreatePivotTable(ByRef srcData As Range) As 数据透视表'--- 从给定的源数据创建数据透视表，' 假设第一行包含有效的列标题数据*

```
Dim thisPivot As 数据透视表
Dim dataSheet As 工作表
Dim ptSheet As 工作表
Dim ptCache As 数据透视缓存
```

*'--- 必须先创建数据透视缓存...*

```
Set ptCache = ThisWorkbook.PivotCaches.Create(SourceType:=xlDatabase, _
    SourceData:=srcData)
```

*'--- ... 然后使用数据透视缓存创建数据透视表*

```
Set ptSheet = ThisWorkbook.Sheets.Add
Set thisPivot = ptCache.CreatePivotTable(TableDestination:=ptSheet.Range("A3"))
Set CreatePivotTable = thisPivot
```

```
End Function
```

As an example, to create a Pivot Table based on the source data shown in the Figures above:

```
Sub test()
    Dim pt As PivotTable
    Set pt = CreatePivotTable(ThisWorkbook.Sheets("Sheet1").Range("A1:E15"))
End Sub
```

*Function CreatePivotTable(ByRef srcData As Range) As PivotTable*

*'--- creates a Pivot Table from the given source data and  
' assumes that the first row contains valid header data  
' for the columns*

```
Dim thisPivot As PivotTable
Dim dataSheet As Worksheet
Dim ptSheet As Worksheet
Dim ptCache As PivotCache
```

*'--- the Pivot Cache must be created first...*

```
Set ptCache = ThisWorkbook.PivotCaches.Create(SourceType:=xlDatabase, _
    SourceData:=srcData)
```

*'--- ... then use the Pivot Cache to create the Table*

```
Set ptSheet = ThisWorkbook.Sheets.Add
Set thisPivot = ptCache.CreatePivotTable(TableDestination:=ptSheet.Range("A3"))
Set CreatePivotTable = thisPivot
```

```
End Function
```

## 第17.3节：透视表范围

这些优秀的参考资料提供了透视表中各种范围的描述和示例。

参考资料

- 在VBA中引用透视表范围 - 来自Jon Peltier的技术博客
- 使用VBA引用Excel透视表范围 - 来自globalconnect Excel VBA

## 第17.4节：格式化透视表数据

此示例更改/设置给定透视表的数据区域（DataBodyRange）中的多个格式。标准Range中的所有可格式化参数均可用。仅格式化数据会影响透视表本身，而不会影响透视缓存。

注意：该属性命名为TableStyle2，因为TableStyle属性不是PivotTable对象属性的成员。

```
定义 thisPivot 作为 数据透视表
定义 ptSheet 作为 工作表
定义 ptField 作为 数据透视字段

Set ptSheet = ThisWorkbook.Sheets("SheetNameWithPivotTable")
Set thisPivot = ptSheet.PivotTables(1)

With thisPivot
    .DataBodyRange.NumberFormat = "_($* #,##0.00_);_($* (#,##0.00);_($* "-"??_);_(@_)"
    .DataBodyRange.HorizontalAlignment = xlRight
    .ColumnRange.HorizontalAlignment = xlCenter
    .TableStyle2 = "PivotStyleMedium9"
End With
```

## Section 17.3: Pivot Table Ranges

These excellent reference sources provide descriptions and illustrations of the various ranges in Pivot Tables.

References

- Referencing Pivot Table Ranges in VBA - from Jon Peltier's Tech Blog
- Referencing an Excel Pivot Table Range using VBA - from globalconnect Excel VBA

## Section 17.4: Formatting the Pivot Table Data

This example changes/sets several formats in the data range area (DataBodyRange) of the given Pivot Table. All formattable parameters in a standard Range are available. Formatting the data only affects the Pivot Table itself, not the Pivot Cache.

NOTE: the property is named TableStyle2 because the TableStyle property is not a member of the PivotTable's object properties.

```
Dim thisPivot As PivotTable
Dim ptSheet As Worksheet
Dim ptField As PivotField

Set ptSheet = ThisWorkbook.Sheets("SheetNameWithPivotTable")
Set thisPivot = ptSheet.PivotTables(1)

With thisPivot
    .DataBodyRange.NumberFormat = "_($* #,##0.00_);_($* (#,##0.00);_($* "-"??_);_(@_)"
    .DataBodyRange.HorizontalAlignment = xlRight
    .ColumnRange.HorizontalAlignment = xlCenter
    .TableStyle2 = "PivotStyleMedium9"
End With
```



# 第18章：绑定

## 第18.1节：早期绑定与晚期绑定

绑定是将对象分配给标识符或变量名的过程。早期绑定（也称为静态绑定）是指在Excel中声明的对象具有特定的对象类型，例如工作表或工作簿。晚期绑定则是在进行一般对象关联时发生，例如Object和Variant声明类型。

早期绑定相较于晚期绑定具有一些优势。

- 早期绑定在运行时的操作速度比晚期绑定快。运行时使用晚期绑定创建对象需要时间，而早期绑定则在VBA项目初次加载时完成。
- 早期绑定通过按序号位置识别键/项对，提供了额外的功能。
- 根据代码结构，早期绑定可能提供额外的类型检查层次，减少错误。
- 在输入绑定对象的属性和方法时，VBE的大小写自动更正功能在早期绑定中有效，而晚期绑定中不可用。

注意： 您必须通过VBE的“工具 → 引用”命令为VBA项目添加相应的引用，以实现早期绑定。

该库引用随后会随项目一起携带；当VBA项目分发并在另一台计算机上运行时，无需重新引用。

```
'循环遍历使用后期绑定创建的字典'
子程序 iterateDictionaryLate()
    定义 k 作为 Variant, dict 作为 Object

    设置 dict = CreateObject("Scripting.Dictionary")
    dict.comparemode = vbTextCompare '不区分大小写的比较模式

    '填充字典
    dict.Add 键:="Red", 值:="Balloon"
    dict.Add 键:="Green", 值:="Balloon"
    dict.Add 键:="Blue", 值:="Balloon"

    '遍历键
    对于每个 k 在 dict.Keys中
        Debug.Print k & " - " & dict.Item(k)
    下一个 k

    dict.Remove "blue" '通过键移除单个键/值对
    dict.RemoveAll '移除所有剩余的键/值对

结束子程序

'遍历使用早期绑定创建的字典'
子程序 iterateDictionaryEarly()
    声明 d 为 长整型, k 为 变体
    声明 dict 为 新建 脚本.Dictionary

    dict.CompareMode = vbTextCompare '不区分大小写的比较模式

    '填充字典
    dict.Add Key:="Red", Item:="Balloon"
    dict.Add Key:="Green", Item:="Balloon"
```

# Chapter 18: Binding

## Section 18.1: Early Binding vs Late Binding

Binding is the process of assigning an object to an identifier or variable name. Early binding (also known as static binding) is when an object declared in Excel is of a specific object type, such as a Worksheet or Workbook. Late binding occurs when general object associations are made, such as the Object and Variant declaration types.

Early binding of references some advantages over late binding.

- Early binding is operationally faster than late binding during run-time. Creating the object with late binding in run-time takes time that early binding accomplishes when the VBA project is initially loaded.
- Early binding offers additional functionality through the identification of Key/Item pairs by their ordinal position.
- Depending on code structure, early binding may offer an additional level of type checking and reduce errors.
- The VBE's capitalization correction when typing a bound object's properties and methods is active with early binding but unavailable with late binding.

**Note:** You must add the appropriate reference to the VBA project through the VBE's Tools → References command in order to implement early binding. This library reference is then carried with the project; it does not have to be re-referenced when the VBA project is distributed and run on another computer.

```
'Looping through a dictionary that was created with late binding'
Sub iterateDictionaryLate()
    Dim k As Variant, dict As Object

    Set dict = CreateObject("Scripting.Dictionary")
    dict.comparemode = vbTextCompare 'non-case sensitive compare model

    'populate the dictionary
    dict.Add Key:="Red", Item:="Balloon"
    dict.Add Key:="Green", Item:="Balloon"
    dict.Add Key:="Blue", Item:="Balloon"

    'iterate through the keys
    For Each k In dict.Keys
        Debug.Print k & " - " & dict.Item(k)
    Next k

    dict.Remove "blue" 'remove individual key/item pair by key
    dict.RemoveAll 'remove all remaining key/item pairs

End Sub

'Looping through a dictionary that was created with early binding'
Sub iterateDictionaryEarly()
    Dim d As Long, k As Variant
    Dim dict As New Scripting.Dictionary

    dict.CompareMode = vbTextCompare 'non-case sensitive compare model

    'populate the dictionary
    dict.Add Key:="Red", Item:="Balloon"
    dict.Add Key:="Green", Item:="Balloon"
```

```
dict.Add Key:="Blue", Item:="Balloon"
dict.Add Key:="White", Item:="Balloon"

'遍历键
对于每个 k 在 dict.Keys中
    Debug.Print k & " - " & dict.Item(k)
下一个 k

'通过计数遍历键
对于 d = 0 到 dict.Count - 1
Debug.Print dict.Keys(d) & " - " & dict.Items(d)
    下一步 d

'通过键集合的边界遍历键
对于 d = LBound(dict.Keys) 到 UBound(dict.Keys)
    Debug.Print dict.Keys(d) & " - " & dict.Items(d)
    下一步 d

dict.Remove "blue"                '通过键移除单个键/项对
dict.Remove dict.Keys(0)          '通过索引位置移除第一个键/项
dict.Remove dict.Keys(UBound(dict.Keys)) '通过索引位置移除最后一个键/项
dict.RemoveAll                    '移除所有剩余的键/项对

End Sub
```

但是，如果您使用早期绑定，并且文档在缺少您所引用的某个库的系统上运行，您将遇到问题。不仅使用缺失库的例程无法正常工作，文档中所有代码的行为也会变得异常。很可能该计算机上文档的代码都无法运行。

这就是晚期绑定的优势所在。使用晚期绑定时，您无需在“工具>引用”菜单中添加引用。在拥有相应库的机器上，代码仍然可以正常工作。在没有该库的机器上，引用该库的命令将无法执行，但文档中的其他代码将继续正常运行。

如果您对所引用的库不够熟悉，编写代码时使用早期绑定可能会有帮助，然后在部署前切换到晚期绑定。这样，您可以在开发过程中利用VBE的智能感知(IntelliSense)和对象浏览器(Object Browser)的优势。

```
dict.Add Key:="Blue", Item:="Balloon"
dict.Add Key:="White", Item:="Balloon"

'iterate through the keys
For Each k In dict.Keys
    Debug.Print k & " - " & dict.Item(k)
Next k

'iterate through the keys by the count
For d = 0 To dict.Count - 1
    Debug.Print dict.Keys(d) & " - " & dict.Items(d)
Next d

'iterate through the keys by the boundaries of the keys collection
For d = LBound(dict.Keys) To UBound(dict.Keys)
    Debug.Print dict.Keys(d) & " - " & dict.Items(d)
Next d

dict.Remove "blue"                'remove individual key/item pair by key
dict.Remove dict.Keys(0)          'remove first key/item by index position
dict.Remove dict.Keys(UBound(dict.Keys)) 'remove last key/item by index position
dict.RemoveAll                    'remove all remaining key/item pairs

End Sub
```

However, if you are using early binding and the document is run on a system that lacks one of the libraries you have referenced, you will encounter problems. Not only will the routines that utilize the missing library not function properly, but the behavior of all code within the document will become erratic. It is likely that none of the document's code will function on that computer.

This is where late binding is advantageous. When using late binding you do not have to add the reference in the Tools>References menu. On machines that have the appropriate library, the code will still work. On machines without that library, the commands that reference the library will not work, but all the other code in your document will continue to function.

If you are not thoroughly familiar with the library you are referencing, it may be useful to use early binding while writing the code, then switch to late binding before deployment. That way you can take advantage of the VBE's IntelliSense and Object Browser during development.

# 第19章：自动筛选；用途和最佳实践

自动筛选的最终目标是以最快的方式从数百或数千行数据中挖掘数据，以便关注我们想要重点关注的项目。它可以接收诸如“文本/数值/颜色”等参数，并且可以在列之间叠加。您可以基于逻辑连接符和规则集为每列连接最多两个条件。备注：自动筛选通过筛选行来工作，没有用于筛选的自动筛选列（至少不是本地支持的）。

## 第19.1节：智能筛选器！

### 问题情境

仓库管理员有一张名为“记录”的表格，里面存储了设施执行的每一次物流操作，他可以根据需要进行筛选，尽管如此，这个过程非常耗时，他希望改进流程以更快地计算查询，例如：我们现在有多少“纸浆”（在所有货架中）？我们现在在第5号货架上有多少纸浆？筛选器是一个很好的工具，但它们在几秒钟内回答这类问题时有一定的局限性。

	A	B	C	D	E	F	G	H	I	J
1	Control Num	DESCRIPTION	QUANTITY	LOCATION	DATE	ACTION		1. How many "Pulp" do we have now? (Total)		1. How many "Pulp" do we have now? (In Rack #5)
2	9005124	Pulp	42	Rack #5	4-Oct-16	In				
15	9005137	Pulp	67	Rack #1	21-Nov-15	Out				
16	9005138	Pulp	92	Rack #3	19-Jun-15	Out				
42	9005164	Pulp	48	Rack #5	1-Dec-15	In				
45	9005167	Pulp	53	Rack #5	17-Mar-15	Out				
50	9005172	Pulp	13	Rack #3	5-Dec-15	In				
55	9005177	Pulp	30	Rack #2	15-Sep-16	In				
56	9005178	Pulp	90	Rack #3	27-Jan-16	Out				
68	9005190	Pulp	67	Rack #7	25-Aug-16	Out				
70	9005192	Pulp	62	Rack #6	7-Nov-15	Out				
71	9005193	Pulp	46	Rack #7	1-Dec-15	Out				
72	9005194	Pulp	6	Rack #2	18-Dec-16	Out				
83	9005205	Pulp	86	Rack #6	30-Mar-16	Out				
102	9005224	Pulp	78	Rack #3	7-Sep-16	Out				
109	9005231	Pulp	19	Rack #1	21-May-15	In				
115	9005237	Pulp	33	Rack #6	14-Jan-15	Out				
121	9005243	Pulp	46	Rack #1	25-Sep-15	Out				
124	9005246	Pulp	48	Rack #1	3-Jan-15	In				
125	9005247	Pulp	39	Rack #3	8-May-16	Out				
142	9005264	Pulp	68	Rack #1	15-Nov-15	In				
146	9005268	Pulp	50	Rack #2	30-Nov-16	In				
154	9005276	Pulp	11	Rack #4	8-Dec-15	In				
156	9005278	Pulp	40	Rack #1	5-Jun-16	In				
169	9005291	Pulp	84	Rack #4	21-Sep-16	Out				
174	9005296	Pulp	31	Rack #1	3-May-16	In				
182	9005304	Pulp	61	Rack #7	9-Apr-16	Out				
190	9005312	Pulp	57	Rack #1	2-Jul-15	Out				
192	9005314	Pulp	56	Rack #2	12-Feb-15	In				
200	9005322	Pulp	43	Rack #7	27-Sep-16	Out				
202	9005324	Pulp	97	Rack #1	16-Apr-16	In				
205	9005327	Pulp	80	Rack #6	8-Nov-16	In				
214	9005336	Pulp	82	Rack #5	27-Jul-15	In				
215	9005337	Pulp	27	Rack #4	17-Sep-16	In				
218	9005340	Pulp	51	Rack #3	16-Nov-15	Out				

### 宏解决方案：

编码者知道，自动筛选器是这类场景中最好的、快速且最可靠的解决方案，因为数据已经存在于工作表中，且输入可以轻松获得——在本例中，由用户输入。采用的方法是创建一个名为“智能筛选器”的工作表，管理员可以根据需要轻松筛选多条数据，计算也会即时完成。

他为此使用了两个模块和Worksheet\_Change事件

# Chapter 19: autofilter ; Uses and best practices

**Autofilter** ultimate goal is to provide in the quickest way possible data mining from hundreds or thousands of rows data in order to get the attention in the items we want to focus on. It can receive parameters such as "text/values/colors" and they can be stacked among columns. You may connect up to 2 criteria per column based in logical connectors and sets of rules. Remark: Autofilter works by filtering rows, there is no Autofilter to filter columns (at least not natively).

## Section 19.1: Smartfilter!

### Problem situation

Warehouse administrator has a sheet ("Record") where every logistics movement performed by the facility is stored, he may filter as needed, although, this is very time consuming and he would like to improve the process in order to calculate inquiries faster, for example: How many "pulp" do we have now (in all racks)? How many pulp do we have now (in rack #5)? Filters are a great tool but, they are somewhat limited to answer these kind of question in matter of seconds.

	A	B	C	D	E	F	G	H	I	J
1	Control Num	DESCRIPTION	QUANTITY	LOCATION	DATE	ACTION		1. How many "Pulp" do we have now? (Total)		1. How many "Pulp" do we have now? (In Rack #5)
2	9005124	Pulp	42	Rack #5	4-Oct-16	In				
15	9005137	Pulp	67	Rack #1	21-Nov-15	Out				
16	9005138	Pulp	92	Rack #3	19-Jun-15	Out				
42	9005164	Pulp	48	Rack #5	1-Dec-15	In				
45	9005167	Pulp	53	Rack #5	17-Mar-15	Out				
50	9005172	Pulp	13	Rack #3	5-Dec-15	In				
55	9005177	Pulp	30	Rack #2	15-Sep-16	In				
56	9005178	Pulp	90	Rack #3	27-Jan-16	Out				
68	9005190	Pulp	67	Rack #7	25-Aug-16	Out				
70	9005192	Pulp	62	Rack #6	7-Nov-15	Out				
71	9005193	Pulp	46	Rack #7	1-Dec-15	Out				
72	9005194	Pulp	6	Rack #2	18-Dec-16	Out				
83	9005205	Pulp	86	Rack #6	30-Mar-16	Out				
102	9005224	Pulp	78	Rack #3	7-Sep-16	Out				
109	9005231	Pulp	19	Rack #1	21-May-15	In				
115	9005237	Pulp	33	Rack #6	14-Jan-15	Out				
121	9005243	Pulp	46	Rack #1	25-Sep-15	Out				
124	9005246	Pulp	48	Rack #1	3-Jan-15	In				
125	9005247	Pulp	39	Rack #3	8-May-16	Out				
142	9005264	Pulp	68	Rack #1	15-Nov-15	In				
146	9005268	Pulp	50	Rack #2	30-Nov-16	In				
154	9005276	Pulp	11	Rack #4	8-Dec-15	In				
156	9005278	Pulp	40	Rack #1	5-Jun-16	In				
169	9005291	Pulp	84	Rack #4	21-Sep-16	Out				
174	9005296	Pulp	31	Rack #1	3-May-16	In				
182	9005304	Pulp	61	Rack #7	9-Apr-16	Out				
190	9005312	Pulp	57	Rack #1	2-Jul-15	Out				
192	9005314	Pulp	56	Rack #2	12-Feb-15	In				
200	9005322	Pulp	43	Rack #7	27-Sep-16	Out				
202	9005324	Pulp	97	Rack #1	16-Apr-16	In				
205	9005327	Pulp	80	Rack #6	8-Nov-16	In				
214	9005336	Pulp	82	Rack #5	27-Jul-15	In				
215	9005337	Pulp	27	Rack #4	17-Sep-16	In				
218	9005340	Pulp	51	Rack #3	16-Nov-15	Out				

### Macro solution:

The coder knows that **autofilters are the best, fast and most reliable solution** in these kind of scenarios since **the data exists already in the worksheet** and the **input for them can be obtained easily** -in this case, by user input-. The approach used is to create a sheet called "SmartFilter" where administrator can easily filter multiple data as needed and calculation will be performed instantly as well. He uses 2 modules and the Worksheet\_Change event for this matter

智能筛选器工作表的代码：

```
Private Sub Worksheet_Change(ByVal Target As Range)
Dim ItemInRange As Range
Const CellsFilters As String = "C2,E2,G2"
调用 ExcelBusy
For Each ItemInRange In Target
如果 Intersect(ItemInRange, Range(CellsFilters)) 不为 Nothing 则调用 Inventory_Filter
Next ItemInRange
调用 ExcelNormal
结束子程序
```

模块1的代码，名为“General\_Functions”

```
子程序 ExcelNormal()
使用 Excel.Application
.EnableEvents = True
.Cursor = xlDefault
.ScreenUpdating = True
.DisplayAlerts = True
.StatusBar = False
.CopyObjectsWithCells = True
结束使用
End Sub
子程序 ExcelBusy()
使用 Excel.Application
.EnableEvents = False
.Cursor = xlWait
.ScreenUpdating = False
.DisplayAlerts = False
.StatusBar = False
.CopyObjectsWithCells = True
结束使用
End Sub
Sub Select_Sheet(NameSheet As String, Optional VerifyExistanceOnly As Boolean)
On Error GoTo Err01Select_Sheet
Sheets(NameSheet).Visible = True
如果 VerifyExistanceOnly = False 则 ' 1. 如果 VerifyExistanceOnly = False
Sheets(NameSheet).Select
Sheets(NameSheet).AutoFilterMode = False
Sheets(NameSheet).Cells.EntireRow.Hidden = False
Sheets(NameSheet).Cells.EntireColumn.Hidden = False
结束 如果 ' 1. 如果 VerifyExistanceOnly = False
如果 1 = 2 则 '99. 如果出错
Err01Select_Sheet:
MsgBox "Err01Select_Sheet: 工作表 " & NameSheet & " 不存在!", vbCritical: 调用
ExcelNormal: On Error GoTo -1: 结束
结束 如果 '99. 如果出错
End Sub
函数 General_Functions_Find_Title(InSheet 作为 字符串, TitleToFind 作为 字符串, 可选 InRange 作为
范围, 可选 IsNeededToExist 作为 布尔值, 可选 IsWhole 作为 布尔值) 作为 范围
定义 DummyRange 作为 范围
出错转到 Err01General_Functions_Find_Title
如果 InRange 是空 则 ' 1. 如果 InRange 是空
设置 DummyRange = IIf(IsWhole = True, Sheets(InSheet).Cells.Find(TitleToFind, LookAt:=xlWhole),
Sheets(InSheet).Cells.Find(TitleToFind, LookAt:=xlPart))
否则 ' 1. 如果 InRange 是空
设置 DummyRange = IIf(IsWhole = True, Sheets(InSheet).Range(InRange.Address).Find(TitleToFind,
LookAt:=xlWhole), Sheets(InSheet).Range(InRange.Address).Find(TitleToFind, LookAt:=xlPart))
结束 If ' 1. 如果 InRange 为 Nothing
设置 General_Functions_Find_Title = DummyRange
```

Code For SmartFilter Worksheet:

```
Private Sub Worksheet_Change(ByVal Target As Range)
Dim ItemInRange As Range
Const CellsFilters As String = "C2,E2,G2"
Call ExcelBusy
For Each ItemInRange In Target
If Not Intersect(ItemInRange, Range(CellsFilters)) Is Nothing Then Call Inventory_Filter
Next ItemInRange
Call ExcelNormal
End Sub
```

Code for module 1, called "General\_Functions"

```
Sub ExcelNormal()
With Excel.Application
.EnableEvents = True
.Cursor = xlDefault
.ScreenUpdating = True
.DisplayAlerts = True
.StatusBar = False
.CopyObjectsWithCells = True
End With
End Sub
Sub ExcelBusy()
With Excel.Application
.EnableEvents = False
.Cursor = xlWait
.ScreenUpdating = False
.DisplayAlerts = False
.StatusBar = False
.CopyObjectsWithCells = True
End With
End Sub
Sub Select_Sheet(NameSheet As String, Optional VerifyExistanceOnly As Boolean)
On Error GoTo Err01Select_Sheet
Sheets(NameSheet).Visible = True
If VerifyExistanceOnly = False Then ' 1. If VerifyExistanceOnly = False
Sheets(NameSheet).Select
Sheets(NameSheet).AutoFilterMode = False
Sheets(NameSheet).Cells.EntireRow.Hidden = False
Sheets(NameSheet).Cells.EntireColumn.Hidden = False
End If ' 1. If VerifyExistanceOnly = False
If 1 = 2 Then '99. If error
Err01Select_Sheet:
MsgBox "Err01Select_Sheet: Sheet " & NameSheet & " doesn't exist!", vbCritical: Call
ExcelNormal: On Error GoTo -1: End
End If '99. If error
End Sub
Function General_Functions_Find_Title(InSheet As String, TitleToFind As String, Optional InRange As
Range, Optional IsNeededToExist As Boolean, Optional IsWhole As Boolean) As Range
Dim DummyRange As Range
On Error GoTo Err01General_Functions_Find_Title
If InRange Is Nothing Then ' 1. If InRange Is Nothing
Set DummyRange = IIf(IsWhole = True, Sheets(InSheet).Cells.Find(TitleToFind, LookAt:=xlWhole),
Sheets(InSheet).Cells.Find(TitleToFind, LookAt:=xlPart))
Else ' 1. If InRange Is Nothing
Set DummyRange = IIf(IsWhole = True, Sheets(InSheet).Range(InRange.Address).Find(TitleToFind,
LookAt:=xlWhole), Sheets(InSheet).Range(InRange.Address).Find(TitleToFind, LookAt:=xlPart))
End If ' 1. If InRange Is Nothing
Set General_Functions_Find_Title = DummyRange
```



```

    如果 1 = 2 或 DummyRange 为 Nothing 则 '99. 如果出错
Err01General_Functions_Find_Title:
    如果 IsNeededToExist = True 则 MsgBox "Err01General_Functions_Find_Title: 标题 '" &
TitleToFind & "' 未在工作表 '" & InSheet & "' 中找到", vbCritical: 调用 ExcelNormal: 出错时
转到 -1: 结束
    结束 If '99. 如果出错
结束函数

```

模块2的代码，名为“Inventory\_Handling”（库存处理）

```

常量 TitleDesc 作为 字符串 = "DESCRIPTION"
常量 TitleLocation 作为 字符串 = "LOCATION"
常量 TitleActn 作为 字符串 = "ACTION"
常量 TitleQty 作为 字符串 = "QUANTITY"
常量 SheetRecords 作为 字符串 = "Record"
常量 SheetSmartFilter 作为 字符串 = "SmartFilter"
常量 RowFilter 作为 长整型 = 2
常量 ColDataToPaste 作为 长整型 = 2
常量 RowDataToPaste 作为 长整型 = 7
常量 RangeInResult 作为 字符串 = "K1"
常量 RangeOutResult 作为 字符串 = "K2"
子程序 Inventory_Filter()
Dim ColDesc As Long: ColDesc = General_Functions_Find_Title(SheetSmartFilter, TitleDesc,
IsNeededToExist:=True, IsWhole:=True).Column
Dim ColLocation As Long: ColLocation = General_Functions_Find_Title(SheetSmartFilter,
TitleLocation, IsNeededToExist:=True, IsWhole:=True).Column
Dim ColActn As Long: ColActn = General_Functions_Find_Title(SheetSmartFilter, TitleActn,
IsNeededToExist:=True, IsWhole:=True).Column
Dim ColQty As Long: ColQty = General_Functions_Find_Title(SheetSmartFilter, TitleQty,
IsNeededToExist:=True, IsWhole:=True).Column
Dim CounterQty As Long
Dim TotalQty As Long
Dim TotalIn As Long
Dim TotalOut As Long
Dim RangeFiltered As Range
Call Select_Sheet(SheetSmartFilter)
If Cells(Rows.Count, ColDataToPaste).End(xlUp).Row > RowDataToPaste - 1 Then
Rows(RowDataToPaste & ":" & Cells(Rows.Count, "B").End(xlUp).Row).Delete
Sheets(SheetRecords).AutoFilterMode = False
    如果 Cells(RowFilter, ColDesc).Value <> "" 或者 Cells(RowFilter, ColLocation).Value <> "" 或者
Cells(RowFilter, ColActn).Value <> "" 那么 ' 1. 如果 Cells(RowFilter, ColDesc).Value <> "" 或者
Cells(RowFilter, ColLocation).Value <> "" 或者 Cells(RowFilter, ColActn).Value <> ""
    使用 Sheets(SheetRecords).UsedRange
    如果 Sheets(SheetSmartFilter).Cells(RowFilter, ColDesc).Value <> "" 那么 .AutoFilter
Field:=General_Functions_Find_Title(SheetRecords, TitleDesc, IsNeededToExist:=True,
IsWhole:=True).Column, Criteria1:=Sheets(SheetSmartFilter).Cells(RowFilter, ColDesc).Value
    如果 Sheets(SheetSmartFilter).Cells(RowFilter, ColLocation).Value <> "" 那么 .AutoFilter
Field:=General_Functions_Find_Title(SheetRecords, TitleLocation, IsNeededToExist:=True,
IsWhole:=True).Column, Criteria1:=Sheets(SheetSmartFilter).Cells(RowFilter, ColLocation).Value
    如果 Sheets(SheetSmartFilter).Cells(RowFilter, ColActn).Value <> "" 那么 .AutoFilter
Field:=General_Functions_Find_Title(SheetRecords, TitleActn, IsNeededToExist:=True,
IsWhole:=True).Column, Criteria1:=Sheets(SheetSmartFilter).Cells(RowFilter, ColActn).Value
    '如果我们不使用筛选器，我们需要使用 For/to 循环或 For/Each Cell in range
    '来判断该行是否符合我们查找的条件，然后
    '将其保存到数组、集合、字典等中
    '例如：For CounterRow = 2 到 TotalRows
    '如果 Sheets(SheetSmartFilter).Cells(RowFilter, ColDesc).Value <> "" 并且
Sheets(SheetRecords).Cells(CounterRow, ColDescInRecords).Value =
Sheets(SheetSmartFilter).Cells(RowFilter, ColDesc).Value 那么
    'Redim Preserve MyUnecessaryArray(UnecessaryNumber) '保存到数组：
(UnecessaryNumber)=MyUnecessaryArray. 或者保存到字典等。最后，我们会转置这个

```

```

If 1 = 2 Or DummyRange Is Nothing Then '99. If error
Err01General_Functions_Find_Title:
    If IsNeededToExist = True Then MsgBox "Err01General_Functions_Find_Title: Ttile '" &
TitleToFind & "' was not found in sheet '" & InSheet & "'", vbCritical: Call ExcelNormal: On Error
GoTo -1: End
    End If '99. If error
End Function

```

Code for module 2, called "Inventory\_Handling"

```

Const TitleDesc As String = "DESCRIPTION"
Const TitleLocation As String = "LOCATION"
Const TitleActn As String = "ACTION"
Const TitleQty As String = "QUANTITY"
Const SheetRecords As String = "Record"
Const SheetSmartFilter As String = "SmartFilter"
Const RowFilter As Long = 2
Const ColDataToPaste As Long = 2
Const RowDataToPaste As Long = 7
Const RangeInResult As String = "K1"
Const RangeOutResult As String = "K2"
Sub Inventory_Filter()
Dim ColDesc As Long: ColDesc = General_Functions_Find_Title(SheetSmartFilter, TitleDesc,
IsNeededToExist:=True, IsWhole:=True).Column
Dim ColLocation As Long: ColLocation = General_Functions_Find_Title(SheetSmartFilter,
TitleLocation, IsNeededToExist:=True, IsWhole:=True).Column
Dim ColActn As Long: ColActn = General_Functions_Find_Title(SheetSmartFilter, TitleActn,
IsNeededToExist:=True, IsWhole:=True).Column
Dim ColQty As Long: ColQty = General_Functions_Find_Title(SheetSmartFilter, TitleQty,
IsNeededToExist:=True, IsWhole:=True).Column
Dim CounterQty As Long
Dim TotalQty As Long
Dim TotalIn As Long
Dim TotalOut As Long
Dim RangeFiltered As Range
Call Select_Sheet(SheetSmartFilter)
If Cells(Rows.Count, ColDataToPaste).End(xlUp).Row > RowDataToPaste - 1 Then
Rows(RowDataToPaste & ":" & Cells(Rows.Count, "B").End(xlUp).Row).Delete
Sheets(SheetRecords).AutoFilterMode = False
    If Cells(RowFilter, ColDesc).Value <> "" Or Cells(RowFilter, ColLocation).Value <> "" Or
Cells(RowFilter, ColActn).Value <> "" Then ' 1. If Cells(RowFilter, ColDesc).Value <> "" Or
Cells(RowFilter, ColLocation).Value <> "" Or Cells(RowFilter, ColActn).Value <> ""
    With Sheets(SheetRecords).UsedRange
    If Sheets(SheetSmartFilter).Cells(RowFilter, ColDesc).Value <> "" Then .AutoFilter
Field:=General_Functions_Find_Title(SheetRecords, TitleDesc, IsNeededToExist:=True,
IsWhole:=True).Column, Criteria1:=Sheets(SheetSmartFilter).Cells(RowFilter, ColDesc).Value
    If Sheets(SheetSmartFilter).Cells(RowFilter, ColLocation).Value <> "" Then .AutoFilter
Field:=General_Functions_Find_Title(SheetRecords, TitleLocation, IsNeededToExist:=True,
IsWhole:=True).Column, Criteria1:=Sheets(SheetSmartFilter).Cells(RowFilter, ColLocation).Value
    If Sheets(SheetSmartFilter).Cells(RowFilter, ColActn).Value <> "" Then .AutoFilter
Field:=General_Functions_Find_Title(SheetRecords, TitleActn, IsNeededToExist:=True,
IsWhole:=True).Column, Criteria1:=Sheets(SheetSmartFilter).Cells(RowFilter, ColActn).Value
    'If we don't use a filter we would need to use a cycle For/to or For/Each Cell in range
    'to determine whether or not the row meets the criteria that we are looking and then
    'save it on an array, collection, dictionary, etc
    'IG: For CounterRow = 2 To TotalRows
    'If Sheets(SheetSmartFilter).Cells(RowFilter, ColDesc).Value <> "" and
Sheets(SheetRecords).cells(CounterRow, ColDescInRecords).Value=
Sheets(SheetSmartFilter).Cells(RowFilter, ColDesc).Value then
    'Redim Preserve MyUnecessaryArray(UnecessaryNumber) ''Save to array:
(UnecessaryNumber)=MyUnecessaryArray. Or in a dictionary, etc. At the end, we would transpose this

```



将数值输入表格，最后

“两者是相同的，但只是试着看看每个逻辑投入的时间。”

如果.Cells(1, 1).End(xlDown).Value <> "" 那么 设置 RangeFiltered = .Rows("2:" & Sheets(SheetRecords).Cells(Rows.Count, "A").End(xlUp).Row).SpecialCells(xlCellTypeVisible)

“如果不是 <>""，意味着没有过滤到数据！”

如果 RangeFiltered 是 Nothing 那么 MsgBox "Err01Inventory\_Filter: 根据给定条件未找到数据！", vbCritical: 调用 ExcelNormal: 结束 RangeFiltered.Copy 目标:=Cells(RowData ToPaste, ColDataToPaste)

TotalQty = Cells(Rows.Count, ColQty).End(xlUp).Row

对于 CounterQty = RowDataToPaste + 1 到 TotalQty

如果 Cells(CounterQty, ColActn).Value = "In" 那么 ' 2. 如果 Cells(CounterQty, ColActn).Value = "In"

TotalIn = Cells(CounterQty, ColQty).Value + TotalIn

否则如果 Cells(CounterQty, ColActn).Value = "Out" 那么 ' 2. 如果 Cells(CounterQty, ColActn).Value =

"In"

TotalOut = Cells(CounterQty, ColQty).Value + TotalOut

结束 如果 ' 2. 如果 Cells(CounterQty, ColActn).Value = "In"

下一个 CounterQty

Range(RangeInResult).Value = TotalIn

Range(RangeOutResult).Value = -(TotalOut)

End With

结束 如果 ' 1. 如果 Cells(RowFilter, ColDesc).Value <> "" 或 Cells(RowFilter, ColLocation).Value <> "" 或 Cells(RowFilter, ColActn).Value <> ""

End Sub

测试和结果：

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	AA	AB
912	9013034	Batch weight	21	Rack #1	9-Jun-16	Out																						
913	9013035	Pectin	72	Rack #7	22-Jun-16	In																						
914	9013036	Sugar	28	Rack #1	5-Aug-15	In																						
915	9013037	Solids content	51	Rack #7	11-Sep-16	In																						
916	9013038	Pulp	45	Rack #3	9-Apr-16	Out																						
917	9013039	Batch weight	19	Rack #4	6-Apr-15	Out																						
918	9013040	Citric Acid	98	Rack #4	17-Jun-16	Out																						
919	9013041	Citric Acid	97	Rack #1	29-Feb-16	In																						
920	9013042	Pulp	57	Rack #5	25-Nov-16	Out																						
921	9013043	Citric Acid	42	Rack #2	27-Feb-16	In																						
922	9013044	Batch weight	54	Rack #1	16-Sep-15	Out																						
923	9013045	Solids content	12	Rack #4	13-Jul-15	In																						
924	9013046	Pulp	79	Rack #4	13-Jul-15	Out																						
925	9013047	Citric Acid	36	Rack #4	15-Nov-16	Out																						
926	9013048	Sugar	35	Rack #3	5-Feb-16	Out																						
927	9013049	Pulp	63	Rack #6	16-Dec-16	Out																						
928	9013050	Solids content	48	Rack #4	1-Mar-15	In																						
929	9013051	Pulp	39	Rack #4	31-May-16	Out																						
930	9013052	Pulp	47	Rack #6	26-Feb-16	In																						
931	9013053	Sugar	6	Rack #6	3-Mar-16	Out																						
932	9013054	Pulp	53	Rack #2	11-Sep-15	Out																						
933	9013055	Solids content	87	Rack #4	19-Jan-15	Out																						
934	9013056	Sugar	48	Rack #7	23-Nov-16	In																						
935	9013057	Solids content	62	Rack #6	15-May-16	Out																						
936	9013058	Batch weight	61	Rack #3	3-Dec-16	Out																						
937	9013059	Citric Acid	64	Rack #7	7-Feb-16	Out																						
938	9013060	Sugar	91	Rack #7	23-Sep-15	Out																						
939	9013061	Citric Acid	29	Rack #1	7-Jul-16	Out																						
940	9013062	Citric Acid	31	Rack #6	17-Feb-16	In																						
941	9013063	Batch weight	53	Rack #1	5-Apr-15	Out																						
942	9013064	Citric Acid	25	Rack #6	30-Jul-15	Out																						
943	9013065	Citric Acid	68	Rack #4	22-Mar-16	Out																						
944	9013066	Boiling time	22	Rack #6	17-Jun-15	In																						
945	9013067	Pectin	99	Rack #2	2-Nov-16	Out																						
946	9013068	Solids content	79	Rack #2	17-Nov-16	Out																						

正如我们在前一张图片中看到的，这个任务已经轻松完成。通过使用autofilters，提供了一个解决方案只需几秒钟即可计算完成，且易于向用户解释——因为用户熟悉此命令——并且对编码者来说只需几行代码。

values into the sheet, at the end

'both are the same, but, just try to see the time invested on each logic.

If .Cells(1, 1).End(xlDown).Value <> "" Then Set RangeFiltered = .Rows("2:" & Sheets(SheetRecords).Cells(Rows.Count, "A").End(xlUp).Row).SpecialCells(xlCellTypeVisible)

'If it is not <>"" means that there was not filtered data!

If RangeFiltered Is Nothing Then MsgBox "Err01Inventory\_Filter: No data was found with the given criteria!", vbCritical: Call ExcelNormal: End

RangeFiltered.Copy Destination:=Cells(RowDataToPaste, ColDataToPaste)

TotalQty = Cells(Rows.Count, ColQty).End(xlUp).Row

For CounterQty = RowDataToPaste + 1 To TotalQty

If Cells(CounterQty, ColActn).Value = "In" Then ' 2. If Cells(CounterQty, ColActn).Value = "In"

TotalIn = Cells(CounterQty, ColQty).Value + TotalIn

ElseIf Cells(CounterQty, ColActn).Value = "Out" Then ' 2. If Cells(CounterQty, ColActn).Value =

"In"

TotalOut = Cells(CounterQty, ColQty).Value + TotalOut

End If ' 2. If Cells(CounterQty, ColActn).Value = "In"

Next CounterQty

Range(RangeInResult).Value = TotalIn

Range(RangeOutResult).Value = -(TotalOut)

End With

End If ' 1. If Cells(RowFilter, ColDesc).Value <> "" Or Cells(RowFilter, ColLocation).Value <> "" Or Cells(RowFilter, ColActn).Value <> ""

End Sub

Testing and results:

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	AA	AB
912	9013034	Batch weight	21	Rack #1	9-Jun-16	Out																						
913	9013035	Pectin	72	Rack #7	22-Jun-16	In																						
914	9013036	Sugar	28	Rack #1	5-Aug-15	In																						
915	9013037	Solids content	51	Rack #7	11-Sep-16	In																						
916	9013038	Pulp	45	Rack #3	9-Apr-16	Out																						
917	9013039	Batch weight	19	Rack #4	6-Apr-15	Out																						
918	9013040	Citric Acid	98	Rack #4	17-Jun-16	Out																						
919	9013041	Citric Acid	97	Rack #1	29-Feb-16	In																						
920	9013042	Pulp	57	Rack #5	25-Nov-16	Out																						
921	9013043	Citric Acid	42	Rack #2	27-Feb-16	In																						
922	9013044	Batch weight	54	Rack #1	16-Sep-15	Out																						
923	9013045	Solids content	12	Rack #4	13-Jul-15	In																						
924	9013046	Pulp	79	Rack #4	13-Jul-15	Out																						
925	9013047	Citric Acid	36	Rack #4	15-Nov-16	Out																						
926	9013048	Sugar	35	Rack #3	5-Feb-16	Out																						
927	9013049	Pulp	63	Rack #6	16-Dec-16	Out																						
928	9013050	Solids content	48	Rack #4	1-Mar-15	In																						
929	9013051	Pulp	39	Rack #4	31-May-16	Out																						
930	9013052	Pulp	47	Rack #6	26-Feb-16	In																						
931	9013053	Sugar	6	Rack #6	3-Mar-16	Out																						
932	9013054	Pulp	53	Rack #2	11-Sep-15	Out																						
933	9013055	Solids content	87	Rack #4	19-Jan-15	Out																						
934	9013056	Sugar	48	Rack #7	23-Nov-16	In																						
935	9013057	Solids content	62	Rack #6	15-May-16	Out																						
936	9013058	Batch weight	61	Rack #3	3-Dec-16	Out																						
937	9013059	Citric Acid	64	Rack #7	7-Feb-16	Out																						
938	9013060	Sugar	91	Rack #7	23-Sep-15	Out																						
939	9013061	Citric Acid	29	Rack #1	7-Jul-16	Out																						
940	9013062	Citric Acid	31	Rack #6	17-Feb-16	In																						
941	9013063	Batch weight	53	Rack #1	5-Apr-15	Out																						
942	9013064	Citric Acid	25	Rack #6	30-Jul-15	Out																						
943	9013065	Citric Acid	68	Rack #4	22-Mar-16	Out																						
944	9013066	Boiling time	22	Rack #6	17-Jun-15	In																						
945	9013067	Pectin	99	Rack #2	2-Nov-16	Out																						
946	9013068	Solids content	79	Rack #2	17-Nov-16	Out																						
	SmartFilter	Record																										

# 第20章：应用程序对象

## 第20.1节：简单的应用程序对象示例：显示Excel和VBE版本

```
子程序 DisplayExcelVersions()  
  
    MsgBox "Excel的版本是 " & Application.Version  
    MsgBox "VBE的版本是 " & Application.VBE.Version  
  
End Sub
```

使用Application.Version属性有助于确保代码仅在兼容的

Excel版本上运行。

## 第20.2节：简单的应用程序对象示例：最小化Excel窗口

此代码使用顶层Application对象来最小化主Excel窗口。

```
子程序 MinimizeExcel()  
  
    Application.WindowState = xlMinimized  
  
End Sub
```

# Chapter 20: Application object

## Section 20.1: Simple Application Object example: Display Excel and VBE Version

```
Sub DisplayExcelVersions()  
  
    MsgBox "The version of Excel is " & Application.Version  
    MsgBox "The version of the VBE is " & Application.VBE.Version  
  
End Sub
```

The use of the Application.Version property is useful for ensuring code only operates on a compatible version of Excel.

## Section 20.2: Simple Application Object example: Minimize the Excel window

This code uses the top level **Application** object to minimize the main Excel window.

```
Sub MinimizeExcel()  
  
    Application.WindowState = xlMinimized  
  
End Sub
```

# 第21章：图表与制图

## 第21.1节：使用范围和固定名称创建图表

图表可以通过直接操作定义图表数据的Series对象来创建。为了获取没有现有图表的Series，你需要在指定的Worksheet上创建一个ChartObject，然后从中获取Chart对象。使用Series对象的优点是可以引用Range对象来设置Values和XValues。这些数据属性将通过对这些范围的引用正确地定义Series。该方法的缺点是设置Name时不会进行相同的转换；它是一个固定值，不会随着原始Range中基础数据的变化而调整。查看SERIES公式可以明显看出名称是固定的。必须通过直接创建SERIES公式来处理这一点。

### 用于创建图表的代码

请注意，此代码包含了Chart和Worksheet的额外变量声明。如果不使用，可以省略它们。但如果你要修改样式或其他图表属性，它们会很有用。

```
Sub CreateChartWithRangesAndFixedName()  
  
    Dim xData As Range  
    Dim yData As Range  
    Dim serName As Range  
  
    '设置范围以获取数据和y值标签  
    Set xData = Range("B3:B12")  
    设置 yData = Range("C3:C12")  
    设置 serName = Range("C2")  
  
    '获取对活动工作表的引用  
    Dim sht 作为 工作表  
    设置 sht = ActiveSheet  
  
    '在位置 (48, 195) 创建一个新的 ChartObject, 宽度400, 高度300  
    Dim chtObj 作为 图表对象  
    设置 chtObj = sht.ChartObjects.Add(48, 195, 400, 300)  
  
    '获取对图表对象的引用  
    Dim cht 作为 图表  
    设置 cht = chtObj.Chart  
  
    '创建新的系列  
    Dim ser 作为 系列  
    设置 ser = cht.SeriesCollection.NewSeries  
  
    ser.Values = yData  
    ser.XValues = xData  
    ser.Name = serName  
  
    ser.ChartType = xlXYScatterLines  
  
End Sub
```

### 代码运行后原始数据/范围及生成的图表

请注意，SERIES 公式包含一个用于系列名称的 "B", 而不是对创建该系列的 Range 的引用。

# Chapter 21: Charts and Charting

## Section 21.1: Creating a Chart with Ranges and a Fixed Name

Charts can be created by working directly with the Series object that defines the chart data. In order to get to the Series without an existing chart, you create a ChartObject on a given Worksheet and then get the Chart object from it. The upside of working with the Series object is that you can set the Values and XValues by referring to Range objects. These data properties will properly define the Series with references to those ranges. The downside to this approach is that the same conversion is not handled when setting the Name; it is a fixed value. It will not adjust with the underlying data in the original Range. Checking the SERIES formula and it is obvious that the name is fixed. This must be handled by creating the SERIES formula directly.

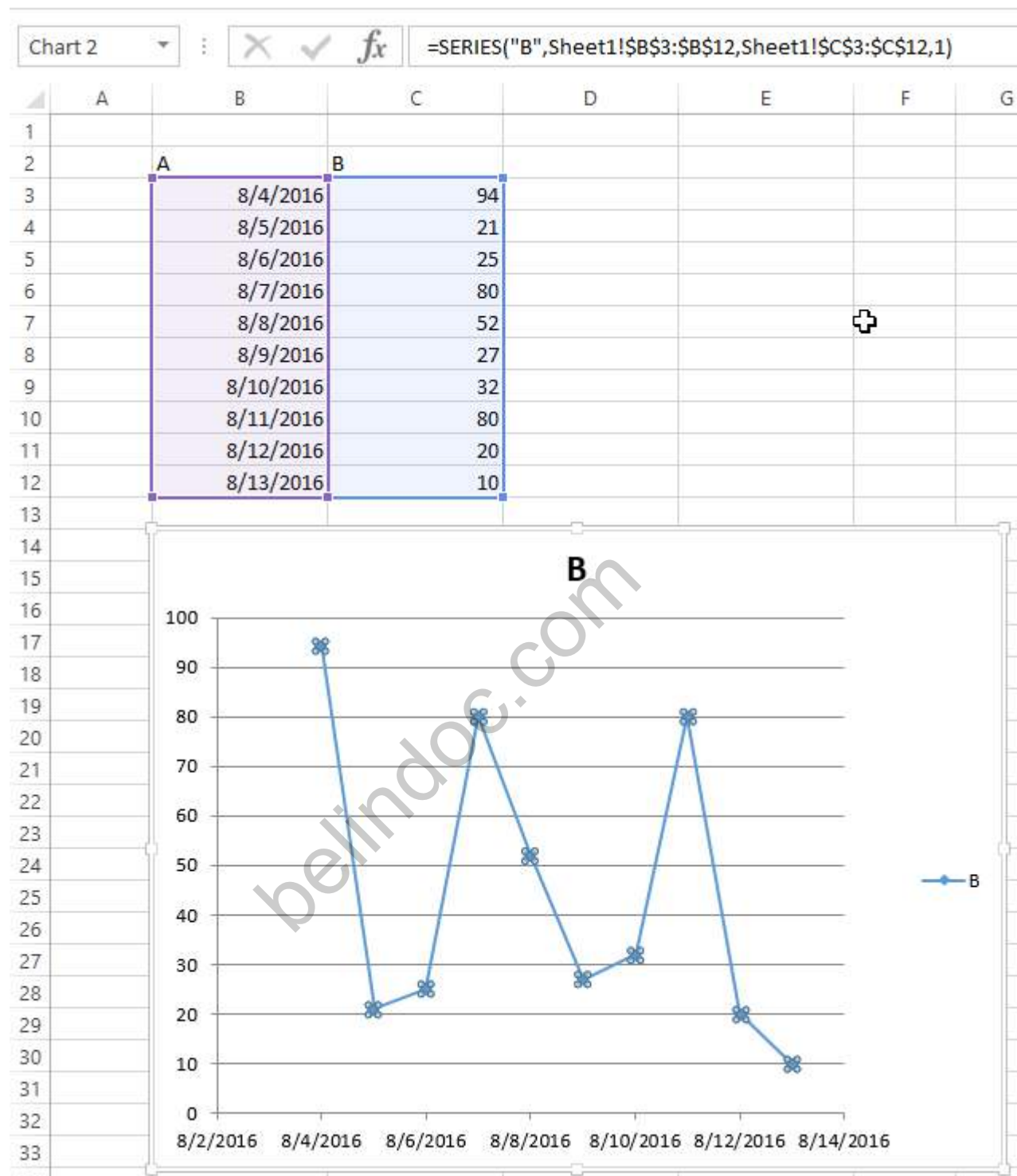
### Code used to create chart

Note that this code contains extra variable declarations for the Chart and Worksheet. These can be omitted if they're not used. They can be useful however if you are modifying the style or any other chart properties.

```
Sub CreateChartWithRangesAndFixedName()  
  
    Dim xData As Range  
    Dim yData As Range  
    Dim serName As Range  
  
    'set the ranges to get the data and y value label  
    Set xData = Range("B3:B12")  
    Set yData = Range("C3:C12")  
    Set serName = Range("C2")  
  
    'get reference to ActiveSheet  
    Dim sht As Worksheet  
    Set sht = ActiveSheet  
  
    'create a new ChartObject at position (48, 195) with width 400 and height 300  
    Dim chtObj As ChartObject  
    Set chtObj = sht.ChartObjects.Add(48, 195, 400, 300)  
  
    'get reference to chart object  
    Dim cht As Chart  
    Set cht = chtObj.Chart  
  
    'create the new series  
    Dim ser As Series  
    Set ser = cht.SeriesCollection.NewSeries  
  
    ser.Values = yData  
    ser.XValues = xData  
    ser.Name = serName  
  
    ser.ChartType = xlXYScatterLines  
  
End Sub
```

### Original data/ranges and resulting Chart after code runs

Note that the SERIES formula includes a "B" for the series name instead of a reference to the Range that created it.



## 第21.2节：创建一个空图表

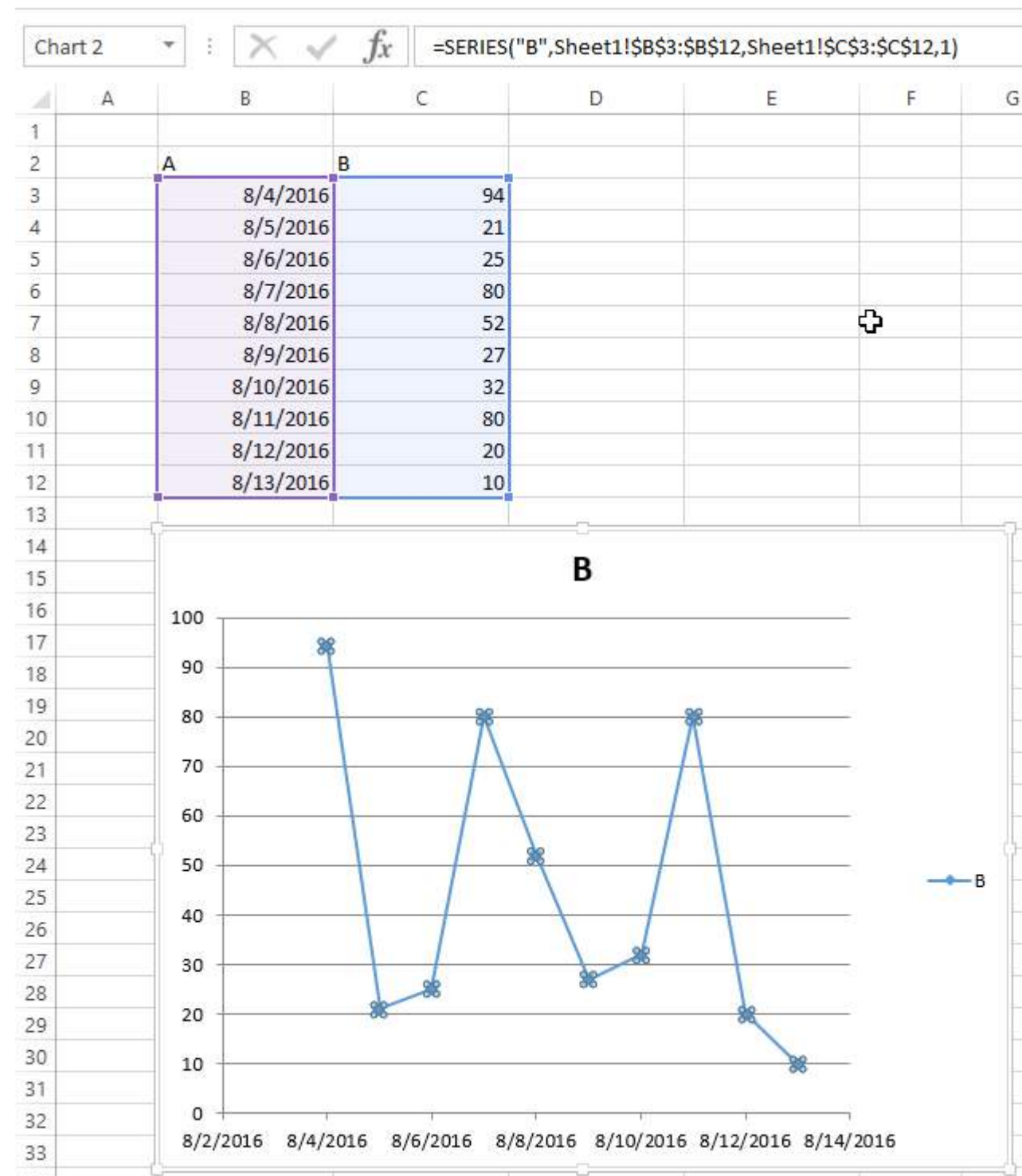
绝大多数图表代码的起点是创建一个空的图表。请注意，这个图表受当前活动的默认图表模板影响，可能实际上并非空白（如果模板已被修改）。

ChartObject的关键是确定其位置。调用语法为ChartObjects.Add(Left, Top,Width, Height)。一旦创建了ChartObject，就可以使用其Chart对象来实际修改图表。ChartObject更像是一个Shape，用于在工作表上定位图表。

### 创建空图表的代码

子程序CreateEmptyChart()

'获取对活动工作表的引用



## Section 21.2: Creating an empty Chart

The starting point for the vast majority of charting code is to create an empty Chart. Note that this Chart is subject to the default chart template that is active and may not actually be empty (if the template has been modified).

The key to the ChartObject is determining its location. The syntax for the call is ChartObjects.Add(Left, Top, Width, Height). Once the ChartObject is created, you can use its Chart object to actually modify the chart. The ChartObject behaves more like a Shape to position the chart on the sheet.

### Code to create an empty chart

Sub CreateEmptyChart()

'get reference to ActiveSheet

```
Dim sht 作为 工作表
设置 sht = ActiveSheet

'在位置(0, 0)创建一个新的ChartObject, 宽度400, 高度300
Dim chtObj As ChartObject
设置 chtObj = sht.ChartObjects.Add(0, 0, 400, 300)

'获取对图表对象的引用
Dim cht 作为 图表
设置 cht = chtObj.Chart

'附加代码以修改空图表
'...

End Sub
```

结果图表



### 第21.3节：通过修改SERIES公式创建图表

为了完全控制新的图表和系列对象（尤其是动态系列名称），必须直接修改SERIES公式。设置范围对象的过程很简单，主要难点只是构建SERIES公式的字符串。

SERIES公式采用以下语法：

```
=SERIES(名称, X值, 值, 顺序)
```

这些内容可以作为引用或数组值提供给数据项。顺序表示系列在图表中的位置。请注意，除非引用带有完整的工作表名称，否则数据引用将无法工作。要查看有效公式的示例，请点击任何现有系列并检查公式栏。

```
Dim sht As Worksheet
Set sht = ActiveSheet

'create a new ChartObject at position (0, 0) with width 400 and height 300
Dim chtObj As ChartObject
Set chtObj = sht.ChartObjects.Add(0, 0, 400, 300)

'get reference to chart object
Dim cht As Chart
Set cht = chtObj.Chart

'additional code to modify the empty chart
'...

End Sub
```

Resulting Chart



### Section 21.3: Create a Chart by Modifying the SERIES formula

For complete control over a new Chart and Series object (especially for a dynamic Series name), you must resort to modifying the SERIES formula directly. The process to set up the Range objects is straightforward and the main hurdle is simply the string building for the SERIES formula.

The SERIES formula takes the following syntax:

```
=SERIES(Name, XValues, Values, Order)
```

These contents can be supplied as references or as array values for the data items. Order represents the series position within the chart. Note that the references to the data will not work unless they are fully qualified with the sheet name. For an example of a working formula, click any existing series and check the formula bar.



使用SERIES公式创建图表并设置数据的代码

请注意，构建SERIES公式的字符串使用了.Address(,,,True)。这确保使用外部范围引用，从而包含带有工作表名称的完整限定地址。如果省略工作表名称，将会出现错误。

```
Sub CreateChartUsingSeriesFormula()

    Dim xData As Range
    Dim yData As Range
    Dim serName As Range

    '设置范围以获取数据和y值标签
    Set xData = Range("B3:B12")
    设置 yData = Range("C3:C12")
    设置 serName = Range("C2")

    '获取对活动工作表的引用
    Dim sht 作为 工作表
    设置 sht = ActiveSheet

    '在位置 (48, 195) 创建一个新的 ChartObject, 宽度400, 高度300
    Dim chtObj 作为 图表对象
    设置 chtObj = sht.ChartObjects.Add(48, 195, 400, 300)

    '获取对图表对象的引用
    Dim cht 作为 图表
    设置 cht = chtObj.Chart

    '创建新的系列
    Dim ser 作为 系列
    Set ser = cht.SeriesCollection.NewSeries

    '设置系列公式
    '=SERIES(name, xData, yData, plotOrder)

    Dim formulaValue As String
    formulaValue = "=SERIES(" & _
        serName.Address(, , , True) & "," & _
        xData.Address(, , , True) & "," & _
        yData.Address(, , , True) & ",1)"

    ser.Formula = formulaValue
    ser.ChartType = xlXYScatterLines

End Sub
```

原始数据和生成的图表

请注意，对于此图表，系列名称已正确设置为所需单元格的范围。这意味着更新将传播到图表中。

Code to create a chart and set up data using the SERIES formula

Note that the string building to create the SERIES formula uses .Address(, , ,True). This ensures that the external Range reference is used so that a fully qualified address with the sheet name is included. You **will get an error if the sheet name is excluded**.

```
Sub CreateChartUsingSeriesFormula()

    Dim xData As Range
    Dim yData As Range
    Dim serName As Range

    'set the ranges to get the data and y value label
    Set xData = Range("B3:B12")
    Set yData = Range("C3:C12")
    Set serName = Range("C2")

    'get reference to ActiveSheet
    Dim sht As Worksheet
    Set sht = ActiveSheet

    'create a new ChartObject at position (48, 195) with width 400 and height 300
    Dim chtObj As ChartObject
    Set chtObj = sht.ChartObjects.Add(48, 195, 400, 300)

    'get reference to chart object
    Dim cht As Chart
    Set cht = chtObj.Chart

    'create the new series
    Dim ser As Series
    Set ser = cht.SeriesCollection.NewSeries

    'set the SERIES formula
    '=SERIES(name, xData, yData, plotOrder)

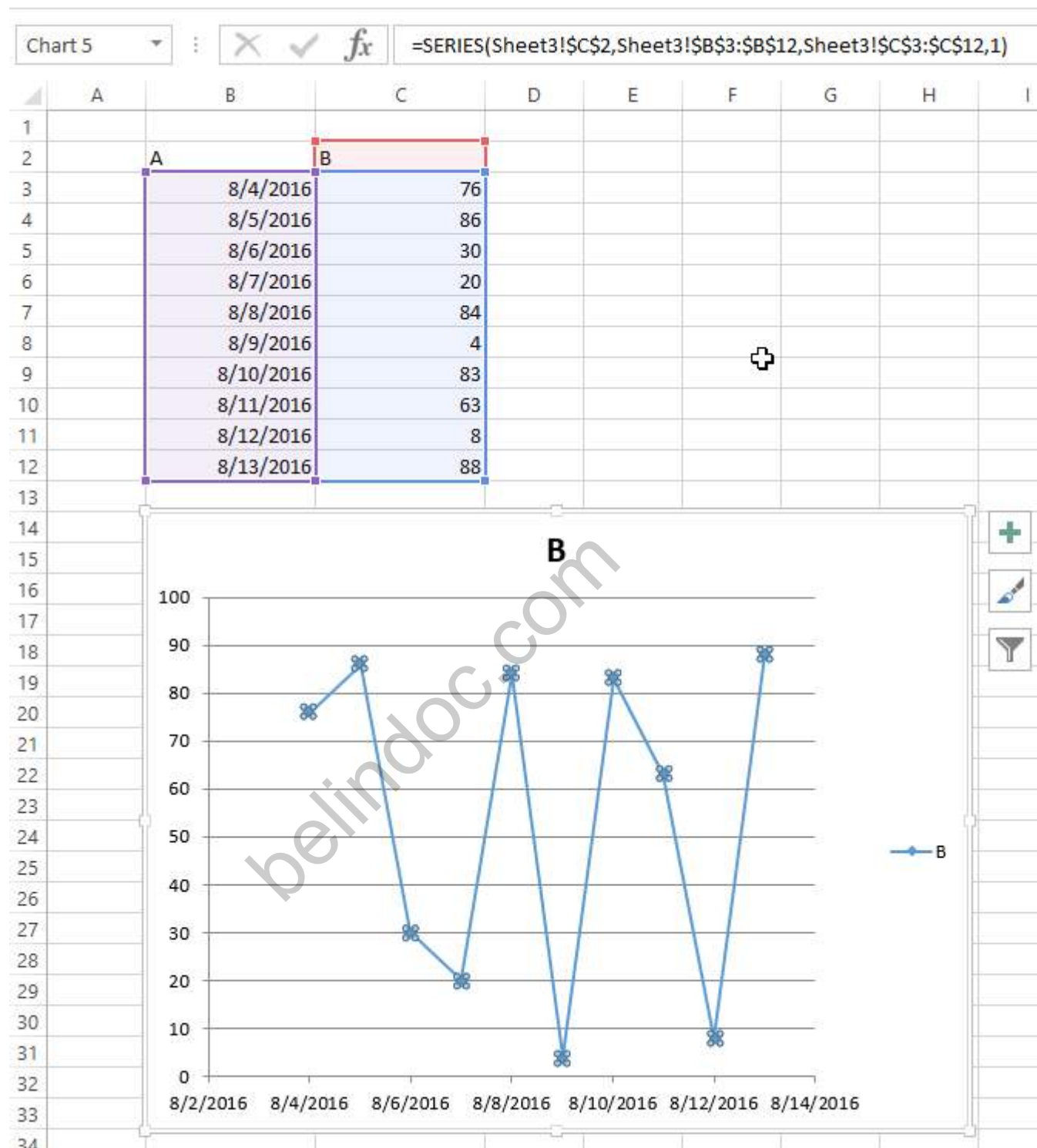
    Dim formulaValue As String
    formulaValue = "=SERIES(" & _
        serName.Address(, , , True) & "," & _
        xData.Address(, , , True) & "," & _
        yData.Address(, , , True) & ",1)"

    ser.Formula = formulaValue
    ser.ChartType = xlXYScatterLines

End Sub
```

Original data and resulting chart

Note that for this chart, the series name is properly set with a range to the desired cell. This means that updates will propagate to the Chart.



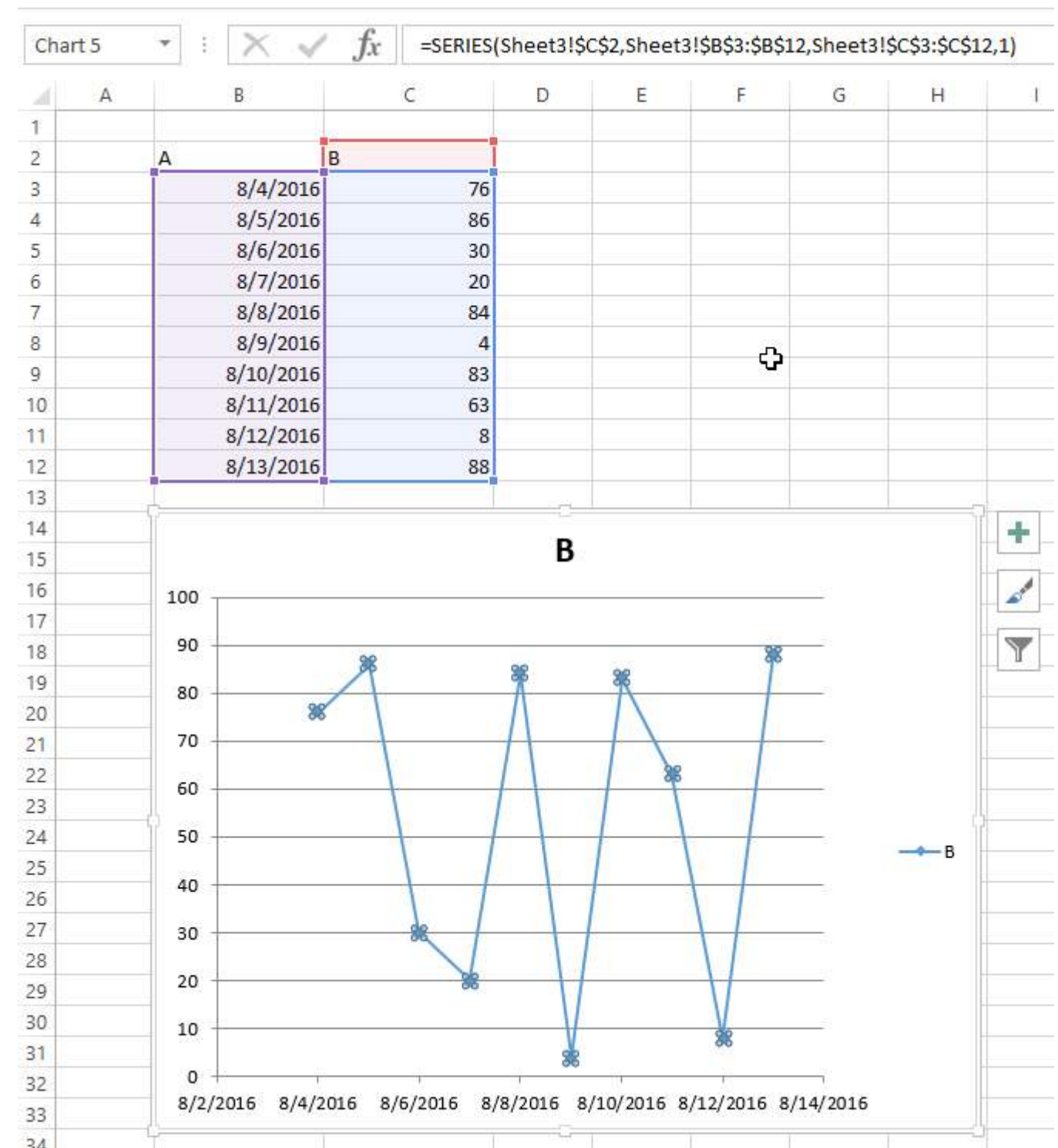
## 第21.4节：将图表排列成网格

在Excel中，处理图表的一个常见任务是统一多个图表在同一工作表上的大小和布局。如果手动操作，可以按住 **ALT** 在调整大小或移动图表时，使其“吸附”到单元格边界。这适用于几个图表，但使用VBA方法要简单得多。

### 创建网格的代码

此代码将从指定的（上，左）位置开始创建图表网格，定义列数和统一的图表大小。图表将按创建顺序排列，并在边缘换行形成新行。

```
Sub CreateGridOfCharts()
```



## Section 21.4: Arranging Charts into a Grid

A common chore with charts in Excel is standardizing the size and layout of multiple charts on a single sheet. If done manually, you can hold down **ALT** while resizing or moving the chart to "stick" to cell boundaries. This works for a couple charts, but a VBA approach is much simpler.

### Code to create a grid

This code will create a grid of charts starting at a given (Top, Left) position, with a defined number of columns, and a defined common chart size. The charts will be placed in the order they were created and wrap around the edge to form a new row.

```
Sub CreateGridOfCharts()
```

```
Dim int_cols As Integer
int_cols = 3

Dim cht_width As Double
cht_width = 250

Dim cht_height As Double
cht_height = 200

Dim offset_vertical As Double
offset_vertical = 195

Dim offset_horz As Double
offset_horz = 40

Dim sht As Worksheet
Set sht = ActiveSheet

Dim count As Integer
count = 0

'遍历当前工作表上的 ChartObjects
Dim cht_obj As ChartObject
For Each cht_obj In sht.ChartObjects

    '使用整数除法和取模来获取网格中的位置
    cht_obj.Top = (count \ int_cols) * cht_height + offset_vertical
    cht_obj.Left = (count Mod int_cols) * cht_width + offset_horz
    cht_obj.Width = cht_width
    cht_obj.Height = cht_height

    count = count + 1

Next cht_obj
End Sub
```

多个图表的结果

这些图片展示了图表的原始随机布局以及运行上述代码后得到的网格布局。

之前

```
Dim int_cols As Integer
int_cols = 3

Dim cht_width As Double
cht_width = 250

Dim cht_height As Double
cht_height = 200

Dim offset_vertical As Double
offset_vertical = 195

Dim offset_horz As Double
offset_horz = 40

Dim sht As Worksheet
Set sht = ActiveSheet

Dim count As Integer
count = 0

'iterate through ChartObjects on current sheet
Dim cht_obj As ChartObject
For Each cht_obj In sht.ChartObjects

    'use integer division and Mod to get position in grid
    cht_obj.Top = (count \ int_cols) * cht_height + offset_vertical
    cht_obj.Left = (count Mod int_cols) * cht_width + offset_horz
    cht_obj.Width = cht_width
    cht_obj.Height = cht_height

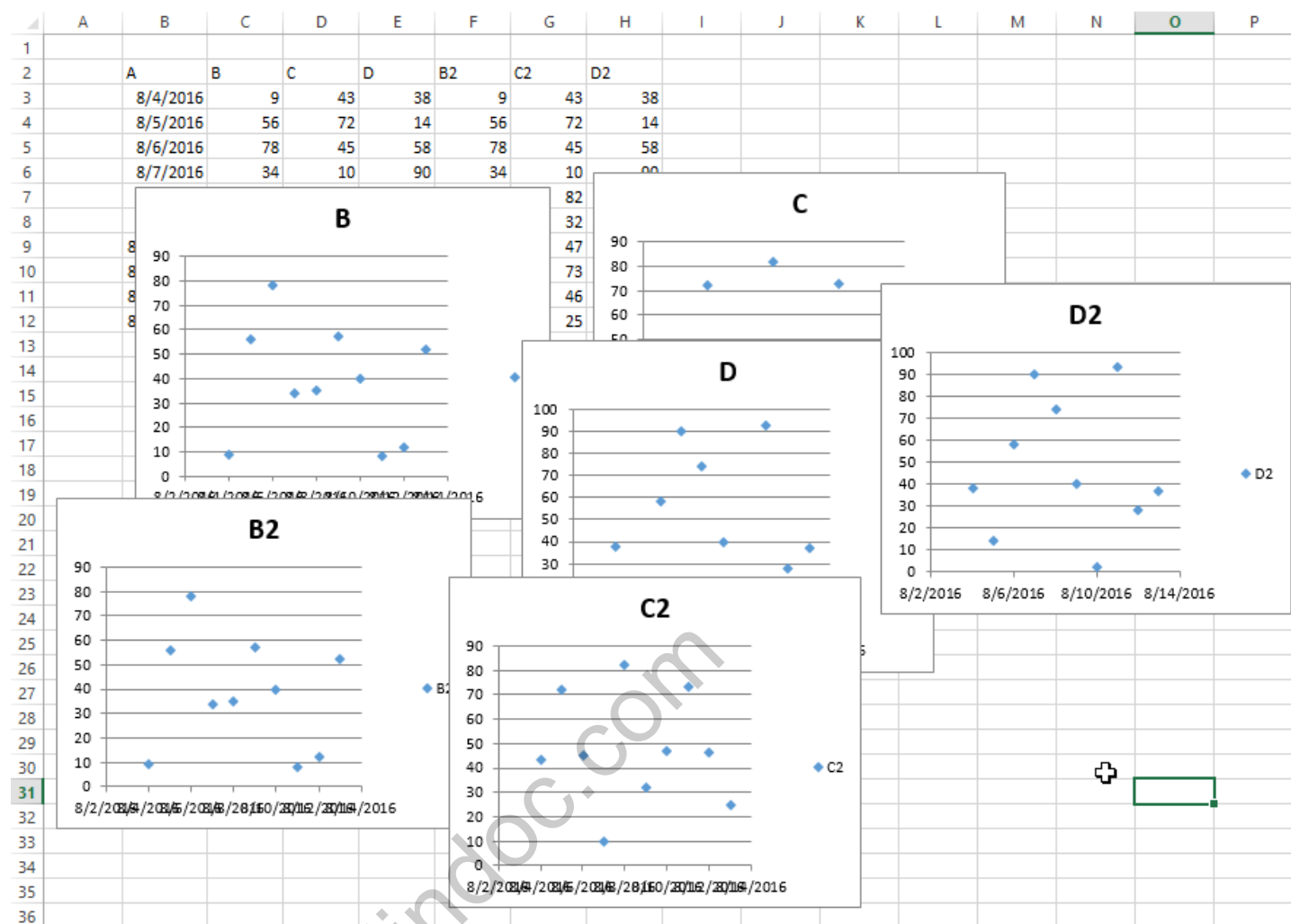
    count = count + 1

Next cht_obj
End Sub
```

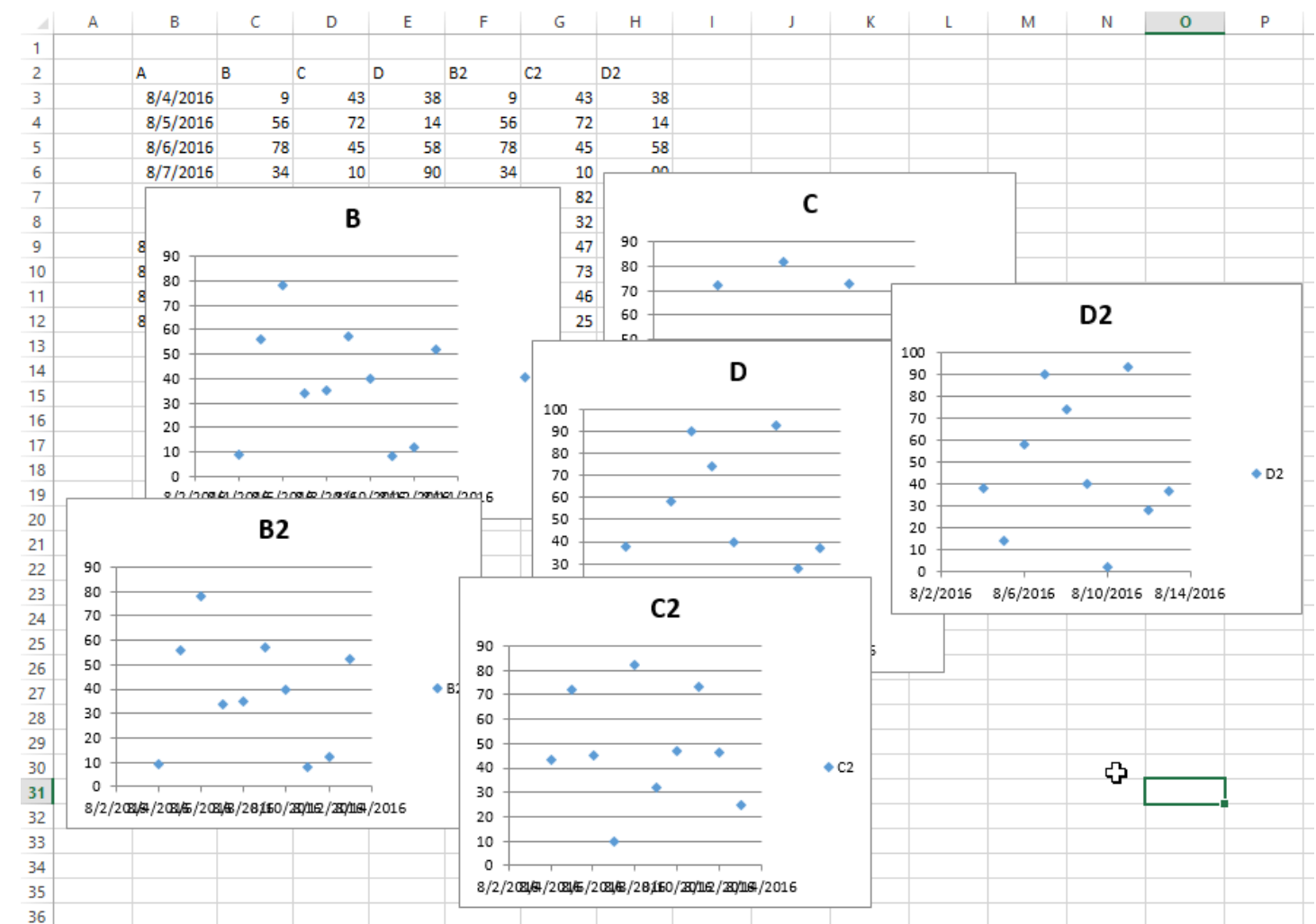
Result with several charts

These pictures show the original random layout of charts and the resulting grid from running the code above.

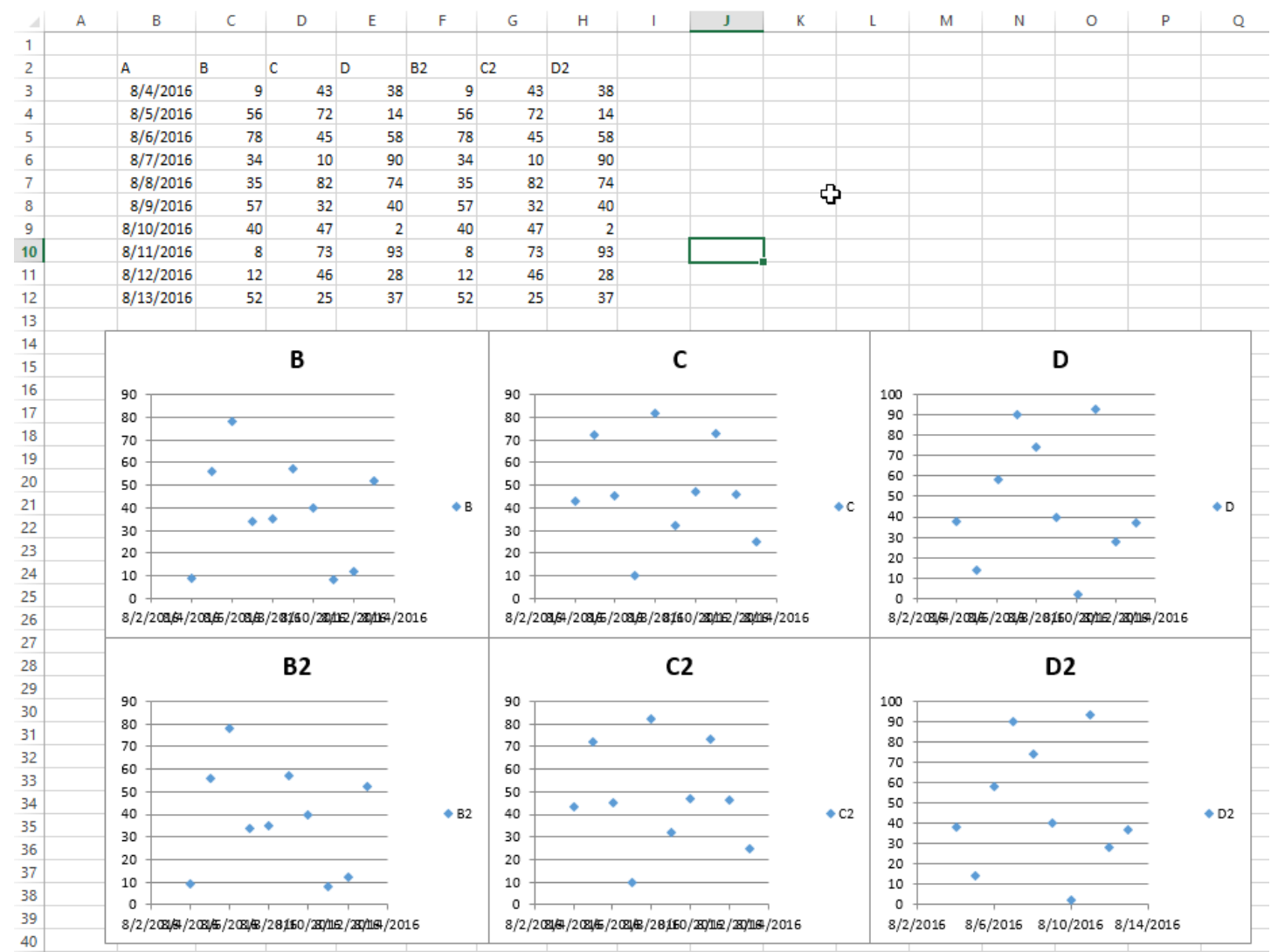
Before



之后



After





# 第22章：CustomDocumentProperties的实际应用

使用CustomDocumentProperties（CDP）是在同一工作簿中以相对安全的方式存储用户定义值的好方法，同时避免在未保护的工作表中直接显示相关单元格的值\*）。

注意：CDP表示一个独立的集合，可与BuiltInDocumentProperties相比较，但允许创建您自己的用户定义属性名称，而不是固定的集合。

\*) 另外，您也可以将值输入到隐藏或“非常隐藏”的工作簿中。

## 第22.1节：组织新的发票号码

增加发票号码并保存其值是一个常见任务。使用CustomDocumentProperties（CDP）是在同一工作簿中以相对安全的方式存储此类号码的好方法，同时避免在未保护的工作表中直接显示相关单元格的值。

附加提示：

另外，您也可以将值输入到隐藏工作表，甚至所谓的“非常隐藏”工作表中（参见使用xlVeryHidden工作表）。当然，也可以将数据保存到外部文件（例如ini文件、csv或任何其他类型）或注册表中。

示例内容：

下面的示例显示

- 一个函数 NextInvoiceNo，用于设置并返回下一个发票号码，一个过程 Del
- eteInvoiceNo，用于完全删除发票 CDP，以及一个过程 showAllCDPs，列出包含所
- 有名称的完整 CDP 集合。不使用 VBA，你也可以通过工作簿的信息来列出它们：信息 | 属性 [下拉菜单：] | 高级属性 | 自定义

您可以通过调用上述函数来获取和设置下一个发票号码（上一个号码加一），该函数返回字符串值以便于添加前缀。“InvoiceNo”在所有过程中隐式用作自定义文档属性（CDP）名称。

```
Dim sNumber As String
sNumber = NextInvoiceNo ()
```

示例代码：

```
Option Explicit

Sub Test()
    Dim sNumber As String
    sNumber = NextInvoiceNo()
    MsgBox "新发票号码: " & sNumber, vbInformation, "新发票号码"
End Sub

Function NextInvoiceNo() As String
' 目的：a) 如果尚不存在，则设置自定义文档属性（CDP）“InvoiceNo”
'        b) 增加CDP值并以字符串形式返回新值
' 声明
    Dim prop As Object
```

# Chapter 22: CustomDocumentProperties in practice

Using CustomDocumentProperties (CDPs) is a good method to store user defined values in a relatively safe way within the same work book, but avoiding to show related cell values simply in an unprotected work sheet \*).

Note: CDPs represent a separate collection comparable to BuiltInDocumentProperties, but allow to create user defined property names of your own instead of a fixed collection.

\*) Alternatively, you could enter values also in a hidden or "very hidden" workbook.

## Section 22.1: Organizing new invoice numbers

Incrementing an invoice number and saving its value is a frequent task. Using CustomDocumentProperties (CDPs) is a good method to store such numbers in a relatively safe way within the same work book, but avoiding to show related cell values simply in an unprotected work sheet.

Additional hint:

Alternatively, you could enter values also in a hidden worksheet or even a so called "very hidden" worksheet (see Using xlVeryHidden Sheets. Of course, it's possible to save data also to external files (e.g. ini file, csv or any other type) or the registry.

Example content:

The example below shows

- a function NextInvoiceNo that sets and returns the next invoice number,
- a procedure DeleteInvoiceNo, that deletes the invoice CDP completely, as well as
- a procedure showAllCDPs listing the complete CDPs collection with all names. Not using VBA, you can also list them via the workbook's information: Info | Properties [DropDown:] | Advanced Properties | Custom

You can get and set the next invoice number (last no plus one) simply by calling the above mentioned function, returning a string value in order to facilitate adding prefixes. "InvoiceNo" is implicitly used as CDP name in all procedures.

```
Dim sNumber As String
sNumber = NextInvoiceNo ()
```

Example code:

```
Option Explicit

Sub Test()
    Dim sNumber As String
    sNumber = NextInvoiceNo()
    MsgBox "New Invoice No: " & sNumber, vbInformation, "New Invoice Number"
End Sub

Function NextInvoiceNo() As String
' Purpose: a) Set Custom Document Property (CDP) "InvoiceNo" if not yet existing
'           b) Increment CDP value and return new value as string
' Declarations
    Dim prop As Object
```

```

Dim ret As String
Dim wb As Workbook
' 设置工作簿和自定义文档属性
Set wb = ThisWorkbook
Set prop = wb.CustomDocumentProperties

' -----
' 如果尚不存在，则生成新的自定义文档属性 "InvoiceNo"
' -----

If Not CDPEExists("InvoiceNo") Then
    ' 设置临时起始值 "0"
    prop.Add "InvoiceNo", False, msoPropertyTypeString, "0"
End If

' -----
' 增加发票号并返回函数值 (字符串)
' -----

ret = Format(Val(prop("InvoiceNo")) + 1, "0")
' a) 设置 CDP "InvoiceNo" = ret
prop("InvoiceNo").value = ret
' 返回函数值
NextInvoiceNo = ret
End Function

Private Function CDPEExists(sCDPName As String) As Boolean
' 目的：如果自定义文档属性 (CDP) 存在则返回True
' 方法：遍历CustomDocumentProperties集合，检查名称参数是否存在
' 参考网址：
http://stackoverflow.com/questions/23917977/alternatives-to-public-variables-in-vba/23918236#23918236
' 参见：
https://answers.microsoft.com/en-us/msoffice/forum/msoffice\_word-mso\_other/using-customdocumentproper-ties-with-vba/91ef15eb-b089-4c9b-a8a7-1685d073fb9f
' 声明
Dim cdp As Variant ' CustomDocumentProperties集合的元素
Dim boo As Boolean ' 表示元素是否存在的布尔值
For Each cdp In ThisWorkbook.CustomDocumentProperties
    If LCase(cdp.Name) = LCase(sCDPName) Then
boo = True ' 找到
Exit For ' 退出循环
End If
' 下一步
CDPEExists = boo ' 返回函数值
End Function

Sub DeleteInvoiceNo()
' 声明
Dim wb As Workbook
Dim prop As Object
' 设置工作簿和自定义文档属性
Set wb = ThisWorkbook
Set prop = wb.CustomDocumentProperties

' -----
' 删除自定义文档属性 "InvoiceNo"
' -----

If CDPEExists("InvoiceNo") Then
    prop("InvoiceNo").Delete
End If

```

End Sub

```

Dim ret As String
Dim wb As Workbook
' Set workbook and CDPs
Set wb = ThisWorkbook
Set prop = wb.CustomDocumentProperties

' -----
' Generate new CDP "InvoiceNo" if not yet existing
' -----

If Not CDPEExists("InvoiceNo") Then
    ' set temporary starting value "0"
    prop.Add "InvoiceNo", False, msoPropertyTypeString, "0"
End If

' -----
' Increment invoice no and return function value as string
' -----

ret = Format(Val(prop("InvoiceNo")) + 1, "0")
' a) Set CDP "InvoiceNo" = ret
prop("InvoiceNo").value = ret
' b) Return function value
NextInvoiceNo = ret
End Function

Private Function CDPEExists(sCDPName As String) As Boolean
' Purpose: return True if custom document property (CDP) exists
' Method: loop thru CustomDocumentProperties collection and check if name parameter exists
' Site: cf.
http://stackoverflow.com/questions/23917977/alternatives-to-public-variables-in-vba/23918236#23918236
' vgl.:
https://answers.microsoft.com/en-us/msoffice/forum/msoffice\_word-mso\_other/using-customdocumentproper-ties-with-vba/91ef15eb-b089-4c9b-a8a7-1685d073fb9f
' Declarations
Dim cdp As Variant ' element of CustomDocumentProperties Collection
Dim boo As Boolean ' boolean value showing element exists
For Each cdp In ThisWorkbook.CustomDocumentProperties
    If LCase(cdp.Name) = LCase(sCDPName) Then
        boo = True ' heureka
        Exit For ' exit loop
    End If
Next
CDPEExists = boo ' return value to function
End Function

Sub DeleteInvoiceNo()
' Declarations
Dim wb As Workbook
Dim prop As Object
' Set workbook and CDPs
Set wb = ThisWorkbook
Set prop = wb.CustomDocumentProperties

' -----
' Delete CDP "InvoiceNo"
' -----

If CDPEExists("InvoiceNo") Then
    prop("InvoiceNo").Delete
End If

```

End Sub

```

Sub showAllCDPs()
' 目的：显示所有自定义文档属性（CDP）及其值（如果已设置）
' 声明
Dim wb      As 工作簿
Dim cdp      As 对象

Dim i        As 整数
Dim maxi     As 整数
Dim s        As 字符串
' 设置工作簿和自定义文档属性
Set wb = ThisWorkbook
Set cdp = wb.CustomDocumentProperties
' 遍历自定义文档属性，获取名称和值
maxi = cdp.Count
For i = 1 To maxi
    出错时继续 ' 以防未设置的值
    s = s & Chr(i + 96) & ") " & _
        cdp(i).Name & "=" & cdp(i).value & vbCr
Next i
' 显示结果字符串
Debug.Print s
End Sub

```

```

Sub showAllCDPs()
' Purpose: Show all CustomDocumentProperties (CDP) and values (if set)
' Declarations
Dim wb      As Workbook
Dim cdp      As Object

Dim i        As Integer
Dim maxi     As Integer
Dim s        As String
' Set workbook and CDPs
Set wb = ThisWorkbook
Set cdp = wb.CustomDocumentProperties
' Loop thru CDP getting name and value
maxi = cdp.Count
For i = 1 To maxi
    On Error Resume Next ' necessary in case of unset value
    s = s & Chr(i + 96) & ") " & _
        cdp(i).Name & "=" & cdp(i).value & vbCr
Next i
' Show result string
Debug.Print s
End Sub

```

# 第23章：通过VBA集成PowerPoint

## 第23.1节：基础知识：从VBA启动PowerPoint

虽然根据所需功能可以更改许多参数并添加各种变体，但此示例展示了启动PowerPoint的基本框架。

注意： 该代码要求已将PowerPoint引用添加到活动的VBA项目中。请参阅引用文档条目，了解如何启用该引用。

首先，定义应用程序、演示文稿和幻灯片对象的变量。虽然可以使用后期绑定，但在适用时最好使用早期绑定。

```
Dim PPApP As PowerPoint.Application
Dim PPPres As PowerPoint.Presentation
Dim PPSlide As PowerPoint.Slide
```

接下来，打开或创建一个新的PowerPoint应用程序实例。这里使用了On Error Resume Next调用，以避免当PowerPoi nt尚未打开时，GetObject抛出错误。有关更详细的说明，请参阅最佳实践主题中的错误处理示例。

```
'如果PowerPoint未运行则打开，否则选择活动实例
On Error Resume Next
Set PPApP = GetObject(, "PowerPoint.Application")
On Error GoTo ErrHandler
If PPApP Is Nothing Then
    '打开 PowerPoint
    Set PPApP = CreateObject("PowerPoint.Application")
    PPApP.Visible = True
End If
```

应用程序启动后，将生成一个新的演示文稿及其中包含的幻灯片以供使用。

```
'生成用于图形创建的新演示文稿和幻灯片
Set PPPres = PPApP.Presentations.Add
Set PPSlide = PPPres.Slides.Add(1, ppLayoutBlank)
```

这里，幻灯片类型设置为4:3比例，启用幻灯片编号，并将窗口最大化显示在屏幕上。这些属性当然可以根据需要进行更改

```
PPApP.ActiveWindow.ViewType = ppViewSlide
PPPres.PageSetup.SlideOrientation = msoOrientationHorizontal
PPPres.PageSetup.SlideSize = ppSlideSizeOnScreen
PPPres.SlideMaster.HeadersFooters.SlideNumber.Visible = msoTrue
PPApP.ActiveWindow.WindowState = ppWindowMaximized
```

完成此代码后，将打开一个带有空白幻灯片的新PowerPoint窗口。通过使用对象变量，可以根据需要添加形状、文本、图形和Excel区域

# Chapter 23: PowerPoint Integration Through VBA

## Section 23.1: The Basics: Launching PowerPoint from VBA

While there are many parameters that can be changed and variations that can be added depending on the desired functionality, this example lays out the basic framework for launching PowerPoint.

**Note:** This code requires that the PowerPoint reference has been added to the active VBA Project. See the References Documentation entry to learn how to enable the reference.

First, define variables for the Application, Presentation, and Slide Objects. While this can be done with late binding, it is always best to use early binding when applicable.

```
Dim PPApP As PowerPoint.Application
Dim PPPres As PowerPoint.Presentation
Dim PPSlide As PowerPoint.Slide
```

Next, open or create a new instance of the PowerPoint application. Here, the On Error Resume Next call is used to avoid an error being thrown by GetObject if PowerPoint has not yet been opened. See the Error Handling example of the Best Practices Topic for a more detailed explanation.

```
'Open PPT if not running, otherwise select active instance
On Error Resume Next
Set PPApP = GetObject(, "PowerPoint.Application")
On Error GoTo ErrHandler
If PPApP Is Nothing Then
    'Open PowerPoint
    Set PPApP = CreateObject("PowerPoint.Application")
    PPApP.Visible = True
End If
```

Once the application has been launched, a new presentation and subsequently contained slide is generated for use.

```
'Generate new Presentation and slide for graphic creation
Set PPPres = PPApP.Presentations.Add
Set PPSlide = PPPres.Slides.Add(1, ppLayoutBlank)
```

Here, the slide type is set to the 4:3 shape with slide numbers enabled and the window maximized on the screen. These properties can, of course, be altered as needed

```
PPApP.ActiveWindow.ViewType = ppViewSlide
PPPres.PageSetup.SlideOrientation = msoOrientationHorizontal
PPPres.PageSetup.SlideSize = ppSlideSizeOnScreen
PPPres.SlideMaster.HeadersFooters.SlideNumber.Visible = msoTrue
PPApP.ActiveWindow.WindowState = ppWindowMaximized
```

Upon completion of this code, a new PowerPoint window with a blank slide will be open. By using the object variables, shapes, text, graphics, and excel ranges can be added as desired

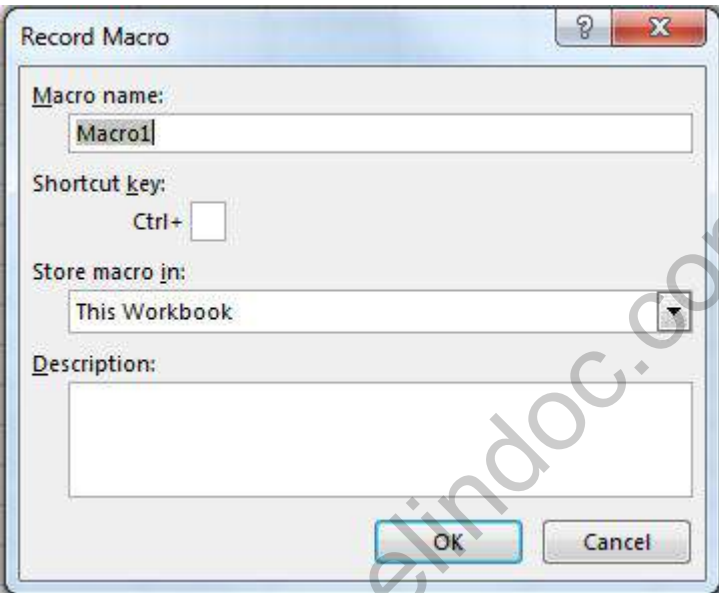
# 第24章：如何录制宏

## 第24.1节：如何录制宏

录制宏最简单的方法是点击Excel左下角的按钮，图标如下：



点击此按钮后，会弹出一个窗口，要求你为宏命名，并决定是否要设置快捷键。同时，还会询问宏的存储位置和描述。你可以选择任何名称，但不允许有空格。



如果你想为宏分配快捷键以便快速使用，请选择一个你能记住的字母，这样你就可以快速且轻松地反复使用该宏。

你可以将宏存储在“此工作簿”、“新工作簿”或“个人宏工作簿”中。如果你希望录制的宏仅在当前工作簿中可用，选择“此工作簿”。如果你想将其保存到一个全新的工作簿，选择“新工作簿”。如果你希望宏在你打开的任何工作簿中都可用，选择“个人宏工作簿”。

填写完弹出窗口后，点击“确定”。

然后执行你想用宏重复的操作。完成后，点击同一个按钮停止录制。按钮现在看起来是这样的：



现在你可以转到“开发工具”选项卡并打开Visual Basic。（或使用Alt + F11）

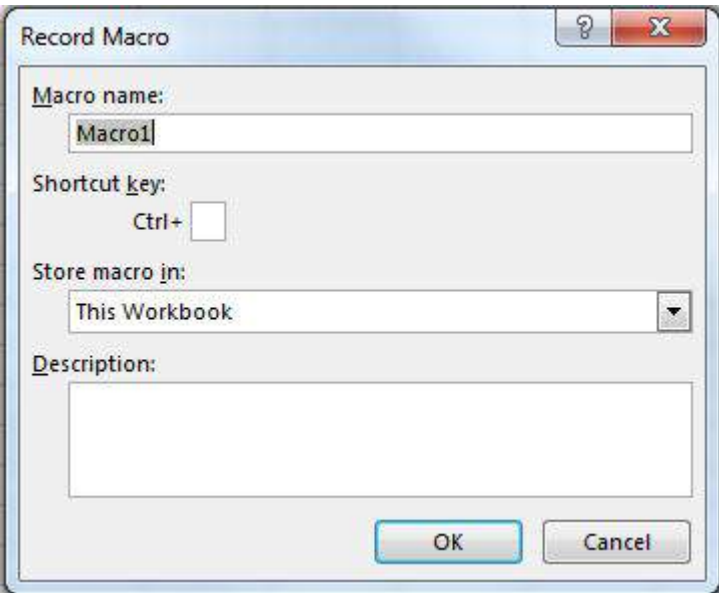
# Chapter 24: How to record a Macro

## Section 24.1: How to record a Macro

The easiest way to record a macro is the button in the lower left corner of Excel looks like this:



When you click on this you will get a pop-up asking you to name the Macro and decide if you want to have a shortcut key. Also, asks where to store the macro and for a description. You can choose any name you want, no spaces are allowed.



If you want to have a shortcut assigned to your macro for quick use choose a letter that you will remember so that you can quickly and easily use the macro over and over.

You can store the macro in "This Workbook," "New Workbook," or "Personal Macro Workbook." If you want the macro you're about to record to be available only in the current workbook, choose "This Workbook." If you want it saved to a brand new workbook, choose "New Workbook." And if you want the macro to be available to any workbook you open, choose "Personal Macro Workbook."

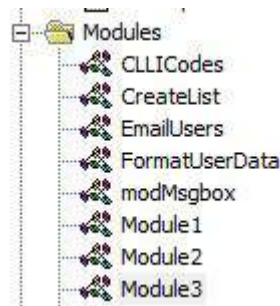
After you have filled out this pop-up click on "Ok".

Then perform whatever actions you want to repeat with the macro. When finished click the same button to stop recording. It now looks like this:



Now you can go to the Developer Tab and open Visual Basic. (or use Alt + F11)





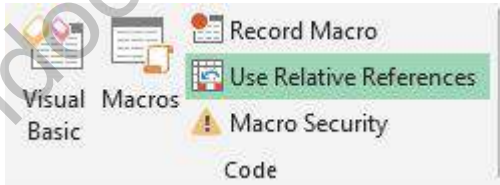
你现在将在“模块”文件夹下看到一个新模块。

最新的模块将包含你刚录制的宏。双击它以打开。

我做了一个简单的复制和粘贴：

```
Sub Macro1()  
'  
' Macro1 宏  
'  
'  
  
Selection.Copy  
Range("A12").Select  
ActiveSheet.Paste  
End Sub
```

如果你不想总是粘贴到“A12”，可以通过勾选“开发工具”选项卡上的“使用相对引用”框来使用相对引用：

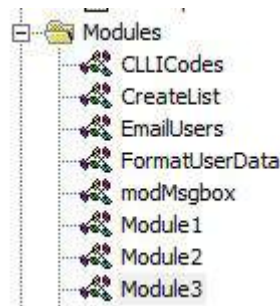


“使用相对引用”框

按照之前相同的步骤，现在将宏转换为如下内容：

```
Sub Macro2()  
'  
' Macro2 宏  
'  
'  
  
Selection.Copy  
ActiveCell.Offset(11, 0).Range("A1").Select  
ActiveSheet.Paste  
End Sub
```

仍然是将“A1”的值复制到向下11行的单元格，但现在你可以从任意起始单元格执行相同的宏，起始单元格的值将被复制到向下11行的单元格。



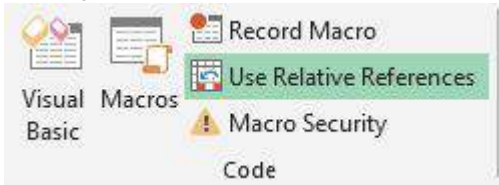
You will now have a new Module under the Modules folder.

The newest module will contain the macro you just recorded. Double-click on it to bring it up.

I did a simple copy and paste:

```
Sub Macro1()  
'  
' Macro1 Macro  
'  
'  
  
Selection.Copy  
Range("A12").Select  
ActiveSheet.Paste  
End Sub
```

If you don't want it to always paste into "A12" you can use Relative References by checking the "Use Relative



References" box on the Developer Tab:

Following the same steps as before will now turn the Macro into this:

```
Sub Macro2()  
'  
' Macro2 Macro  
'  
'  
  
Selection.Copy  
ActiveCell.Offset(11, 0).Range("A1").Select  
ActiveSheet.Paste  
End Sub
```

Still copying the value from "A1" into a cell 11 rows down, but now you can perform the same macro with any starting cell and the value from that cell will be copied to the cell 11 rows down.

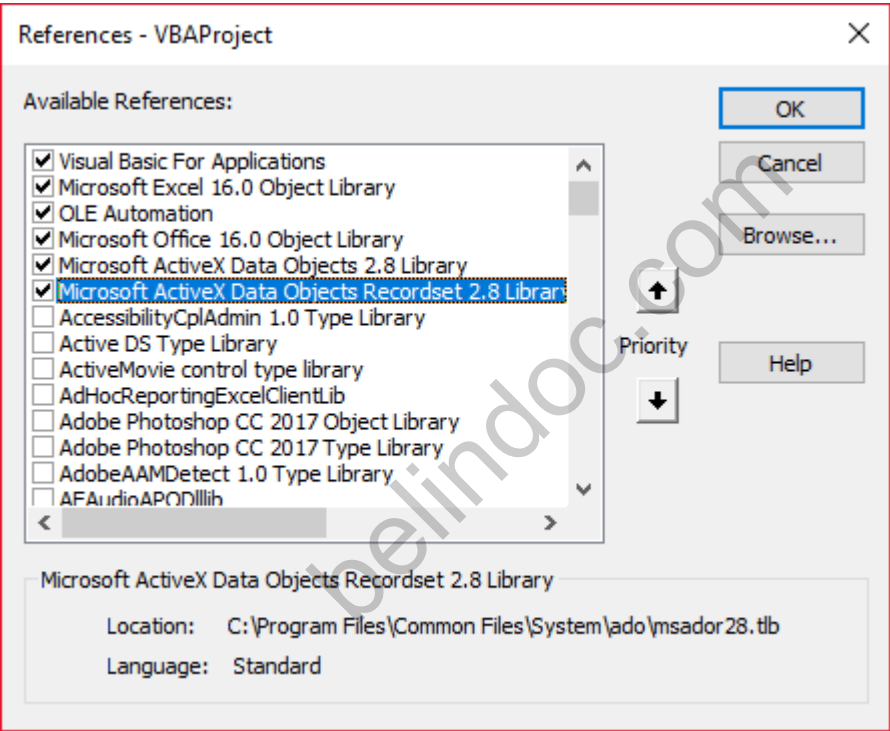
# 第25章：Excel VBA中的SQL - 最佳实践

## 第25.1节：如何在VBA中使用ADODB.Connection？

要求：

向项目中添加以下引用：

- Microsoft ActiveX Data Objects 2.8 Library
- Microsoft ActiveX Data Objects Recordset 2.8 Library



声明变量

```
Private mDataBase 作为新的 ADODB.Connection
Private mRS 作为新的 ADODB.Recordset
Private mCmd 作为新的 ADODB.Command
```

创建连接

a. 使用Windows身份验证

```
Private Sub OpenConnection(pServer 作为 字符串, pCatalog 作为 字符串)
    调用 mDataBase.Open("Provider=SQLOLEDB;Initial Catalog=" & pCatalog & ";Data Source=" & pServer & ";Integrated Security=SSPI")
    mCmd.ActiveConnection = mDataBase
End Sub
```

b. 使用SQL Server身份验证

```
Private Sub OpenConnection2(pServer 作为 字符串, pCatalog 作为 字符串, pUser 作为 字符串, pPsw 作为 字符串)
    调用 mDataBase.Open("Provider=SQLOLEDB;Initial Catalog=" & pCatalog & ";Data Source=" & pServer & ";Integrated Security=SSPI;User ID=" & pUser & ";Password=" & pPsw)
    mCmd.ActiveConnection = mDataBase
End Sub
```

执行SQL命令

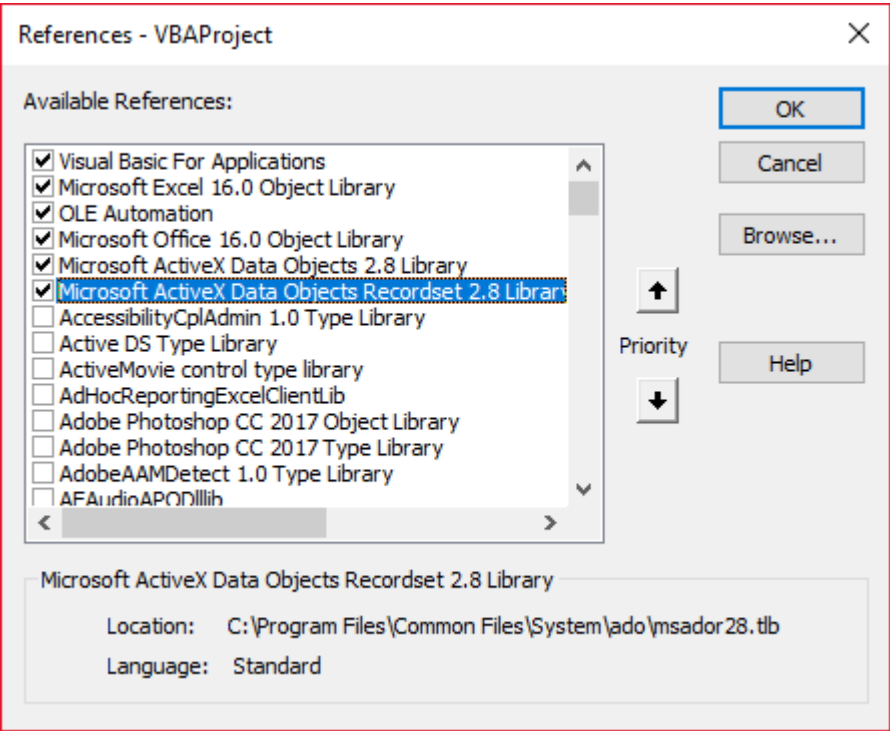
# Chapter 25: SQL in Excel VBA - Best Practices

## Section 25.1: How to use ADODB.Connection in VBA?

Requirements:

Add following references to the project:

- Microsoft ActiveX Data Objects 2.8 Library
- Microsoft ActiveX Data Objects Recordset 2.8 Library



Declare variables

```
Private mDataBase As New ADODB.Connection
Private mRS As New ADODB.Recordset
Private mCmd As New ADODB.Command
```

Create connection

a. with Windows Authentication

```
Private Sub OpenConnection(pServer As String, pCatalog As String)
    Call mDataBase.Open("Provider=SQLOLEDB;Initial Catalog=" & pCatalog & ";Data Source=" & pServer & ";Integrated Security=SSPI")
    mCmd.ActiveConnection = mDataBase
End Sub
```

b. with SQL Server Authentication

```
Private Sub OpenConnection2(pServer As String, pCatalog As String, pUser As String, pPsw As String)
    Call mDataBase.Open("Provider=SQLOLEDB;Initial Catalog=" & pCatalog & ";Data Source=" & pServer & ";Integrated Security=SSPI;User ID=" & pUser & ";Password=" & pPsw)
    mCmd.ActiveConnection = mDataBase
End Sub
```

Execute sql command

```
Private Sub ExecuteCmd(sql As String)
    mCmd.CommandText = sql
    设置 mRS = mCmd.Execute
结束子程序
```

从记录集读取数据

```
私有子程序 ReadRS()
    当(mRS.EOF)不为真时循环
    调试打印 "发货人ID: " & mRS.Fields("ShipperID").Value & " 公司名称: " &
    mRS.Fields("CompanyName").Value & " 电话: " & mRS.Fields("Phone").Value
        调用 mRS.MoveNext
    循环
End Sub
```

关闭连接

```
私有子程序 CloseConnection()
    调用 mDataBase.Close
    设置 mRS = Nothing
    设置 mCmd = Nothing
    设置 mDataBase = Nothing
结束子程序
```

如何使用？

```
公共子程序 Program()
    调用 OpenConnection("ServerName", "NORTHWND")
    调用 ExecuteCmd("INSERT INTO [NORTHWND].[dbo].[Shippers]([CompanyName],[Phone]) Values ('speedy
shipping','(503) 555-1234')")
    调用 ExecuteCmd("SELECT * FROM [NORTHWND].[dbo].[Shippers]")
    调用 ReadRS
    调用 CloseConnection
结束子程序
```

结果

承运人ID: 1 公司名称: Speedy Express 电话: (503) 555-9831

承运人ID: 2 公司名称: United Package 电话: (503) 555-3199

承运人ID: 3 公司名称: Federal Shipping 电话: (503) 555-9931

承运人ID: 4 公司名称: speedy shipping 电话: (503) 555-1234

```
Private Sub ExecuteCmd(sql As String)
    mCmd.CommandText = sql
    Set mRS = mCmd.Execute
End Sub
```

Read data from record set

```
Private Sub ReadRS()
    Do While Not (mRS.EOF)
        Debug.Print "ShipperID: " & mRS.Fields("ShipperID").Value & " CompanyName: " &
        mRS.Fields("CompanyName").Value & " Phone: " & mRS.Fields("Phone").Value
        Call mRS.MoveNext
    Loop
End Sub
```

Close connection

```
Private Sub CloseConnection()
    Call mDataBase.Close
    Set mRS = Nothing
    Set mCmd = Nothing
    Set mDataBase = Nothing
End Sub
```

How to use it?

```
Public Sub Program()
    Call OpenConnection("ServerName", "NORTHWND")
    Call ExecuteCmd("INSERT INTO [NORTHWND].[dbo].[Shippers]([CompanyName],[Phone]) Values ('speedy
shipping','(503) 555-1234')")
    Call ExecuteCmd("SELECT * FROM [NORTHWND].[dbo].[Shippers]")
    Call ReadRS
    Call CloseConnection
End Sub
```

Result

ShipperID: 1 CompanyName: Speedy Express Phone: (503) 555-9831

ShipperID: 2 CompanyName: United Package Phone: (503) 555-3199

ShipperID: 3 CompanyName: Federal Shipping Phone: (503) 555-9931

ShipperID: 4 CompanyName: speedy shipping Phone: (503) 555-1234

# 第26章：Excel-VBA优化

Excel-VBA优化也指通过文档和附加细节实现更好的错误处理编码。这里展示了这一点。

## 第26.1节：通过扩展调试优化错误查找

使用行号.....并在出错时进行记录（“查看Erl的重要性”）检测哪一行引发错误是任何调试的重要部分，并

缩小了查找原因的范围。用简短描述记录已识别的错误行，配合模块和过程名称，是成功错误追踪的完整做法。下面的示例将这些数据保存到日志文件中。

### 背景

错误对象返回错误编号（Err.Number）和错误描述（Err.Description），但并不明确回答错误发生的位置。 **Erl** 函数则可以，但前提是你需要在代码中添加 \*行号 ）（顺便说一句，这是对以前Basic时代的几个妥协之一）。

如果根本没有错误行，那么 Erl 函数返回0；如果行号不完整，则返回过程的最后一个前置行号。

```
Option Explicit

Public Sub MyProc1()
    Dim i As Integer
    Dim j As Integer
    On Error GoTo LogErr
    10j = 1 / 0 ' 引发错误
    okay:
    Debug.Print "i=" & i
Exit Sub

LogErr:
MsgBox LogErrors("MyModule", "MyProc1", Err), vbExclamation, "错误 " & Err.Number
Stop
Resume Next
End Sub

Public Function LogErrors( _
    ByVal sModule As String, _
    ByVal sProc As String, _
    Err As ErrObject) As String
' 目的：将错误编号、描述和 Erl 写入日志文件并返回错误文本
    Dim sLogFile As String: sLogFile = ThisWorkbook.Path & Application.PathSeparator &
"LogErrors.txt"
    Dim sLogTxt As String
    Dim lFile As Long

' 创建错误文本
    sLogTxt = sModule & "|" & sProc & "|Erl " & Erl & "|Err " & Err.Number & "|" & Err.Description

    On Error Resume Next
    lFile = FreeFile

    Open sLogFile For Append As lFile
    Print #lFile, Format$(Now(), "yy.mm.dd hh:mm:ss "); sLogTxt
```

# Chapter 26: Excel-VBA Optimization

Excel-VBA Optimization refers also to coding better error handling by documentation and additional details. This is shown here.

## Section 26.1: Optimizing Error Search by Extended Debugging

Using Line Numbers ... and documenting them in case of error ("The importance of seeing Erl")

Detecting which line raises an error is a substantial part of any debugging and narrows the search for the cause. To document identified error lines with a short description completes a successful error tracking, at best together with the names of module and procedure. The example below saves these data to a log file.

### Back ground

The error object returns error number (Err.Number) and error description (Err.Description), but doesn't explicitly respond to the question where to locate the error. The **Erl** function, however, does, but on condition that you add \*line numbers ) to the code (BTW one of several other concessions to former Basic times).

If there are no error lines at all, then the Erl function returns 0, if numbering is incomplete you'll get the procedure's last preceding line number.

```
Option Explicit

Public Sub MyProc1()
    Dim i As Integer
    Dim j As Integer
    On Error GoTo LogErr
    10j = 1 / 0 ' raises an error
    okay:
    Debug.Print "i=" & i
Exit Sub

LogErr:
MsgBox LogErrors("MyModule", "MyProc1", Err), vbExclamation, "Error " & Err.Number
Stop
Resume Next
End Sub

Public Function LogErrors( _
    ByVal sModule As String, _
    ByVal sProc As String, _
    Err As ErrObject) As String
' Purpose: write error number, description and Erl to log file and return error text
    Dim sLogFile As String: sLogFile = ThisWorkbook.Path & Application.PathSeparator &
"LogErrors.txt"
    Dim sLogTxt As String
    Dim lFile As Long

' Create error text
    sLogTxt = sModule & "|" & sProc & "|Erl " & Erl & "|Err " & Err.Number & "|" & Err.Description

    On Error Resume Next
    lFile = FreeFile

    Open sLogFile For Append As lFile
    Print #lFile, Format$(Now(), "yy.mm.dd hh:mm:ss "); sLogTxt
```

```
Print #lFile,
Close lFile
' 返回错误文本
LogErrors = sLogTxt
End Function
```

'显示日志文件的附加代码

```
Sub ShowLogFile()
    Dim sLogFile As String: sLogFile = ThisWorkbook.Path & Application.PathSeparator & "LogErrors.txt"

    On Error GoTo LogErr
    Shell "notepad.exe " & sLogFile, vbNormalFocus

    okay:
    On Error Resume Next
    Exit Sub

LogErr:
MsgBox LogErrors("MyModule", "ShowLogFile", Err), vbExclamation, "错误号 " & Err.Number
Resume okay
End Sub
```

## 第26.2节：禁用工作表更新

禁用工作表的计算可以显著减少宏的运行时间。此外，禁用事件、屏幕更新和分页符也会有益。以下Sub可用于任何宏以实现此目的。

```
Sub OptimizeVBA(isOn As Boolean)
Application.Calculation = IIf(isOn, xlCalculationManual, xlCalculationAutomatic)
Application.EnableEvents = Not(isOn)
Application.ScreenUpdating = Not(isOn)
ActiveSheet.DisplayPageBreaks = Not(isOn)
End Sub
```

为优化请遵循以下伪代码：

```
Sub MyCode()

    OptimizeVBA True

    '您的代码写在这里

    OptimizeVBA False

End Sub
```

## 第26.3节：行删除 - 性能

- 删除行很慢，特别是在循环遍历单元格并逐行删除时
- 另一种方法是使用自动筛选来隐藏要删除的行
- 复制可见区域并粘贴到新的工作表中
- 完全删除初始工作表

```
Print #lFile,
Close lFile
' Return error text
LogErrors = sLogTxt
End Function
```

'Additional Code to show log file

```
Sub ShowLogFile()
Dim sLogFile As String: sLogFile = ThisWorkbook.Path & Application.PathSeparator & "LogErrors.txt"

On Error GoTo LogErr
Shell "notepad.exe " & sLogFile, vbNormalFocus

okay:
On Error Resume Next
Exit Sub

LogErr:
MsgBox LogErrors("MyModule", "ShowLogFile", Err), vbExclamation, "Error No " & Err.Number
Resume okay
End Sub
```

## Section 26.2: Disabling Worksheet Updating

Disabling calculation of the worksheet can decrease running time of the macro significantly. Moreover, disabling events, screen updating and page breaks would be beneficial. Following Sub can be used in any macro for this purpose.

```
Sub OptimizeVBA(isOn As Boolean)
Application.Calculation = IIf(isOn, xlCalculationManual, xlCalculationAutomatic)
Application.EnableEvents = Not(isOn)
Application.ScreenUpdating = Not(isOn)
ActiveSheet.DisplayPageBreaks = Not(isOn)
End Sub
```

For optimization follow the below pseudo-code:

```
Sub MyCode()

    OptimizeVBA True

    'Your code goes here

    OptimizeVBA False

End Sub
```

## Section 26.3: Row Deletion - Performance

- Deleting rows is slow, specially when looping through cells and deleting rows, one by one
- A different approach is using an AutoFilter to hide the rows to be deleted
- Copy the visible range and Paste it into a new WorkSheet
- Remove the initial sheet entirely



- 使用此方法，删除的行越多，速度越快

示例：

```
Option Explicit

'已删除行数：775,153，总行数：1,000,009，耗时：1.87秒

Public Sub DeleteRows()
    Dim oldWs As 工作表, newWs As 工作表, wsName As 字符串, ur As 区域

    Set oldWs = ThisWorkbook.ActiveSheet
    wsName = oldWs.Name
    Set ur = oldWs.Range("F2", oldWs.Cells(oldWs.Rows.Count, "F").End(xlUp))

    Application.ScreenUpdating = False
    Set newWs = Sheets.Add(After:=oldWs)    '创建一个新的工作表

    With ur    '复制自动筛选后的可见区域（相应修改Criteria1和Criteria2）
        .AutoFilter Field:=1, Criteria1:="<>0", Operator:=xlAnd, Criteria2:="<>"
        oldWs.UsedRange.Copy
    End With
    '将所有可见数据粘贴到新的工作表（数值和格式）
    使用 newWs.Range(oldWs.UsedRange.Cells(1).Address)
        .PasteSpecial xlPasteColumnWidths
        .PasteSpecial xlPasteAll
    newWs.Cells(1, 1).Select: newWs.Cells(1, 1).Copy
    End With

    With Application
        .CutCopyMode = False
        .DisplayAlerts = False
        oldWs.Delete
        .DisplayAlerts = True
        .ScreenUpdating = True
    End With
    newWs.Name = wsName
End Sub
```

第26.4节：在执行大型宏之前禁用所有Excel功能

以下过程将在工作簿和工作表级别临时禁用所有Excel功能

- FastWB() 是一个接受开启或关闭标志的切换函数
- FastWS() 接受一个可选的工作表对象，或无参数
- 如果缺少 ws 参数，则会对集合中的所有工作表开启或关闭所有功能
  - 自定义类型可用于在关闭之前捕获所有设置
  - 在过程结束时，可以恢复初始设置

```
Public Sub FastWB(Optional ByVal opt As Boolean = True)
    With Application
        .Calculation = IIf(opt, xlCalculationManual, xlCalculationAutomatic)
        If .DisplayAlerts <> Not opt Then .DisplayAlerts = Not opt
    End With
End Sub
```

- With this method, the more rows to delete, the faster it will be

Example:

```
Option Explicit

'Deleted rows: 775,153，Total Rows: 1,000,009，Duration: 1.87 sec

Public Sub DeleteRows()
    Dim oldWs As Worksheet, newWs As Worksheet, wsName As String, ur As Range

    Set oldWs = ThisWorkbook.ActiveSheet
    wsName = oldWs.Name
    Set ur = oldWs.Range("F2", oldWs.Cells(oldWs.Rows.Count, "F").End(xlUp))

    Application.ScreenUpdating = False
    Set newWs = Sheets.Add(After:=oldWs)    'Create a new WorkSheet

    With ur    'Copy visible range after Autofilter (modify Criteria1 and 2 accordingly)
        .AutoFilter Field:=1, Criteria1:="<>0", Operator:=xlAnd, Criteria2:="<>"
        oldWs.UsedRange.Copy
    End With
    'Paste all visible data into the new WorkSheet (values and formats)
    With newWs.Range(oldWs.UsedRange.Cells(1).Address)
        .PasteSpecial xlPasteColumnWidths
        .PasteSpecial xlPasteAll
        newWs.Cells(1, 1).Select: newWs.Cells(1, 1).Copy
    End With

    With Application
        .CutCopyMode = False
        .DisplayAlerts = False
        oldWs.Delete
        .DisplayAlerts = True
        .ScreenUpdating = True
    End With
    newWs.Name = wsName
End Sub
```

Section 26.4: Disabling All Excel Functionality Before executing large macros

The procedures bellow will temporarily disable all Excel features at WorkBook and WorkSheet level

- FastWB() is a toggle that accepts On or Off flags
- FastWS() accepts an Optional WorkSheet object, or none
- If the ws parameter is missing it will turn all features on and off for all WorkSheets in the collection
  - A custom type can be used to capture all settings before turning them off
  - At the end of the process, the initial settings can be restored

```
Public Sub FastWB(Optional ByVal opt As Boolean = True)
    With Application
        .Calculation = IIf(opt, xlCalculationManual, xlCalculationAutomatic)
        If .DisplayAlerts <> Not opt Then .DisplayAlerts = Not opt
    End With
End Sub
```

```

    If .DisplayStatusBar <> Not opt Then .DisplayStatusBar = Not opt
    If .EnableAnimations <> Not opt Then .EnableAnimations = Not opt
    If .EnableEvents <> Not opt Then .EnableEvents = Not opt
    If .ScreenUpdating <> Not opt Then .ScreenUpdating = Not opt
End With
FastWS , opt
End Sub

```

```

Public Sub FastWS(Optional ByVal ws As Worksheet, Optional ByVal opt As Boolean = True)
    If ws Is Nothing Then
        For Each ws In Application.ThisWorkbook.Sheets
            OptimiseWS ws, opt
        Next
    Else
        OptimiseWS ws, opt
    End If
End Sub
Private Sub OptimiseWS(ByVal ws As Worksheet, ByVal opt As Boolean)
    With ws
        .DisplayPageBreaks = False
        .EnableCalculation = Not opt
        .EnableFormatConditionsCalculation = Not opt
        .EnablePivotTable = Not opt
    End With
End Sub

```

将所有 Excel 设置恢复为默认值

```

Public Sub XlResetSettings() '默认 Excel 设置
    With Application
        .Calculation = xlCalculationAutomatic
        .DisplayAlerts = True
        .DisplayStatusBar = True
        .EnableAnimations = False
        .EnableEvents = True
        .ScreenUpdating = True
        Dim sh As Worksheet
        For Each sh In Application.ThisWorkbook.Sheets
            With sh
                .DisplayPageBreaks = False
                .EnableCalculation = True
                .EnableFormatConditionsCalculation = True
                .EnablePivotTable = True
            End With
        Next sh
    End With
End Sub

```

## 第26.5节：执行时间检查

不同的过程可能会产生相同的结果，但它们使用的处理时间不同。为了检查哪个更快，可以使用如下代码：

```

time1 = Timer

对于每个 iCell 在 MyRange
    iCell = "文本"
下一个 iCell

```

```

    If .DisplayStatusBar <> Not opt Then .DisplayStatusBar = Not opt
    If .EnableAnimations <> Not opt Then .EnableAnimations = Not opt
    If .EnableEvents <> Not opt Then .EnableEvents = Not opt
    If .ScreenUpdating <> Not opt Then .ScreenUpdating = Not opt
End With
FastWS , opt
End Sub

```

```

Public Sub FastWS(Optional ByVal ws As Worksheet, Optional ByVal opt As Boolean = True)
    If ws Is Nothing Then
        For Each ws In Application.ThisWorkbook.Sheets
            OptimiseWS ws, opt
        Next
    Else
        OptimiseWS ws, opt
    End If
End Sub
Private Sub OptimiseWS(ByVal ws As Worksheet, ByVal opt As Boolean)
    With ws
        .DisplayPageBreaks = False
        .EnableCalculation = Not opt
        .EnableFormatConditionsCalculation = Not opt
        .EnablePivotTable = Not opt
    End With
End Sub

```

Restore all Excel settings to default

```

Public Sub XlResetSettings() 'default Excel settings
    With Application
        .Calculation = xlCalculationAutomatic
        .DisplayAlerts = True
        .DisplayStatusBar = True
        .EnableAnimations = False
        .EnableEvents = True
        .ScreenUpdating = True
        Dim sh As Worksheet
        For Each sh In Application.ThisWorkbook.Sheets
            With sh
                .DisplayPageBreaks = False
                .EnableCalculation = True
                .EnableFormatConditionsCalculation = True
                .EnablePivotTable = True
            End With
        Next sh
    End With
End Sub

```

## Section 26.5: Checking time of execution

Different procedures can give out the same result, but they would use different processing time. In order to check out which one is faster, a code like this can be used:

```

time1 = Timer

For Each iCell In MyRange
    iCell = "text"
Next iCell

```

```
time2 = Timer

对于 i = 1 到 30
    MyRange.Cells(i) = "文本"
下一个 i

time3 = Timer

debug.print "过程1时间: " & cStr(time2-time1)
debug.print "过程2时间: " & cStr(time3-time2)
```

MicroTimer:

```
Private Declare PtrSafe Function getFrequency Lib "Kernel32" Alias "QueryPerformanceFrequency"
(cyFrequency As Currency) As Long
Private Declare PtrSafe Function getTickCount Lib "Kernel32" Alias "QueryPerformanceCounter"
(cyTickCount As Currency) As Long

Function MicroTimer() As Double
    Dim cyTicks1 As Currency
    Static cyFrequency As Currency

    MicroTimer = 0
    如果 cyFrequency = 0 那么 getFrequency cyFrequency '获取频率
    getTickCount cyTicks1 '获取计数
    如果 cyFrequency 那么 MicroTimer = cyTicks1 / cyFrequency '返回秒数
结束函数
```

## 第26.6节：使用With块

使用 With 块可以加快运行宏的过程。与其写一个范围、图表名称、工作表等，不如像下面这样使用 With 块；

```
With ActiveChart
    .Parent.Width = 400
    .Parent.Height = 145
    .Parent.Top = 77.5 + 165 * step - replacer * 15
    .Parent.Left = 5
End With
```

这比下面的写法更快：

```
ActiveChart.Parent.Width = 400
ActiveChart.Parent.Height = 145
ActiveChart.Parent.Top = 77.5 + 165 * step - replacer * 15
ActiveChart.Parent.Left = 5
```

注意事项：

- 一旦进入 With 块，对象就不能更改。因此，不能使用单个 With 语句来影响多个不同的对象
- 不要在 With 块中跳入或跳出。如果 With 块中的语句被执行，但 With 或 End With 语句未被执行，一个包含对该对象引用的临时变量将保留在内存中，直到你退出过程
- 不要在 With 语句内部循环，尤其是当缓存对象用作迭代器时
- 你可以通过将一个 With 块嵌套在另一个 With 块内来嵌套 With 语句。然而，由于外层的成员

```
time2 = Timer

For i = 1 To 30
    MyRange.Cells(i) = "text"
Next i

time3 = Timer

debug.print "Proc1 time: " & cStr(time2-time1)
debug.print "Proc2 time: " & cStr(time3-time2)
```

MicroTimer:

```
Private Declare PtrSafe Function getFrequency Lib "Kernel32" Alias "QueryPerformanceFrequency"
(cyFrequency As Currency) As Long
Private Declare PtrSafe Function getTickCount Lib "Kernel32" Alias "QueryPerformanceCounter"
(cyTickCount As Currency) As Long

Function MicroTimer() As Double
    Dim cyTicks1 As Currency
    Static cyFrequency As Currency

    MicroTimer = 0
    If cyFrequency = 0 Then getFrequency cyFrequency 'Get frequency
    getTickCount cyTicks1 'Get ticks
    If cyFrequency Then MicroTimer = cyTicks1 / cyFrequency 'Returns Seconds
End Function
```

## Section 26.6: Using With blocks

Using with blocks can accelerate the process of running a macro. Instead writing a range, chart name, worksheet, etc. you can use with-blocks like below;

```
With ActiveChart
    .Parent.Width = 400
    .Parent.Height = 145
    .Parent.Top = 77.5 + 165 * step - replacer * 15
    .Parent.Left = 5
End With
```

Which is faster than this:

```
ActiveChart.Parent.Width = 400
ActiveChart.Parent.Height = 145
ActiveChart.Parent.Top = 77.5 + 165 * step - replacer * 15
ActiveChart.Parent.Left = 5
```

Notes:

- Once a With block is entered, object can't be changed. As a result, you can't use a single With statement to affect a number of different objects
- Don't jump into or out of With blocks.** If statements in a With block are executed, but either the With or End With statement is not executed, **a temporary variable containing a reference to the object remains in memory until you exit the procedure**
- Don't Loop inside With statements, especially if the cached object is used as an iterator
- You can nest With statements by placing one With block within another. However, because members of outer

在内部 With 块中屏蔽了 With 块时，必须在内部 With 块中为外部 With 块中对象的任何成员提供完全限定的对象引用。

嵌套示例：

此示例使用 With 语句对单个对象执行一系列语句。  
对象及其属性是用于说明目的的通用名称。

```
With MyObject
    .Height = 100           '等同于 MyObject.Height = 100。
    .Caption = "Hello World" '等同于 MyObject.Caption = "Hello World"。
    使用 .Font
        .Color = Red        '等同于 MyObject.Font.Color = Red。
        .Bold = True        '等同于 MyObject.Font.Bold = True。
    MyObject.Height = 200    '最内层的 With 指的是 MyObject.Font (必须限定)
End With
```

关于 MSDN 的更多信息

With blocks are masked within the inner With blocks, you must provide a fully qualified object reference in an inner With block to any member of an object in an outer With block.

Nesting Example:

This example uses the With statement to execute a series of statements on a single object.  
The object and its properties are generic names used for illustration purposes only.

```
With MyObject
    .Height = 100           'Same as MyObject.Height = 100.
    .Caption = "Hello World" 'Same as MyObject.Caption = "Hello World".
    With .Font
        .Color = Red        'Same as MyObject.Font.Color = Red.
        .Bold = True        'Same as MyObject.Font.Bold = True.
        MyObject.Height = 200 'Inner-most With refers to MyObject.Font (must be qualified)
    End With
End With
```

More Info on [MSDN](#)

# 第27章：VBA安全性

## 第27.1节：为您的VBA设置密码保护

有时您的VBA中包含敏感信息（例如密码），您不希望用户访问。您可以通过为VBA项目设置密码来实现对这些信息的基本保护。

请按照以下步骤操作：

- 1.打开您的Visual Basic编辑器（Alt + F11）
- 2.导航到 工具 -> VBA项目属性...
- 3.切换到“保护”选项卡
- 4.勾选“锁定项目以供查看”复选框
- 5.在“密码”和“确认密码”文本框中输入您想要的密码

现在，当有人想在Office应用程序中访问您的代码时，首先需要输入密码。但请注意，即使是强密码的VBA项目也很容易被破解。

# Chapter 27: VBA Security

## Section 27.1: Password Protect your VBA

Sometimes you have sensitive information in your VBA (e.g., passwords) that you don't want users to have access to. You can achieve basic security on this information by password-protecting your VBA project.

Follow these steps:

- 1. Open your Visual Basic Editor (Alt + F11)
- 2. Navigate to Tools -> VBAProject Properties...
- 3. Navigate to the Protection tab
- 4. Check off the "Lock project for viewing" checkbox
- 5. Enter your desired password in the Password and Confirm Password textboxes

Now when someone wants to access your code within an Office application, they will first need to enter the password. Be aware, however, that even a strong VBA project password is trivial to break.



# 第28章：调试与故障排除

## 第28.1节：立即窗口

如果您想测试一行宏代码，而不需要运行整个子程序，可以直接在立即窗口中输入命令，然后按ENTER运行该行代码。

要测试一行代码的输出，可以在前面加上问号?，直接将结果打印到立即窗口。或者，您也可以使用print命令来打印输出。

在Visual Basic编辑器中，按CTRL + G打开立即窗口。要将当前选定的工作表重命名为“ExampleSheet”，请在立即窗口中输入以下内容并按ENTER

```
ActiveSheet.Name = "ExampleSheet"
```

要直接在立即窗口打印当前选定工作表的名称

```
? ActiveSheet.Name
ExampleSheet
```

此方法在将内置函数或用户自定义函数应用到代码之前，测试其功能非常有用。下面的示例演示了如何使用立即窗口测试函数或一系列函数的输出，以确认预期结果。

```
'在此示例中，立即窗口用于确认一系列Left和Right
字符串方法将返回期望的字符串

'预期输出: "value"
print Left(Right("1111value1111",9),5) ' <---- 这里写的代码，按下回车键
value ' <---- 输出
```

立即窗口也可以用来设置或重置 Application、工作簿或其他所需属性。如果你在子程序中有 Application.EnableEvents = False，且该子程序意外抛出错误，导致程序关闭而未将该值重置为 True（这可能导致令人沮丧且意外的功能异常），那么这时可以直接在立即窗口中输入命令并运行：

```
? Application.EnableEvents ' <---- 测试 "EnableEvents" 当前状态
False ' <---- 输出
Application.EnableEvents = True ' <---- 将属性值重置为 True
? Application.EnableEvents ' <---- 测试 "EnableEvents" 当前状态
True ' <---- 输出
```

对于更高级的调试技巧，可以使用冒号：作为行分隔符。这可以用于多行表达式，例如下面示例中的循环。

```
x = Split("a,b,c",","): For i = LBound(x,1) to UBound(x,1): Debug.Print x(i): Next i ' <---- 输入
然后按回车
a ' <---- 输出
b ' <---- 输出
c ' <---- 输出
```

# Chapter 28: Debugging and Troubleshooting

## Section 28.1: Immediate Window

If you would like to test a line of macro code without needing to run an entire sub, you can type commands directly into the Immediate Window and hit ENTER to run the line.

For testing the output of a line, you can precede it with a question mark ? to print directly to the Immediate Window. Alternatively, you can also use the print command to have the output printed.

While in the Visual Basic Editor, press CTRL + G to open the Immediate Window. To rename your currently selected sheet to "ExampleSheet", type the following in the Immediate Window and hit ENTER

```
ActiveSheet.Name = "ExampleSheet"
```

To print the currently selected sheet's name directly in the Immediate Window

```
? ActiveSheet.Name
ExampleSheet
```

This method can be very useful to test the functionality of built in or user defined functions before implementing them in code. The example below demonstrates how the Immediate Window can be used to test the output of a function or series of functions to confirm an expected.

```
'In this example, the Immediate Window was used to confirm that a series of Left and Right
'string methods would return the desired string

'expected output: "value"
print Left(Right("1111value1111",9),5) ' <---- written code here, ENTER pressed
value ' <---- output
```

The Immediate Window can also be used to set or reset Application, Workbook, or other needed properties. This can be useful if you have Application.EnableEvents = False in a subroutine that unexpectedly throws an error, causing it to close without resetting the value to True (which can cause frustrating and unexpected functionality. In that case, the commands can be typed directly into the Immediate Window and run:

```
? Application.EnableEvents ' <---- Testing the current state of "EnableEvents"
False ' <---- Output
Application.EnableEvents = True ' <---- Resetting the property value to True
? Application.EnableEvents ' <---- Testing the current state of "EnableEvents"
True ' <---- Output
```

For more advanced debugging techniques, a colon : can be used as a line separator. This can be used for multi-line expressions such as looping in the example below.

```
x = Split("a,b,c",","): For i = LBound(x,1) to UBound(x,1): Debug.Print x(i): Next i ' <----Input
this and press enter
a ' <----Output
b ' <----Output
c ' <----Output
```

第28.2节：使用计时器查找性能瓶颈

优化速度的第一步是找到代码中最慢的部分。VBA函数 Timer 返回自午夜以来经过的秒数，精度为1/256秒（3.90625毫秒），适用于基于Windows的PC。VBA函数 Now 和 Time 的精度仅到秒。

```
Dim start As Double      ' Timer 返回 Single 类型，但转换为 Double 以避免
start = Timer             ' 在立即窗口中出现科学计数法如 3.90625E-03
' ... 代码的其他部分
Debug.Print Timer - start; "部分1中的秒数"

start = Timer
' ... 代码的另一部分
Debug.Print Timer - start; "第2部分的秒数"
```

第28.3节：调试器本地变量窗口

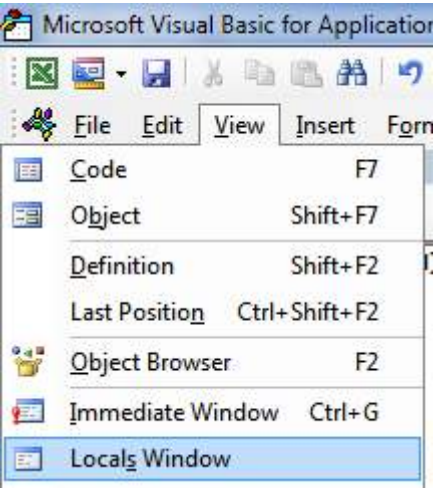
本地变量窗口提供了对当前运行的函数或子程序作用域内变量和对象的值的便捷访问。它是调试代码和逐步跟踪变化以发现问题的必备工具。它还允许你探索可能之前不知道存在的属性。

以下是一个示例，

```
Option Explicit
Sub LocalsWindowExample()
    Dim findMeInLocals As Integer
    Dim findMEInLocals2 As Range

    findMeInLocals = 1
    Set findMEInLocals2 = ActiveWorkbook.Sheets(1).Range("A1")
End Sub
```

在 VBA 编辑器中，点击 查看 --> 局部变量窗口



然后在子程序内点击后，使用 F8 单步执行代码，我们会在赋值给 findMeInLocals 之前停止。下面你可以看到该值为 0 --- 这就是如果你从未给它赋值时会使用的值。该范围对象是 'Nothing'。

Section 28.2: Use Timer to Find Bottlenecks in Performance

The first step in optimizing for speed is finding the slowest sections of code. The Timer VBA function returns the number of seconds elapsed since midnight with a precision of 1/256th of a second (3.90625 milliseconds) on Windows based PCs. The VBA functions Now and Time are only accurate to a second.

```
Dim start As Double      ' Timer returns Single, but converting to Double to avoid
start = Timer             ' scientific notation like 3.90625E-03 in the Immediate window
' ... part of the code
Debug.Print Timer - start; "seconds in part 1"

start = Timer
' ... another part of the code
Debug.Print Timer - start; "seconds in part 2"
```

Section 28.3: Debugger Locals Window

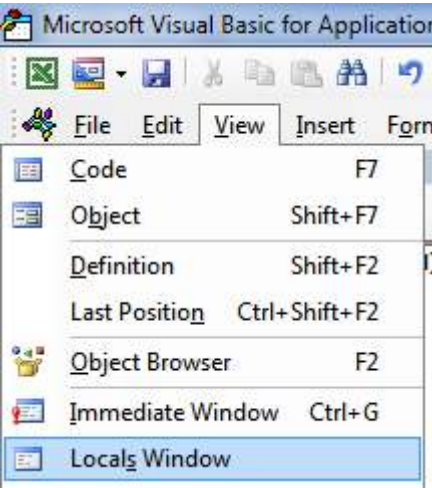
The Locals window provides easy access to the current value of variables and objects within the scope of the function or subroutine you are running. It is an essential tool to debugging your code and stepping through changes in order to find issues. It also allows you to explore properties you might not have known existed.

Take the following example,

```
Option Explicit
Sub LocalsWindowExample()
    Dim findMeInLocals As Integer
    Dim findMEInLocals2 As Range

    findMeInLocals = 1
    Set findMEInLocals2 = ActiveWorkbook.Sheets(1).Range("A1")
End Sub
```

In the VBA Editor, click View --> Locals Window



Then by stepping through the code using F8 after clicking inside the subroutine, we have stopped before getting to assigning findMeInLocals. Below you can see the value is 0 --- and this is what would be used if you never assigned it a value. The range object is 'Nothing'.

Option Explicit  
Sub LocalsWindowExample()  
Dim findMeInLocals As Integer  
Dim findMEInLocals2 As Range  
  
findMeInLocals = 1  
Set findMEInLocals2 = ActiveWorkbook.Sheets(1).Range("A1")  
  
End Sub

Locals

VBAProject.Sheet1.LocalsWindowExample

Expression	Value	Type
Me		Sheet1/Sheet1
findMeInLocals	0	Integer
findMEInLocals2	Nothing	Range

如果我们在子程序结束前停止，可以看到变量的最终值。

Option Explicit  
Sub LocalsWindowExample()  
Dim findMeInLocals As Integer  
Dim findMEInLocals2 As Range  
  
findMeInLocals = 1  
Set findMEInLocals2 = ActiveWorkbook.Sheets(1).Range("A1")  
  
End Sub

Locals

VBAProject.Sheet1.LocalsWindowExample

Expression	Value	Type
Me		Sheet1/Sheet1
findMeInLocals	1	Integer
findMEInLocals2	Nothing	Range

我们可以看到 findMeInLocals 的值为 1，类型为 Integer，FindMeInLocals2 的类型为 Range/Range。如果点击 + 号，我们可以展开该对象并查看其属性，比如 count 或 column。

Locals

VBAProject.Sheet1.LocalsWindowExample

Expression	Value	Type
Me		Sheet1/Sheet1
findMeInLocals	1	Integer
findMEInLocals2		Range/Range
AddIndent	False	Variant/Boolean
AllowEdit	True	Boolean
Application		Application/Application
Areas		Areas/Areas
Borders		Borders/Borders
Cells		Range/Range
Column	1	Long
ColumnWidth	8.43	Variant/Double
Comment	Nothing	Comment
Count	1	Long
CountLarge	1	Variant/Unsupported variant type
Creator	xlCreatorCode	XICreator
CurrentArray	<No cells were found.>	Range
CurrentRegion		Range/Range
Dependents	<No cells were found.>	Range
DirectDependents	<No cells were found.>	Range
DirectPrecedents	<No cells were found.>	Range
DisplayFormat		DisplayFormat/DisplayFormat

## 第 28.4 节：Debug.Print

要将错误代码描述列表打印到立即窗口，传递给 Debug.Print 函数：

```
Private Sub ListErrCodes()  
    Debug.Print "列出错误代码描述"  
    For i = 0 To 65535  
        e = Error(i)  
        If e <> "应用程序定义或对象定义的错误" Then Debug.Print i & ": " & e  
    Next i  
End Sub
```

Option Explicit  
Sub LocalsWindowExample()  
Dim findMeInLocals As Integer  
Dim findMEInLocals2 As Range  
  
findMeInLocals = 1  
Set findMEInLocals2 = ActiveWorkbook.Sheets(1).Range("A1")  
  
End Sub

Locals

VBAProject.Sheet1.LocalsWindowExample

Expression	Value	Type
Me		Sheet1/Sheet1
findMeInLocals	0	Integer
findMEInLocals2	Nothing	Range

If we stop right before the subroutine ends, we can see the final values of the variables.

Option Explicit  
Sub LocalsWindowExample()  
Dim findMeInLocals As Integer  
Dim findMEInLocals2 As Range  
  
findMeInLocals = 1  
Set findMEInLocals2 = ActiveWorkbook.Sheets(1).Range("A1")  
  
End Sub

Locals

VBAProject.Sheet1.LocalsWindowExample

Expression	Value	Type
Me		Sheet1/Sheet1
findMeInLocals	1	Integer
findMEInLocals2	Nothing	Range

We can see findMeInLocals with a value of 1 and type of Integer, and FindMeInLocals2 with a type of Range/Range. If we click the + sign we can expand the object and see its properties, such as count or column.

Locals

VBAProject.Sheet1.LocalsWindowExample

Expression	Value	Type
Me		Sheet1/Sheet1
findMeInLocals	1	Integer
findMEInLocals2		Range/Range
AddIndent	False	Variant/Boolean
AllowEdit	True	Boolean
Application		Application/Application
Areas		Areas/Areas
Borders		Borders/Borders
Cells		Range/Range
Column	1	Long
ColumnWidth	8.43	Variant/Double
Comment	Nothing	Comment
Count	1	Long
CountLarge	1	Variant/Unsupported variant type
Creator	xlCreatorCode	XICreator
CurrentArray	<No cells were found.>	Range
CurrentRegion		Range/Range
Dependents	<No cells were found.>	Range
DirectDependents	<No cells were found.>	Range
DirectPrecedents	<No cells were found.>	Range
DisplayFormat		DisplayFormat/DisplayFormat

## Section 28.4: Debug.Print

To print a listing of the Error Code descriptions to the Immediate Window, pass it to the Debug.Print function:

```
Private Sub ListErrCodes()  
    Debug.Print "List Error Code Descriptions"  
    For i = 0 To 65535  
        e = Error(i)  
        If e <> "Application-defined or object-defined error" Then Debug.Print i & ": " & e  
    Next i  
End Sub
```

您可以通过以下方式显示即时窗口：

- 从菜单栏选择**View | Immediate Window**
- 使用快捷键**Ctrl-G**

## 第28.5节：停止

调用停止命令时，程序执行将暂停。从此处可以恢复执行或逐步执行。

```
Sub Test()  
    Dim TestVar 作为String  
    TestVar = "Hello World"  
    Stop          '子程序将执行到此处然后等待用户  
    MsgBox TestVar  
End Sub
```

## 第28.6节：向代码添加断点

您可以通过点击VBA代码行左侧的灰色列轻松添加断点，执行将在此处停止。该列会出现一个红点，断点代码也会被红色高亮显示。

您可以在代码中添加多个断点，恢复执行可通过点击菜单栏中的“播放”图标实现。并非所有代码行都可以设置断点，例如变量定义行、过程的第一行或最后一行以及注释行不能被选为断点。



You can show the Immediate Window by:

- Selecting **View | Immediate Window** from the menu bar
- Using the keyboard shortcut **Ctrl-G**

## Section 28.5: Stop

The Stop command will pause the execution when called. From there, the process can be resumed or be executed step by step.

```
Sub Test()  
    Dim TestVar as String  
    TestVar = "Hello World"  
    Stop          'Sub will be executed to this point and then wait for the user  
    MsgBox TestVar  
End Sub
```

## Section 28.6: Adding a Breakpoint to your code

You can easily add a breakpoint to your code by clicking on the grey column to the left of the line of your VBA code where you want execution to stop. A red dot appears in the column and the breakpoint code is also highlighted in red.

You can add multiple breakpoints throughout your code and resuming execution is achieved by pressing the "play" icon in your menu bar. Not all code can be a breakpoint as variable definition lines, the first or last line of a procedure and comment lines cannot be selected as a breakpoint.

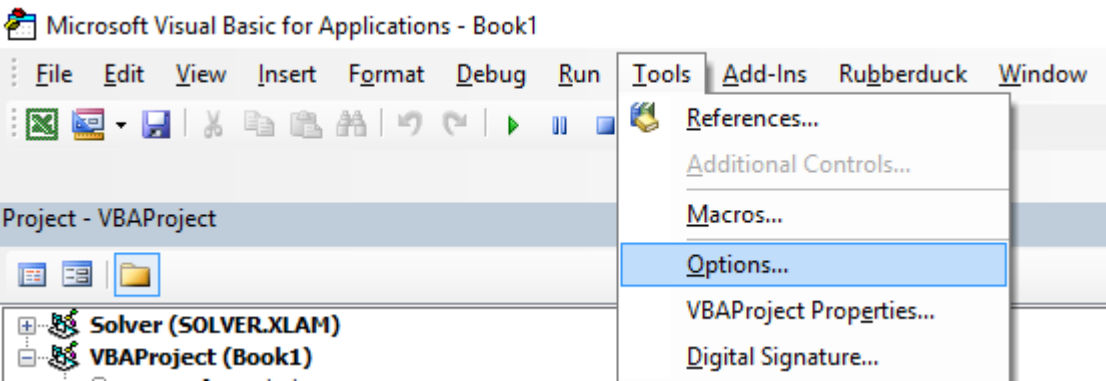




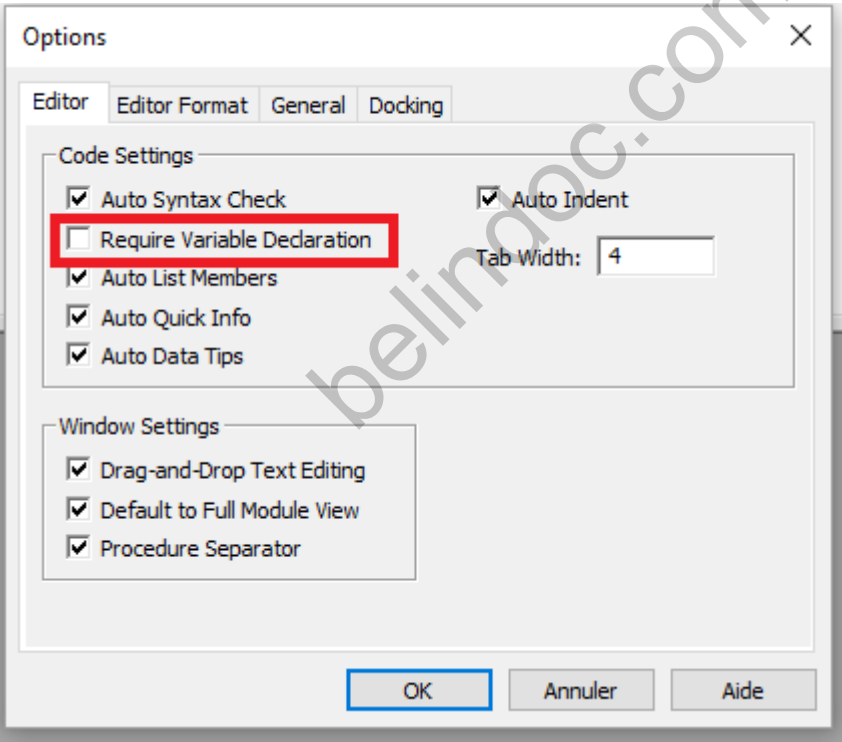
# 第29章：VBA最佳实践

## 第29.1节：始终使用“Option Explicit”

在VBA编辑器窗口中，从“工具”菜单选择“选项”：



然后在“编辑器”选项卡中，确保勾选了“要求变量声明”：



选择此选项将自动在每个VBA模块的顶部添加Option Explicit。

**小提示：** 这适用于到目前为止尚未打开的模块、类模块等。因此，如果你之前已经查看过例如Sheet1的代码，然后才激活“要求变量声明”选项，Option Explicit 将不会被添加！

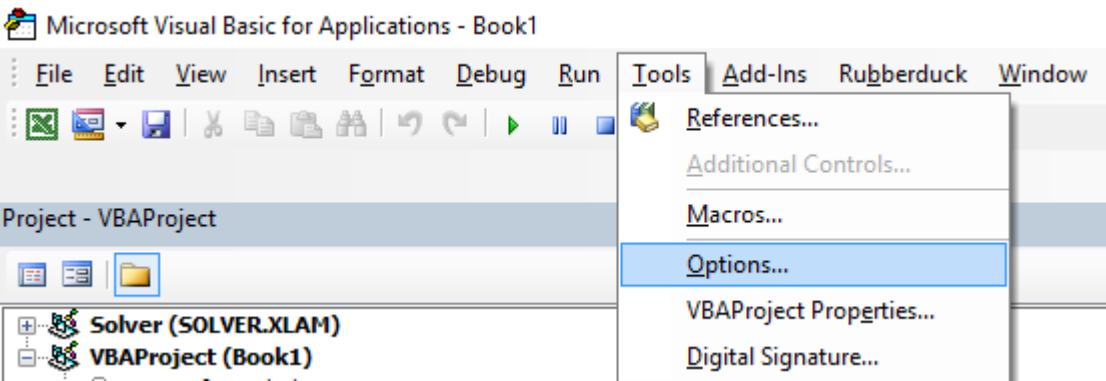
Option Explicit 要求每个变量在使用前必须被定义，例如使用Dim语句。未启用Option Explicit时，任何未识别的词都会被VBA编译器视为新变量，类型为Variant，这会导致因拼写错误而极难发现的错误。启用Option Explicit后，任何未识别的词都会引发编译错误，指出出错的代码行。

示例：

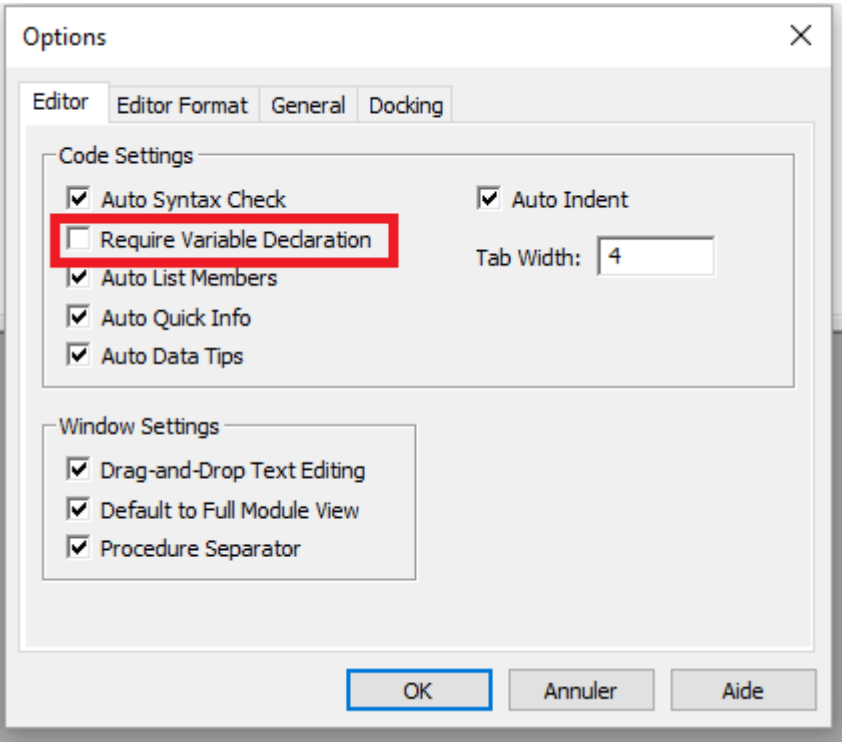
# Chapter 29: VBA Best Practices

## Section 29.1: ALWAYS Use "Option Explicit"

In the VBA Editor window, from the Tools menu select "Options":



Then in the "Editor" tab, make sure that "Require Variable Declaration" is checked:



Selecting this option will automatically put **Option** Explicit at the top of every VBA module.

**Small note:** This is true for the modules, class modules, etc. that haven't been opened so far. So if you already had a look at e.g. the code of Sheet1 before activating the option "Require Variable Declaration", **Option** Explicit will not be added!

**Option** Explicit requires that every variable has to be defined before use, e.g. with a **Dim** statement. Without **Option** Explicit enabled, any unrecognized word will be assumed by the VBA compiler to be a new variable of the Variant type, causing extremely difficult-to-spot bugs related to typographical errors. With **Option** Explicit enabled, any unrecognized words will cause a compile error to be thrown, indicating the offending line.

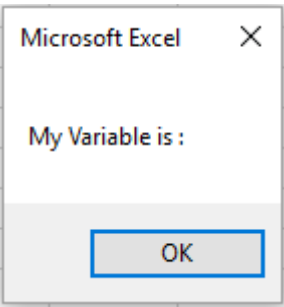
Example：



如果你运行以下代码：

```
Sub Test()  
my_variable = 12  
MsgBox "我的变量是：" & myvariable  
End Sub
```

你将看到以下消息：




你写成了myvariable而不是my\_variable，导致消息框显示一个空变量。如果你使用Option Explicit，这种错误就不可能发生，因为你会收到一个编译错误消息提示问题所在。

Option Explicit

```
Sub Test()  
my_variable = 12  
MsgBox "My Variable is：" & myvariable  
End Sub
```

Microsoft Visual Basic for Applications

 Compile error:  
Variable not defined

OK

Aide

现在如果你添加正确的声明：

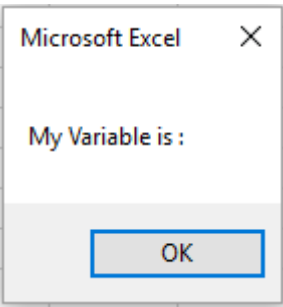
```
Sub Test()  
Dim my_variable As Integer  
my_variable = 12  
MsgBox "我的变量是：" & myvariable  
End Sub
```

你将得到一个错误消息，准确指出myvariable的错误：

If you run the following code：

```
Sub Test()  
my_variable = 12  
MsgBox "My Variable is：" & myvariable  
End Sub
```

You will get the following message：




You have made an error by writing myvariable instead of my\_variable, then the message box displays an empty variable. If you use Option Explicit, this error is not possible because you will get a compile error message indicating the problem.

Option Explicit

```
Sub Test()  
my_variable = 12  
MsgBox "My Variable is：" & myvariable  
End Sub
```

Microsoft Visual Basic for Applications

 Compile error:  
Variable not defined

OK

Aide

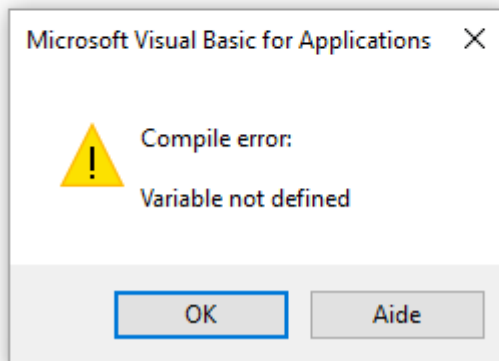
Now if you add the correct declaration：

```
Sub Test()  
Dim my_variable As Integer  
my_variable = 12  
MsgBox "My Variable is：" & myvariable  
End Sub
```

You will obtain an error message indicating precisely the error with myvariable：

## Option Explicit

```
Sub Test()  
    Dim my_variable As Integer  
    my_variable = 12  
    MsgBox "My Variable is : " & myvariable  
End Sub
```



### 关于Option Explicit和数组的说明（声明动态数组）：

你可以使用ReDim语句在过程内隐式声明数组。

- 使用ReDim语句时要小心不要拼写错误数组名称
- 即使模块中包含Option Explicit语句，也会创建一个新数组

```
Dim arr() as Long
```

```
ReDim ar() '创建新数组 "ar" - "ReDim ar()" 的作用类似于 "Dim ar()"
```

## 第29.2节：操作数组，而非操作区域

Office博客 - Excel VBA性能编码最佳实践通常，通过尽可能

避免使用Range，可以获得最佳性能。在此示例中，我们将整个Range对象读入数组，对数组中的每个数字进行平方运算，然后将数组返回给Range。

这只访问了Range两次，而循环则会对读写操作访问20次。

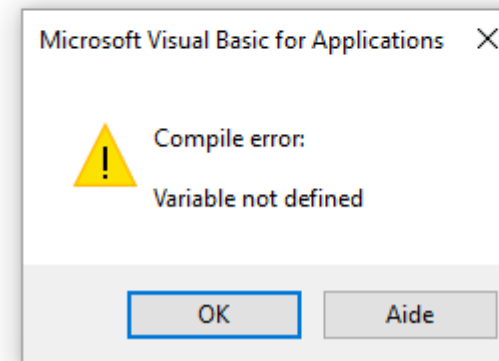
```
Option Explicit  
Sub WorkWithArrayExample()
```

```
    Dim DataRange As Variant  
    Dim Irow As Long  
    Dim Icol As Integer  
    DataRange = ActiveSheet.Range("A1:A10").Value ' 一次性从Excel表格中读取所有值，存入数组
```

```
    For Irow = LBound(DataRange,1) To UBound(DataRange, 1) ' 获取行数。  
        For Icol = LBound(DataRange,2) To UBound(DataRange, 2) ' 获取列数。  
            DataRange(Irow, Icol) = DataRange(Irow, Icol) * DataRange(Irow, Icol) ' 单元格值的平方  
        Next Icol  
    下一行
```

## Option Explicit

```
Sub Test()  
    Dim my_variable As Integer  
    my_variable = 12  
    MsgBox "My Variable is : " & myvariable  
End Sub
```



### Note on Option Explicit and Arrays ([Declaring a Dynamic Array](#)):

You can use the ReDim statement to declare an array implicitly within a procedure.

- Be careful not to misspell the name of the array when you use the ReDim statement
- Even if the Option Explicit statement is included in the module, a new array will be created

```
Dim arr() as Long
```

```
ReDim ar() 'creates new array "ar" - "ReDim ar()" acts like "Dim ar()"
```

## Section 29.2: Work with Arrays, Not With Ranges

Office Blog - Excel VBA Performance Coding Best Practices

Often, best performance is achieved by avoiding the use of Range as much as possible. In this example we read in an entire Range object into an array, square each number in the array, and then return the array back to the Range. This accesses Range only twice, whereas a loop would access it 20 times for the read/writes.

```
Option Explicit  
Sub WorkWithArrayExample()
```

```
    Dim DataRange As Variant  
    Dim Irow As Long  
    Dim Icol As Integer  
    DataRange = ActiveSheet.Range("A1:A10").Value ' read all the values at once from the Excel grid, put into an array
```

```
    For Irow = LBound(DataRange,1) To UBound(DataRange, 1) ' Get the number of rows.  
        For Icol = LBound(DataRange,2) To UBound(DataRange, 2) ' Get the number of columns.  
            DataRange(Irow, Icol) = DataRange(Irow, Icol) * DataRange(Irow, Icol) ' cell.value^2  
        Next Icol  
    Next Irow
```

ActiveSheet.Range("A1:A10").Value = DataRange ' 一次性将所有结果写回到区域

End Sub

更多带有定时示例的技巧和信息可以在Charles Williams的《编写高效VBA用户自定义函数（第1部分）》以及该系列的其他文章中找到。

### 第29.3节：在宏执行期间切换属性

在任何编程语言中，避免过早优化是最佳实践。但是，如果测试显示你的代码运行过慢，通过在运行时关闭应用程序的一些属性，可能会获得一定的速度提升。将以下代码添加到标准模块中：

```
Public Sub SpeedUp( _
SpeedUpOn As Boolean, _
    Optional xlCalc As XlCalculation = xlCalculationAutomatic _
)
    With Application
        If SpeedUpOn Then
            .ScreenUpdating = False
            .Calculation = xlCalculationManual
            .EnableEvents = False
            .DisplayStatusBar = False '如果你不显示任何消息
            ActiveSheet.DisplayPageBreaks = False '注意这是工作表级别的设置
        Else
            .ScreenUpdating = True
            .Calculation = xlCalc
            .EnableEvents = True
            .DisplayStatusBar = True
            ActiveSheet.DisplayPageBreaks = True
        End If
    End With
End Sub
```

更多信息请参见Office博客 - Excel VBA性能编码最佳实践

并且只需在宏的开始和结束调用它：

```
Public Sub SomeMacro
    '存储初始的“计算”状态
    Dim xlCalc As XlCalculation
    xlCalc = Application.Calculation

    SpeedUp True

    'code here ...

    '通过给第二个参数赋值，初始的“计算”状态得以恢复
    '否则它被设置为“xlCalculationAutomatic”
    SpeedUp False, xlCalc
End Sub
```

虽然这些在很大程度上可以被视为常规Public Sub过程的“增强”，但对于更改一个或多个工作表上数值的Worksheet\_Change和Workbook\_SheetChange私有事件宏，禁用事件处理（Application.EnableEvents = False）应被视为强制要求。未禁用事件触发将导致事件宏在数值更改时递归运行，可能导致工作簿“冻结”。请记住在离开事件宏之前重新启用事件，可能通过“安全退出”错误处理程序实现。

ActiveSheet.Range("A1:A10").Value = DataRange ' writes all the results back to the range at once

End Sub

More tips and info with timed examples can be found in Charles Williams's Writing efficient VBA UDFs (Part 1) and other articles in the series.

### Section 29.3: Switch off properties during macro execution

It is best practice in any programming language to **avoid premature optimization**. However, if testing reveals that your code is running too slowly, you may gain some speed by switching off some of the application’s properties while it runs. Add this code to a standard module:

```
Public Sub SpeedUp( _
    SpeedUpOn As Boolean, _
    Optional xlCalc As XlCalculation = xlCalculationAutomatic _
)
    With Application
        If SpeedUpOn Then
            .ScreenUpdating = False
            .Calculation = xlCalculationManual
            .EnableEvents = False
            .DisplayStatusBar = False 'in case you are not showing any messages
            ActiveSheet.DisplayPageBreaks = False 'note this is a sheet-level setting
        Else
            .ScreenUpdating = True
            .Calculation = xlCalc
            .EnableEvents = True
            .DisplayStatusBar = True
            ActiveSheet.DisplayPageBreaks = True
        End If
    End With
End Sub
```

More info on Office Blog - Excel VBA Performance Coding Best Practices

And just call it at beginning and end of macros:

```
Public Sub SomeMacro
    'store the initial "calculation" state
    Dim xlCalc As XlCalculation
    xlCalc = Application.Calculation

    SpeedUp True

    'code here ...

    'by giving the second argument the initial "calculation" state is restored
    'otherwise it is set to 'xlCalculationAutomatic'
    SpeedUp False, xlCalc
End Sub
```

While these can largely be considered "enhancements" for regular Public Sub procedures, disabling event handling with Application.EnableEvents = False should be considered mandatory for Worksheet\_Change and Workbook\_SheetChange private event macros that change values on one or more worksheets. Failure to disable event triggers will cause the event macro to recursively run on top of itself when a value changes and can lead to a "frozen" workbook. Remember to turn events back on before leaving the event macro, possibly through a "safe exit" error handler.

```
Option Explicit

Private Sub Worksheet_Change(ByVal Target As Range)
    如果 Intersect(Target, Range("A:A")) 不为 Nothing, 则
        On Error GoTo bm_Safe_Exit
Application.EnableEvents = False

        '可能更改工作表上数值的代码写在这里

    End If
bm_Safe_Exit:
Application.EnableEvents = True
End Sub
```

有一点需要注意：虽然禁用这些设置会提升运行时间，但可能会使调试应用程序变得更加困难。如果代码不正常工作，请注释掉SpeedUp True调用，直到找到问题所在。

这点尤其重要，如果你在工作表中写入单元格后，再从工作表函数中读取计算结果，因为xlCalculationManual会阻止工作簿计算。为避免禁用SpeedUp，你可能需要在特定点加入Application.Calculate以运行计算。

注意： 由于这些是Application本身的属性，您需要确保在宏退出之前再次启用它们。这使得使用错误处理程序并避免多个退出点（即 End 或 Unload Me）特别重要。

使用错误处理：

```
Public Sub SomeMacro()
    '保存初始的"计算"状态
    Dim xlCalc As XlCalculation
    xlCalc = Application.Calculation

    On Error GoTo Handler
    SpeedUp True

    'code here ...
i = 1 / 0
CleanExit:
SpeedUp False, xlCalc
Exit Sub
Handler:
    '处理错误
    简历 CleanExit
结束子程序
```

第29.4节：在可用时使用VB常量

```
如果 MsgBox("点击确定") = vbOK 则

可以替代

如果 MsgBox("点击确定") = 1 则
```

以提高可读性。

```
Option Explicit

Private Sub Worksheet_Change(ByVal Target As Range)
    If Not Intersect(Target, Range("A:A")) Is Nothing Then
        On Error GoTo bm_Safe_Exit
        Application.EnableEvents = False

        'code that may change a value on the worksheet goes here

    End If
bm_Safe_Exit:
    Application.EnableEvents = True
End Sub
```

One caveat: While disabling these settings will improve run time, they may make debugging your application much more difficult. If your code is *not* functioning correctly, comment out the SpeedUp True call until you figure out the problem.

This is particularly important if you are writing to cells in a worksheet and then reading back in calculated results from worksheet functions since the xlCalculationManual prevents the workbook from calculating. To get around this without disabling SpeedUp, you may want to include Application.Calculate to run a calculation at specific points.

NOTE: Since these are properties of the Application itself, you need to ensure that they are enabled again before your macro exits. This makes it particularly important to use error handlers and to avoid multiple exit points (i.e. End or Unload Me).

With error handling:

```
Public Sub SomeMacro()
    'store the initial "calculation" state
    Dim xlCalc As XlCalculation
    xlCalc = Application.Calculation

    On Error GoTo Handler
    SpeedUp True

    'code here ...
i = 1 / 0
CleanExit:
    SpeedUp False, xlCalc
Exit Sub
Handler:
    'handle error
    Resume CleanExit
End Sub
```

Section 29.4: Use VB constants when available

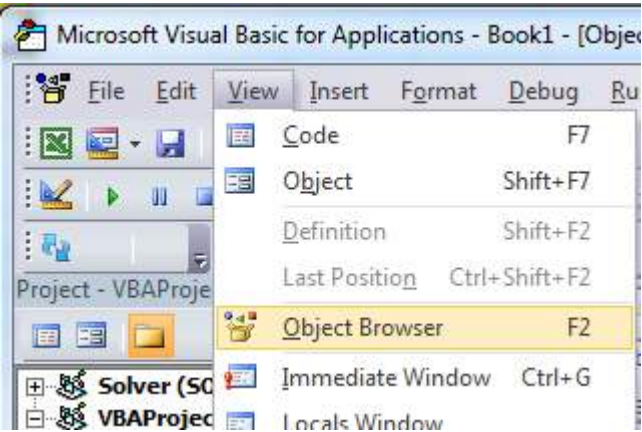
```
If MsgBox("Click OK") = vbOK Then

can be used in place of

If MsgBox("Click OK") = 1 Then
```

in order to improve readability.

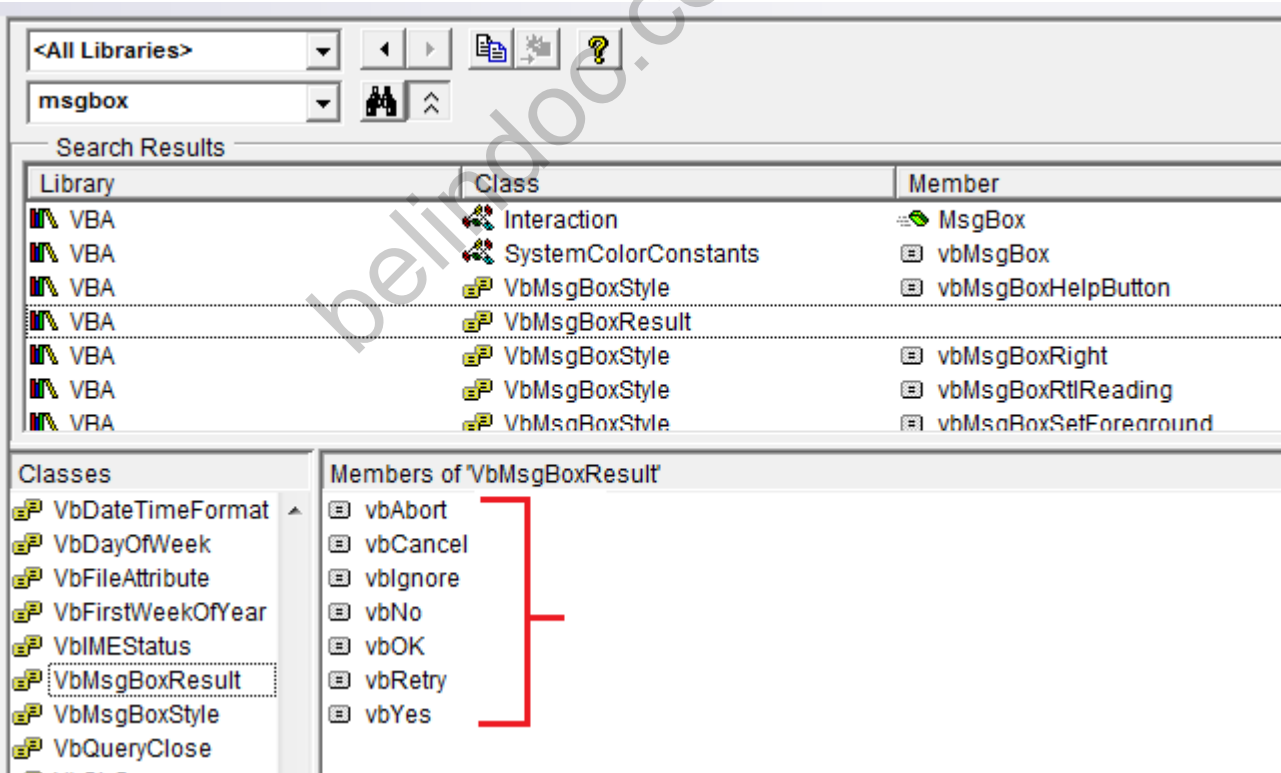
使用 对象浏览器 查找可用的VB常量。 查看 → 对象浏览器 或 F2 从VB编辑器中。



输入要搜索的类



查看可用成员



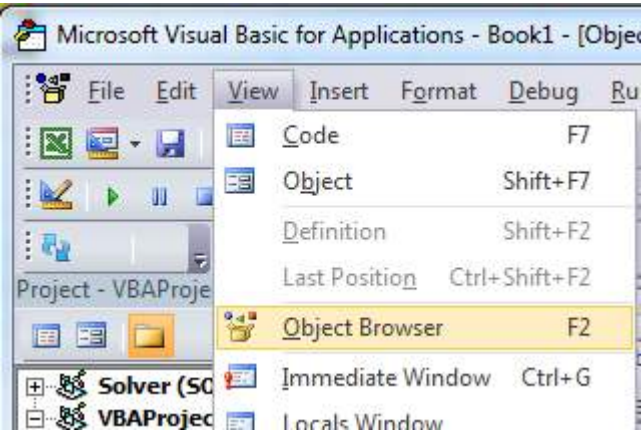
## 第29.5节：避免使用SELECT或ACTIVATE

你几乎不会想在代码中使用SELECT或Activate，但某些Excel方法确实要求先激活工作表或工作簿，才能按预期工作。

如果你刚开始学习VBA，通常会建议你使用宏录制器记录操作，然后查看代码。例如，我录制了在Sheet2的单元格D3中输入数值的操作，宏代码如下：

```
Option Explicit
Sub Macro1()
```

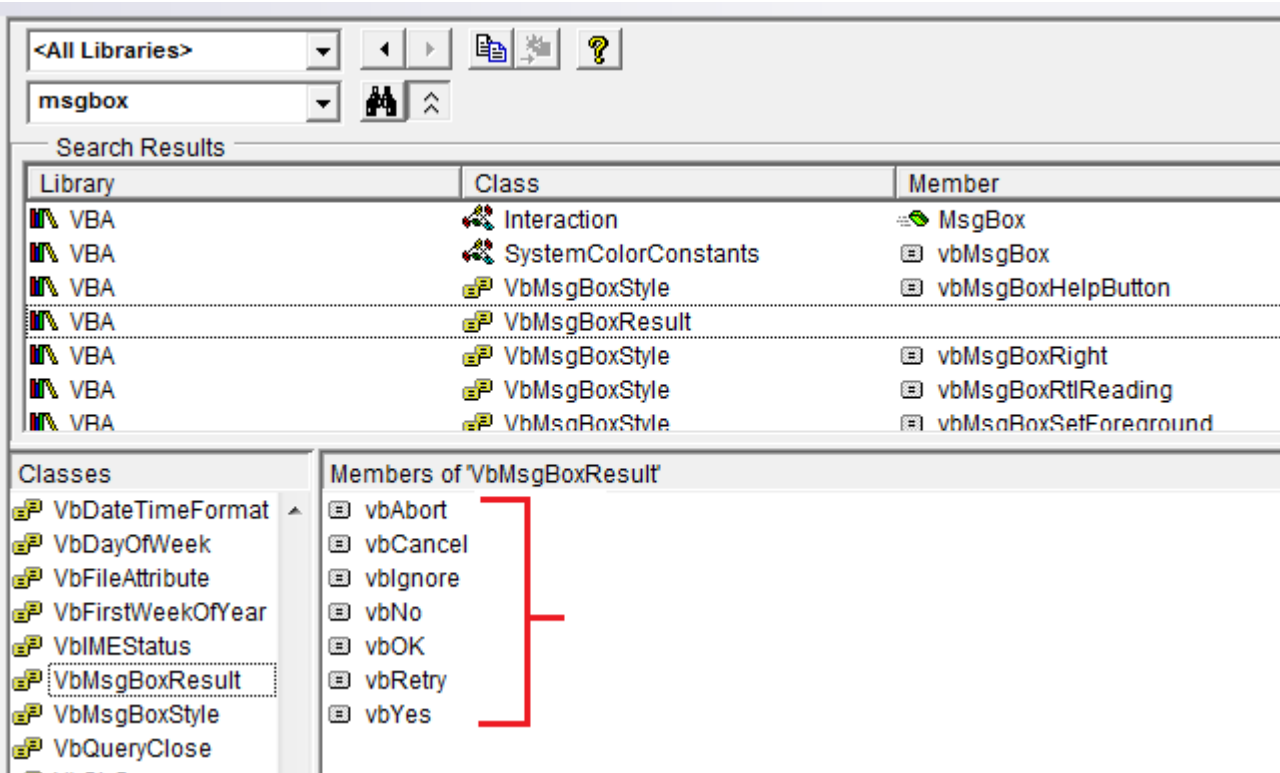
Use *Object Browser* to find available VB constants. *View* → *Object Browser* or F2 from VB Editor.



Enter class to search



View members available



## Section 29.5: Avoid using SELECT or ACTIVATE

It is **very** rare that you'll ever want to use **SELECT** or **Activate** in your code, but some Excel methods do require a worksheet or workbook to be activated before they'll work as expected.

If you're just starting to learn VBA, you'll often be suggested to record your actions using the macro recorder, then go look at the code. For example, I recorded actions taken to enter a value in cell D3 on Sheet2, and the macro code looks like this:

```
Option Explicit
Sub Macro1()
```



```
'
' Macro1 宏
'

Sheets("Sheet2").Select
Range("D3").Select
ActiveCell.FormulaR1C1 = "3.1415" '(见下方**注释)
Range("D4").Select
End Sub
```

但请记住，宏录制器会为你的每一个（用户）操作生成一行代码。这包括点击工作表标签选择Sheet2（Sheets("Sheet2").Select）、点击单元格D3输入数值前的选择操作（Range("D3").Select），以及按回车键（实际上是“选择”当前选中单元格下方的单元格：Range("D4").Select）。

使用.Select存在多个问题：

- **工作表并不总是被指定。如果录制时没有切换工作表，代码将在不同的活动工作表上产生不同的结果。**
- **.Select() 执行缓慢。即使将 Application.ScreenUpdating 设置为 False，**  
这也是一个不必要的操作。
- **.Select() 是不受控的。**如果 Application.ScreenUpdating 保持为 True，Excel 实际上会选择单元格、工作表、窗体.....无论你是否正在操作什么。这对眼睛来说很有压力，看起来也非常不舒服。
- **.Select() 会触发监听器。**这已经有点高级了，但除非有相应的处理，否则像 Worksheet\_SelectionChange() 这样的函数会被触发。

当你在 VBA 中编程时，所有的“输入”操作（即 SELECT 语句）都不再必要。你的代码可以简化为一条语句来给单元格赋值：

```
'--- 好的
ActiveWorkbook.Sheets("Sheet2").Range("D3").Value = 3.1415

'--- 更好
Dim myWB As Workbook
Dim myWS As Worksheet
Dim myCell As Range

Set myWB = ThisWorkbook '*** 见注释2
Set myWS = myWB.Sheets("Sheet2")
Set myCell = myWS.Range("D3")

myCell.Value = 3.1415
```

（上面“更好”的示例展示了使用中间变量来分离单元格引用的不同部分。）  
良好的示例总是能正常工作，但在代码模块较长时可能非常繁琐，且如果引用拼写错误，调试起来会更困难。

**\*\*注意：**宏录制器对你输入的数据类型做了许多假设，在本例中是将字符串值作为公式输入以创建该值。你的代码不必这样做，可以像上面所示直接将数值赋给单元格。

**\*\*注意2：**推荐的做法是将本地工作簿变量设置为ThisWorkbook而不是ActiveWorkbook（除非你明确需要）。原因是你的宏通常需要/使用VBA代码所在工作簿中的资源，并且不会查看该工作簿之外的内容——除非你

```
'
' Macro1 Macro
'

Sheets("Sheet2").Select
Range("D3").Select
ActiveCell.FormulaR1C1 = "3.1415" '(see **note below)
Range("D4").Select
End Sub
```

Remember though, the macro recorder creates a line of code for EACH of your (user) actions. This includes clicking on the worksheet tab to select Sheet2 (Sheets("Sheet2").Select), clicking on cell D3 before entering the value (Range("D3").Select), and using the Enter key (which is effectively "selecting" the cell below the currently selected cell: Range("D4").Select).

There are multiple issues with using .Select here:

- **The worksheet is not always specified.** This happens if you don't switch worksheets while recording, and means that the code will yield different results for different active worksheets.
- **.Select() is slow.** Even if Application.ScreenUpdating is set to False, this is an unnecessary operation to be processed.
- **.Select() is unruly.** If Application.ScreenUpdating is left to True, Excel will actually select the cells, the worksheet, the form... whatever it is you're working with. This is stressful to the eyes and really unpleasant to watch.
- **.Select() will trigger listeners.** This is a bit advanced already, but unless worked around, functions like Worksheet\_SelectionChange() will be triggered.

When you're coding in VBA, all of the "typing" actions (i.e. SELECT statements) are no longer necessary. Your code may be reduced to a single statement to put the value in the cell:

```
'--- GOOD
ActiveWorkbook.Sheets("Sheet2").Range("D3").Value = 3.1415

'--- BETTER
Dim myWB As Workbook
Dim myWS As Worksheet
Dim myCell As Range

Set myWB = ThisWorkbook '*** see NOTE2
Set myWS = myWB.Sheets("Sheet2")
Set myCell = myWS.Range("D3")

myCell.Value = 3.1415
```

(The BETTER example above shows using intermediate variables to separate different parts of the cell reference. The GOOD example will always work just fine, but can be very cumbersome in much longer code modules and more difficult to debug if one of the references is mistyped.)

**\*\*NOTE:** the macro recorder makes many assumptions about the type of data you're entering, in this case entering a string value as a formula to create the value. Your code doesn't have to do this and can simply assign a numerical value directly to the cell as shown above.

**\*\*NOTE2:** the recommended practice is to set your local workbook variable to ThisWorkbook instead of ActiveWorkbook (unless you explicitly need it). The reason is your macro will generally need/use resources in whatever workbook the VBA code originates and will NOT look outside of that workbook -- again, unless you

明确指示代码操作另一个工作簿。当你在Excel中打开多个工作簿时，*ActiveWorkbook*是当前焦点所在的工作簿，这可能与VBA编辑器中查看的工作簿不同。所以你可能认为自己在操作一个工作簿，实际上引用的是另一个。ThisWorkbook指的是包含正在执行代码的工作簿。

## 第29.6节：始终定义并设置对所有工作簿和工作表的引用

当处理多个打开的工作簿时，每个工作簿可能包含多个工作表，最好定义并设置对所有工作簿和工作表的引用。

不要依赖ActiveWorkbook或ActiveSheet，因为它们可能被用户更改。

下面的代码示例演示了如何将“Raw\_Data”工作表中“Data.xlsx”工作簿的一个区域复制到“Refined\_Data”工作表中“Results.xlsx”工作簿。

该过程还演示了如何在不使用SELECT方法的情况下进行复制和粘贴。

```
Option Explicit

Sub CopyRanges_BetweenShts()

    Dim wbSrc As Workbook
    Dim wbDest As Workbook
    Dim shtCopy As Worksheet
    Dim shtPaste As Worksheet

    ' 通过名称设置对所有工作簿的引用，不依赖于 ActiveWorkbook
    Set wbSrc = Workbooks("Data.xlsx")
    Set wbDest = Workbooks("Results.xlsx")

    ' 通过名称设置对所有工作表的引用，不依赖于 ActiveSheet
    Set shtCopy = wbSrc.Sheet1 '// "Raw_Data" 工作表
    Set shtPaste = wbDest.Sheet2 '// "Refined_Data" 工作表

    ' 从 "Data" 工作簿复制范围到 "Results" 工作簿，且不使用 Select
    shtCopy.Range("A1:C10").Copy _
    Destination:=shtPaste.Range("A1")

End Sub
```

## 第29.7节：使用描述性变量命名

代码中使用描述性名称和结构有助于减少注释的必要性

```
Dim ductWidth As Double
Dim ductHeight As Double
Dim ductArea As Double

ductArea = ductWidth * ductHeight
```

优于

```
Dim a, w, h

a = w * h
```

explicitly direct your code to work with another workbook. When you have multiple workbooks open in Excel, the *ActiveWorkbook* is the one with the focus *which may be different from the workbook being viewed in your VBA Editor*. So you think you're executing in a one workbook when you're really referencing another. ThisWorkbook refers to the workbook containing the code being executed.

## Section 29.6: Always define and set references to all Workbooks and Sheets

When working with multiple open Workbooks, each of which may have multiple Sheets, it's safest to define and set reference to all Workbooks and Sheets.

**Don't rely** on ActiveWorkbook or ActiveSheet as they might be changed by the user.

The following code example demonstrates how to copy a range from “Raw\_Data” sheet in the “Data.xlsx” workbook to “Refined\_Data” sheet in the “Results.xlsx” workbook.

The procedure also demonstrates how to copy and paste without using the **SELECT** method.

```
Option Explicit

Sub CopyRanges_BetweenShts()

    Dim wbSrc As Workbook
    Dim wbDest As Workbook
    Dim shtCopy As Worksheet
    Dim shtPaste As Worksheet

    ' set reference to all workbooks by name, don't rely on ActiveWorkbook
    Set wbSrc = Workbooks("Data.xlsx")
    Set wbDest = Workbooks("Results.xlsx")

    ' set reference to all sheets by name, don't rely on ActiveSheet
    Set shtCopy = wbSrc.Sheet1 '// "Raw_Data" sheet
    Set shtPaste = wbDest.Sheet2 '// "Refined_Data") sheet

    ' copy range from "Data" workbook to "Results" workbook without using Select
    shtCopy.Range("A1:C10").Copy _
    Destination:=shtPaste.Range("A1")

End Sub
```

## Section 29.7: Use descriptive variable naming

Descriptive names and structure in your code help make comments unnecessary

```
Dim ductWidth As Double
Dim ductHeight As Double
Dim ductArea As Double

ductArea = ductWidth * ductHeight
```

is better than

```
Dim a, w, h

a = w * h
```

当你将数据从一个地方复制到另一个地方时，无论是单元格、区域、工作表还是工作簿，这尤其有用。可以使用如下名称来帮助自己：

```
Dim myWB As Workbook
Dim srcWS As Worksheet
Dim destWS As Worksheet
Dim srcData As Range
Dim destData As Range

Set myWB = ActiveWorkbook
Set srcWS = myWB.Sheets("Sheet1")
Set destWS = myWB.Sheets("Sheet2")
Set srcData = srcWS.Range("A1:A10")
Set destData = destWS.Range("B11:B20")
destData = srcData
```

如果在一行中声明多个变量，确保为每个变量指定类型，例如：

```
Dim ductWidth As Double, ductHeight As Double, ductArea As Double
```

下面的写法只会声明最后一个变量，前面的变量将保持为Variant类型：

```
Dim ductWidth, ductHeight, ductArea As Double
```

## 第29.8节：记录你的工作

记录你的工作是个好习惯，尤其是在为动态工作负载编写代码时。好的注释应该解释代码为什么要做某事，而不是代码做了什么。

```
Function Bonus(EmployeeTitle as String) as Double
    If EmployeeTitle = "Sales" Then
        奖金 = 0      '销售代表获得佣金而非奖金
    否则
        奖金 = 0.10
    End If
End Function
```

如果你的代码晦涩难懂，需要注释来解释其功能，考虑重写代码使其更清晰，而不是通过注释来解释。例如，以下写法：

```
Sub 复制销售数字
    Dim 包含周末 作为布尔值

    '布尔值可以作为整数来评估，True 为 -1，False 为 0。
    '这里用来调整范围，如果包含周末，则从 5 行调整到 7 行。
    Range("A1:A" & 5 - (包含周末 * 2)).Copy
    Range("B1").PasteSpecial
End Sub
```

使代码更清晰易懂，例如：

```
Sub 复制销售数字
    Dim 包含周末 作为布尔值
    Dim 一周天数 作为整数

    如果 包含周末 那么
        一周天数 = 7
```

This is especially helpful when you are copying data from one place to another, whether it's a cell, range, worksheet, or workbook. Help yourself by using names such as these:

```
Dim myWB As Workbook
Dim srcWS As Worksheet
Dim destWS As Worksheet
Dim srcData As Range
Dim destData As Range

Set myWB = ActiveWorkbook
Set srcWS = myWB.Sheets("Sheet1")
Set destWS = myWB.Sheets("Sheet2")
Set srcData = srcWS.Range("A1:A10")
Set destData = destWS.Range("B11:B20")
destData = srcData
```

If you declare multiple variables in one line make sure to specify a type for *every* variable like:

```
Dim ductWidth As Double, ductHeight As Double, ductArea As Double
```

The following will only declare the last variable and the first ones will remain Variant:

```
Dim ductWidth, ductHeight, ductArea As Double
```

## Section 29.8: Document Your Work

It's good practice to document your work for later use, especially if you are coding for a dynamic workload. Good comments should explain why the code is doing something, not what the code is doing.

```
Function Bonus(EmployeeTitle as String) as Double
    If EmployeeTitle = "Sales" Then
        Bonus = 0      'Sales representatives receive commission instead of a bonus
    Else
        Bonus = .10
    End If
End Function
```

If your code is so obscure that it requires comments to explain what it is doing, consider rewriting it to be more clear instead of explaining it through comments. For example, instead of:

```
Sub CopySalesNumbers
    Dim IncludeWeekends as Boolean

    'Boolean values can be evaluated as an integer, -1 for True, 0 for False.
    'This is used here to adjust the range from 5 to 7 rows if including weekends.
    Range("A1:A" & 5 - (IncludeWeekends * 2)).Copy
    Range("B1").PasteSpecial
End Sub
```

Clarify the code to be easier to follow, such as:

```
Sub CopySalesNumbers
    Dim IncludeWeekends as Boolean
    Dim DaysinWeek as Integer

    If IncludeWeekends Then
        DaysinWeek = 7
```

```
Else
一周天数 = 5
End If
Range("A1:A" & DaysinWeek).Copy
Range("B1").PasteSpecial
End Sub
```

## 第29.9节：错误处理

良好的错误处理可以防止最终用户看到VBA运行时错误，并帮助开发人员轻松诊断和纠正错误。

VBA中有三种主要的错误处理方法，其中两种除非代码中特别要求，否则应避免用于分发的程序。

*On Error GoTo 0 '避免使用*

或

*On Error Resume Next '避免使用*

推荐使用：

*On Error GoTo <line> '推荐使用*

### On Error GoTo 0

如果您的代码中未设置错误处理，On Error GoTo 0 是默认的错误处理程序。在此模式下，任何运行时错误都会弹出典型的 VBA 错误消息，允许您结束代码或进入调试模式，以识别错误来源。在编写代码时，这种方法是最简单且最有用的，但对于分发给最终用户的代码，应始终避免使用，因为这种方法非常难看且最终用户难以理解。

### On Error Resume Next

On Error Resume Next 会导致 VBA 忽略运行时抛出的所有错误，直到错误处理程序被更改为止，适用于错误调用之后的所有代码行。在非常特定的情况下，这行代码是有用的，但应避免在这些情况之外使用。例如，当从 Excel 宏启动一个独立程序时，On Error Resume Next 调用是有用的，如果你不确定该程序是否已经打开：

```
'在此示例中，我们使用 On Error Resume Next 调用打开了一个 PowerPoint 实例
Dim PPApP As PowerPoint.Application
Dim PPPres As PowerPoint.Presentation
Dim PPSlide As PowerPoint.Slide

'如果PowerPoint未运行则打开，否则选择活动实例
On Error Resume Next
Set PPApP = GetObject(, "PowerPoint.Application")
On Error GoTo ErrHandler
If PPApP Is Nothing Then
'打开 PowerPoint
Set PPApP = CreateObject("PowerPoint.Application")
PPApP.Visible = True
End If
```

```
Else
DaysinWeek = 5
End If
Range("A1:A" & DaysinWeek).Copy
Range("B1").PasteSpecial
End Sub
```

## Section 29.9: Error Handling

Good error handling prevents end users from seeing VBA runtime errors and helps the developer easily diagnose and correct errors.

There are three main methods of Error Handling in VBA, two of which should be avoided for distributed programs unless specifically required in the code.

*On Error GoTo 0 'Avoid using*

or

*On Error Resume Next 'Avoid using*

Prefer using:

*On Error GoTo <line> 'Prefer using*

### On Error GoTo 0

If no error handling is set in your code, On Error GoTo 0 is the default error handler. In this mode, any runtime errors will launch the typical VBA error message, allowing you to either end the code or enter debug mode, identifying the source. While writing code, this method is the simplest and most useful, but it should always be avoided for code that is distributed to end users, as this method is very unsightly and difficult for end users to understand.

### On Error Resume Next

On Error Resume Next will cause VBA to ignore any errors that are thrown at runtime for all lines following the error call until the error handler has been changed. In very specific instances, this line can be useful, but it should be avoided outside of these cases. For example, when launching a separate program from an Excel Macro, the On Error Resume Next call can be useful if you are unsure whether or not the program is already open:

```
'In this example, we open an instance of Powerpoint using the On Error Resume Next call
Dim PPApP As PowerPoint.Application
Dim PPPres As PowerPoint.Presentation
Dim PPSlide As PowerPoint.Slide

'Open PPT if not running, otherwise select active instance
On Error Resume Next
Set PPApP = GetObject(, "PowerPoint.Application")
On Error GoTo ErrHandler
If PPApP Is Nothing Then
'Open PowerPoint
Set PPApP = CreateObject("PowerPoint.Application")
PPApP.Visible = True
End If
```



如果我们没有使用 **On Error Resume Next** 调用，而 PowerPoint 应用程序尚未打开，GetObject 方法将抛出错误。因此，**On Error Resume Next** 是必要的，以避免创建两个应用程序实例。

注意： 最佳实践是在不再需要 On Error Resume Next 调用时，立即 重置错误处理程序

On Error GoTo <line>

这种错误处理方法推荐用于所有分发给其他用户的代码。这允许程序员精确控制 VBA 如何处理错误，通过将代码发送到指定的行。标签可以是任意字符串（包括数字字符串），并将代码发送到后面带冒号的对应字符串。可以通过多次调用 On Error GoTo <line> 来使用多个错误处理块。下面的子程序演示了 On Error GoTo <line> 调用的语法。

注意： 必须将 Exit Sub 行放在第一个错误处理程序之上以及每个后续错误处理程序之前，以防止代码在没有错误调用的情况下自然进入该块。因此，最佳实践和可读性要求将错误处理程序放在代码块的末尾。

```
Sub YourMethodName()  
    On Error GoTo errorHandler  
    ' 在此插入代码  
    On Error GoTo secondErrorHandler  
  
    Exit Sub '退出子程序这一行是必需的，否则代码将继续  
           '运行到错误处理块，可能导致错误  
  
errorHandler:  
MsgBox "错误 " & Err.Number & ": " & Err.Description & " 于 " & _  
      VBE.ActiveCodePane.CodeModule, vbOKOnly, "错误"  
    Exit Sub  
  
secondErrorHandler:  
    If Err.Number = 424 Then '对象未找到错误（仅作参考）  
        Application.ScreenUpdating = True  
        Application.EnableEvents = True  
        Exit Sub  
    Else  
MsgBox "错误 " & Err.Number & ": " & Err.Descriptiion  
        Application.ScreenUpdating = True  
        Application.EnableEvents = True  
        Exit Sub  
    End If  
Exit Sub  
  
End Sub
```

如果你在错误处理代码中退出方法，确保进行清理：

- 撤销任何部分完成的操作
- 关闭文件
- 重置屏幕更新
- 重置计算模式
- 重置事件
- 重置鼠标指针
- 在对象实例上调用卸载方法，这些实例在End Sub之后仍然存在

Had we not used the **On Error Resume Next** call and the Powerpoint application was not already open, the GetObject method would throw an error. Thus, **On Error Resume Next** was necessary to avoid creating two instances of the application.

**Note:** It is also a best practice to *immediately* reset the error handler as soon as you no longer need the **On Error Resume Next** call

On Error GoTo <line>

This method of error handling is recommended for all code that is distributed to other users. This allows the programmer to control exactly how VBA handles an error by sending the code to the specified line. The tag can be filled with any string (including numeric strings), and will send the code to the corresponding string that is followed by a colon. Multiple error handling blocks can be used by making different calls of **On Error GoTo <line>**. The subroutine below demonstrates the syntax of an **On Error GoTo <line>** call.

**Note:** It is essential that the **Exit Sub** line is placed above the first error handler and before every subsequent error handler to prevent the code from naturally progressing into the block *without* an error being called. Thus, it is best practice for function and readability to place error handlers at the end of a code block.

```
Sub YourMethodName()  
    On Error GoTo errorHandler  
    ' Insert code here  
    On Error GoTo secondErrorHandler  
  
    Exit Sub 'The exit sub line is essential, as the code will otherwise  
           'continue running into the error handling block, likely causing an error  
  
errorHandler:  
    MsgBox "Error " & Err.Number & ": " & Err.Description & " in " & _  
      VBE.ActiveCodePane.CodeModule, vbOKOnly, "Error"  
    Exit Sub  
  
secondErrorHandler:  
    If Err.Number = 424 Then 'Object not found error (purely for illustration)  
        Application.ScreenUpdating = True  
        Application.EnableEvents = True  
        Exit Sub  
    Else  
        MsgBox "Error " & Err.Number & ": " & Err.Description  
        Application.ScreenUpdating = True  
        Application.EnableEvents = True  
        Exit Sub  
    End If  
Exit Sub  
  
End Sub
```

If you exit your method with your error handling code, ensure that you clean up:

- Undo anything that is partially completed
- Close files
- Reset screen updating
- Reset calculation mode
- Reset events
- Reset mouse pointer
- Call unload method on instances of objects, that persist after the **End Sub**



- 重置状态栏

## 第29.10节：永远不要假设工作表

即使你的所有工作都针对单个工作表，明确在代码中指定工作表仍然是一个非常好的习惯。这个习惯使得以后扩展代码，或者将部分（或全部）Sub或Function在其他地方重用变得更加容易。许多开发者养成了在代码中为工作表使用相同局部变量名的习惯，使得代码重用更加直接。

例如，以下代码是模糊的——但有效！——只要开发者不激活或切换到不同的工作表：

```
Option Explicit
Sub ShowTheTime()
    '--- 在工作表的单元格A1中显示当前时间和日期
    Cells(1, 1).Value = Now() ' 不要在没有工作表引用的情况下使用Cells !
End Sub
```

如果Sheet1处于活动状态，则Sheet1上的单元格A1将填充当前的日期和时间。但如果用户因任何原因切换工作表，则代码将更新当前活动的工作表。目标工作表不明确。

最佳做法是始终明确标识代码所引用的工作表：

```
Option Explicit
Sub ShowTheTime()
    '--- 在工作表的单元格A1中显示当前时间和日期
    Dim myWB As Workbook
    Set myWB = ThisWorkbook
    Dim timestampSH As Worksheet
    Set timestampSH = myWB.Sheets("Sheet1")
    timestampSH.Cells(1, 1).Value = Now()
End Sub
```

上述代码明确标识了工作簿和工作表。虽然看起来有些多余，但养成良好的目标引用习惯可以避免将来的问题。

## 第29.11节：避免将属性或方法的名称重新用作变量名

通常不被认为是“最佳实践”，将属性或方法的保留名称重新用作你自己的过程和变量的名称。

不良形式 - 虽然以下代码（严格来说）是合法且可运行的，但将Find方法以及Row、Column和Address属性重新用作变量名，可能导致名称歧义引发问题/冲突，而且总体上令人困惑。

```
Option Explicit

Sub find()
    Dim row As Long, column As Long
    Dim find As String, address As Range

    find = "something"

    使用 ThisWorkbook.Worksheets("Sheet1").Cells
```

- Reset status bar

## Section 29.10: Never Assume The Worksheet

Even when all your work is directed at a single worksheet, it's still a very good practice to explicitly specify the worksheet in your code. This habit makes it much easier to expand your code later, or to lift parts (or all) of a Sub or Function to be re-used someplace else. Many developers establish a habit of (re)using the same local variable name for a worksheet in their code, making re-use of that code even more straightforward.

As an example, the following code is ambiguous -- but works! -- as long the developer doesn't activate or change to a different worksheet:

```
Option Explicit
Sub ShowTheTime()
    '--- displays the current time and date in cell A1 on the worksheet
    Cells(1, 1).Value = Now() ' don't refer to Cells without a sheet reference!
End Sub
```

If Sheet1 is active, then cell A1 on Sheet1 will be filled with the current date and time. But if the user changes worksheets for any reason, then the code will update whatever the worksheet is currently active. The destination worksheet is ambiguous.

The best practice is to always identify which worksheet to which your code refers:

```
Option Explicit
Sub ShowTheTime()
    '--- displays the current time and date in cell A1 on the worksheet
    Dim myWB As Workbook
    Set myWB = ThisWorkbook
    Dim timestampSH As Worksheet
    Set timestampSH = myWB.Sheets("Sheet1")
    timestampSH.Cells(1, 1).Value = Now()
End Sub
```

The code above is clear in identifying both the workbook and the worksheet. While it may seem like overkill, creating a good habit concerning target references will save you from future problems.

## Section 29.11: Avoid re-purposing the names of Properties or Methods as your variables

It is generally not considered 'best practice' to re-purpose the reserved names of Properties or Methods as the name(s) of your own procedures and variables.

**Bad Form** - While the following is (strictly speaking) legal, working code the re-purposing of the Find method as well as the Row, Column and Address properties can cause problems/conflicts with name ambiguity and is just plain confusing in general.

```
Option Explicit

Sub find()
    Dim row As Long, column As Long
    Dim find As String, address As Range

    find = "something"

    With ThisWorkbook.Worksheets("Sheet1").Cells
```

```

    设置 address = .SpecialCells(xlCellTypeLastCell)
    row = .find(what:=find, after:=address).row      '注意 .row 不是大写
    column = .find(what:=find, after:=address).column '注意 .column 不是大写

    Debug.Print "第一个 'something' 位于 " & .Cells(row, column).address(0, 0)
End With
End Sub
```

良好形式 - 通过将所有保留字重命名为与原词相近但唯一的名称，避免了任何潜在的命名冲突。

```

Option Explicit

子程序 myFind()
    声明 rw 作为 长整型, col 作为 长整型
    声明 wht 作为 字符串, lastCell 作为 范围

    wht = "something"

    使用 ThisWorkbook.Worksheets("Sheet1").Cells
    设置 lastCell = .SpecialCells(xlCellTypeLastCell)
    rw = .Find(What:=wht, After:=lastCell).Row      '☒ 注意 .Find 和 .Row
    col = .Find(What:=wht, After:=lastCell).Column  '☒ .Find 和 .Column

    Debug.Print "第一个 'something' 位于 " & .Cells(rw, col).Address(0, 0)
End With
End Sub
```

虽然有时你可能想有意地重写一个标准方法或属性以符合你自己的规范，但这种情况很少见。大多数情况下，避免重用保留名称来定义你自己的结构。

## 第29.12节：避免在Excel中使用ActiveCell或ActiveSheet

使用ActiveCell或ActiveSheet如果代码在错误的位置执行（无论出于何种原因），可能会导致错误。

```

ActiveCell.Value = "Hello"
'会将“Hello”放入当前选中的单元格
Cells(1, 1).Value = "Hello"
'将始终在当前选定工作表的A1单元格中放置“Hello”

ActiveSheet.Cells(1, 1).Value = "Hello"
'将在当前选定工作表的A1单元格中放置“Hello”
Sheets("MySheetName").Cells(1, 1).Value = "Hello"
'将始终在名为“MySheetName”的工作表的A1单元格中放置“Hello”
```

- 使用Active\*可能会在长时间运行的宏中引发问题，如果用户感到无聊并点击了另一个工作表或打开了另一个工作簿。
- 如果代码打开或创建了另一个工作簿，可能会引发问题。
- 如果代码使用了Sheets("MyOtherSheet").Select，且你忘记了在开始读取或写入之前所在的工作表，可能会引发问题。

```

    Set address = .SpecialCells(xlCellTypeLastCell)
    row = .find(what:=find, after:=address).row      '< note .row not capitalized
    column = .find(what:=find, after:=address).column '< note .column not capitalized

    Debug.Print "The first 'something' is in " & .Cells(row, column).address(0, 0)
End With
End Sub
```

Good Form - With all of the reserved words renamed into close but unique approximations of the originals, any potential naming conflicts have been avoided.

```

Option Explicit

Sub myFind()
    Dim rw As Long, col As Long
    Dim wht As String, lastCell As Range

    wht = "something"

    With ThisWorkbook.Worksheets("Sheet1").Cells
        Set lastCell = .SpecialCells(xlCellTypeLastCell)
        rw = .Find(What:=wht, After:=lastCell).Row      '◀ note .Find and .Row
        col = .Find(What:=wht, After:=lastCell).Column  '◀ .Find and .Column

        Debug.Print "The first 'something' is in " & .Cells(rw, col).Address(0, 0)
    End With
End Sub
```

While there may come a time when you want to intentionally rewrite a standard method or property to your own specifications, those situations are few and far between. For the most part, stay away from reusing reserved names for your own constructs.

## Section 29.12: Avoid using ActiveCell or ActiveSheet in Excel

Using ActiveCell or ActiveSheet can be source of mistakes if (for any reason) the code is executed in the wrong place.

```

ActiveCell.Value = "Hello"
'will place "Hello" in the cell that is currently selected
Cells(1, 1).Value = "Hello"
'will always place "Hello" in A1 of the currently selected sheet

ActiveSheet.Cells(1, 1).Value = "Hello"
'will place "Hello" in A1 of the currently selected sheet
Sheets("MySheetName").Cells(1, 1).Value = "Hello"
'will always place "Hello" in A1 of the sheet named "MySheetName"
```

- The use of Active\* can create problems in long running macros if your user gets bored and clicks on another worksheet or opens another workbook.
- It can create problems if your code opens or creates another workbook.
- It can create problems if your code uses Sheets("MyOtherSheet").Select and you've forgotten which sheet you were on before you start reading from or writing to it.

第29.13节：WorksheetFunction对象的执行速度快于等效的用户自定义函数（UDF）

VBA是在运行时编译的，这对其性能有很大的负面影响，所有内置的功能都会更快，尽量使用它们。

举例来说，我正在比较SUM和COUNTIF函数，但你可以用它来解决任何可以用

WorksheetFunctions解决的问题。

对于这些函数的初步尝试是遍历范围并逐个单元格处理（使用范围）：

```
Sub UseRange()  
    Dim rng 作为 Range  
    Dim Total 作为 Double  
    Dim CountLessThan01 作为 Long  
  
    Total = 0  
    CountLessThan01 = 0  
    For Each rng 在 Sheets(1).Range("A1:A100")  
        Total = Total + rng.Value2  
        If rng.Value < 0.1 Then  
            CountLessThan01 = CountLessThan01 + 1  
        End If  
    Next rng  
    Debug.Print Total & ", " & CountLessThan01  
End Sub
```

一种改进方法是将范围值存储在数组中并进行处理：

```
Sub 使用数组()  
    Dim 要汇总的数据 As Variant  
    Dim i As Long  
    Dim 总计 As Double  
    Dim 小于0.1的计数 As Long  
  
    要汇总的数据 = Sheets(1).Range("A1:A100").Value2 '比.Value更快  
    总计 = 0  
    CountLessThan01 = 0  
    For i = 1 To 100  
        总计 = 总计 + 要汇总的数据(i, 1)  
        If 要汇总的数据(i, 1) < 0.1 Then  
            小于0.1的计数 = 小于0.1的计数 + 1  
        End If  
    Next i  
    Debug.Print Total & ", " & CountLessThan01  
End Sub
```

但是，你无需编写任何循环，可以使用Application.Worksheetfunction，这对于执行简单公式非常方便：

```
Sub UseWorksheetFunction()  
    Dim Total As Double  
    Dim CountLessThan01 As Long  
  
    With Application.WorksheetFunction  
        Total = .Sum(Sheets(1).Range("A1:A100"))  
        CountLessThan01 = .CountIf(Sheets(1).Range("A1:A100"), "<0.1")  
    End With
```

Section 29.13: WorksheetFunction object executes faster than a UDF equivalent

VBA is compiled in run-time, which has a huge negative impact on it's performance, everything built-in will be faster, try to use them.

As an example I'm comparing SUM and COUNTIF functions, but you can use if for anything you can solve with WorksheetFunctions.

A first attempt for those would be to loop through the range and process it cell by cell (using a range):

```
Sub UseRange()  
    Dim rng as Range  
    Dim Total As Double  
    Dim CountLessThan01 As Long  
  
    Total = 0  
    CountLessThan01 = 0  
    For Each rng in Sheets(1).Range("A1:A100")  
        Total = Total + rng.Value2  
        If rng.Value < 0.1 Then  
            CountLessThan01 = CountLessThan01 + 1  
        End If  
    Next rng  
    Debug.Print Total & ", " & CountLessThan01  
End Sub
```

One improvement can be to store the range values in an array and process that:

```
Sub UseArray()  
    Dim DataToSummarize As Variant  
    Dim i As Long  
    Dim Total As Double  
    Dim CountLessThan01 As Long  
  
    DataToSummarize = Sheets(1).Range("A1:A100").Value2 'faster than .Value  
    Total = 0  
    CountLessThan01 = 0  
    For i = 1 To 100  
        Total = Total + DataToSummarize(i, 1)  
        If DataToSummarize(i, 1) < 0.1 Then  
            CountLessThan01 = CountLessThan01 + 1  
        End If  
    Next i  
    Debug.Print Total & ", " & CountLessThan01  
End Sub
```

But instead of writing any loop you can use Application.Worksheetfunction which is very handy for executing simple formulas:

```
Sub UseWorksheetFunction()  
    Dim Total As Double  
    Dim CountLessThan01 As Long  
  
    With Application.WorksheetFunction  
        Total = .Sum(Sheets(1).Range("A1:A100"))  
        CountLessThan01 = .CountIf(Sheets(1).Range("A1:A100"), "<0.1")  
    End With
```

```
Debug.Print Total & ", " & CountLessThan01
End Sub
```

或者，对于更复杂的计算，你甚至可以使用Application.Evaluate：

```
Sub UseEvaluate()
    Dim Total As Double
    Dim CountLessThan01 As Long

    With Application
        Total = .Evaluate("SUM(" & Sheet1.Range("A1:A100").Address( _
            external:=True) & ")")
        CountLessThan01 = .Evaluate("COUNTIF('Sheet1'!A1:A100,""<0.1"")")
    End With

    Debug.Print Total & ", " & CountLessThan01
End Sub
```

最后，分别运行上述两个子程序各25,000次，以下是平均（5次测试）时间，单位为毫秒（当然每台电脑上会有所不同，但相互比较时表现类似）：

- 1.使用WorksheetFunction：2156 毫秒
- 2.使用数组：2219 毫秒 (+ 3 %)
- 3.使用Evaluate：4693 毫秒 (+ 118 %)
- 4.使用Range：6530 毫秒 (+ 203 %)

```
Debug.Print Total & ", " & CountLessThan01
End Sub
```

Or, for more complex calculations you can even use Application.Evaluate:

```
Sub UseEvaluate()
    Dim Total As Double
    Dim CountLessThan01 As Long

    With Application
        Total = .Evaluate("SUM(" & Sheet1.Range("A1:A100").Address( _
            external:=True) & ")")
        CountLessThan01 = .Evaluate("COUNTIF('Sheet1'!A1:A100,""<0.1"")")
    End With

    Debug.Print Total & ", " & CountLessThan01
End Sub
```

And finally, running above Subs 25,000 times each, here is the average (5 tests) time in milliseconds (of course it'll be different on each pc, but compared to each other they'll behave similarly):

- 1. UseWorksheetFunction: 2156 ms
- 2. UseArray: 2219 ms (+ 3 %)
- 3. UseEvaluate: 4693 ms (+ 118 %)
- 4. UseRange: 6530 ms (+ 203 %)



# 第30章：Excel VBA技巧与窍门

## 第30.1节：使用xlVeryHidden工作表

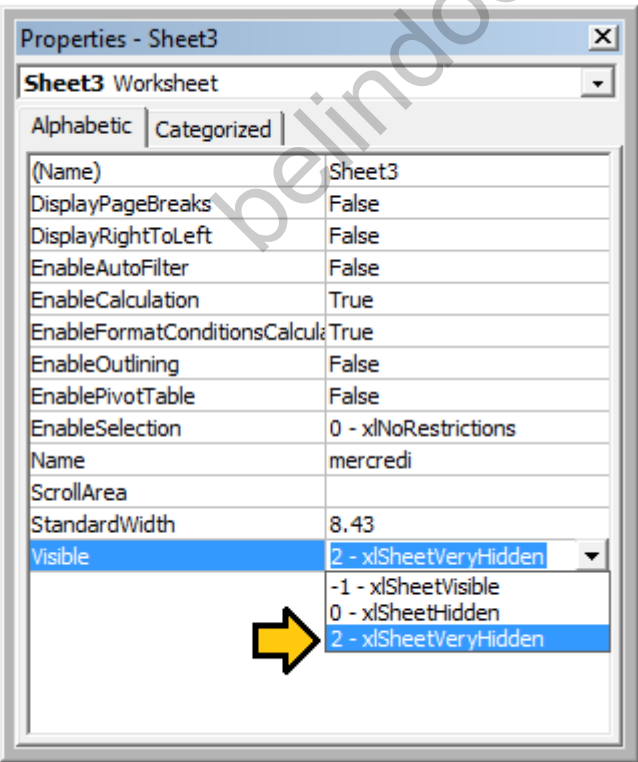
Excel中的工作表对Visible属性有三种选项。这些选项由xlSheetVisibility枚举中的常量表示，具体如下：

- 1. xlVisible或xlSheetVisible值：-1（新建工作表的默认值）
- 2. xlHidden或xlSheetHidden值：0
- 3. xlVeryHidden或xlSheetVeryHidden值：2

可见工作表表示工作表的默认可见性。它们在工作表标签栏中可见，可以自由选择 and 查看。隐藏工作表在工作表标签栏中不可见，因此无法选择。但是，可以通过右键点击工作表标签并选择“取消隐藏”来取消隐藏隐藏的工作表。另一方面，极度隐藏（Very Hidden）工作表仅能通过Visual Basic编辑器访问。这使得它们成为在多个Excel实例之间

存储数据以及存储应对最终用户隐藏的数据的极其有用的工具。工作表可以通过VBA代码中的命名引用访问，方便使用存储的数据。

要手动将工作表的 .Visible 属性更改为 xlSheetVeryHidden，请打开 VBE 的属性窗口 (F4)，选择您想要更改的工作表，然后使用第十三行的下拉菜单进行选择。



要在代码中将工作表的 .Visible 属性更改为 xlSheetVeryHidden<sup>1</sup>，同样访问 .Visible 属性并分配一个新值。

```
with Sheet3
    .Visible = xlSheetVeryHidden
end with
```

<sup>1</sup> xlVeryHidden 和 xlSheetVeryHidden 都返回数值 2（它们是可互换的）。

# Chapter 30: Excel VBA Tips and Tricks

## Section 30.1: Using xlVeryHidden Sheets

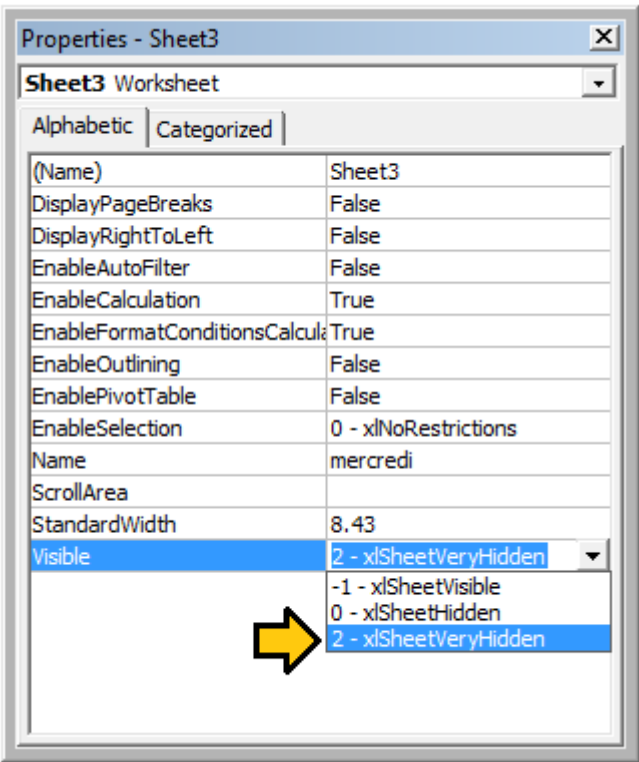
Worksheets in excel have three options for the Visible property. These options are represented by constants in the xlSheetVisibility enumeration and are as follows:

- 1. xlVisible or xlSheetVisible value: -1 (the default for new sheets)
- 2. xlHidden or xlSheetHidden value: 0
- 3. xlVeryHidden xlSheetVeryHidden value: 2

Visible sheets represent the default visibility for sheets. They are visible in the sheet tab bar and can be freely selected and viewed. Hidden sheets are hidden from the sheet tab bar and are thus not selectable. However, hidden sheets can be unhidden from the excel window by right clicking on the sheet tabs and selecting "Unhide"

Very Hidden sheets, on the other hand, are *only* accessible through the Visual Basic Editor. This makes them an incredibly useful tool for storing data across instances of excel as well as storing data that should be hidden from end users. The sheets can be accessed by named reference within VBA code, allowing easy use of the stored data.

To manually change a worksheet's .Visible property to xlSheetVeryHidden, open the VBE's Properties window (F4), select the worksheet you want to change and use the drop-down in the thirteenth row to make your selection.



To change a worksheet's .Visible property to xlSheetVeryHidden<sup>1</sup> in code, similarly access the .Visible property and assign a new value.

```
with Sheet3
    .Visible = xlSheetVeryHidden
end with
```

<sup>1</sup> Both **xlVeryHidden** and **xlSheetVeryHidden** return a numerical value of **2** (they are interchangeable).



## 第30.2节：使用带分隔符的字符串代替动态数组

在 VBA 中使用动态数组处理非常大的数据集时可能相当笨重且耗时。当在动态数组中存储简单数据类型（字符串、数字、布尔值等）时，可以通过使用 Split() 函数配合一些巧妙的字符串处理，避免 VBA 中动态数组所需的 ReDim Preserve 语句。例如，我们将查看一个循环，该循环根据某些条件将一系列范围内的值添加到字符串中，然后使用该字符串填充列表框的值。

```
Private Sub UserForm_Initialize()  
  
Dim Count As Long, DataString As String, Delimiter As String  
  
For Count = 1 To ActiveSheet.UsedRows.Count  
    If ActiveSheet.Range("A" & Count).Value <> "Your Condition" Then  
        RowString = RowString & Delimiter & ActiveSheet.Range("A" & Count).Value  
        Delimiter = "><" '通过在循环中设置分隔符，可以防止字符串中出现多余的分隔符  
    End If  
    下一个 计数  
  
ListBox1.List = Split(DataString, Delimiter)  
  
End Sub
```

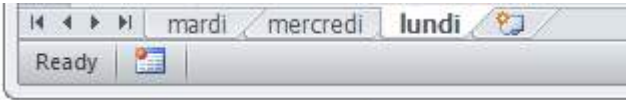
分隔符字符串本身可以设置为任何值，但明智的做法是选择一个不会自然出现在集合中的值。例如，假设你正在处理一列日期。在这种情况下，使用.、-或/作为分隔符是不明智的，因为日期格式可能使用这些符号中的任意一个，这会生成比预期更多的数据点。

注意：使用此方法存在限制（即字符串的最大长度），因此在处理非常大的数据集时应谨慎使用。这不一定是创建VBA动态数组最快或最有效的方法，但它是一个可行的替代方案。

## 第30.3节：工作表的 .Name、.Index 或 .CodeName

我们知道“最佳实践”要求范围对象应明确引用其父工作表。工作表可以通过其 .Name 属性、数字型的 .Index 属性或 .CodeName 属性来引用，但用户可以通过简单地拖动标签页重新排序工作表队列，或者在未受保护的工作簿中双击标签页并输入新名称来重命名工作表。

考虑一个标准的三个工作表。你已经将这三个工作表依次重命名为星期一、星期二和星期三，并编写了引用它们的VBA子过程。现在假设有一个用户将星期一移动到了工作表队列的末尾，然后另一个用户觉得工作表名称用法语看起来更好。现在你的工作簿中，工作表名称标签队列看起来大致如下。



如果你使用了以下任一工作表引用方法，你的代码现在将无法正常工作。

```
'通过 .Name 引用工作表  
with worksheets("Monday")
```

## Section 30.2: Using Strings with Delimiters in Place of Dynamic Arrays

Using Dynamic Arrays in VBA can be quite clunky and time intensive over very large data sets. When storing simple data types in a dynamic array (Strings, Numbers, Booleans etc.), one can avoid the ReDim Preserve statements required of dynamic arrays in VBA by using the Split() function with some clever string procedures. For example, we will look at a loop that adds a series of values from a range to a string based on some conditions, then uses that string to populate the values of a ListBox.

```
Private Sub UserForm_Initialize()  
  
Dim Count As Long, DataString As String, Delimiter As String  
  
For Count = 1 To ActiveSheet.UsedRows.Count  
    If ActiveSheet.Range("A" & Count).Value <> "Your Condition" Then  
        RowString = RowString & Delimiter & ActiveSheet.Range("A" & Count).Value  
        Delimiter = "><" 'By setting the delimiter here in the loop, you prevent an extra occurrence of the delimiter within the string  
    End If  
    Next Count  
  
ListBox1.List = Split(DataString, Delimiter)  
  
End Sub
```

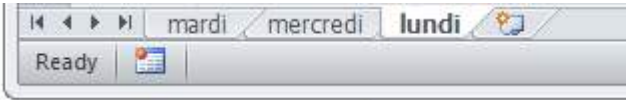
The Delimiter string itself can be set to any value, but it is prudent to choose a value which will not naturally occur within the set. Say, for example, you were processing a column of dates. In that case, using ., -, or / would be unwise as delimiters, as the dates could be formatted to use any one of these, generating more data points than you anticipated.

**Note:** There are limitations to using this method (namely the maximum length of strings), so it should be used with caution in cases of very large datasets. This is not necessarily the fastest or most effective method for creating dynamic arrays in VBA, but it is a viable alternative.

## Section 30.3: Worksheet .Name, .Index or .CodeName

We know that 'best practise' dictates that a range object should have its parent worksheet explicitly referenced. A worksheet can be referred to by its .Name property, numerical .Index property or its .CodeName property but a user can reorder the worksheet queue by simply dragging a name tab or rename the worksheet with a double-click on the same tab and some typing in an unprotected workbook.

Consider a standard three worksheet. You have renamed the three worksheets Monday, Tuesday and Wednesday in that order and coded VBA sub procedures that reference these. Now consider that one user comes along and decides that Monday belongs at the end of the worksheet queue then another comes along and decides that the worksheet names look better in French. You now have a workbook with a worksheet name tab queue that looks something like the following.



If you had used either of the following worksheet reference methods, your code would now be broken.

```
'reference worksheet by .Name  
with worksheets("Monday")
```

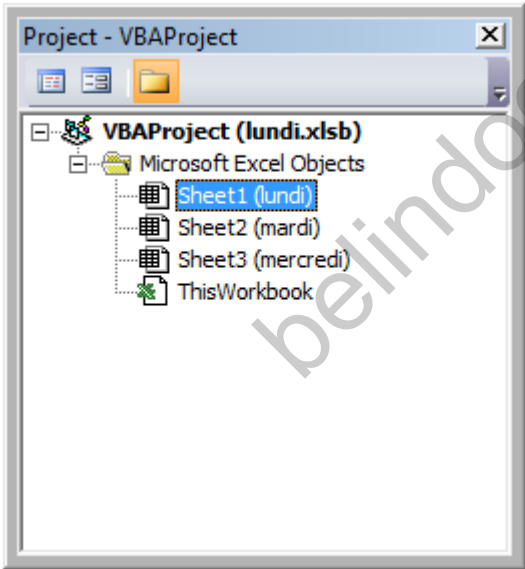
```
操作代码示例；例如：
.Range(.Cells(2, "A"), .Cells(.Rows.Count, "A").End(xlUp)) = 1
以...结束

通过序号 .Index 引用工作表
使用 worksheets(1)
操作代码示例；例如：
.Range(.Cells(2, "A"), .Cells(.Rows.Count, "A").End(xlUp)) = 1
以...结束
```

原始顺序和原始工作表名称都已被破坏。但是，如果您使用了工作表的 .CodeName 属性，您的子过程仍然可以正常运行

```
使用 Sheet1
操作代码示例；例如：
.Range(.Cells(2, "A"), .Cells(.Rows.Count, "A").End(xlUp)) = 1
以...结束
```

下面的图片显示了 VBA 项目窗口 ([Ctrl]+R)，其中按 .CodeName 列出工作表，然后按.Name（括号内）。它们显示的顺序不会改变；序号 .Index 是根据它们在工作表窗口名称标签队列中的显示顺序确定的。



虽然重命名 .CodeName 不常见，但并非不可能。只需打开 VBE 的属性窗口 ([F4])。

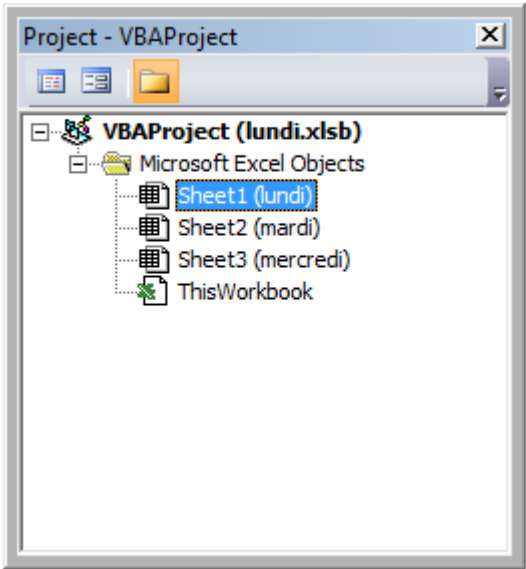
```
'operation code here; for example:
.Range(.Cells(2, "A"), .Cells(.Rows.Count, "A").End(xlUp)) = 1
end with

'reference worksheet by ordinal .Index
with worksheets(1)
'operation code here; for example:
.Range(.Cells(2, "A"), .Cells(.Rows.Count, "A").End(xlUp)) = 1
end with
```

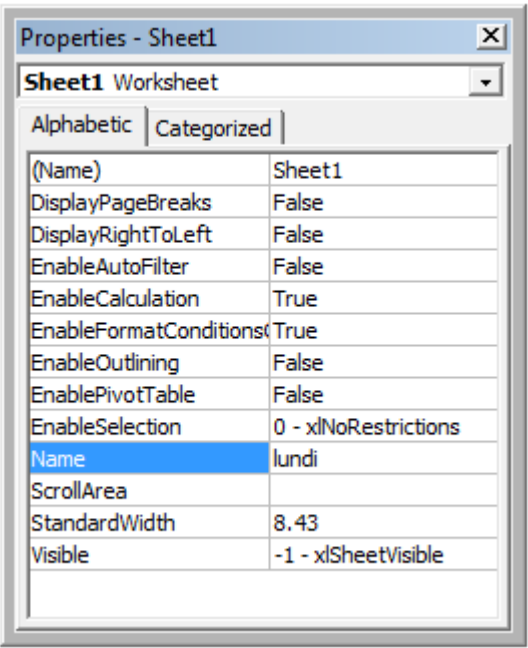
Both the original order and the original worksheet name have been compromised. However, if you had used the worksheet's .CodeName property, your sub procedure would still be operational

```
with Sheet1
'operation code here; for example:
.Range(.Cells(2, "A"), .Cells(.Rows.Count, "A").End(xlUp)) = 1
end with
```

The following image shows the VBA Project window ([Ctrl]+R) which lists the worksheets by .CodeName then by .Name (in brackets). The order they are displayed does not change; the ordinal .Index is taken by the order they are displayed in the name tab queue in the worksheet window.



While it is uncommon to rename a .CodeName, it is not impossible. Simply open the VBE's Properties window ([F4]).



工作表的 .CodeName 在第一行。工作表的 .Name 在第十行。两者均可编辑。

### 第30.4节：Excel 形状的双击事件

默认情况下，Excel 中的形状没有专门处理单击与双击的方法，仅包含“OnAction”属性以允许你处理点击事件。然而，有时你的代码可能需要对双击进行不同（或专门）的处理。以下子程序可以添加到你的 VBA 项目中，当将其设置为形状的OnAction例程时，允许你对双击进行操作。

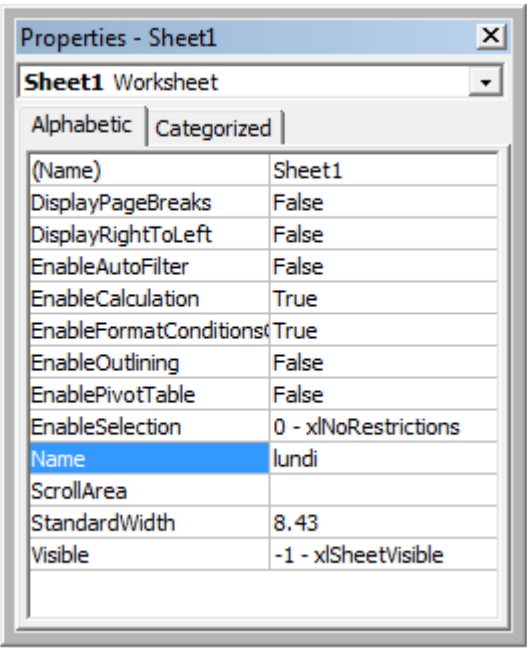
```
Public Const DOUBLECLICK_WAIT 作为 Double = 0.25 '可修改以调整点击延迟
Public LastClickObj 作为 String, LastClickTime 作为 Date

Sub ShapeDoubleClick()

    If LastClickObj = "" Then
        LastClickObj = Application.Caller
        LastClickTime = CDb1(Timer)
    Else
        If CDb1(Timer) - LastClickTime > DOUBLECLICK_WAIT Then
            LastClickObj = Application.Caller
            LastClickTime = CDb1(Timer)
        Else
            If LastClickObj = Application.Caller Then
                '在此处编写你想要的双击代码
                LastClickObj = ""
            Else
                LastClickObj = Application.Caller
                LastClickTime = CDb1(Timer)
            结束 If
        结束 If
    End If

End Sub
```

该例程将使形状在功能上忽略第一次点击，仅在指定时间内的第二次点击时运行你想要的代码。



The worksheet .CodeName is in the first row. The worksheet's .Name is in the tenth. Both are editable.

### Section 30.4: Double Click Event for Excel Shapes

By default, Shapes in Excel do not have a specific way to handle single vs. double clicks, containing only the "OnAction" property to allow you to handle clicks. However, there may be instances where your code requires you to act differently (or exclusively) on a double click. The following subroutine can be added into your VBA project and, when set as the OnAction routine for your shape, allow you to act on double clicks.

```
Public Const DOUBLECLICK_WAIT as Double = 0.25 'Modify to adjust click delay
Public LastClickObj As String, LastClickTime As Date

Sub ShapeDoubleClick()

    If LastClickObj = "" Then
        LastClickObj = Application.Caller
        LastClickTime = CDb1(Timer)
    Else
        If CDb1(Timer) - LastClickTime > DOUBLECLICK_WAIT Then
            LastClickObj = Application.Caller
            LastClickTime = CDb1(Timer)
        Else
            If LastClickObj = Application.Caller Then
                'Your desired Double Click code here
                LastClickObj = ""
            Else
                LastClickObj = Application.Caller
                LastClickTime = CDb1(Timer)
            End If
        End If
    End If

End Sub
```

This routine will cause the shape to functionally ignore the first click, only running your desired code on the second click within the specified time span.

## 第30.5节：打开文件对话框 - 多文件

此子程序是一个快速示例，演示如何允许用户选择多个文件，然后对这些文件路径执行操作，例如获取文件名并通过 `debug.print` 发送到控制台。

```
Option Explicit

Sub OpenMultipleFiles()
    Dim fd As FileDialog
    Dim fileChosen As Integer
    Dim i As Integer
    Dim basename As String
    Dim fso As Variant
    Set fso = CreateObject("Scripting.FileSystemObject")
    Set fd = Application.FileDialog(msoFileDialogFilePicker)
    basename = fso.GetBaseName(ActiveWorkbook.Name)
    fd.InitialFileName = ActiveWorkbook.Path ' 将默认位置设置为活动工作簿路径
    fd.InitialView = msoFileDialogViewList
    fd.AllowMultiSelect = True

    fileChosen = fd.Show
    If fileChosen = -1 Then
        '打开所选的每个文件
        For i = 1 To fd.SelectedItems.Count
            Debug.Print (fd.SelectedItems(i))
            Dim fileName As String
            ' 对文件执行某些操作。
            fileName = fso.GetFileName(fd.SelectedItems(i))
            Debug.Print (fileName)
        Next i
    End If
End Sub
```

## Section 30.5: Open File Dialog - Multiple Files

This subroutine is a quick example on how to allow a user to select multiple files and then do something with those file paths, such as get the file names and send it to the console via `debug.print`.

```
Option Explicit

Sub OpenMultipleFiles()
    Dim fd As FileDialog
    Dim fileChosen As Integer
    Dim i As Integer
    Dim basename As String
    Dim fso As Variant
    Set fso = CreateObject("Scripting.FileSystemObject")
    Set fd = Application.FileDialog(msoFileDialogFilePicker)
    basename = fso.GetBaseName(ActiveWorkbook.Name)
    fd.InitialFileName = ActiveWorkbook.Path ' Set Default Location to the Active Workbook Path
    fd.InitialView = msoFileDialogViewList
    fd.AllowMultiSelect = True

    fileChosen = fd.Show
    If fileChosen = -1 Then
        'open each of the files chosen
        For i = 1 To fd.SelectedItems.Count
            Debug.Print (fd.SelectedItems(i))
            Dim fileName As String
            ' do something with the files.
            fileName = fso.GetFileName(fd.SelectedItems(i))
            Debug.Print (fileName)
        Next i
    End If
End Sub
```

# 第31章：常见错误

## 第31.1节：限定引用

当引用工作表、区域或单个单元格时，重要的是要完全限定引用。

例如：

```
ThisWorkbook.Worksheets("Sheet1").Range(Cells(1, 2), Cells(2, 3)).Copy
```

未完全限定：Cells引用没有关联工作簿和工作表。没有明确引用时，Cells默认指向ActiveSheet。因此，如果当前ActiveSheet不是Sheet1，这段代码将失败（产生错误结果）。

最简单的修正方法是使用With语句，如下所示：

```
With ThisWorkbook.Worksheets("Sheet1")
    .Range(.Cells(1, 2), .Cells(2, 3)).Copy
End With
```

或者，你可以使用工作表变量。（如果代码需要引用多个工作表，比如从一个表复制数据到另一个表，这通常是首选方法。）

```
Dim ws1 As Worksheet
Set ws1 = ThisWorkbook.Worksheets("Sheet1")
ws1.Range(ws1.Cells(1, 2), ws1.Cells(2, 3)).Copy
```

另一个常见问题是引用工作表集合时未限定工作簿。例如：

```
Worksheets("Sheet1").Copy
```

工作表Sheet1未完全限定，且缺少工作簿。如果代码中引用了多个工作簿，可能会失败。请改用以下之一：

```
ThisWorkbook.Worksheets("Sheet1") ' <--ThisWorkbook 指的是包含正在运行的 VBA 代码的工作簿

Workbooks("Book1").Worksheets("Sheet1") ' <--其中 Book1 是包含 Sheet1 的工作簿
```

但是，避免使用以下方式：

```
ActiveWorkbook.Worksheets("Sheet1") ' <--有效，但如果激活了另一个工作簿，引用将被更改
```

同样，对于range对象，如果没有明确限定，range将引用当前活动的工作表：

```
Range("a1")
```

等同于：

```
ActiveSheet.Range("a1")
```

# Chapter 31: Common Mistakes

## Section 31.1: Qualifying References

When referring to a worksheet, a range or individual cells, it is important to fully qualify the reference.

For example:

```
ThisWorkbook.Worksheets("Sheet1").Range(Cells(1, 2), Cells(2, 3)).Copy
```

Is not fully qualified: The Cells references do not have a workbook and worksheet associated with them. Without an explicit reference, Cells refers to the ActiveSheet by default. So this code will fail (produce incorrect results) if a worksheet other than Sheet1 is the current ActiveSheet.

The easiest way to correct this is to use a With statement as follows:

```
With ThisWorkbook.Worksheets("Sheet1")
    .Range(.Cells(1, 2), .Cells(2, 3)).Copy
End With
```

Alternatively, you can use a Worksheet variable. (This will most likely be preferred method if your code needs to reference multiple Worksheets, like copying data from one sheet to another.)

```
Dim ws1 As Worksheet
Set ws1 = ThisWorkbook.Worksheets("Sheet1")
ws1.Range(ws1.Cells(1, 2), ws1.Cells(2, 3)).Copy
```

Another frequent problem is referencing the Worksheets collection without qualifying the Workbook. For example:

```
Worksheets("Sheet1").Copy
```

The worksheet Sheet1 is not fully qualified, and lacks a workbook. This could fail if multiple workbooks are referenced in the code. Instead, use one of the following:

```
ThisWorkbook.Worksheets("Sheet1") ' <--ThisWorkbook refers to the workbook containing
                                     ' the running VBA code
Workbooks("Book1").Worksheets("Sheet1") ' <--Where Book1 is the workbook containing Sheet1
```

However, avoid using the following:

```
ActiveWorkbook.Worksheets("Sheet1") ' <--Valid, but if another workbook is activated
                                     ' the reference will be changed
```

Similarly for range objects, if not explicitly qualified, the range will refer to the currently active sheet:

```
Range("a1")
```

Is the same as:

```
ActiveSheet.Range("a1")
```



## 第31.2节：在循环中删除行或列

如果你想在循环中删除行（或列），你应该总是从范围的末尾开始循环，并在每一步向后移动。  
如果使用以下代码：

```
Dim i As Long
With Workbooks("Book1").Worksheets("Sheet1")
    For i = 1 To 4
        If IsEmpty(.Cells(i, 1)) Then .Rows(i).Delete
    Next i
End With
```

你会遗漏一些行。例如，如果代码删除了第3行，那么第4行会变成第3行。然而，变量*i*会变成4。因此，在这种情况下，代码会遗漏一行并检查另一行，而这行之前不在范围内。

正确的代码应该是

```
Dim i As Long
With Workbooks("Book1").Worksheets("Sheet1")
    For i = 4 To 1 Step -1
        If IsEmpty(.Cells(i, 1)) Then .Rows(i).Delete
    Next i
End With
```

## 第31.3节：ActiveWorkbook 与 ThisWorkbook

ActiveWorkbook 和 ThisWorkbook 有时会被VBA新手混用，而没有完全理解每个对象所对应的内容，这可能导致运行时出现不期望的行为。这两个对象都属于Application对象

ActiveWorkbook 对象指的是在执行时刻Excel应用程序对象中当前处于最顶层视图的工作簿。（例如，当引用该对象时，你可以看到并与之交互的工作簿）

```
Sub ActiveWorkbookExample()

    '// 假设"Other Workbook.xlsx"的A1单元格中写有"Bar"。

    ActiveWorkbook.ActiveSheet.Range("A1").Value = "Foo"
    Debug.Print ActiveWorkbook.ActiveSheet.Range("A1").Value '// 输出 "Foo"

    Workbooks.Open("C:\Users\BloggsJ\Other Workbook.xlsx")
    Debug.Print ActiveWorkbook.ActiveSheet.Range("A1").Value '// 输出 "Bar"

    Workbooks.Add 1
    Debug.Print ActiveWorkbook.ActiveSheet.Range("A1").Value '// 无输出

End Sub
```

The ThisWorkbook 对象指的是代码执行时所属的工作簿。

```
Sub ThisWorkbookExample()

    '// 假设开始时此代码位于当前激活的同一工作簿中

    ActiveWorkbook.Sheet1.Range("A1").Value = "Foo"
    Workbooks.Add 1
```

## Section 31.2: Deleting rows or columns in a loop

If you want to delete rows (or columns) in a loop, you should always loop starting from the end of range and move back in every step. In case of using the code:

```
Dim i As Long
With Workbooks("Book1").Worksheets("Sheet1")
    For i = 1 To 4
        If IsEmpty(.Cells(i, 1)) Then .Rows(i).Delete
    Next i
End With
```

You will miss some rows. For example, if the code deletes row 3, then row 4 becomes row 3. However, variable *i* will change to 4. So, in this case the code will miss one row and check another, which wasn't in range previously.

The right code would be

```
Dim i As Long
With Workbooks("Book1").Worksheets("Sheet1")
    For i = 4 To 1 Step -1
        If IsEmpty(.Cells(i, 1)) Then .Rows(i).Delete
    Next i
End With
```

## Section 31.3: ActiveWorkbook vs. ThisWorkbook

ActiveWorkbook and ThisWorkbook sometimes get used interchangeably by new users of VBA without fully understanding which each object relates to, this can cause undesired behaviour at run-time. Both of these objects belong to the Application Object

The ActiveWorkbook object refers to the workbook that is currently in the top-most view of the Excel application object at the time of execution. (e.g. *The workbook that you can see and interact with at the point when this object is referenced*)

```
Sub ActiveWorkbookExample()

    '// Let's assume that 'Other Workbook.xlsx' has "Bar" written in A1.

    ActiveWorkbook.ActiveSheet.Range("A1").Value = "Foo"
    Debug.Print ActiveWorkbook.ActiveSheet.Range("A1").Value '// Prints "Foo"

    Workbooks.Open("C:\Users\BloggsJ\Other Workbook.xlsx")
    Debug.Print ActiveWorkbook.ActiveSheet.Range("A1").Value '// Prints "Bar"

    Workbooks.Add 1
    Debug.Print ActiveWorkbook.ActiveSheet.Range("A1").Value '// Prints nothing

End Sub
```

The ThisWorkbook object refers to the workbook in which the code belongs to at the time it is being executed.

```
Sub ThisWorkbookExample()

    '// Let's assume to begin that this code is in the same workbook that is currently active

    ActiveWorkbook.Sheet1.Range("A1").Value = "Foo"
    Workbooks.Add 1
```

```
ActiveWorkbook.ActiveSheet.Range("A1").Value = "Bar"

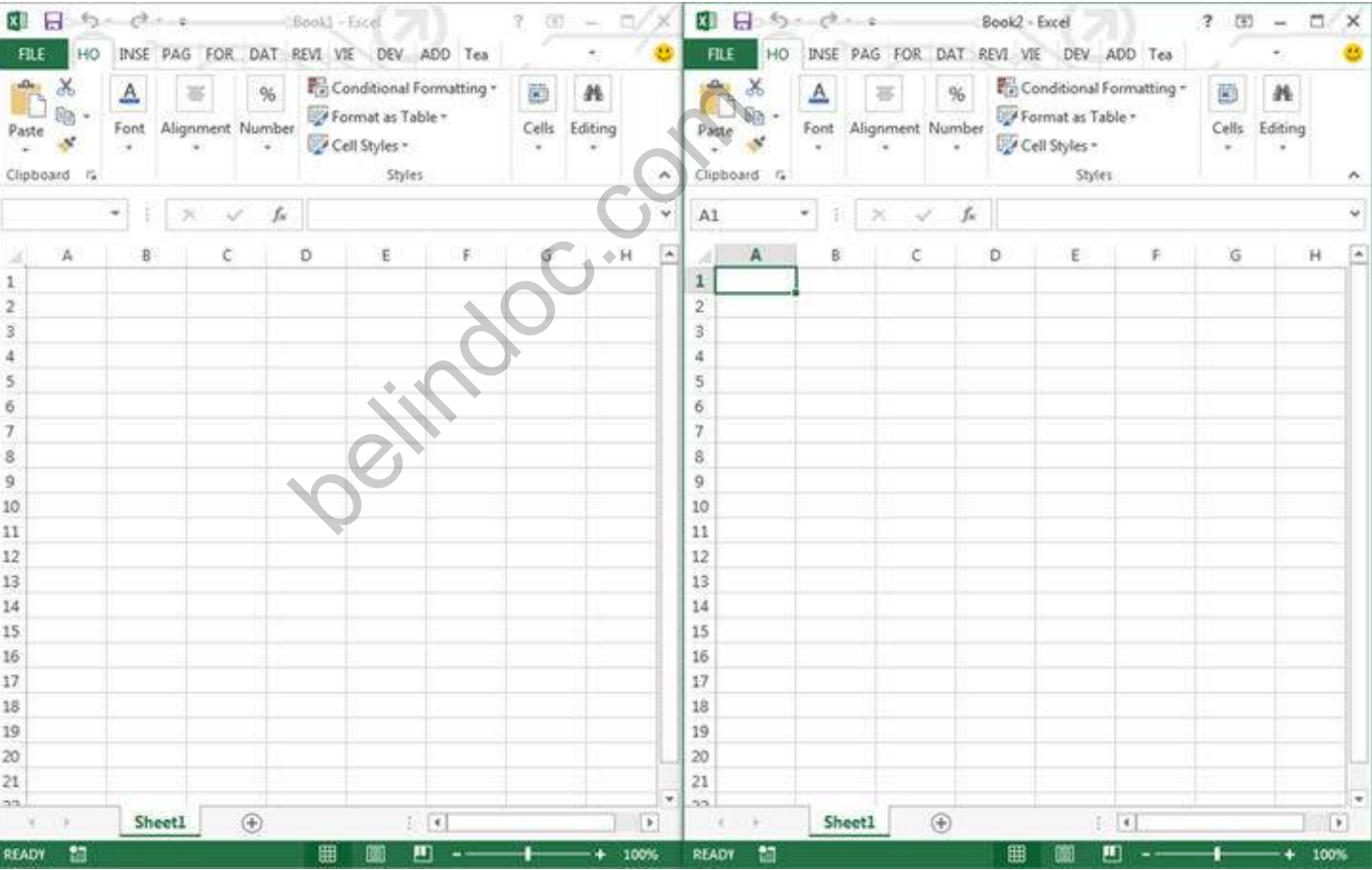
Debug.Print ActiveWorkbook.ActiveSheet.Range("A1").Value '// 输出 "Bar"
Debug.Print ThisWorkbook.Sheet1.Range("A1").Value '// 输出 "Foo"

End Sub
```

第31.4节：单文档界面与多文档界面

请注意，Microsoft Excel 2013（及更高版本）使用单文档界面（SDI），而Excel 2010（及更早版本）使用多文档界面（MDI）。

这意味着对于 Excel 2013（SDI），单个 Excel 实例中的每个工作簿都包含其自己的功能区 UI：



相反，对于 Excel 2010，单个 Excel 实例中的每个工作簿使用的是公共功能区 UI（MDI）：

```
ActiveWorkbook.ActiveSheet.Range("A1").Value = "Bar"

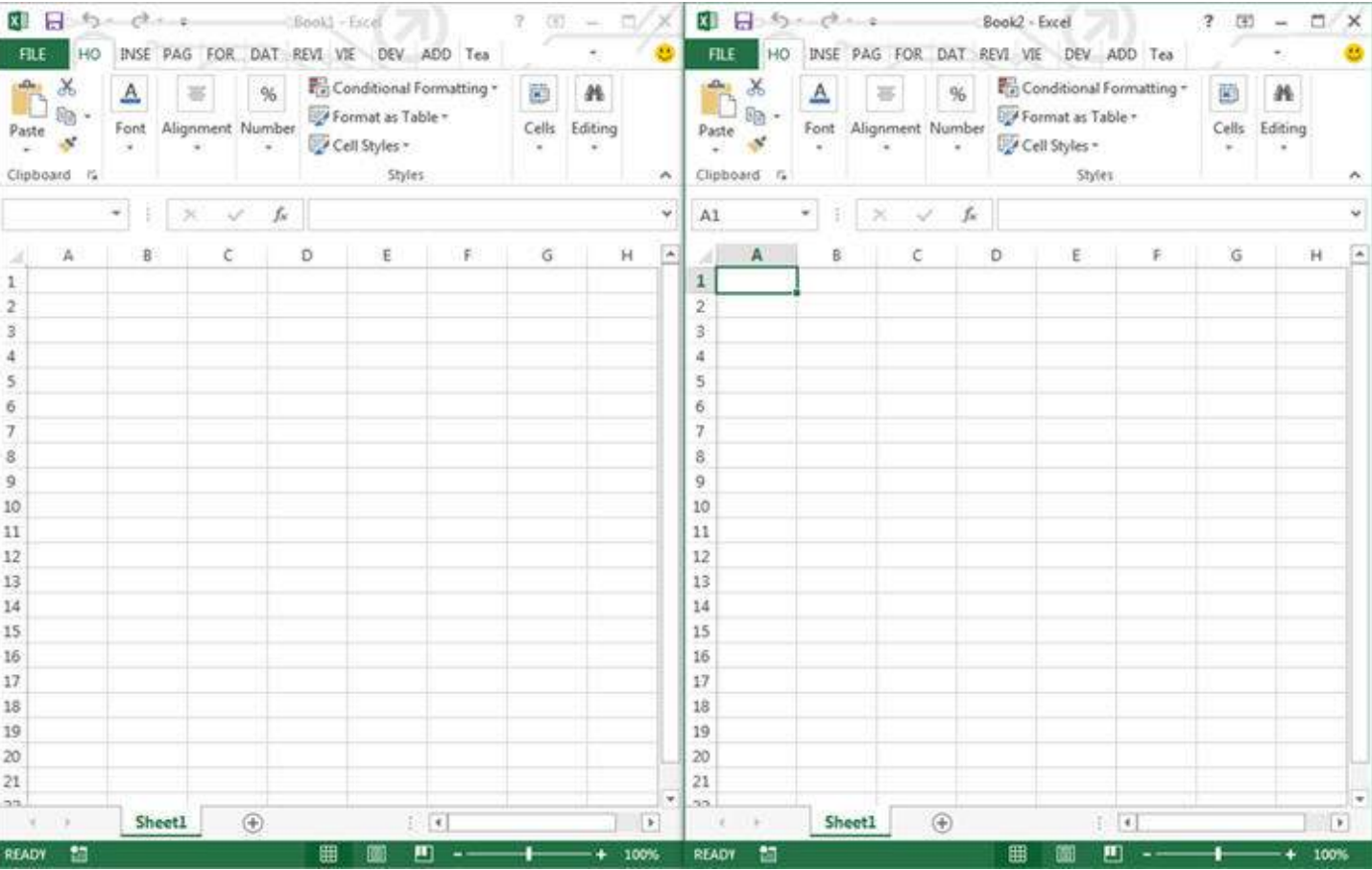
Debug.Print ActiveWorkbook.ActiveSheet.Range("A1").Value '// Prints "Bar"
Debug.Print ThisWorkbook.Sheet1.Range("A1").Value '// Prints "Foo"

End Sub
```

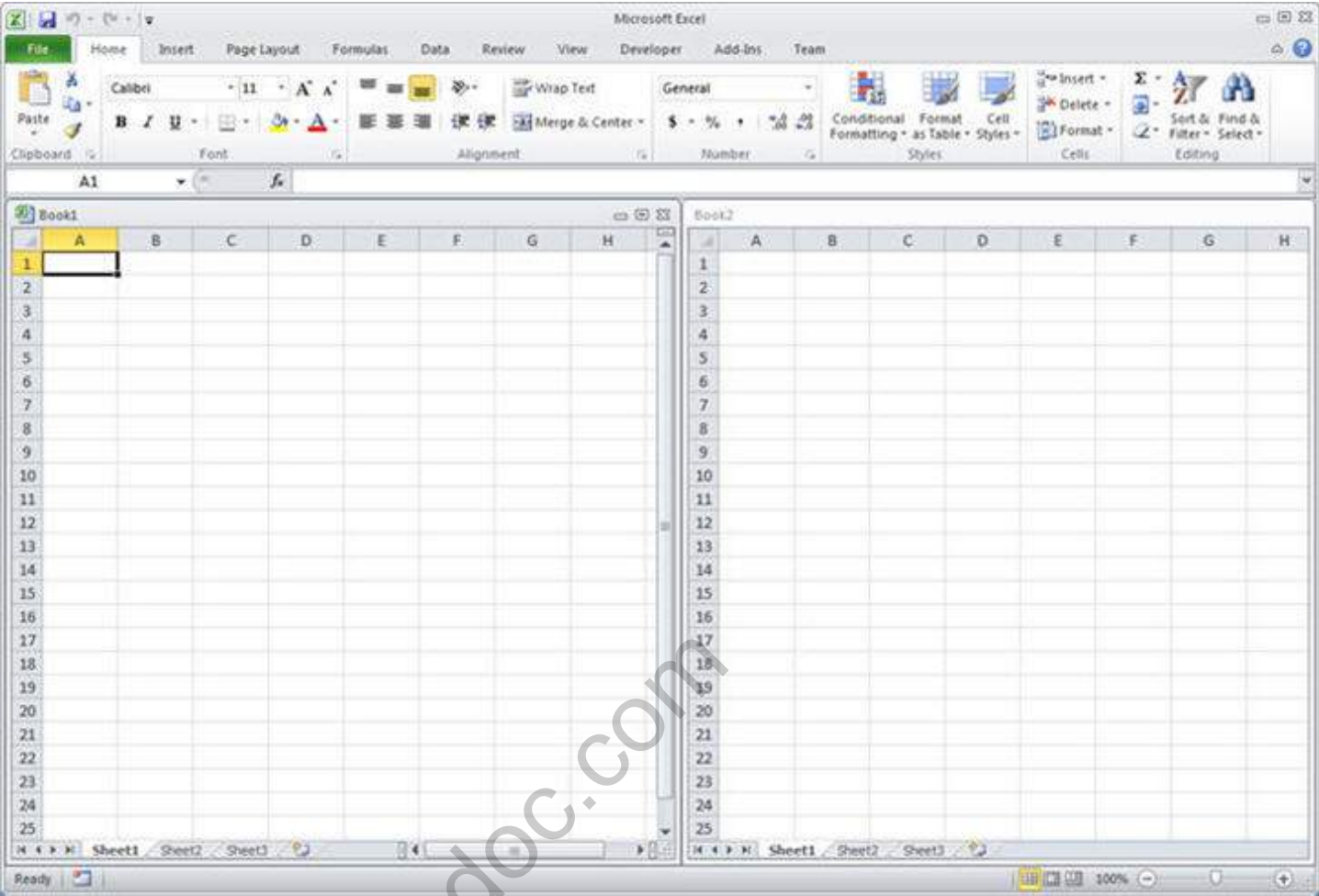
Section 31.4: Single Document Interface Versus Multiple Document Interfaces

Be aware that Microsoft Excel 2013 (and higher) uses Single Document Interface (SDI) and that Excel 2010 (And below) uses Multiple Document Interfaces (MDI).

This implies that for Excel 2013 (SDI), each workbook in a single instance of Excel contains its **own** ribbon UI:



Conversely for Excel 2010, each workbook in a single instance of Excel utilized a **common** ribbon UI (MDI):



如果您想迁移与功能区交互的 VBA 代码（2010 <-> 2013），这会引发一些重要问题。

必须创建一个过程，以便在 Excel 2013 及更高版本中跨所有工作簿保持功能区 UI 控件的状态一致。

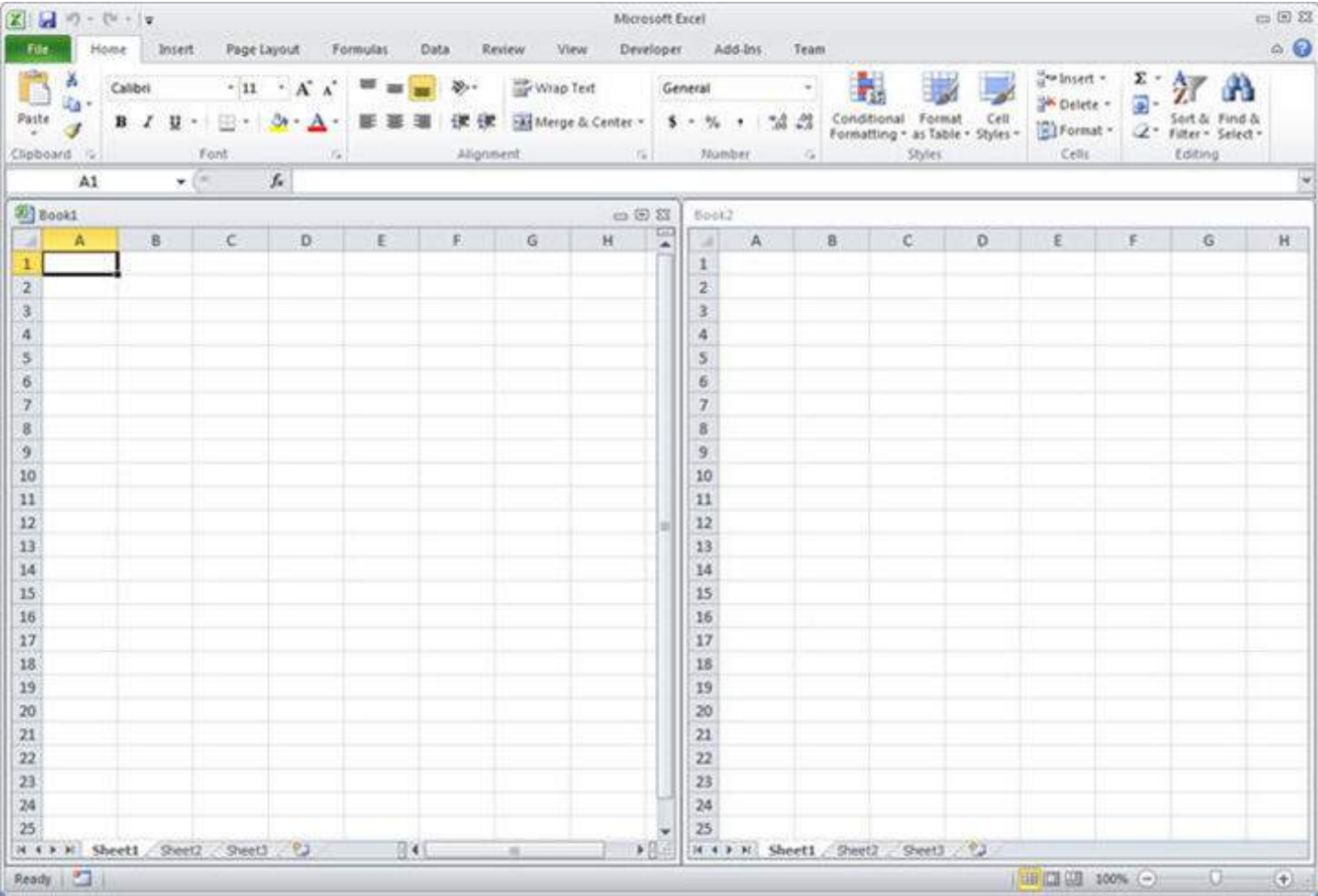
注意：

- 1. 所有 Excel 应用程序级别的窗口方法、事件和属性保持不变。  
(Application.ActiveWindow, Application.Windows 等...)
- 2. 在 Excel 2013 及更高版本 (SDI) 中，所有工作簿级别的窗口方法、事件和属性现在都作用于顶层窗口。可以通过

Application.Hwnd 获取该顶层窗口的句柄

欲了解更多详情，请参阅此示例的来源：MSDN。

这也会给无模式用户窗体带来一些麻烦。解决方案见Here。



This raise some important issues if you want to migrate a VBA code (2010 <->2013) that interact with the Ribbon.

A procedure has to be created to update ribbon UI controls in the same state across all workbooks for Excel 2013 and Higher.

Note that :

- 1. All Excel application-level window methods, events, and properties remain unaffected.  
(Application.ActiveWindow, Application.Windows ...)
- 2. In Excel 2013 and higher (SDI) all of the workbook-level window methods, events, and properties now operate on the top level window. It is possible to retrieve the handle of this top level window with Application.Hwnd

To get more details, see the source of this example: MSDN.

This also causes some trouble with modeless userforms. See Here for a solution.

学分

非常感谢所有来自Stack Overflow Documentation的人员帮助提供这些内容，  
更多更改可以发送至[web@petercv.com](mailto:web@petercv.com)以发布或更新新内容

<a href="#">亚当</a>	第4章
<a href="#">亚历克西斯·奥尔森</a>	第29章
<a href="#">阿隆·阿德勒</a>	第2章
<a href="#">安迪·特拉</a>	第5章和第30章
<a href="#">布拉尼斯拉夫·科拉尔</a>	第1章、第4章和第29章
<a href="#">拜伦·沃尔</a>	第21章
<a href="#">脾气暴躁的船长</a>	第18章和第20章
<a href="#">切尔</a>	第27章和第29章
<a href="#">科迪·G.</a>	第1章、第28章、第29章和第30章
<a href="#">共产国际</a>	第29章
<a href="#">好奇</a>	第14章
<a href="#">道格·科茨</a>	第1章、第4章和第12章
<a href="#">EEM</a>	第1章
<a href="#">伊根·沃尔夫</a>	第31章
<a href="#">Etheur</a>	第28章
<a href="#">Excel开发者</a>	第11章
<a href="#">FreeMan</a>	第29章
<a href="#">genespos</a>	第29章
<a href="#">戈登·贝尔</a>	第1章和第31章
<a href="#">格雷戈里</a>	第28章
<a href="#">Hubisan</a>	第2章、第14章和第29章
<a href="#">Jeeped</a>	第8章、第18章、第29章和第30章
<a href="#">jlookup</a>	第18章
<a href="#">乔尔·斯波尔斯基</a>	第1章、第4章和第20章
<a href="#">朱利安·库赫尔鲍尔</a>	第28章
<a href="#">卡兹</a>	第1章
<a href="#">凯尔</a>	第28章和第29章
<a href="#">马特·尤哈斯</a>	第29章
<a href="#">宏观人</a>	第1章、第29章、第30章和第31章
<a href="#">马利克</a>	第1章、第8章、第18章、第29章和第31章
<a href="#">马苏德</a>	第26章
<a href="#">米格尔_柳</a>	第2章
<a href="#">迈克</a>	第24章
<a href="#">米奇180</a>	第14章
<a href="#">蒙基脸</a>	第29章
<a href="#">保罗·比卡</a>	第14章、第26章和第29章
<a href="#">彼得T</a>	第10章、第17章和第29章
<a href="#">波特兰跑者</a>	第5章和第29章
<a href="#">PH</a>	第29章
<a href="#">正交</a>	第7章和第15章
<a href="#">R3uK</a>	第6章和第14章
<a href="#">RGA</a>	第1、14、23、28、29和30章
<a href="#">罗比</a>	第24章
<a href="#">罗恩·麦克马洪</a>	第28章
<a href="#">Sgdva</a>	第19章
<a href="#">沙欣</a>	第2章
<a href="#">沙伊·拉多</a>	第1、12、14和29章

Credits

Thank you greatly to all the people from Stack Overflow Documentation who helped provide this content,  
more changes can be sent to [web@petercv.com](mailto:web@petercv.com) for new content to be published or updated

<a href="#">Adam</a>	Chapter 4
<a href="#">Alexis Olson</a>	Chapter 29
<a href="#">Alon Adler</a>	Chapter 2
<a href="#">Andy Terra</a>	Chapters 5 and 30
<a href="#">Branislav Kollár</a>	Chapters 1, 4 and 29
<a href="#">Byron Wall</a>	Chapter 21
<a href="#">Captain Grumpy</a>	Chapters 18 and 20
<a href="#">Chel</a>	Chapters 27 and 29
<a href="#">Cody G.</a>	Chapters 1, 28, 29 and 30
<a href="#">Comintern</a>	Chapter 29
<a href="#">curious</a>	Chapter 14
<a href="#">Doug Coats</a>	Chapters 1, 4 and 12
<a href="#">EEM</a>	Chapter 1
<a href="#">Egan Wolf</a>	Chapter 31
<a href="#">Etheur</a>	Chapter 28
<a href="#">Excel Developers</a>	Chapter 11
<a href="#">FreeMan</a>	Chapter 29
<a href="#">genespos</a>	Chapter 29
<a href="#">Gordon Bell</a>	Chapters 1 and 31
<a href="#">Gregor y</a>	Chapter 28
<a href="#">Hubisan</a>	Chapters 2, 14 and 29
<a href="#">Jeeped</a>	Chapters 8, 18, 29 and 30
<a href="#">jlookup</a>	Chapter 18
<a href="#">Joel Spolsky</a>	Chapters 1, 4 and 20
<a href="#">Julian Kuchlbauer</a>	Chapter 28
<a href="#">Kaz</a>	Chapter 1
<a href="#">Kyle</a>	Chapters 28 and 29
<a href="#">Máté Juhász</a>	Chapter 29
<a href="#">Macro Man</a>	Chapters 1, 29, 30 and 31
<a href="#">Malick</a>	Chapters 1, 8, 18, 29 and 31
<a href="#">Masoud</a>	Chapter 26
<a href="#">Miguel Ryu</a>	Chapter 2
<a href="#">Mike</a>	Chapter 24
<a href="#">Miqi180</a>	Chapter 14
<a href="#">Munkeeface</a>	Chapter 29
<a href="#">paul bica</a>	Chapters 14, 26 and 29
<a href="#">PeterT</a>	Chapters 10, 17 and 29
<a href="#">Portland Runner</a>	Chapters 5 and 29
<a href="#">P□H</a>	Chapter 29
<a href="#">quadrature</a>	Chapters 7 and 15
<a href="#">R3uK</a>	Chapters 6 and 14
<a href="#">RGA</a>	Chapters 1, 14, 23, 28, 29 and 30
<a href="#">Robby</a>	Chapter 24
<a href="#">Ron McMahon</a>	Chapter 28
<a href="#">Sgdva</a>	Chapter 19
<a href="#">Shahin</a>	Chapter 2
<a href="#">Shai Rado</a>	Chapters 1, 12, 14 and 29

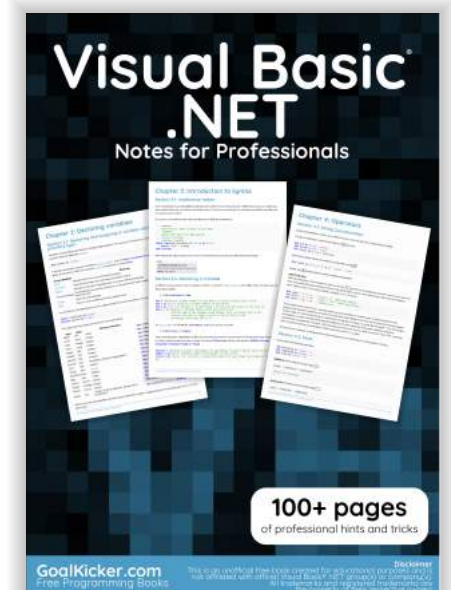
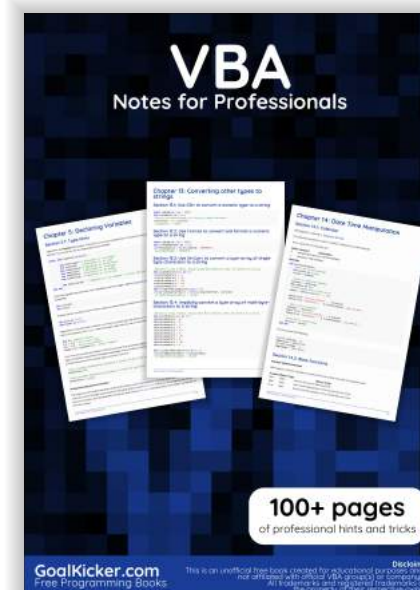
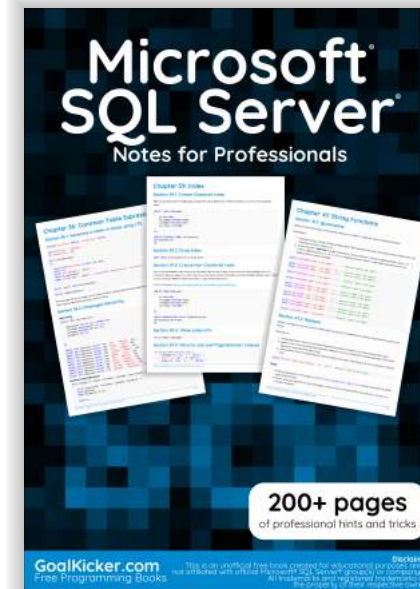
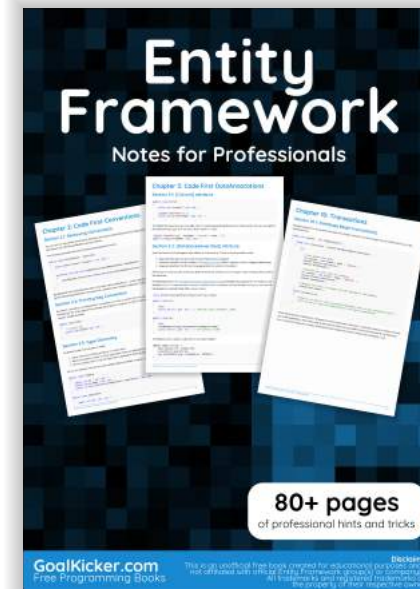
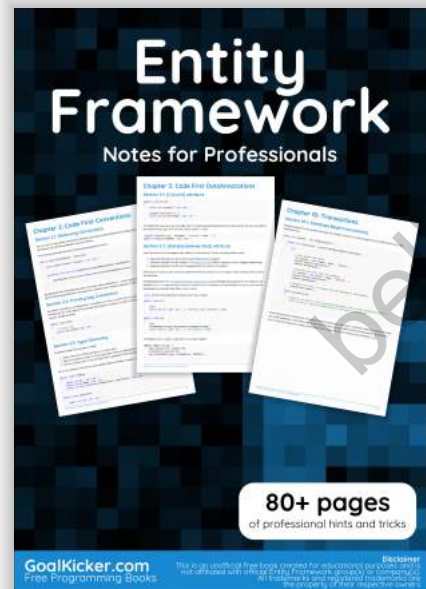
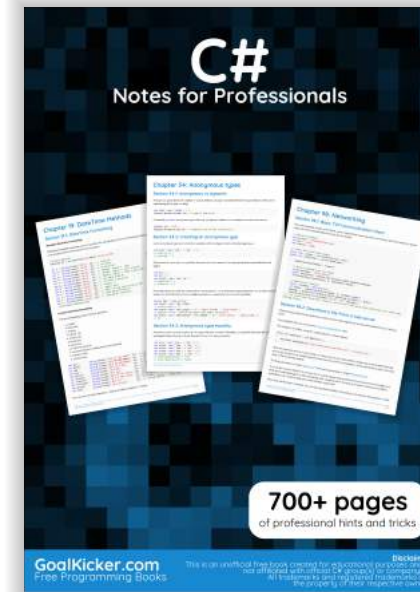
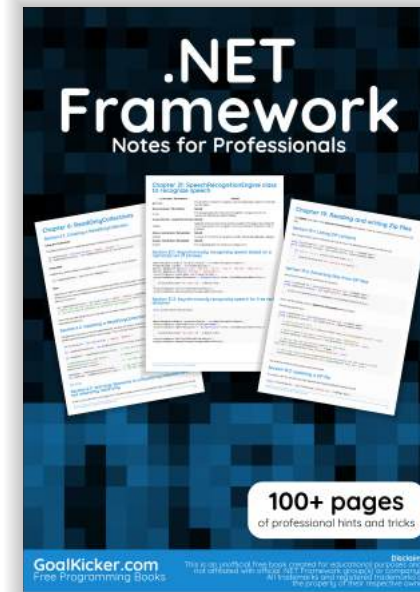
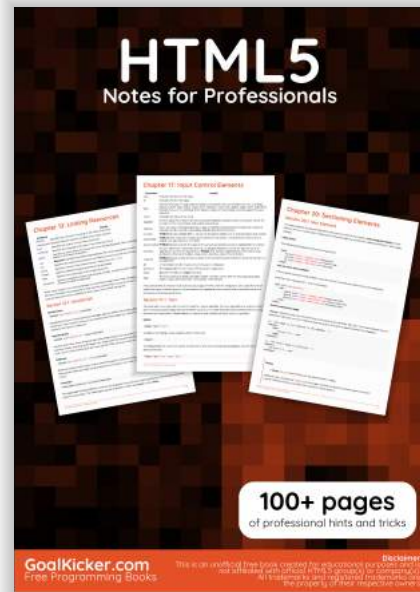


<a href="#">斯莱</a>	第8章和第14章
<a href="#">斯特凡·皮诺</a>	第29章
<a href="#">SteveES</a>	第3章
<a href="#">史蒂文·施罗德</a>	第28章和第29章
<a href="#">SWa</a>	第31章
<a href="#">T.M.</a>	第7、22和26章
<a href="#">TheGuyThatDoesn'tKnowMuch</a>	第27章
<a href="#">ThunderFrame</a>	第29章
<a href="#">吐司</a>	第1、28和31章
<a href="#">user3561813</a>	第8章
<a href="#">维加德</a>	第4章和第8章
<a href="#">绝望者</a>	第29章
<a href="#">维佳塔</a>	第13章
<a href="#">Zsmaster</a>	第9章、第16章和第25章

<a href="#">Slai</a>	Chapters 8 and 14
<a href="#">Stefan Pinnow</a>	Chapter 29
<a href="#">SteveES</a>	Chapter 3
<a href="#">Steven Schroeder</a>	Chapters 28 and 29
<a href="#">SWa</a>	Chapter 31
<a href="#">T.M.</a>	Chapters 7, 22 and 26
<a href="#">TheGuyThatDoesn'tKnowMuch</a>	Chapter 27
<a href="#">ThunderFrame</a>	Chapter 29
<a href="#">Toast</a>	Chapters 1, 28 and 31
<a href="#">user3561813</a>	Chapter 8
<a href="#">Vegard</a>	Chapters 4 and 8
<a href="#">Verzweifler</a>	Chapter 29
<a href="#">Vityata</a>	Chapter 13
<a href="#">Zsmaster</a>	Chapters 9, 16 and 25



## 你可能也喜欢



## You may also like