Бесплатная электронная книга

учусь

excel-vba

Free unaffiliated eBook created from
Stack Overflow contributors.

#excel-vba

Около

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: excel-vba

It is an unofficial and free excel-vba ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official excel-vba.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

глава 1: Начало работы с excel-vba

замечания

Microsoft Excel включает в себя всеобъемлющий язык программирования макросов, который называется VBA. Этот язык программирования предоставляет вам как минимум три дополнительных ресурса:

1. Автоматически управлять Excel из кода с помощью макросов. По большей части все, что пользователь может сделать, манипулируя Excel из пользовательского интерфейса, можно сделать, написав код в Excel VBA.
2. Создайте новые пользовательские функции рабочего листа.
3. Взаимодействуйте Excel с другими приложениями, такими как Microsoft Word, PowerPoint, Internet Explorer, Блокнот и т. Д.

VBA означает Visual Basic для приложений. Это специальная версия почтенного языка программирования Visual Basic, на котором появились макросы Microsoft Excel с середины 1990-х годов.

ВАЖНЫЙ
Пожалуйста, убедитесь, что любые примеры или темы, созданные в теге excel-vba, **специфичны** и имеют **отношение** к использованию VBA с Microsoft Excel. Любые предлагаемые темы или примеры, которые являются общими для языка VBA, должны быть отклонены, чтобы предотвратить дублирование усилий.

- по темам:

  ✓ *Создание и взаимодействие с объектами листа*
  ✓ *Класс* `WorksheetFunction` *и соответствующие методы*
  ✓ *Использование перечисления* `xlDirection` *для перемещения по диапазону*

- примеры вне темы:

  ✗ *Как создать цикл «для каждого»*
  ✗ *Класс* `MsgBox` *и способ отображения сообщения*
  ✗ *Использование WinAPI в VBA*

Версии

VB

Версия	Дата выхода
VB6	1998-10-01
VB7	2001-06-06
WIN32	1998-10-01
Win64	2001-06-06
MAC	1998-10-01

превосходить

Версия	Дата выхода
16	2016-01-01
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

Examples

Объявление переменных

Чтобы явно объявить переменные в VBA, используйте оператор `Dim` , за которым следуют имя и тип переменной. Если переменная используется без объявления или если тип не указан, ему будет присвоен тип `Variant` .

Используйте оператор `Option Explicit` в первой строке модуля, чтобы заставить все

переменные быть объявлены перед использованием (см. ВСЕГДА Используйте «Option Explicit»).

Всегда использовать `Option Explicit` настоятельно рекомендуется, потому что он помогает предотвратить ошибки опечатки / орфографии и гарантирует, что переменные / объекты останутся в их предполагаемом типе.

```
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
```

Несколько переменных могут быть объявлены в одной строке с использованием запятых в качестве разделителей, но **каждый тип должен быть объявлен отдельно** или по умолчанию будет использоваться тип `Variant` .

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

Переменные также могут быть объявлены с использованием суффиксов типа «Тип данных» ($% &! # @), Однако использование их все больше и больше обескураживается.

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

Другими способами объявления переменных являются:

- `Static` like: `Static CounterVariable as Integer`

  Когда вы используете инструкцию Static вместо оператора Dim, объявленная переменная сохраняет свое значение между вызовами.

- `Public` как: `Public CounterVariable as Integer`

  Публичные переменные могут использоваться в любых процедурах в проекте. Если публичная переменная объявлена в стандартном модуле или модуле класса, ее также можно использовать в любых проектах, которые ссылаются на проект, в котором объявлена публичная переменная.

- `Private` как: `Private CounterVariable as Integer`

  Частные переменные могут использоваться только процедурами в одном модуле.

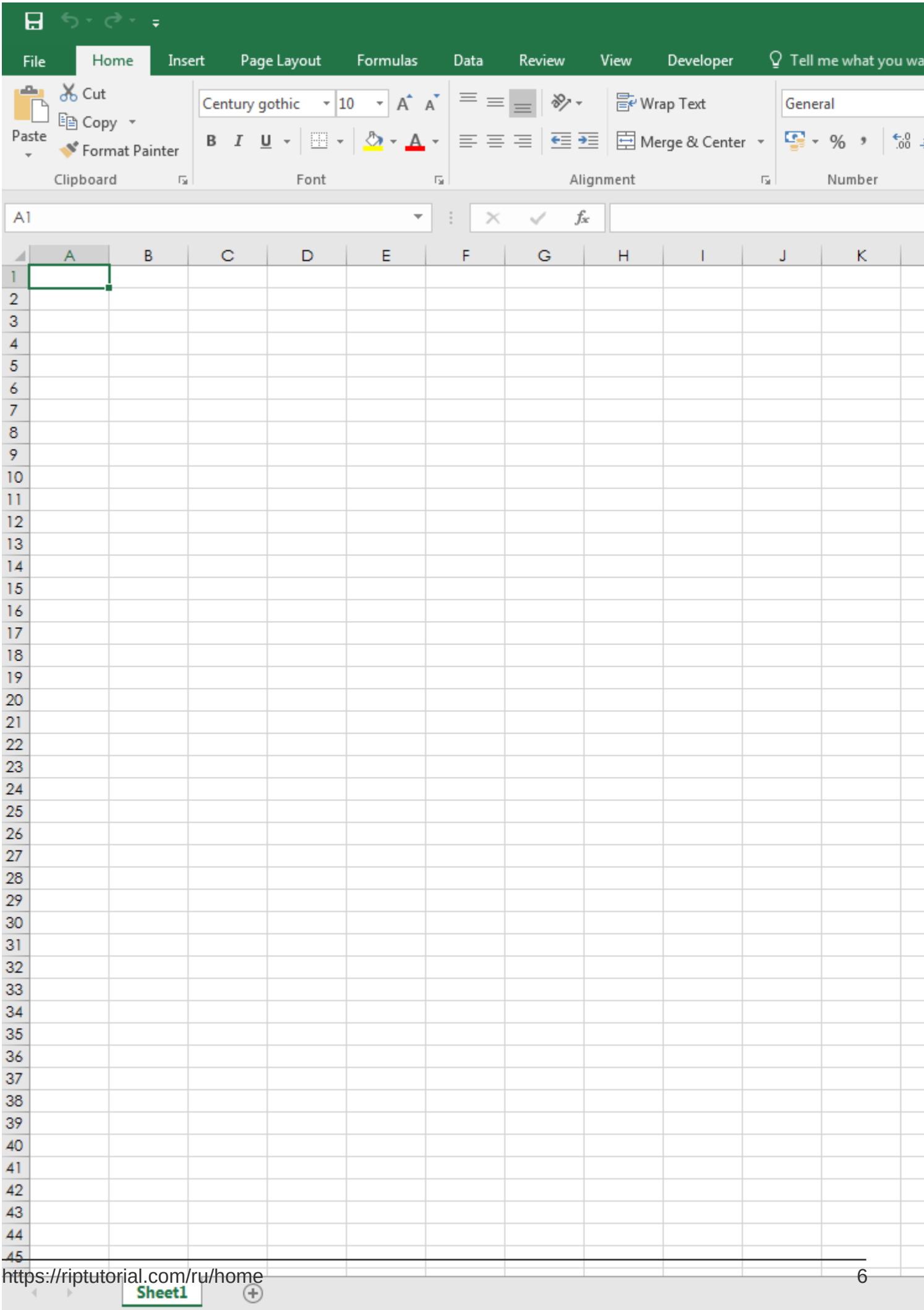
Источник и дополнительная информация:

MSDN-объявление переменных

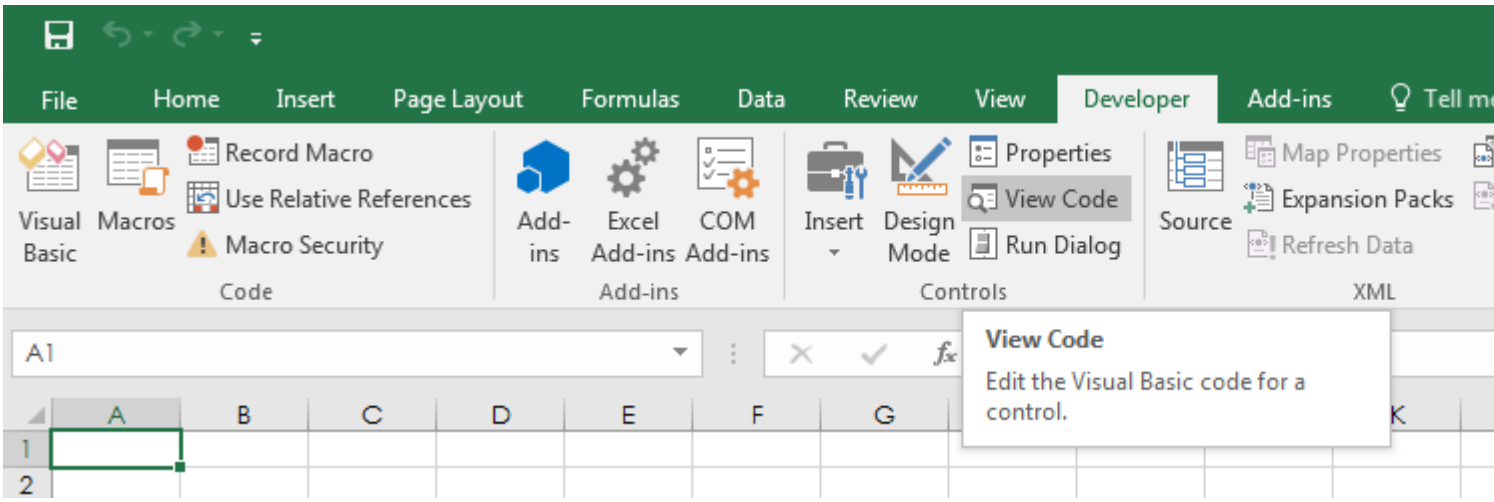
Типовые символы (Visual Basic)

Открытие редактора Visual Basic (VBE)

Шаг 1. Откройте рабочую книгу.

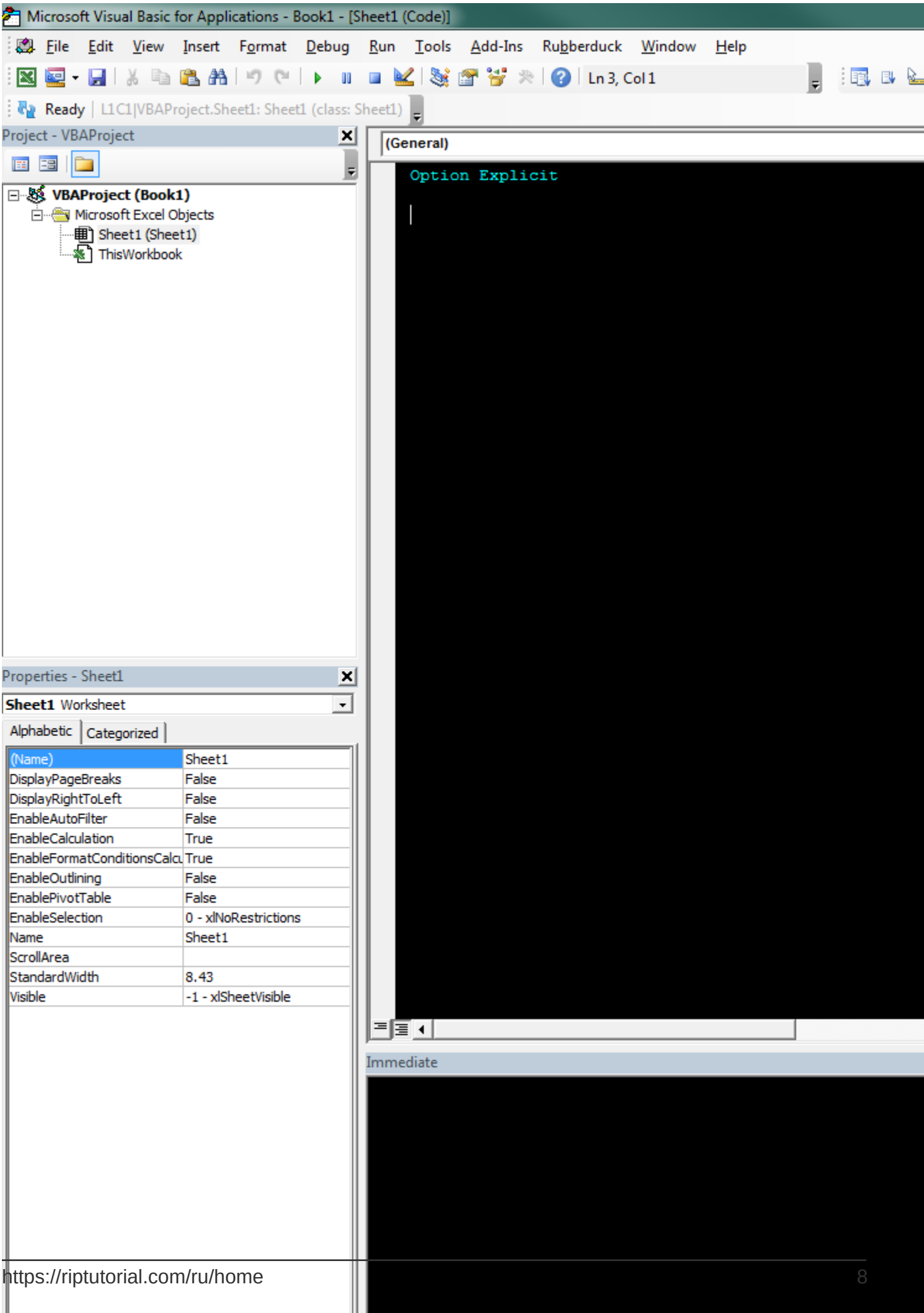

File
Home
Insert
Page Layout
Formulas
Data
Review
View
Developer
Tell me what you wa
Cut
Copy
Format Painter
Paste
Century gothic
10
Wrap Text
General
Merge & Center
Clipboard
Font
Alignment
Number
A1
Sheet1

Visual Basic»,

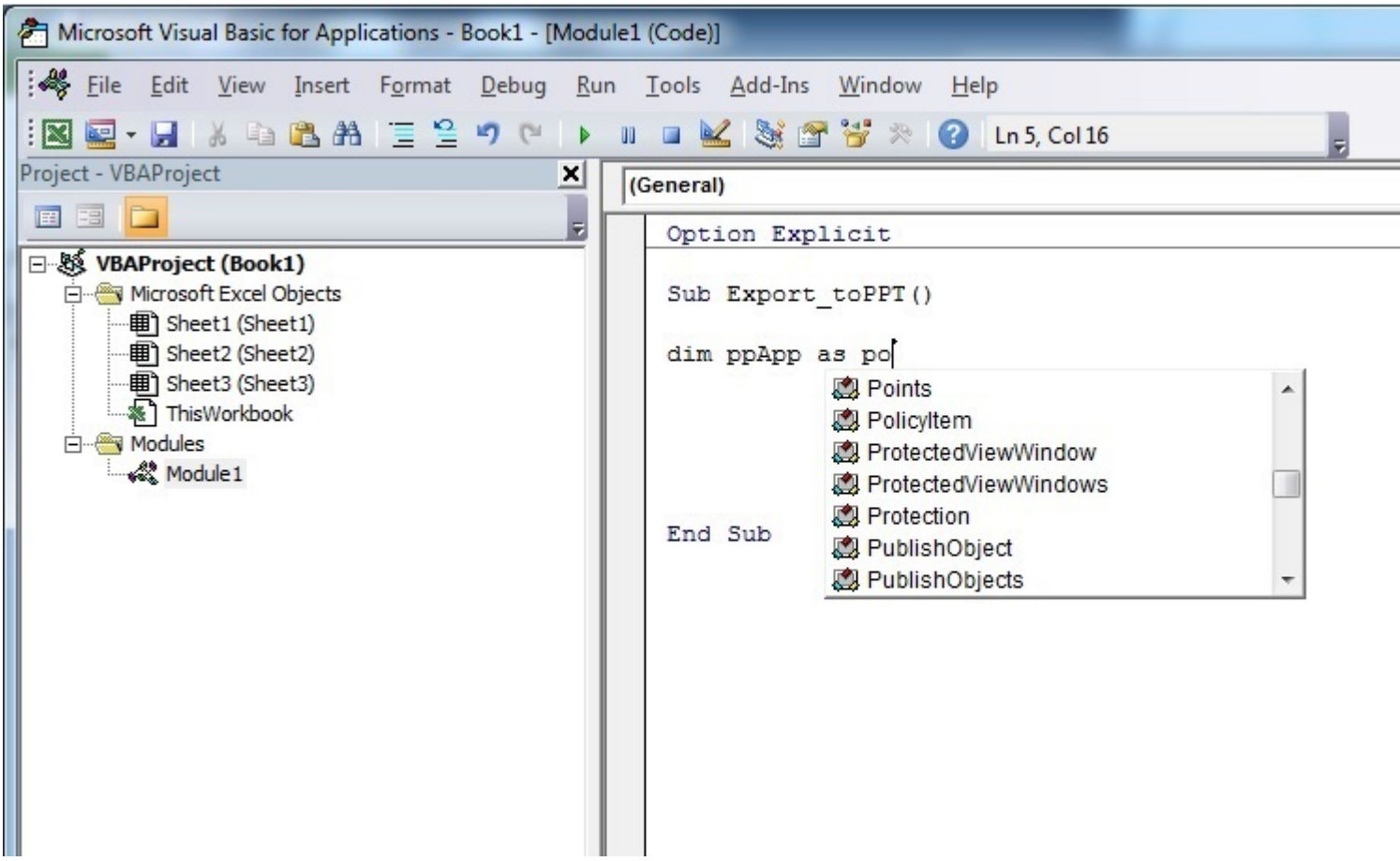


Шаг 2 Вариант C: вкладка «Вид»> «Макросы»> нажмите кнопку «Изменить», чтобы открыть существующий макрос

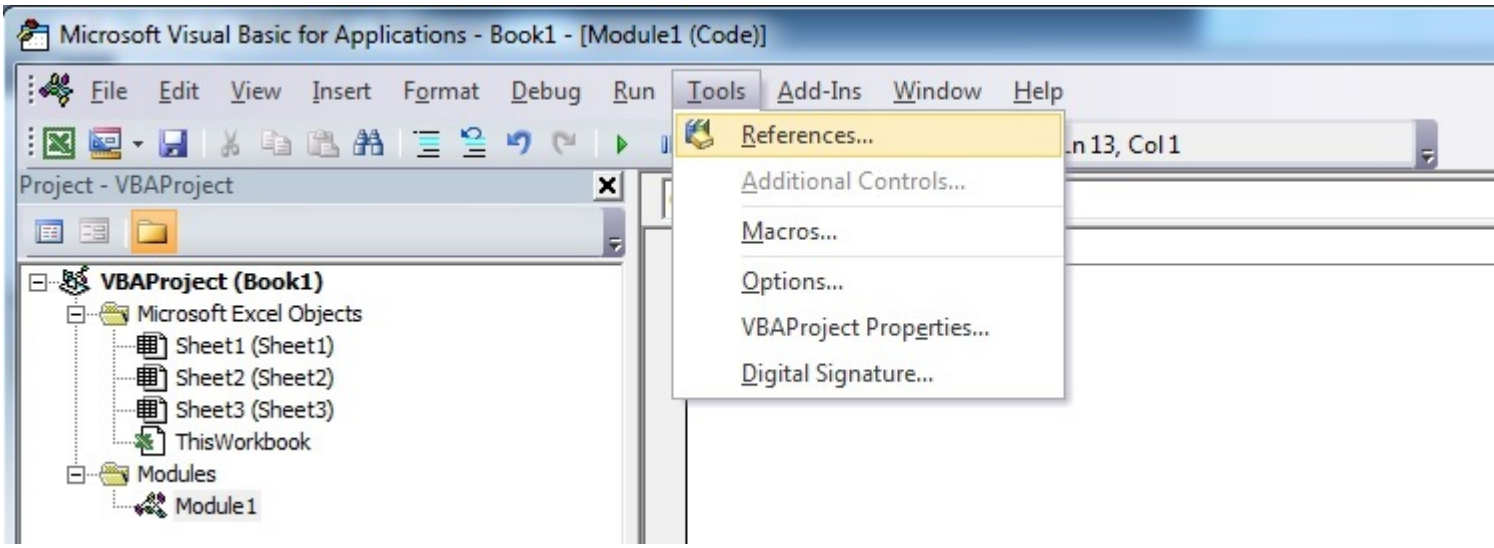
Все три из этих параметров откроют редактор Visual Basic (VBE):


Microsoft Visual Basic for Applications - Book1 - [Sheet1 (Code)]
File Edit View Insert Format Debug Run Tools Add-Ins Rubberduck Window Help
Ln 3, Col 1
Ready
L1C1|VBAProject.Sheet1: Sheet1 (class: Sheet1)
Project - VBAProject
VBAProject (Book1)
Microsoft Excel Objects
Sheet1 (Sheet1)
ThisWorkbook
(General)
Option Explicit
Properties - Sheet1
Sheet1 Worksheet
Alphabetic
Categorized
(Name) Sheet1
DisplayPageBreaks False
DisplayRightToLeft False
EnableAutoFilter False
EnableCalculation True
EnableFormatConditionsCalcu True
EnableOutlining False
EnablePivotTable False
EnableSelection 0 - xlNoRestrictions
Name Sheet1
ScrollArea
StandardWidth 8.43
Visible -1 - xlSheetVisible
Immediate

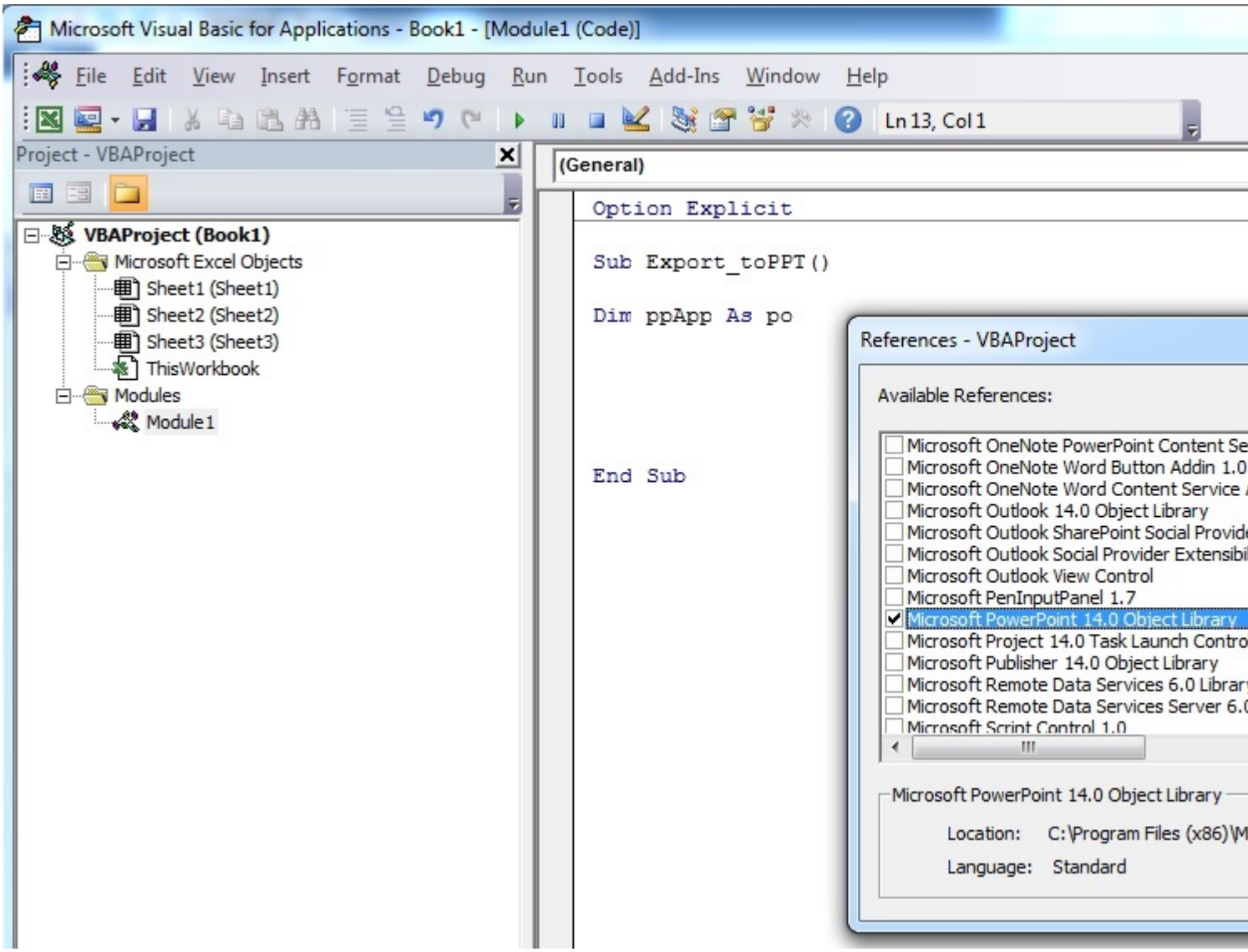
в существующий проект VB. Как видно, в настоящее время библиотека объектов PowerPoint недоступна.



Шаг 1 : выберите ***Инструменты*** меню ***-> Ссылки ...***

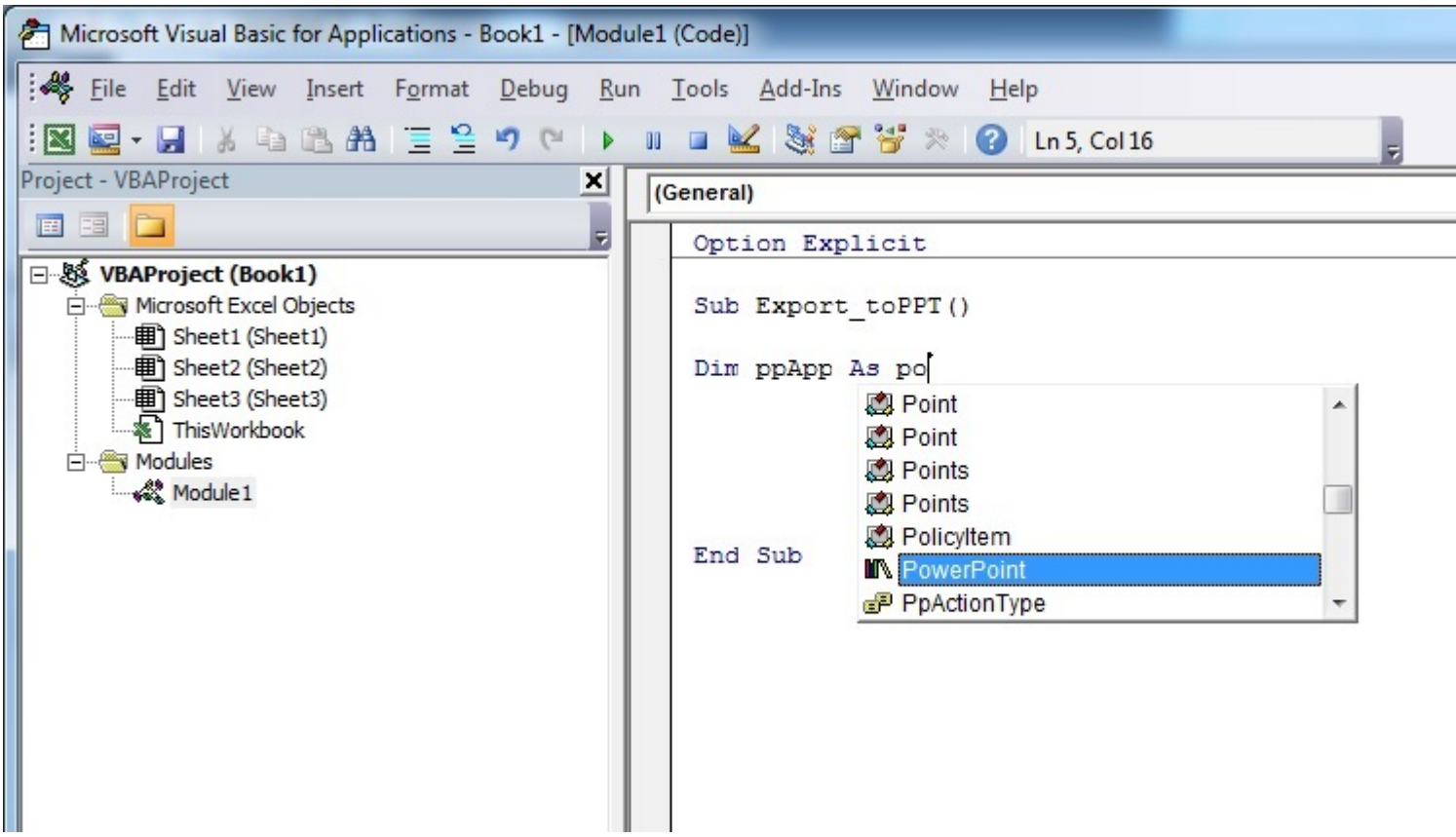


Шаг 2. Выберите ссылку, которую вы хотите добавить. В этом примере прокрутите страницу вниз, чтобы найти « ***Библиотека объектов Microsoft PowerPoint 14.0*** », а затем нажмите « **OK** ».

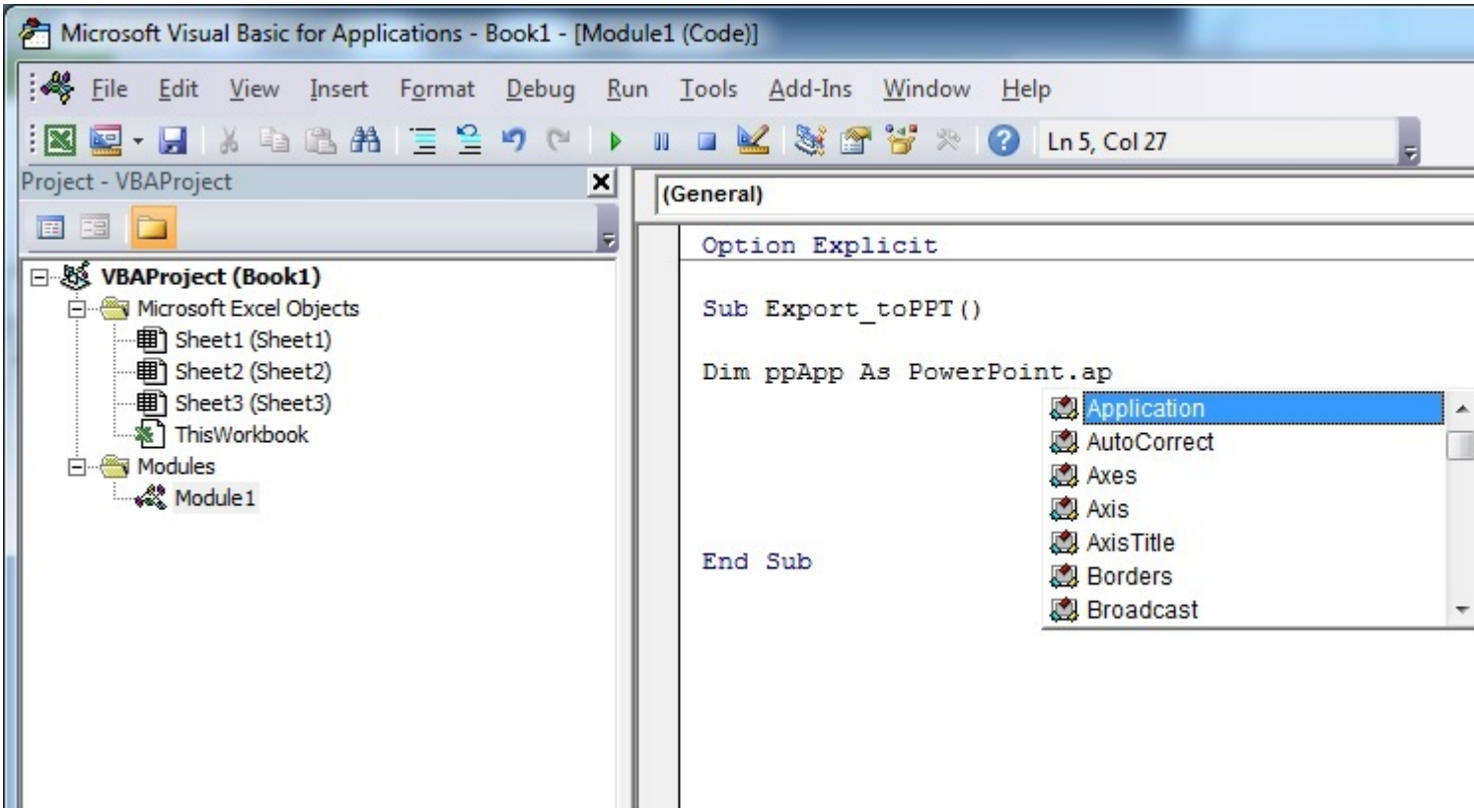


Примечание. PowerPoint 14.0 означает, что версия Office 2010 установлена на ПК.

Шаг 3 : в редакторе VB, как только вы нажмете **Ctrl + Space** вместе, вы получите опцию автозаполнения PowerPoint.

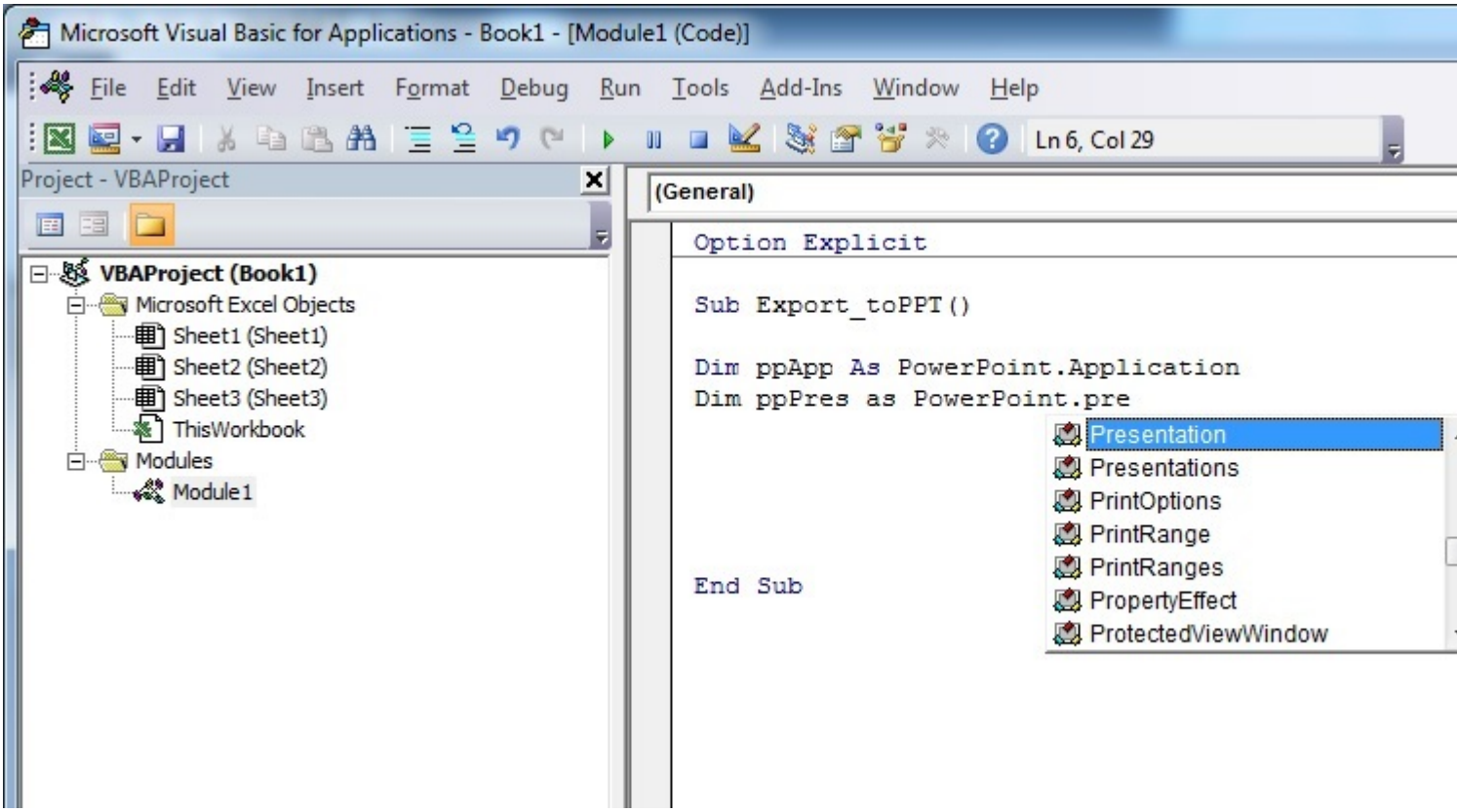


После выбора `PowerPoint` и нажатия `.` , появляется другое меню со всеми объектами, связанными с библиотекой объектов PowerPoint. В этом примере показано, как выбрать `Application` PowerPoint.

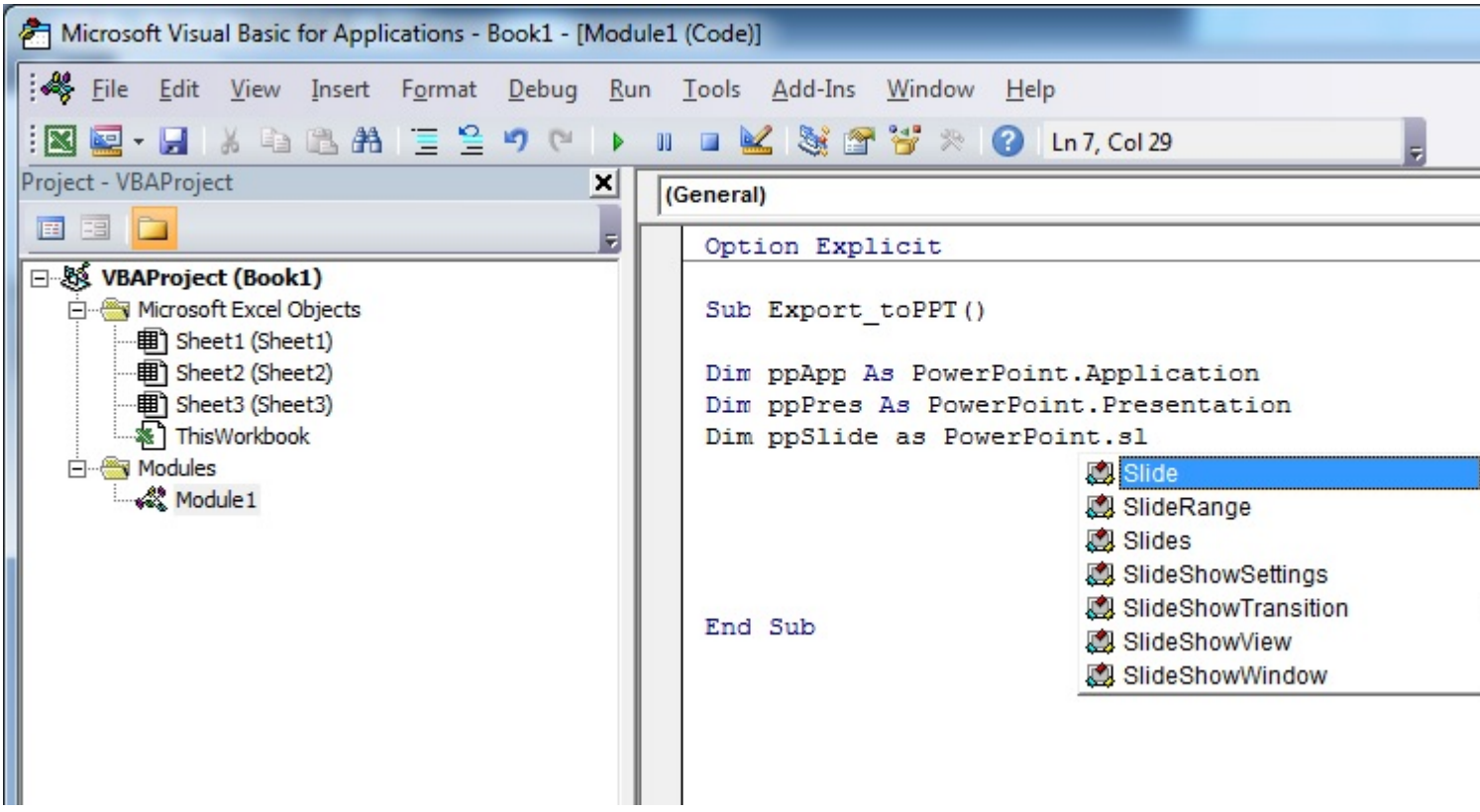


Шаг 4. Теперь пользователь может объявить больше переменных, используя библиотеку объектов PowerPoint.

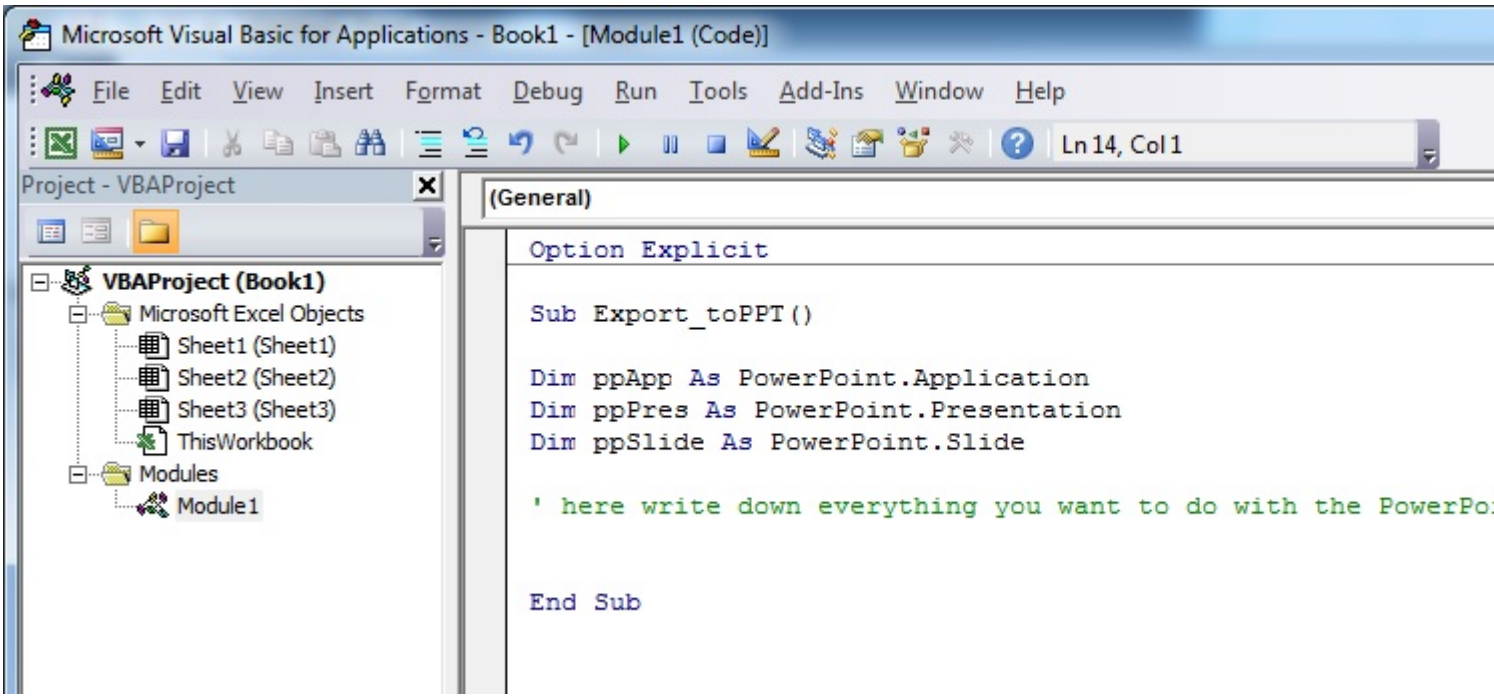
Объявите переменную, ссылающуюся на объект `Presentation` библиотеки объектов PowerPoint.



Объявите другую переменную, ссылающуюся на объект `Slide` библиотеки объектов PowerPoint.



Теперь секция объявления переменных выглядит как на снимке экрана ниже, и пользователь может начать использовать эти переменные в своем коде.

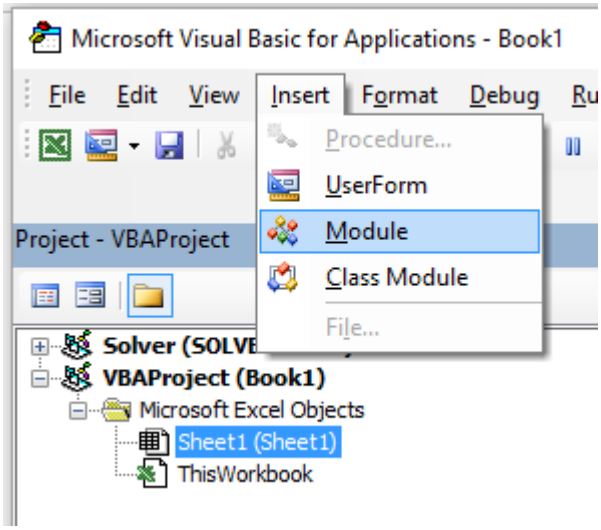


Код версии этого учебника:

```
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
```

Привет, мир

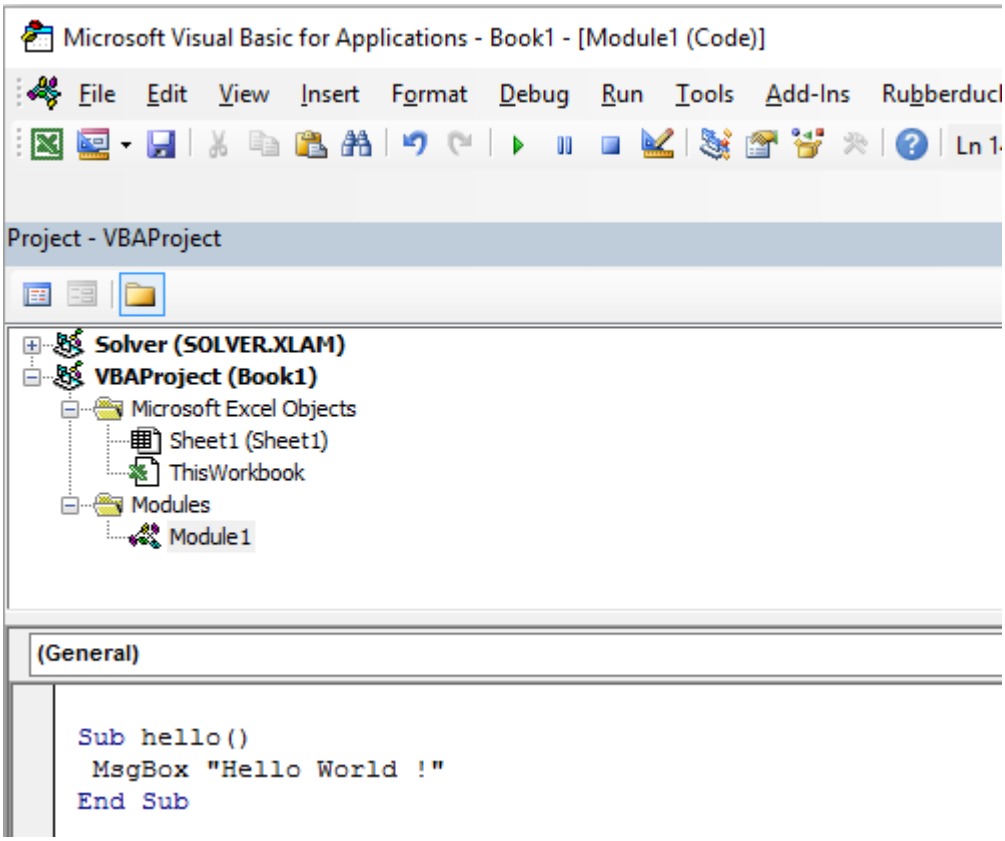
1. Откройте редактор Visual Basic (см. Раздел Открытие редактора Visual Basic)
2. Нажмите «Вставить» -> «Модуль», чтобы добавить новый модуль:



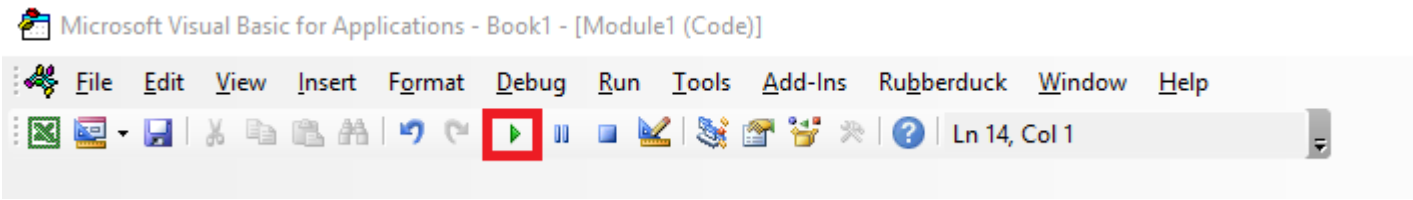
3. Скопируйте и вставьте следующий код в новый модуль:

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

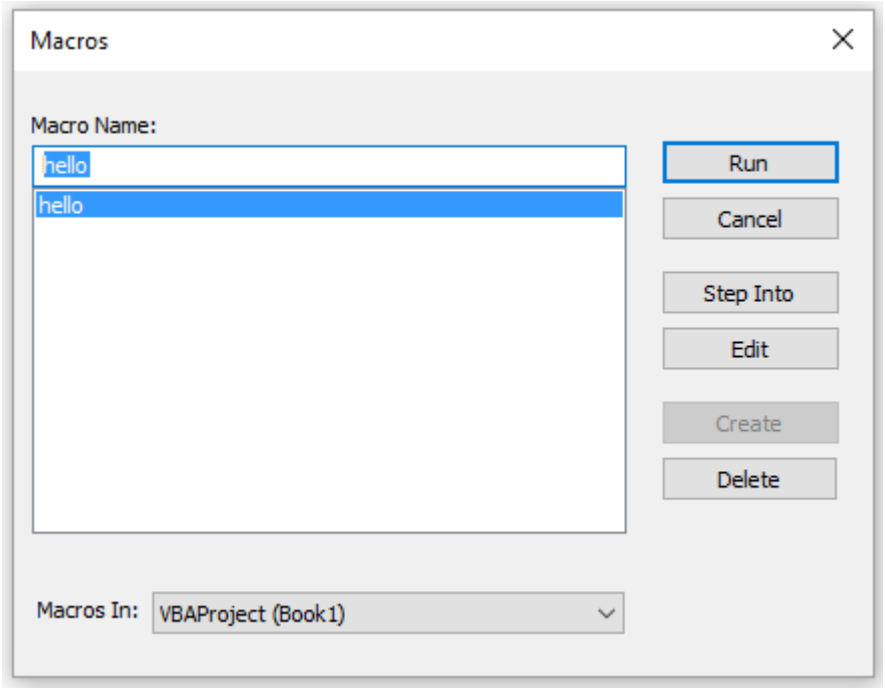
Чтобы получить :



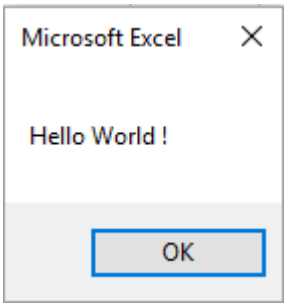
4. Нажмите на зеленую стрелку «play» (или нажмите F5) на панели инструментов Visual Basic, чтобы запустить программу:



5. Выберите новый созданный вспомогательный «привет» и нажмите « `Run` :



6. Сделано, вы должны увидеть следующее окно:

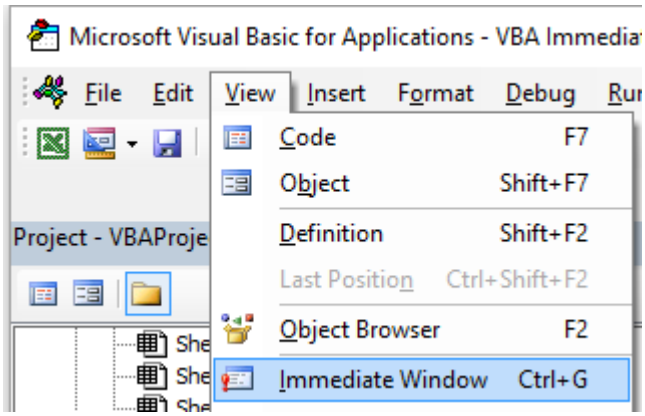


Начало работы с объектной моделью Excel

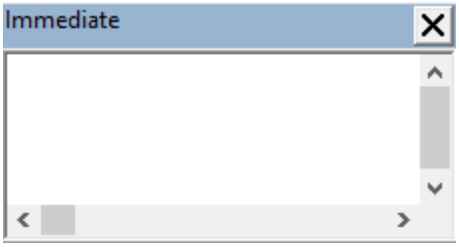
Этот пример намеревается быть нежным знакомством с объектной моделью Excel **для начинающих** .

---

1. Откройте редактор Visual Basic (VBE)
2. Нажмите «Вид» -> «Немедленное окно», чтобы открыть окно «Немедленное» (или `Ctrl` + `G`):



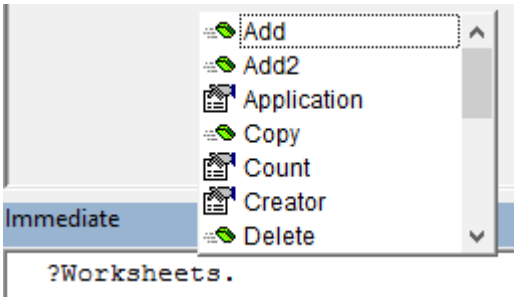
3. Вы должны увидеть следующее Немедленное Окно внизу на VBE:



Это окно позволяет вам непосредственно тестировать код VBA. Итак, давайте начнем, введите эту консоль:

```
?Worksheets.
```

VBE имеет intellisense, а затем он должен открыть всплывающую подсказку, как на следующем рисунке:



Выберите .Count в списке или непосредственно введите `.Cout` чтобы получить:

```
?Worksheets.Count
```

4. Затем нажмите Enter. Выражение оценивается, и оно должно возвращаться 1. Это указывает количество Рабочего листа, которое в настоящее время присутствует в книге. Значок вопроса (`?`) Является псевдонимом для Debug.Print.

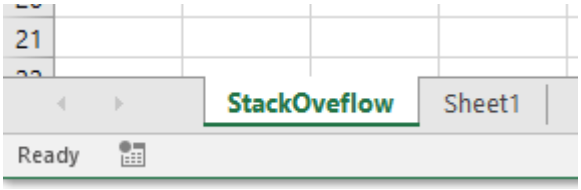
Рабочие листы - это **объект,** а граф - **метод** . Excel имеет несколько объектов (`Workbook` , `Worksheet` , `Range` , `Chart` ..), и каждый из них содержит специальные методы и свойства. Полный список объектов можно найти в справочной системе Excel VBA . Рабочий лист Объект представлен здесь .

Эта ссылка Excel VBA должна стать вашим основным источником информации об объектной модели Excel.

5. Теперь давайте попробуем другое выражение, введите (без символа `?`):

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

6. Нажмите Ввод. Это должно создать новый рабочий лист под названием `StackOverflow.` :



Чтобы понять это выражение, вам нужно прочитать функцию «Добавить» в вышеупомянутой ссылке Excel. Вы найдете следующее:

```
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,
 or macro sheet.
```

Поэтому `Worksheets.Add()` создает новый рабочий лист и возвращает его. Рабочий лист (**без s**) сам по себе является объектом, который можно найти в документации, а `Name` - одно из его **свойств** (см. Здесь). Он определяется как:

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

Итак, исследуя определения различных объектов, мы можем понять этот код `Worksheets.Add().Name = "StackOveflow"` .

`Add()` создает и добавляет новый рабочий лист и возвращает **ссылку** на него, тогда мы устанавливаем его **свойство** Name в значение "StackOverflow"

Теперь давайте будем более формальными, Excel содержит несколько объектов. Эти объекты могут состоять из одного или нескольких коллекций объектов Excel того же класса. Это относится к `WorkSheets` который представляет собой коллекцию объекта `Worksheet` . Каждый объект имеет некоторые свойства и методы, с которыми может взаимодействовать программист.

Модель объекта Excel относится к **иерархии объектов** Excel

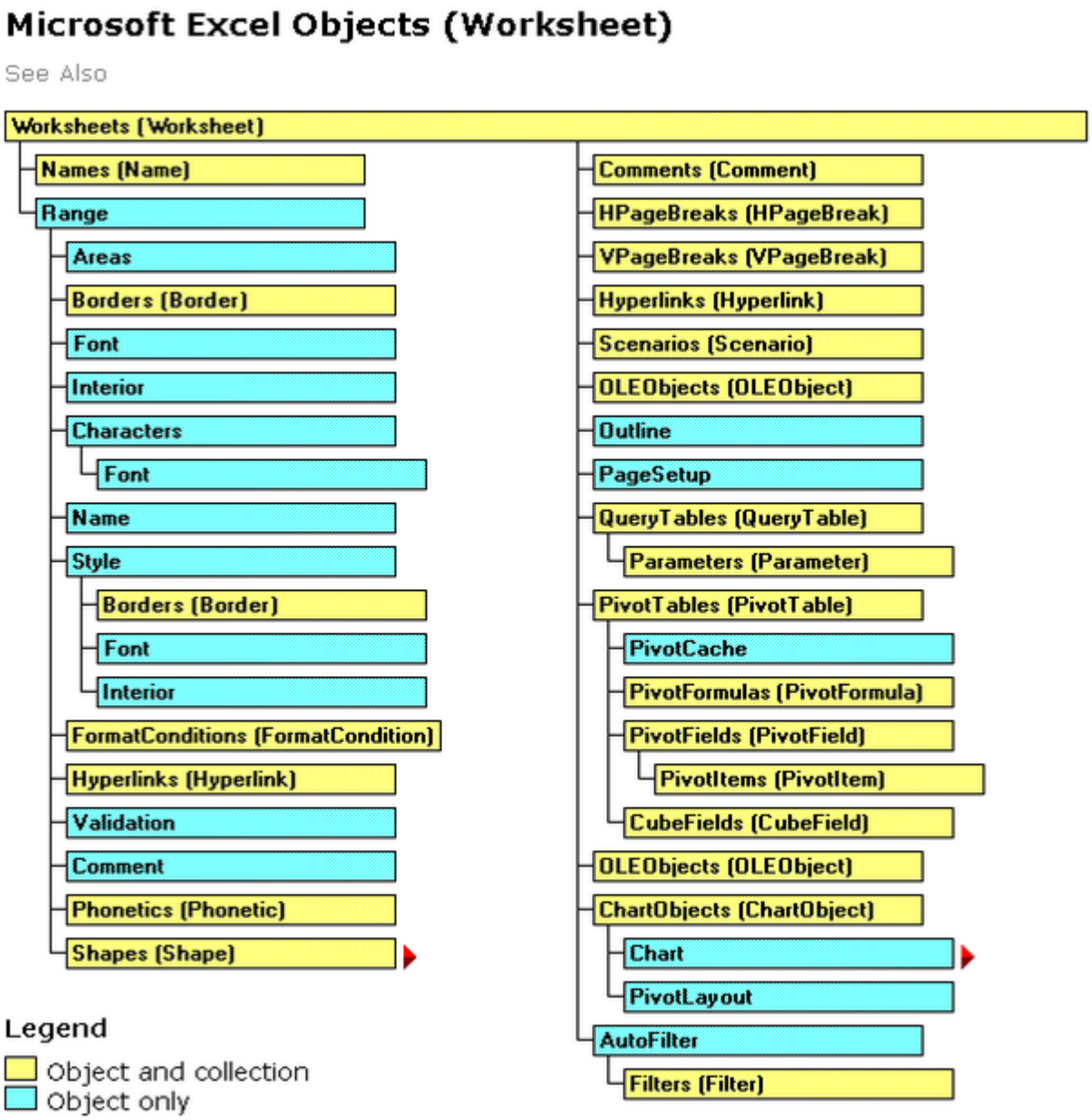
В верхней части всех объектов находится объект `Application` , он представляет собой экземпляр Excel. Программирование в VBA требует хорошего понимания этой иерархии,

потому что нам всегда нужна ссылка на объект, чтобы иметь возможность вызвать метод или установить / получить свойство.

(Очень упрощенную) Модель объекта Excel может быть представлена как,

```
Application
 Workbooks
 Workbook
Worksheets
 Worksheet
  Range
```

Более подробная версия для объекта Worksheet (как в Excel 2007) показана ниже,



Полную модель объектов Excel можно найти здесь .

Наконец, некоторые объекты могут иметь `events` (например: `Workbook.WindowActivate`), которые также являются частью объектной модели Excel.

Прочитайте Начало работы с excel-vba онлайн: https://riptutorial.com/ru/excel-vba/topic/777/начало-работы-с-excel-vba

глава 2: CustomDocumentProperties на практике

Вступление

Использование CustomDocumentProperties (CDP) - хороший метод для хранения определенных пользователем значений относительно безопасным способом в пределах одной рабочей книги, но избегая показывать связанные значения соты просто в незащищенном рабочем листе *).

Примечание. CDP представляют собой отдельную коллекцию, сопоставимую с BuiltInDocumentProperties, но позволяют создавать собственные имена свойств пользователя, а не фиксированную коллекцию.

*) Кроме того, вы можете вводить значения также в скрытую или «очень скрытую» книгу.

Examples

Организация новых номеров счетов

Частая задача - увеличение номера счета и сохранение его стоимости. Использование CustomDocumentProperties (CDP) - хороший метод для хранения таких чисел относительно безопасным способом в пределах одной рабочей книги, но избегая показывать связанные значения соты просто на незащищенном рабочем листе.

Дополнительный намек:

Кроме того, вы можете вводить значения также в скрытом листе или даже в так называемом «очень скрытом» листе (см. Использование xlVeryHidden Sheets . Конечно, можно сохранять данные также во внешние файлы (например, ini-файл, csv или любой другой тип) или реестра.

Пример содержимого :

Пример ниже показывает

- функция NextInvoiceNo, которая устанавливает и возвращает следующий номер счета-фактуры,
- процедура DeleteInvoiceNo, которая полностью удаляет CDP счета-фактуры, а также
- процедура showAllCDPs перечисляет полную коллекцию CDP со всеми именами. Не используя VBA, вы также можете перечислить их через информацию о книге: Info | Свойства [DropDown:] | Дополнительные свойства | изготовленный на заказ

Вы можете получить и установить следующий номер счета (последний нет плюс один), просто вызвав вышеупомянутую функцию, возвращая строковое значение, чтобы облегчить добавление префиксов. «InvoiceNo» неявно используется как имя CDP во всех процедурах.

```
Dim sNumber As String
sNumber = NextInvoiceNo ()
```

Пример кода:

```
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
' Set workbook and CDPs
  Set wb = ThisWorkbook
  Set prop = wb.CustomDocumentProperties

  ' ------------------------------------------------------
  ' Generate new CDP "InvoiceNo" if not yet existing
  ' ------------------------------------------------------
    If Not CDPExists("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 CDPExists(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-
customdocumentproperties-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
  CDPExists = 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 CDPExists("InvoiceNo") Then
    prop("InvoiceNo").Delete
 End If
```

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
```

Прочитайте CustomDocumentProperties на практике онлайн: https://riptutorial.com/ru/excel-vba/topic/10932/customdocumentproperties-на-практике

глава 3: SQL в Excel VBA - лучшие практики

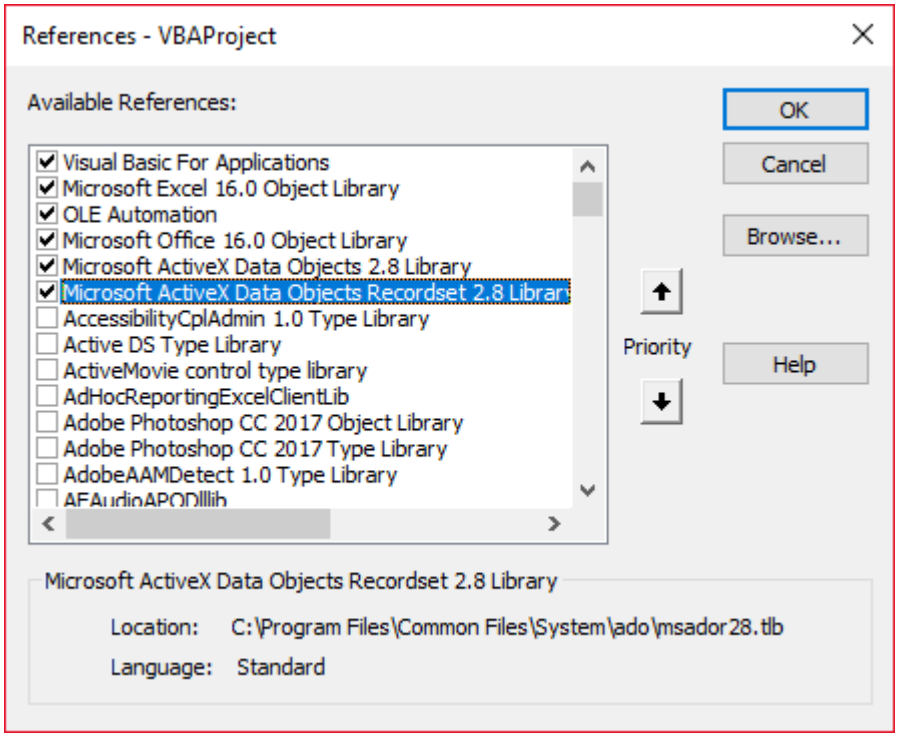
Examples

Как использовать ADODB.Connection в VBA?

Требования:

Добавьте следующие ссылки на проект:

- Microsoft ActiveX Data Objects 2.8 Библиотека
- Microsoft ActiveX Data Objects Recordset 2.8 Library



Объявить переменные

```
Private mDataBase As New ADODB.Connection
Private mRS As New ADODB.Recordset
Private mCmd As New ADODB.Command
```

Создать соединение

а. с проверкой подлинности Windows

```
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
```

б. с проверкой подлинности SQL Server

```
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
```

Выполнить команду sql

```
Private Sub ExecuteCmd(sql As String)
    mCmd.CommandText = sql
    Set mRS = mCmd.Execute
End Sub
```

Чтение данных из набора записей

```
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
```

Закрыть соединение

```
Private Sub CloseConnection()
    Call mDataBase.Close
    Set mRS = Nothing
    Set mCmd = Nothing
    Set mDataBase = Nothing
```

```
End Sub
```

Как это использовать?

```
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
```

Результат

ShipperID: 1 Название компании: Speedy Express Телефон: (503) 555-9831

ShipperID: 2 CompanyName: United Package Телефон: (503) 555-3199

ShipperID: 3 CompanyName: Federal Shipping Телефон: (503) 555-9931

ShipperID: 4 Название компании: быстрая доставка Телефон: (503) 555-1234

Прочитайте SQL в Excel VBA - лучшие практики онлайн: https://riptutorial.com/ru/excel-vba/topic/9958/sql-в-excel-vba---лучшие-практики

глава 4: Workbooks

Examples

Приложения

Во многих приложениях Excel код VBA выполняет действия, направленные на книгу, в которой он содержится. Вы сохраняете эту книгу с расширением «.xlsm», а макросы VBA фокусируются только на рабочих листах и данных внутри. Однако часто бывает, когда вам нужно объединять или объединять данные из других книг или записывать некоторые данные в отдельную книгу. Открытие, закрытие, сохранение, создание и удаление других книг является общей потребностью для многих приложений VBA.

В любое время в редакторе VBA вы можете просматривать и получать доступ к любым и всем рабочим книгам, которые в настоящее время открываются этим экземпляром Excel, с использованием свойства `Workbooks` объекта `Application` . Документация MSDN объясняет это ссылками.

Когда использовать ActiveWorkbook и эту книгу

Это лучшая практика VBA, чтобы всегда указывать ту книгу, на которую ссылается ваш код VBA. Если эта спецификация опущена, то VBA предполагает, что код направлен на `ActiveWorkbook` рабочую книгу (`ActiveWorkbook`).

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

Однако, когда несколько книг одновременно открыты - особенно, особенно, когда код VBA запущен из надстройки Excel, ссылки на `ActiveWorkbook` могут быть сбиты с толку или неправильно направлены. Например, надстройка с UDF, которая проверяет время суток и сравнивает ее со значением, хранящимся на одном из рабочих листов надстройки (которые обычно не видны пользователю), должно будет явно определить, какая книга ссылаясь. В нашем примере наша открытая (и активная) рабочая тетрадь имеет формулу в ячейке A1 `=EarlyOrLate()` и не имеет никакого VBA, написанного для этой активной книги. В нашей надстройке мы имеем следующую пользовательскую функцию (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
```

Код для UDF записывается и сохраняется в установленной надстройке Excel. Он использует данные, хранящиеся на листе в надстройке под названием «WatchTime». Если UDF использовал `ActiveWorkbook` вместо `ThisWorkbook` , тогда он никогда не сможет гарантировать, какая книга была предназначена.

Открытие книги (новая), даже если она уже открыта

Если вы хотите открыть книгу, которая уже открыта, то получение задания из коллекции `Workbooks` является простым:

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

Если вы хотите создать новую книгу, а затем использовать `Workbooks` объект коллекции , чтобы `Add` новую запись.

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

Бывают случаи, когда вы не можете или (или не заботитесь), если рабочая книга, которая вам нужна, уже открыта или нет, или возможно, не существует. Примерная функция показывает, как всегда возвращать действительный объект рабочей книги.

```
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
        '--- 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
```

Сохранение книги без запроса пользователя

Зачастую сохранение новых данных в существующей книге с помощью VBA вызовет всплывающий вопрос, отметив, что файл уже существует.

Чтобы предотвратить этот всплывающий вопрос, вам необходимо подавить эти типы предупреждений.

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

Изменение стандартного количества рабочих листов в новой рабочей книге

Заданное по умолчанию количество рабочих листов, созданных в новой книге Excel, обычно равно трем. Ваш код VBA может явно указывать количество рабочих листов в новой книге.

```
'--- 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
```

Прочитайте Workbooks онлайн: https://riptutorial.com/ru/excel-vba/topic/2969/workbooks

глава 5: автофильтр; Использование и передовая практика

Вступление

Конечная цель ***Autofilter*** заключается в том, чтобы как можно быстрее обеспечить сбор данных из сотен или тысяч данных строк, чтобы привлечь внимание к предметам, на которые мы хотим сосредоточиться. Он может получать такие параметры, как «текст / значения / цвета», и они могут быть уложены между столбцами. Вы можете подключить до двух критериев для каждого столбца на основе логических соединителей и наборов правил. Примечание. Автофильтр работает путем фильтрации строк, для фильтрации столбцов (по крайней мере, не изначально) нет фильтра автофильтра.

замечания

«Чтобы использовать Autofilter в VBA, нам нужно вызвать, по крайней мере, следующие параметры:

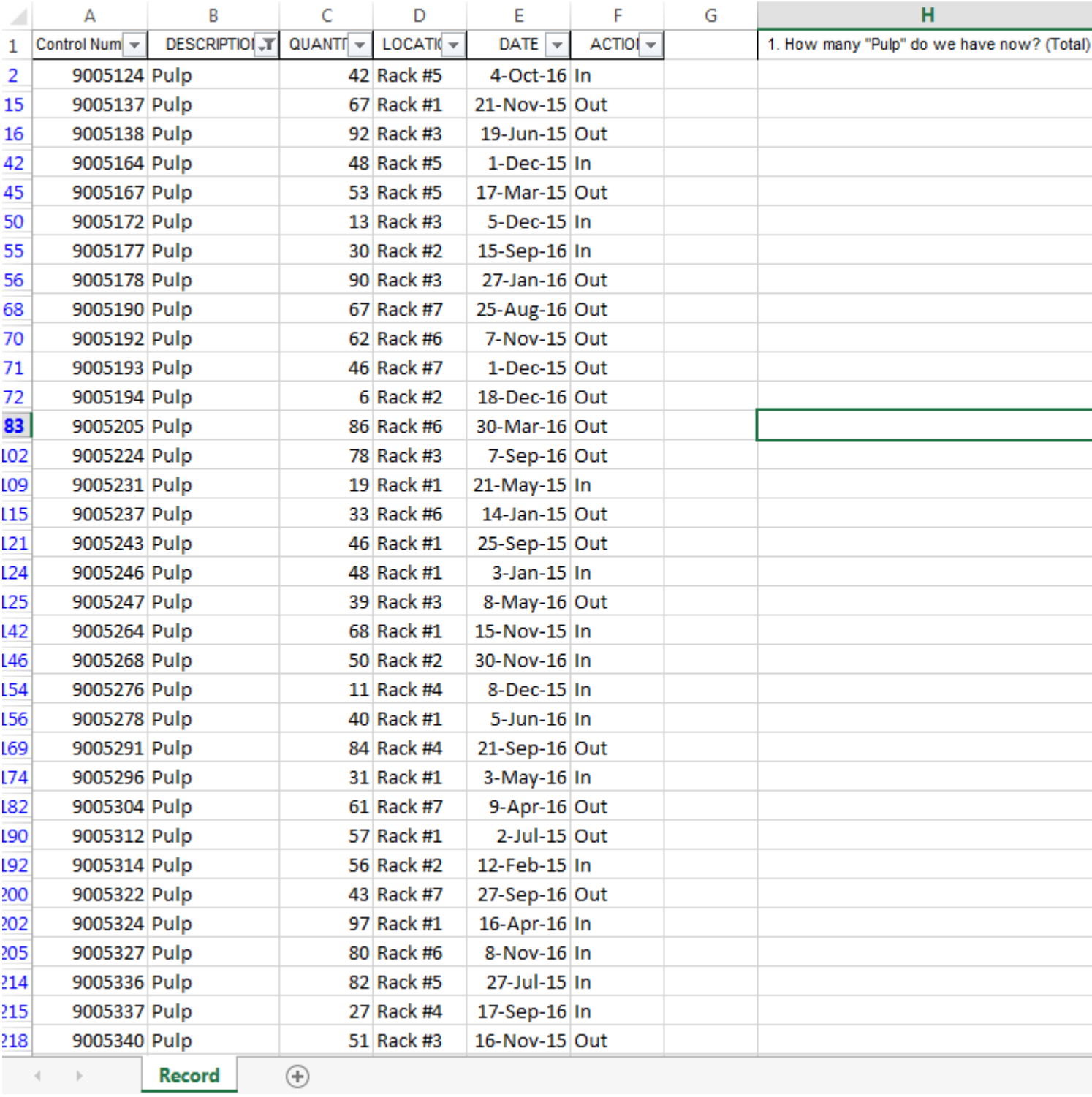
Лист («MySheet»). Диапазон («MyRange»). Поле Autofilter = (ColumnNumberWithin «MyRange» ToBeFilteredInNumericValue) Критерии1: = «WhatIWantToFilter»

«Есть много примеров либо в Интернете, либо здесь, в stackoverflow

Examples

Smartfilter!

Проблемная ситуация
Администратор склада имеет листок («Запись»), где хранится каждое движение логистики, выполняемое средством, он может фильтровать по мере необходимости, хотя это очень трудоемко, и он хотел бы улучшить процесс, чтобы быстрее вычислить запросы, для Пример. Сколько у нас «целлюлозы» (во всех стойках)? Сколько целлюлозы мы имеем сейчас (в стойке № 5)? Фильтры - отличный инструмент, но они несколько ограничены, чтобы ответить на этот вопрос в считанные секунды.



	A	B	C	D	E	F	G	H
1	Control Num	DESCRIPTIO	QUANTI	LOCATI	DATE	ACTIO		1. How many "Pulp" do we have now? (Total)
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		

Record

Макро-решение:

Кодер знает, что ***автофильтры - лучшее, быстрое и надежное решение*** в таких сценариях, поскольку ***данные уже есть на рабочем листе,*** и ***вход для них может быть легко получен*** - в этом случае - с помощью пользовательского ввода.

Используемый подход заключается в создании листа под названием «SmartFilter», где администратор может легко фильтровать несколько данных по мере необходимости, а расчет также выполняется мгновенно.

Он использует 2 модуля и событие `Worksheet_Change` для этого вопроса

Код для рабочего листа SmartFilter:

```
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
```

Код для модуля 1, называемый «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,
```

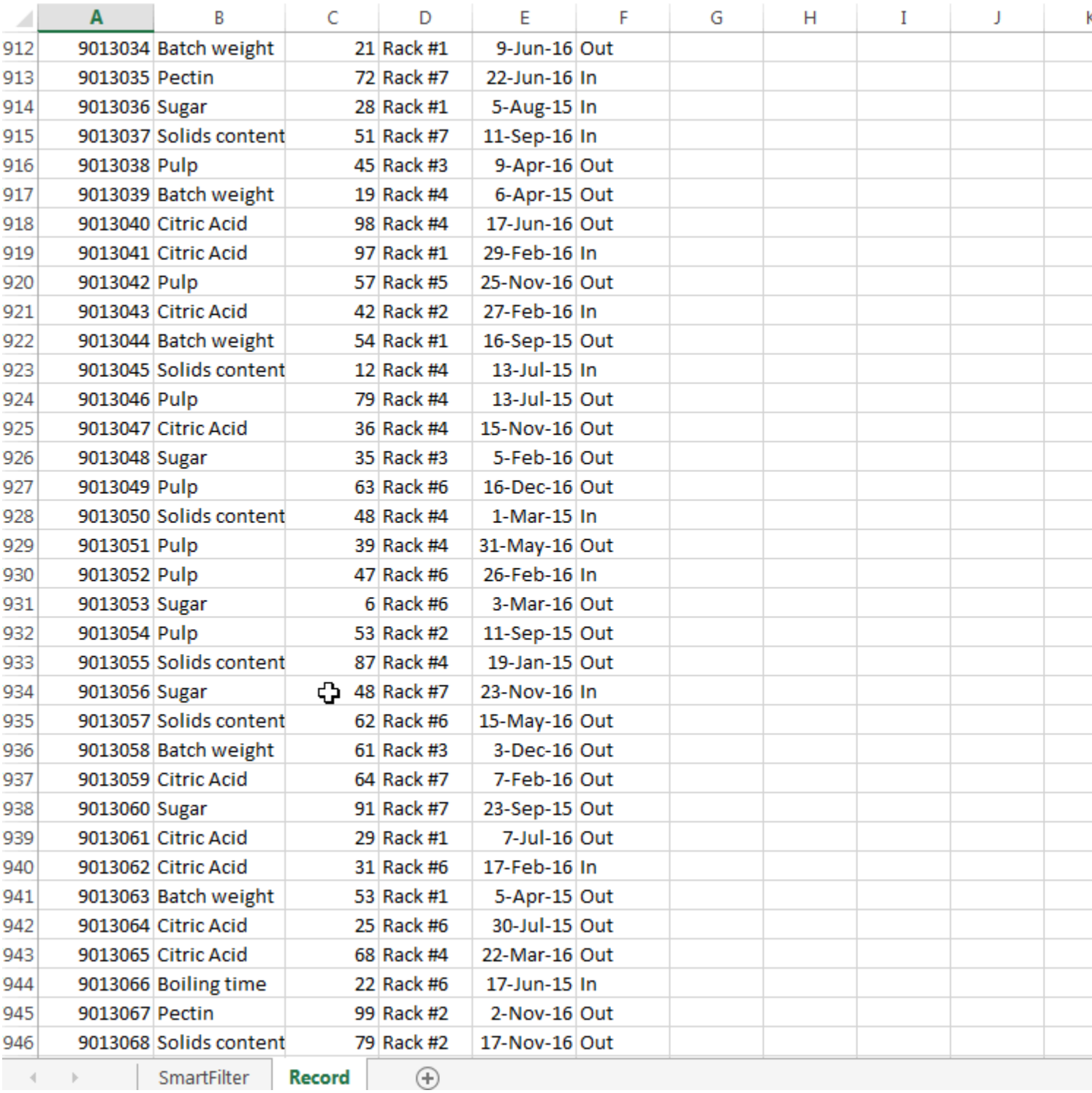
```
Sheets(InSheet).Range(InRange.Address).Find(TitleToFind, LookAt:=xlWhole),
Sheets(InSheet).Range(InRange.Address).Find(TitleToFind, LookAt:=xlPart))
    End If ' 1. If InRange Is Nothing
    Set General_Functions_Find_Title = DummyRange
    If 1 = 2 Or DummyRange Is Nothing Then '99. If error
Err01General_Functions_Find_Title:
    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
```

Код для модуля 2, называемый "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 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
```

Тестирование и результаты:



	A	B	C	D	E	F	G	H	I	J
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

Как мы видели на предыдущем изображении, эта задача была достигнута легко. Используя ***автофильтры,*** было предоставлено решение, которое ***требует*** всего лишь ***секунд для вычисления, легко объяснить пользователю, так*** как он / она знаком с этой командой, и ***сделал несколько строк для кодера.***

Прочитайте автофильтр; Использование и передовая практика онлайн: https://riptutorial.com/ru/excel-vba/topic/8645/автофильтр--использование-и-передовая-практика

глава 6: Безопасность VBA

Examples

Пароль Защитите свой VBA

Иногда у вас есть конфиденциальная информация в вашем VBA (например, пароли), к которой вы не хотите, чтобы пользователи имели доступ. Вы можете обеспечить базовую безопасность этой информацией, защищая паролем свой проект VBA.

Следуй этим шагам:

1. Откройте редактор Visual Basic (Alt + F11)
2. Перейдите в Инструменты -> Свойства VBAProject ...
3. Перейдите на вкладку «Защита»
4. Отметьте флажок «Заблокировать проект для просмотра»
5. Введите желаемый пароль в текстовые поля «Пароль» и «Подтверждение пароля»

Теперь, когда кто-то хочет получить доступ к вашему коду в приложении Office, им сначала нужно будет ввести пароль. Имейте в виду, однако, что даже сильный пароль проекта VBA тривиальен для разрыва.

Прочитайте Безопасность VBA онлайн: https://riptutorial.com/ru/excel-vba/topic/7642/безопасность-vba

глава 7: Графики и диаграммы

Examples

Создание диаграммы с диапазонами и фиксированное имя

Графики могут быть созданы путем непосредственной работы с объектом `Series` который определяет данные диаграммы. Чтобы перейти к `Series` без диаграммы exisitng, вы создаете `ChartObject` на данном `Worksheet` а затем получаете объект `Chart` из него. Поверхность работы с объектом « `Series` заключается в том, что вы можете установить `Values` и `XValues` , обратившись к объектам `Range` . Эти свойства данных будут правильно определять `Series` со ссылками на эти диапазоны. Недостатком этого подхода является то, что при настройке `Name` не обрабатывается одно и то же преобразование; это фиксированное значение. Он не будет корректироваться с базовыми данными в исходном `Range` . Проверка формулы `SERIES` и очевидно, что имя исправлено. Это необходимо обработать, создав формулу `SERIES` напрямую.

Код, используемый для создания диаграммы

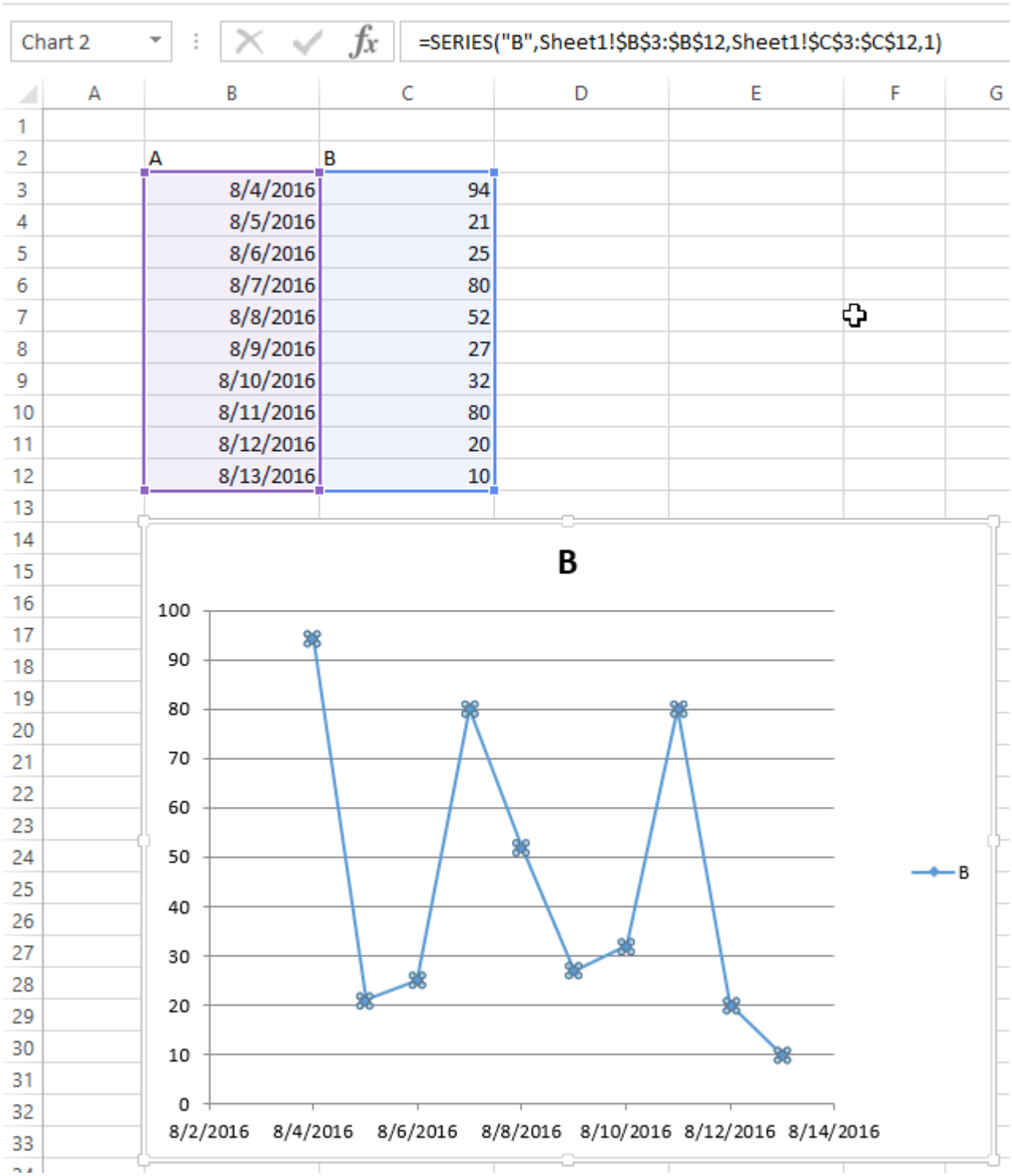
Обратите внимание, что этот код содержит объявления дополнительных переменных для `Chart` и `Worksheet` . Они могут быть опущены, если они не используются. Они могут быть полезны, однако, если вы изменяете стиль или любые другие свойства диаграммы.

```
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
```

Исходные данные / диапазоны и итоговая `Chart` после `Chart` кода

Обратите внимание, что формула `SERIES` включает в себя `"B"` для названия серии вместо ссылки на `Range` который ее создал.



Создание пустой диаграммы

Отправной точкой для подавляющего большинства графического кода является создание пустой `Chart` . Обратите внимание , что эта `Chart` является предметом шаблона диаграммы по умолчанию , который является активным и не может на самом деле быть пустым (если шаблон был изменен).

Ключ к `ChartObject` определяет его местоположение. Синтаксис вызова - `ChartObjects.Add(Left, Top, Width, Height)` . После создания `ChartObject` вы можете использовать его объект `Chart` для фактического изменения диаграммы. `ChartObject` ведет себя больше как `Shape` чтобы расположить диаграмму на листе.

Код для создания пустой диаграммы

```
Sub CreateEmptyChart()

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

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

End Sub
```

Результирующая диаграмма

Создание диаграммы путем изменения формулы SERIES

Для полного контроля над новым объектом `Chart` и `Series` (особенно для динамического названия `Series`) вы должны прибегнуть к модификации формулы `SERIES` напрямую. Процесс создания объектов `Range` является простым, и основным препятствием является просто построение строки для формулы `SERIES` .

Формула `SERIES` принимает следующий синтаксис:

```
=SERIES(Name,XValues,Values,Order)
```

Это содержимое может быть предоставлено в виде ссылок или значений массива для элементов данных. `Order` представляет собой серию позиций в диаграмме. Обратите внимание, что ссылки на данные не будут работать, если они не полностью соответствуют имени листа. Для примера рабочей формулы щелкните любую существующую серию и проверьте панель формул.

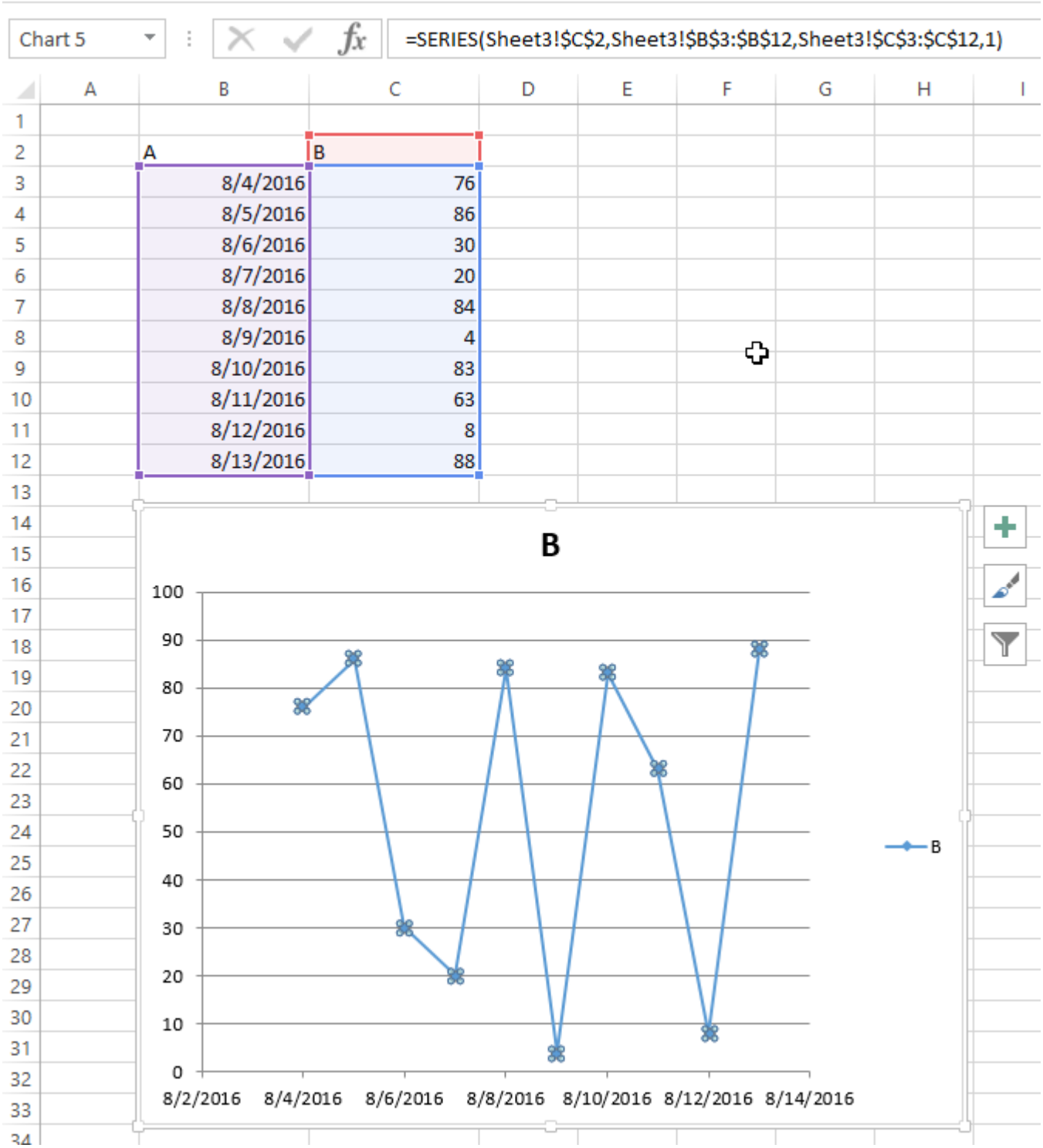
Код для создания диаграммы и настройки данных с использованием формулы `SERIES`

Обратите внимание, что построение строки для создания формулы `SERIES` использует `.Address(,,,True)` . Это гарантирует, что ссылка *внешнего* диапазона используется так, чтобы был включен полный адрес с именем листа. Вы **получите сообщение об ошибке, если имя листа исключено** .

```
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 refernce 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
```

Исходные данные и итоговая диаграмма

Обратите внимание, что для этой диаграммы имя серии правильно задано с диапазоном до нужной ячейки. Это означает, что обновления будут распространяться на `Chart` .



Размещение диаграмм в сетке

Обычная работа с графиками в Excel - это стандартизация размера и компоновки нескольких диаграмм на одном листе. Если сделать это вручную, вы можете удерживать `ALT` при изменении размера или перемещении диаграммы, чтобы «придерживаться» границ ячеек. Это работает для пары диаграмм, но подход VBA намного проще.

Код для создания сетки

Этот код создаст сетку диаграмм, начинающихся с заданной (верхней, левой) позиции, с

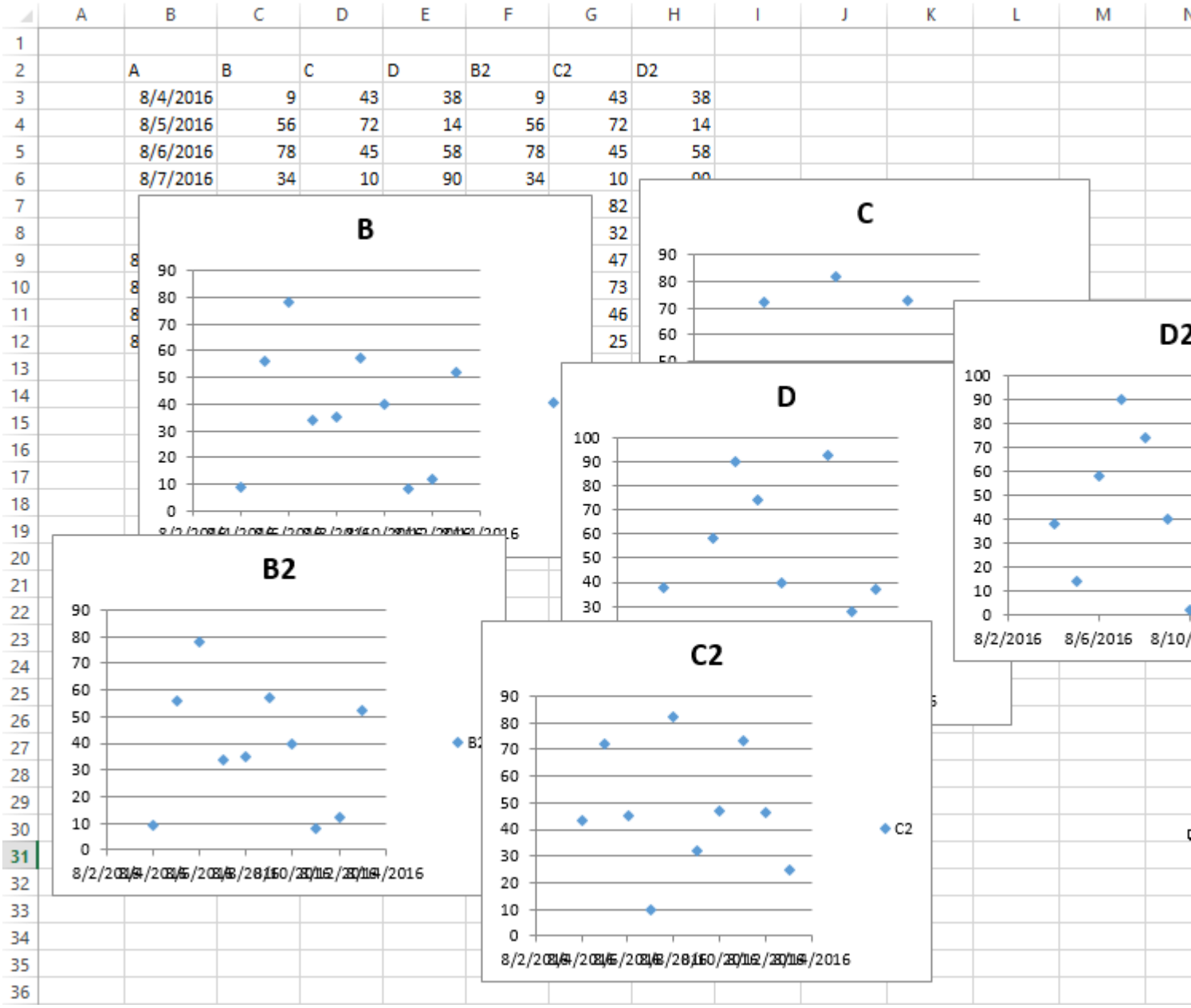
определенным количеством столбцов и определенным общим размером диаграммы. Графики будут размещены в том порядке, в котором они были созданы, и обернут вокруг края, чтобы сформировать новую строку.

```
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

    '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
```

Результат с несколькими графиками

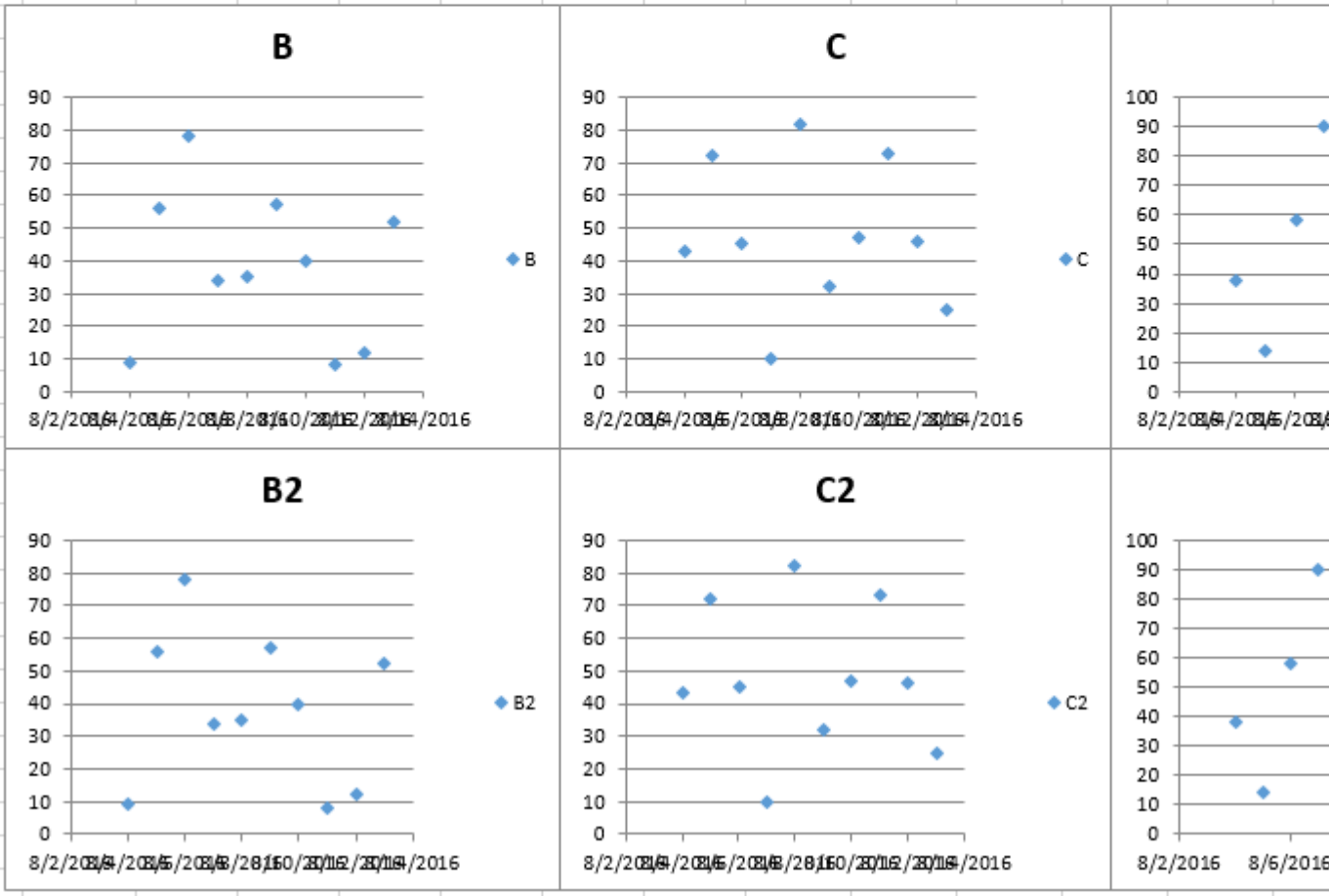
Эти снимки показывают исходную случайную компоновку диаграмм и результирующую сетку от запуска кода выше.

До


A
B
C
D
E
F
G
H
I
J
K
L
M
1
2
A
B
C
D
B2
C2
D2
3
8/4/2016
9
43
38
9
43
38
4
8/5/2016
56
72
14
56
72
14
5
8/6/2016
78
45
58
78
45
58
6
8/7/2016
34
10
90
34
10
7
82
8
32
9
47
10
73
11
46
12
25
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
B
C
D
D2
B2
C2
90
80
70
60
50
40
30
20
10
0
100
8/2/2016
8/6/2016
8/10/
B2
C2

После

	A	B	C	D	E	F	G	H
1								
2		A	B	C	D	B2	C2	D2
3		8/4/2016	9	43	38	9	43	38
4		8/5/2016	56	72	14	56	72	14
5		8/6/2016	78	45	58	78	45	58
6		8/7/2016	34	10	90	34	10	90
7		8/8/2016	35	82	74	35	82	74
8		8/9/2016	57	32	40	57	32	40
9		8/10/2016	40	47	2	40	47	2
10		8/11/2016	8	73	93	8	73	93
11		8/12/2016	12	46	28	12	46	28
12		8/13/2016	52	25	37	52	25	37



Прочитайте Графики и диаграммы онлайн: https://riptutorial.com/ru/excel-vba/topic/4968/графики-и-диаграммы

глава 8: Диапазоны и ячейки

Синтаксис

- **Set** - оператор, используемый для установки ссылки на объект, например, на диапазон
- **Для каждого** - оператор, используемый для прокрутки каждого элемента в коллекции

замечания

Обратите внимание, что имена переменных `r` , `cell` и другие могут быть названы, как вам нравится, но должны быть названы соответствующим образом, чтобы код был более понятным для вас и других.

Examples

Создание диапазона

Диапазон нельзя создать или заполнить так же, как строка:

```
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
```

Считается лучшей практикой, чтобы квалифицировать ваши ссылки , поэтому в дальнейшем мы будем использовать один и тот же подход.
Подробнее о создании объектных переменных (например, Range) в MSDN . Подробнее о Set Statement на MSDN .

Существуют разные способы создания одного и того же диапазона:

```
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
```

Обратите внимание на пример, что ячейки (2, 1) эквивалентны диапазону («A2»). Это происходит потому, что Cells возвращает объект Range.
Некоторые источники: Chip Pearson-Cells Within Ranges ; Объект диапазона MSDN ; John Walkenback - ссылка на диапазоны в коде VBA .

Также обратите внимание, что в любом случае, когда число используется в объявлении диапазона, а сам номер находится вне кавычек, например Range («A» & 2), вы можете поменять это число на переменную, содержащую целое число / долго. Например:

```
Sub RangeIteration()
    Dim wb As Workbook, ws As Worksheet
    Dim r As Range

    Set wb = ThisWorkbook
    Set ws = wb.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
```

Если вы используете двойные циклы, ячейки лучше:

```
Sub RangeIteration2()
    Dim wb As Workbook, ws As Worksheet
    Dim r As Range

    Set wb = ThisWorkbook
    Set ws = wb.Worksheets(1)

    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
```

Способы обращения к одной ячейке

Самый простой способ ссылаться на одну ячейку на текущем листе Excel - это просто вставить форму A1 в ссылку в квадратных скобках:

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

Обратите внимание, что квадратные скобки - это просто удобный синтаксический сахар для метода `Evaluate` объекта `Application` , так что технически это идентично следующему коду:

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

Вы также можете вызвать метод `Cells` который принимает строку и столбец и возвращает ссылку на ячейку.

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

Помните, что всякий раз, когда вы передаете строку и столбец в Excel из VBA, строка всегда первая, за ней следует столбец, что запутывает, потому что это противоположно общей нотации `A1` где сначала отображается столбец.

В обоих этих примерах мы не указали рабочий лист, поэтому Excel будет использовать активный лист (лист, который находится впереди в пользовательском интерфейсе). Вы можете указать активный лист явно:

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

Или вы можете указать имя определенного листа:

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

Существует множество методов, которые можно использовать для перехода от одного диапазона к другому. Например, метод `Rows` может использоваться для доступа к отдельным строкам любого диапазона, и метод `Cells` может использоваться для доступа к отдельным ячейкам строки или столбца, поэтому следующий код относится к ячейке C1:

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

Сохранение ссылки на ячейку переменной

Чтобы сохранить ссылку на ячейку в переменной, вы должны использовать синтаксис `Set` , например:

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

ПОТОМ...

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

Почему требуется ключевое слово `Set` ? `Set` указывает Visual Basic, что значение в правой части `=` означает объект.

Смещение недвижимости

- **Смещение (строки, столбцы)** - оператор, используемый для статической ссылки на другую точку из текущей ячейки. Часто используется в циклах. Следует понимать, что положительные числа в разделе строк перемещаются вправо, поскольку негативы перемещаются влево. С положительными позициями столбцов вниз и негативы двигаются вверх.

т.е.

```
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
```

Этот код выбирает B2, помещает туда новую строку, затем перемещает эту строку обратно в A1 после очистки B2.

Как перемещать диапазоны (по горизонтали по вертикали и наоборот)

```
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
```

Примечание. Copy / PasteSpecial также имеет параметр «Вставить транспонирование», который также обновляет формулы транспонированных ячеек.

Прочитайте Диапазоны и ячейки онлайн: https://riptutorial.com/ru/excel-vba/topic/1503/диапазоны-и-ячейки

глава 9: Именованные диапазоны

Вступление

Тема должна включать информацию, конкретно связанную с именованными диапазонами в Excel, включая методы создания, изменения, удаления и доступа к определенным именованным диапазонам.

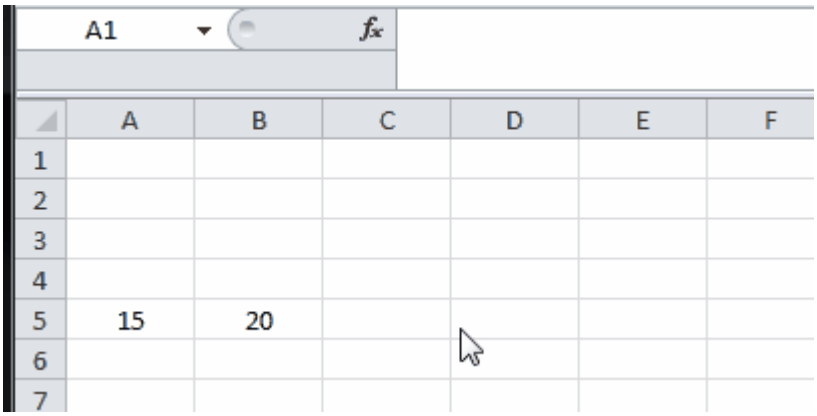
Examples

Определить именованный диапазон

Использование именованных диапазонов позволяет описать значение содержимого ячейки (я) и использовать это определенное имя вместо фактического адреса ячейки.

Например, формулу `=A5*B5` можно заменить на `=Width*Height` чтобы упростить чтение и понимание формулы.

Чтобы определить новый именованный диапазон, выберите ячейку или ячейки для имени, а затем введите новое имя в поле «Имя» рядом с панелью формул.



> Примечание. Именованные диапазоны по умолчанию относятся к глобальной области, что означает, что к ним можно получить доступ из любой точки книги. Старые версии Excel позволяют дублировать имена, поэтому необходимо избегать дублирования имен глобальной области, иначе результаты будут непредсказуемыми. Используйте вкладку «Диспетчер имен» на вкладке «Формулы», чтобы изменить область действия.

Использование именных диапазонов в VBA

Создайте новый именованный диапазон под названием «MyRange», назначенный ячейке `A1`

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

Удалить определенный именованный диапазон по имени

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

Доступ к именованному диапазону по имени

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

Доступ к названию диапазона с ярлыком

Как и любой другой диапазон , именованные диапазоны могут быть доступны напрямую с помощью ярлыка, который не требует создания объекта `Range` . Три строки из выдержки из вышеприведенного кода могут быть заменены одной строкой:

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

> Примечание. Свойством по умолчанию для диапазона является его значение, поэтому `[MyRange]` совпадает с `[MyRange].Value`

Вы также можете вызвать методы в диапазоне. Следующий выбирает `MyRange` :

```
[MyRange].Select
```

> Примечание. Одно предостережение состоит в том, что нотация ярлыка не работает со словами, которые используются в другом месте библиотеки VBA. Например, диапазон с именем `Width` не будет доступен как `[Width]` но будет работать, как ожидалось, при доступе через `ThisWorkbook.Worksheets("Sheet1").Range("Width")`

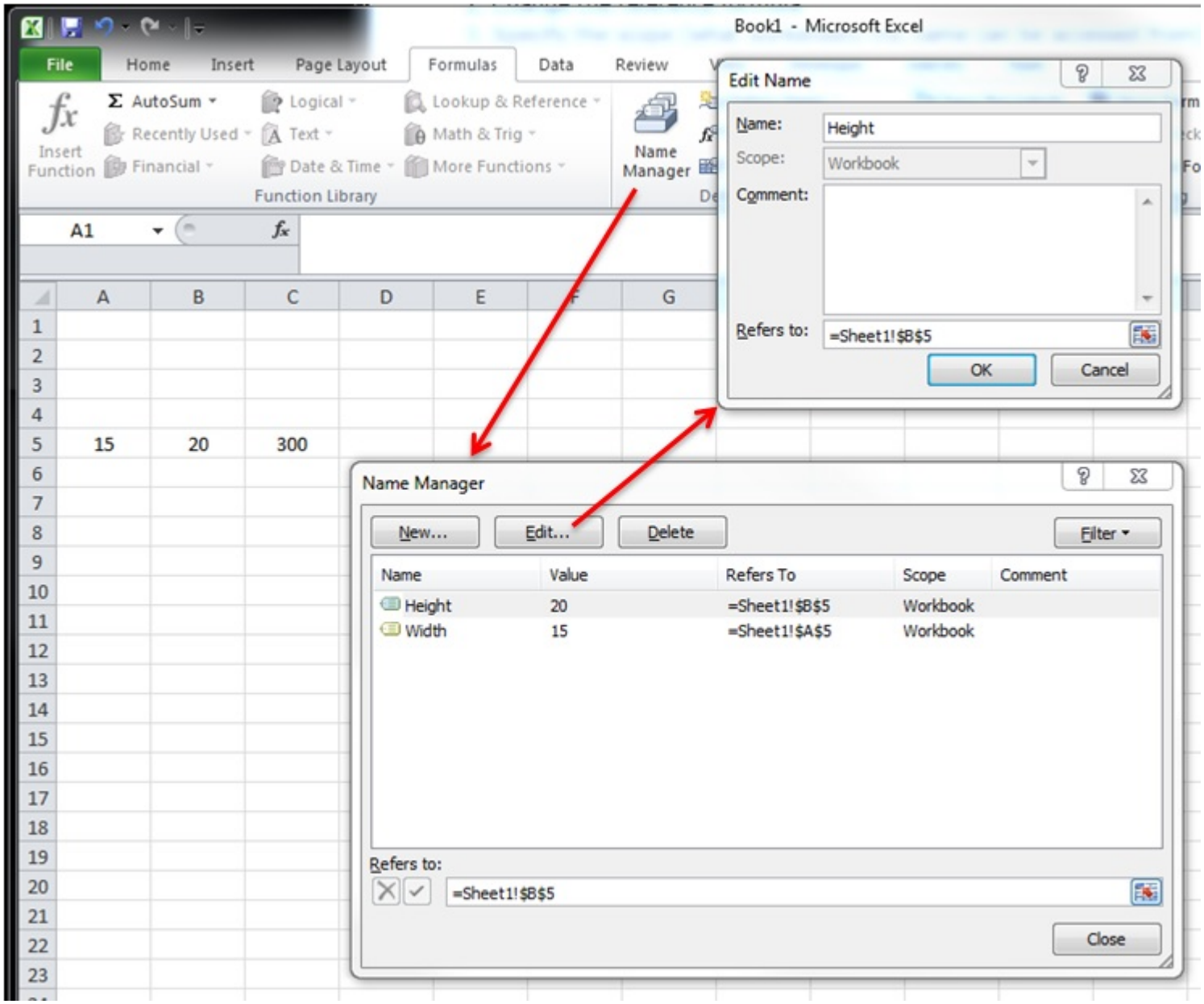
Управление именованным диапазоном (диапазонами) с помощью диспетчера имен

Вкладка «Формулы»> «Определенная группа имен»> «Диспетчер имен»

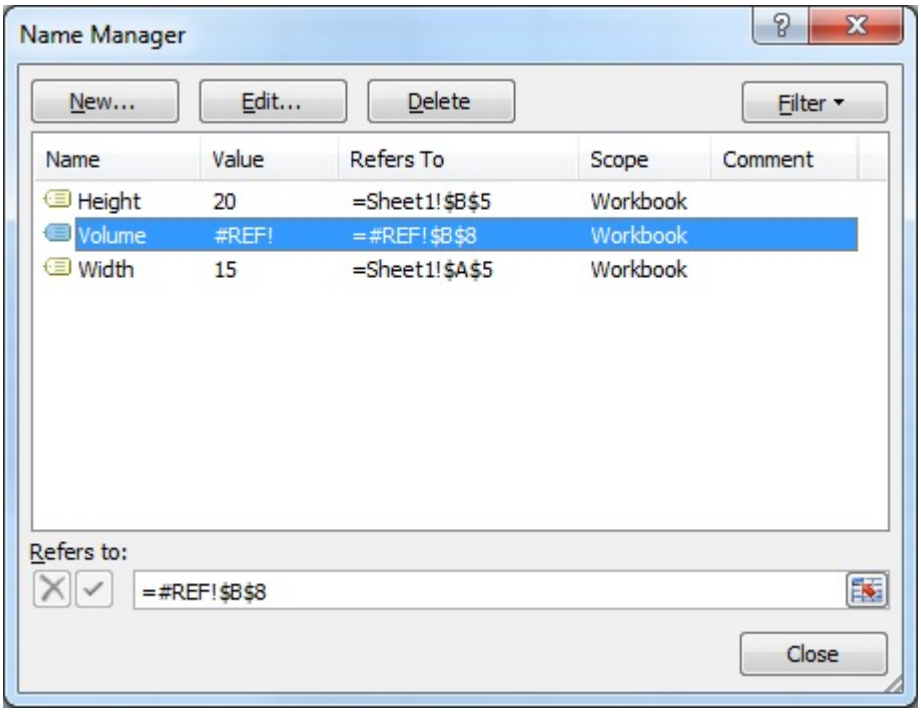
Именованный менеджер позволяет:

1. Создайте или измените имя
2. Создать или изменить ссылку на ячейку
3. Создать или изменить область действия

4. Удалить существующий именованный диапазон

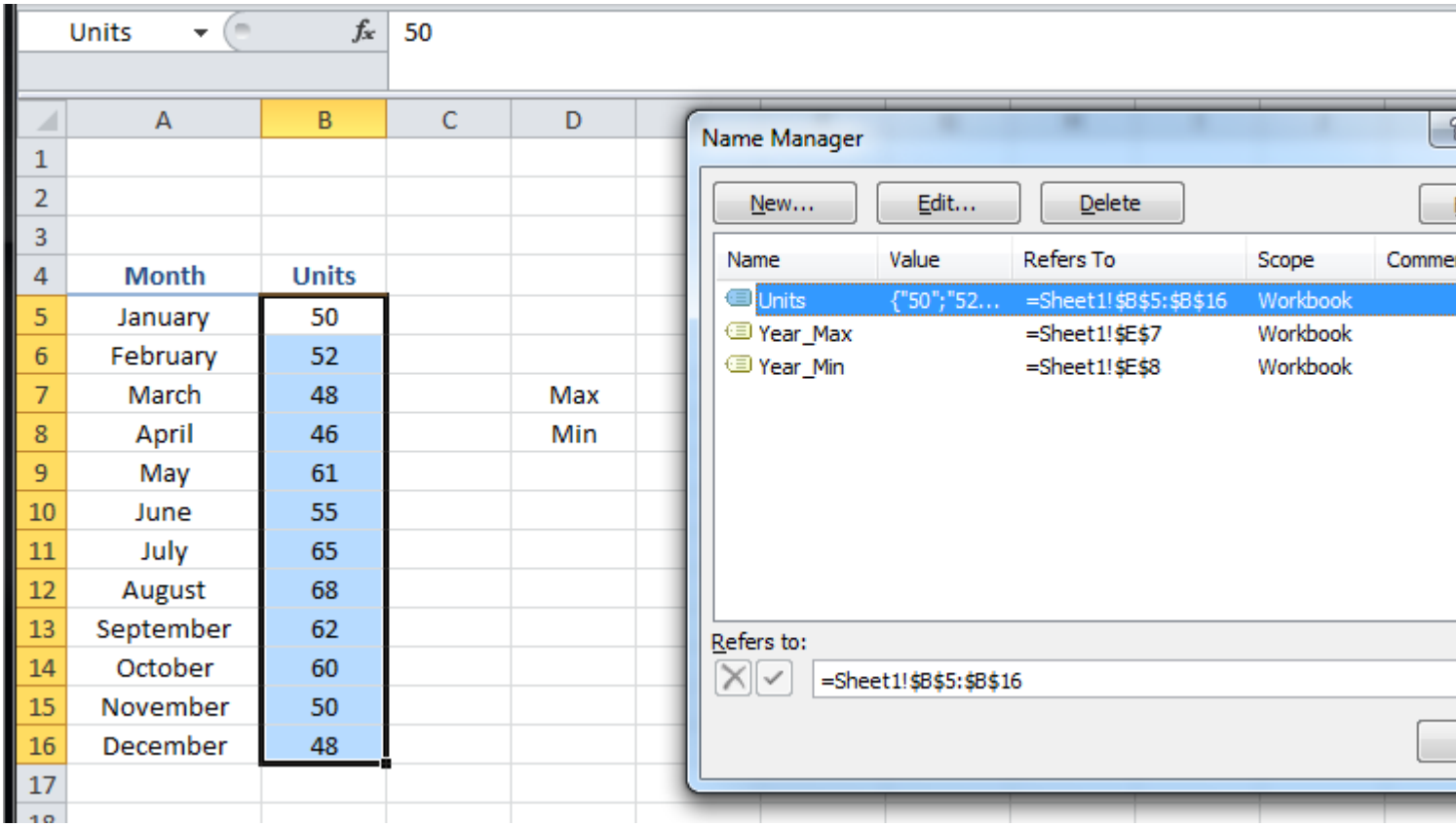


Named Manager предоставляет полезный быстрый поиск неработающих ссылок.



Именованные массивы диапазонов

Примерный лист



Код

```
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
```

Результат

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			

Прочитайте Именованные диапазоны онлайн: https://riptutorial.com/ru/excel-vba/topic/8360/именованные-диапазоны

глава 10: Интеграция PowerPoint через VBA

замечания

В этом разделе демонстрируется множество способов взаимодействия с PowerPoint через VBA. От показа данных на слайдах до создания диаграмм PowerPoint является очень мощным инструментом при использовании в сочетании с Excel. Таким образом, в этом разделе делается попытка продемонстрировать различные способы использования VBA для автоматизации этого взаимодействия.

Examples

Основы: запуск PowerPoint из VBA

Хотя есть много параметров, которые можно изменить, и варианты, которые могут быть добавлены в зависимости от желаемой функциональности, в этом примере излагается основная основа для запуска PowerPoint.

> **Примечание.** Этот код требует, чтобы ссылка PowerPoint была добавлена в активный проект VBA. См References запись документации , чтобы узнать , как включить ссылку.

Во-первых, определите переменные для объектов Application, Presentation и Slide. Хотя это можно сделать с поздним связыванием, всегда лучше использовать раннее связывание, когда это применимо.

```
Dim PPApp As PowerPoint.Application
Dim PPPres As PowerPoint.Presentation
Dim PPSlide As PowerPoint.Slide
```

Затем откройте или создайте новый экземпляр приложения PowerPoint. Здесь вызов `On Error Resume Next` используется, чтобы избежать ошибки, `GetObject` если PowerPoint еще не открыта. Более подробное объяснение см. В примере обработки ошибок в разделе «Лучшая практика».

```
'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
```

После запуска приложения создается новая презентация и впоследствии содержащий слайд.

```
'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
```

По завершении этого кода откроется новое окно PowerPoint с пустым слайдом. При использовании переменных объекта могут быть добавлены формы, текст, графика и диапазоны excel по желанию

Прочитайте Интеграция PowerPoint через VBA онлайн: https://riptutorial.com/ru/excel-vba/topic/2327/интеграция-powerpoint-через-vba

глава 11: Использовать объект Worksheet, а не объект Sheet

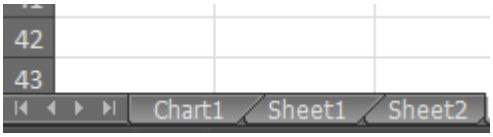
Вступление

Множество пользователей VBA рассматривают синонимы Worksheets and Sheets. Они не.

Объект «Листы» состоит из листов и диаграмм. Таким образом, если у нас есть диаграммы в нашей книге Excel, мы должны быть осторожны, а не использовать `Sheets` и `Worksheets` качестве синонимов.

Examples

Распечатайте имя первого объекта




```
Option Explicit

Sub CheckWorksheetsDiagram()

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

End Sub
```

Результат:

```
Sheet1
Chart1
Chart1
```

Прочитайте Использовать объект Worksheet, а не объект Sheet онлайн: https://riptutorial.com/ru/excel-vba/topic/9996/использовать-объект-worksheet--а-не-объект-sheet

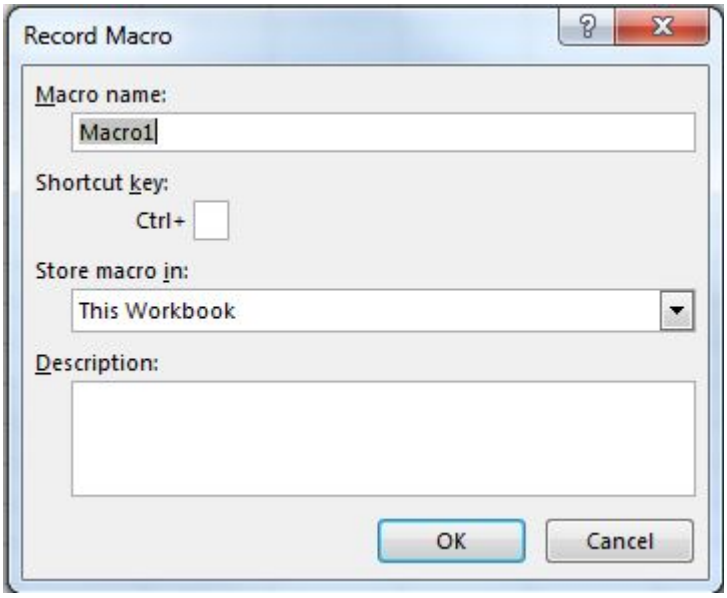
глава 12: Как записать макрос

Examples

Как записать макрос

Самый простой способ записи макроса - кнопка в левом нижнем углу Excel выглядит следующим образом: 

Когда вы нажмете на это, вы получите всплывающее окно с просьбой назвать макрос и решить, хотите ли вы иметь комбинацию клавиш. Кроме того, спрашивает, где хранить макрос и описание. Вы можете выбрать любое имя, которое вам нужно, не допускается использование пробелов.



Если вы хотите, чтобы ярлык, назначенный вашему макросу для быстрого использования, выберите письмо, которое вы запомните, чтобы вы могли быстро и легко использовать макрос снова и снова.

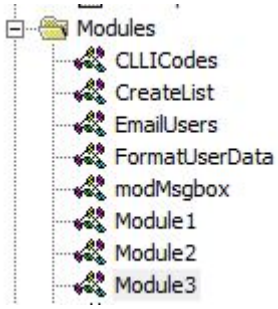
Вы можете сохранить макрос в «Эта книга», «Новая рабочая книга» или «Личная книга макросов». Если вы хотите, чтобы макрос, который вы собираетесь записывать, был доступен только в текущей книге, выберите «This Workbook». Если вы хотите, чтобы он был сохранен в совершенно новой книге, выберите «Новая рабочая книга». И если вы хотите, чтобы макрос был доступен для любой открытой книги, выберите «Personal Macro Workbook».

После заполнения этого всплывающего окна нажмите «OK».

Затем выполните любые действия, которые вы хотите повторить с помощью макроса. По завершении нажмите ту же кнопку, чтобы остановить запись. Теперь он выглядит так:

Теперь вы можете перейти на вкладку «Разработчик» и открыть Visual Basic. (или используйте Alt + F11)

Теперь у вас будет новый модуль в папке «Модули».

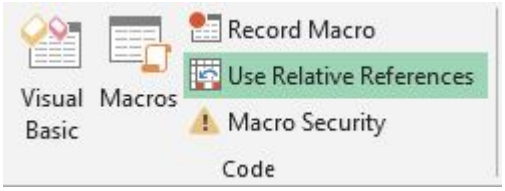


Самый новый модуль будет содержать только что записанный макрос. Дважды щелкните по нему, чтобы поднять его.

Я сделал простую копию и вставку:

```
Sub Macro1()
'
' Macro1 Macro
'

'
    Selection.Copy
    Range("A12").Select
    ActiveSheet.Paste
End Sub
```

Если вы не хотите, чтобы он всегда вставлялся в «A12», вы можете использовать Relative References, установив флажок «Использовать относительные ссылки» на вкладке «Разработчик»:



Следуя тем же шагам, что и раньше, теперь включите Макро:

```
Sub Macro2()
'
' Macro2 Macro
'

'
    Selection.Copy
```

```
    ActiveCell.Offset(11, 0).Range("A1").Select
    ActiveSheet.Paste
End Sub
```

Все еще копируя значение из «A1» в ячейку 11 строк вниз, но теперь вы можете выполнить один и тот же макрос с любой начальной ячейкой, а значение из этой ячейки будет скопировано в ячейку 11 строк вниз.

Прочитайте Как записать макрос онлайн: https://riptutorial.com/ru/excel-vba/topic/8204/как-записать-макрос

глава 13: Лучшие практики VBA

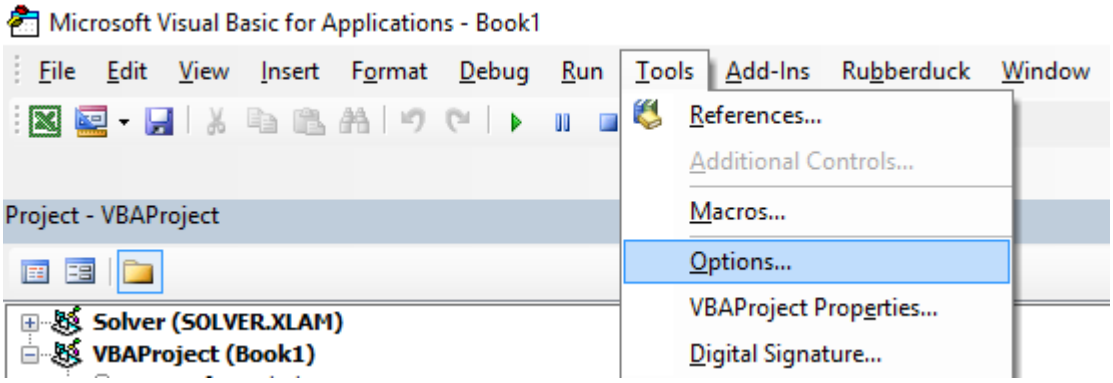
замечания

Мы все знаем их, но эти практики гораздо менее очевидны для тех, кто начинает программировать в VBA.

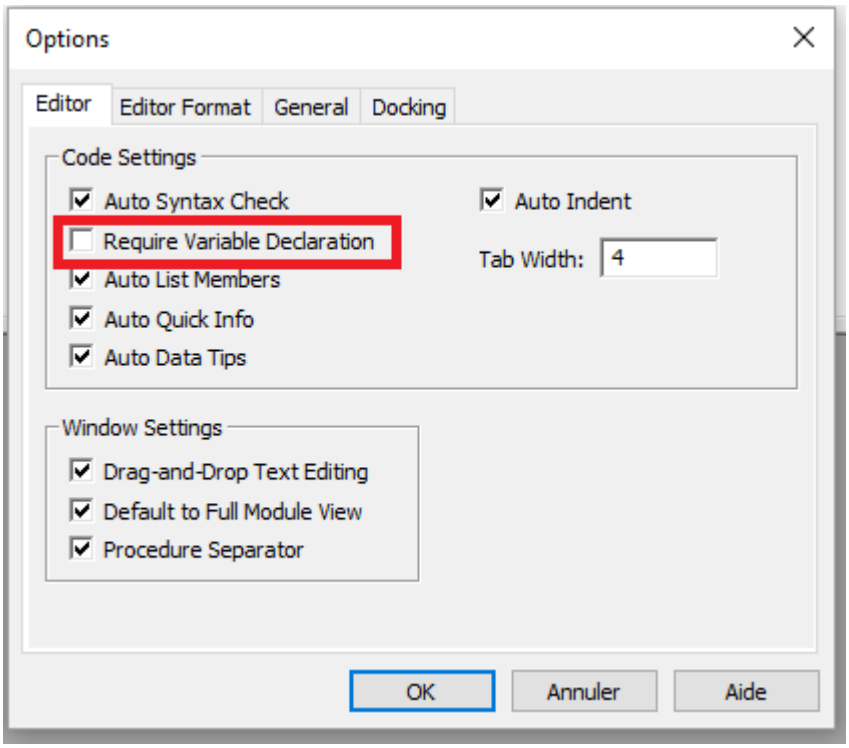
Examples

ВСЕГДА Используйте «Option Explicit»

В окне редактора VBA в меню «Сервис» выберите «Параметры»:



Затем на вкладке «Редактор» убедитесь, что «Требовать переменную декларацию» отмечен:



При выборе этой опции автоматически добавляется `Option Explicit` в верхней части каждого модуля VBA.

> **Небольшое примечание:** это верно для модулей, модулей классов и т. Д., Которые пока не были открыты. Поэтому, если вы уже рассмотрели, например, код `Sheet1` перед тем как активировать опцию «Требовать объявление переменной», `Option Explicit` не будет добавлен!

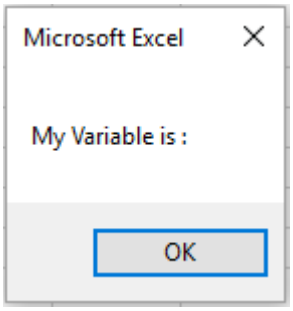
`Option Explicit` требует, чтобы каждая переменная была определена перед использованием, например, с помощью оператора `Dim` . Без `Option Explicit` , любое непризнанное слово будет приниматься компилятором VBA как новая переменная типа `Variant` , вызывая чрезвычайно сложные ошибки, связанные с типографскими ошибками. Если `Option Explicit` включена, любые непризнанные слова вызовут ошибку компиляции, указывая на строку нарушения.

Пример :

Если вы запустите следующий код:

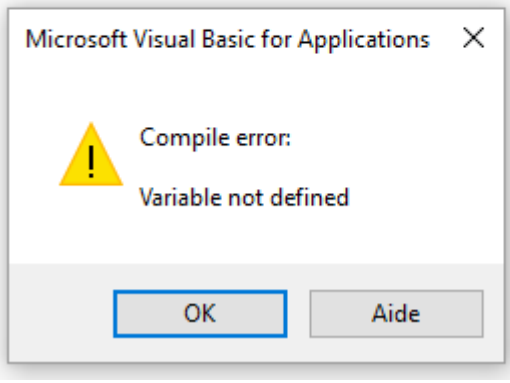
```
Sub Test()
  my_variable = 12
  MsgBox "My Variable is : " & myvariable
End Sub
```

Появится следующее сообщение:



Вы сделали ошибку, написав `myvariable` вместо `my_variable` , затем в поле сообщения отображается пустая переменная. Если вы используете `Option Explicit` , эта ошибка невозможна, потому что вы получите сообщение об ошибке компиляции, указывающее на проблему.

```
Option Explicit

Sub Test()
  my_variable = 12
  MsgBox "My Variable is : " & myvariable
End Sub
```

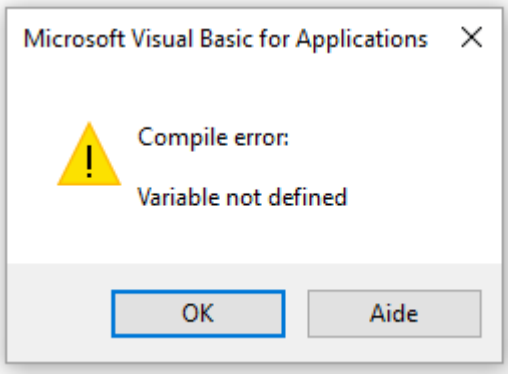


Теперь, если вы добавите правильное объявление:

```
Sub Test()
  Dim my_variable As Integer
  my_variable = 12
  MsgBox "My Variable is : " & myvariable
End Sub
```

Вы получите сообщение об ошибке с указанием ошибки с `myvariable` :

```
Option Explicit

Sub Test()
  Dim my_variable As Integer
  my_variable = 12
  MsgBox "My Variable is : " & myvariable
End Sub
```



Примечание по Option Explicit и Array (объявление динамического массива):

Вы можете использовать оператор ReDim, чтобы объявить массив неявно в рамках процедуры.

- Будьте осторожны, чтобы не пропустить имя массива при использовании

оператора ReDim

- Даже если оператор Option Explicit включен в модуль, будет создан новый массив

```
Dim arr() as Long

ReDim ar() 'creates new array "ar" - "ReDim ar()" acts like "Dim ar()"
```

Работа с массивами, а не с диапазонами

Офисный блог - лучшие методы кодирования производительности Excel VBA

Часто достигается наилучшая производительность, позволяя максимально избегать использования `Range` . В этом примере мы читаем весь объект `Range` в массив, каждый квадрат в массиве, а затем возвращаем массив обратно в `Range` . Это позволяет получить доступ к `Range` только дважды, тогда как цикл будет обращаться к нему 20 раз для чтения / записи.

```
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 ' writes all the results back to the range at
once

End Sub
```

Дополнительные советы и информация с приуроченными примерами можно найти в статье Чарльза Уильямса «Эффективные VBA UDF» (часть 1) и других статьях в этой серии .

Используйте константы VB, если они доступны

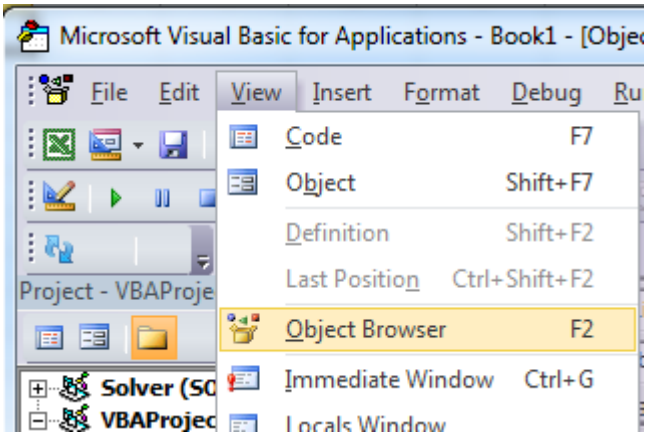
```
If MsgBox("Click OK") = vbOK Then
```

могут быть использованы вместо

```
If MsgBox("Click OK") = 1 Then
```

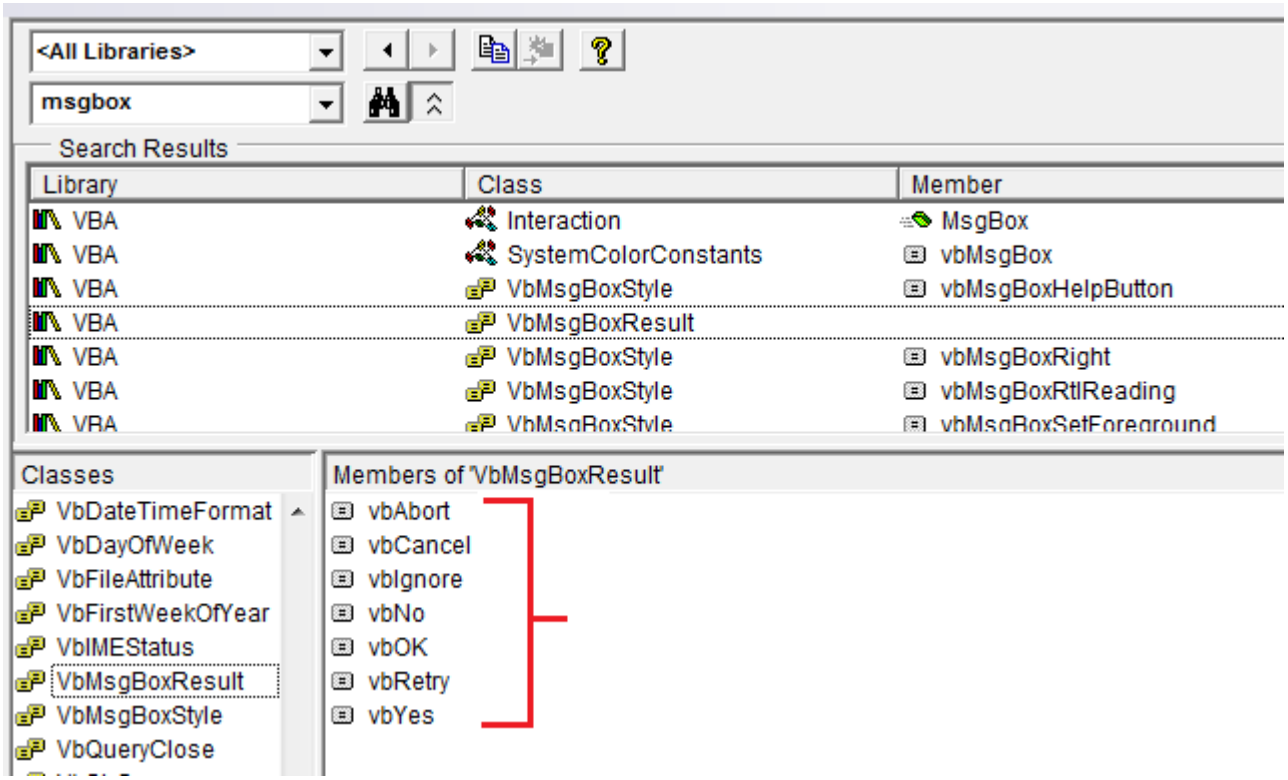
чтобы улучшить удобочитаемость.

Используйте *Object Browser*, чтобы найти доступные константы VB. *Просмотр → Обозреватель объектов* или `F2` из редактора VB.



Введите класс для поиска

Просмотреть участников



Использовать описательные имена переменных

Описательные имена и структура в коде помогут сделать комментарии ненужными

```
Dim ductWidth  As Double
Dim ductHeight As Double
Dim ductArea   As Double
```

```
ductArea = ductWidth * ductHeight
```

лучше, чем

```
Dim a, w, h

a = w * h
```

Это особенно полезно при копировании данных из одного места в другое, будь то ячейка, диапазон, рабочий лист или рабочая книга. Помогите себе, используя такие имена:

```
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
```

Если вы объявляете несколько переменных в одной строке, обязательно указывайте тип для *каждой* переменной:

```
Dim ductWidth As Double, ductHeight As Double, ductArea As Double
```

Следующие будут объявлять только последнюю переменную, а первая останется `Variant` :

```
Dim ductWidth, ductHeight, ductArea As Double
```

Обработка ошибок

Хорошая обработка ошибок не позволяет конечным пользователям видеть ошибки времени выполнения VBA и помогает разработчику легко диагностировать и исправлять ошибки.

Существует три основных метода обработки ошибок в VBA, два из которых следует избегать для распределенных программ, если это специально не требуется в коде.

```
On Error GoTo 0 'Avoid using
```

или же

```
On Error Resume Next 'Avoid using
```

Предпочитают использовать:

```
On Error GoTo <line> 'Prefer using
```

По ошибке GoTo 0

Если в вашем коде не установлена ошибка, `On Error GoTo 0` является обработчиком ошибок по умолчанию. В этом режиме любые ошибки времени выполнения запускают типичное сообщение об ошибке VBA, позволяющее либо закончить код, либо ввести режим `debug` , идентифицируя источник. При написании кода этот метод является самым простым и полезным, но его всегда следует избегать для кода, который распространяется среди конечных пользователей, поскольку этот метод очень непригляден и затруднен для понимания конечными пользователями.

Вкл.

`On Error Resume Next` VBA будет игнорировать любые ошибки, возникающие во время выполнения для всех строк, следующих за вызовом ошибки, до тех пор, пока обработчик ошибок не будет изменен. В очень конкретных случаях эта строка может быть полезна, но ее следует избегать за пределами этих случаев. Например, при запуске отдельной программы из макроса Excel вызов `On Error Resume Next` может быть полезен, если вы не уверены, открыта ли программа или нет:

```
'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
```

Если бы мы не использовали `GetObject` вызов `On Error Resume Next` и приложение Powerpoint еще не было открыто, метод `GetObject` бы ошибку. Таким образом, для устранения двух экземпляров приложения необходимо было `On Error Resume Next` .

Примечание. Также рекомендуется *сразу же* сбросить обработчик ошибок, как только вам больше не понадобится `On Error Resume Next` вызова»

Вкл. Ошибка GoTo <строка>

Этот метод обработки ошибок рекомендуется для всего кода, который распространяется среди других пользователей. Это позволяет программисту точно контролировать, как VBA обрабатывает ошибку, отправив код в указанную строку. Тег можно заполнить любой строкой (включая числовые строки) и отправить код в соответствующую строку, за которой следует двоеточие. Несколько блоков обработки ошибок можно использовать, выполняя различные вызовы `On Error GoTo <line>` . Нижеприведенная подпрограмма демонстрирует синтаксис вызова `On Error GoTo <line>` .

Примечание. Важно, чтобы строка `Exit Sub` была помещена над первым обработчиком ошибок и перед каждым последующим обработчиком ошибок, чтобы предотвратить естественный переход кода в блок *без* вызываемой ошибки. Таким образом, лучше всего использовать функцию и читаемость для размещения обработчиков ошибок в конце блока кода.

```
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.Desctription
        Application.ScreenUpdating = True
        Application.EnableEvents = True
        Exit Sub
    End If
    Exit Sub

End Sub
```

Если вы выйдете из своего метода с кодом обработки ошибок, убедитесь, что вы очистили:

- Отменить все, что частично завершено
- Закрыть файлы
- Сбросить обновление экрана
- Сбросить режим расчета

- Сбросить события
- Сбросить указатель мыши
- Вызов метода выгрузки для экземпляров объектов, которые сохраняются после `End Sub`
- Сброс строки состояния

Документируйте свою работу

Хорошей практикой является документирование вашей работы для последующего использования, особенно если вы кодируете динамическую рабочую нагрузку. Хорошие комментарии должны объяснять, почему код что-то делает, а не то, что делает код.

```
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
```

Если ваш код настолько неясен, что он требует комментариев, чтобы объяснить, что он делает, подумайте о том, чтобы переписать его, чтобы быть более понятным, а не объяснять его с помощью комментариев. Например, вместо:

```
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
```

Уточнить код, который будет проще следовать, например:

```
Sub CopySalesNumbers
    Dim IncludeWeekends as Boolean
    Dim DaysinWeek as Integer

    If IncludeWeekends Then
        DaysinWeek = 7
    Else
        DaysinWeek = 5
    End If
    Range("A1:A" & DaysinWeek).Copy
    Range("B1").PasteSpecial
End Sub
```

Отключение свойств во время выполнения макроса

Лучше всего на любом языке программирования **избегать преждевременной**

оптимизации. Однако, если тестирование показывает, что ваш код работает слишком медленно, вы можете получить некоторую скорость, отключив некоторые свойства приложения во время его запуска. Добавьте этот код в стандартный модуль:

```
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
```

Дополнительная информация о Office Blog - Excel VBA Performance Code Best Practices

И просто назовите его в начале и в конце макросов:

```
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
```

Хотя они могут в значительной степени рассматриваться как «усовершенствования» для обычных `Public Sub` процедур `Public Sub` , отключение обработки событий с помощью `Application.EnableEvents = False` должно считаться обязательным для макросов частного события `Worksheet_Change` и `Workbook_SheetChange` которые изменяют значения на одном или нескольких листах. Невозможность отключить триггеры событий заставит макрос события рекурсивно запускаться поверх себя, когда значение изменится и может привести к «замороженной» книге. Не забудьте снова включить события, прежде чем покинуть макрос события, возможно, с помощью обработчика ошибок «безопасного выхода».

```
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
```

Одно предостережение. Хотя отключение этих настроек улучшит время выполнения, они могут затруднить отладку приложения. Если код *не* работает надлежащим образом , закомментируйте `SpeedUp True` вызов , пока вы не выяснить проблему.

Это особенно важно, если вы пишете ячейки на листе, а затем читаете в вычисленных результатах из функций листа, поскольку `xlCalculationManual` позволяет `xlCalculationManual` книгу. Чтобы обойти это без отключения `SpeedUp` , вы можете включить `Application.Calculate` для выполнения расчета в определенных точках.

ПРИМЕЧАНИЕ. Поскольку это свойства самого `Application` , вам нужно убедиться, что они снова включены до того, как вы закончите макрос. Это делает особенно важным использование обработчиков ошибок и избежание множественных точек выхода (например, `End` или `Unload Me`).

С обработкой ошибок:

```
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
```

Избегайте использования ActiveCell или ActiveSheet в Excel

Использование `ActiveCell` или `ActiveSheet` может быть источником ошибок , если (по любой причине) код выполняется в неправильном месте.

```
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"
```

- Использование `Active*` может создавать проблемы в длинных макросах, если ваш пользователь скучает и нажимает на другой рабочий лист или открывает другую книгу.
- Это может создать проблемы, если ваш код открывается или создает другую книгу.
- Это может создать проблемы, если ваш код использует `Sheets("MyOtherSheet").Select` и вы забыли, какой лист вы были, прежде чем начинать читать или писать на него.

Никогда не допускайте рабочий лист

Даже когда вся ваша работа направлена на один рабочий лист, все же очень хорошая практика явно указывать рабочий лист в вашем коде. Эта привычка значительно облегчает расширение вашего кода позже или поднять части (или все) `Sub` или `Function` которые будут повторно использоваться где-то еще. Многие разработчики устанавливают привычку (re) использовать одно и то же имя локальной переменной для рабочего листа в своем коде, что делает повторное использование этого кода еще более простым.

Например, следующий код неоднозначен, но работает! - пока разработчик не активирует или не переключается на другой рабочий лист:

```
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
```

Если `Sheet1` активен, ячейка A1 на Листе 1 будет заполнена текущей датой и временем. Но если пользователь по какой-либо причине меняет листы, тогда код обновит все текущие рабочие листы. Рабочий лист адресата неоднозначен.

Лучшая практика - всегда определять, какой лист, на который ссылается ваш код:

```
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
```

Вышеприведенный код ясен при определении как рабочей книги, так и рабочего листа. Хотя это может показаться излишним, создание хорошей привычки в отношении целевых ссылок избавит вас от будущих проблем.

Избегайте использования SELECT или ACTIVATE

Очень редко вы когда-либо захотите использовать `Select` или `Activate` в своем коде, но некоторые методы Excel требуют, чтобы рабочий лист или рабочая книга были активированы до того, как они будут работать должным образом.

Если вы только начинаете изучать VBA, вам часто предлагается записать ваши действия с помощью макросъемщика, а затем взглянуть на код. Например, я записал действия, предпринятые для ввода значения в ячейке D3 на Sheet2, и макрокоманда выглядит следующим образом:

```
Option Explicit
Sub Macro1()
'
' Macro1 Macro
'

'
    Sheets("Sheet2").Select
    Range("D3").Select
    ActiveCell.FormulaR1C1 = "3.1415"   '(see **note below)
    Range("D4").Select
End Sub
```

Помните, однако, макрорекордер создает строку кода для КАЖДОГО из ваших (пользовательских) действий. Это включает в себя щелчок на вкладке рабочего листа, чтобы выбрать Sheet2 (`Sheets("Sheet2").Select`), щелкнув по ячейке D3 перед вводом значения (`Range("D3").Select`) и с помощью клавиши Enter (которая эффективно « выбрав «ячейку ниже текущей выбранной ячейки: `Range("D4").Select`).

Существует несколько проблем с использованием. `.Select` здесь:

- **Рабочий лист не всегда указывается.** Это происходит, если вы не меняете рабочие листы во время записи и означает, что код даст разные результаты для разных активных рабочих листов.
- **`.Select()` работает медленно.** Даже если для параметра `Application.ScreenUpdating` установлено значение `False` , это необработанная операция, которая должна быть обработана.
- **`.Select()` неуправляем.** Если `Application.ScreenUpdating` остается равным `True` , Excel будет фактически выбирать ячейки, рабочий лист, форму ... независимо от того, с чем вы работаете. Это стрессово для глаз и действительно неприятно смотреть.
- **`.Select()` вызовет прослушиватели.** Это уже немного продвинуто, но если не работать, будут запускаться такие функции, как `Worksheet_SelectionChange()` .

Когда вы кодируете в VBA, все действия «набрав» (т. `Select` Команды `Select`) больше не нужны. Ваш код может быть сведен к одному оператору, чтобы поместить значение в ячейку:

```
'--- 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
```

(Пример BETTER выше показывает использование промежуточных переменных для разделения разных частей ссылки на ячейку. Пример GOOD всегда будет работать очень хорошо, но может быть очень громоздким в гораздо более длинных модулях кода и более сложным для отладки, если одна из ссылок неверна.)

** ПРИМЕЧАНИЕ. Макросъемщик делает много предположений о типе данных, которые вы вводите, в этом случае вводите строковое значение в качестве формулы для создания значения. Ваш код не должен делать этого и может просто назначить числовое значение непосредственно ячейке, как показано выше.

** Примечание 2: рекомендуемая практика , чтобы установить локальную переменную рабочую книгу `ThisWorkbook` вместо `ActiveWorkbook` (если явно не нужно). Причина заключается в том, что ваш макрос обычно должен / использовать ресурсы в любой книге, из которой возникает код VBA, и НЕ будет выглядеть за пределами этой книги - опять же, если вы явно не назовете свой код работать с другой книгой. Когда вы открываете несколько книг в Excel, `ActiveWorkbook` - это та, *которая может отличаться от рабочей книги, просматриваемой в редакторе VBA* . Итак, вы думаете, что работаете в одной книге, когда вы действительно ссылаетесь на другую. `ThisWorkbook` относится к книге, содержащей исполняемый код.

Всегда определяйте и устанавливайте ссылки на все книги и таблицы

При работе с несколькими открытыми рабочими книгами, каждая из которых может иметь несколько листов, наиболее безопасно определить и установить ссылку на все книги и таблицы.

Не полагайтесь на `ActiveWorkbook` или `ActiveSheet` поскольку они могут быть изменены пользователем.

В следующем примере кода показано , как скопировать диапазон от листа *«Raw_Data»* в

книге *«Data.xlsx»* на листе *«Refined_Data»* в книге *«Results.xlsx».*

Эта процедура также демонстрирует, как копировать и вставлять, не используя метод `Select` .

```
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
```

Объект WorksheetFunction выполняется быстрее, чем эквивалент UDF

VBA компилируется во время выполнения, что оказывает огромное негативное влияние на его производительность, все встроенное будет быстрее, попробуйте использовать их.

В качестве примера я сравниваю функции SUM и COUNTIF, но вы можете использовать, если для чего-нибудь, что вы можете решить с помощью WorkSheetFunctions.

Первой попыткой для них было бы перебрать диапазон и обработать его ячейкой по ячейке (используя диапазон):

```
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
```

Одним из улучшений может быть сохранение значений диапазона в массиве и процесс, который:

```
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
```

Но вместо написания любого цикла вы можете использовать `Application.Worksheetfunction` что очень удобно для выполнения простых формул:

```
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
```

Или, для более сложных вычислений вы можете даже использовать `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
```

И, наконец, работая над Subs 25 000 раз каждый, вот среднее (5 тестов) время в миллисекундах (конечно, это будет отличаться на каждом ПК, но по сравнению друг с другом они будут вести себя аналогичным образом):

1. UseWorksheetFunction: 2156 мс
2. UseArray: 2219 мс (+ 3%)
3. UseEvaluate: 4693 мс (+ 118%)
4. UseRange: 6530 мс (+ 203%)

Избегайте повторного назначения имен свойств или методов в качестве переменных

Обычно не считается «лучшей практикой» повторно назначать зарезервированные имена свойств или методов как имя (имена) ваших собственных процедур и переменных.

Плохая форма. В то время как следующий (строго говоря) законный, рабочий код повторное назначение метода Find, а также свойства Row , Column и Address могут вызвать проблемы / конфликты с неоднозначностью имени и просто путают в целом.

```
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
        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
```

Хорошая форма. Все зарезервированные слова, переименованные в близкие, но уникальные аппроксимации оригиналов, избегали любых возможных конфликтов имен.

```
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
```

Хотя может наступить время, когда вы хотите намеренно переписать стандартный метод

или свойство в соответствии со своими собственными спецификациями, эти ситуации немногочисленны и далеки друг от друга. По большей части избегайте повторного использования зарезервированных имен для ваших собственных конструкций.

Прочитайте Лучшие практики VBA онлайн: https://riptutorial.com/ru/excel-vba/topic/1107/лучшие-практики-vba

глава 14: Массивы

Examples

Заполнение массивов (добавление значений)

Существует множество способов заполнения массива.

непосредственно

```
'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"
'...
```

Использование функции Array ()

```
'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"
```

От диапазона

```
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)
'-> 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 с оценкой ()

```
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)
```

Использование функции Split ()

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

Динамические массивы (изменение размера массива и динамическая обработка)

Из-за отсутствия эксклюзивного содержимого Excel-VBA этот пример был перенесен в документацию VBA.

Ссылка: динамические массивы (изменение размера массива и динамическая обработка)

Жесткие массивы (массивы массивов)

Из-за отсутствия эксклюзивного содержимого Excel-VBA этот пример был перенесен в документацию VBA.

Ссылка: Jagged Arrays (массивы массивов)

Проверьте, инициализирован ли массив (если он содержит элементы или нет).

Общей проблемой может быть попытка выполнить итерацию по массиву, в котором нет значений. Например:

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

Чтобы избежать этой проблемы и проверить, содержит ли массив Array элементы, используйте этот параметр *oneliner* :

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

Динамические массивы [Объявление массива, изменение размера]

```
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) '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
```

Прочитайте Массивы онлайн: https://riptutorial.com/ru/excel-vba/topic/2027/массивы

глава 15: Методы поиска последней использованной строки или столбца на листе

замечания

Вы можете найти хорошее объяснение того, почему другие методы не поощряются / неточны: http://stackoverflow.com/a/11169920/4628637

Examples

Найдите последнюю непустую ячейку в столбце

В этом примере мы рассмотрим метод возврата последней непустой строки в столбец для набора данных.

Этот метод будет работать независимо от пустых областей в наборе данных.

Однако *следует* ***соблюдать осторожность,*** *если задействованы* ***объединенные ячейки*** , поскольку метод `End` будет «остановлен» против объединенной области, возвращая первую ячейку объединенной области.

Кроме того, непустые ячейки в ***скрытых строках*** не будут учитываться.

```
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
```

Чтобы устранить указанные выше ограничения, строка:
`LastRow = wS.Cells(wS.Rows.Count, "A").End(xlUp).Row`

могут быть заменены на:

1. для последнего использованного ряда `"Sheet1"` :

   `LastRow = wS.UsedRange.Row - 1 + wS.UsedRange.Rows.Count .`

2. для последней непустой ячейки столбца `"A"` в `"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
```

Найти последнюю строку с использованием именованного диапазона

Если у вас есть Именованный диапазон в вашем листе, и вы хотите динамически получить последнюю строку этого динамического именованного диапазона. Также охватывает случаи, когда Named Range не начинается с первой строки.

```
Sub FindingLastRow()

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
```

Обновить:
Потенциальная лазейка была отмечена @Jeeped для aa named range с несмежными строками, поскольку она генерирует неожиданный результат. Чтобы решить эту проблему, код пересматривается, как показано ниже.
Аппции: таргеты 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
```

Получить строку последней ячейки в диапазоне

```
'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
```

Найти последнюю непустую колонку в рабочем листе

```
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
```

Вы можете выбрать один из двух вариантов, если хотите узнать, нет ли данных на листе:

- НЕТ: Используйте LastCol_1: вы можете использовать его непосредственно в `wS.Cells(...,LastCol_1(wS))`
- ДА: Использовать LastCol_0: вам нужно проверить, если результат, полученный вами от функции, равен 0 или нет, прежде чем использовать его

```
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
```

Свойства объекта Err автоматически сбрасываются до нуля после выхода функции.

```
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
```

Последняя ячейка в Range.CurrentRegion

`Range.CurrentRegion` - прямоугольная область диапазона, окруженная пустыми ячейками. Пустые ячейки с такими формулами, как `=""` или `'` , не считаются пустыми (даже `ISBLANK` Excel).

```
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)
```

Найти последнюю непустую строку в рабочем листе

```
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
```

Вы можете выбрать один из двух вариантов, если хотите узнать, нет ли данных на листе:

- НЕТ: Используйте LastRow_1: вы можете использовать его непосредственно в `wS.Cells(LastRow_1(wS),...)`
- YES: Использовать LastRow_0: вам нужно проверить, есть ли результат, полученный вами от функции, 0 или нет, прежде чем использовать его

```
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
```

Найти последнюю непустую ячейку в строке

В этом примере мы рассмотрим метод возврата последнего непустого столбца в строке.

Этот метод будет работать независимо от пустых областей в наборе данных.

Однако *следует* ***соблюдать осторожность,*** *если задействованы* ***объединенные ячейки*** , поскольку метод `End` будет «остановлен» против объединенной области, возвращая первую ячейку объединенной области.

Кроме того, непустые ячейки в ***скрытых столбцах*** не будут учитываться.

```
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
```

Найти последнюю непустую ячейку в рабочем листе - Производительность (массив)

- Первая функция, использующая массив, **намного быстрее**
- Если `.ThisWorkbook.ActiveSheet` без необязательного параметра, по умолчанию будет `.ThisWorkbook.ActiveSheet`
- Если диапазон пуст, он будет возвращать `Cell( 1, 1 )` по умолчанию вместо `Nothing`

Скорость:

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

. Измеряется с помощью 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$(uArr(r, c))) > 0 Then
                    lRow = r:   Exit For
                End If
            End If
        Next
        If lRow > 0 Then Exit For
    Next
    If lRow = 0 Then lRow = ubR
    For c = ubC To 1 Step -1    '---------------------------------------------- last col
        For r = lRow To 1 Step -1
            If Not IsError(uArr(r, c)) Then
                If Len(Trim$(uArr(r, c))) > 0 Then
                    Set GetLastCell = ws.Cells(lRow + uRng.Row - 1, c + uRng.Column - 1)
                    Exit Function
                End If
            End If
        Next
    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
    If cyFrequency = 0 Then getFrequency cyFrequency        'Get frequency
    getTickCount cyTicks1                                   'Get ticks
    If cyFrequency Then MicroTimer = cyTicks1 / cyFrequency 'Returns Seconds
End Function
```

Прочитайте Методы поиска последней использованной строки или столбца на листе онлайн: https://riptutorial.com/ru/excel-vba/topic/918/методы-поиска-последней-использованной-строки-или-столбца-на-листе

глава 16: Объединенные ячейки / диапазоны

Examples

Подумайте дважды, прежде чем использовать объединенные ячейки / диапазоны

Прежде всего, объединенные ячейки могут только улучшить внешний вид ваших листов.

Так что это буквально последнее, что вам нужно делать, как только ваш лист и книга полностью функциональны!

Где данные в объединенном диапазоне?

Когда вы объедините диапазон, вы увидите только один блок.

Данные будут в самой **первой ячейке этого диапазона** , а **остальные будут пустыми** !

Один хороший момент: нет необходимости заполнять все ячейки или диапазон после объединения, просто заполните первую ячейку! ;)

Другие аспекты этого объединенного диапазона являются глобально отрицательными:

- Если вы используете метод поиска последней строки или столбца , вы рискуете некоторыми ошибками
- Если вы выполняете цикл по строкам, и вы объединили некоторые диапазоны для лучшей читаемости, вы столкнетесь с пустым ячеек, а не с величиной, отображаемой объединенным диапазоном

Прочитайте Объединенные ячейки / диапазоны онлайн: https://riptutorial.com/ru/excel-vba/topic/7308/объединенные-ячейки---диапазоны

глава 17: Объект приложения

замечания

Excel VBA поставляется с всеобъемлющей *объектной моделью,* которая содержит классы и объекты, которые можно использовать для управления любой частью запускающего приложения Excel. Одним из наиболее распространенных объектов, которые вы будете использовать, является объект **Application** . Это верхний уровень, который представляет текущий исполняемый экземпляр Excel. Почти все, что не связано с конкретной книгой Excel, находится в объекте **Application** .

Объект *Application* , как объект верхнего уровня, имеет буквально сотни свойств, методов и событий, которые могут использоваться для управления каждым аспектом Excel.

Examples

Пример простого примера приложения: сведение к минимуму окна Excel

Этот код использует объект **приложения** верхнего уровня для минимизации основного окна Excel.

```
Sub MinimizeExcel()

    Application.WindowState = xlMinimized

End Sub
```

Пример простого примера приложения: отображение версии Excel и VBE

```
Sub DisplayExcelVersions()

    MsgBox "The version of Excel is " & Application.Version
    MsgBox "The version of the VBE is " & Application.VBE.Version

End Sub
```

Использование свойства Application.Version полезно для обеспечения того, что код работает только на совместимой версии Excel.

Прочитайте Объект приложения онлайн: https://riptutorial.com/ru/excel-vba/topic/5645/объект-приложения

глава 18: Объект файловой системы

Examples

Файл, папка, привод существует

Файл существует:

```
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
```

Папка существует:

```
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
```

Диск существует:

```
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
```

Основные операции с файлами

Копирование:

```
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
```

Переехать:

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

Удалять:

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

Основные операции с папками

Создайте:

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

Копирование:

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

Переехать:

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

Удалять:

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

Другие операции

Получить имя файла:

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

Результат: *Makro.txt*

Получить базовое имя:

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

Результат: *Макро*

Получить имя добавочного номера:

```
Sub GetExtensionName()
    Dim fso as Scripting.FileSystemObject
```

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

Результат: *txt*

Получить имя диска:

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

Результат: *c:*

Прочитайте Объект файловой системы онлайн: https://riptutorial.com/ru/excel-vba/topic/9933/объект-файловой-системы

глава 19: Оптимизация Excel-VBA

Вступление

Оптимизация Excel-VBA также относится к кодированию лучшей обработки ошибок с помощью документации и дополнительной информации. Это показано здесь.

замечания

*) Номера строк представляют собой целые числа, то есть подписанный 16-битный тип данных в диапазоне от -32,768 до 32,767, в противном случае вы получаете переполнение. Обычно номера строк вставляются с шагом 10 по части кода или всем процедурам модуля в целом.

Examples

Отключение обновления рабочего листа

Отключение вычисления рабочего листа может значительно сократить время выполнения макроса. Более того, отключение событий, обновление экрана и разрывы страниц были бы полезными. `Sub` может использоваться в любых макросах для этой цели.

```
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
```

Для оптимизации следуйте приведенному ниже псевдокоду:

```
Sub MyCode()

    OptimizeVBA True

    'Your code goes here

    OptimizeVBA False

End Sub
```

Проверка времени выполнения

Различные процедуры могут выдавать один и тот же результат, но они будут использовать другое время обработки. Чтобы проверить, какой из них быстрее, можно использовать

такой код:

```
time1 = Timer

For Each iCell In MyRange
   iCell = "text"
Next iCell

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
```

Использование с блоками

Использование с блоками может ускорить процесс запуска макроса. Вместо написания диапазона, имени диаграммы, рабочего листа и т. Д. Вы можете использовать с-блоками, как показано ниже;

```
With ActiveChart
    .Parent.Width = 400
    .Parent.Height = 145
    .Parent.Top = 77.5 + 165 * step - replacer * 15
    .Parent.Left = 5
End With
```

Это быстрее, чем это:

```
ActiveChart.Parent.Width = 400
ActiveChart.Parent.Height = 145
ActiveChart.Parent.Top = 77.5 + 165 * step - replacer * 15
```

```
ActiveChart.Parent.Left = 5
```

Заметки:

- После ввода блока «С» объект нельзя изменить. В результате вы не можете использовать один оператор WITH для воздействия на несколько разных объектов
- **Не прыгайте в или из С блоками** . Если выполняются операторы в блоке С, но оператор WITH или End With не выполняется, **временная переменная, содержащая ссылку на объект, остается в памяти до тех пор, пока вы не выйдете из процедуры**
- Не замыкайтесь внутри. С операторами, особенно если кешированный объект используется как итератор
- Вы можете вставлять с утверждениями, помещая один с блоком в другой. Однако, поскольку элементы внешних блоков Block замаскированы внутри внутреннего блока С, вы должны предоставить полностью квалифицированную ссылку на объект во внутреннем блоке с любым элементом объекта во внешнем блоке С.

Пример гнездования:

В этом примере используется оператор WITH для выполнения ряда операторов для одного объекта.
Объект и его свойства - это общие имена, используемые только для иллюстрации.

```
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
```

Дополнительная информация о MSDN

Удаление строк - производительность

- Удаление строк происходит медленно, особенно при переходе по ячейкам и удалении строк один за другим
- Другой подход - использовать AutoFilter, чтобы скрыть удаляемые строки
- Скопируйте видимый диапазон и вставьте его в новый рабочий лист
- Удалите исходный лист целиком

- С помощью этого метода, чем больше строк будет удалено, тем быстрее будет

Пример:

```
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
```

Отключение всех функций Excel Перед выполнением больших макросов

Нижеприведенные процедуры временно отключат все функции Excel на уровне WorkBook и WorkSheet

- FastWB () - это переключатель, который принимает флаги Вкл. Или Выкл.
- FastWS () принимает необязательный объект WorkSheet или нет
- Если параметр ws отсутствует, он включит и выключит все функции для всех рабочих таблиц в коллекции
    - Пользовательский тип может использоваться для захвата всех параметров перед их отключением
    - В конце процесса начальные настройки могут быть восстановлены

```
Public Sub FastWB(Optional ByVal opt As Boolean = True)
    With Application
        .Calculation = IIf(opt, xlCalculationManual, xlCalculationAutomatic)
        If .DisplayAlerts <> Not opt Then .DisplayAlerts = Not opt
        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
```

Восстановить все настройки Excel по умолчанию.

```
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
    End With
End Sub
```

Оптимизация поиска ошибок с помощью расширенной отладки

Использование номеров строк ... и документирование их в случае ошибки («важность

просмотра Erl»)

Обнаружение какой строки вызывает ошибку, является существенной частью любой отладки и сужает поиск причины. Для документирования идентифицированных строк ошибок с кратким описанием завершается успешное отслеживание ошибок, в лучшем случае вместе с именами модуля и процедуры. Ниже приведен пример сохранения этих данных в файл журнала.

Фон

Объект ошибки возвращает номер ошибки (Err.Number) и описание ошибки (Err.Description), но явно не отвечает на вопрос, где найти ошибку. Однако функция **Erl** делает, но при условии, что вы добавляете * *номера строк)* к коду (BTW одна из нескольких других уступок бывшим основным временам).

Если линий ошибок вообще нет, то функция Erl возвращает 0, если нумерация неполна, вы получите последний номер строки предыдущей строки.

```
Option Explicit


Public Sub MyProc1()
Dim i As Integer
Dim j As Integer
On Error GoTo LogErr
10     j = 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
' Return error text
  LogErrors = sLogTxt
 End Function
```

' Дополнительный код для отображения файла журнала

```
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
```

Прочитайте Оптимизация Excel-VBA онлайн: https://riptutorial.com/ru/excel-vba/topic/9798/оптимизация-excel-vba

глава 20: Отладка и устранение неполадок

Синтаксис

- Debug.Print (строка)
- Стоп () / Стоп

Examples

Debug.Print

Чтобы распечатать список описаний кодов ошибок в окне «Немедленное», перейдите к функции `Debug.Print` :

```
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
```

Вы можете показать окно «Немедленное»:

- Выбор **V** iew | **I** mmediate Window из строки меню
- Использование сочетания клавиш **Ctrl-G**

Стоп

Команда Stop приостанавливает выполнение при вызове. Оттуда процесс можно возобновить или выполнить шаг за шагом.

```
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
```

Немедленное окно

Если вы хотите протестировать строку макрокода, не требуя запуска всего суб, вы можете вводить команды непосредственно в окно Immediate и нажимать `ENTER` для запуска строки.

Для тестирования вывода строки вы можете поставить перед ним знак вопроса `?` для печати непосредственно в окне Immediate. В качестве альтернативы вы также можете

использовать команду `print` для вывода вывода.

В редакторе Visual Basic нажмите `CTRL + G` чтобы открыть окно Immediate. Чтобы переименовать текущий выбранный лист в «ExampleSheet», введите следующее в окне «Немедленное» и нажмите `ENTER`

```
   ActiveSheet.Name = "ExampleSheet"
```

Чтобы напечатать имя текущего выбранного листа непосредственно в окне «Немедленное»

```
? ActiveSheet.Name
ExampleSheet
```

Этот метод может быть очень полезен для проверки функциональности встроенных или определенных пользователем функций перед их внедрением в код. В приведенном ниже примере показано, как окно Immediate можно использовать для проверки вывода функции или ряда функций для подтверждения ожидаемого.

```
'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
```

Непосредственное окно также может использоваться для установки или сброса приложений, рабочей книги или других необходимых свойств. Это может быть полезно, если у вас есть `Application.EnableEvents = False` в подпрограмме, которая неожиданно вызывает ошибку, заставляя ее закрываться без сброса значения в `True` (что может вызвать разочарование и неожиданную функциональность. В этом случае команды могут быть введены непосредственно в окно Immediate и запустите:

```
? 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
```

Для более сложных методов отладки двоеточие `:` можно использовать в качестве разделителя строк. Это можно использовать для многострочных выражений, таких как цикл в приведенном ниже примере.

```
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
```

Использование таймера для поиска узких мест в производительности

Первым шагом в оптимизации скорости является поиск самых медленных разделов кода. Функция `Timer` VBA возвращает количество секунд, прошедших с полуночи, с точностью 1/256 секунды (3,90625 миллисекунды) на компьютерах под управлением Windows. Функции VBA `Now` и `Time` верны с точностью до секунды.

```
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"
```

Добавление точки останова к вашему коду

Вы можете легко добавить точку останова в свой код, нажав на серый столбец слева от строки вашего кода VBA, где вы хотите остановить выполнение. В столбце появляется красная точка, а код точки останова также выделяется красным цветом.

Вы можете добавить несколько точек останова по всему вашему коду и возобновить выполнение, нажав значок «играть» в строке меню. Не весь код может быть точкой останова в качестве строк определения, первая или последняя строка процедуры и строки комментариев не могут быть выбраны в качестве точки останова.

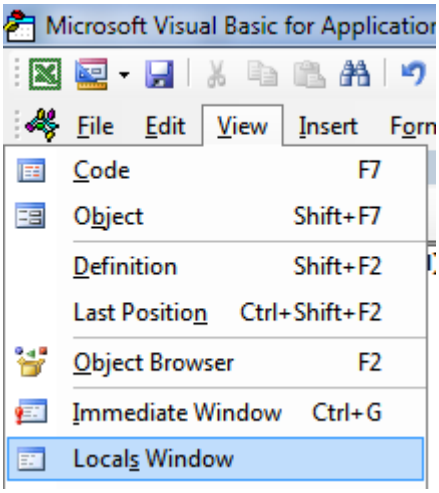
Окно локаторов отладки

Окно Locals обеспечивает легкий доступ к текущему значению переменных и объектов в рамках функции или подпрограммы, в которой вы работаете. Это важный инструмент для отладки вашего кода и перехода к изменениям, чтобы найти проблемы. Он также позволяет вам исследовать свойства, которые, возможно, не были известны.

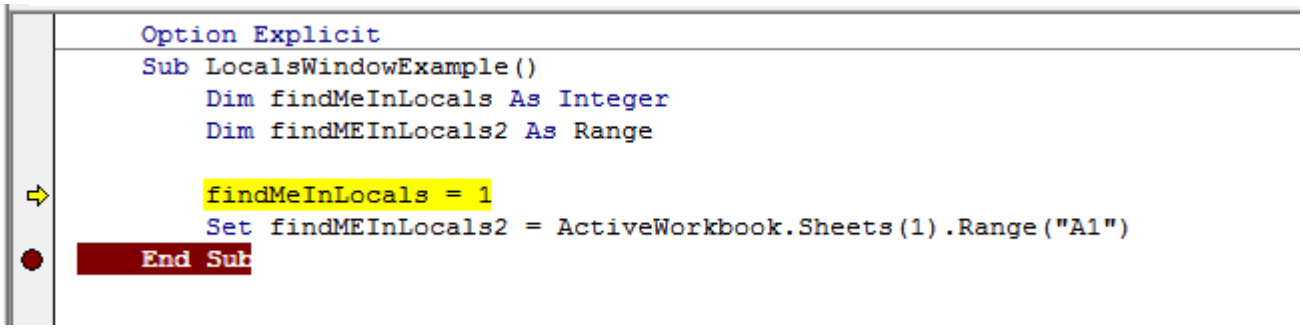
Возьмем следующий пример:

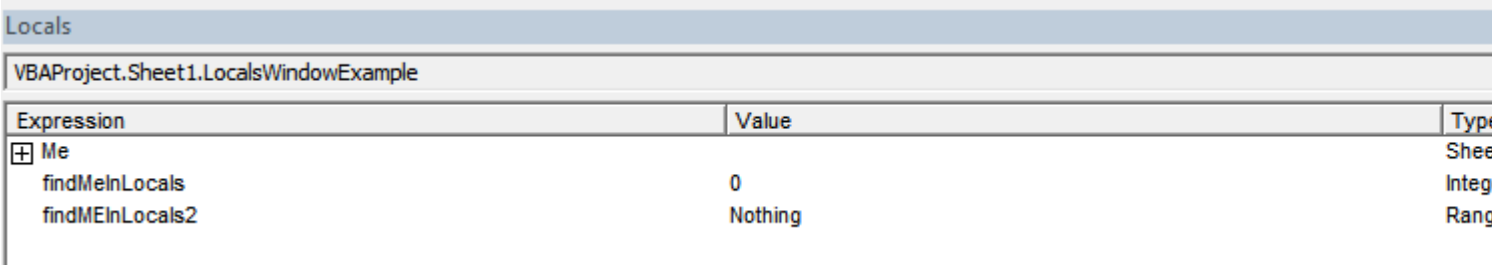
```
Option Explicit
Sub LocalsWindowExample()
    Dim findMeInLocals As Integer
    Dim findMEInLocals2 As Range

    findMeInLocals = 1
    Set findMEInLocals2 = ActiveWorkbook.Sheets(1).Range("A1")
End Sub
```

В редакторе VBA нажмите «Вид» -> «Окно локалей».



Затем, перейдя через код с помощью F8 после нажатия внутри подпрограммы, мы остановились перед тем, как назначить findMeinLocals. Ниже вы можете увидеть, что значение равно 0 --- и это то, что будет использовано, если вы никогда не назначили ему значение. Объект диапазона - «Ничего».

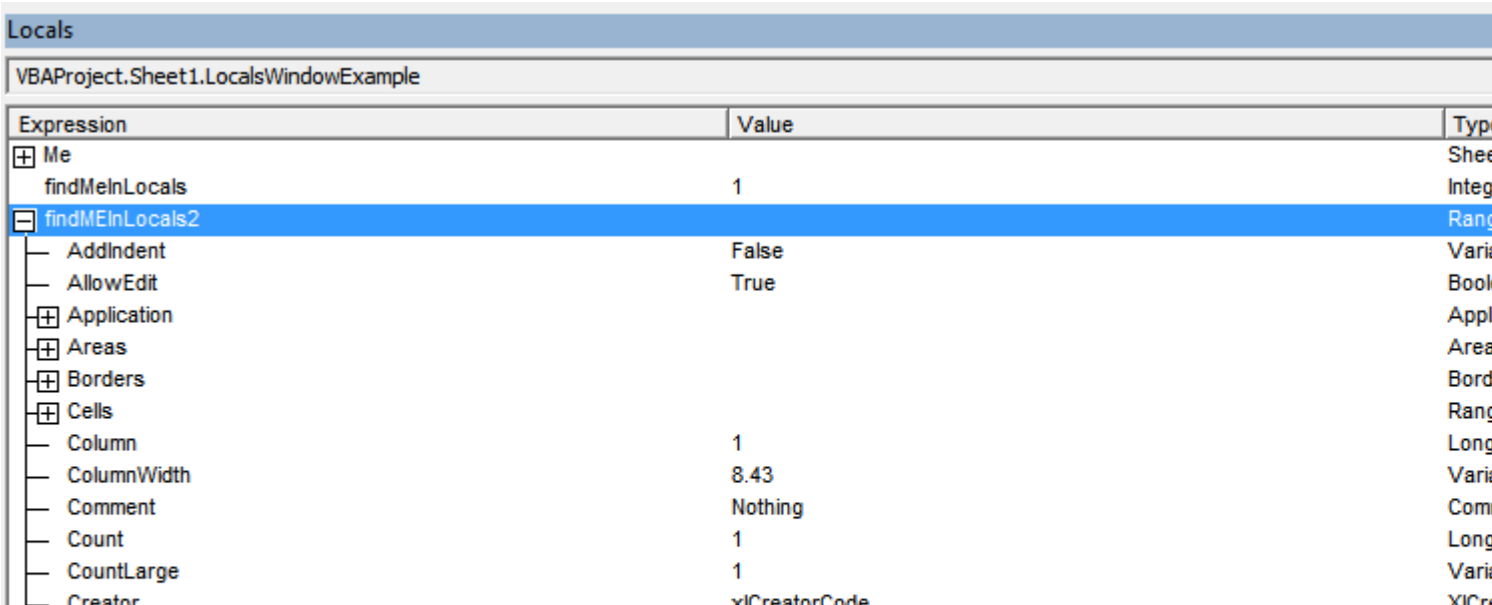




Если мы остановимся до завершения подпрограммы, мы увидим окончательные значения переменных.

```
Option Explicit
Sub LocalsWindowExample()
    Dim findMeInLocals As Integer
    Dim findMEInLocals2 As Range

    findMeInLocals = 1
    Set findMEInLocals2 = ActiveWorkbook.Sheets(1).Range("A1")
End Sub
```

Мы можем видеть findMeInLocals со значением 1 и типом Integer и FindMeInLocals2 с типом Range / Range. Если щелкнуть знак +, мы можем развернуть объект и увидеть его свойства, такие как счетчик или столбец.



Прочитайте Отладка и устранение неполадок онлайн: https://riptutorial.com/ru/excel-vba/topic/861/отладка-и-устранение-неполадок

глава 21: переплет

Examples

Ранняя привязка против поздней привязки

Связывание - это процесс назначения объекта идентификатору или имени переменной. Раннее связывание (также известное как статическое связывание) - это когда объект, объявленный в Excel, имеет определенный тип объекта, такой как рабочий лист или рабочая книга. Позднее связывание происходит при создании общих ассоциаций объектов, таких как типы объявлений Object и Variant.

Раннее связывание ссылок имеет некоторые преимущества перед поздним связыванием.

- Раннее связывание оперативно быстрее, чем позднее связывание во время выполнения. Создание объекта с поздним связыванием во время выполнения требует времени, которое раннее связывание выполняется, когда проект VBA изначально загружается.
- Раннее связывание предлагает дополнительную функциональность путем идентификации пар ключей / предметов по их порядковой позиции.
- В зависимости от структуры кода раннее связывание может предложить дополнительный уровень проверки типов и уменьшить ошибки.
- Коррекция капитализации VBE при вводе свойств и методов связанного объекта активна с ранней привязкой, но недоступна с поздним связыванием.

  Примечание. Чтобы реализовать раннее связывание, вы должны добавить соответствующую ссылку на проект VBA с помощью команды «Инструменты» → «Ссылки» VBE.

  Эта библиографическая ссылка затем переносится с проектом; ему не нужно повторно ссылаться, когда проект VBA распространяется и запускается на другом компьютере.

```
'Looping through a dictionary that was created with late binding¹
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[1]
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"

    '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
```

Однако, если вы используете раннее связывание, и документ запускается в системе, в которой отсутствует одна из библиотек, на которую вы ссылались, вы столкнетесь с проблемами. Мало того, что подпрограммы, которые используют недостающую библиотеку, не работают должным образом, но поведение всего кода внутри документа станет неустойчивым. Вполне вероятно, что ни один из кодов документа не будет работать на этом компьютере.

Это то, где поздняя привязка выгодна. При использовании позднего связывания вам не нужно добавлять ссылку в меню «Инструменты»> «Ссылки». На машинах, имеющих соответствующую библиотеку, код будет работать. На машинах без этой библиотеки команды, ссылающиеся на библиотеку, не будут работать, но все остальные коды в вашем документе будут продолжать функционировать.

Если вы не знакомы с библиотекой, на которую вы ссылаетесь, может быть полезно использовать раннее связывание при написании кода, а затем перейти к позднему связыванию перед развертыванием. Таким образом, вы можете использовать возможности

IntelliSense и обозревателя объектов VBE во время разработки.

Прочитайте переплет онлайн: https://riptutorial.com/ru/excel-vba/topic/3811/переплет

глава 22: Поиск повторяющихся значений в диапазоне

Вступление

В определенные моменты вы будете оценивать диапазон данных, и вам нужно будет найти дубликаты в нем. Для больших наборов данных существует ряд подходов, которые вы можете использовать, используя код VBA или условные функции. В этом примере используется простое условие if-then в течение двух вложенных циклов for-next для проверки того, равна ли каждая ячейка в диапазоне для любой другой ячейки в диапазоне.

Examples

Найти дубликаты в диапазоне

Следующие тесты варьируются от A2 до A7 для повторяющихся значений. **Примечание.** Этот пример иллюстрирует возможное решение в качестве первого подхода к решению. Быстрее использовать массив, чем диапазон, и можно использовать коллекции или словари или методы xml для проверки дубликатов.

```
    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("Duplicates")

' 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
```

В зависимости от ваших потребностей пример может быть изменен - например, верхний предел n может быть значением строки последней ячейки с данными в диапазоне, или действие в случае условия Истина If может быть отредактировано для извлечения дубликата ценность в другом месте. Однако механика рутины не изменилась.

Прочитайте Поиск повторяющихся значений в диапазоне онлайн:
https://riptutorial.com/ru/excel-vba/topic/8295/поиск-повторяющихся-значений-в-диапазоне

глава 23: Пользовательские функции (UDF)

Синтаксис

1. **Функция functionName (argumentVariable As dataType, argumentVariable2 As dataType, Необязательный аргументVariable3 В качестве dataType) В качестве функцииReturnDataType**
   Базовое объявление функции. Каждая функция нуждается в имени, но ей не нужно принимать какие-либо аргументы. Он может принимать 0 аргументов или может принимать заданное количество аргументов. Вы также можете объявить аргумент как необязательный (это означает, что не имеет значения, укажите ли вы его при вызове функции). Рекомендуется использовать тип переменной для каждого аргумента, а также возвращать тип данных, которые сама функция будет возвращать.

2. **functionName = theVariableOrValueBeingReturned**
   Если вы используете другие языки программирования, вы можете использовать ключевое слово `Return` . Это не используется в VBA - вместо этого мы используем имя функции. Вы можете установить его на содержимое переменной или на какое-то прямое значение. Обратите внимание, что если вы задали тип данных для возврата функции, переменная или данные, которые вы поставляете на этот раз, должны быть из этого типа данных.

3. **Конечная функция**
   Обязательный. Указывает конец `Function` блока и должен быть таким образом в конце. VBE обычно предоставляет это автоматически при создании новой функции.

замечания

Пользовательская функция (также называемая UDF) относится к определенной функции, созданной пользователем. Его можно назвать функцией рабочего листа (ex: `=SUM(...)`) или использоваться для возврата значения в выполняемый процесс в процедуре Sub. UDF возвращает значение, как правило, из информации, переданной в нее как один или несколько параметров.

Его можно создать:

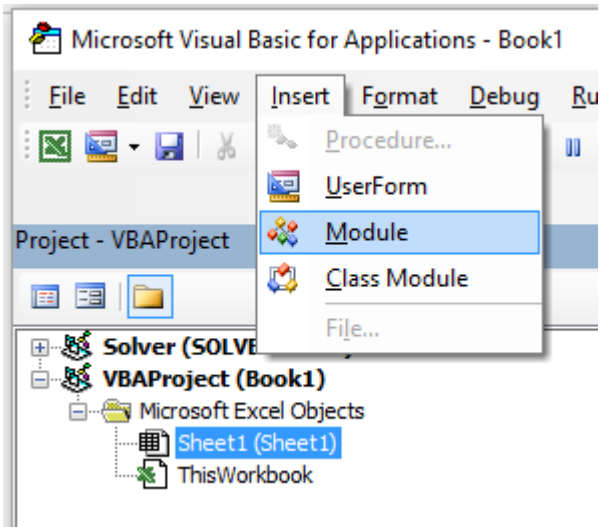
1. используя VBA.
2. используя API Excel C - создав XLL, который экспортирует скомпилированные функции в Excel.

3. используя COM-интерфейс.

Examples

UDF - Hello World

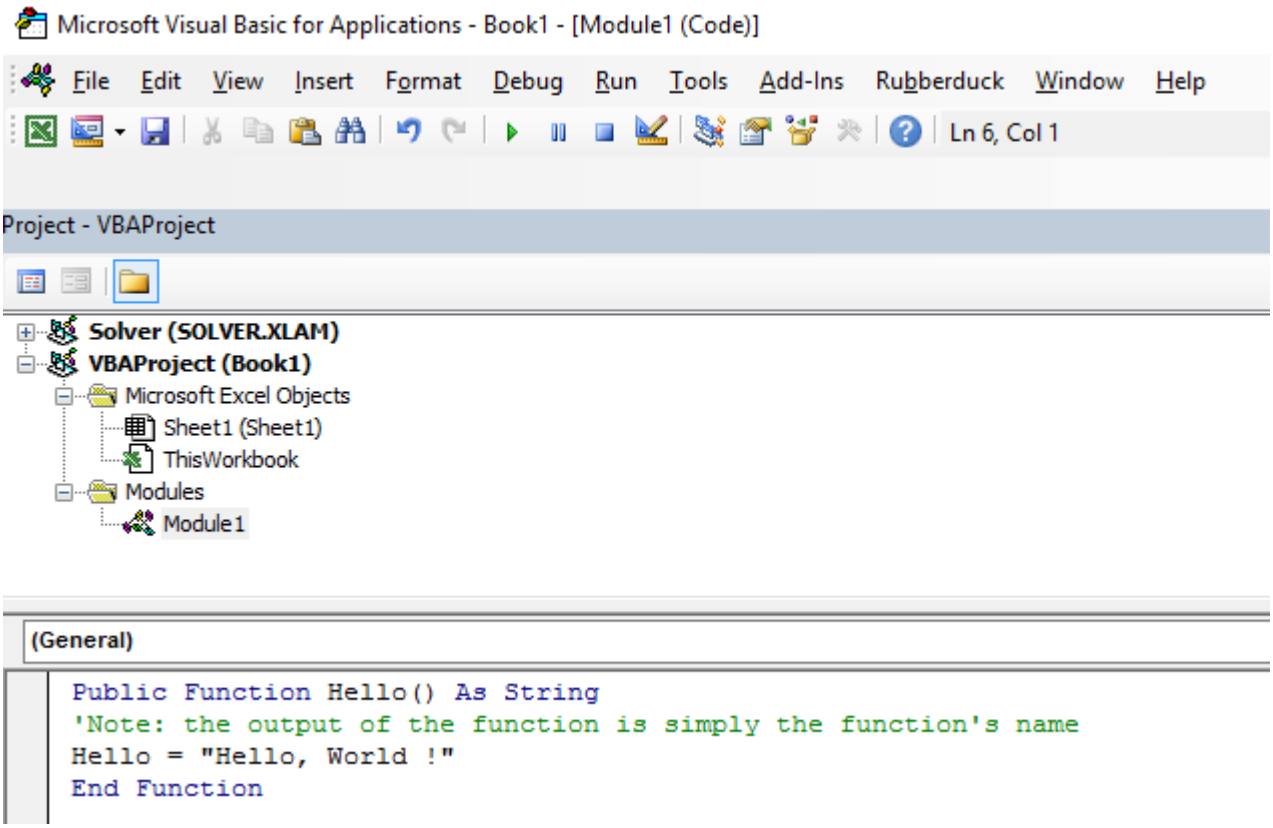
1. Открыть Excel
2. Откройте редактор Visual Basic (см. Раздел Открытие редактора Visual Basic)
3. Добавьте новый модуль, нажав Insert -> Module:



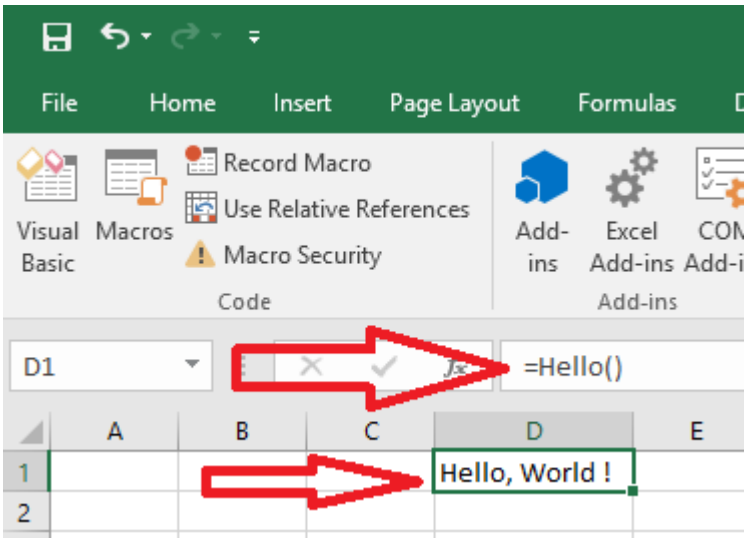
4. Скопируйте и вставьте следующий код в новый модуль:

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

Чтобы получить :



5. Вернитесь к своей книге и введите «= Hello ()» в ячейку, чтобы увидеть «Hello World».



Разрешить полную колонку без штрафа

Легче реализовать некоторые UDF на листе, если в качестве параметров можно передать полные ссылки на столбцы. Однако из-за явного характера кодирования любая петля, включающая эти диапазоны, может обрабатывать сотни тысяч ячеек, которые полностью пусты. Это уменьшает ваш проект VBA (и рабочую книгу) до замороженного беспорядка, в то время как ненужные не-значения обрабатываются.

Цикл через ячейки листа является одним из самых медленных методов выполнения задачи, но иногда это неизбежно. Сокращение работы, выполняемой до того, что на самом деле требуется, имеет смысл.

Решение состоит в том, чтобы обрезать полный столбец или ссылки полной строки на свойство Worksheet.UsedRange с помощью метода Intersect . Следующий образец будет свободно реплицировать собственную SUMIF-функцию рабочего листа, так что *критерий_range* также будет изменен в соответствии с *sum_range,* поскольку каждое значение в *sum_range* должно сопровождаться значением в *критерие_range* .

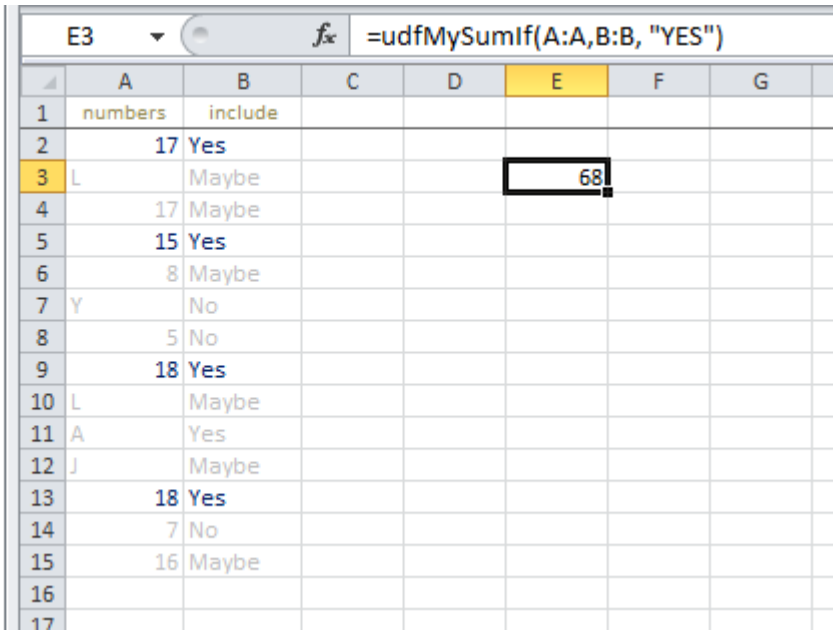
Application.Caller для UDF, используемого на листе, является ячейкой, в которой он находится. Свойство .Parent ячейки является листом . Это будет использоваться для определения .UsedRange.

В листе кода модуля:

```
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
```

Синтаксис:

```
=udfMySumIf(*sum_range*, *criteria_range*, [*criteria*])
```



Хотя это довольно упрощенный пример, он адекватно демонстрирует передачу двух полных столбцов (1 048 576 строк каждая), но обрабатывает только 15 строк данных и критериев.

Связанная официальная документация MSDN по отдельным методам и свойствам предоставлена Microsoft ™.

Count Уникальные значения в диапазоне

```
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
```

Коллекции

Прочитайте Пользовательские функции (UDF) онлайн: https://riptutorial.com/ru/excel-vba/topic/1070/пользовательские-функции--udf-

глава 24: Прокрутка всех листов в активной рабочей книге

Examples

Получить все имена рабочих листов в 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
```

Цикл через все листы во всех файлах в папке

```
 Sub Theloopofloops()

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


 path = "pathtofile(s)" & "\"
 Filename = Dir(path & "*.xl??")
 Set wsO = 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
```

```
specifiying 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 False
     Filename = Dir
 Loop
 End Sub
```

Прочитайте Прокрутка всех листов в активной рабочей книге онлайн: https://riptutorial.com/ru/excel-vba/topic/1144/прокрутка-всех-листов-в-активной-рабочей-книге

глава 25: Работа с таблицами Excel в VBA

Вступление

Этот раздел посвящен работе с таблицами в VBA и предполагает знание таблиц Excel. В VBA, или, скорее, в Object Object Model, таблицы называются ListObjects. Наиболее часто используемыми свойствами ListObject являются ListRow (s), ListColumn (s), DataBodyRange, Range и HeaderRowRange.

Examples

Создание экземпляра 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
```

Работа с 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
```

Преобразование таблицы Excel в обычный диапазон

```
Dim lo as ListObject

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

Прочитайте Работа с таблицами Excel в VBA онлайн: https://riptutorial.com/ru/excel-vba/topic/9753/работа-с-таблицами-excel-в-vba

глава 26: Распространенные ошибки

Examples

Квалификационные ссылки

При обращении к `worksheet` , `range` или отдельным `cells` важно полностью квалифицировать ссылку.

Например:

```
ThisWorkbook.Worksheets("Sheet1").Range(Cells(1, 2), Cells(2, 3)).Copy
```

Не является полностью квалифицированным: ссылки на `Cells` не имеют рабочей книги и рабочего листа, связанных с ними. Без явной ссылки Cells ссылается на `ActiveSheet` по умолчанию. Таким образом, этот код не сработает (`ActiveSheet` неверные результаты), если `ActiveSheet` отличным от `Sheet1` является текущий `ActiveSheet` .

Самый простой способ исправить это использовать `With` утверждением следующим образом :

```
With ThisWorkbook.Worksheets("Sheet1")
    .Range(.Cells(1, 2), .Cells(2, 3)).Copy
End With
```

Кроме того, вы можете использовать переменную Worksheet. (Это, скорее всего, будет предпочтительным, если ваш код должен ссылаться на несколько рабочих листов, например, копирование данных с одного листа на другой.)

```
Dim ws1 As Worksheet
Set ws1 = ThisWorkbook.Worksheets("Sheet1")
ws1.Range(ws1.Cells(1, 2), ws1.Cells(2, 3)).Copy
```

Другая частая проблема заключается в ссылке на коллекцию Рабочих таблиц без квалификации рабочей книги. Например:

```
Worksheets("Sheet1").Copy
```

`Sheet1` не имеет полной квалификации и отсутствует рабочая тетрадь. Это может завершиться неудачей, если в коде ссылаются несколько книг. Вместо этого используйте одно из следующих действий:

```
ThisWorkbook.Worksheets("Sheet1")       '<--ThisWorkbook refers to the workbook containing
                                        'the running VBA code
Workbooks("Book1").Worksheets("Sheet1") '<--Where Book1 is the workbook containing Sheet1
```

Однако не используйте следующее:

```
ActiveWorkbook.Worksheets("Sheet1")     '<--Valid, but if another workbook is activated
                                        'the reference will be changed
```

Аналогично для объектов `range` , если они явно не определены, `range` будет относиться к текущему активному листу:

```
Range("a1")
```

Такой же как:

```
ActiveSheet.Range("a1")
```

Удаление строк или столбцов в цикле

Если вы хотите удалить строки (или столбцы) в цикле, вы всегда должны начинать цикл с конца диапазона и перемещаться назад на каждом шаге. В случае использования кода:

```
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
```

Вы пропустите несколько строк. Например, если код удаляет строку 3, то строка 4 становится строкой 3. Однако переменная `i` изменится на 4. Таким образом, в этом случае код пропустит одну строку и проверит другую, которая ранее не была в диапазоне.

Правильный код будет

```
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
```

ActiveWorkbook против ThisWorkbook

`ActiveWorkbook` и `ThisWorkbook` иногда становятся взаимозаменяемыми новыми пользователями VBA без полного понимания того, к чему относится каждый объект, что может привести к нежелательному поведению во время выполнения. Оба этих объекта относятся к объекту приложения

Объект `ActiveWorkbook` относится к `ActiveWorkbook` которая в настоящее время находится в

самом верхнем виде объекта приложения Excel во время выполнения. *(например, книгу, которую вы можете видеть и взаимодействовать в момент, когда этот объект ссылается)*

```
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
```

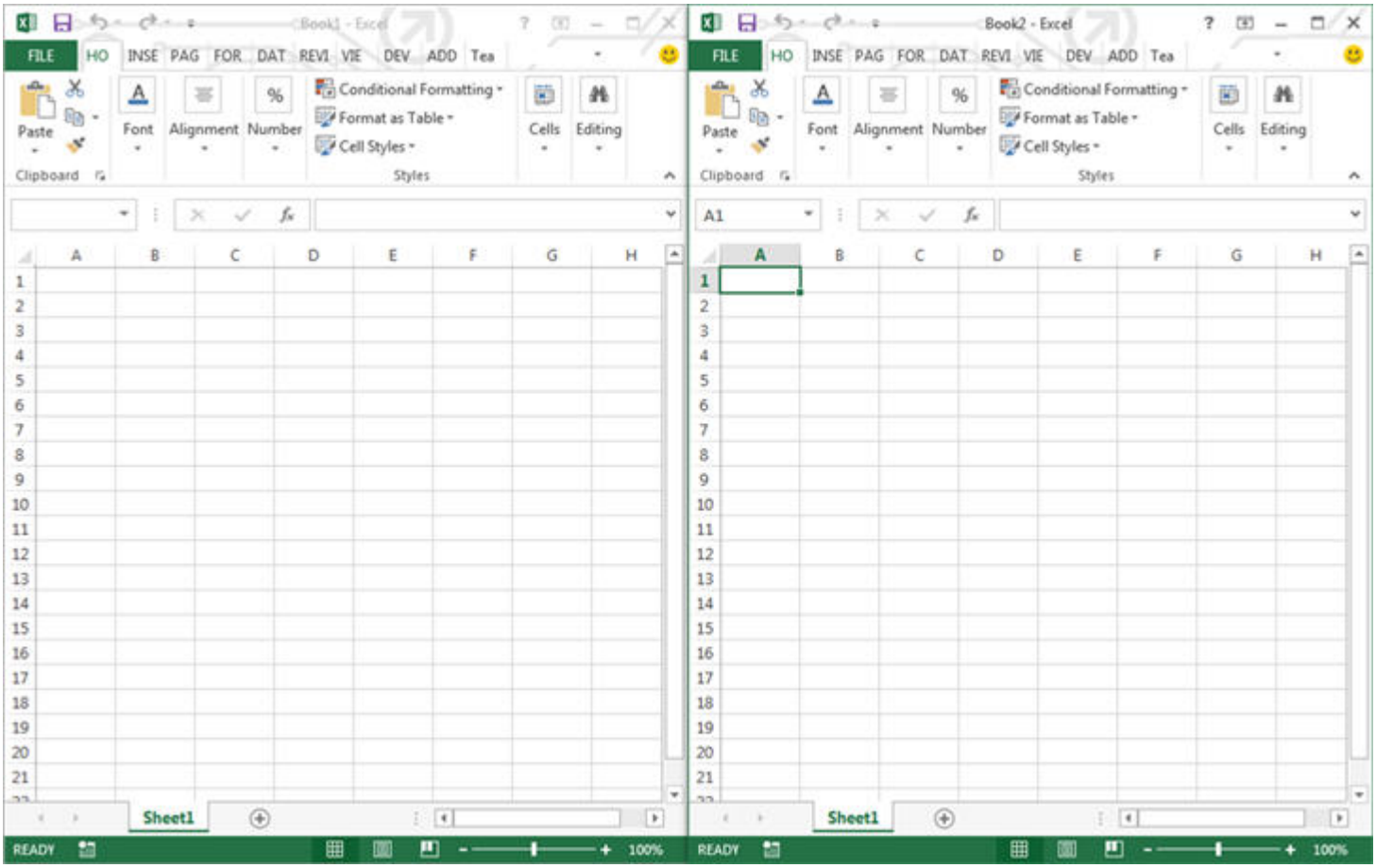
Объект `ThisWorkbook` относится к книге, в которой код принадлежит к моменту его выполнения.

```
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 '// Prints "Bar"
    Debug.Print ThisWorkbook.Sheet1.Range("A1").Value '// Prints "Foo"

End Sub
```

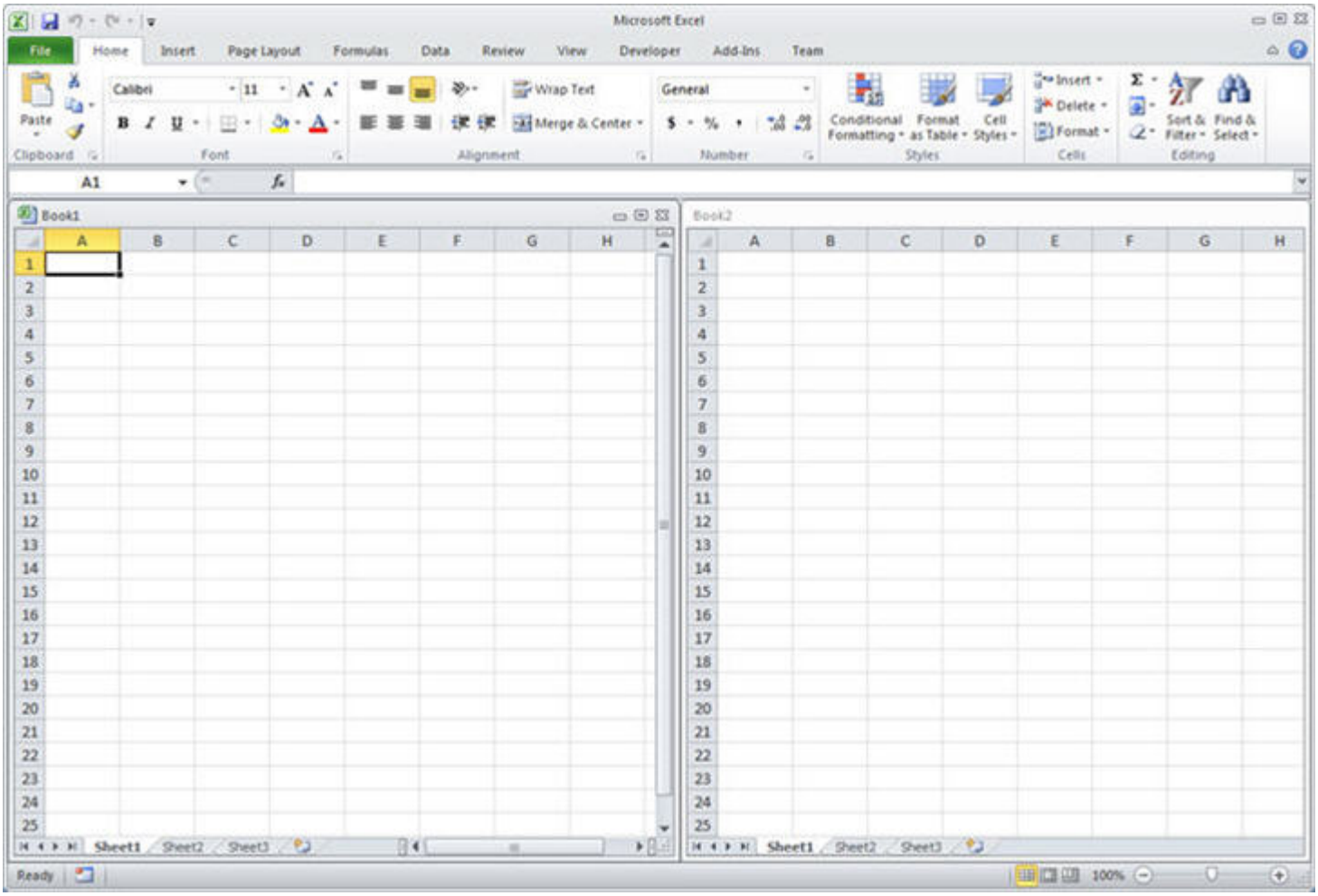
Интерфейс с одним документом и несколькими документами

> Имейте в виду, что Microsoft Excel 2013 (и выше) использует Single Document Interface (SDI), а Excel 2010 (и ниже) использует Multiple Document Interfaces (MDI).

Это означает, что для Excel 2013 (SDI) каждая рабочая тетрадь в одном экземпляре Excel содержит **собственный** интерфейс ленты:



И наоборот, для Excel 2010 каждая рабочая тетрадь в одном экземпляре Excel использовала **общий** интерфейс (MDI) с лентой:



Это вызывает некоторые важные проблемы, если вы хотите перенести код VBA (2010 <-> 2013), который взаимодействует с лентой.

> Необходимо создать процедуру обновления ленточных элементов управления пользовательского интерфейса в одном и том же состоянии во всех книгах для Excel 2013 и выше.

Обратите внимание, что :

1. Все методы, события и свойства окна на уровне приложения Excel остаются неизменными. (`Application.ActiveWindow` , `Application.Windows` ...)
2. В Excel 2013 и выше (SDI) все окна, методы и свойства окна на рабочем столе теперь работают в окне верхнего уровня. Можно получить дескриптор этого окна верхнего уровня с помощью `Application.Hwnd`

Чтобы получить более подробную информацию, см. Источник этого примера: MSDN .

Это также вызывает некоторые проблемы с немодальными пользовательскими формами. См. Здесь для решения.

Прочитайте Распространенные ошибки онлайн: https://riptutorial.com/ru/excel-vba/topic/1576/распространенные-ошибки

глава 27: Сводные таблицы

замечания

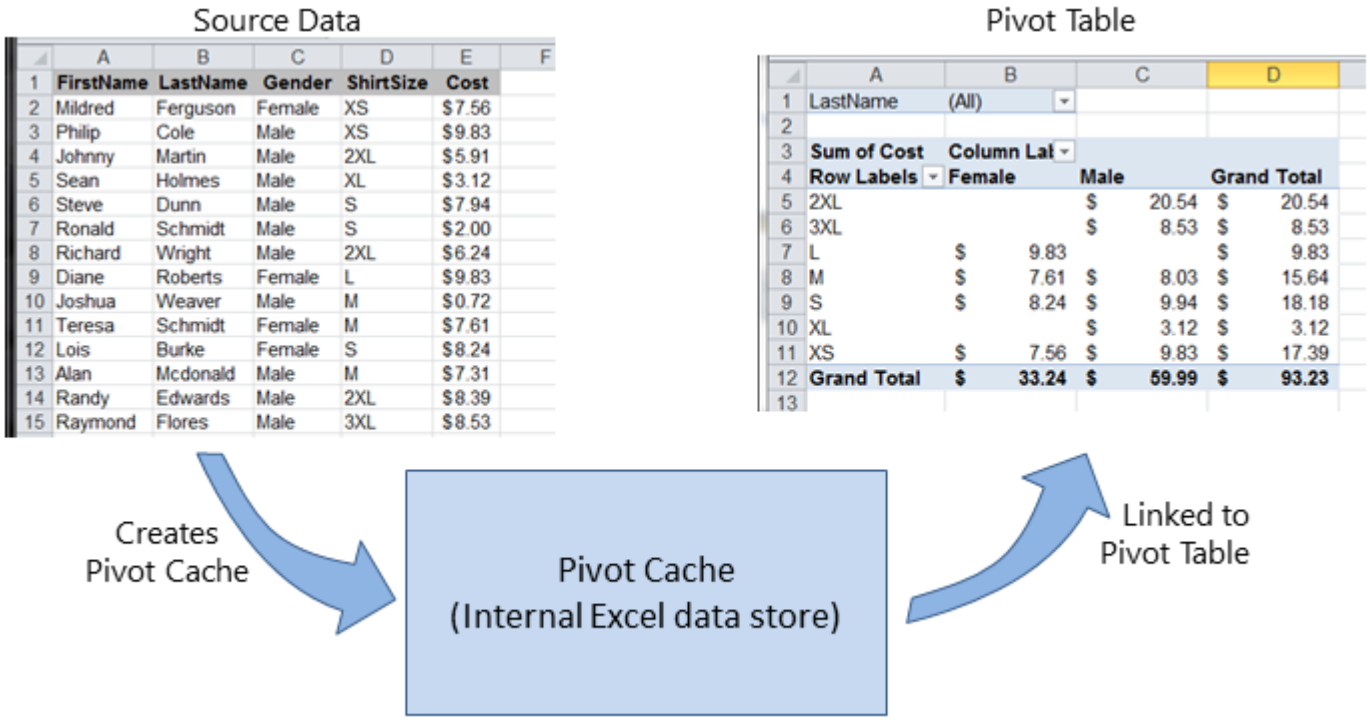
В Интернете много отличных справочных и примерных источников. Некоторые примеры и объяснения создаются здесь как точка сбора для быстрого ответа. Более подробные иллюстрации могут быть связаны с внешним контентом (вместо копирования существующего оригинального материала).

Examples

Создание сводной таблицы

Одной из самых мощных возможностей в Excel является использование сводных таблиц для сортировки и анализа данных. Использование VBA для создания и управления Pivots проще, если вы понимаете взаимосвязь Pivot Tables с Pivot Caches и как ссылаться и использовать разные части таблиц.

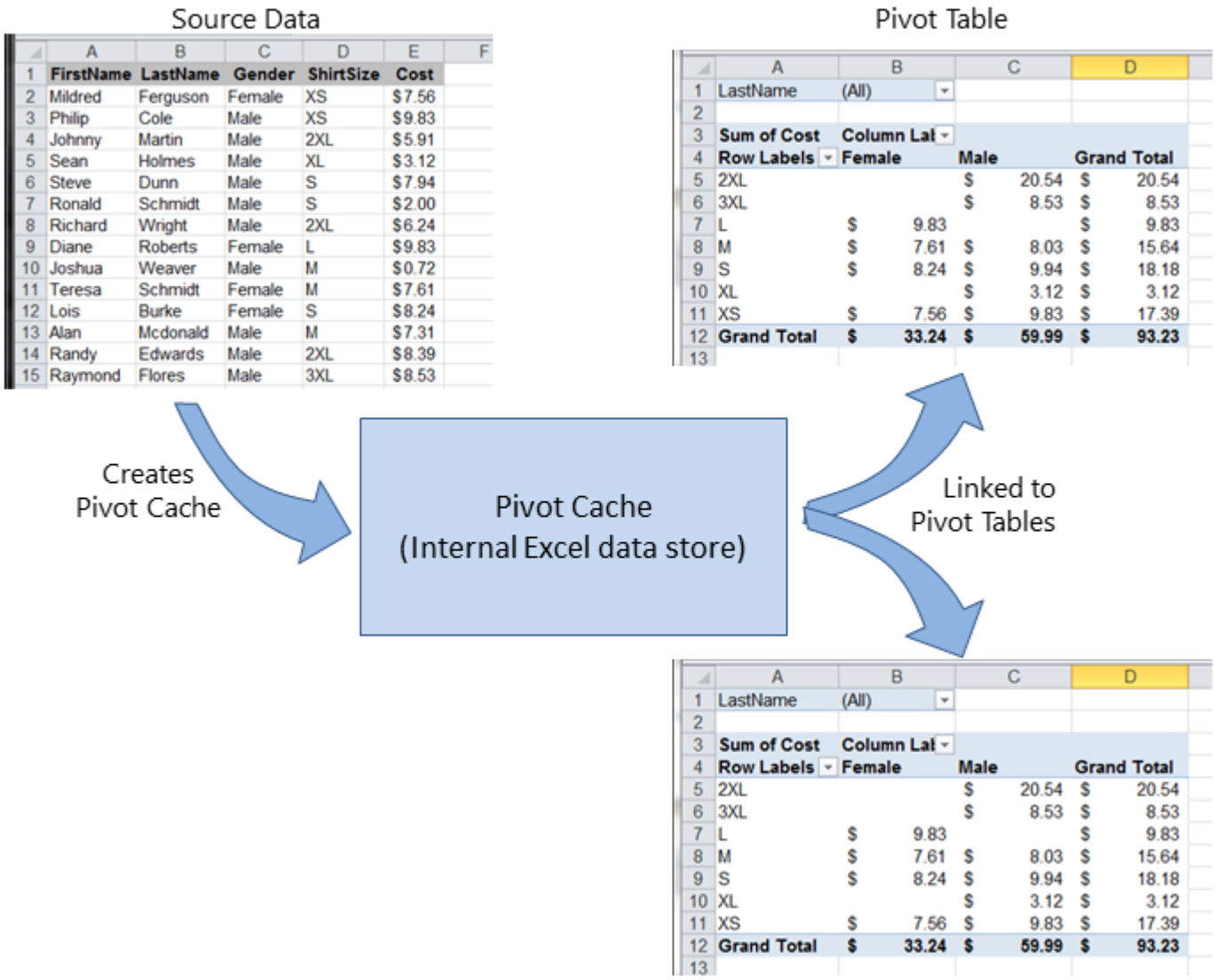
В самом основном, ваши исходные данные представляют собой область `Range` данных на `Worksheet` . Эта область данных **ДОЛЖНА** идентифицировать столбцы данных с строкой заголовка в качестве первой строки в диапазоне. После создания сводной таблицы пользователь может просматривать и изменять исходные данные в любое время. Однако изменения не могут быть автоматически или немедленно отражены в самой сводной таблице, поскольку существует промежуточная структура хранения данных, называемая сводным кэшем, которая напрямую связана с самой сводной таблицей.



	A	B	C	D	E
1	FirstName	LastName	Gender	ShirtSize	Cost
2	Mildred	Ferguson	Female	XS	$7.56
3	Philip	Cole	Male	XS	$9.83
4	Johnny	Martin	Male	2XL	$5.91
5	Sean	Holmes	Male	XL	$3.12
6	Steve	Dunn	Male	S	$7.94
7	Ronald	Schmidt	Male	S	$2.00
8	Richard	Wright	Male	2XL	$6.24
9	Diane	Roberts	Female	L	$9.83
10	Joshua	Weaver	Male	M	$0.72
11	Teresa	Schmidt	Female	M	$7.61
12	Lois	Burke	Female	S	$8.24
13	Alan	Mcdonald	Male	M	$7.31
14	Randy	Edwards	Male	2XL	$8.39
15	Raymond	Flores	Male	3XL	$8.53

	A	B	C	D
1	LastName	(All)		
2				
3	Sum of Cost	Column Lal		
4	Row Labels	Female	Male	Grand Total
5	2XL		$ 20.54	$ 20.54
6	3XL		$ 8.53	$ 8.53
7	L	$ 9.83		$ 9.83
8	M	$ 7.61	$ 8.03	$ 15.64
9	S	$ 8.24	$ 9.94	$ 18.18
10	XL		$ 3.12	$ 3.12
11	XS	$ 7.56	$ 9.83	$ 17.39
12	Grand Total	$ 33.24	$ 59.99	$ 93.23
13				

Если требуется несколько сводных таблиц, основанных на одних и тех же исходных данных, сводный кэш можно повторно использовать в качестве внутреннего хранилища данных для каждого из сводных таблиц. Это хорошая практика, поскольку она экономит память и уменьшает размер файла Excel для хранения.



Source Data

	A	B	C	D	E	F
1	FirstName	LastName	Gender	ShirtSize	Cost	
2	Mildred	Ferguson	Female	XS	$7.56	
3	Philip	Cole	Male	XS	$9.83	
4	Johnny	Martin	Male	2XL	$5.91	
5	Sean	Holmes	Male	XL	$3.12	
6	Steve	Dunn	Male	S	$7.94	
7	Ronald	Schmidt	Male	S	$2.00	
8	Richard	Wright	Male	2XL	$6.24	
9	Diane	Roberts	Female	L	$9.83	
10	Joshua	Weaver	Male	M	$0.72	
11	Teresa	Schmidt	Female	M	$7.61	
12	Lois	Burke	Female	S	$8.24	
13	Alan	Mcdonald	Male	M	$7.31	
14	Randy	Edwards	Male	2XL	$8.39	
15	Raymond	Flores	Male	3XL	$8.53	

Pivot Table

	A	B	C	D
1	LastName	(All)		
2				
3	Sum of Cost	Column Lal		
4	Row Labels	Female	Male	Grand Total
5	2XL		$ 20.54	$ 20.54
6	3XL		$ 8.53	$ 8.53
7	L	$ 9.83		$ 9.83
8	M	$ 7.61	$ 8.03	$ 15.64
9	S	$ 8.24	$ 9.94	$ 18.18
10	XL		$ 3.12	$ 3.12
11	XS	$ 7.56	$ 9.83	$ 17.39
12	Grand Total	$ 33.24	$ 59.99	$ 93.23
13				

	A	B	C	D
1	LastName	(All)		
2				
3	Sum of Cost	Column Lal		
4	Row Labels	Female	Male	Grand Total
5	2XL		$ 20.54	$ 20.54
6	3XL		$ 8.53	$ 8.53
7	L	$ 9.83		$ 9.83
8	M	$ 7.61	$ 8.03	$ 15.64
9	S	$ 8.24	$ 9.94	$ 18.18
10	XL		$ 3.12	$ 3.12
11	XS	$ 7.56	$ 9.83	$ 17.39
12	Grand Total	$ 33.24	$ 59.99	$ 93.23
13				

В качестве примера, чтобы создать сводную таблицу на основе исходных данных, показанных на приведенных выше рисунках:

```
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
```

Ссылки Объект Pivot Table MSDN

Сводные диапазоны таблиц

Эти превосходные справочные источники предоставляют описания и иллюстрации различных диапазонов в сводных таблицах.

Рекомендации

- Ссылка на диапазоны Pivot Table в VBA - из Технического блога Джона Пельтье
- Ссылка на таблицу сводных таблиц Excel с использованием VBA - из globaliconnect Excel VBA

Добавление полей в сводную таблицу

Две важные вещи, которые следует учитывать при добавлении полей в сводную таблицу, - это ориентация и позиция. Иногда разработчик может предположить, где находится поле, поэтому всегда яснее явно определять эти параметры. Эти действия влияют только на данную сводную таблицу, а не на сводный кеш.

```
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
```

Форматирование данных сводной таблицы

В этом примере изменяется / `DataBodyRange` несколько форматов в области диапазона данных (`DataBodyRange`) данной сводной таблицы. Доступны все форматируемые параметры в стандартном `Range` . Форматирование данных влияет только на сводную таблицу, а не на

сводный кеш.

ПРИМЕЧАНИЕ: свойство называется `TableStyle2` потому что свойство `TableStyle` не является членом свойств объекта `PivotTable` .

```
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
```

Прочитайте Сводные таблицы онлайн: https://riptutorial.com/ru/excel-vba/topic/3797/сводные-таблицы

глава 28: Советы и подсказки Excel VBA

замечания

Эта тема состоит из множества полезных советов и трюков, обнаруженных пользователями SO через их опыт в кодировании. Это часто примеры способов обхода общих разочарований или способов использования Excel более «умным» способом.

Examples

Использование xlVeryHidden Sheets

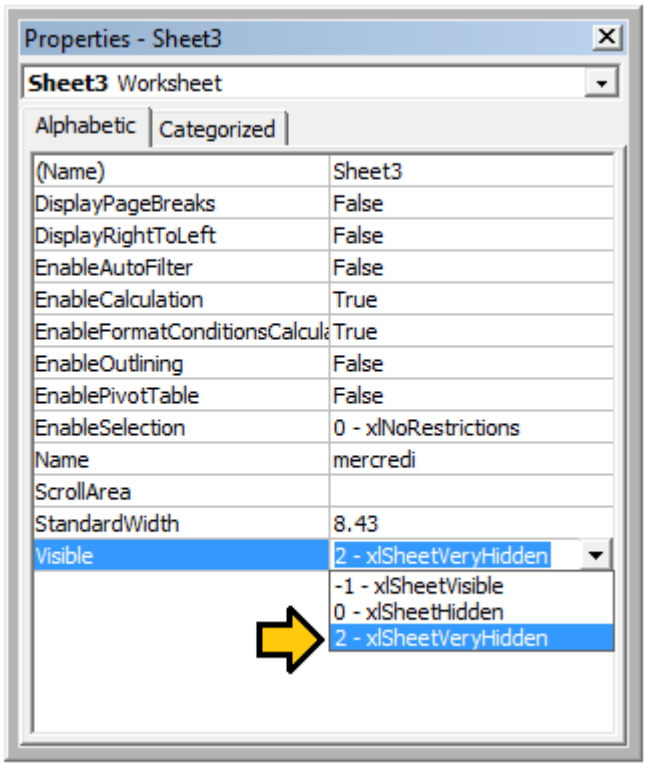
Рабочие листы в excel имеют три варианта свойства `Visible` . Эти параметры представлены константами в перечислении `xlSheetVisibility` и следующие:

1. `xlVisible` или `xlSheetVisible` значение: `-1` (по умолчанию для новых листов)
2. `xlHidden` или `xlSheetHidden` : `0`
3. `xlVeryHidden` `xlSheetVeryHidden` : `2`

Видимые листы представляют видимость по умолчанию для листов. Они видны на панели вкладок листов и могут быть свободно выбраны и просмотрены. Скрытые листы скрыты от панели вкладок листа и поэтому не могут быть выбраны. Однако скрытые листы могут быть скрыты из окна Excel, щелкнув правой кнопкой мыши на вкладках листа и выбрав «Unhide»,

Скрытые листы, с другой стороны, доступны *только* через редактор Visual Basic. Это делает их невероятно полезным инструментом для хранения данных через экземпляры excel, а также для хранения данных, которые должны быть скрыты от конечных пользователей. Листы могут быть доступны по именованной ссылке в коде VBA, что позволяет легко использовать сохраненные данные.

Чтобы вручную изменить свойство .Visible для xlSheetVeryHidden, откройте окно свойств VBE (`F4`), выберите рабочий лист, который вы хотите изменить, и используйте раскрывающийся список в тринадцатой строке, чтобы сделать свой выбор.



Чтобы изменить свойство .Visible рабочего листа на xlSheetVeryHidden[1] в коде, аналогичным образом получите доступ к свойству .Visible и назначьте новое значение.

```
with Sheet3
    .Visible = xlSheetVeryHidden
end with
```

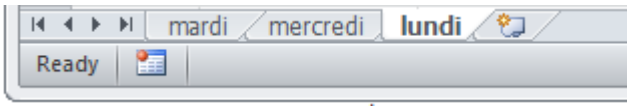
---

[1] Оба **xlVeryHidden** и **xlSheetVeryHidden** возвращают числовое значение **2** (они взаимозаменяемы).

Рабочий лист .Name, .Index или .CodeName

Мы знаем, что «лучшая практика» диктует, что объект диапазона должен иметь свой родительский лист, на который явно ссылаются. Рабочий лист может ссылаться на его свойство .Name, численное свойство .Index или его свойство .CodeName, но пользователь может изменить порядок очереди на рабочий лист, просто перетащив вкладку имен или переименуйте рабочий лист с двойным щелчком на той же вкладке, а некоторые набрав незащищенную книгу.

Рассмотрим стандартный три листа. Вы переименовали три листа в понедельник, вторник и среду в этом порядке и закодировали подпрограммы VBA, которые ссылаются на них. Теперь подумайте, что один пользователь приходит и решает, что понедельник принадлежит в конце очереди на листе, а затем другой приходит и решает, что имена рабочих листов лучше выглядят на французском языке. Теперь у вас есть рабочая книга с таблицей табуляции с названием листа, которая выглядит примерно так:



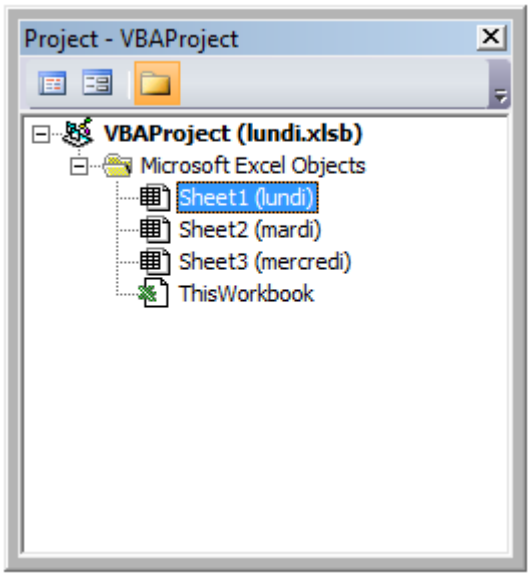
Если вы использовали какой-либо из следующих эталонных методов, ваш код теперь будет разбит.

```
'reference worksheet by .Name
with worksheets("Monday")
    '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
```

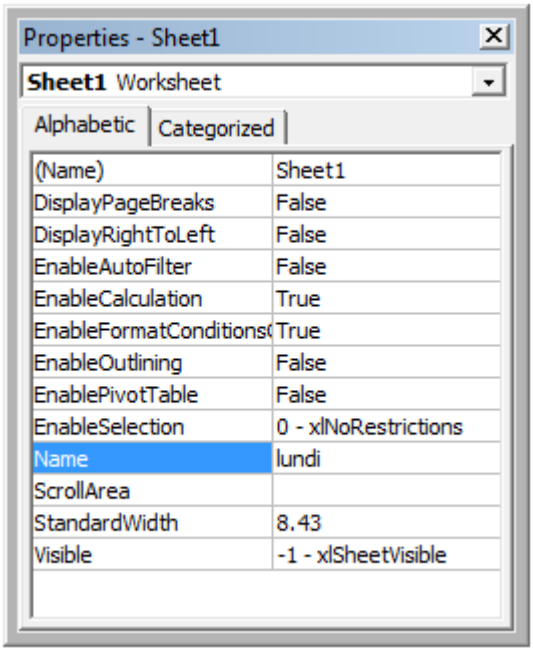
И первоначальный заказ, и имя исходного листа были скомпрометированы. Однако, если вы использовали свойство .CodeName рабочего листа, ваша вспомогательная процедура все равно будет работать

```
with Sheet1
    'operation code here; for example:
    .Range(.Cells(2, "A"), .Cells(.Rows.Count, "A").End(xlUp)) = 1
end with
```

На следующем рисунке показано окно проекта VBA ([Ctrl] + R), в котором перечислены рабочие листы по .CodeName, затем по .Name (в скобках). Приказ, который они отображают, не изменяется; ординал .Index берется по порядку, который они отображаются в очереди вкладок имен в окне рабочего листа.



Хотя это редко можно переименовать .CodeName, это не невозможно. Просто откройте окно свойств VBE ([F4]).



Рабочий лист .CodeName находится в первой строке. Имя рабочего листа находится в десятой части. Оба доступны для редактирования.

Использование строк с разделителями на месте динамических массивов

Использование динамических массивов в VBA может быть довольно неуклюжим и трудоемким по сравнению с очень большими наборами данных. При хранении простых типов данных в динамическом массиве (Strings, Numbers, Booleans и т. Д.) Можно избежать операторов `ReDim Preserve` необходимых для динамических массивов в VBA, используя функцию `Split()` с некоторыми умными строковыми процедурами. Например, мы рассмотрим цикл, который добавляет ряд значений из диапазона в строку на основе некоторых условий, а затем использует эту строку для заполнения значений 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
occurance of the delimiter within the string
    End If
Next Count

ListBox1.List = Split(DataString, Delimiter)

End Sub
```

Сама строка `Delimiter` может быть установлена на любое значение, но разумно выбрать значение, которое естественно не произойдет в наборе. Скажем, например, вы обрабатывали столбец дат. В этом случае, используя `.` , `-` , или `/` будет неразумным в

качестве разделителей, поскольку даты могут быть отформатированы для использования любого из них, генерируя больше точек данных, чем вы ожидали.

Примечание. Существуют ограничения на использование этого метода (а именно, максимальная длина строк), поэтому его следует использовать с осторожностью в случаях очень больших наборов данных. Это не обязательно самый быстрый или наиболее эффективный способ создания динамических массивов в VBA, но это жизнеспособная альтернатива.

Событие Double Click для форм Excel

По умолчанию в образцах Excel нет определенного способа обработки одиночных или двойных кликов, содержащих только свойство «OnAction», позволяющее обрабатывать клики. Однако могут быть случаи, когда ваш код требует от вас действовать по-разному (или исключительно) при двойном щелчке. Следующая подпрограмма может быть добавлена в ваш проект VBA и, когда она установлена в качестве процедуры `OnAction` для вашей фигуры, позволяет действовать двойным щелчком.

```
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 = CDbl(Timer)
    Else
        If CDbl(Timer) - LastClickTime > DOUBLECLICK_WAIT Then
            LastClickObj = Application.Caller
            LastClickTime = CDbl(Timer)
        Else
            If LastClickObj = Application.Caller Then
                'Your desired Double Click code here
                LastClickObj = ""
            Else
                LastClickObj = Application.Caller
                LastClickTime = CDbl(Timer)
            End If
        End If
    End If

End Sub
```

Эта процедура заставит форму функционально игнорировать первый клик, только запуск вашего желаемого кода на второй клик в течение указанного промежутка времени.

Открыть диалоговое окно «Файл» - несколько файлов

Эта подпрограмма - это быстрый пример того, как разрешить пользователю выбирать несколько файлов, а затем делать что-то с этими файловыми путями, например получить имена файлов и отправить их на консоль через 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
```

Прочитайте Советы и подсказки Excel VBA онлайн: https://riptutorial.com/ru/excel-vba/topic/2240/советы-и-подсказки-excel-vba

глава 29: Создание выпадающего меню в Active Worksheet с помощью Combo Box

Вступление

Это простой пример, демонстрирующий, как создать раскрывающееся меню в Активном листе вашей книги, вставив в лист объект ActiveX ActiveX. Вы сможете вставить одну из пяти песен Jimi Hendrix в любую активированную ячейку листа и сможете ее очистить соответственно.

Examples

Меню пользователя Jimi Hendrix

В общем, код помещается в модуль листа.

Это событие Worksheet_SelectionChange, которое срабатывает каждый раз, когда на активном листе выбрана другая ячейка. Вы можете выбрать «Рабочий лист» из первого раскрывающегося меню над окном кода и «Selection_Change» в раскрывающемся меню рядом с ним. В этом случае каждый раз, когда вы активируете ячейку, код перенаправляется на код Combo Box.

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

   ComboBox1_Change

End Sub
```

Здесь подпрограмма, посвященная ComboBox, по умолчанию кодируется для события Change. В нем есть фиксированный массив, заполненный всеми параметрами. Не опция CLEAR в последней позиции, которая будет использоваться для очистки содержимого ячейки. Затем массив передается в Combo Box и переходит к рутине, которая выполняет эту работу.

```
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
```

Массив передается в подпрограмму, которая заполняет ячейки именем песни или нулевым значением для их удаления. Во-первых, целочисленной переменной присваивается значение позиции выбора, которую делает пользователь. Затем Combo Box перемещается в верхний левый угол ячейки, который пользователь активирует, и его размеры настраиваются так, чтобы сделать его более жидким. Затем активной ячейке присваивается значение в позиции в целочисленной переменной, которая отслеживает выбор пользователя. Если пользователь выбирает CLEAR из опций, ячейка освобождается.

Вся процедура повторяется для каждой выбранной ячейки.

```
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
```

Пример 2: варианты не включены

Этот пример используется при указании параметров, которые не могут быть включены в базу данных доступного жилья и сопутствующих им услуг.

Он основывается на предыдущем примере с некоторыми отличиями:

1. Две процедуры больше не нужны для одного комбинированного блока, который выполняется путем объединения кода в одну процедуру.
2. Использование свойства LinkedCell позволяет каждый раз правильно вводить пользовательский выбор
3. Включение функции резервного копирования для обеспечения активной ячейки находится в правильном столбце и кодеке предотвращения ошибок на основе предыдущего опыта, где числовые значения будут отформатированы как строки при

заполнении активной ячейки.

```
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

            '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
```

Вышеуказанный макрос запускается каждый раз, когда ячейка активируется событием SelectionChange в модуле рабочего листа:

```
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
```

Прочитайте Создание выпадающего меню в Active Worksheet с помощью Combo Box онлайн: https://riptutorial.com/ru/excel-vba/topic/8929/создание-выпадающего-меню-в-active-worksheet-с-помощью-combo-box

глава 30: Условное форматирование с использованием VBA

замечания

Вы не можете определить более трех условных форматов для диапазона. Используйте метод Modify для изменения существующего условного формата или используйте метод Delete для удаления существующего формата перед добавлением нового.

Examples

FormatConditions.Add

Синтаксис:

```
FormatConditions.Add(Type, Operator, Formula1, Formula2)
```

Параметры:

название	Обязательный / необязательный	Тип данных
Тип	необходимые	XlFormatConditionType
оператор	Необязательный	Вариант
Формула 1	Необязательный	Вариант
Formula2	Необязательный	Вариант

XlFormatConditionType enumaration:

название	Описание
xlAboveAverageCondition	Выше среднего условия
xlBlanksCondition	Состояние бланков
xlCellValue	Значение ячейки

название	Описание
xlColorScale	Цветовая гамма
xlDatabar	Databar
xlErrorsCondition	Условие ошибок
xlExpression	выражение
XlIconSet	Набор значков
xlNoBlanksCondition	Отсутствие состояния пробелов
xlNoErrorsCondition	Нет ошибок.
xlTextString	Текстовая строка
xlTimePeriod	Временной период
xlTop10	Топ-10 значений
xlUniqueValues	Уникальные значения

Форматирование по значению ячейки:

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

Операторы:

название
xlBetween
xlEqual
xlGreater
xlGreaterEqual
xlLess
xlLessEqual

название
xlNotBetween
xlNotEqual

Если Type является выражением xlExpression, аргумент Operator игнорируется.

Форматирование по тексту содержит:

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

Операторы:

название	Описание
xlBeginsWith	Начинается с указанного значения.
xlContains	Содержит указанное значение.
xlDoesNotContain	Не содержит указанного значения.
xlEndsWith	Завершить указанное значение

Форматирование по времени

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

Операторы:

название
xlYesterday

название
xlTomorrow
xlLast7Days
xlLastWeek
xlThisWeek
xlNextWeek
xlLastMonth
xlThisMonth
xlNextMonth

Удалить условный формат

Удалите все условные форматы в диапазоне:

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

Удалите все условные форматы на листе:

```
Cells.FormatConditions.Delete
```

FormatConditions.AddUniqueValues

Выделение повторяющихся значений

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

Выделение уникальных значений

```
With Range("E1:E100").FormatConditions.AddUniqueValues
   With .Font
```

```
        .Bold = True
        .ColorIndex = 3
    End With
End With
```

FormatConditions.AddTop10

Выделение верхних 5 значений

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

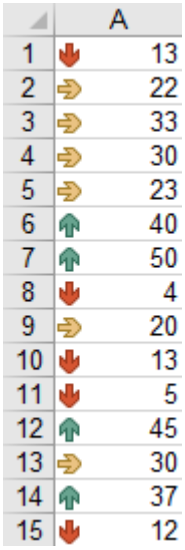
FormatConditions.AddAboveAverage

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

Операторы:

название	Описание
XlAboveAverage	Выше среднего
XlAboveStdDev	Выше стандартного отклонения
XlBelowAverage	Ниже среднего
XlBelowStdDev	Ниже стандартного отклонения
XlEqualAboveAverage	Равно выше среднего
XlEqualBelowAverage	Равновесие ниже среднего

FormatConditions.AddIconSetCondition



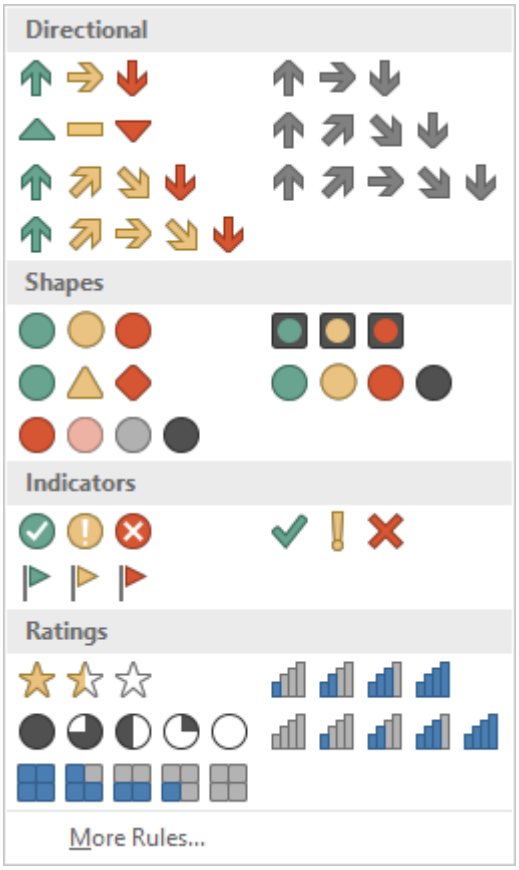
	A
1	13
2	22
3	33
4	30
5	23
6	40
7	50
8	4
9	20
10	13
11	5
12	45
13	30
14	37
15	12

```
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
```

IconSet:

название
xl3Arrows
xl3ArrowsGray
xl3Flags
xl3Signs
xl3Stars
xl3Symbols
xl3Symbols2
xl3TrafficLights1

название
xl3TrafficLights2
xl3Triangles
xl4Arrows
xl4ArrowsGray
xl4CRV
xl4RedToBlack
xl4TrafficLights
xl5Arrows
xl5ArrowsGray
xl5Boxes
xl5CRV
xl5Quarters



Тип:

название
xlConditionValuePercent
xlConditionValueNumber
xlConditionValuePercentile
xlConditionValueFormula

Оператор:

название	Значение
xlGreater	5
xlGreaterEqual	7

Значение:

Возвращает или устанавливает пороговое значение для значка в условном формате.

Прочитайте Условное форматирование с использованием VBA онлайн: https://riptutorial.com/ru/excel-vba/topic/9912/условное-форматирование-с-использованием-vba

глава 31: Условные утверждения

Examples

Утверждение If

Оператор `If` позволяет использовать другой код в зависимости от оценки условного (логического) оператора. Условным выражением является выражение, которое оценивает либо `True` либо `False` , например `x > 2` .

Существует три шаблона, которые можно использовать при реализации оператора `If` , которые описаны ниже. Обратите внимание , что `If` условная оценка всегда следует `Then` .

1. Оценка одного `If` условное утверждение и что-то делает, если оно `True`

Одиночная строка `If` оператор

Это самый короткий способ использования « `If` и это полезно, когда только одно утверждение должно выполняться при оценке `True` . При использовании этого синтаксиса весь код должен находиться в одной строке. Не включайте `End If` в конце строки.

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

`If` блок

Если несколько строк кода необходимо выполнить при оценке `True` , может использоваться блок `If` .

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

Обратите внимание, что если используется многострочный блок `If` , требуется соответствующий `End If` .

2. Оценка одного условного оператора `If` , делая одно, если оно `True` и делает что-то другое, если оно `False`

Одиночная строка `If` , `Else` statement

Это можно использовать, если один оператор должен выполняться при оценке `True` и другое утверждение должно выполняться при `False` оценке. Будьте осторожны с использованием этого синтаксиса, так как для читателей зачастую неясно, что есть инструкция `Else` . При использовании этого синтаксиса весь код должен находиться в одной строке. Не включайте `End If` в конце строки.

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

`If` , `Else` block

Используйте блок `If` , `Else` чтобы добавить ясность в ваш код, или если несколько строк кода должны быть выполнены либо при проверке `True` либо в `False` .

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

Обратите внимание, что если используется многострочный блок `If` , требуется соответствующий `End If` .

3. Оценка многих условных операторов, когда предыдущие утверждения являются `False` и делают что-то другое для каждого

Этот шаблон является наиболее общим использованием `If` и будет использоваться, когда существует много неперекрывающихся условий, требующих различного лечения. В отличие от первых двух шаблонов, этот случай требует использования блока `If` , даже если для каждого условия будет выполняться только одна строка кода.

`If` , `ElseIf` , `...` , `Else` block

Вместо того, чтобы создавать много блоков `If` ниже, может использоваться `ElseIf` чтобы оценить дополнительное условие. `ElseIf` оценивается только в том случае, если предшествует. `If` оценка `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
```

Поскольку многие управляющие операторы `ElseIf` могут быть включены между `If` и `End If` мере необходимости. Элемент управления `Else` не требуется при использовании `ElseIf` (хотя это рекомендуется), но если он включен, он должен быть окончательным контрольным заявлением до `End If` .

Прочитайте Условные утверждения онлайн: https://riptutorial.com/ru/excel-vba/topic/9632/условные-утверждения

кредиты

S. No	Главы	Contributors
1	Начало работы с excel-vba	Branislav Kollár, chris neilsen, Cody G., Comintern, Community, Doug Coats, EEM, Gordon Bell, Jeeped, Joel Spolsky, Kaz, Laurel, LucyMarieJ, Macro Man, Malick, Maxime Porté, Regis, RGA, Ron McMahon, SandPiper, Shai Rado, Taylor Ostberg, whytheq
2	CustomDocumentProperties на практике	T.M.
3	SQL в Excel VBA - лучшие практики	Zsmaster
4	Workbooks	PeterT
5	автофильтр; Использование и передовая практика	Sgdva
6	Безопасность VBA	Chel, TheGuyThatDoesn'tKnowMuch
7	Графики и диаграммы	Byron Wall
8	Диапазоны и ячейки	Adam, Branislav Kollár, Doug Coats, Gregor y, Jbjstam, Joel Spolsky, Julian Kuchlbauer, Máté Juhász, Miguel_Ryu, Patrick Wynne, Vegard
9	Именованные диапазоны	Andre Terra, Portland Runner
10	Интеграция PowerPoint через VBA	mnoronha, RGA
11	Использовать объект Worksheet, а не объект Sheet	Vityata
12	Как записать макрос	Mike, Robby
13	Лучшие практики VBA	Alexis Olson, Branislav Kollár, Chel, Cody G., Comintern, EEM, FreeMan, genespos, Hubisan, Huzaifa Essajee, Jeeped, JKAbrams, Kumar Sourav, Kyle, Macro Man, Malick, Máté Juhász, Munkeeface, paul bica, Peh,

		PeterT, Portland Runner, RGA, Shai Rado, Stefan Pinnow, Steven Schroeder, Taylor Ostberg, ThunderFrame, Verzweifler, Vityata
14	Массивы	Alon Adler, Hubisan, Miguel_Ryu, Shahin
15	Методы поиска последней использованной строки или столбца на листе	curious, Hubisan, Máté Juhász, Michael Russo, Miqi180, paul bica, R3uK, Raystafarian, RGA, Shai Rado, Slai, Thomas Inzina, YowE3K
16	Объединенные ячейки / диапазоны	R3uK
17	Объект приложения	Captain Grumpy, Joel Spolsky
18	Объект файловой системы	Zsmaster
19	Оптимизация Excel-VBA	Masoud, paul bica, T.M.
20	Отладка и устранение неполадок	Cody G., Etheur, Gregor y, Julian Kuchlbauer, Kyle, Malick, Michael Russo, RGA, Ron McMahon, Slai, Steven Schroeder, Taylor Ostberg
21	переплет	Captain Grumpy, EEM, Jeeped, jlookup, Malick, Raystafarian
22	Поиск повторяющихся значений в диапазоне	quadrature, T.M.
23	Пользовательские функции (UDF)	Jeeped, Malick, Slai, user3561813, Vegard
24	Прокрутка всех листов в активной рабочей книге	Doug Coats, Shai Rado
25	Работа с таблицами Excel в VBA	Excel Developers
26	Распространенные ошибки	Egan Wolf, Gordon Bell, Macro Man, Malick, Peh, SWa, Taylor Ostberg
27	Сводные таблицы	PeterT
28	Советы и подсказки Excel VBA	Andre Terra, Cody G., Jeeped, Kumar Sourav, Macro Man, RGA
29	Создание выпадающего	Macro Man, quadrature, R3uK

	меню в Active Worksheet с помощью Combo Box	
30	Условное форматирование с использованием VBA	Zsmaster
31	Условные утверждения	SteveES