Майк МакГрат

EXCEL VBA

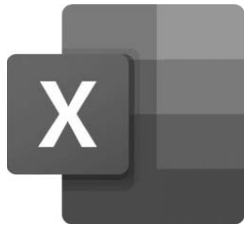
СТАНЬ ПРОДВИНУТЫМ ПОЛЬЗОВАТЕЛЕМ ЗА НЕДЕЛЮ

АВТОМАТИЗИРУЙТЕ ЗАДАЧИ
•
СОЗДАВАЙТЕ МАКРОСЫ
•
УЧИТЕСЬ НА ПОНЯТНЫХ ПРИМЕРАХ

3-е издание

ПРОСТОЙ САМОУЧИТЕЛЬ ПО СОЗДАНИЮ МАКРОСОВ

БОМБОРА
ИЗДАТЕЛЬСТВО



Mike McGrath

EXCEL VBA

IN EASY STEPS

3rd Edition

БОМБОРА
ИЗДАТЕЛЬСТВО
Москва 2022

Майк МакГрат

EXCEL VBA

СТАНЬ ПРОДВИНУТЫМ ПОЛЬЗОВАТЕЛЕМ ЗА НЕДЕЛЮ

3-е издание

БОМБОРА
ИЗДАТЕЛЬСТВО
Москва 2022

УДК 004.67
ББК 32.973.26-018.2
М15

Mike McGrath
Excel VBA in Easy Steps

МакГрат, Майк.

М15 Excel VBA. Стань продвинутым пользователем за неделю / Майк МакГрат ; [перевод с английского М.А. Райтмана]. — Москва : Эксмо, 2022. — 240 с. : ил. — (Excel для всех).

ISBN 978-5-04-121944-4

Пошаговый самоучитель по языку VBA, при помощи которого создаются макросы для Excel, поможет вам стать продвинутым пользователем и повысить свою эффективность работы в этой программе в несколько раз. Книга снабжена множеством иллюстраций, а вся теория объясняется на доступных даже для полных новичков примерах. Внутри вы найдете полезные советы, предостережения и сможете скачать архив с бесплатными примерами для работы с ними на компьютере.

УДК 004.67
ББК 32.973.26-018.2

ISBN 978-5-04-121944-4

Оглавление

1 Первые шаги

Добро пожаловать в увлекательный мир Excel VBA (Visual Basic for Applications). В этой главе вы узнаете, как создать макрос VBA для книг Excel.

Знакомство с Excel VBA

Visual Basic for Applications (VBA) — это язык программирования, встроенный в электронные таблицы Excel и остальные продукты Microsoft Office. С помощью VBA можно решить множество задач, с которыми не справятся стандартные инструменты Excel. Также VBA автоматизирует многие рутинные процессы.

Если вы только начинаете работу с Excel, ознакомьтесь с нашей книгой «Excel 2019 in easy steps».

Мы рекомендуем заниматься по этой книге продвинутым пользователям Excel, которые умеют:

- создавать книги и добавлять листы;
- искать материалы в книге и на листе;
- использовать ленточный интерфейс;
- именовать ячейки и диапазоны;
- использовать функции рабочего листа.

Для демонстрации примеров использовалась версия программы Excel 2019. Многие из них также актуальны и для более ранних версий Excel.

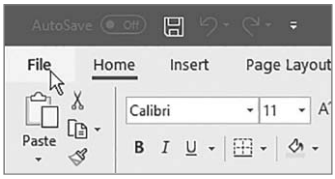
Все примеры из этой книги вы можете бесплатно скачать по ссылке: http://addons.eksmo.ru/it/excelvba_examples.zip/

Включение VBA

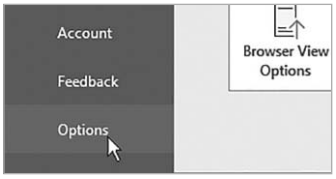
Перед началом работы вам нужно включить VBA в настройках Excel:

1. Откройте Excel и выберите пункт Blank workbook (Пустая книга).

2 Когда книга откроется, нажмите вкладку **File** (Файл) на ленточном интерфейсе.

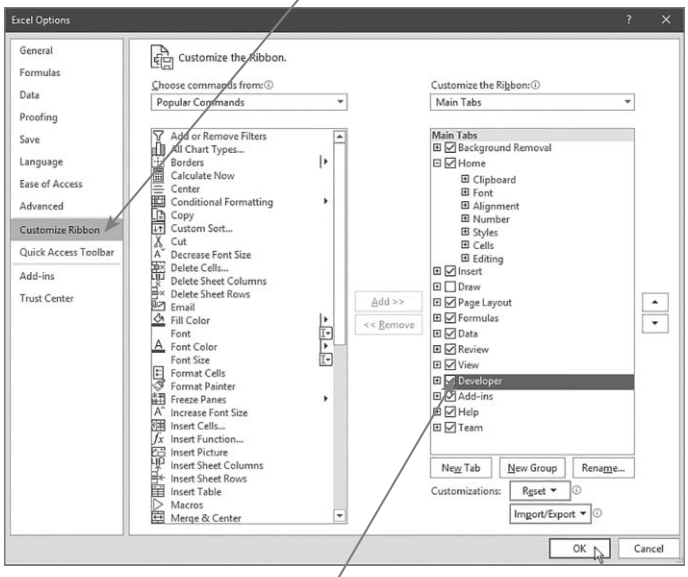


3 Выберите строку **Options** (Параметры). Откроется диалоговое окно **Excel Options** (Параметры Excel).



Вы можете также открыть диалоговое окно, нажав сочетание клавиш **Alt+F+T.**

4 Выберите пункт **Customize Ribbon** (Настроить ленту) в левом столбце.

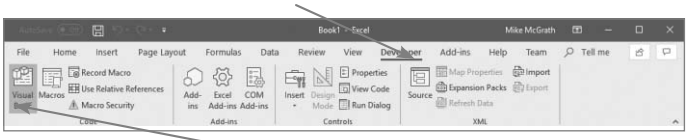


5 Проверьте, включено ли отображение вкладки **Developer** (Разработчик) в правом столбце.

6 Нажмите кнопку **OK**, чтобы принять изменения и закрыть диалоговое окно.

В диалоговом окне параметров напротив строки **Developer** (Разработчик) вы можете щелкнуть мышью по значку + и раскрыть группу элементов. Если вы щелкнете правой кнопкой мыши по любой из групп элементов, контекстное меню предложит вам параметры для изменения групп, которые будут отображаться на панели **Developer** (Разработчик).

7 Теперь панель **Developer** (Разработчик) отображается на ленточной панели.

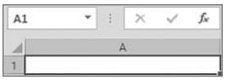


8 Перейдите на вкладку **Developer** (Разработчик). В группе элементов **Code** (Код) вы увидите кнопку **Visual Basic** — теперь VBA активирован.

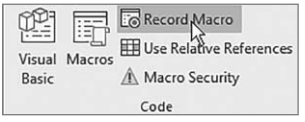
Запись макроса

Активировав VBA способом, указанным в предыдущем разделе, вы можете создать простое приложение посредством записи«макроса» для сохранения действий:

1 В Excel откройте пустую книгу, затем выберете ячейку A1.



2 На панели **Developer** (Разработчик) нажмите кнопку **Record Macro** (Запись макроса) в группе элементов **Code** (Код). Откроется диалоговое окно **Record Macro** (Запись макроса).

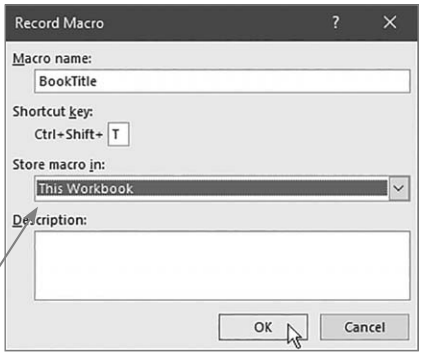


В диалоговом окне **Record Macro** (Запись макроса) вы можете добавить описание макроса.

3 Напишите любое название в поле **Macro name** (Имя макроса), например, **BookTitle**.

Макрос — это набор инструкций программирования, которые хранятся в коде VBA.

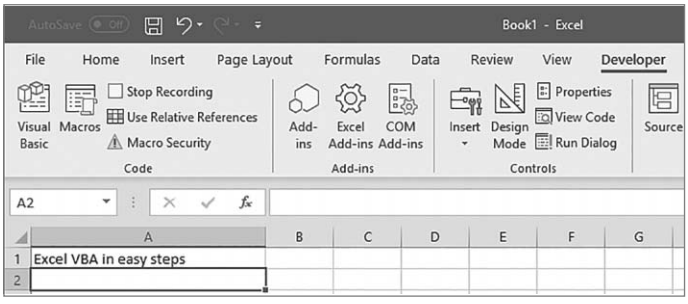
4 Затем напишите букву в поле **Shortcut key** (Сочетание клавиш), например, **T**. Таким образом вы назначите макросу сочетание клавиш **Ctrl + Shift + T**.



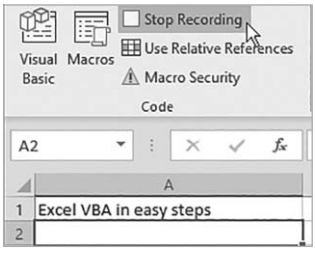
5 В раскрывающемся списке ниже выберите пункт **This Workbook** (Эта книга).

6 Нажмите клавишу **OK**, чтобы закрыть диалоговое окно, и начните запись действий.

7 Напечатайте название этой книги в ранее выбранную ячейку A1, затем нажмите клавишу **Enter**, чтобы сохранить название в ячейке.



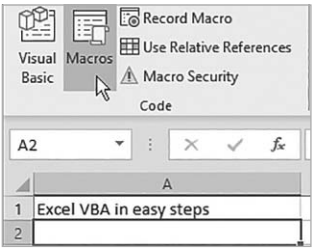
8 Обратите внимание, что фокус сместился, — после того, как вы нажали клавишу **Enter**, ячейка A2 была выбрана автоматически.

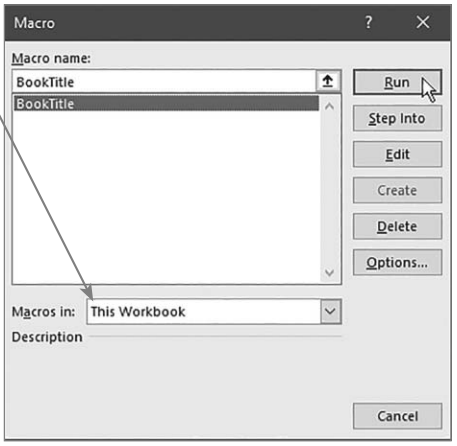


Когда вы начнете запись, кнопка **Record Macro** (Запись макроса) изменится на **Stop Recording** (Остановить запись).

9 Теперь нажмите кнопку **Stop Recording** (Остановить запись), чтобы прекратить запись действий.

10 Нажмите кнопку **Macros** (Макросы) в группе элементов **Code** (Код). Откроется диалоговое окно **Macro** (Макросы), в котором перечислены макросы, сохраненные в **This Workbook** (Эта книга).





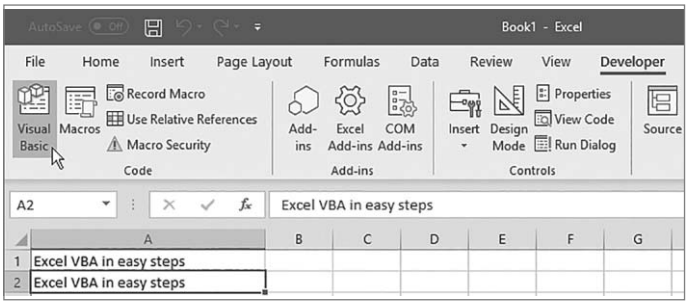
11 Выберите макрос **BookTitle**, затем нажмите кнопку **Run** (Выполнить) для выполнения макроса. Название книги автоматически появится в ячейке **A2**.

Чтобы открыть диалоговое окно макроса, вы можете использовать сочетание клавиш **Alt + F8**.

Просмотр кода макроса

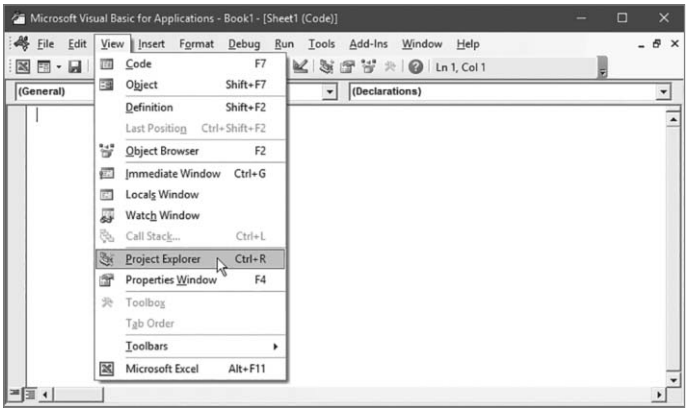
Создав макрос способом, описанным ранее, вы можете просмотреть инструкции программирования VBA в редакторе Visual Basic:

1 Чтобы запустить редактор Visual Basic, нажмите кнопку **Visual Basic** на панели **Developer** (Разработчик).



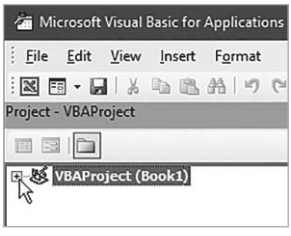
Для открытия редактора Visual Basic вы можете воспользоваться сочетанием клавиш **Alt + F11**.

2 В редакторе Visual Basic выберите команду меню **View ⇨ Project Explorer** (Вид ⇨ Обозреватель проектов), чтобы открыть соответствующую панель.

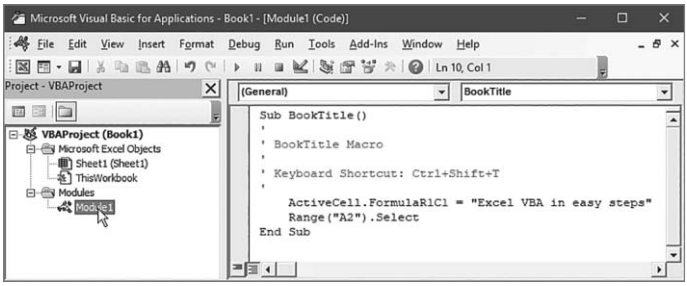


При открытии редактора Visual Basic панель **Project Explorer** (Обозреватель проектов) уже может быть открыта, однако пользователю полезно самому научиться открывать и закрывать эти элементы, чтобы полностью ознакомиться с интерфейсом редактора Visual Basic.

3 Для просмотра группы элементов на панели **Project Explorer** (Обозреватель проектов) нажмите кнопку **+** возле строки **Book1**.



4 Чтобы увидеть код макроса, на панели **Project Explorer** (Обозреватель проектов) в папке **Modules** дважды щелкните мышью по пункту **Module1**.



В круглых скобках **()** в первой строке кода должны быть написаны параметры. Больше информации см. в главе 6.

Анализ программного кода

- **Sub BookTitle()** указывает на начало подпрограммы (**Sub**) с именем как у макроса (**BookTitle**), который вы именовали перед началом записи.
- **BookTitle Macro** — примечание, которое означает, что эта подпрограмма была создана для макроса с указанным названием.
- **Keyboard Shortcut: Ctrl+Shift+T** — еще одно примечание, описывающее выбранное вами сочетание клавиш для выполнения макроса.
- **ActiveCell.FormulaR1C1 = "Excel VBA in easy steps"** — эта инструкция была написана в то время, когда вы вводили название книги в ячейку и нажимали клавишу **Enter**.
- **Range("A2").Select** — эта инструкция была написана в тот момент, когда фокус сместился на ячейку A2.
- **End Sub** означает конец подпрограммы. Эта строка была записана в тот момент, когда вы закончили запись макроса.

Все строки, начинающиеся с апострофа, при выполнении макроса игнорируются.

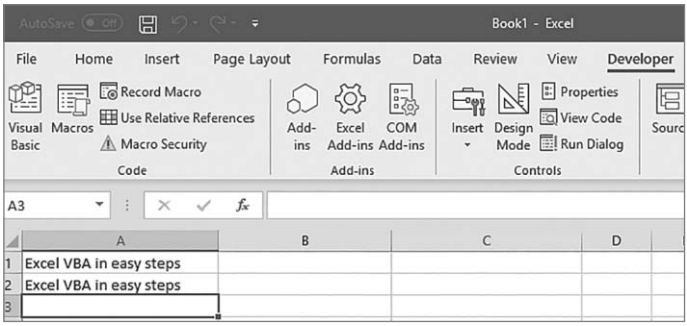
Использование различных цветов в коде применяется для выделения синтаксических элементов и облегчения чтения кода. Редактор Visual Basic автоматически активирует эту функцию. Синим цветом выделяются «ключевые слова», которые имеют важное значение в коде, а зеленым — комментарии, описывающие код. Для удобства читателя эти цвета используются во всех примерах в этой книге.

Тестирование макроса

Прежде чем записывать макрос способом, описанным ранее, пользователь в диалоговом окне **Record Macro** (Запись макроса) указывает сочетания клавиш. Давайте проверим, как таким образом можно запустить макрос:

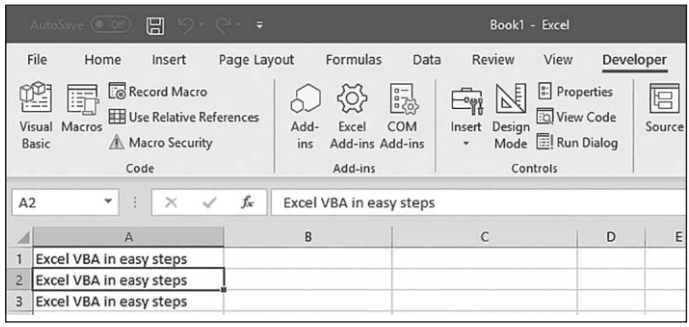
1. Открыв редактор Visual Basic, выберите команду меню **View ⇨ Microsoft Excel** (Вид ⇨ Microsoft Excel) или щелкните мышью по значку ⊠, чтобы вернуться к интерфейсу Excel.

2. Выберите пустую ячейку **A3**.



3. Теперь нажмите сочетание клавиш **Ctrl + Shift + T**, чтобы попробовать выполнить макрос. В выбранной ячейке должно появиться название книги, а фокус вернется на ячейку **A2**.

Для закрытия редактора Visual Basic вы можете использовать сочетание клавиш **Alt + F11**.



Если вы попытаетесь использовать сочетание клавиш, которое уже используется для другого макроса, то появится диалоговое окно с просьбой указать другое сочетание клавиш, чтобы не было совпадений.

Важно помнить, что ячейка **A1** была выбрана до записи макроса. Если вы выберете нужную ячейку уже после начала записи, то программа примет это действие за «инструкцию», и при каждом запуске макроса название книги будет записываться только в ячейку **A1**.

Изменение кода макроса

Теперь вы знаете, что можете запустить макрос и с помощью кнопки **Run** (Выполнить) из диалогового окна, и с помощью сочетания клавиш **Ctrl + Shift + T**. Но, скорее всего, вам не нужно, чтобы после каждого выполненного макроса фокус возвращался на ячейку **A2**. Вы можете изменить код, чтобы удалить инструкции для фокуса, а заодно изменить стиль шрифта:

При написании инструкции в случае ввода знака разделителя или точки будет появляться всплывающее окно со списком предложений, где вы можете выбрать нужный элемент.

1. На панели **Developer** (Разработчик) в группе элементов **Code** (Код) нажмите кнопку **Visual Basic**. Появится окно редактора Visual Basic.

2. На панели **Project Explorer** (Обозреватель проектов) дважды щелкните мышью по пункту **Module1**, чтобы увидеть код макроса VBA.

3. Удалите строку, которая отвечает за возвращение фокуса.

```
Range("A2").Select
```

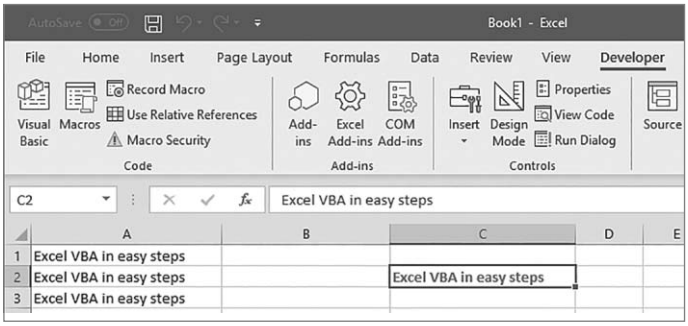
4 Добавьте следующие инструкции в любое место кода, чтобы изменить цвет надписи на полужирный красный:

```
ActiveCell.Font.Bold = True
ActiveCell.Font.Color = vbRed
```

5 Нажмите кнопку **Save** (Сохранить) ( ).

```
(General)                                   BookTitle
Sub BookTitle()
'
' BookTitle Macro
'
' Keyboard Shortcut: Ctrl+Shift+T
'
    ActiveCell.Font.Bold = True
    ActiveCell.Font.Color = vbRed

    ActiveCell.FormulaR1C1 = "Excel VBA in easy steps"

End Sub
```

6 Вернитесь в Excel и выберите любую ячейку, затем нажмите сочетание клавиш **Ctrl + Shift + T** для выполнения измененного макроса.



В Visual Basic есть восемь цветовых констант: **vbRed**, **vbGreen**, **vbBlue**, **vbYellow**, **vbMagenta**, **vbCyan**, **vbBlack** и **vbWhite**. Больше информации о них ищите в главе 3.

Несмотря на то что строки в коде VBA имеют строгий порядок сверху вниз, в этом макросе их порядок не важен. Вы можете изменить оформление ячейки до или после добавления текста.

Ссылки в Excel

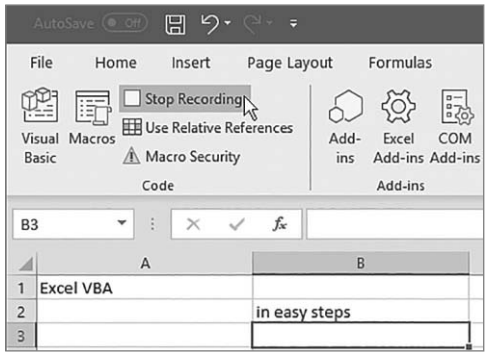
В Excel существуют два варианта записи макросов, которые отличаются способом обращения к ячейке. Режим записи, установленный по умолчанию и использованный в предыдущих примерах, обращается к ячейкам по их абсолютному положению на листе — например, ячейка **A1**, **A2**, **A3** и т. д. Другой способ записи макросов предполагает рассмотрение положения ячеек относительно остальных, при этом происходит смещение на определенное количество

Макросы, записанные с помощью относительных ссылок, более гибкие, потому что их можно использовать в любом месте книги.

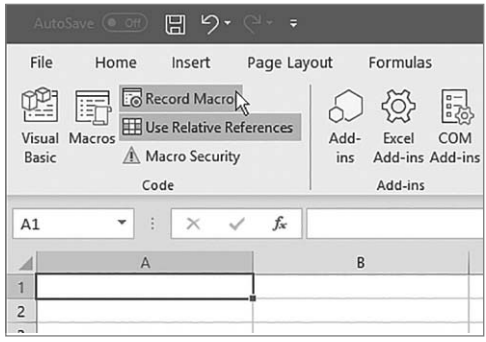
строк и столбцов относительно другой ячейки. Разница между этими способами существенна: макросы, в которых используются абсолютные ссылки, всегда ссылаются на одно и то же расположение вне зависимости от расположения выбранной ячейки. Макросы, в которых используются относительные ссылки, обращаются к ячейке, которая смещена на определенное количество столбцов и строк от выбранной ячейки.

1. Очистите все ячейки на листе, выберите ячейку **A1** и выполните макрос под названием **AbsoluteBookTitle**.

2. Напишите название этой книги, затем выберите ячейку **B2** и напишите название серии книг.



3. Нажмите клавишу **Enter**, затем нажмите кнопку **Stop Recording** (Остановить запись).

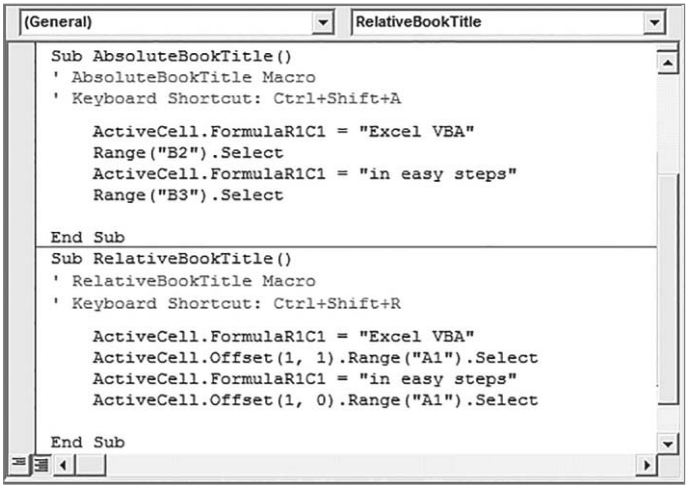


Для запуска этих макросов также можно использовать сочетание клавиш: например, **Ctrl + Shift + A** (Абсолютный) и **Ctrl + Shift + R** (Относительный).

4. Очистите все ячейки на листе, выберите ячейку **A1** и нажмите кнопку **Use Relative References** (Относительные ссылки) в группе элементов **Code** (Код).

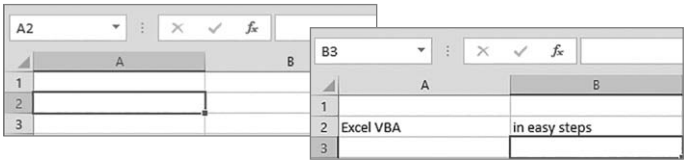
5. Начните макрос под названием **RelativeBookTitle** и повторите шаги *2*–3 для выполнения макроса.

6. Нажмите кнопку **Visual Basic**, чтобы открыть редактор. Сравните код каждого макроса.



При выборе ячейки **B2** абсолютная ссылка ссылается на ячейку по ее имени. При этом относительная ссылка ссылается на ячейку как на смещение на одну строку и один столбец от выбранной ячейки. Для сравнения:

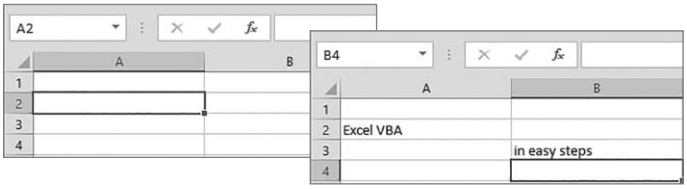
7. Очистите все ячейки, выберите ячейку **A2** и запустите макрос под названием **AbsoluteBookTitle**.



В целях экономии места в книге на этой иллюстрации пустые строки были удалены.

В этом примере макрос, который использует абсолютную ссылку на ячейку, записывает название серии книг в ячейку **B2**, в то время как макрос, использующий относительную ссылку, записывает название серии книг в ячейку **B3**, поскольку она смещена на одну строку и один столбец от изначально выбранной ячейки.

8 Снова очистите все ячейки, выберите ячейку **A2** и запустите макрос под названием **RelativeBookTitle**.

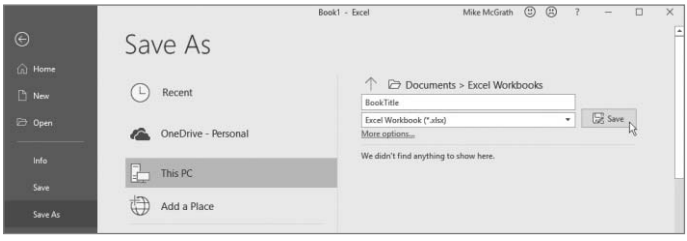


Сохранение макросов

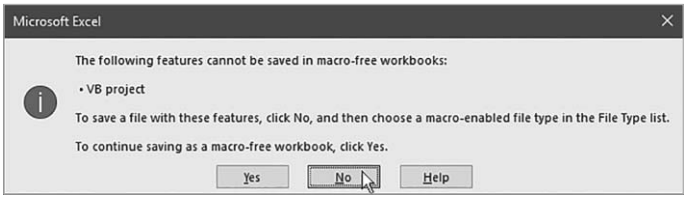
Начиная с версии Excel 2007, все книги имеют стандартный тип файла *.xlsx*, однако такие файлы не могут содержать макросы Visual Basic. Для того чтобы сохранить книгу со всеми компонентами, необходимо сохранить ее в формате **Excel Macro-Enabled Workbook** (Книга Excel с поддержкой макросов), тип файла — *.xlsm*. Если вы решите сохранить книгу с макросом в формате *.xlsx*, то получите предупреждение от Excel, что в данном случае код макроса будет утерян:

Выберите расположение папки, куда бы вы хотели сохранить книгу. В нашем примере папка под названием *Excel Workbooks* расположена в папке с документами пользователя.

1 В Excel выберите команду меню **File ⇨ Save As** (Файл ⇨ Сохранить как), напишите название книги **BookTitle** и нажмите кнопку **Save** (Сохранить).

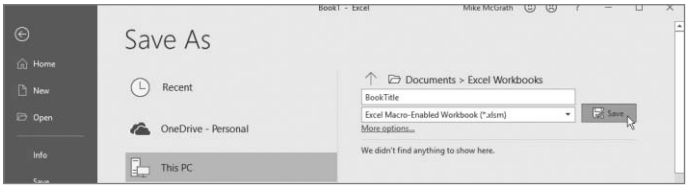


2 Если вы сохраняете книгу с макросом, то появится окно с предупреждением, точно ли вы хотите продолжить: если вам нужно сохранить книгу с макросом, нажмите кнопку **No** (Нет).



Нажмите кнопку ▼, чтобы открыть список форматов и выбрать тот, в котором вы хотите сохранить файл.

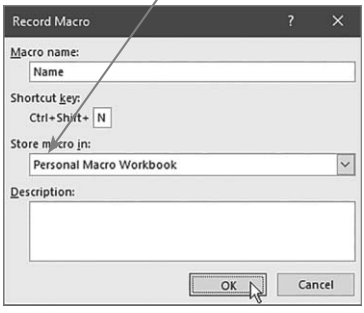
3 Измените тип файла на **Excel Macro-Enabled Workbook** (Книга Excel с поддержкой макросов) и нажмите кнопку **Save** (Сохранить), чтобы сохранить книгу полностью.



Несмотря на то что большинство макросов используются только в конкретной книге, есть универсальные макросы, которые подойдут для разных книг. Такие макросы можно сохранить в личной книге макросов. Этот файл имеет имя *personal.xlsb* и автоматически открывается в фоновом режиме при каждом запуске Excel, поэтому содержащиеся в этом файле макросы могут использоваться в любой книге. Чтобы сохранить макрос в личной книге макросов, выберите соответствующий параметр в диалоговом окне **Record Macro** (Запись макроса) перед началом записи макроса:

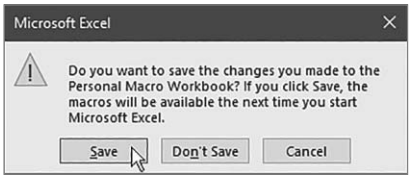
Обычно личная книга макросов запускается в скрытом окне. Чтобы сделать окно видимым, перейдите на вкладку **View** (Вид) и нажмите кнопку **Unhide** (Отобразить) в группе элементов **Window** (Окно).

1 Нажмите кнопку **Record Macro** (Запись макроса) и назовите макрос как **Name**. Затем в раскрывающемся списке выберите пункт **Personal Macro Workbook** (Личная книга макросов).



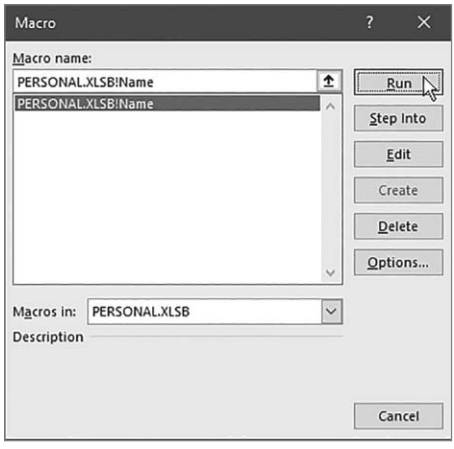
2 Впишите название книги в выбранную ячейку, нажмите кнопку **Stop Recording** (Остановить запись) и закройте Excel.

3 Появится диалоговое окно с вопросом, хотите ли вы сохранить изменения в личной книге макросов. Нажмите кнопку **Save** (Сохранить) для сохранения макроса.



4 Снова зайдите в Excel и выберите пункт **Blank workbook** (Пустая книга), затем нажмите кнопку **Macros** (Макросы) в группе элементов **Code** (Код).

5 Выберите сохраненный макрос под названием **Name** и нажмите кнопку **Run** (Выполнить), чтобы ввести название в ячейку.



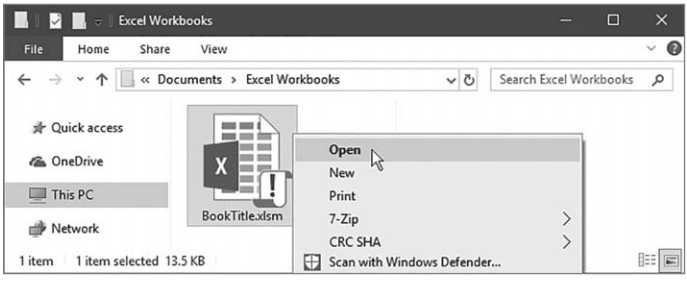
Безопасность при работе с макросами

Файлы в формате Excel Workbook (Книга Excel) (.xlsx) считаются безопасными, так как содержат только данные. В то же время файлы в формате Excel Macro-Enabled Workbook (Книга Excel с поддержкой макросов) (.xlsm) могут представлять угрозу, потому что содержат исполняемые инструкции. Программа Excel учитывает это и автоматически отключает макросы в файлах в формате Excel Macro-Enabled Workbook (Книга Excel с поддержкой макросов) до тех пор, пока пользователь не выразит согласие с тем, что он понимает всю опасность и риски. При открытии книги с макросами появляется запрос безопасности, который предложит пользователю включить макросы. Если пользователь соглашается, то книга считается безопасной, а запрос безопасности больше не появится.

В качестве альтернативы этому методу можно создать папку, в которой будут храниться безопасные книги. Такой метод позволяет размещать файлы Excel Workbook (Книга Excel) с макросами и запускать их без всяких ограничений:

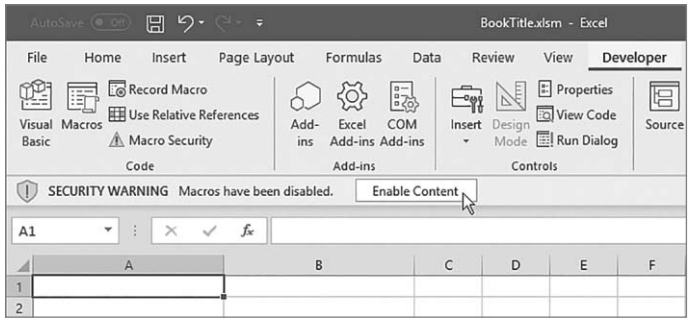
Оба типа файлов — и *.xlsx*, и *.xlsm* — хранят данные в формате XML. Программа Excel также поддерживает формат *.xlsb*, но в таких файлах данные хранятся в двоичном формате. Некоторые пользователи предпочитают последний вариант, однако данные в формате XML проще использовать при работе с другим программным обеспечением.

1. Перейдите в папку, где располагается файл формата Excel MacroEnabled Workbook (Книга Excel с поддержкой макросов), и откройте файл в Excel.

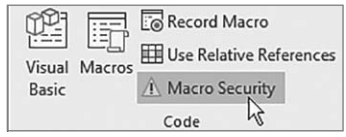


2. Если вы согласны на постоянную активацию макросов в данной книге, нажмите кнопку **Enable Content** (Включить содержимое).

Макросы могут использоваться для распространения вредоносных программ — пожалуйста, будьте осторожны при их активации в файлах, полученных из незнакомых источников.



3 Затем нажмите кнопку **Macro Security** (Безопасность макросов) в группе элементов **Code** (Код). Откроется диалоговое окно **Trust Center** (Центр управления безопасностью).

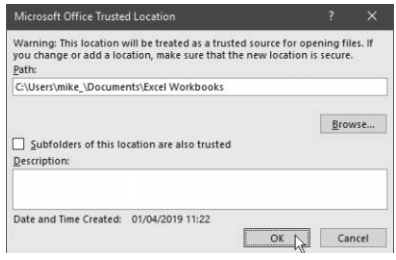


4 В этом диалоговом окне в левом столбце выберите пункт **Trusted Locations** (Надежные расположения).

5 Нажмите кнопку **Add new location…** (Добавить новое расположение…). Откроется диалоговое окно **Microsoft Office Trusted Location** (Надежное расположение Microsoft Office).

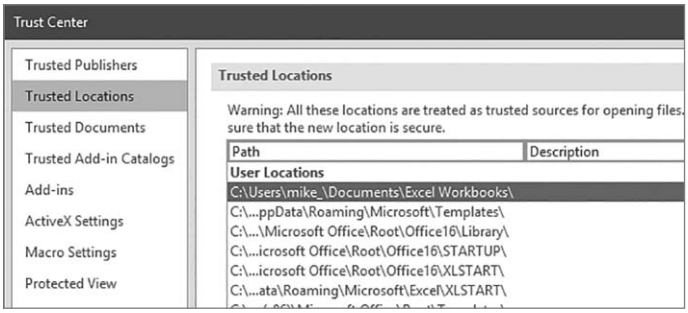
6 Нажмите кнопку **Browse** (Обзор) для выбора каталога, где бы вы хотели хранить файлы формата Excel Macro-Enabled Workbooks (Книга Excel с поддержкой макросов).

7 Нажмите кнопку **OK**. Вы увидите, что выбранное расположение добавится



Вы также можете использовать настройку **Trusted Documents** (Надежные документы), чтобы работать с книгой без ограничений.

в список **Trusted Locations** (Надежные расположения).



Все книги, расположенные в каталогах **Trusted Locations** (Надежные расположения), будут открываться без запросов безопасности.

Заключение

- **VBA** (Visual Basic for Applications) — это язык программирования, встроенный в Excel и обладающий особенностями, которых нет в стандартных инструментах Excel.
- Настройки разработчика активируют VBA и добавляют панель **Developer** (Разработчик) на ленту программы Excel.
- В группе элементов **Code** (Код) панели **Developer** (Разработчик) есть кнопка **Record Macro** (Запись макроса) для создания макроса VBA.
- Кнопка **Macros** (Макросы) в группе элементов **Code** (Код) позволяет создавать макросы.
- Кнопка **Visual Basic** в группе элементов **Code** (Код) открывает редактор для проверки инструкций макросов.
- Подпрограммы макросов хранятся в папке проекта **Module1**.
- Подпрограммы содержат в себе инструкции и комментарии.
- Для запуска макроса можно использовать определенное сочетание клавиш или кнопку **Run** (Выполнить) в диалоговом окне **Macro** (Макросы).

- Инструкции макроса можно изменить в **редакторе Visual Basic**.

- В Excel режим записи макросов по умолчанию ссылается на ячейки согласно их абсолютному положению.

- Кнопка **Use Relative References** (Относительные ссылки) в группе элементов **Code** (Код) активирует еще один способ записи макросов, который ссылается на ячейки согласно их относительному положению.

- Книгу с макросами следует сохранять в формате **Excel Macro-Enabled Workbook** (Книга Excel с поддержкой макросов) и выбирать тип файла **.xlsm**.

- Универсальные макросы можно сохранить в расположении **Personal Macro Workbook** (Личная книга макросов). Таким образом сохраненный макрос будет доступен в любой книге.

- Excel автоматически отключает макросы в файлах формата **Excel Macro-Enabled Workbook** (Книга Excel с поддержкой макросов) до тех пор, пока вы сами не разрешите использовать их.

- Вы можете обозначить папку как **надежное расположение** и сохранять туда файлы формата **Excel Macro-Enabled Workbooks** (Книга Excel с поддержкой макросов). В таком случае вам будет доступен запуск книги без запросов безопасности.

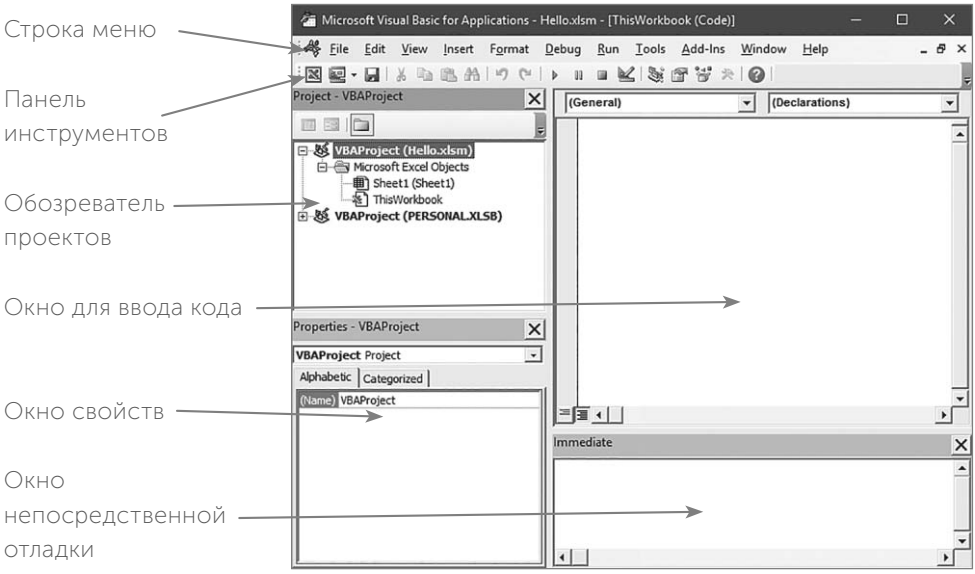
2 Написание макросов

В этой главе вы узнаете, как написать свои собственные инструкции для макроса VBA и как ссылаться на книги, листы и ячейки.

Обзор редактора Visual Basic

Для открытия редактора Visual Basic вы также можете использовать сочетание клавиш **Alt + F11**.

Редактор Visual Basic — это программа, которая при запуске Excel открывается в фоновом режиме. Вы можете отобразить ее окно, нажав кнопку Visual Basic на вкладке **Developer** (Разработчик) в группе элементов **Code** (Код). В редакторе есть несколько окон, которые можно перемещать, открывать и закрывать. На рисунке ниже показаны самые нужные из компонентов и их расположение:



Пока вы не выберете один из пунктов на панели **Project Explorer** (Обозреватель проектов) редактора Visual Basic, окно для ввода кода будет пустым. Если перечисленные окна не видны:

Вы можете перетаскивать окна и закреплять их в верхней, нижней, левой или правой части основного окна редактора Visual Basic.

1. Выберите команду меню **View ⇨ Project Explorer** (Вид ⇨ Обозреватель проектов) или нажмите сочетание клавиш **Ctrl + R**.

2. Выберите команду меню **View ⇨ Properties Window** (Вид ⇨ Окно свойств) или нажмите клавишу **F4**.

3. Выберите команду меню **View ⇨ Immediate Window** (Вид ⇨ Окно непосредственной отладки) или нажмите сочетание клавиш **Ctrl + G**.

4. Нажав и удерживая левую кнопку мыши на заголовке каждого из окон, перетащите их и разместите в порядке, указанном на примере выше.

Основные компоненты редактора Visual Basic

- **Строка меню**. В меню можно найти необходимые команды для создания, форматирования, выполнения и отладки макросов VBA. Многие команды могут выполняться нажатием определенных сочетаний клавиш, которые указаны в меню.

- **Панель инструментов**. На ней расположены значки распространенных действий, которые можно выполнить одним щелчком. Здесь вы найдете такие функции как **Debug** (Отладка), **Edit** (Правка) и **UserForm** (Форма пользователя). Каждую панель инструментов можно настроить так, как вам удобно.

- **Обозреватель проектов**. Панель **Project Explorer** (Обозреватель проектов) позволяет просмотреть список всех книг, которые в данный момент открыты в Excel. Содержимое в книге представлено в виде дерева, которое вы можете развернуть, нажав кнопку ⊞, и свернуть, нажав кнопку ⊟. В каждом проекте есть папка **Microsoft Excel Objects**, содержащая в себе узел объекта книги и узел объекта каждого отдельного листа. Если в книге есть макросы, то вы увидите папку под названием **Modules**, в которой найдете узлы объектов модуля.

- **Окно для ввода кода** позволяет просмотреть код любого элемента модуля, который был выбран на панели **Project Explorer** (Обозреватель

Откройте меню **View ⇨ Toolbars** (Вид ⇨ Панели инструментов), чтобы увидеть список доступных панелей, которые вы можете открыть в редакторе Visual Basic.

проектов). Вы можете написать свой код для нужного макроса или внести изменения в существующий, чтобы поменять логику работы макроса.

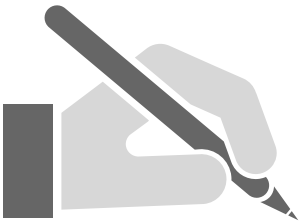
- **Окно свойств**. Здесь вы найдете список свойств для текущего элемента, которые можно редактировать. Для более простой работы можно упорядочить свойства по алфавиту или категории. Изменение любого значения в списке приведет к изменению выбранного свойства.

- **Окно непосредственной отладки** предполагает прямую работу с движком Visual Basic. Здесь вы можете написать собственный код, а затем выполнить его. Оно особенно полезно при тестировании кода и отладки.

Введите любой код Visual Basic в окно непосредственной отладки и нажмите клавишу **Enter**. Попробуйте запустить функцию **MsgBox «Hi»**.

Чтобы более углубленно использовать программы, для продвинутых пользователей вдобавок к стандартным окнам есть другие, проиллюстрированные в этой книге. К таким окнам можно отнести **Object Browser** (Обозреватель объектов), **Locals window** (Окно локальных переменных) и **Watch window** (Окно контрольных значений). Вы можете открыть их с помощью меню **View** (Вид), однако для комфортной работы с программой достаточно окон, перечисленных выше. Чтобы закрыть любое окно в редакторе Visual Basic, нажмите кнопку × в правом верхнем углу.

Создание макроса



Если при записи макроса вы решили сохранить его в рабочей книге, то папка *Modules* появится автоматически. В ней будет располагаться узел объекта **Module1**, в который автоматически запишутся все инструкции VBA. Если вы собираетесь создать макрос путем написания инструкций вручную, то сначала следует добавить в проект новый модуль VBA. Это можно сделать с помощью редактора Visual Basic. В окно кода впишите следующие три фрагмента:

- **Объявления** — это операторы, предоставляющие необходимую информацию для использования в модуле.
- **Подпрограммы** — это программные инструкции, с помощью которых можно выполнять действия в книге.
- **Функции** — это программные инструкции, которые возвращают одно значение вызывающей программе.

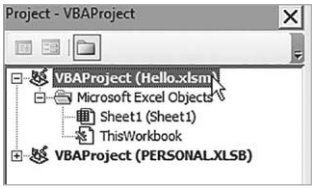
В одном модуле VBA может быть несколько объявлений, подпрограмм и функций, поэтому весь код хранится вместе. В более крупных проектах в целях удобства код может храниться в нескольких модулях, но это никак не влияет на производительность макроса:

Подпрограммы похожи тем, что они автоматически записываются при записи макроса. (Больше информации об объявлениях и функциях в следующих главах.)

1 На панели **Developer** (Разработчик) нажмите кнопку **Visual Basic** и откройте редактор.

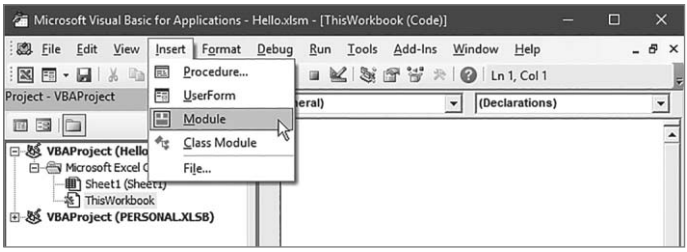
Hello.xlsm

2 На панели **Project Explorer** (Обозреватель проектов) выберите название проекта.



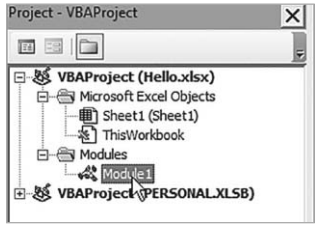
Для добавления модуля на панели **Project Explorer** (Обозреватель проектов) вы можете сначала щелкнуть по названию проекта правой кнопкой мыши, а затем из контекстного меню выбрать команду **Insert** ⇨ **Module** (Вставить — Модуль).

3 В меню редактора Visual Basic выберите команду **Insert** ⇨ **Module** (Вставить ⇨ Модуль).



Редактор Visual Basic дифференцирует синтаксис при вводе кода, автоматически выделяя все ключевые слова синим цветом.

4 На панели **Project Explorer** (Обозреватель проектов) появится папка **Modules**.



5 Откройте папку **Modules**, дважды щелкните мышью по пункту **Module1** и откройте окно для ввода кода.

6 В окне кода напишите показанный ниже код, а затем нажмите клавишу **Enter**.

```
Sub Hello()
```

В Visual Basic есть четыре константы значков окна сообщения: **vbCritical**, **vbQuestion**, **vbInformation** и **vbExclamation**.

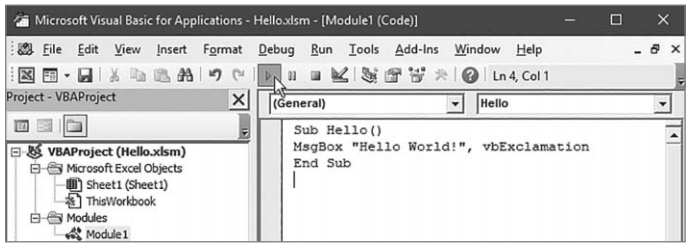
7 Обратите внимание на то, что теперь курсор смещен на одну строку вниз, а внизу будет написана строка **End Sub**, которая введет ваш код в подпрограмму.

8 Затем напишите строку кода

```
MsgBox "Hello World!", vbExclamation
```

9 Нажмите кнопку **Run** (Выполнить) (или клавишу **F5**), чтобы открыть диалоговое окно **Macros** (Макросы), и снова нажмите кнопку **Run** (Выполнить) для выполнения макроса.

Чтобы установить сочетание клавиш, выйдите в Excel, нажмите кнопку **Macros** (Макросы), выберите нужный макрос и нажмите **Options** (Параметры).



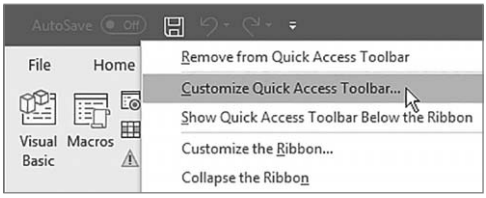
10 Для закрытия окна сообщения нажмите кнопку **OK**, затем сохраните проект в формате **Excel Macro-Enabled Workbook** (Книга Excel с поддержкой макросов) (*.xlsm*).

Настройка панели быстрого доступа

Скорее всего, вы захотите добавить часто используемые макросы на панель быстрого доступа:

BookTitle.xlsm

1 Щелкните правой кнопкой мыши по панели быстрого доступа и выберите команду **Customize Quick Access Toolbar…** (Настройка панели быстрого доступа…) из контекстного меню. Откроется диалоговое окно с настройками программы Excel.



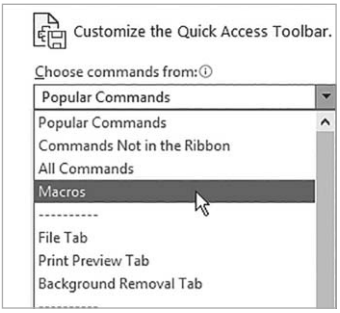
2 В левом столбце выберите пункт **Quick Access Toolbar** (Панель быстрого доступа).

3 Откройте раскрывающийся список **Choose commands from** (Выбрать команды) и выберите пункт **Macros** (Макросы).

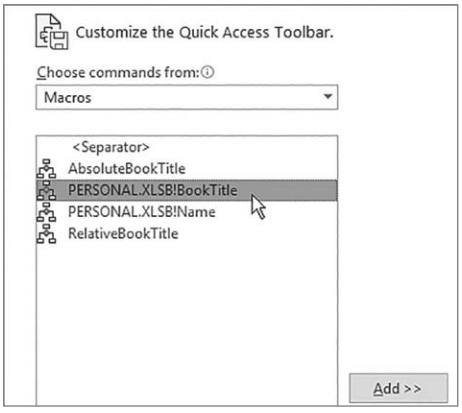
Для удобства на панель быстрого доступа вы можете добавить кнопки часто используемых команд.

Вы также можете добавить кнопки для макросов, которые были сохранены в этой книге или в личной книге макросов.

4 Теперь выберите макросы, кнопки для запуска которых вы хотите добавить на панель быстрого доступа.

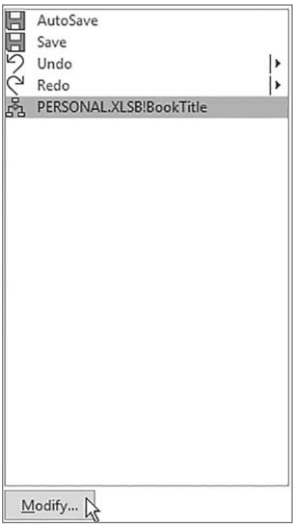


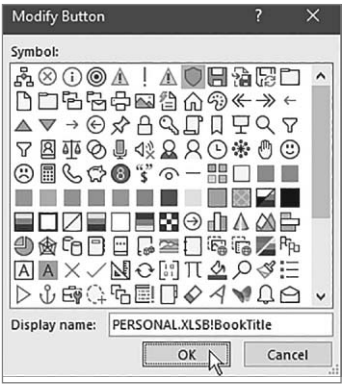
5 Нажмите кнопку **Add** (Добавить). Выбранные макросы появятся в правом столбце.



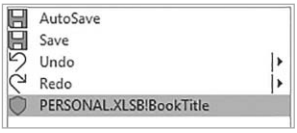
На панели быстрого доступа кнопки не будут цветными.

6 Нажмите кнопку **Modify...** (Изменить...) под правым столбцом. Откроется диалоговое окно **Modify Button** (Изменить кнопку).

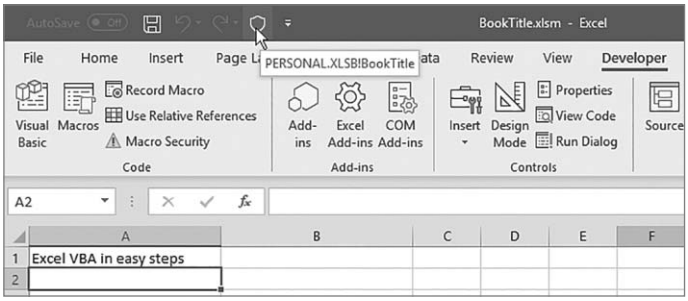




7 В этом окне выберите подходящий значок, затем нажмите кнопку **OK**. Значок появится в правом столбце.



8 В диалоговом окне **Options** (Параметры) нажмите кнопку **OK**. Теперь вы можете увидеть добавленные кнопки на панели быстрого доступа.



9 Нажмите кнопку, чтобы запустить соответствующий макрос.

Чтобы удалить кнопку с панели быстрого доступа, щелкните по ней правой кнопкой мыши и выберите пункт **Remove from Quick Access Toolbar** (Удалить с панели быстрого доступа).

Добавление элементов управления формы

В Excel есть большое количество элементов графического интерфейса, которые более известны как «элементы управления формы». Для упрощенного взаимодействия пользователь может добавить их на свой рабочий лист. Самый известный элемент управления формы — кнопка, с которой можно связать выполнение макроса:

1 Создайте новый пустой лист, откройте редактор Visual Basic и добавьте модуль.

2 На панели **Project Explorer** (Обозреватель проектов) выберите пункт **Module1**, затем напишите показанный ниже код. Таким образом вы создадите макрос под названием **DateTime**.

DateTime.xlsm

```
Sub DateTime()
MsgBox Now
End Sub
```

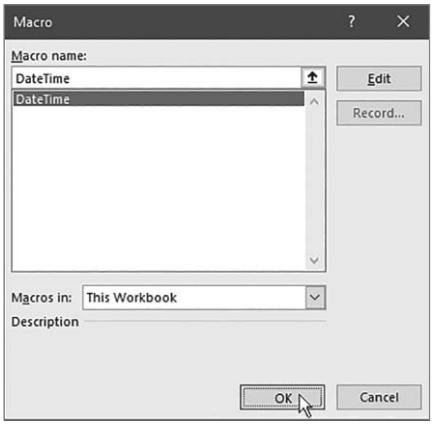
Существуют как стандартные **элементы управления формы**, предназначенные для рабочих листов, так и аналогичные **элементы управления ActiveX**, которые обычно используются в диалоговых окнах Excel UserForm (о них мы поговорим в главе 10). Используйте стандартные **элементы управления формы**, чтобы разместить компоненты на листе.

3 Вернитесь в Excel. Для просмотра всех доступных элементов управления формы выберите пункт **Insert** (Вставить) в группе элементов **Controls** (Элементы управления) на панели **Developer** (Разработчик).

4 Выберите пункт **Button** (Кнопка) в разделе **Form Controls** (Элементы управления формы).

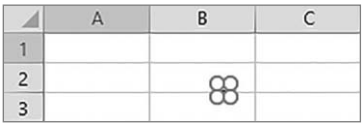
5 На рабочем листе щелкните мышью в позиции, где бы вы хотели расположить кнопку. Откроется диалоговое окно **Assign Macro** (Назначить макрос объекту).

6 Выберите макрос **DateTime** из расположения **This Workbook** (Эта книга).

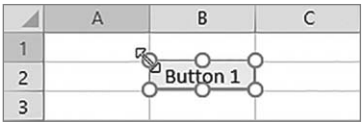


7 Нажмите кнопку **OK**, чтобы назначить макрос кнопке, и закройте диалоговое окно.

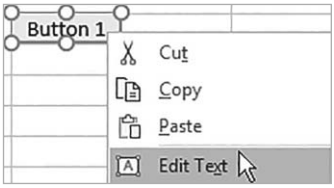
8 Вы увидите четыре маркера в точке щелчка мыши.



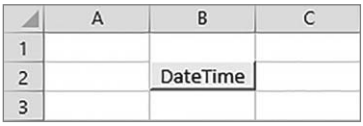
9 Потяните за маркер, чтобы изменить размер кнопки.

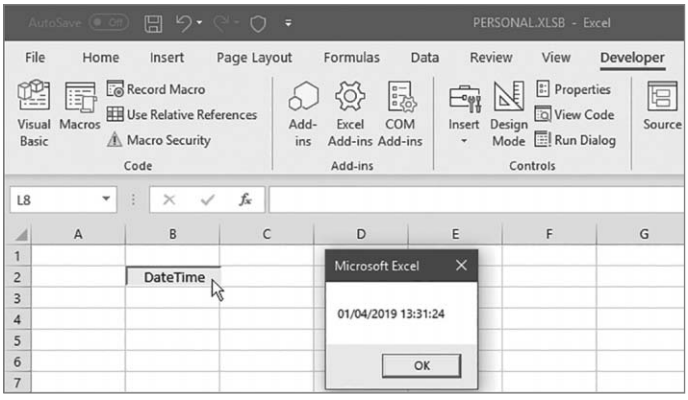


10 Затем щелкните правой кнопкой мыши по кнопке и выберите пункт **Edit Text** (Изменить текст).



11 Курсор сместится к тексту кнопки. Поменяйте текст на **DateTime**.





В контекстном меню также есть опция, с помощью которой можно изменить оформление кнопки. Однако мы советуем вам оставить стандартное оформление, к которому уже привыкли пользователи.

Удалить элемент управления формы очень легко — выберите ненужный элемент и нажмите клавишу **Del**.

12 Выберите любую другую ячейку, чтобы принять изменения, затем нажмите кнопку на листе и выполните макрос.

Определение иерархии

Объекты

В реальном мире каждый элемент можно назвать «объектом», а каждый объект состоит в иерархии с другими объектами. Например, ваш дом — объект. Внутри него находятся комнаты, которые также можно назвать объектами. Внутри комнат стоит мебель и т. д.

Если вы откроете редактор Visual Basic и разверните любой из проектов, то увидите, что там есть папка **Microsoft Excel Objects**. Это потому, что все в Excel подчиняется иерархии.

Объект **Application** (Приложение) располагается на верхнем уровне иерархии, и его можно сравнить с вашим домом. Внутри него находится книга, содержащая рабочий лист, который, в свою очередь, содержит диапазон. В коде VBA вы можете ссылаться на любой объект в иерархии. Например, можно выбрать ячейку **A1** на рабочем листе **Sheet1** (Лист1) вот так:

```
Application.ThisWorkbook.Sheets("Sheet1").Range("A1").
Select
```

В данном случае **ThisWorkbook** представляет уровень объекта книги, а **Sheets("Sheet1")** — уровень объекта рабочего листа. К счастью, вам не придется указывать все уровни иерархии, поскольку Excel автоматически понимает, что вы ссылаетесь на приложение, активную рабочую книгу и рабочий лист, поэтому вы можете выбрать ячейку A1 вот так:

```
Range("A1").Select
```

Коллекции

Подобно тому, как на вашей улице может располагаться несколько домов, многие объекты в Excel

содержат коллекции. Например, в каждом объекте книги есть определенное количество рабочих листов. Каждому рабочему листу присваивается свой порядковый номер, который определяет положение листа в коллекции — это можно сравнить с домом, на котором висит табличка с названием улицы. В коде VBA вы можете ссылаться на любой рабочий лист, просто указав его порядковый номер или название для алгоритма **Worksheets()**. К примеру, вы можете выбрать ячейку **A1** на листе с названием **Sheet1** (Лист1), используя любую из предоставленных ниже строк кода:

```
Worksheets(1).Range("A1").Select

Worksheets("Sheet1").Range("A1").Select
```

Свойства

Характеристики объекта больше известны как его «свойства». Свойства вашего дома — это его цвет и возраст. Некоторые свойства — цвет, например — вы можете изменить, но другие изменению не подлежат — вы не можете поменять, к примеру, год постройки вашего дома.

Точно также и в Excel одни свойства объекта можно изменять, а другие — нет. В коде VBA вы ссылаетесь на свойство по его имени и можете менять значения для «гибких» свойств. Например, вы можете переименовать рабочий лист, присвоив новое значение свойству **Name**:

```
Worksheets("Sheet1").Name = "DataSheet"
```

Содержимое ячейки можно отобразить, ссылаясь на свойство, используемое только для чтения, с именем **Text**:

```
MsgBox Range("A1").Text
```

Действия

В Excel постоянно может быть активна только одна книга, один рабочий лист и одна ячейка. VBA понимает это и предоставляет свойства приложения (**Application**), позволяющие писать код, который

Термины «функция» и «метод» в какой-то степени синонимичны, потому что каждый из них обеспечивает функциональность. Выражаясь техническими терминами, метод — это функция, которая связана с объектом, а функция независима.

Для просмотра свойств объекта вы можете использовать панель **Project Explorer** (Обозреватель проектов) в редакторе Visual Basic. Выберите команду меню **View** ⇨ **Object Browser** (Вид ⇨ Обозреватель объектов) или нажмите кнопку **F2**.

Чтобы увидеть название рабочего листа, выполните код **MsgBox ActiveSheet.Name**, а также **MsgBox ActiveCell.Text** для просмотра содержимого выбранной вами ячейки.

не относится к какой-то конкретной книге, листу или диапазону:

Свойство	Описание
ActiveWorkbook	Активная книга
ActiveSheet	Активный лист или диаграмма
ActiveCell	Активная ячейка
ActiveChart	Активная диаграмма
ActiveWindow	Активное окно
Selection	Выбранный объект
ThisWorkbook	Книга, содержащая код VBA

Определение диапазона

Согласно терминам программирования функция или метод, принимающие разное количество аргументов, называются «перегруженными».

Метод **Range()** позволяет ссылаться на одну ячейку или диапазон ячеек в соответствии с данными, указанными в круглых скобках. Например, если вы укажете один аргумент, заключенный двойными кавычками, то метод будет ссылаться на одну ячейку. Если укажете два аргумента, то они будут обозначать начало и конец диапазона ячеек.

Один аргумент метода **Range()** может указывать на ячейку согласно ее позиции, например, **"A1"**, а также согласно ее имени, если оно есть. Кроме того, один аргумент может ссылаться на несколько ячеек, если при записи данных вы разделите их запятыми. А еще один аргумент может ссылаться на ячейку, находящуюся на пересечении двух указанных диапазонов, которые разделены пробелом. Также один аргумент может указывать на диапазон ячеек согласно их начальному и конечному расположению, разделенному двоеточием, например, **"A1: C1"**. Два аргумента метода **Range()** могут определять диапазон ячеек по их начальному и конечному расположению: вам нужно разделить аргументы запятой, например, **"A1", "C1"**.

Метод **Range()** может ссылаться и на определенный объект рабочего листа, например, **Worksheets("Sheet1")**, или на активный объект листа **ActiveSheet**, однако при желании это можно опустить,

потому что Excel будет считать, что код VBA ссылается на активный рабочий лист.

У каждой ячейки есть свойство **Clear**, которое можно использовать для удаления содержимого ячейки или стилей оформления, а также свойство **Interior.Color**, которому можно присвоить константу для изменения цвета фона:

SetRange.xlsm

1. Откройте новый пустой лист, откройте редактор Visual Basic и добавьте в проект модуль.

2. На панели **Project Explorer** (Обозреватель проектов) выберите пункт **Module1**, затем напишите показанный ниже код в окно кода для создания макроса с названием **SetRange**.

```
Sub SetRange ()
' Сюда помещаются инструкции (Шаги 3–10).
End Sub
```

3. Добавьте показанный ниже код для удаления оформления и содержимого ячейки.

```
Worksheets("Sheet1").Range("A1: C8").Clear
```

4. Добавьте код для ссылки на ячейку согласно ее расположению.

```
Worksheets("Sheet1").Range("A1").Interior.Color =
vbRed
```

5. Добавьте код, чтобы назвать отдельную ячейку **TopCell**.

```
ActiveSheet.Range("B1").Name = "TopCell"
```

6. Теперь добавьте код для ссылки на ячейку согласно ее названию.

```
ActiveSheet.Range("TopCell").Interior.Color =
vbGreen
```

7. Добавьте код для ссылки на ячейку в области пересечения.

```
ActiveSheet.Range("A1: C1 C1: C3").Interior.Color =
vbBlue
```

Для обнаружения методов объекта в редакторе Visual Basic вы можете использовать панель **Object Browser** (Обозреватель объектов). Выберите команду меню **View ⇨ Object Browser** (Вид ⇨ Обозреватель объектов) или нажмите клавишу **F2**.

Если активный лист не рабочий, то инструкции метода **Range()** выдадут ошибку.

Вы можете указать в диапазоне один или два аргумента, но результат будет одинаковым.

8 Затем добавьте код для ссылки на несколько ячеек согласно их абсолютному расположению.

```
Range("B3, A4, C4").Interior.Color = vbYellow
```

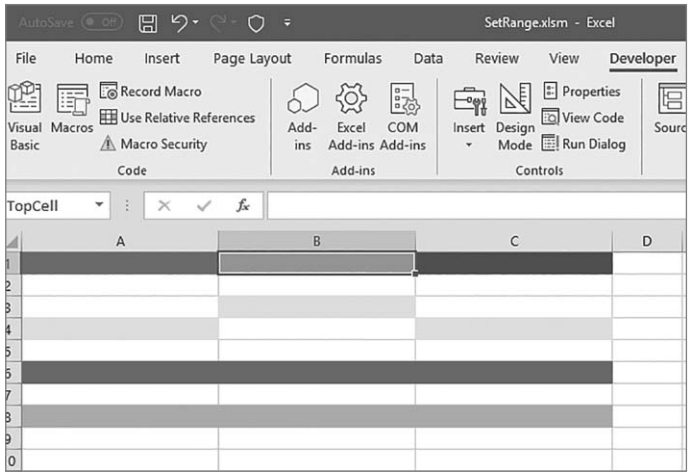
9 Добавьте код для ссылки на диапазон ячеек, указав диапазон как один аргумент.

```
Range("A6: C6").Interior.Color = vbMagenta
```

10 Наконец, добавьте код для ссылки на диапазон ячеек, указав начало и конец диапазона как два аргумента.

```
Range("A8", "C8").Interior.Color = vbCyan
```

11 Выполните макрос. Вы увидите, как у отдельных ячеек и у диапазона ячеек меняется цвет фона.



Адресация ячеек

Объект **Cells** (Ячейки) рабочего листа — это все ячейки, расположенные на рабочем листе. Он имеет свойство **Clear**, которое можно использовать для удаления содержимого ячейки и ее оформления, а также свойство **HorizontalAlignment**, с помощью которого ячейке можно присвоить значение константы Excel, чтобы установить выравнивание ее содержимого.

Метод **Cells()** позволяет ссылаться на одну ячейку согласно координатам, указанным в скобках. Один аргумент будет ссылаться на ячейку согласно ее порядковому положению на листе в диапазоне от 1 до 17179869184. Лучше, когда два аргумента ссылаются на одну ячейку согласно ее номеру строки и числовому или алфавитному расположению в столбце.

Максимальный размер рабочего листа Excel составляет 1048576 строк и 16384 столбца. Всего это 17179869184 ячеек!

Метод **Cells()** также может использоваться для указания аргументов метода **Range()**. Например, вместо того чтобы указывать диапазон ячеек как **Range ("A1", "C1")**, вы можете указать диапазон как **Range (Cells (1, "A"), Cells (1, "C"))**.

Свойство **Select** может быть дополнено методом **Cells()** для выбора ячейки. Вы можете воспользоваться им для ссылки на ячейку, следующую после активной. В качестве альтернативы можно обратиться к методу **Offset()** для ссылки на ячейку относительно выбранной ячейки. Укажите значение аргументов строки и столбца:

SetCells.xlsm

1. Откройте новый пустой лист, затем редактор Visual Basic и добавьте в проект модуль.

2. На панели **Project Explorer** (Обозреватель проектов) выберите пункт **Module1** и напишите показанный ниже код в окно кода. Таким образом вы создадите макрос под названием **SetCells**.

```
Sub SetCells()
' Сюда помещаются инструкции (Шаги 3–11).
End Sub
```

3. Добавьте код для удаления всего содержимого и стилей.

```
Worksheets("Sheet1").Cells.Clear
```

4. Добавьте код для центрирования содержимого ячеек.

```
Worksheets("Sheet1").Cells.HorizontalAlignment =
xlCenter
```

Метод **Cells()** особенно полезен для создания ссылки на ячейку в циклах. Читайте подробнее в главе 5.

5. Добавьте код для ссылки на самую первую ячейку на листе согласно ее порядковому номеру.

```
ActiveSheet.Cells(1).Interior.Color = vbMagenta
```

6. Добавьте код для ссылки на ячейку согласно ее расположению.

```
ActiveSheet.Cells(2, 2).Interior.Color = vbCyan
```

7. Добавьте код для ссылки на ячейку согласно ее абсолютному расположению.

```
Cells(3, "C").Interior.Color = vbYellow
```

8. Теперь добавьте ссылку на другую ячейку согласно ее расположению на листе, пропуская объект рабочего листа.

```
Range(Cells(2, "D"), Cells(2, "F")).Cells(1) = "A"
Range(Cells(2, "D"), Cells(2, "F")).Cells(2) = "B"
Range(Cells(2, "D"), Cells(2, "F")).Cells(3) = "C"
```

9. Добавьте код для выбора отмеченной ячейки.

```
Cells(1, 7).Select
```

10. Добавьте код для ссылки на ячейки согласно их абсолютному положению на листе.

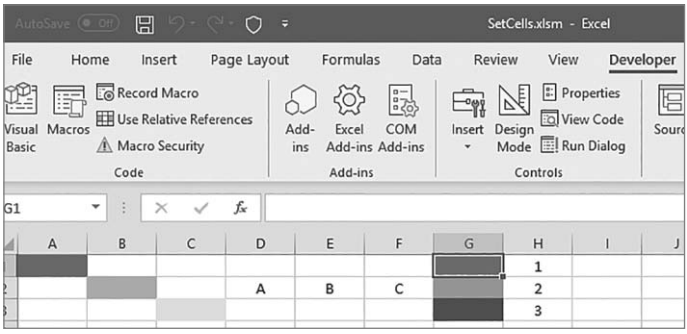
```
ActiveCell.Cells(1).Interior.Color = vbRed
ActiveCell.Cells(2).Interior.Color = vbGreen
ActiveCell.Cells(3).Interior.Color = vbBlue
```

Обратите внимание на то, что каждая ячейка имеет свойство **Value**, с помощью которого можно присвоить значение. Каждая ячейка также имеет не редактируемое свойство **Text**.

11. Теперь добавьте код для ссылки на ячейки согласно их относительному положению от выбранной ячейки.

```
ActiveCell.Offset(0, 1).Value = 1
ActiveCell.Offset(1, 1).Value = 2
ActiveCell.Offset(2, 1).Value = 3
```

12. Выполните макрос. Вы увидите, как выбранные вами ячейки поменяют свой цвет и содержимое.



Заключение

- В редакторе Visual Basic есть панель **Project Explorer** (Обозреватель проектов), которая отображает содержимое всего проекта в виде дерева.
- Код VBA макроса можно написать в модуле в **окне для ввода кода** редактора Visual Basic.
- Один модуль VBA может содержать в себе несколько объявлений, подпрограмм и функций.
- Начало кода подпрограммы начинается с ключевого слова **Sub**, затем следует название, данное пользователем, и круглые скобки **()**.
- Встроенная функция **MsgBox** позволяет создать символ и сообщение на рабочем листе.
- Для удобства вы можете добавить макрос на **панель быстрого доступа**.
- **Элементы управления формы —** это компоненты интерфейса, которые вы можете добавить на свой рабочий лист для удобного взаимодействия.
- На рабочем листе можно установить **кнопку** для запуска необходимого макроса.
- Все объекты в Excel подчиняются четкой иерархии. На самом верху цепи находится **Application**.

- Уровень **Application** включает в себя объект книги, на который можно ссылаться так: **Workbooks()**, **ThisWorkbook** или **ActiveWorkbook**.

- Объект книги включает в себя объект рабочего листа, на который можно ссылаться так: **WorkSheets()**, **Sheets()** и **ActiveSheet**.

- Свойства объекта ячейки — это **Name**, **Value** и **Text**.

- Методы **Range()** и **Cells()** имеют свойство **Select**, которое выбирает ячейку, а метод **ActiveCell** ссылается на выбранную ячейку.

- Аргументы метода **Range()** могут ссылаться на ячейку согласно ее расположению, имени или области пересечения.

- Аргументы метода **Range()** могут ссылаться на диапазон ячеек или на список ячеек, разделенных запятой.

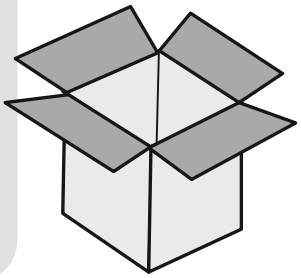
- Объект **Cells** представляет все ячейки на листе.

- Аргументы метода **Cells()** могут ссылаться на отдельную ячейку согласно ее положению на листе или согласно ее порядковому положению.

3 Хранение значений

В этой главе вы узнаете, как сохранять различные типы данных в макросах VBA.

Создание переменных



Переменная — это контейнер в макросе VBA, в котором значение определенного типа данных может храниться в памяти компьютера. На сохраненные данные можно ссылаться, используя имя переменной, — новые значения будут присвоены по мере выполнения макроса. Программист может сам выбирать имя для переменной, но лишь при условии, что оно соответствует соглашениям об именовании, которые указаны в таблице ниже:

Правило наименования	Пример
НЕЛЬЗЯ использовать ключевые слова	**Next**
НЕЛЬЗЯ использовать арифметические символы	**a+b*c**
НЕЛЬЗЯ использовать знаки пунктуации	**%$#@!**
НЕЛЬЗЯ использовать пробелы	**no spaces**
НЕЛЬЗЯ начинать имя с цифры	**2bad**
НЕЛЬЗЯ использовать другие языки	**переменная**
МОЖНО использовать цифры в любом другом месте	**good1**
МОЖНО использовать буквы разного регистра	**UPdown**
МОЖНО использовать нижнее подчеркивание	**is_ok**

Имя переменной может содержать до 254 символов, однако рекомендуется использовать короткие названия.

Для упрощения кода VBA для переменных рекомендуется выбирать понятные имена. Имена переменных могут состоять из нескольких слов, причем их можно сделать более читаемыми с помощью ГорбатогоРегистра. В таком формате вы создаете имя из объединения нескольких слов, каждое из которых начинается с заглавной буквы. В качестве примера приведем имя **MyFirstVariable**. Также можно использовать только строчные буквы и разделять слова нижним подчеркиванием, например, **my_first_variable**. Выберите подходящий вариант именования переменных и используйте его во всем коде макроса VBA.

VBA не учитывает регистр, так что не стоит давать переменным имена, отличающиеся только регистром.

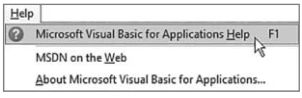
Переменная создается путем присвоения значения действительному имени в «объявлении» переменной:

```
OneDozen = 12
IsValid = True
user_name = "Mike McGrath"
```

В таблице ниже указаны 60 ключевых слов, которые имеют большое значение в Visual Basic. Эти слова нельзя использовать в качестве имен в коде макроса:

Ключевые слова			
And	As	Base	Boolean
Byte	ByVal	Call	Case
Const	Date	Debug	Dim
Do	Double	Each	Else
ElseIf	End	Error	Exit
Explicit	False	For	Function
GoTo	If	In	Integer
Is	Long	Loop	Me
Mod	Next	Not	Nothing
Object	On	Option	Optional
Or	ParamArray	Preserve	Print
Private	Public	ReDim	Resume
Select	Set	Static	String
Sub	Then	To	True
Until	While	With	Xor

Чтобы открыть список всех ключевых слов, в меню **Help** (Справка) редактора Visual Basic выберите команду **Microsoft Visual Basic for Applications Help** (Справка по продукту Microsoft Visual Basic for Applications). Откроется веб-страница, где вы можете провести поиск по запросу «Visual Basic keywords».



Если быть точным, количество ключевых слов в языке Visual Basic превышает 60, однако перечисленные слова общеизвестны и чаще всего используются в примерах в нашей книге.

Определение типов данных

Для номеров строк Excel используйте тип данных **Long**, поскольку он превышает максимальный диапазон типа данных **Integer**.

Переменные могут хранить различные типы данных: целые числа, вещественные числа, строки текста и логические значения True или False. Когда при объявлении данным назначают допустимое имя переменной, VBA автоматически создает переменную, которая подходит для назначения типа данных. На каждый тип приходится определенное количество байтов памяти. В таблице ниже вы можете увидеть типы данных Visual Basic, а также размер выделенной для них памяти и допустимый диапазон значений:

Когда переменной присваивается начальное значение, считается, что она «инициализирована». Однако объявлению переменной, которая определяет тип данных, нельзя присвоить значение в объявлении, так как такая переменная должна быть инициализирована отдельно.

Тип данных	Размер	Диапазон значений
Byte	1 бит	От **0** до **255**
Boolean	2 бита	**True** или **False**
Integer	2 бита	От **–32768** до **32767**
Long	4 бита	От **–2147483648** до **2147483647**
Double	8 битов	От **–1.79769313486231570E+308** до **–4.94065645841246544E-324** и от **4.94065645841246544E-324** до **1.79769313486231570E+308**
Date	8 битов	От **0:00:00 January 1, 0001** до **11:59:59 December 31, 9999**
String	Длина строки	От 0 до 2 миллиардов символов Юникода

При желании в процессе объявления переменной вы можете выбрать тип данных, который может содержать переменная. Благодаря этому в памяти займется определенное пространство до того, как переменной будет присвоено значение в другой строке кода. Для этого в объявлении вам нужно написать

ключевое слово **Dim** перед именем переменной и ключевое слово **As** перед типом данных:

Dim *имя-переменной* As *тип-данных*

Например:

```
Dim OneDozen As Integer
Dim IsValid As Boolean
Dim user_name As String
```

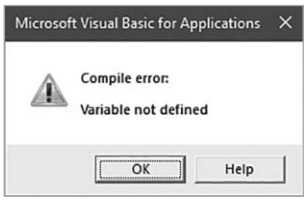
Если в объявлении переменной не указан тип данных, то переменная имеет тип **Variant**, которому можно присвоить данные любого типа. Однако, если вам не нужен именно этот тип данных, то рекомендуем сменить его — так вы сможете использовать память более эффективно.

Для наилучшей практики рекомендуется всегда добавлять показанный ниже код в самом начале модуля:

```
Option Explicit
```

Всегда добавляйте строку **Option Explicit**, так как представленный код предотвращает ошибки в именах переменных.

Если вы попытаетесь создать переменную, не имеющую определенный тип данных, то компилятор VBA выдаст ошибку.



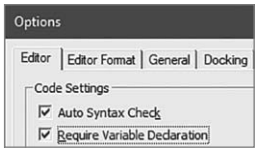
Вы можете ссылаться на значение, которое содержится в переменной, через имя. Также можно узнать тип данных любой переменной, просто указав ее имя в качестве аргумента встроенного метода **TypeName()**:

1. Начните модуль макроса VBA с определения типов данных.
   ```
   Option Explicit
   ```

2. Добавьте подпрограмму для объявления переменных.
   ```
   Sub FirstVar()
   Dim OneDozen As Integer
   ' Сюда помещаются инструкции (Шаги 3 и 4).
   End Sub
   ```

FirstVar.xlsm

Строка **Option Explicit** может автоматически вставляться в начале каждого модуля. Для этого в редакторе Visual Basic выберите команду меню **Tools ⇨ Options** (Инструменты ⇨ Параметры), откройте вкладку **Editor** (Редактор) и установите флажок **Require Variable Declaration** (Обязательное декларирование переменных).



Если числовые значения заключены в двойные кавычки, то программа оценивает их как строку. Оператор + выполняет сложение числовых операндов. Например, ответ на выражение 2+2 равен 4, но если выражение будет представлено как "2"+"2", то ответ будет равен "22".

3. Теперь добавьте код для присвоения начального значения.

```
OneDozen = 12
```

4. Добавьте код для ссылки на хранимое значение и распознавания типа данных переменной.

```
MsgBox OneDozen, vbOKOnly, "Value" MsgBox
TypeName(OneDozen), vbOKOnly, "Data Type"
```

5. Выполните макрос. Вы увидите тип данных и значение переменной.

Управление строками

Текст, назначенный переменной **строкового типа**, всегда должен быть заключен в двойные кавычки «» — так обозначается начало и конец строки. Вы можете объединить строки между собой: для этого используйте оператор объединения **&**. Таким образом вы образуете новую длинную строку.

В VBA есть несколько встроенных функций, которые можно использовать для управления строками. Вы можете узнать длину строки, просто указав имя переменной **String** в качестве аргумента для метода **Len()**. Пробелы можно удалить с помощью метода **Trim()**. Эта функция особенно полезна при работе с информацией, введенной пользователем, потому что в таком случае в код случайно могут быть включены символы пробела. С помощью метода **LCase()** вы можете сменить регистр строки на нижний, а с помощью метода **UCase** — на верхний. Особенно полезно это при сравнении, потому что так вы получаете гарантию, что тексты совпадают. Если вам нужно найти подстроку, то можно выполнить поиск

по строке — просто укажите имя переменной **String** и подстроку в качестве аргументов метода **InStr()**.

Все даты в VBA имеют тип **Date**. Значения должны быть в формате месяц/день/год и заключены в символы решетки **##**. Вы можете обозначить значение **Date** переменной String: укажите имя переменной **Date** и спецификатор преобразования в качестве аргументов функции **Format()**:

Дата и время	Спецификаторы преобразования	Пример
m	Номер месяца	2
mmmm	Название месяца	Февраль
d	Номер дня	14
dddd	Название дня недели	Вторник
yyyy	Год полностью	2021
hh	Час с нулями	08
nn	Минуты с нулями	05
ss	Секунды с нулями	03
am/pm	12-часовой формат времени	am

Существуют сокращения для множества спецификаторов преобразования.

1 Начните модуль макроса VBA с подпрограммы, которая объявляет строковые переменные.

```
Sub StringDate()
Dim Str As String
' Сюда помещаются инструкции (Шаги 2–6)
End Sub
```

StringDate.xlsm

2 Добавьте код для инициализации переменной, объединенных строк и удаления пробелов.

```
Str = " Database "
Str = " Excel " & Str
Str = Trim(Str)
```

Выполнить второй поиск не получится, так как строка была преобразована в нижний регистр.

3 Добавьте код для отображения нового значения строки и количества символов строки.

```
Range("A1").Value = Str
Range("B1").Value = Len(Str)
```

Обратите внимание на то, что результат поиска выдает номер по позиции первого символа в строке, где найдена подстрока, и ноль, если подстрока не найдена.

4 Добавьте код для изменения регистра строки и поиска двух подстрок, а также для отображения новой строки и результатов поиска.

```
Str = LCase(Str)
Range("A2").Value = Str
Range("B2").Value = InStr(Str, "data")
Range("C2").Value = InStr(Str, "Data")
```

5 Объявите и инициализируйте переменную даты.

```
Dim Valentine As Date
Valentine = #2/14/2021#
```

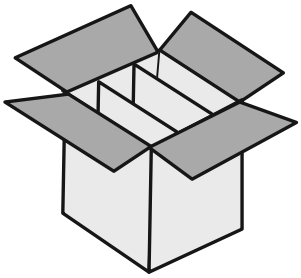
6 Теперь выведите дату и измененную часть даты.

```
Range("A3").Value = Valentine
Str = Format(Valentine, "mmmm, d") Range("B3").
Value = Str
```

7 Выполните макрос, чтобы увидеть строки и даты.

	A	B	C	D
1	Excel Database	18		
2	excel database	11	0	
3	02/14/2021	February, 14		

Работа с массивами



В отличие от обычной переменной переменная массива может хранить несколько элементов данных.

Элементы данных последовательно сохраняются в «элементах» массива, которые по умолчанию имеют порядковый номер. Обычно нумерация начинается с нуля. Следовательно, первое значение массива будет храниться в нулевом элементе массива, второе — в первом элементе массива и т. д.

Переменная массива объявляется так же, как и обычные переменные, однако вам следует указать порядковый номер последнего элемента — таким образом вы устанавливаете размер массива. Его

нужно указать как аргумент в круглых скобках после имени массива:

Dim *имя-массива(последний-индекс)* As *тип-файла*

Теперь каждому элементу объявленного массива можно присвоить значение: просто укажите имя массива и номер индекса в круглых скобках:

имя-массива* (*номер*) = *значение

Часто может потребоваться создать массив с пронумерованными элементами, где номер индекса начинается не с нуля. Для этого вам нужно указать порядковый номер первого и последнего элементов в качестве аргумента объявления с ключевым словом **To**:

Dim *имя-массива* (*первый-индекс* To *последний-индекс)* As *тип-файла*

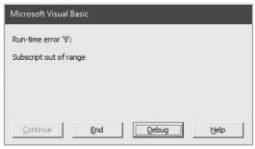
Если вы хотите, чтобы VBA начинал нумерацию индекса массива с единицы, то вы можете добавить показанный ниже код в самом начале модуля перед другим кодом:

Option Base 1

При объявлении массива для создания массива с переменным числом элементов вы можете не указывать аргумент в круглых скобках. Размер такого массива вы сможете установить позже с помощью ключевого слова **ReDim** до того, как элементам будут присвоены значения. С помощью ключевого слова **ReDim** размер массива можно изменять несколько раз, однако, если вы не будете использовать код с ключевым словом **Preserve**, то значения элементов массива будут утеряны:

Dim *имя-массива* () As *тип-файла*
ReDim *имя-массива* (*первый-индекс* To *последний-индекс)*
ReDim Preserve *имя-массива* (*первый-индекс* To *последний-индекс)*

Если во время написания кода вы попытаетесь указать ссылку на несуществующий порядковый номер элемента, то при выполнении макроса VBA выдаст диалоговое окно с ошибкой «Subscript out of range» (Индекс выходит за пределы допустимого диапазона).



Главное преимущество переменных массива будет показано позже с использованием операторов цикла. Узнайте подробнее в главе 5.

FirstArray.xlsm

1 Начните модуль макроса VBA с подпрограммы, которая объявляет строковый массив из трех элементов.

```
Sub FirstArray()
Dim Fruit(2) As String
' Сюда помещаются инструкции (Шаги 2–6).
End Sub
```

Свойство **Text** ячейки позволяет отображать ее содержимое, однако это не обязательно то же самое, что и свойство ячейки **Value**, которое хранится внутри ячейки и используется в формулах и при вычислениях.

2 Добавьте код для инициализации каждого элемента массива и отображения значения элемента с первым номером.

```
Fruit(0) = "Apple"
Fruit(1) = "Banana"
Fruit(2) = "Cherry"
Range("A1") = "First Fruit: " & Fruit(1)
```

3 Теперь объявите другой строковый массив из трех элементов, на этот раз указав первый и последний номер элемента.

```
Dim Veg(1 To 3) As String
```

4 Добавьте код для инициализации каждого элемента массива и отображения значения элемента с первым номером.

```
Veg(1) = "Artichoke"
Veg(2) = "Broccoli"
Veg(3) = "Cabbage"
Range("B1") = "First Veg: " & Veg(1)
```

5 Затем объявите динамический строковый массив и определите его размер, указав первый и последний номер элемента.

```
Dim Flower() As String
ReDim Flower(1 To 3)
```

Первый элемент в массиве **Fruit** имеет нулевой порядковый номер, так что второй элемент будет иметь первый номер.

6 Добавьте код для инициализации каждого элемента массива и отображения элемента с последним номером.

```
Flower(1) = "Azalea"
Flower(2) = "Buttercup"
Flower(3) = "Crocus"
Range("C1") = "Final Flower: " & Flower(3)
```

7 Выполните макрос. Вы увидите значения элементов массива.

	A	B	C	D
1	First Fruit: Banana	First Veg: Artichoke	Final Flower: Crocus	
2				

Описание измерений

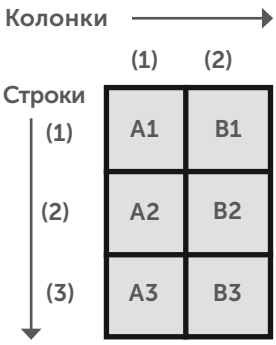
Массив, созданный только с одним порядковым номером, — это одномерный массив. В таком массиве все элементы отображаются в одной строке:

Элемент					
Порядковые номера	(0)	(1)	(2)	(3)	(4)

Массивы могут содержать несколько индексов. Массив с двумя порядковыми номерами называется двумерным массивом. В таком массиве элементы располагаются в нескольких строках:

Первый индекс (0) (1)					
	F	G	H	I	J
Второй индекс	(0)	(1)	(2)	(3)	(4)

В многомерных массивах ссылку на значение, которое содержится в каждом элементе, нужно указывать через номер каждого индекса. Например, в представленном выше двумерном массиве элемент в диапазоне **1, 2** соответствует букве «H».



Двумерные массивы полезны для хранения информации о строках и столбцах. В таком случае первый индекс представляет строку, а второй — столбцы.

Многомерные массивы создаются таким же образом, как и одномерные. Однако при создании многомерного массива в объявлении в качестве аргумента вам нужно указать размер каждого индекса. Не забудьте разделить их запятыми:

Dim *имя-массива* (*final, final, final*) As *тип-данных*

Очень сложно представить массивы, которые содержат в себе больше трех индексов.

Часто может требоваться создать массив с пронумерованными элементами, где номер индекса начинается не с нуля. Для этого вам нужно указать порядковый номер первого и последнего элементов в качестве аргумента объявления с ключевым словом **To**:

Dim *имя-массива* (*first* To *final*, *first* To *final*) As *тип-данных*

Если вы хотите, чтобы VBA начинал нумерацию индекса массива не с нуля, а с единицы, то можно добавить показанный ниже код в самом начале модуля перед другим кодом:

```
Option Base 1
```

Возможно создать многомерный массив, содержащий до 60 индексов, однако вы редко увидите массив с более чем тремя индексами.

Array2D.xlsm

1. Начните модуль макроса VBA с добавления кода, чтобы нумерация начиналась с цифры один.
```
Option Base 1
```

2. Добавьте подпрограмму для объявления двумерного строкового массива из трех элементов с индексами.
```
Sub Array2D()

Dim Basket(3, 3) As String
' Сюда помещаются инструкции (Шаги 3–6).

End Sub
```

3. Теперь добавьте код для инициализации каждого элемента массива с первым индексом.
```
Basket(1, 1) = "Apple"
Basket(1, 2) = "Banana"
Basket(1, 3) = "Cherry"
```

4. Затем добавьте код для инициализации каждого элемента массива со вторым индексом.
```
Basket(2, 1) = "Artichoke"
Basket(2, 2) = "Broccoli"
Basket(2, 3) = "Cabbage"
```

Как и одномерный массив, вы можете объявить массив с переменным числом элементов. Размер каждого индекса должен быть указан в инструкции **ReDim**.

5. Добавьте код для отображения значений первого индекса в первой строке рабочего листа.

```
Range("A1").Value = Basket(1, 1)
Range("B1").Value = Basket(1, 2)
Range("C1").Value = Basket(1, 3)
```

6. Добавьте код для отображения значений второго индекса во второй строке рабочего листа.

```
Range("A2").Value = Basket(2, 1)
Range("B2").Value = Basket(2, 2)
Range("C2").Value = Basket(2, 3)
```

7. Выполните макрос для просмотра

	A	B	C	D
1	Apple	Banana	Cherry	
2	Artichoke	Broccoli	Cabbage	
3				

Вам может показаться, что назначение отдельных элементов массива отдельным ячейкам — это утомительное занятие, и будете правы. В дальнейших примерах мы покажем, как можно эффективно назначать несколько элементов нескольким ячейкам.

Представление объектов

Кроме обычных переменных и переменных массива, в VBA есть специальные «объектные переменные». Они представляют собой весь объект Excel, например, **Worksheet** (рабочий лист) или **Range** (диапазон). Объявление такой переменной похоже на объявление обычной переменной, однако есть одно различие: объявление объектной переменной определяет тип объекта, а не тип данных. Для этого используйте показанный ниже синтаксис:

Dim *имя-переменной* As *тип-объекта*

Затем вы можете назначить соответствующий тип объекта с помощью ключевого слова **Set**:

Set *имя-переменной* = *объект*

Объектные переменные объекта **Range** особенно пригодятся для переноса данных из ячеек рабочего листа в переменные VBA и обратно. Для этого

Переменной типа **Variant** могут быть присвоены данные любого типа, однако такие переменные следует использовать только в случае крайней необходимости.

у объекта **Range** есть метод **Resize()**, который можно использовать для размещения строк и столбцов: просто укажите их в качестве двух аргументов.

Переменной типа **Variant** могут быть присвоены значения строк и столбцов с помощью метода **Range()**: происходит копирование данных из ячеек в переменную. Важно понимать, что в таком случае всегда будет создаваться двумерный массив, в котором строки будут представляться первым индексом, а столбцы — вторым. Вы получите такой результат даже в том случае, если присваиваете значение только одной строки, столбца или ячейки.

Методы **LBound()** и **UBound()** возвращают «измерения» индекса массива.

У массива, в который назначаются данные рабочего листа, в одном индексе содержится только одно значение. Это «нижняя граница» индекса массива. На нее можно ссылаться с помощью аргумента **LBound()**: просто укажите имя массива и номер индекса в качестве его аргументов. Похожий аргумент **UBound()** ссылается на «верхнюю границу» массива: в качестве двух аргументов укажите имя массива и номер индекса. Таким образом вы обозначите количество элементов в одном индексе массива.

Возможность определять размер индекса массива означает, что метод **UBound()** может использоваться в качестве аргумента для метода **Resize()** объекта **Range**, чтобы он соответствовал размеру массива и размеру объекта **Range**. Потом весь массив может быть назначен объекту **Range** примерно такого же размера: происходит копирование данных из переменной VBA обратно в ячейки рабочего листа.

ObjectVar.xlsm

1 Для начала напишите значения в ячейки A1:C5.

	A	B	C	D	E	F	G	H	I	J
1	101	201	301							
2	102	202	302							
3	103	203	303							
4	104	204	304							
5	105	205	305							

2 Начните модуль VBA с подпрограммы, которая объявляет объектную переменную и переменную массива.

```
Sub ObjectVar()

Dim Obj As Range
Dim Data As Variant
' Сюда помещаются инструкции (Шаги 3–5).

End Sub
```

В шаге 3 размер объекта **Range** изменяется, и теперь он содержит 5 строк и 1 столбец.

3 Добавьте код для чтения и записи отдельного столбца.

```
Data = Range("A1: A5")
Set Obj = Range("E1")
Set Obj = Obj.Resize(UBound(Data, 1), 1)
Obj.Value = Data
```

В шаге 4 размер объекта **Range** изменяется, и теперь он содержит 1 строку и 3 столбца.

4 Теперь добавьте код для чтения и записи отдельной строки.

```
Data = Range("A1: C1")
Set Obj = Range("G1")
Set Obj = Obj.Resize(1, UBound(Data, 2))
Obj.Value = Data
```

В шаге 5 размер объекта **Range** изменяется, и теперь он содержит 3 строки и 3 столбца.

5 Наконец, добавьте код для чтения и записи нескольких строк и столбцов.

```
Data = Range("A1: C3")
Set Obj = Range("G3")
Set Obj = Obj.Resize(UBound(Data, 1),
UBound(Data, 2))
Obj.Value = Data
```

6 Выполните макрос. Вы увидите измененные значения.

	A	B	C	D	E	F	G	H	I	J
1	101	201	301		101		101	201	301	
2	102	202	302		102					
3	103	203	303		103		101	201	301	
4	104	204	304		104		102	202	302	
5	105	205	305		105		103	203	303	

Объявление констант

При объявлении константы вам нужно указать тип данных; в противном случае VBA сам определит его согласно присвоенному значению.

Значение, которое хранится в переменной, при выполнении кода может изменяться на другое значение того же типа данных. Если макрос использует значение, которое после его выполнения не изменится, то рекомендуется сохранить его в «постоянном» хранилище, а не в переменной.

Константа объявляется с помощью ключевого слова **Const**, затем следует написать имя, выбранное программистом, и указать тип данных. В отличие от объявления переменных, объявление константы содержит присвоение значения:

Const *имя-константы* As *тип-данных* = *значение*

Благодаря использованию имени константы код может стать более читабельным. Например, если вы создаете код для фиксированной налоговой ставки, то вам лучше сохранить значение в константе. Ее имя вы можете использовать во всем коде. Если налоговая ставка когда-нибудь изменится, то код можно обновить, просто поменяв старую ставку на новую:

1 Начните модуль макроса VBA с подпрограммы, которая объявляет константу для налоговой ставки 7% и содержит две переменные.

```
Sub FirstConst()

Const TaxRate As Double = 0.07
Dim Price As Double
Dim Tax As Double
' Сюда помещаются инструкции (Шаги 2 и 3).

End Sub
```

Знак плюс **+** используется для сложения, а символ звездочки ***** для умножения. Больше информации в главе 4 («Арифметические операторы»).

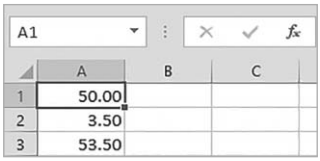
2 Теперь добавьте код для инициализации каждой из переменных.

```
Price = ActiveCell.Value
Tax = Price * TaxRate
```

3. Затем добавьте код для отображения налога и окончательной суммы.

```
ActiveCell.Offset(1, 0) = Tax
ActiveCell.Offset(2, 0) = Price + Tax
```

4. Выберите ячейку, которая содержит текущее значение, а затем выполните макрос. В итоге вы увидите рассчитанный налог и окончательную сумму.



В VBA есть множество предопределенных констант, которые вы можете использовать при написании собственного кода. В имени таких констант есть префикс **vb**. Например, цветовая константа **vbRed**, о которой было написано в главе 1. В Excel есть собственные константы, которые вы можете использовать в макросах, — они имеют префикс **xl**. Например, **xlCenter**. Можно просматривать и искать константы VBA и Excel на панели **Object Browser** (Обозреватель объектов) в редакторе Visual Basic:

1. Откройте редактор Visual Basic, затем выберите команду меню **View ⇨ Object Browser** (Вид ⇨ Обозреватель объектов) или нажмите клавишу **F2**. Откроется панель **Object Browser** (Обозреватель объектов).

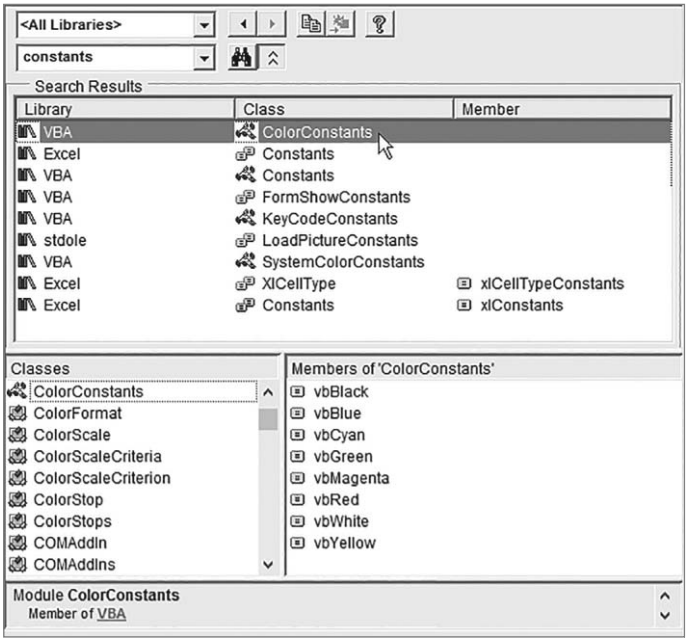
2. В верхнем раскрывающемся списке выберите пункт **VBA** или **Excel** исходя из того, какие константы вы хотите использовать, либо пункт **All libraries** (Все библиотеки).

3. В нижнем раскрывающемся списке напишите **constants**, затем щелкните мышью по значку бинокля для поиска.

Средство записи макросов записывает код, используя предопределенные константы VBA и Excel. Советуем вам изучить записанный код макроса и разобраться в том, какие константы используются в разных целях.

Вспомогательное средство IntelliSense постоянно выдает подсказки при написании кода. Убедитесь в том, что у вас включена функция **Auto List Members** (Автоматический поиск членов) в **Tools** ⇨ **Options** ⇨ **Editor** (Сервис ⇨ Параметры ⇨ Текстовый редактор).

4 Когда появятся результаты поиска, выберите пункт **Class** (Класс). Затем вы увидите список предопределенных констант.



Заключение

- Переменная — это контейнер с именем, в котором могут храниться данные. На эти данные можно ссылаться с помощью имени переменной.
- Присваивая переменной имя, программист должен выбрать его в соответствии с соглашением об именовании.
- Объявление переменной определяет тип данных, которые могут храниться этой переменной, например, **Integer**, **Double** или **Boolean**.
- Если в объявлении переменной не указан тип данных, то ей будет присвоен тип **Variant**, в котором могут храниться данные любого типа.
- Если вы укажете в начале кода макроса строку **Option Explicit**, то объявления переменных должны включать тип данных.
- Значения, присвоенные переменной **строкового типа**, должны быть заключены в двойные кавычки "". Они обозначают начало и конец строки.
- Значения, присвоенные переменной **Date**, должны быть заключены в символы решетки **##** и быть в формате месяц/день/год.
- Массив может хранить несколько значений в разных элементах. Вы можете ссылаться на них, используя имя массива и порядковый номер элемента.
- По умолчанию порядковый номер элементов массива начинается с нуля, однако, если вы добавите код **Option Base 1**, то нумерация начнется с цифры один.
- Объявление переменной массива должно определять размер индекса — в качестве аргумента нужно указать количество аргументов.

- Объявления многомерных массивов указывают размер каждого индекса в виде аргументов, которые разделены запятыми.

- Двумерные массивы полезны при хранении информации о строках в элементах с первым номером и информации о столбцах в элементах со вторым номером.

- Объектная переменная может быть и целым рабочим листом (**Worksheet**), и определенным диапазоном (**Range**). Для правильной работы такая переменная требует ключевое слово **Set**.

- Метод **Range()** можно использовать для присвоения значений ячеек переменной типа **Variant**.

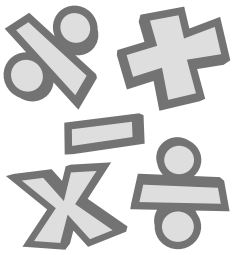
- С помощью аргумента **UBound()** вы можете изменить размер объекта **Range**.

- С помощью ключевого слова **Const** можно присвоить фиксированное значение постоянной переменной.

4 Выполнение операций

В этой главе представлены операторы Visual Basic и операции, которые они выполняют.

Арифметические операторы



Перед вами арифметические операторы, используемые в VBA, и выполняемые ими действия:

Оператор	Операция
+	Сложение
–	Вычитание
*	Умножение
/	Деление
Mod	Деление по модулю
^	Возведение в степень

Оператор + имеет двойное значение, потому что отвечает не только за сложение чисел, но и за объединение строк.

Операторы сложения, вычитания, умножения и деления действуют согласно ожиданиям. Однако при группировке выражений, где используются два или больше оператора, следует проявлять осторожность. Например:

```
result = 4 + 8 * 2 – 6 / 3
```

Порядок, в котором должны выполняться действия, не указан, но, если он установлен по умолчанию, то мы получаем:

```
4 + 16 – 2 = 18
```

Значения, которые используются операторами для формирования выражений, называются операндами: в выражении **2 + 3** числа **2** и **3** — это операнды выражения.

Вы можете изменить порядок, добавив в выражение круглые скобки:

```
result =((4 + 8) * 2) – (6 / 3)
```

Выражение, заключенное во внутренние скобки, решается в первую очередь. Теперь результат такой:

```
(12 * 2) – (6 / 3)
24 – 2 = 22
```

Оператор деления по модулю **Mod** разделит первый операнд на второй и выдаст остаток. Так вы можете определить, имеет ли число четное или нечетное значение. Оператор ^ (возведение в степень) выдает результат первого операнда, возведенного в степень второго операнда.

1 Начните модуль макроса VBA с подпрограммы, которая объявляет и инициализирует две переменные.

Arithmetic.xlsm

```
Sub Arithmetic()

Dim a As Integer
Dim b As Integer
a = 8
b = 4

' Сюда помещаются инструкции (Шаги 2–5)

End Sub
```

2 Добавьте код для отображения результатов сложения и вычитания.

```
Range("A1: C1") = _
Array("Addition:", "8 + 4 =",(a + b))
Range("A2: C2") = _
Array("Subtraction:", "8 – 4 =",(a — b))
```

Обратите внимание, что символ нижнего подчеркивания _ обозначает продолжение строки кода.

3 Добавьте код для отображения результатов умножения и деления.

```
Range("A3: C3") = _
Array("Multiplication:", "8 * 4 =",(a * b))
Range("A4: C4") = _
Array("Division:", "8 / 4 =",(a / b))
```

4 Добавьте код для отображения результата деления по модулю.

```
Range("A5: C5") = _
Array("Modulus:", "8 Mod 4 =",(a Mod b))
```

5 Теперь добавьте код для отображения результата возведения в степень.

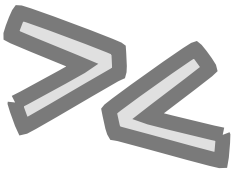
```
Range("A6: C6") = _
Array("Exponent:", "8 ^ 4 =",(a ^ b))
```

В данном случае метод **Array()** используется для создания массива из трех элементов, который соответствует диапазону из выбранных трех ячеек.

6 Выполните макрос для просмотра результата арифметических операций.

	A	B	C
1	Addition:	8 + 4 =	12
2	Subtraction:	8 - 4 =	4
3	Multiplication:	8 * 4 =	32
4	Division:	8 / 4 =	2
5	Modulus:	8 Mod 4 =	0
6	Exponent:	8 ^ 4 =	4096

Операторы сравнения



Операторы сравнения, используемые в VBA, представлены в таблице ниже вместе с описанием сравнения, которое они проводят:

Оператор	Операция
=	Равно (Равенство)
<>	Не равно (Неравенство)
>	Больше чем
<	Меньше чем
>=	Больше или равно
<=	Меньше или равно

ASCII — американская таблица кодов для обмена информацией. В десятичной системе исчисления кода символов прописных букв A–Z находится в диапазоне 65–90, а коды символов строчных букв a–z располагается в диапазоне 97–122.

Каждый оператор сравнения в зависимости от проведенного сравнения выдает логическое значение **True** (Истина) и **False** (Ложь). Оператор = (Равно) сравнивает два операнда операции, и при равенстве их значений выдаст **True**. При неравенстве значений оператор выдаст **False**. Если оба операнда — это одно и то же число, то они равны. Если два операнда представлены строками, то оператор для выполнения сравнения численно проверяет значение кода ASCII обоих операндов.

Оператор **<>** (Не равно) выдает ответ **True**, если два операнда не равны. Этот оператор использует те же правила, что и оператор = (Равно). Операторы равенства и неравенства пригодятся вам при проверке состояния двух переменных при выполнении условного ветвления в макросе в соответствии с результатом.

Оператор **>** (Больше чем) сравнивает два операнда и выдает ответ **True**, если первый операнд больше второго. Оператор **<** (Меньше чем) выполняет точно такое же сравнение, но выдает ответ **True**, если первый операнд по значению меньше второго. Эти операторы чаще всего используются для проверки значения счетчика итераций в цикле.

Если после оператора **>** (Больше чем) или **<** (Меньше чем) вы добавите оператор = (Равенство), то получите ответ **True** при условии равенства значений двух операндов.

Comparison.xlsm

1 Начните модуль макроса VBA с подпрограммы, которая объявляет и инициализирует пять переменных.

```
Sub Comparison()

Dim Nil As Integer
Dim Num As Integer
Dim Max As Integer
Dim Lower As String
Dim Upper As String

Nil = 0
Num = 0
Max = 1
Lower = "a"
Upper = "A"

' Сюда помещаются инструкции (Шаги 2–3)

End Sub
```

2 Теперь добавьте код для отображения результатов равенства и неравенства.

```
Range("A1: C1") = _
        Array("Equality:", "0 = 0",(Nil = Num))
Range("A2: C2") = _
        Array("Equality:", "a = A",(Lower = Upper))
Range("A3: C3") = _
        Array("Inequality:", "0 <> 1",(Nil <> Max))
```

При сравнении строк, которые содержат несколько символов, также учитывается их порядок: например, **«ABC»** не равно **«BAC»**.

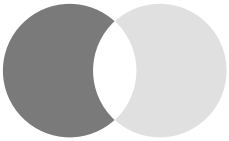
3 Теперь добавьте код для отображения результатов операций больше или меньше.

```
Range("A4: C4") = _
        Array("Greater:", "0 > 1",(Nil > Max))
Range("A5: C5") = _
        Array("Less:", "0 < 1",(Nil < Max))
Range("A6: C6") = _
        Array("Less Or Equal:", "0 <= 1",(Nil <= Max))
```

4 Выполните макрос. Отобразятся результаты проведенных операций сравнения.

	A	B	C	
1	Equality:	0 = 0	TRUE	
2	Equality:	a = A	FALSE	
3	Inequality:	0 <> 1	TRUE	
4	Greater:	0 > 1	FALSE	
5	Less:	0 < 1	TRUE	
6	Less Or Equal:	0 <= 1	TRUE	

Логические операторы

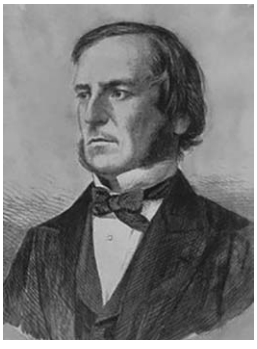


Перед вами логические операторы, которые используются в VBA, и выполняемые ими операции:

Оператор	Операция
And	Логическое И
Or	Логическое ИЛИ
Xor	Логическое исключающее ИЛИ
Not	Логическое НЕТ

Логические операторы используются с операндами, которые имеют логическое (булево) значение **True** или **False**, или выражениями, которые выдают **True** или **False**.

Термин «булево» относится к системе логического мышления, которую разработал английский математик Джордж Буль (1815–1864).



Оператор **And** (Логическое И) оценивает два операнда и выдает **True** только в том случае, если оба операнда имеют логическое значение. В ином случае оператор выдаст **False**. Он особенно полезен при условном ветвлении, когда направление макроса определяется тестированием двух условий. Если оба условия выполняются, то макрос будет развиваться в одном направлении, а если нет, то в другом.

В отличие от оператора **And**, для которого оба операнда должны быть **True**, оператор **Or** (Логическое ИЛИ) оценивает операнды и выдает **True** в том случае, если один или два операнда сами выдают **True**. Если ни один из операндов не соответствует условию, то оператор выдаст **False**. **Or** будет полезен при совершении действий, когда выполнено одно или два условия. Оператор **Xor** (Логическое исключающее ИЛИ) работает практически так же, однако выдает **True** в том случае, если только один из операндов выдает **True**.

Оператор **Not** (Логическое НЕТ) — это унарный оператор, который используется только с одним операндом. Он возвращает обратное значение этого операнда, так что если переменная **A** имеет значение **True**, то **Not A** будет иметь значение **False**. Этот оператор полезен для изменения значения переменной в последовательных итерациях цикла с помощью кода а-ля **A = Not A**. Таким образом, на каждой итерации цикла логическое значение будет заменяться на противоположное, например, включение и выключение переключателя света.

Logic.xlsm

1 Начните модуль макроса VBA с подпрограммы, которая объявляет и инициализирует две переменные.

```
Sub Logic()

Dim Yes As Boolean
Dim No As Boolean
Yes = True
No = False

' Сюда помещаются инструкции (Шаги 2–5)

End Sub
```

2 Добавьте код для отображения результатов соединения.

```
Range("A1: E1") = Array("AND Logic:", _
        "True And True =",(Yes And Yes), _
        "True And False =",(Yes And No))
```

3 Добавьте код для отображения результатов дизъюнкции.

```
Range("A3: E3") = Array("OR Logic:", _
        "True Or True =",(Yes Or Yes), _
        "True Or False =",(Yes Or No))
```

4 Добавьте код для отображения результатов исключения.

```
Range("A5: E5") = Array("XOR Logic:", _
        "True Xor True =",(Yes Xor Yes), _
        "True Xor False =",(Yes Xor No))
```

5 Добавьте код для отображения результатов отрицания.

```
Range("A7: E7") = Array("NOT Logic:", _
        "Yes =", Yes, "Not Yes =",(Not Yes))
```

6 Выполните макрос и посмотрите результаты логических операций.

	A	B	C	D	E
1	AND Logic:	True And True =	TRUE	True And False =	FALSE
2					
3	OR Logic:	True Or True =	TRUE	True Or False =	TRUE
4					
5	XOR Logic:	True Xor True =	FALSE	True Xor False =	TRUE
6					
7	NOT Logic:	Yes =	TRUE	Not Yes =	FALSE

Обратите внимание на то, что выражение **False And False** выдает **False**, а не **True**, как можно было подумать из выражения «минус на минус дает плюс».

Объединение строк

Во избежание непредвиденных результатов не используйте оператор **+** для объединения строк.

VBA предоставляет два оператора, с помощью которых можно объединить строковые значения:

- Оператор объединения **&** всегда оценивает операнды как строковые значения, даже если на самом деле они числовые и не заключены в кавычки. Этот оператор всегда будет выдавать результат объединенной строки, поэтому он рекомендуется для объединения строк. Например:

  "1" & "2" возвращает **"12"**
  "1" & 2 так же возвращает **"12"**

- Оператор сложения **+** не всегда оценивает операнды как строковые значения, если один из операндов числовой. Этот оператор не всегда выдает результат в виде объединенной строки, поэтому рекомендуется использовать его только для сложения. Например:

  "1" + "2" возвращает **"12"**
  Но **"1" + 2** возвращает **"3"**

Все пробелы в объединенной строке сохраняются.

Объединенные строки просто объединяют содержащиеся в них символы в том порядке, в котором они указаны в строке. Часто вам захочется разделить строки пробелом или любым другим разделителем. Объединив несколько строк, вы можете объединить промежуточную строку, которая содержит разделитель:

```
Concat = Str1 & " " & Str2
Concat = Str1 & ", " & Str2
```

Если вы захотите объединить большое количество строк, то показанный ниже метод быстро вас утомит и приведет к тому, что код станет нечитаемым. Альтернатива в таком случае — это метод **Join()**, который принимает массив в качестве первого аргумента. Объединяемые строки могут быть указаны в качестве аргументов метода **Array()**:

```
Concat = Join(Array(Str1, Str2, Str3, Str4))
```

Метод **Join()** автоматически вставляет один пробельный разделитель в объединенную возвращаемую строку, однако в качестве второго аргумента вы можете указать другой разделитель:

```
Concat = Join(Array(Str1, Str2, Str3, Str4), ", ")
```

Если вы хотите, чтобы объединенная возвращаемая строка отображала соединенные строки на отдельных строках, то вы можете указать разделитель как разрыв строки с помощью константы **vbNewLine**.

JoinStr.xlsm

Вы можете встретить в коде константу **vbCrLf**, которая используется для разрыва строк. Она делает то же самое, что и константа **vbNewLine**. Сюда можно отнести и возврат каретки с переводом строки, потому что раньше у компьютеров еще не было мониторов и все печаталось на бумаге.

Для того чтобы начать новую строку, каретку необходимо было сместить к левому краю бумаги, а затем поднять лист вверх на одну строку. (На иллюстрации изображен компьютер Philips P320 1972 года выпуска.)

1. Начните модуль макроса VBA с подпрограммы, которая объявляет и инициализирует пять переменных.

```
Sub JoinStr()

Dim Name As String
Dim Address As String
Dim City As String
Dim Zip As String
Dim Concat As String

' Сюда помещаются инструкции (Шаги 2–5)

End Sub
```

2. Теперь добавьте код для инициализации четырех переменных.

```
Name = "Ripley's"
Address = "175 Jefferson St."
City = "San Francisco"
Zip = "CA 94133"
```

3. Добавьте код для отображения объединенной строки, разделяющей значения двух переменных многоточием и пробелами.

```
Concat = Name & " ... " & City
Range("A1") = Concat
```

4. Теперь добавьте код для отображения объединенной строки, разделяющей значения четырех переменных запятой и пробелами.

```
Concat = Join(Array(Name, Address, City, Zip), ", ")
Range("B1") = Concat
```

5 Добавьте код для отображения объединенной строки, разделяющей значения четырех переменных разрывом строки.

```
Concat = _
  Join(Array(Name, Address, City, Zip), vbNewLine)
Range("C1") = Concat
```

6 Выполните макрос. Вы увидите объединенные строки.

	A	B	C
1	Ripley's ... San Francisco	Ripley's, 175 Jefferson St., San Francisco, CA 94133	Ripley's 175 Jefferson St. San Francisco CA 94133
2			

Приоритеты операций

Когда выражение содержит несколько операторов, то порядок, в котором будут выполняться операции, определяется «приоритетом операторов», установленным в VBA по умолчанию. Например, в выражении **3 * 8 + 4** по умолчанию определено, что сначала выполняется умножение, поэтому результат будет равен 28 (24+4).

В таблице ниже представлен список приоритетов операторов в порядке убывания: операторы на верхних строках имеют наивысший приоритет. Следовательно, операторы, располагающиеся в самом низу таблицы, выполняются в последнюю очередь. Если в выражении есть операторы с равным приоритетом, то операции выполняются согласно порядку, установленному в этой таблице, — то есть слева направо.

Для четкого порядка выполнения операций рекомендуется использовать круглые скобки **()**, а не полагаться на порядок приоритета выполнения операций, установленный по умолчанию.

Оператор	Операция	Порядок
^	Возведение в степень	1
*	Умножение	2
/	Деление	
Mod	Деление по модулю	3
+	Сложение	4
-	Вычитание	
&	Объединение	5

Оператор	Операция	Порядок
=	Равенство	
<>	Неравенство	
>	Больше чем	6
<	Меньше чем	
>=	Больше или равно	
<=	Меньше или равно	
Not	Логическое НЕТ	
And	Логическое И	
Or	Логическое ИЛИ	7
Xor	Логическое исключающее ИЛИ	

Для операторов с равным приоритетом круглые скобки заменяют порядок слева направо.

Выражения, которые содержат круглые скобки, определяют порядок приоритета выполнения операций. Сначала операции выполняются во внутренних скобках, затем во внешних. В самую последнюю очередь выполняются операции вне скобок.

Precedence.xlsm

1 Начните модуль макроса VBA с подпрограммы, которая объявляет и инициализирует три переменные.

```
Sub Precedence()

Dim A As Integer
Dim B As Integer
Dim C As Integer

A = 8
B = 4
C = 2

' Сюда помещаются инструкции (Шаги 2–5)

End Sub
```

Обратите внимание на то, что выражение целиком заключено в круглые скобки — так проще указать его в качестве аргумента. Это никак не влияет на порядок работы, но делает код более понятным.

2 Добавьте код для выполнения вычисления согласно порядку, установленному программой по умолчанию.

```
Range("A1: C1") = Array("Default Order:", _
        "8 * 4 + 2 =",(A * B + C))
```

3 Теперь добавьте код для выполнения точно такого же вычисления согласно принудительному порядку.

```
Range("A2: C2") = Array("Forced Order:", _
        "8 * (4 + 2) =",(A *(B + C)))
```

4 Затем добавьте код для отображения результата вычисления, в котором порядок был установлен программой.

```
Range("A4: C4") = Array("Default Order:", _
        "8 ^ 4 / 2 =",(A ^ B / C))
```

С такими результатами: **32 + 2 = 34**, **8 * 6 = 48** и **4096 / 2 = 2048**; **8 ^ 2 = 64**.

5 Добавьте код для отображения результата вычисления, в котором порядок был принудительный.

```
Range("A5: C5") = Array("Forced Order:", _
        "8 ^ (4 / 2) =",(A ^(B / C)))
```

6 Выполните макрос. Вы увидите, что результаты вычислений получились разными.

	A	B	C	D
1	Default Order:	8 * 4 + 2 =	34	
2	Forced Order:	8 * (4 + 2) =	48	
3				
4	Default Order:	8 ^ 4 / 2 =	2048	
5	Forced Order:	8 ^ (4 / 2) =	64	
6				

Заключение

- Работа арифметических операторов для сложения **+**, вычитания **–**, умножения ***** и деления **/** соответствует ожиданиям.

- Выражения, заключенные в круглые скобки **()** для читаемости кода, будут выполняться в первую очередь. Сначала выполняются выражения во внутренних скобках, затем во внешних.

- Оператор **Mod** (деление по модулю) делит первый операнд на второй и выдает остаток.

- Оператор **^** (возведение в степень) выдает результат первого операнда, возведенного в степень второго операнда.

- Операторы сравнения **=** (равенство), **<>** (неравенство), **>** (больше чем) и **<** (меньше чем) выдают результат **True** или **False**.

- Добавление оператора **=** (равенство) после кода **>** (больше чем) и **<** (меньше чем) приведет к тому, что оператор будет выдавать результат **True** при условии, что два операнда имеют одинаковое значение.

- Логические операторы **And, Or, Xor** и **Not** используются с операндами, имеющими логическое значение.

- Логический оператор **Not** (логическое НЕТ) используется с одним операндом и выдает его обратное значение.

- Оператор **&** (объединение) может использоваться для объединения строк и всегда выдает строковое значение.

- Оператор **+** (сложение) может использоваться для объединения строк, но не будет выдавать строковое значение, если один из операндов имеет числовое значение.

- Метод **Array()** принимает несколько значений строк для создания массива.

- Метод **Join()** в качестве аргумента может принять массив строк и в качестве результата выдает объединенную строку с пробелом в качестве разделителя.

- Метод **Join()** позволяет указать второй метод для использования другого разделителя, например, такого как константа **vbNewLine**.

- Выражения, содержащие несколько операторов, будут выполнять вычисления в порядке приоритета операторов, установленном программой по умолчанию. Если пользователь включает в выражение круглые скобки, то порядок изменяется принудительно.

5 Создание инструкций

В этой главе вы узнаете, как инструкции могут анализировать выражения и влиять на вариант вычислений, с которым будет выполняться макрос.

Управление ветвями

Условный оператор **If** — это одна из важнейших функций в Visual Basic и многих других языков программирования.

Ключевые слова **If-Then** используются для выполнения проверки условия, с помощью которого можно проверить логическое значение любого выражения. Инструкции будут выполняться только в том случае, если выражение имеет логическое значение **True**. В конце всегда должно стоять ключевое слово **End If**, которое ставится под последней выполняемой инструкцией. Если проверяемое выражение имеет логическое значение **False**, то VBA пропускает этот код и переходит к следующей части макроса.

Синтаксис условного оператора **If-Then** выглядит так:

If ***тестируемое-выражение*** **Then**

инструкции-выполняются-если-выражение-истинно
инструкции-выполняются-если-выражение-истинно
инструкции-выполняются-если-выражение-истинно

End If

Вы можете не заключать выражение в круглые скобки, однако это сделает код более читабельным.

Вы можете проверить несколько выражений с помощью логического оператора **And**. В таком случае инструкции будут выполняться, только если оба выражения имеют значение **True**:

If(*тестируемое-выражение* And *тестируемое-выражение*) Then

инструкции-выполняются-если-выражение-истинно
инструкции-выполняются-если-выражение-истинно
инструкции-выполняются-если-выражение-истинно

End If

Для проверки нескольких выражений вы также можете использовать вложенные условные операторы **If-Then**. Тогда инструкции будут выполняться лишь в том случае, если проверка условий выдаст ответ **True**.

```
If тестируемое-выражение Then

        If тестируемое-выражение Then

            инструкции-выполняются-если-выражение-
            истинно
            инструкции-выполняются-если-выражение-
            истинно
            инструкции-выполняются-если-выражение-
            истинно

        End If

End If
```

Вы также можете выполнить несколько проверок условия **If-Then**. В таком случае создаются альтернативные ветви кода.

IfThen.xlsm

1. Начните модуль макроса VBA с подпрограммы, которая объявляет две переменные.

```
Sub IfThen()

Dim Number As Integer
Dim Message As String
' Сюда помещаются инструкции (Шаги 2–5).

End Sub
```

2. Добавьте код для инициализации переменных с порядковым значением ячейки и сообщения по умолчанию.

```
Number = ActiveCell.Value
Message = "Number Is Below Six"
```

3. Добавьте условный оператор, который заменит сообщение по умолчанию, но только в том случае, если проверка выдаст ответ **True**.

```
If Number > 5 Then

        Message = "Number Is Below Six"
        ' Сюда помещаются инструкции (Шаг 4).

End If
```

В этом примере ни одна инструкция не выполняется, так как проверка условия выдает ответ **False**. В таком случае используется значение сообщения по умолчанию.

4 Теперь добавьте вложенный условный оператор, который добавит замененное сообщение, но только в том случае, если проверка выдаст ответ **True**.

```
If(Number Mod 2 = 0) Then

    Message = Message & _
    vbNewLine & "Number Is Even"

End If
```

Для более подробной информации о том, как добавить элемент управления формы на панель быстрого доступа, см. главу 2.

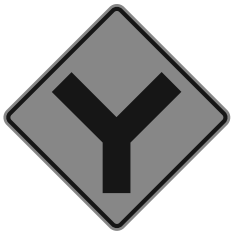
5 Добавьте код для отображения сообщения.

```
ActiveCell.Offset(0, 1).Text = Message
```

6 На рабочий лист добавьте кнопку для выполнения макроса. Затем введите числа и нажмите кнопку, чтобы просмотреть сообщения.

	A	B	C	D
1	Please Enter A Number :	5	Number Is Below Six	
2		6	Number Is Above Five Number Is Even	Run IfThen Macro
3		7	Number Is Above Five	

Альтернативное ветвление



Часто вы можете расширить условный оператор **If-Then** с помощью оператора **Else**. Инструкции выполнятся только в том случае, если при проверке их значение будет **False**. Расширенный условный оператор **If-Then** обладает следующим синтаксисом:

If ***тестируемое-выражение*** **Then**

инструкции-выполняются-если-выражение-истинно
инструкции-выполняются-если-выражение-истинно

инструкции-выполняются-если-выражение-истинно

Else

инструкции-выполняются-если-выражение-ложно
инструкции-выполняются-если-выражение-ложно
инструкции-выполняются-если-выражение-ложно

End If

Для оценки альтернативных выражений можно также использовать оператор **ElseIf**. Тогда инструкции будут выполняться только в том случае, если при проверке значение выражения будет **True**. Вы также можете добавить оператор **Else**, с помощью которого будут выполняться инструкции, если значение выражений, вычисленное операторами **If** и **ElseIf**, будет **False**. Синтаксис будет выглядеть вот так:

Ключевое слово **ElseIf** нельзя записать в виде двух отдельных слов **Else** и **If**.

If *тестируемое-выражение* **Then**

инструкции-выполняются-если-выражение-истинно
инструкции-выполняются-если-выражение-истинно
инструкции-выполняются-если-выражение-истинно

ElseIf *тестируемое-выражение* **Then**

инструкции-выполняются-если-выражение-истинно
инструкции-выполняются-если-выражение-истинно
инструкции-выполняются-если-выражение-истинно

Else

инструкции-выполняются-если-выражение-ложно
инструкции-выполняются-если-выражение-ложно
инструкции-выполняются-если-выражение-ложно

End If

Каждая условная конструкция **If-Then** должна заканчиваться оператором **End If**. Этот оператор должен находиться в последней строке.

Это основной метод программирования, благодаря которому у макроса есть несколько направлений работы, зависящих от результата вычислений. Такой метод называется условным ветвлением. В этом

случае после проверки условия будет выполнен только один из двух наборов инструкций. Также в VBA есть функция **IIf()**:

IIf(*тестируемое-выражение, выполнение-если-истина, выполнение-если-ложь*)

IfElse.xlsm

1 Начните модуль макроса VBA с подпрограммы, которая объявляет две переменные и инициализирует одну из них.

```
Sub IfElse()

Dim Message As String
Dim ThisHour As Integer
ThisHour = Hour(Now)

' Сюда помещаются инструкции (Шаги 2 и 3).

End Sub
```

Метод **Hour()** выдает значение часа, установленного в аргументе **Date**, в диапазоне от 0 до 23.

2 Добавьте проверку условия для инициализации первой переменной через проверку значения второй переменной.

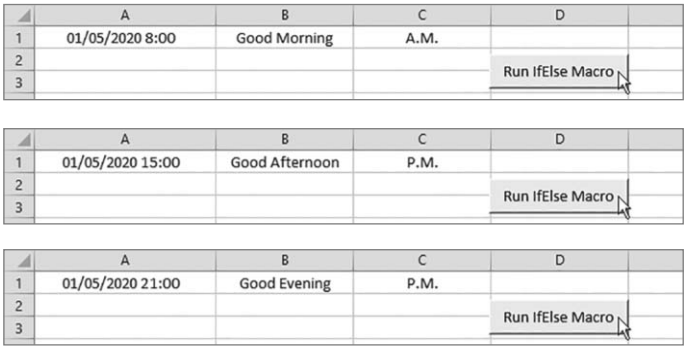
```
If(ThisHour > 17) Then
        Message = "Good Evening"
ElseIf(ThisHour > 11) Then
        Message = "Good Afternoon"
Else
        Message = "Good Morning"
End If
```

3 Теперь добавьте код для отображения текущего времени, приветственного сообщения и текущего времени суток.

```
Range("A1") = Now
Range("B1") = Message
Range("C1") = IIf(ThisHour > 11, "P.M.", "A.M.")
```

Константа **Now** выдает текущую дату и время.

4 Для активации макроса на рабочий лист добавьте кнопку, а затем щелкните по ней мышью. Появится время и сообщение.



Выбор ветвей

Условное ветвление, представленное несколькими операторами **If-Then-Else**, становится более эффективным с конструкцией **Select-Case**. В данном случае проверочное выражение оценивает только одно условие.

Оператор **Select-Case** работает весьма необычным способом. Заданное значение пишется в качестве аргумента оператора, а затем происходит сравнение этого значения с группой операторов **Case**. При обнаружении совпадения выполняются инструкции оператора **Case**.

Оператор **Select-Case** закончит свою работу после нахождения всех совпадений, код будет выполняться и дальше без поиска совпадений.

Список операторов **Case** может закрывать оператор **Case Else**. Он используется для добавления части кода, которая будет выполняться в том случае, если ни в одном из операторов **Case** не будет найдено совпадений.

Каждая конструкция **Select-Case** должна заканчиваться ключевыми словами **End Select**. Данные ключевые слова пишутся на следующей строке после последнего оператора **Case**. Синтаксис оператора **Select-Case** выглядит так:

Желательно сделать так, чтобы оператор **Case Else** был добавлен, хоть он и используется только для вывода сообщения в том случае, если конструкция **Select-Case** не может найти совпадение.

```
Select Case тестируемое-выражение

Case соответствующее-значение
    инструкции-выполняются-если-найдено-совпадение
    инструкции-выполняются-если-найдено-совпадение
Case соответствующее-значение
    инструкции-выполняются-если-найдено-совпадение
    инструкции-выполняются-если-найдено-совпадение
Case соответствующее-значение
    инструкции-выполняются-если-найдено-совпадение
    инструкции-выполняются-если-найдено-совпадение
Case Else
    инструкции-выполняются-если-не-найдено-
    совпадение
    инструкции-выполняются-если-не-найдено-
    совпадение

End Select
```

Конструкция **Select-Case** может содержать любое количество операторов **Case**. В каждом из операторов могут быть несколько инструкций, которые будут выполняться при нахождении совпадений.

Оператор **Case** используется для поиска совпадений более чем одного значения. Просто укажите значения в виде разделенного запятыми списка:

```
Case соответствующее-значение, соответствующее-
значение, соответствующее-значение
```

SelectCase.xlsm

1 Начните модуль макроса VBA с подпрограммы, которая объявляет и инициализирует одну переменную.

```
Sub SelectCase()

Dim Number As Integer
Number = ActiveCell.Value

' Конструкцию вставлять сюда (Шаг 2).

End Sub
```

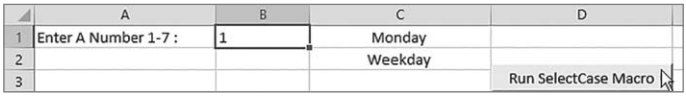
2 Добавьте конструкцию для проверки значения переменной.

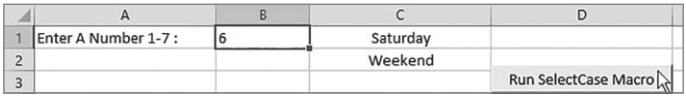
```
Select Case Number

' Сюда помещаются инструкции (Шаг 3).

End Select
```

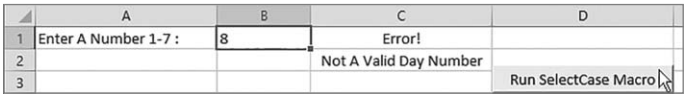
3 Добавьте код, который выполняет поиск совпадения значений.

```
Case 6, 7
  ActiveCell.Offset(0, 1) = WeekdayName(Number)
  ActiveCell.Offset(1, 1) = "Weekend"

Case 1, 2, 3, 4, 5
  ActiveCell.Offset(0, 1) = WeekdayName(Number)
  ActiveCell.Offset(1, 1) = "Weekday"

Case Else
  ActiveCell.Offset(0, 1) = "Error!"
  ActiveCell.Offset(1, 1) = "Not A Valid Day Number"
```

4 На рабочий лист добавьте кнопку для выполнения макроса. Нажмите кнопку для просмотра результата совпадений и ошибок.

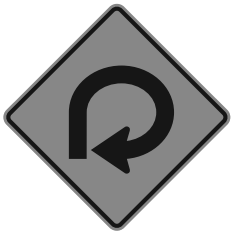






Встроенный метод **WeekdayName()** выдает название дня. Нумерация дней недели начинается с понедельника (1). Раньше в VBA нумерация дней недели начиналась с воскресенья (1).

Управление циклами



Цикл — это фрагмент кода, который постоянно повторяется. Одно полное выполнение всех инструкций в цикле называется итерацией. Длительность цикла зависит от проверки условия, выполняемого внутри самого цикла. Цикл завершится только в том случае, если проверка условия значения выражения выдаст ответ **False**; если ответ будет **True**, то цикл будет выполняться до бесконечности.

В VBA есть четыре вида цикла:

- Цикл **For-Next** выполняет указанное количество итераций.
- Цикл **Do-While** выполняет итерации лишь в том случае, когда проверка условия дает ответ **True**.
- Цикл **Do-Until** выполняет итерации лишь в том случае, когда проверка условия дает ответ **False**.
- Цикл **For-Each** выполняет итерации для просмотра всех элементов в массиве.

Указав величину шага, вы можете увеличить счетчик для каждой выполняемой итерации. Например, **For i = 1 To 10 Step 2** будет считаться 1, 3, 5, 7, 9.

Возможно, наиболее интересный цикл — **For-Next** со следующим синтаксисом:

For ***счетчик*** **=** ***начало*** **To** ***конец***

выполняемые-инструкции
выполняемые-инструкции
выполняемые-инструкции

Next ***счетчик***

С самого начала счетчик имеет «начальное» значение выполняемых итераций, сделанных циклом. Для этого используется переменная целочисленного типа, которая обозначается буквой ***i***. С помощью кода **Next** в конце каждой выполняемой итерации значение счетчика увеличивается на единицу. При выполнении каждой последующей итерации значение счетчика проверяется. Это нужно для того,

чтобы удостовериться, что оно меньше значения, указанного после ключевого слова **To**. Пока проверка условия выдает ответ **True**, выполнение итераций продолжается. Как только проверка условия выдаст ответ **False**, цикл сразу же завершится.

Циклы могут быть вложены друг в друга: так обеспечивается выполнение всех итераций внутреннего и внешних циклов.

ForNext.xlsm

1 Начните модуль макроса VBA с подпрограммы, которая объявляет две переменные, выбранные в качестве счетчики цикла.

```
Sub ForNext()

Dim i As Integer
Dim j As Integer

' Код цикла вставлять сюда (Шаг 2).

End Sub
```

В этом примере показано преимущество метода **Cells()** — он позволяет использовать значения переменных для аргументов строки и столбца.

2 Теперь добавьте код цикла, который отобразит значение счетчиков в цветной строке.

```
For i = 1 To 3

Cells(1, i).Interior.Color = vbRed
Cells(1, i).Font.Color = vbWhite
Cells(1, i) = "Outer Loop " & i

' Код цикла вставлять сюда (Шаг 3).

Next i
```

3 Теперь добавьте вложенный код цикла, который отобразит собственное значение счетчика в столбце.

```
For j = 1 To 10

Cells((j+1), i) = "Inner Loop " & j

Next j
```

Аргумент строк во вложенном цикле пронумерован от 2 до 11. При выполнении каждой итерации к значению счетчика аргумента прибавлялась единица.

4 На рабочий лист добавьте кнопку для выполнения макроса. Нажмите ее для просмотра значений циклов.

	A	B	C	D
1	Outer Loop 1	Outer Loop 2	Outer Loop 3	
2	Inner Loop 1	Inner Loop 1	Inner Loop 1	
3	Inner Loop 2	Inner Loop 2	Inner Loop 2	Run ForNext Macro
4	Inner Loop 3	Inner Loop 3	Inner Loop 3	
5	Inner Loop 4	Inner Loop 4	Inner Loop 4	
6	Inner Loop 5	Inner Loop 5	Inner Loop 5	
7	Inner Loop 6	Inner Loop 6	Inner Loop 6	
8	Inner Loop 7	Inner Loop 7	Inner Loop 7	
9	Inner Loop 8	Inner Loop 8	Inner Loop 8	
10	Inner Loop 9	Inner Loop 9	Inner Loop 9	
11	Inner Loop 10	Inner Loop 10	Inner Loop 10	

Выполнение циклов

Цикл **Do-While** — это альтернативный вариант цикла **For-Next**, который описан в предыдущем примере. Для цикла **Do-While** потребуется проверочное выражение, и он будет выполнять итерации до тех пор, пока значение выражения остается **True**. Тело цикла также должно иметь средство обновления, которое в определенный момент изменит значение выражения с **True** на **False**. В противном случае будет создан бесконечный цикл, который заблокирует программу.

Убедитесь в том, что проверочное выражение в цикле **Do-While** станет **False**. Если значение проверочного выражения всегда будет оставаться **True**, то создастся бесконечный цикл.

Каждый цикл **Do-While** должен заканчиваться ключевым словом **Loop**, так что синтаксис будет выглядеть так:

Do While ***тестируемое-выражение***

выполняемые-инструкции
средство-обновления

Loop

Также можно использовать несколько иной вариант синтаксиса. Тогда цикл всегда будет выполнять хотя бы одну итерацию указанных инструкций. В данном случае проверочное выражение нужно поставить в конец конструкции цикла:

```
Do

    выполняемые-инструкции
    средство-обновления

Loop While тестируемое-выражение
```

Do-Until — еще одна альтернатива циклу **For-Next**. Он во многом похож на цикл **Do-While**, однако он продолжит выполнение инструкций до тех пор, пока проверочное выражение имеет значение **False**. Тело цикла также должно содержать средство обновления, которое изменит значение проверочного выражения на **True**. В ином случае будет создан бесконечный цикл, который заблокирует программу.

Циклы **Do-While** и **Do-until** не имеют никаких различий, поэтому выбор одного из циклов — дело вкуса.

Каждый цикл **Do-Until** тоже должен заканчиваться ключевым словом **Loop**, так что синтаксис будет выглядеть так:

```
Do Until тестируемое-выражение

    выполняемые-инструкции
    средство-обновления

Loop
```

Проверочное выражение можно поставить и в конец цикла **Do-Until**, как и в **Do-While**. В любом из них средство обновления изменяет числовое значение, появляющееся в проверочном выражении. Именно так высчитывается количество итераций, необходимых для выполнения в этом цикле.

1. Начните модуль макроса VBA с подпрограммы, которая объявит переменную счетчика и переменную массива.

DoWhile.xlsm

```
Sub DoWhile()

    Dim i As Integer
    Dim Nums(1 To 3) As Integer

    ' Сюда помещаются инструкции (Шаги 2–4).

    End Sub
```

В этом примере показано преимущество нумерации элементов массива в соответствии с номерами строк.

Первый цикл здесь увеличивает счетчик до тех пор, пока значение не достигнет четырех. Второй цикл уменьшает счетчик до тех пор, пока значение не достигнет нуля.

2 Добавьте код для инициализации значений переменных.

```
i = 1
Nums(1) = 100
Nums(2) = 200
Nums(3) = 300
```

3 Теперь добавьте код цикла, который проведет прямой счет до трех.

```
Do While i < 4
Cells(i, 1).Interior.Color = vbYellow
Cells(i, 1) = "Iteration " & i
Cells(i, 2) = Nums(i)
i = i + 1
Loop
```

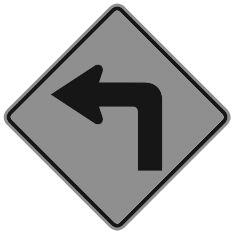
4 Затем добавьте код цикла, который посчитает до трех в обратном порядке.

```
Do
i = i — 1
Cells(i, 3).Interior.Color = vbCyan
Cells(i, 3) = "Iteration " & i
Cells(i, 4) = Nums(i)
Loop While i > 1
```

5 На рабочий лист добавьте кнопку для выполнения макроса. Нажмите ее для просмотра значений цикла.

	A	B	C	D	E
1	Iteration 1	100	Iteration 1	100	
2	Iteration 2	200	Iteration 2	200	Run DoWhile Macro
3	Iteration 3	300	Iteration 3	300	
4					

Прерывание циклов



Если указанное в коде условие было выполнено, для преждевременного завершения цикла вы можете использовать ключевое слово **Exit**. В инструкции **Exit** нужно указать конструкцию, выполнение которой следует завершить. Например, **Exit For**.

Оператор **Exit** располагается внутри блока инструкций цикла. Перед ним должна быть выполнена проверка условия. Если получено значение **True**, то выполнение цикла завершается, и программа переходит к выполнению другой задачи. Например, в случае получения значения **True**, интерпретатор переходит к выполнению «внешнего» цикла.

GoToExit.xlsm

1 Начните модуль макроса VBA с подпрограммы, которая очистит блок ячеек, а затем объявит две переменные.

```
Sub GoToExit()

Range("A1: C3").Clear

Dim i As Integer
Dim j As Integer
' Сюда помещаются инструкции (Шаги 2 и 3).

End Sub
```

2 Теперь добавьте код для инициализации переменных.

```
i = 1
j = 1
```

3 Затем добавьте вложенные циклы, которые отобразят их значения счетчика в очищенных ячейках.

```
For i = 1 To 3
  For j = 1 To 3

    ' Сюда помещаются инструкции (Шаги 5 и 7).

    Cells(i, j) = "Running i=" & i & " j=" & j
    ' Метку вставлять сюда (Шаг 6).
```

Метод **Clear** удаляет все содержимое определенного ряда ячеек.

```
    Next j
Next i
```

4 На рабочий лист добавьте кнопку для выполнения макроса. Нажав на нее, вы увидите, что в ячейках отобразились значения счетчика.

	A	B	C	D
1	Running i=1 j=1	Running i=1 j=2	Running i=1 j=3	
2	Running i=2 j=1	Running i=2 j=2	Running i=2 j=3	Run GoToExit Macro
3	Running i=3 j=1	Running i=3 j=2	Running i=3 j=3	

В этом примере оператор **Exit** прерывает выполнение второй и третьей итерации вложенного цикла, при этом внешний цикл пытается запустить внутренний цикл во второй раз.

5 Теперь добавьте показанную ниже инструкцию **Exit** в самое начало вложенного цикла, чтобы завершить внутренний цикл. Снова нажмите кнопку и посмотрите, как изменятся значения.

```
If(i = 2 And j = 2) Then
Cells(i, j).Font.Color = vbRed
Cells(i, j) = "Exit At i=" & i & " j=" & j
Exit For
End If
```

	A	B	C	D
1	Running i=1 j=1	Running i=1 j=2	Running i=1 j=3	
2	Running i=2 j=1	Exit At i=2 j=2		Run GoToExit Macro
3	Running i=3 j=1	Running i=3 j=2	Running i=3 j=3	

Раньше оператор **GoTo** очень часто использовался неправильно, и в обычных случаях лучше к нему не обращаться. Однако показанный здесь способ его применения для прерывания итераций цикла допустим.

Ключевое слово **GoTo** можно использовать для пропуска одной итерации цикла. При написании ключевого слова программа сразу переходит к метке, написанной в самом конце цикла.

Метка — это допустимое имя идентификатора, после которого всегда должно быть добавлено двоеточие:. С помощью имени идентификатора оператор **GoTo** указывает макросу, какую строчку кода нужно выполнять следующей.

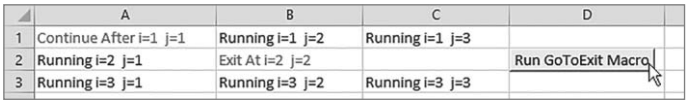
Оператор **GoTo** находится внутри блока инструкций цикла. Перед этим оператором всегда должна стоять проверка условия. Если проверка условия выдает ответ **True**, то одна итерация перестает выполняться.

6 Добавьте показанную ниже метку в конце вложенного цикла.

```
Continue:
```

7 Добавьте оператор **GoTo** в начало вложенного цикла — так вы пропустите выполнение первой итерации цикла. Затем снова запустите макрос и посмотрите на изменения.

```
If(i = 1 And j = 1) Then
Cells(i, j).Font.Color = vbRed
Cells(i, j) = "Continue After i=" & i & " j=" & j
GoTo Continue
End If
```



	A	B	C	D
1	Continue After i=1 j=1	Running i=1 j=2	Running i=1 j=3	
2	Running i=2 j=1	Exit At i=2 j=2		Run GoToExit Macro
3	Running i=3 j=1	Running i=3 j=2	Running i=3 j=3	

В данном случае оператор **GoTo** пропускает выполнение первой итерации вложенного цикла, в то время как внешний цикл пытается запустить его в первый раз.

Итерирование циклов

В дополнение к циклам **For-Next**, **Do-While** и **Do-Until**, описанным ранее в этой главе, в VBA есть специальный цикл **For-Each**, который полезен при работе с коллекциями. В отличие от других циклов, ему не требуется счетчик, потому что он выполняет итерацию всех элементов коллекции, пока не достигнет последнего: например, все листы в книге или все ячейки в определенном диапазоне.

Все объекты в Excel подчиняются строгой иерархии. Многие объекты также имеют коллекции. Подробнее в главе 2.

Для выполнения цикла **For-Each** необходима переменная типа объекта элемента, которая при выполнении итерации будет содержать каждый элемент. Коллекция указывается для ключевого слова **In** с использованием следующей конструкции:

For Each ***элемент*** **In** ***коллекция***

выполняемые-инструкции
выполняемые-инструкции
выполняемые-инструкции

Next ***элемент***

Как и другие циклы, цикл **For-Each** может быть вложенным. Таким образом он сможет выполнить

итерацию каждой ячейки в пределах указанного диапазона на каждом листе в книге:

ForEach.xlsm

1. Создайте новую книгу с рабочими листами, отображающими два квартала года с названиями месяцев и суммами в ячейках.

	A	B	C
1	January	February	March
2	$1,000.00	-$500.00	$800.00
3			

Q1 Q2

	A	B	C
1	April	May	June
2	-$250.00	$1,200.00	$800.00
3			

Q1 Q2

2. Начните модуль макроса VBA с подпрограммы, которая объявит две объектные переменные.

```
Sub ForEach()

Dim Sheet As Worksheet
Dim Cell As Range

' Код цикла вставлять сюда (Шаг 3).

End Sub
```

3. Теперь добавьте цикл для выполнения итерации на каждом рабочем листе.

```
For Each Sheet In ActiveWorkbook.Worksheets

' Код цикла вставлять сюда (Шаги 4 и 5).

Next Sheet
```

4. Затем добавьте вложенный цикл для выполнения итерации диапазона ячеек для смены регистра названия каждого месяца.

```
For Each Cell In Sheet.Range("A1: C1")
Cell.Value = UCase(Cell.Value)
Next Cell
```

Вы можете использовать метод **LCase()**, чтобы изменить регистр на нижний. А метод **WorksheetFunction.Proper()** применяется для капитализации первой буквы в строке текста.

5 Добавьте вложенный цикл для выполнения итерации диапазона ячеек для смены цвета шрифта указанных сумм.

```
For Each Cell In Sheet.Range("A2: C2")

If Cell.Value < 0 Then
Cell.Font.Color = vbRed
Else
Cell.Font.Color = vbBlue
End If

Next Cell
```

6 На первом рабочем листе добавьте кнопку для выполнения макроса. Нажмите кнопку, чтобы выполнить все изменения.

Неважно, сколько у вас листов в книге, потому что цикл может применяться к любому числу — просто добавьте к данному примеру третий и четвертый кварталы.

Оператор with

Код в макросах VBA часто требует установки нескольких свойств для выбранного объекта, поэтому вы можете столкнуться с проблемой избыточного ввода кода. К примеру, вы захотите установить разные свойства для ячейки с помощью вот такого кода:

```
ActiveSheet.Cells(1, 1).Interior.ColorIndex = 6
ActiveSheet.Cells(1, 1).Font.ColorIndex = 21
ActiveSheet.Cells(1, 1).Font.Bold = True
```

Однако существует конструкция **With-End**, которая решает проблему избыточного ввода кода путем назначения общего объекта, так что вы можете написать:

Перед каждой конечной частью цепочки следует поставить точку.

```
With ActiveSheet.Cells(1, 1)
        .Interior.ColorIndex = 6
        .Font.ColorIndex = 21
        .Font.Bold = True
End With
```

Так вы можете назначить любой объект в иерархической цепочке. С помощью следующего кода устанавливаются свойства шрифта:

```
With ActiveSheet.Cells(1, 1).Font
        .ColorIndex = 21
        .Bold = True
End With
```

Из-за использования конструкции **With-End** код может стать более трудным для чтения, поэтому для удобства в конечных частях цепочек делаются отступы. Однако эта конструкция улучшает производительность, потому что в таком случае вы ссылаетесь на объект только один раз.

WithEnd.xlsm

1. Начните модуль макроса VBA с подпрограммы, которая очистит несколько ячеек, а затем объявит три переменные.

```
Sub GoToExit()

Range("A1: H7").Clear

Dim Row As Integer
Dim Column As Integer
Dim Index As Integer

' Сюда помещаются инструкции (Шаги 2 и 3).

End Sub
```

2. Добавьте код для инициализации двух переменных.

```
Row = 1
Column = 1
```

3. Добавьте цикл для установки атрибутов в 56 ячейках.

```
For Index = 1 To 56

With ActiveSheet.Cells(Row, Column)
        .ColumnWidth = 6.5
        .RowHeight = 25
        .Font.Bold = True
        .HorizontalAlignment = xlCenter
        .VerticalAlignment = xlCenter
        .Value = Index
        .Interior.ColorIndex = Index
        .Interior.TintAndShade = 0.3
End With

' Сюда помещаются инструкции (Шаг 4).

Next Index
```

Параметр **ColorIndex** — это цветовая палитра с номерами от 1 до 56. С помощью параметра **TintAndShade** можно изменять значение яркости/затемнения от –1 до 1.

4 Теперь добавьте код для сдвига на следующий столбец.

```
If(Column Mod 8 = 0) Then
Column = 1
Row = Row + 1
Else
Column = Column + 1
End If
```

Оператор деления по модулю **Mod** особенно полезен при работе со строками и столбцами. Больше информации см. в главе 4.

5 На рабочий лист добавьте кнопку для выполнения макроса. Нажмите ее. Вы увидите, что ячейки изменили цвет с помощью параметра **ColorIndex**.

	A	B	C	D	E	F	G	H	I	J
1	1	2	3	4	5	6	7	8		
2	9	10	11	12	13	14	15	16		
3	17	18	19	20	21	22	23	24		
4	25	26	27	28	29	30	31	32		
5	33	34	35	36	37	38	39	40	Run WithEnd Macro	
6	41	42	43	44	45	46	47	48		
7	49	50	51	52	53	54	55	56		
8										

Заключение

- Ключевые слова **If-Then** проводят проверку условия. Она проверяет логическое значение выбранного выражения.
- С помощью вложенной конструкции **If-Then** можно проверить несколько выражений. Конструкция всегда должна заканчиваться оператором **End If**.
- Конструкция **If-Then** может включать в себя ключевое слово **Else**. Таким образом появляется возможность выбрать инструкции, которые будут выполняться при условии, если проверка логического значения выражения выдаст ответ **False**.
- Конструкция **If-Then** может включать в себя ключевое слово **Elself**. Так будут выполняться альтернативные проверки условия.
- Конструкция **Select-Case** сопоставляет значение из ряда операторов **Case** и завершается оператором **End Select**.
- Цикл **For-Next** выполняет указанное количество итераций.
- Цикл **Do-While** выполняет итерации лишь в том случае, когда результат условного выражения **True**.
- Цикл **Do-Until** выполняет итерации лишь в том случае, когда условное выражение **False**.
- Цикл **For-Each** выполняет итерации для перебора всех элементов в массиве.
- Два вложенных цикла производят выполнение всех итераций вложенного, а затем внешнего цикла.
- Ключевое слово **Exit** можно использовать вместе с проверкой условия для полного выхода из цикла.

- Ключевое слово **GoTo** нужно для прекращения выполнения одиночной итерации цикла. В таком случае все выполняемые инструкции пропускаются.

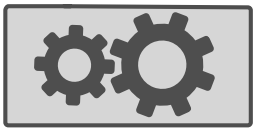
- Цикл **For-Each** выполняет итерацию каждого объекта в коллекции, указанной в операторе **In**.

- Конструкция **With-End** обозначает объект в иерархической цепочке. Таким образом вы избежите избыточного ввода кода.

6 Выполнение процедур

В этой главе вы узнаете, как управлять подпрограммами и решать ошибки в макросах VBA.

Вызов подпрограмм



Макрос VBA может содержать бесконечное количество инструкций, выполняющихся при каждом запуске макроса. Выполнение этих инструкций называется «процедурой». Существуют два вида процедур: один из них — функция — будет рассмотрен в следующей главе, а второму — подпрограмме — посвящена эта глава. Процедура «функция» оформляется с помощью ключевого слова **Function**, а процедура «подпрограмма» — с помощью ключевого слова **Sub**. Главное отличие между этими двумя процедурами заключается в том, что функция возвращает результат, а подпрограмма — нет.

Подробнее о доступности читайте в следующем разделе.

Обычно большие подпрограммы желательно разделить на несколько процедур, каждая из которых будет выполнять определенную задачу. В таком случае для каждой задачи вам нужно создать отдельную подпрограмму, а главная процедура будет активировать ее в нужный момент. Мы рекомендуем этот подход, потому что он упрощает код.

Отдельные подпрограммы создаются с помощью ключевого слова **Sub**, следом за которым нужно написать допустимое идентификационное имя. Обычно они доступны глобально. Это означает, что их можно использовать для выполнения инструкций с помощью ключевого слова **Call**, следом за которым нужно написать имя любой подпрограммы в рабочей книге:

CallMain.xlsm

1. Начните модуль макроса VBA с подпрограммы, которая будет вызывать другие процедуры подпрограммы:

```
Sub Main()

' Сюда помещаются инструкции (Шаг 5).

End Sub
```

2. Теперь добавьте подпрограмму, которая введет в ячейку актуальный день недели.

```
Sub GetDay()
```

```
Range("A1") = Format(Now, "dddd")

End Sub
```

3 Теперь добавьте подпрограмму, которая введет в ячейку актуальную дату.

```
Sub GetDate()

Range("B1") = Format(Now, "mmmm d, yyyy")

End Sub
```

4 Добавьте подпрограмму, которая введет в ячейку актуальное время.

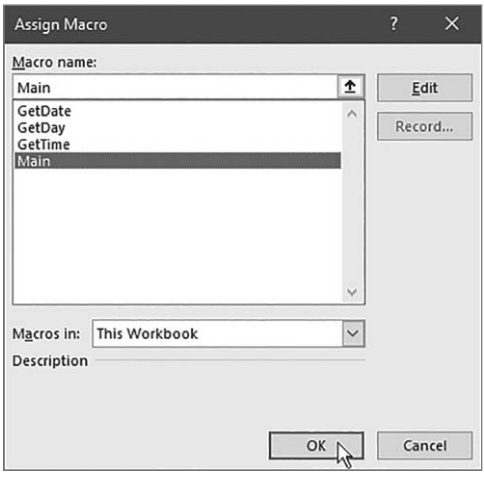
```
Sub GetTime()

Range("C1") = Format(Now, "hh: nn am/pm")

End Sub
```

5 Затем добавьте код для выполнения процедур в каждой подпрограмме.

```
Call GetDay
Call GetDate
Call GetTime
```

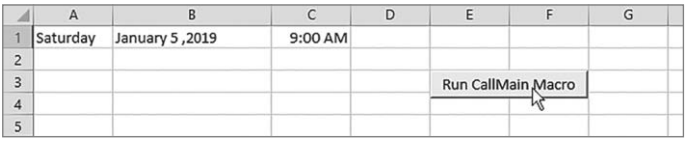
В инструкциях **Call** после написания имени подпрограммы ставить скобки не нужно.

6 На рабочий лист добавьте кнопку, в диалоговом окне **Assign Macro** (Назначить макрос объекту) выберите процедуру **Main** и нажмите **OK**.

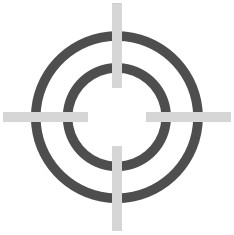


В диалоговом окне отображаются все публичные процедуры. Для выполнения вы можете назначить любую из них элементу **Button** формы.

7 Нажмите кнопку, чтобы главная подпрограмма выполнила «внутренние» процедуры для отображения времени и даты.



Изменение области видимости



Область видимости определяет уровень доступности процедур и переменных в коде макроса. Если в объявлении область видимости не указана, то по умолчанию для переменных она будет локальной, то есть они станут доступны только в той процедуре, в которой были объявлены. Если вы попробуете ссылаться на локальную переменную, объявленную в другой процедуре, то столкнетесь с ошибкой компиляции **Variable not defined** (Переменная не определена).

Можно создать переменную с более широкой областью видимости с помощью ключевого слова **Dim**. Объявлять ее нужно в разделе объявлений в самом начале модуля. Таким образом, переменная будет доступна в любой процедуре в этом модуле:

Dim *имя-переменной* As *тип-данных*

Большинство объявлений располагаются в процедурах. В разделе объявлений макроса находятся только директивы типа **Option Explicit** и объявления нелокальных переменных.

В качестве альтернативы вы можете создать переменную с глобальной областью видимости — просто объявите ее с помощью ключевого слова **Public** в разделе объявлений в самом начале модуля. Глобальная переменная будет доступна любой процедуре во всех других модулях рабочей книги:

Public *имя-переменной* As *тип-данных*

По возможности лучше избегать использования нелокальных переменных, потому что это может привести к конфликту, который локальная область видимости может предотвратить.

По умолчанию процедуры имеют глобальную область видимости, поэтому они доступны любой процедуре в любом модуле. Вы можете создать частную процедуру с ограниченной областью видимости — просто объявите ее с помощью ключевого слова **Private**:

Private *имя-процедуры*()

Частная процедура (**Private**) доступна только для процедур, располагающихся в том же модуле. Другие процедуры тоже могут ее вызвать, однако ее нельзя назначить элементу **Button** формы. Процедуры такого типа часто используются в программировании и рекомендуются для защиты вашего кода:

AccessScope.xlsm

1. Начните модуль макроса VBA с объявления двух переменных в начале кода.
```
Dim ModuleVar As Integer
Public GlobalVar As Integer
```

2. Добавьте процедуру, которая инициализирует каждую переменную.
```
Sub Main()

ModuleVar = 4
GlobalVar = 16
' Сюда помещается инструкция (Шаг 4).

End Sub
```

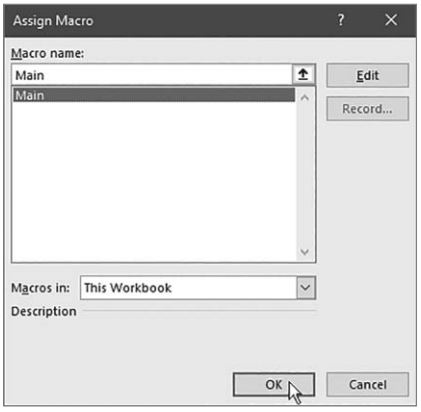
3. Затем добавьте другую процедуру, которая будет ссылаться на оба значения переменных для отображения итогового значения.
```
Private Sub ModuleProcedure()

ActiveCell.Value = ModuleVar * GlobalVar

End Sub
```

4. Теперь добавьте код для выполнения описанной выше процедуры.
```
Call ModuleProcedure
```

5. На рабочий лист добавьте кнопку, затем в диалоговом окне **Assign Macro** (Назначить

В этом примере каждая процедура ссылается на нелокальные переменные, однако вы можете использовать локальные переменные и их значения. Просто применяйте их в качестве аргументов при вызове других процедур. Подробнее см. далее в этой главе.

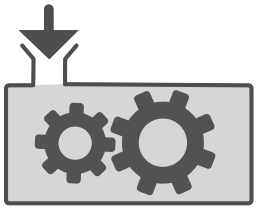
макрос объекту) выберите процедуру **Main** и нажмите кнопку **OK**.



Обратите внимание, что в диалоговом окне не отображаются частные процедуры с областью видимости **Private**. Именно поэтому вы не сможете выполнить их с помощью элемента **Button** формы.

6 Нажмите кнопку, чтобы главная подпрограмма выполнила «внутренние» локальные процедуры для отображения итогового значения.

Передача аргументов



Процедуры VBA могут быть созданы для «приема» аргументов. Их значения пригождаются инструкциям этих процедур. Аргументы могут быть любого типа: переменной, константой, выражением или литеральным значением (число или строка текста).

Важно понимать, что вы можете передать аргументы процедуре двумя способами:

- **По ссылке —** этот способ используется по умолчанию. Передача переменной в памяти создает ее адрес. Это означает, что любые изменения действительны для исходной переменной.
- **По значению —** менее распространенный способ. Передается только копия исходной переменной, а это значит, что изменения не будут действительны для оригинала.

В этом разделе показана передача аргументов по значению. См. подробнее в следующем разделе.

Для создания принимающей аргумент процедуры при объявлении задайте имя идентификатора для аргумента в круглых скобках:

Sub *имя-процедуры(имя-аргумента)*

Вы также можете ссылаться на аргумент внутри процедуры — просто используйте его имя идентификатора.

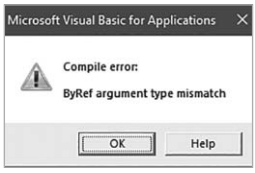
По умолчанию тип аргумента будет **Variant**, который принимает любой тип данных. Во избежание проблем рекомендуется указывать допустимый тип данных в объявлении:

Sub *имя-процедуры*(*имя-аргумента* As *тип-данных*)

Вы можете указать несколько аргументов в виде списка, разделяя их запятыми. Аргументы могут иметь разные типы данных:

Sub *имя-процедуры*(*арг1* As *тип*, *арг2* As *тип*, *арг3* As *тип*)

В таком случае вызывающая инструкция должна передать нужное количество аргументов, и каждый аргумент должен иметь правильный тип данных. Попытка передать аргумент не того типа данных приведет к ошибке компиляции **ByRef argument type mismatch** (Несоответствие типа аргумента ByRef).



1 Начните модуль макроса VBA с подпрограммы, которая объявит и инициализирует две переменные.

FirstArgs.xlsm

```
Sub Main()

Dim Number As Integer
Dim Factor As Integer

Number = 4
Factor = 16

' Сюда помещаются инструкции (Шаг 3).

End Sub
```

Имена аргументов могут не совпадать с именами передаваемых переменных. Вы можете дать им любое имя, однако с одинаковыми именами намного легче отслеживать значения.

2 Теперь объявите частную процедуру с областью видимости **Private**, которая должна будет принять только два аргумента типа **Integer** для ссылки на инструкцию, которая отобразит результат.

```
Private Sub Multiply(Number As Integer, Factor As
Integer)

ActiveCell.Value = Number * Factor

End Sub
```

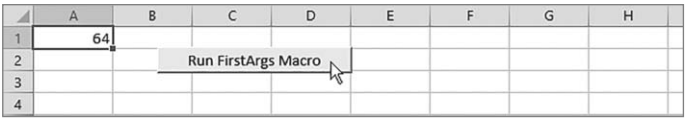
3 Теперь добавьте код для вызова процедуры, написанной выше, передавая аргументы с правильным типом данных по ссылке.

```
Call Multiply(Number, Factor)
```

Этот пример аналогичен предыдущему, однако в нем не используются нелокальные переменные.

4 На рабочий лист добавьте кнопку, затем в диалоговом окне **Assign Macro** (Назначить макрос объекту) выберите процедуру **Main** и нажмите кнопку **OK**.

5 Нажмите кнопку, чтобы главная подпрограмма выполнила «внутренние» процедуры для отображения итогового значения.



Добавление модулей

В проекте VBA может быть бесконечное количество модулей.

Многие программисты для удобства предпочитают создавать процедуры в отдельных модулях. Вы можете добавить модули с помощью меню редактора Visual Basic, выбрав команду меню **Insert** ⇨ **Module** (Вставить ⇨ Модуль) или на панели **Project Explorer** (Обозреватель проектов), щелкнув правой кнопкой мыши и выбрав пункт **Insert** ⇨ **Module** (Вставить ⇨ Модуль) в контекстном меню.

Процедуры, созданные в стандартном модуле VBA (например **Module2**), доступны в этом модуле и в первом стандартном модуле (то есть **Module1**),

так как такие процедуры глобальны по умолчанию. Даже если вы не объявите их вместе с ключевым словом **Private**, они не будут отображаться в списке диалогового окна **Assign Macro** (Назначить макрос объекту).

Процедуры в добавленных модулях могут быть вызваны так же, как и любые другие процедуры, но только в том случае, если они не были объявлены с ключевым словом **Private**. Вы можете создать процедуры для принятия аргументов по ссылке или по значению из любого модуля в проекте.

Во избежание ошибок важно понимать разницу между передачей аргументов по ссылке и по значению. В процедуре аргумент переменной, переданной по значению, можно изменять по-разному, при этом значение исходной переменной не изменится. Иногда стоит воспользоваться этим способом, потому что при передаче аргумента по ссылке все изменения затрагивают не только аргумент переменной, но и значение исходной переменной. Чтобы передать аргумент по значению, в объявлении добавьте ключевое слово **ByVal** перед именем аргумента:

1. Начните модуль макроса VBA с подпрограммы, которая объявит и инициализирует переменную.

```
Sub Main()

Dim Number As Integer
Number = 4

' Сюда помещаются инструкции (Шаги 5–7).

End Sub
```

2. Затем выберите команду меню **Insert ⇨ Module** (Вставить ⇨ Модуль) и добавьте второй модуль.

3. Начните второй модуль с объявления процедуры, которая отобразит результат, получившийся при передаче аргумента по значению.

```
Sub CubeByVal(ByVal Number As Integer)
Number = (Number * Number * Number)
```

AddModule.xlsm

Ключевое слово **ByVal** применяется только к одному аргументу! Если вы хотите передать несколько аргументов по значению, вам нужно использовать ключевое слово **ByVal** перед каждым аргументом.

```
Range("D1: E1") = Array("Cubed Value:", Number)

End Sub
```

4 Во второй модуль добавьте такую же процедуру, которая теперь отобразит результат, получившийся при передаче аргумента по ссылке.

```
Sub CubeByRef(Number As Integer)

Number = (Number * Number * Number)
Range("D3: E3") = Array("Cubed Value:", Number)

End Sub
```

5 Вернитесь к первому модулю и добавьте код, позволяющий дважды отобразить значение исходной переменной.

```
Range("A1: C1") =_
  Array("ByVal", "Initial Var Value:", Number)
Range("A3: C3") =_
  Array("ByRef", "Initial Var Value:", Number)
```

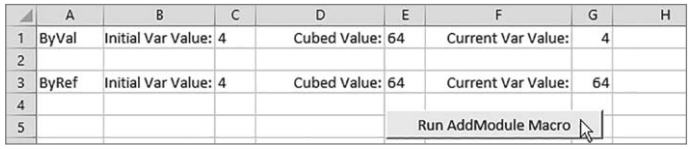
6 Добавьте вызов для передачи переменной по значению, а затем отобразите ее значение после изменений.

```
Call CubeByVal(Number)
Range("F1: G1") = Array("Current Var Value:", Number)
```

7 Затем добавьте вызов для передачи переменной по ссылке, а затем отобразите ее значение после изменений.

```
Call CubeByRef(Number)
Range("F3: G3") = Array("Current Var Value:",
Number)
```

8 На рабочий лист добавьте кнопку для запуска макроса, затем нажмите ее для сравнения значений передачи переменной по значению и по ссылке.



Не забывайте, что по умолчанию аргументы передаются по ссылке, так что любые изменения отразятся на значении исходной переменной.

Сохранение значений

Локальные переменные, располагающиеся внутри процедуры, существуют только тогда, когда процедура выполняет свои инструкции. При ее завершении переменная не сохраняет содержащееся в ней значение. Если вы хотите, чтобы значение после завершения процедуры сохранялось, объявите ее с ключевым словом **Static**, а не **Dim**:

Static ***имя-переменной*** **As** ***тип-данных***

Если вы хотите создать процедуру, в которой после завершения все переменные будут сохранять содержащееся в них значение, то объявите процедуру с помощью ключевого слова **Static**:

Static Sub ***имя-процедуры(список-аргументов)***

Обычно тип процедуры, созданной с целью сохранения значений переменных, может передавать значения аргументов для изменения сохраненного значения, но только при том условии, что нужная книга открыта. Однако при закрытии книги переменные потеряют измененные значения.

При объявлении процедуры вы можете указать аргументы с помощью ключевого слова **Optional**, которое позволит вызывающей стороне указывать или опускать значение аргумента при вызове процедуры. В таком случае нужно указать значение аргумента по умолчанию, которое будет использовать процедура, потому что вызывающая сторона опустит значение аргумента:

Каждый аргумент в списке аргументов, стоящий после **необязательного** аргумента, также должен быть **необязательным**. Рекомендуется объявлять **необязательные** аргументы в конце списка.

Static Sub ***имя-процедуры***(**Optional** ***имя-аргумента*** **=** ***значение-по-умолчанию***)

Если вы укажете ноль в значении аргумента по умолчанию, то «вызывающая» сторона получит доступ к числовому значению аргумента и сможет изменить сохраненное значение:

FixValue.xlsm

1 Начните модуль макроса VBA с основной подпрограммы.

```
Sub Main()

' Сюда помещается инструкция (Шаг 7).

End Sub
```

2 Теперь добавьте еще одну подпрограмму, которая сохранит значения переменных и примет один аргумент.

```
Static Sub FixValue(Optional Number As Integer =
0)

' Сюда помещаются инструкции (Шаги 3–6).

End Sub
```

3 Во вторую подпрограмму добавьте объявления двух переменных, которые сохранят свои значения.

```
Dim Count As Integer
Dim Total As Integer
```

4 Затем добавьте код, который будет увеличивать значение первой переменной на один каждый раз, когда она будет вызываться.

```
Count = Count + 1
```

5 Теперь добавьте код для увеличения значения второй переменной каждый раз, когда она будет передаваться вызывающей стороной.

```
If(Number > 0) Then
Total = Total + Number
End If
```

6 Добавьте код для отображения количества раз, сколько вызывалась процедура, и конечного сохраненного значения.

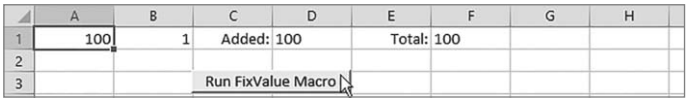
```
Range("B1: F1") = _
    Array(Count, "Added:", Number, "Total:", Total)
```

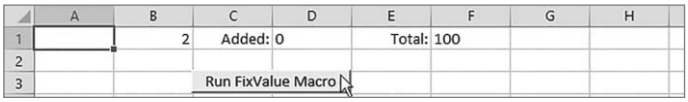
Вы можете объявить процедуру с помощью ключевого слова **Static**, а переменные — с помощью **Dim**. Вы также можете объявить процедуру без использования ключевого слова **Static**, а переменные — с помощью **Static** вместо **Dim**.

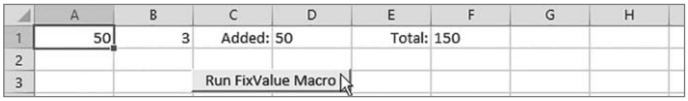
7. Вернемся к первой подпрограмме. Добавьте показанный ниже код для вызова второй подпрограммы и передачи значения.

```
Call FixValue(Range("A1").Value)
```

8. На рабочий лист добавьте кнопку для вызова макроса, затем введите значения в ячейку **A1** и нажмите кнопку, чтобы увидеть конечный результат.

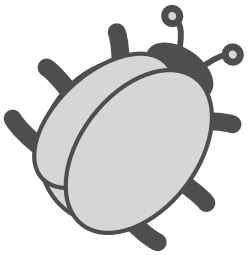






Если в окне редактора Visual Basic вы нажмете кнопку **Stop** (Стоп), то макрос сбросится, и вы утеряете все значения переменных.

Отладка кода



Бывают моменты, когда лучше детально изучить работу программы, разбирая все по строчкам и находя любые ошибки. В редакторе Visual Basic вы можете поставить точку останова, с помощью которой можно остановить выполнение процедуры на определенной строке. В меню **Debug** (Отладка) вы можете выбрать пункт **Step Into** (Шаг вперед) и переходить по строчке за один раз. Когда вы отлаживаете код, то можете открыть панель **Locals window** (Окно локальных переменных) для отслеживания значения отдельных переменных, например, во вложенном цикле:

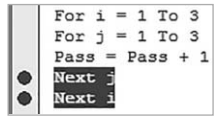
1. Начните модуль макроса VBA с подпрограммы, которая объявит три переменных и будет содержать вложенные циклы.

```
Sub Main()

Dim i As Integer
Dim j As Integer
```

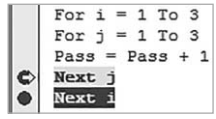
```
Dim Pass As Integer
For i = 1 To 3
For j = 1 To 3
Pass = Pass + 1
Next j
Next i

End Sub
```

При установке точки останова на поле появляется точка. Вы можете нажать на нее, чтобы удалить. Желтые стрелки и выделенные строки кода указывают текущее положение интерпретатора.

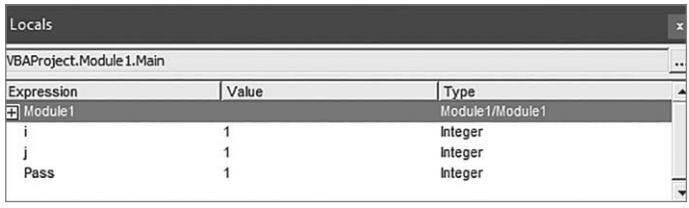
2 В окне для ввода кода щелкните мышью по серому полю напротив каждой строки, содержащей ключевое слово **Next**, для создания двух точек останова.



3 Нажмите кнопку **Start** (Запуск). Программа будет выполняться до первой точки останова.



4 Выберите команду меню **View ⇨ Locals Window** (Вид ⇨ Окно локальных переменных) для открытия соответствующей панели. Обратите внимание на значение каждой переменной.



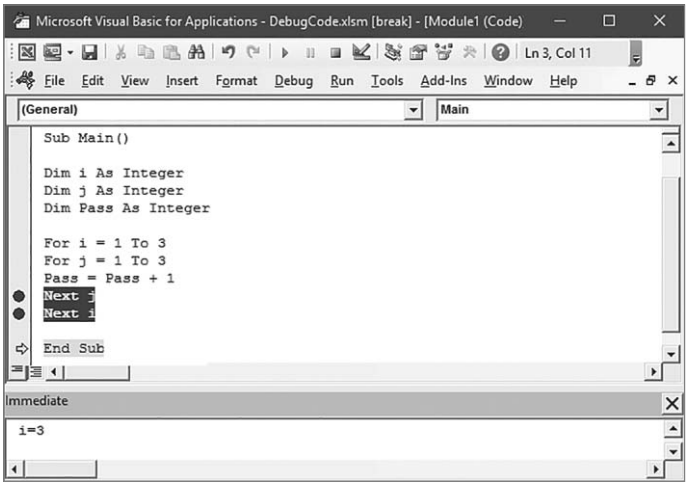
5 Каждый раз, когда вы будете нажимать кнопку **Start** (Запуск), значения переменных будут меняться. Продолжайте, пока не дойдете до третьей внешней инструкции **Next**.

Ошибки программирования очень часто называют багами, а процесс их отслеживания — отладкой.

6 Используйте **Debug ⇨ Step Into** (Отладка ⇨ Шаг вперед), чтобы дойти до конца подпрограммы.

Для выхода из цикла в конце подпрограммы значение каждой переменной счетчика

превысило верхний предел, установленный в инструкциях **For**. Всего было выполнено девять итераций (3 на 3).



7 Для завершения нажмите кнопку **Stop** (Стоп), а затем еще раз нажмите кнопку **Start** (Запуск), чтобы программа выполнилась до первой точки останова.

8 Выберите команду меню **View ⇨ Immediate Windows** (Вид ⇨ Окно непосредственной отладки).

9 На открывшейся панели введите **i = 3** и нажмите клавишу **Enter**. Вы сразу же перейдете на начало третьего внешнего цикла.

10 Выбирайте команду меню **Debug ⇨ Step Into** (Отладка ⇨ Шаг вперед), чтобы пошагово провести итерации в цикле до конца.

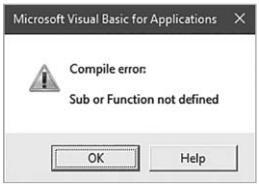
На панели **Locals window** (Окно локальных переменных) вы можете увидеть все переменные в той области видимости, в какой они выполняются в макросе.

Любой код, который вы пишете на панели **Immediate Window** (Окно непосредственной отладки), применяется к отлаживаемой программе, но не изменяет ее код. Попробуйте ввести **MsgBox i + j** на панели **Immediate Window** (Окно непосредственной отладки) и нажать клавишу **Enter**.

Перед началом работы проверьте настройки в группе элементов управления **Error Trapping** (Перехват ошибок) редактора Visual Basic. В меню редактора выберите команду **Tools** ⇨ **Options** (Сервис ⇨ Параметры) и на вкладке **General** (Общее) установите переключатель в положение **Break on Unhandled Errors** (Прерывать на необработанных ошибках).

Обработка ошибок

Редактор Visual Basic помогает находить синтаксические ошибки в коде. При запуске макроса у вас появится диалоговое окно, в котором будет описан характер ошибки, а сам макрос не будет выполнен:



Так как выполнение макроса прервано, программист должен исправить обнаруженную ошибку до того, как макрос будет выполнен до конца. То же самое происходит и при возникновении ошибки во время выполнения программы: VBA останавливает процесс и выдает диалоговое окно с описанием характера ошибки.

Разработчик макроса должен понимать, как во время выполнения программы появляется ошибка, и продумать, что предпринять для ее исправления. Если на ваш взгляд ошибка не столь важна, то позвольте VBA игнорировать ошибку. Чтобы диалоговое окно с ошибкой больше не отображалось, в начале процедуры добавьте показанный ниже код:

```
On Error Resume Next
```

В качестве альтернативы пользовательский обработчик ошибок можно указать в самом конце процедуры, после инструкции **Exit Sub**. Это приведет к пропуску выполнения процедуры без отображения стандартного диалога вывода ошибок. Если вы хотите использовать собственный обработчик ошибок, то добавьте показанный ниже код в начало процедуры:

```
On Error GoTo ErrorHandler
```

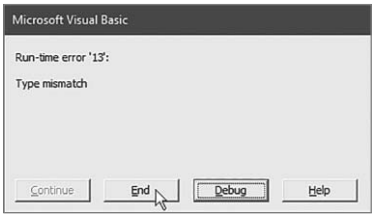
Ваш обработчик ошибок получит стандартное описание ошибки благодаря свойству **Description** объекта **Err**, однако вы можете использовать собственное описание:

По умолчанию метка называется **ErrorHandler**, однако вы можете назвать ее как угодно.

1 Начните модуль макроса VBA с подпрограммы, которая объявит и инициализирует переменную, а затем отобразит результат.

```
Sub Main()

' Statement to be inserted here (Steps 3 & 5).

Dim Number As Double
Number = Range("B1")
Range("C1: D1") = Array("Tripled:",(Number * 3))

' Statement to be inserted here (Step 6).

End Sub
```

2 На рабочий лист добавьте кнопку для выполнения макроса, затем в ячейку **B1** введите нечисловое значение и нажмите кнопку. Появится окно ошибки.



3 Нажмите кнопку **End** (Завершить), чтобы закрыть окно. Добавьте показанный ниже код в начало процедуры.

```
On Error Resume Next
```

4 Снова введите нечисловое значение в ячейку **B1** и запустите макрос. Процедура выполнится, но результат будет не тот, который вы ожидали.

	A	B	C	D
1	Please Enter A Number To Triple :	Fifty	Tripled:	0
2				
3			Run ErrorHandler Macro	

5 Измените код, который вы вставили в начало кода в шаге 3 вот так:

```
On Error GoTo ErrorHandler
```

Обратите внимание на то, что в обычном диалоговом окне ошибки есть кнопка для ее отладки, и это не лучшее решение со стороны разработчиков.

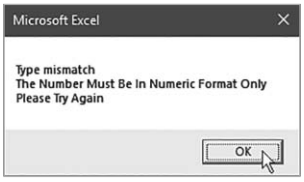
6 Теперь в конец процедуры добавьте пользовательский обработчик ошибок перед строкой **End Sub**.

```
Exit Sub
ErrorHandler: MsgBox Err.Description & vbNewLine _
& "The Number Must Be In Numeric Format Only" _
& vbNewLine & "Please Try Again"
```

Если вы хотите отображать собственное описание ошибки, то вам не нужно отображать стандартное описание.

7 Снова введите нечисловое значение в ячейку **B1** и выполните макрос. Вы увидите собственное сообщение об ошибке.

8 В ячейку **B1** введите числовое значение, выполните макрос. Процедура выполнится и выдаст ожидаемый результат.



	A	B	C	D
1	Please Enter A Number To Triple :	50	Tripled:	150
2				
3			Run ErrorHandler Macro	

Заключение

- Подпрограмму можно объявить с помощью ключевого слова **Sub**, а также вызвать с помощью ключевого слова **Call**, следом за которым нужно написать имя подпрограммы.

- По умолчанию все переменные локальны, а процедуры глобальны.

- В разделе объявления вы можете создать переменные уровня модуля с помощью ключевого слова **Dim** и глобальные переменные с помощью ключевого слова **Public**.

- С помощью ключевого слова **Private** можно ограничить область видимости подпрограммы пределами модуля.

- Изменения аргументов, переданных по ссылке, *влияют* на значение исходной переменной; изменения аргументов, переданных по значению, *не влияют* на значение исходной переменной.

- Ключевое слово **As** можно использовать для указания типа данных аргумента.

- При объявлении процедуры вы можете включить несколько аргументов в круглые скобки — просто разделите их запятыми.

- Ключевое слово **ByVal** позволяет передавать аргумент по значению.

- Значение, содержащееся в локальной переменной, при завершении процедуры утрачивается. Для сохранения значения переменную нужно объявлять **статической** (со словом **Static**).

- При объявлении процедуры может использоваться ключевое слово **Static**, которое гарантирует сохранение значений всех переменных после завершения процедуры.

- Ключевое слово **Optional** можно использовать при объявлении процедуры — тогда вызывающая сторона пропустит аргумент.

- Необязательные аргументы должны иметь значение по умолчанию. Обычно их указывают в конце списка аргументов.

- При отладке ошибок точки останова позволяют рассмотреть код построчно.

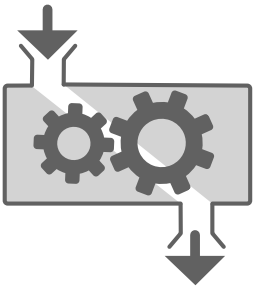
- Для игнорирования ошибок в начале процедуры вы можете написать строку **On Error Resume Next**.

- В начало процедуры вы можете вставить код **On Error GoTo**. При обнаружении ошибок он передает управление пользовательскому обработчику ошибок, который добавляется после инструкции **Exit Sub**.

7 Использование функций

В этой главе мы обсудим, как работать с многократно используемыми функциями в макросах VBA.

Определение функции



Пользовательская функция в макросах VBA — это процедура, похожая на подпрограмму. К примеру, они одинаково принимают аргументы. Однако в отличие от подпрограммы функция всегда возвращает значение аргумента вызывающей стороне.

Пользовательские функции создаются при помощи ключевого слова **Function**, следом за которым нужно написать имя функции, соответствующее соглашению об именах. После имени функции ставятся круглые скобки, в которые можно поместить список аргументов. Вы также можете указать тип данных возвращаемого значения с помощью ключевого слова **As**. Функция всегда закрывается ключевыми словами **End Function**, при этом возвращаемое значение должно быть присвоено имени функции внутри конструкции.

Очень часто пользовательские функции называют определяемыми пользователем функциями.

Синтаксис выглядит так:

Function ***имя-функции(список-аргументов)*** **As** ***тип-данных***
выполняемые-инструкции
имя-функции = значение
End Function

Как вы понимаете, в коде VBA функции могут быть вызваны из других процедур. Возвращаемое функцией значение может использоваться в вызывающей процедуре.

Вы также можете использовать знак? — это сокращение для ключевого слова **print** на панели **Immediate Window** (Окно непосредственной отладки).

Вы также можете вызвать функцию с помощью ключевого слова **print** на панели **Immediate Window** (Окно непосредственной отладки) редактора Visual Basic. Таким образом вы увидите значение, которое возвращает функция.

Функцию можно вызвать как с помощью формулы рабочего листа, так и из встроенных функций Excel, например, **SUM** и **AVERAGE**. Однако не все функции могут использоваться в формулах. Так как функция всего лишь возвращает значение вызывающей стороне, ее можно использовать лишь для определения диапазонов, указанных в качестве аргумента, или для

отображения возвращенного значения в ячейке или ячейках, к которым была применена формула.

Обычно функции передают аргументы вызывающей стороны и возвращают конечный результат после изменения значения, но вы можете создать формулу без аргументов. Такие функции будут просто возвращать значение.

FirstFunc.xlsm

1. Начните модуль VBA с функции, которая объявит две переменные и возвратит строковое значение.

```
Function SENDTO() As String

Dim printer As String
Dim port As Integer

' Сюда помещаются инструкции (Шаги 2 и 3).

End Function
```

2. Добавьте код для инициализации каждой переменной.

```
printer = Application.ActivePrinter
port = InStrRev(printer, "on") — 2
```

В свойстве **Application.ActivePrinter** есть строка, в которой написано имя принтера и порт, но функция удаляет информацию о порте, поэтому в строке остается только имя принтера.

3. Теперь добавьте код, который принудит функцию вернуть значение.

```
SENDTO = Left(printer, port)
```

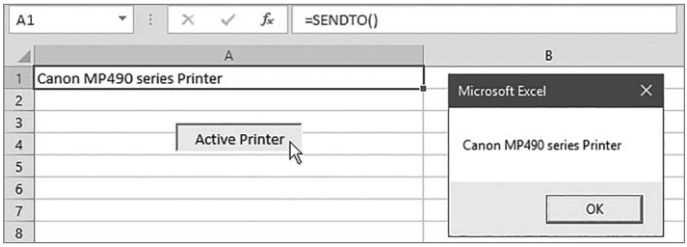
4. Откройте панель **Immediate Window** (Окно непосредственной отладки), напишите **print SENDTO** и нажмите клавишу **Enter** для проверки функции.

5. Добавьте подпрограмму, которая вызовет функцию для отображения возвращенной информации.

```
Sub PrintTo()
MsgBox SENDTO()
End Sub
```

Значение, возвращенное этой функцией, будет отличаться от описания вашего принтера.

6 Примените функцию к ячейке **A1**, а затем добавьте на лист кнопку для запуска подпрограммы.



Вызов функции

Когда вы применяете пользовательскую функцию к ячейке, она вызывается для немедленного возвращения значения. Если функция выполняет операции со значениями, принадлежащими другим ячейкам, то она будет ссылаться на текущее значение ячеек. Если используемые значения не указаны в качестве аргумента функции, то при каждом их изменении функция вызываться не будет:

Volatile.xlsm

1 Начните модуль VBA с функции, которая объявит две переменные и возвратит числовое значение.

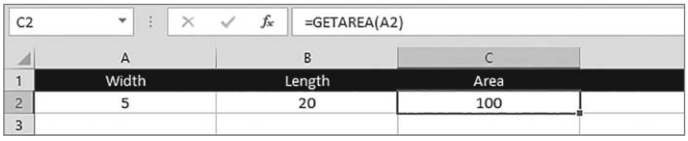
```
Function GETAREA(cell As Range) As Double

' Сюда помещаются инструкции (Шаг 6).

Dim width As Double
Dim length As Double

' Сюда помещаются инструкции (Шаги 2 и 3).

End Function
```

2 Затем добавьте код для инициализации переменных со значениями из других ячеек.

```
width = cell.Value
length = cell.Offset(0, 1).Value
```

3 Теперь добавьте финальный код, позволяющий назначить возвращаемое значение.
GETAREA = width * length

4 Примените функцию к ячейкам с числовым значением в столбцах слева. Вы увидите возвращенное значение.

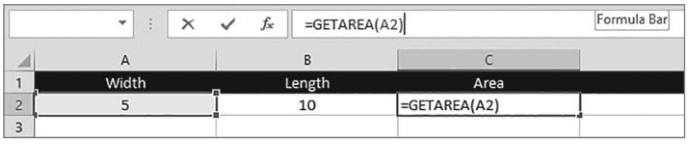


5 Измените значение любой из ячеек, используемых в функции. Функция не вызовется для изменения результата.

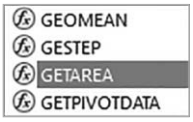
Мы советуем вызывать функцию каждый раз, когда значение других ячеек меняется. К счастью, в VBA есть возможность вызывать пользовательскую функцию при изменении любой ячейки:

6 Измените пользовательскую функцию путем добавления показанной ниже инструкции перед объявлением переменных.
Application.Volatile True

7 Переместите курсор на панель формул, а затем нажмите клавишу **Enter** для обновления функции на рабочем листе.

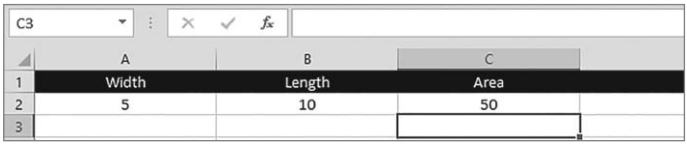


Для удобства имена пользовательских функций отображаются рядом с именами встроенных функций Excel.

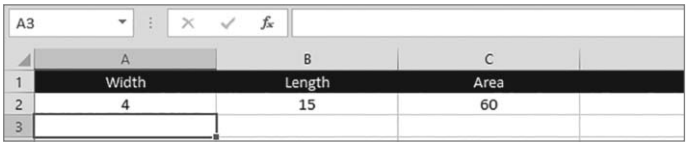


Вы можете опустить ключевое слово **True**, так как это значение по умолчанию свойства **Application.Volatile**.

8 Вы увидите, что функция, вызываемая для возврата значения, использует текущие значения ячеек.

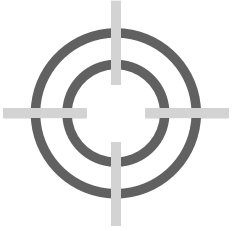


9 Измените значение любой из ячеек, используемых в функции. Теперь функция вызывается для изменения результата.



Вы можете изменить этот пример, заменив код на **Application.Volatile False**.

Область видимости функции



Пользовательская функция VBA по умолчанию имеет локальную область видимости, которая позволяет использовать ее в других процедурах и в формулах. Функция, которая будет использоваться только в других процедурах, должна иметь частную область видимости — **Private**. В таком случае она не отобразится в списке функций Excel. К тому же пользовательская функция может быть объявлена как статическая — тогда она будет сохранять значения переменных после их передачи, например, служебная функция, которая возвращает текущее значение состояния:

FuncScope.xlsm

1 В строку добавьте диапазон значений, затем добавьте метку, обозначающую их границы, а также кнопку для выполнения изменений.

	A	B	C	D	E	F	G	H	I	J	K	L
1	Average Monthly Temperatures : New York											
2	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec
3	2	2	4	11	16	22	25	24	20	14	9	9
4												
5	Centigrade				Change Temperature Scale							

2 Начните модуль VBA с частной функции, которая выполнит одно из двух вычислений для аргумента значения ячейки согласно аргументу логического выражения.

```
Private Function _
CONVERSION(flag As Boolean, cell As Range) As
Double

If(flag = True) Then
CONVERSION =((cell.Value * 9) / 5) + 32
Else
CONVERSION =((cell.Value — 32) * 5) / 9
End If

End Function
```

Символ подчеркивания _ используется для продолжения кода.

3 Затем добавьте частную статическую функцию, которая будет менять значение логической переменной каждый раз, когда она вызывается.

```
Private Static Function SETFLAG() As Boolean

Dim status As Boolean
If(IsEmpty(status) Or(status = False)) Then
status = True
Else
status = False
End If
SETFLAG = status
End Function
```

Встроенная функция **IsEmpty()** выдает ответ **True** только в том случае, если аргумент переменной не был инициализирован.

4 Начните подпрограмму с объявления двух переменных.

```
Sub ChangeScale()

Dim cell As Range
Dim flag As Boolean

' Statements to be inserted here (Steps 5–7).

End Sub
```

В подпрограмму добавьте инструкцию **MsgBox** — так вы сможете отслеживать изменения значения состояния каждый раз, когда нажимаете кнопку.

5 Добавьте код, который инициализирует переменную с текущим состоянием с помощью вызова частной статической функции.

```
flag = SETFLAG()
```

6 Затем добавьте инструкции, которые установят метку с нужным значением для текущего состояния.

```
If(flag = True) Then
Range("A5").Value = "Fahrenheit"
Else
Range("A5:).Value = "Centigrade"
End If
```

В этом примере код вызывает частную функцию 12 раз. В следующем примере показано, как сделать это более эффективно.

7 Добавьте цикл, который изменит значение каждой ячейки в строке, вызывая частную функцию при каждом выполнении итерации.

```
For Each cell In Range("A3: L3")
cell.Value = CONVERSION(flag, cell)
Next cell
```

8 Назначьте кнопке подпрограмму макроса, затем щелкните по ней мышью. Вы увидите, что значения и метка изменятся.

	A	B	C	D	E	F	G	H	I	J	K	L
1	Average Monthly Temperatures : New York											
2	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec
3	35.6	35.6	39.2	51.8	60.8	71.6	77	75.2	68	57.2	48.2	48.2
4												
5	Fahrenheit				Change Temperature Scale							

	A	B	C	D	E	F	G	H	I	J	K	L
1	Average Monthly Temperatures : New York											
2	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec
3	2	2	4	11	16	22	25	24	20	14	9	9
4												
5	Centigrade				Change Temperature Scale							

Передача массива аргументов

Пользовательская функция VBA может передавать массивы в качестве аргументов. В таком случае она обработает каждый элемент массива и возвратит значение вызывающей стороне. Массив состоит из диапазона ячеек, значение которых можно изменять с помощью функции после одного вызова. Чаще это намного удобнее, чем вызывать функцию по несколько раз. Также она может быть указана как аргумент в объявлении другой функции, так что возвращаемое ею значение будет использоваться в виде аргумента:

ArrayArg.xlsm

1 Добавьте в строку диапазон значений, затем добавьте метку, обозначающую их границу, а также кнопку для выполнения изменений.

	A	B	C	D	E	F	G	H	I	J	K	L
1	Average Monthly Precipitation : New York											
2	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec
3	3.9	2.95	4.06	3.94	4.45	3.5	4.53	4.13	3.98	3.39	3.82	3.58
4												
5	Inches				Change Precipitation Scale							

2 Начните модуль VBA с частной функции, которая выполнит одно из двух вычислений для значения аргумента ячейки согласно состоянию логического аргумента, а также возвратит значение метки.

```
Private Function _

CONVERSION(flag As Boolean, row As Range) As
String

Dim cell As Range

If(flag = True) Then

For Each cell In row
cell = Round((cell.Value * 25.4))
Next cell
CONVERSION = "Millimeters"
```

Встроенная функция **Round()** округляет десятичное число до целого или до того количества десятичных знаков, которое указано во втором аргументе.

```
Else

For Each cell In row
cell = Round((cell.Value / 25.4), 2)
Next cell
CONVERSION = "Inches"

End If

End Function
```

Частная статическая функция (**Private Static**) ничем не отличается от функции в предыдущем примере: код обработки перемещен из подпрограммы в частную функцию.

3 Теперь добавьте частную статическую функцию, которая будет менять значение логической переменной при каждом вызове.

```
Private Static Function SETFLAG() As Boolean

Dim status As Boolean
If(IsEmpty(status) Or(status = False)) Then
status = True
Else
status = False
End If
SETFLAG = status
End Function
```

4 Добавьте подпрограмму, которая установит метку с нужным значением для текущего состояния, а также изменит значение ячейки в строке, вызвав частную функцию только один раз.

```
Sub ChangeScale()

Range("A5").Value = _
CONVERSION(SETFLAG(), Range("A3: L3"))

End Sub
```

5 Назначьте кнопке подпрограмму макроса, затем щелкните по ней мышью. Вы увидите, что значения и метки изменятся.

	A	B	C	D	E	F	G	H	I	J	K	L
1	Average Monthly Precipitation : New York											
2	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec
3	99	75	103	100	113	89	115	105	101	86	97	91
4												
5	Millimeters				Change Precipitation Scale							

	A	B	C	D	E	F	G	H	I	J	K	L
1	Average Monthly Precipitation : New York											
2	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec
3	3.9	2.95	4.06	3.94	4.45	3.5	4.53	4.13	3.98	3.39	3.82	3.58
4												
5	Inches				Change Precipitation Scale							

Код в этом примере намного лучше, чем в предыдущем, так как он вызывает частную функцию лишь один раз.

Обратите внимание, что это глобально доступная подпрограмма с частной областью видимости (**Public**), в которой может содержаться только одна инструкция. Все из-за того, что код обработки в частных функциях скрыт. Это считается хорошим стилем программирования.

Определение параметров

В объявлении пользовательской функции аргументы можно указать как необязательные — просто поставьте перед именем аргумента ключевое слово **Optional**. Все необязательные аргументы должны располагаться в конце списка аргументов.

С помощью встроенной функции **IsMissing()** вы можете проверить, была ли совершена передача необязательного аргумента от вызывающей стороны. Если встроенная функция выдаст ответ **True**, то аргумент был передан. Если же ответ **False**, значит, аргумент может использоваться только в том случае, если его тип данных — **Variant**.

В объявлении функции вы можете передать неограниченное количество необязательных аргументов. Для этого перед именем аргумента массива укажите ключевое слово **ParamArray**. Этот аргумент должен быть последним в списке, а его тип данных должен быть **Variant**:

1. Начните модуль VBA с объявления функции, которая будет иметь один обязательный и два необязательных аргумента.

```
Private Function GETAREA(width As Double, _

Optional length As Variant, _
```

FuncOpt.xlsm

```
Optional height As Variant) As Double

' Сюда помещаются инструкции (Шаг 2).

End Function
```

2 Теперь объявите функцию, которая выдаст значение согласно количеству аргументов, переданных вызывающей стороной.

```
If IsMissing(length) Then
GETAREA = width * width
ElseIf IsMissing(height) Then
GETAREA = width * length
Else
GETAREA =(width * length) / 2
End If
```

Для функции **IsMissing()** вы можете указать только аргумент типа **Variant**.

3 Добавьте объявление функции, которая идентифицирует количество необязательных аргументов.

```
Private Function _
GETSUM(ParamArray nums() As Variant) As Double

' Сюда помещаются инструкции (Шаг 4).

End Function
```


4 Добавьте код для возврата окончательного результата всех переданных числовых аргументов.

```
Dim num As Variant
For Each num In nums
GETSUM = GETSUM + num
Next num
```

Так как тип данных **Variant** может обрабатывать любые виды данных, то вызывающая сторона может использовать смешанные типы аргументов: целые числа, строки, выражения, диапазоны.

5 Добавьте подпрограмму, которая вызовет первую функцию, пропустив первый обязательный аргумент.

```
Sub Caller()

Range("A1").Value = "Square: " & GETAREA(10)

' Сюда помещаются инструкции (Шаги 6–8).

End Sub
```

6. Добавьте код, который вызовет первую функцию, пропустив первый обязательный аргумент и первый необязательный аргумент.

```
Range("B1").Value = "Rectangle: " & GETAREA(10, 5)
```

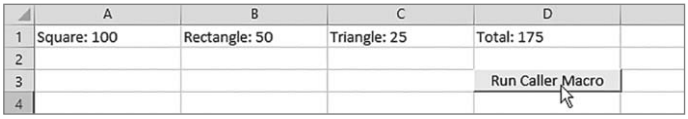
7. Затем добавьте код, который вызовет первую функцию, пропустив первый обязательный аргумент и два необязательных аргумента.

```
Range("C1").Value = "Triangle: " & GETAREA(10, 5, 4)
```

8. Теперь добавьте код, который вызовет вторую функцию, пропустив три необязательных аргумента.

```
Range("D1").Value = "Total: " & GETSUM _
(GETAREA(10), GETAREA(10, 5), GETAREA(10, 5, 4))
```

9. Назначьте подпрограмму макроса кнопке, затем щелкните по ней мышью. Вы увидите значения, возвращенные с помощью передачи значений необязательных аргументов.



Ключевому слову **ParamArray** может быть присвоен только один массив типа данных **Variant**.

Возвращение ошибок

Стандартные значения ошибок Excel, появляющиеся при обнаружении какой-либо из них, можно добавить в собственные пользовательские функции VBA. Это делается с помощью встроенной функции **CVErr()**. Просто укажите значение константы Excel из таблицы ниже в качестве аргумента для возврата соответствующей ошибки:

Константа Excel	Возврат	Ошибка
xlErrDiv0	**#DIV/0!**	Деление на ноль
xlErrNA	**#N/A**	Значение недоступно
xlErrName	**#NAME?**	Названный объект не найден
xlErrNull	**#NULL!**	Неверно указанный диапазон

Все значения ошибок Excel числовые, однако функция **CVErr()** меняет их на более понятные значения.

Константа Excel	Возврат	Ошибка
xlErrNum	#NUM!	Ожидаемое числовое значение
xlErrRef	#REF!	Ссылка на недопустимую ячейку
xlErrValue	#VALUE!	Недопустимое значение

Последний символ в имени ошибки **xlErrDiv0** — ноль, а не заглавная буква «О».

Наличие пользовательских функций, при возникновении ошибки возвращающих ее действительное значение в ячейку, гарантирует, что все ссылающиеся на ячейку формулы будут правильно идентифицировать ошибку. Вы можете сами узнать, есть ли у ячейки значение ошибки Excel — для этого нужно указать адрес ячейки в качестве аргумента встроенной функции **IsError()**:

1. Начните модуль VBA с функции, которая возвратит значение ошибки в том случае, если оно будет нечисловым.

```
Private Function NUM_ERROR() As Variant
If Not IsNumeric(Range("A1").Value) Then
NUM_ERROR = CVErr(xlErrNum)
End If
End Function
```

2. Затем добавьте функцию, которая возвратит значение ошибки в том случае, если ячейка не будет иметь значения.

```
Private Function DIV_ERROR() As Variant
If IsEmpty(Range("A2")) Then
DIV_ERROR = CVErr(xlErrDiv0)
End If
End Function
```

Обычно из-за отсутствия необязательного аргумента ошибка не возвращается, однако в этом примере необязательный аргумент используется для демонстрации возвращаемого значения ошибки.

3. Теперь добавьте функцию, которая возвратит значение ошибки, если необязательного аргумента нет.

```
Private Function _
NA_ERROR(Optional arg As Variant) As Variant
If IsMissing(arg) Then
NA_ERROR = CVErr(xlErrNA)
End If

End Function
```

4 Добавьте подпрограмму, которая напишет в ячейку текстовое значение.

```
Sub CreateErrors()

Range("A1").Value = "Errors:"

' Сюда помещаются инструкции (Шаги 5–7).

End Sub
```

5 Добавьте код, который в ячейках укажет значение ошибок.

```
Range("B1: D1").Value = _
Array(NUM_ERROR(), DIV_ERROR(), NA_ERROR())
```

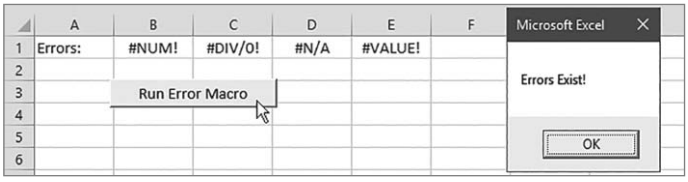
6 Добавьте код, который с помощью стандартной функции Excel будет пытаться добавить значения ячеек, имеющих ошибки.

```
Range("E1").Value = Application.Sum("A1: D1")
```

7 Добавьте код для отображения сообщения, если в ячейке есть ошибка.

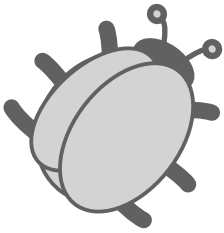
```
If IsError(Range("E1").Value) Then
MsgBox "Errors Exist!"
End If
```

8 Назначьте подпрограмму макроса кнопке, затем щелкните по ней мышью. Вы увидите значения ошибок и сообщения.



Обратите внимание на то, что стандартные функции Excel стоят на верхнем уровне иерархии. Вы можете ссылаться на них с помощью префикса **Application**. В этом примере используется функция SUM, однако редактор VB меняет регистр собственного имени с верхнего на регистр заголовка.

Отладка функций



Когда вы вызываете функцию из строки формул Excel, сообщения об ошибках не будут отображаться в диалоговом окне ошибок Excel. Вместо этого функция возвратит значение ошибки Excel **#VALUE!**. Чтобы найти ее причину, нужно вызвать функцию из подпрограммы, потому что только так получится открыть диалоговое окно ошибок VBA.

Для отладки кода в сам код функции желательно вставить инструкцию **MsgBox**, которая будет отображать значения переменных по мере их выполнения. При отображении каждого сообщения выполнение функции будет приостанавливаться — таким образом вы сможете контролировать изменения переменной.

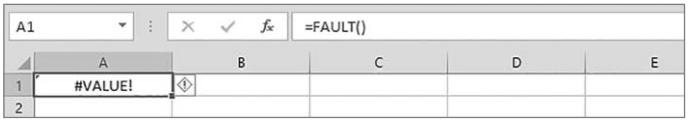
В качестве альтернативы в код функции можно вставить инструкцию **Debug.Print**, которая отображает значения переменных по мере их выполнения. Этот код отображает значения на панели **Immediate Window** (Окно непосредственной отладки) и позволяет отслеживать изменения переменной:

1. Начните модуль VBA с пользовательской функции, которая заполнит элементы массива из цикла, а затем возвратит конечное значение массива.

```
Function FAULT() As Integer

Dim nums(3) As Integer
Dim counter As Integer

For counter = 0 To 3
nums(counter) = counter

' Сюда помещаются инструкции (Шаг 6).

Next counter

' Сюда помещается инструкция (Шаг 6).

FAULT = nums(counter)

End Function
```

Если вы хотите использовать функцию в строке формул Excel, то при ее объявлении не используйте ключевое слово **Private**.

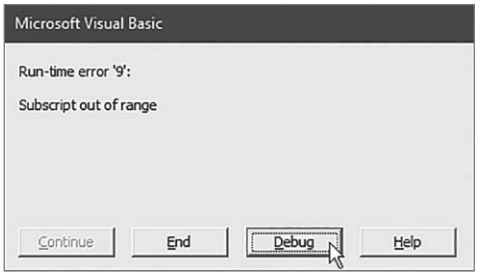
2 Выберите любую ячейку, затем напишите вызов функции в панель формул Excel и нажмите клавишу **Enter**. Вы увидите, что функция возвратит значение ошибки.



3 Вернитесь в редактор Visual Basic и добавьте подпрограмму для вызова функции.

```
Sub Caller()
Range("A1").Value = FAULT()
End Sub
```

4 Нажмите кнопку **Start** (Запуск), чтобы выполнить макрос. Появится диалоговое окно сообщения об ошибке с ее описанием.



5 Нажмите кнопку **Debug** (Отладка), чтобы отследить место с ошибкой.

```
FAULT = nums(i)
```

6 Добавьте показанный ниже код, чтобы узнать, как меняется значение переменной **counter** во время выполнения кода.

```
Debug.Print counter
```

7 Выберите команду меню **View** ⇨ **Immediate window** (Вид ⇨ Окно непосредственной отладки), а затем снова нажмите кнопку **Start** (Запуск).

8 Счетчик превысил верхний предел. Измените присвоение, чтобы исправить ошибку.

```
FAULT = nums(counter — 1)
```

Для проверки кода функции вы можете установить точки останова, о которых читали в главе 6.

Если в шаге 6 вставить **счетчик MsgBox**, можно отслеживать переменную по всему коду.

Описание функций

Если вы создали пользовательскую функцию для использования в качестве формулы, то можете сделать ее похожей на стандартные встроенные функции Excel. Метод **Application.MacroOptions** позволяет описать формулу и ее аргументы. С его помощью также можно указать категорию, к которой будет относиться функция в диалоговом окне **Insert Function** (Вставить функцию). Выбранная категория должна быть указана с помощью номера, указанного в таблице ниже:

1. Начните модуль VBA с пользовательской функции, которая возвратит значение, вычисленное из значений аргументов.

```
Function BULK_DISCOUNT _
(quantity As Integer, price As Double) As Double

If quantity >= 100 Then
BULK_DISCOUNT =(quantity * price) * 0.1
Else
BULK_DISCOUNT = 0
End If

End Function
```

2. Добавьте подпрограмму, которая опишет функцию.

```
Sub DescribeFunction()

Application.MacroOptions _
  Macro:= "BULK_DISCOUNT", _
  Description:= _
  "Calculates 10% discount for quantities 100+", _
  Category:= 1, _
  ArgumentDescriptions:= Array("Quantity", "Unit
Price")

End Sub
```

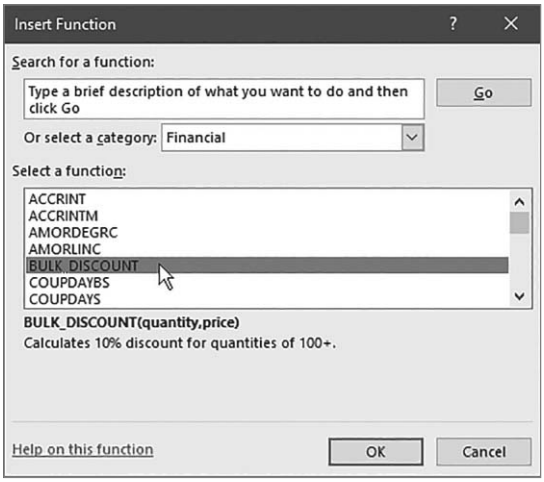
Для введения описания функции вам нужно запустить подпрограмму только один раз.

3. Теперь нажмите кнопку **Start** (Запуск) для выполнения подпрограммы и применения описания.

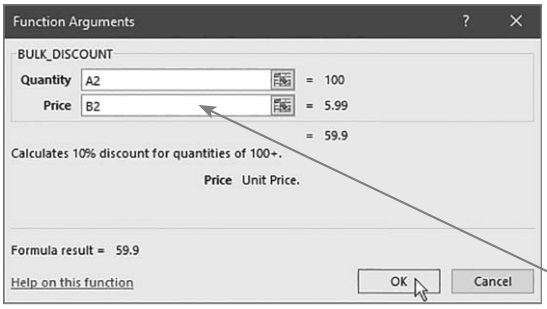
4 Вернитесь в Excel и выберите пустую ячейку, затем нажмите кнопку **Insert Function** (Вставить функцию), чтобы открыть одноименное диалоговое окно.

5 Выберите указанную вами категорию, чтобы увидеть сохраненную функцию, например, **Financial** (Финансы).

6 Выберите вашу функцию, затем нажмите кнопку **OK**. Откроется диалоговое окно **Function Arguments** (Аргументы функции).



7 Для указания значения аргументов введите ссылки на ячейки. Затем нажмите кнопку **OK** для применения формулы.



8 Перетащите маркер заполнения ячейки для копирования формулы в другие ячейки, как и любой встроенной функции Excel.

Категории	
1	Финансы
2	Дата и время
3	Математика и тригонометрия
4	Статистика
5	Опрос и ссылки
6	База данных
7	Текст
8	Логический объект
9	Информация
10	Программотехника
11	Куб
12	Совместимость
13	Интернет

1	Financial
2	Date & Time
3	Math & Trig
4	Statistical
5	Lookup & Reference
6	Database
7	Text
8	Logical
9	Information
10	Engineering
11	Cube
12	Compatibility
13	Web

Обратите внимание, что ваши описания появляются в каждом диалоговом окне функции.

Заключение

- Пользовательская функция VBA всегда возвращает значение вызывающей стороне.
- При объявлении функции после ключевого слова **Function** должны стоять круглые скобки **()**, в которых может быть указан список аргументов.
- Рекомендуется указание типа данных возвращаемого значения функции с помощью ключевого слова **As**.
- Пользовательская функция может быть вызвана из других процедур или из формулы.
- Функции возвращают значение вызывающей стороне или ячейке, к которой была применена формула.
- Функции, использующие значения других ячеек, не будут вызываться при каждом изменении значений других ячеек. Вы можете изменить это с помощью метода **Application.Volatile**.
- Пользовательская функция по умолчанию имеет область видимости **Public**.
- Функции, которые будут использоваться другими программами, должны быть объявлены частными. В таком случае они не будут отображаться в списке функций.
- Вы можете объявить пользовательскую функцию как статическую, и значение переменной сохранится после ее вызова.
- Вызывая функцию, лучше не делать этого несколько раз, а указать диапазон в качестве аргумента массива.
- Аргументы функции, объявленные как необязательные, должны находиться в конце списка аргументов.

- Встроенная функция **IsMissing()** позволяет определить, была ли выполнена передача необязательного аргумента с типом данных **Variant** от вызывающей стороны или нет.

- С помощью ключевого слова **ParamArray** в список аргументов можно написать неограниченное количество необязательных аргументов типа **Variant**.

- Вы можете использовать встроенную функцию **CVErr()** для возврата ошибки Excel из функции.

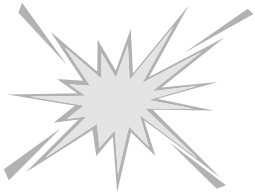
- Часто при отладке кода бывает полезным вставить инструкции **MsgBox** и **Debug.Print**. Они помогут вам проверять значения переменных.

- Метод **Application.MacroOptions** позволяет описывать функции таким образом, что они будут казаться встроенными.

8 Распознавание событий

В этой главе вы узнаете, как можно использовать макросы VBA для реакции на действия пользователя.

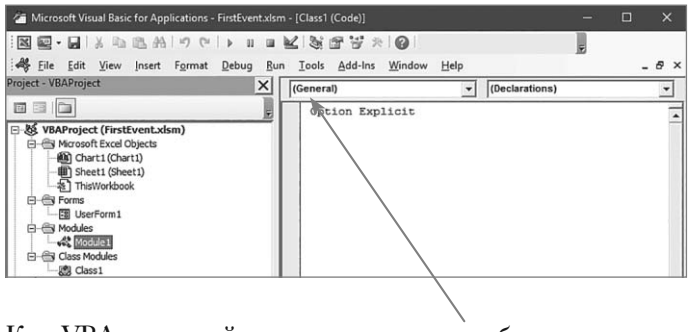
Создание обработчиков событий



Во время работы в Excel пользователь своими действиями вызывает разные события. Например, открытие книги, изменение ячейки или нажатие любой кнопки. Excel распознает каждое событие, а пользователь может написать код макроса в редакторе Visual Basic, чтобы программа реагировала на каждое из них. Однако важно понимать, что есть несколько отличающихся друг от друга видов событий:

- **События книги** затрагивают всю книгу, например, ее открытие или закрытие.
- **События листа** касаются отдельного рабочего листа, например, изменение значения ячейки.
- **События диаграммы** относятся к отдельной диаграмме, например, назначение точки данных.
- **События объекта-приложения** взаимодействуют с объектами Excel, например, создание новой книги.
- **События пользовательской формы** затрагивают отдельную пользовательскую форму, например, когда проверяется элемент управления формы **CheckBox**.
- **События программы** относятся к самой программе Excel, в частности события **OnTime** и **OnKey**.

Добавьте лист диаграммы, а затем с помощью меню **Insert** (Вставка) добавьте модули **UserForm** и **Class** на панели **Project Explorer** (Обозреватель проектов), как показано на рисунке.



Код VBA, который реагирует на эти события, — это процедуры, известные как обработчики событий.

Для распознавания события в соответствующем модуле нужно написать код обработчика. Все предыдущие примеры были созданы с использованием модуля **General**. Модуль вы можете увидеть и выбрать в окне редактора Visual Basic.

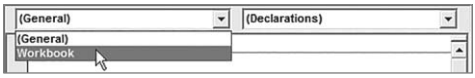
В раскрывающемся списке изначально доступен только модуль **General**, поэтому другой модуль нужно выбирать на панели **Project Explorer** (Обозреватель проектов).

FirstEvent.xlsm

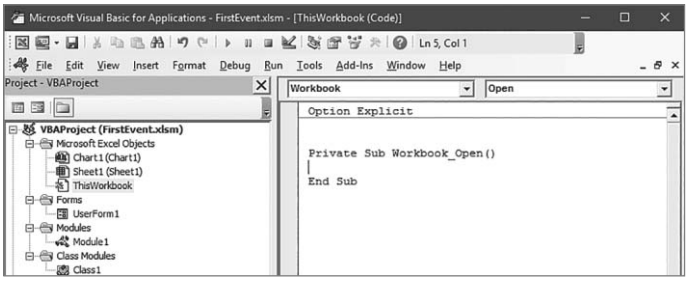
Чтобы увидеть модуль в окне редактора, щелкните правой кнопкой мыши по узлу и выберите пункт **View Code** (Просмотр кода) в контекстном меню.

1. На панели **Project Explorer** (Обозреватель проектов) дважды щелкните мышью по узлу **ThisWorkbook** (ЭтаКнига).

2. Чтобы убедиться в том, что элемент книги был добавлен, щелкните мышью по стрелке для раскрытия списка в редакторе Visual Basic.



3. Когда вы выберете пункт **Workbook**, курсор сместится в поле обработчика событий **Workbook_Open()**, подготовленный к созданию ответа на событие открытия книги.



4. Дважды щелкните мышью по остальным узлам, чтобы выбрать соответствующий модуль — **Sheet1** (Лист1) для модуля **Worksheet**, **Chart1** (Диаграмма1) для модуля **Chart**, **Class1** для модуля **Class** и **UserForm1** для модуля **UserForm**.

Узел **UserForm** откроется в окне конструктора — именно там, где вы добавляете визуальные элементы управления формы. Если вы выберете команду меню **View ⇨ Code** (Вид ⇨ Код), то откроется окно, в котором можно писать обработчики событий для формы.

Чтобы создавать обработчиков для вышеуказанных событий, необходимо использовать соответствующий модуль:

- **Модуль Workbook** для событий книги.
- **Модуль Worksheet** для событий рабочего листа (на этом листе).
- **Модуль Chart** для событий диаграммы (в этой диаграмме).
- **Модуль Class** для событий объекта-приложения.
- **Модуль UserForm** для событий пользовательской формы (в этой пользовательской форме).
- **Модуль General** для событий **OnTime** и **OnKey** объекта **Application**.

События открытия книги

При открытии книги в Excel происходит событие **Open**. И нет ничего удивительного в том, что отреагировавший на это событие обработчик событий называется **Workbook_Open()** и добавляется в модуль **Workbook**. Это один из самых часто используемых обработчиков событий, который выполняет такие задачи, как отображение приветственного сообщения или выбор определенного листа или ячейки.

Событие **Activate** выполняется точно так же: пользователь открывает книгу, и срабатывает обработчик событий **Workbook_Activate()**. Событие **Activate** отличается от события **Open** тем, что возникает в том случае, если пользователь переходит к открытой книге от другой книги. Событие **Open** возникает только при первом открытии.

Несколько событий, которые активируются одним и тем же действием, возникают не одновременно,

а в строгой последовательности. В этом примере событие **Open** запускается перед событием **Activate**:

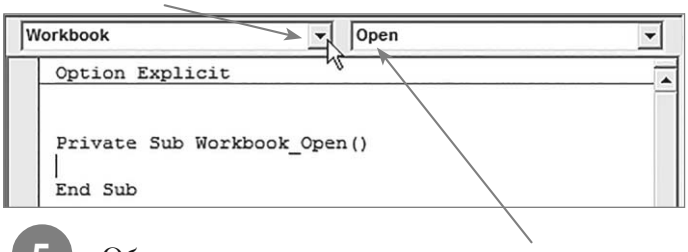
OpenBook.xlsm

1. Откройте книгу **FirstEvent** из последнего примера.

2. Откройте другую книгу и назовите ее **OpenBook**, а затем запустите редактор Visual Basic.

3. На панели **Project Explorer** (Обозреватель проектов) дважды щелкните мышью по узлу **ThisWorkbook** (ЭтаКнига).

4. Из раскрывающегося списка слева выберите пункт **Workbook**. Таким образом вы добавите обработчик событий **Workbook_Open()**.



С помощью раскрывающегося списка справа вы можете добавить любой подходящий обработчик событий в модуль, выбранный в раскрывающемся списке слева.

5. Обратите внимание, что теперь в раскрывающемся списке справа указано имя события для обработчика — в этом месте располагается курсор.

6. Добавьте показанные ниже инструкции, чтобы при открытии новой книги появлялось приветственное сообщение.

```
Dim span As String
If Time() < 0.5 Then
span = "Morning"
ElseIf Time() < 0.75 Then
span = "Afternoon"
Else
span = "Evening"
End If
MsgBox "Good " & span & ", " & Application.
UserName
```

В этом примере мы не набирали объявление подпрограммы **Sub** и инструкции **End Sub**, так как редактор Visual Basic вставляет их в код автоматически при выборе события в раскрывающемся списке справа.

7. Затем в раскрывающемся списке справа выберите пункт **Activate**. Таким образом вы

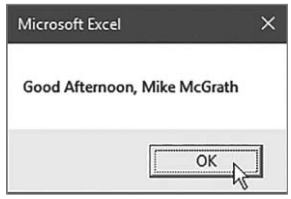
добавите обработчик событий **Workbook_ Activate()**.

8 Добавьте показанные ниже инструкции для выбора ячейки и отображения приветственного сообщения при открытии книги.

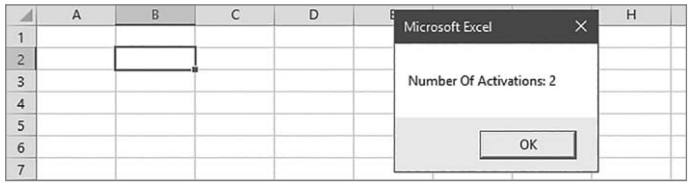
```
Static counter As Integer
counter = counter + 1
MsgBox "Number Of Activations: " & counter
Range("B2").Select
```

Приветственное сообщение будет разным в зависимости от времени суток и имени пользователя.

9 Сохраните книгу, закройте и откройте снова. При открытии вы увидите приветственное сообщение.



10 Переключитесь на первую открытую книгу, затем вернитесь во вторую. Это вызовет событие **Activate**.



Обратите внимание на то, что ячейка **B2** была выбрана из-за необходимости.

События изменения книги

Когда пользователь добавляет новый рабочий лист в книгу Excel, возникает событие **NewSheet**, на которое реагирует обработчик событий **Workbook_NewSheet()**. Обработчику событий передается единственный аргумент **Object**, который определяет тип листа как рабочий лист или диаграмму. Для этого он может сам проверить аргумент.

При открытии в книге листа любого типа возникает событие **SheetActivate**. На него реагирует обработчик событий **Workbook_SheetActivate()**, которому также передается единственный аргумент **Object**, определяющий тип листа как рабочий лист или диаграмму. Вы можете использовать это для ссылки на значения ячейки на листе или на значение данных в диаграмме:

ChangeBook.xlsm

1. Откройте редактор Visual Basic, а затем на панели **Project Explorer** (Обозреватель проектов) дважды щелкните по узлу **ThisWorkbook** (ЭтаКнига).

2. В раскрывающемся списке слева выберите пункт **Workbook**. Это модуль книги.

3. В раскрывающемся списке справа выберите пункт **NewSheet**. Таким образом вы добавите обработчик событий, код которого показан ниже.

```
Private Sub Workbook_NewSheet(ByVal Sh As Object)

' Сюда помещаются инструкции (Шаг 4).

End Sub
```

4. Добавьте инструкции для установки ширины столбца и значения ячейки в том случае, если новому листу присвоен тип.

```
If TypeName(Sh) = "Worksheet" Then
Sh. Cells.ColumnWidth = 32
Range("A1").Value = "Worksheet Added: " & Now()
End If
```

Имя аргумента **Sh** выбирается редактором Visual Basic для обозначения объекта листа.

5 Теперь из раскрывающегося списка справа выберите пункт элемент **SheetActivate**. Таким образом вы добавите обработчик событий, код которого показан ниже.

```
Private Sub Workbook_SheetActivate(ByVal Sh As
Object)

' Сюда помещаются инструкции (Шаг 6).

End Sub
```

Свойство **Values** объекта **SeriesCollection** на листе диаграммы содержит значение данных из диапазона ячеек, выбранного на этапе создания диаграммы.

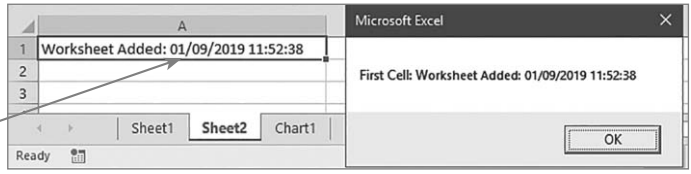
6 Добавьте тест для определения типа листа и отображения данных из ячейки или диаграммы рабочего листа.

```
If TypeName(Sh) = "Worksheet" Then
MsgBox "First Cell: " & Range("A1").Value
ElseIf TypeName(Sh) = "Chart" Then
Dim data As Variant
data = Sh. SeriesCollection(1).Values
MsgBox "First Data: " & data(1)
End If
```

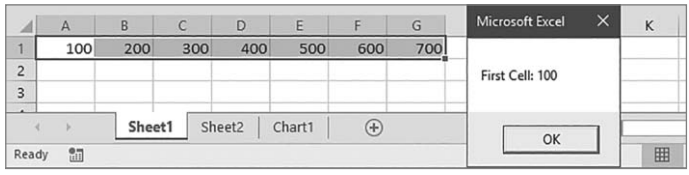
7 Выберите диапазон значений ячеек рабочего листа и добавьте диаграмму, а затем переместите ее на новый лист.

Обратите внимание, что из-за события **NewSheet** столбцы стали широкими.

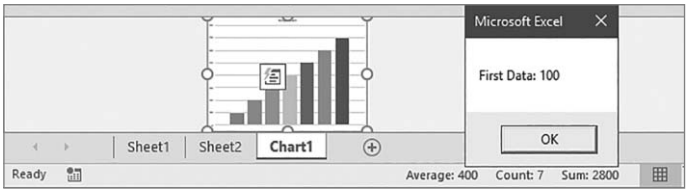
8 Добавьте новую страницу. Это вызовет событие **NewSheet**.



9 Вернитесь к первому листу. Это вызовет событие **SheetActivate**. Появится окно, в котором будет отображено значение ячейки.



10 Вернитесь к листу с диаграммой. Это вызовет событие **SheetActivate**. Появится окно, в котором отобразится значение данных.



События закрытия книги

Когда пользователь закрывает книгу Excel, возникает событие **BeforeClose**, на которое реагирует обработчик событий **Workbook_BeforeClose()**. Ему передается единственный **логический** аргумент, определяющий состояние отмены как **False**. Обработчик событий может изменить значение логического аргумента на **True**, тем самым предотвратив закрытие книги, если требуемое условие не выполняется.

При выводе книги на печать возникает событие **BeforePrint**. Обработчику событий **Workbook_BeforePrint()** также передается единственный **логический** аргумент, определяющий состояние отмены как **False**. Он может изменить значение логического аргумента на **True**, тем самым предотвратив печать книги, если требуемое условие не выполняется.

Когда пользователь сохраняет книгу, возникает событие **BeforeSave**, на которое реагирует обработчик событий **Workbook_BeforeSave()**. Ему в свою очередь передаются два логических аргумента, определяющих состояние отмены как **False** и возникновение диалогового окна **Save As** (Сохранить как) как **True**. Обработчик событий может изменить значение логического аргумента на **True**, тем самым предотвратив сохранение книги, если требуемое условие не выполняется.

Если пользователь попытается закрыть несохраненную книгу, то Excel отобразит окно с вопросом, не хочет ли пользователь сохранить книгу перед закрытием. Помните, что событие **BeforeClose** происходит до появления этого сообщения.

CloseBook.xlsm

1. Создайте список элементов рабочего листа, которые необходимо сложить.

	A	B	C	D	E
1	Item 1	$150.00			
2	Item 2	$250.00			
3	Item 3	$200.00			
4	TOTAL:				

2. Откройте редактор Visual Basic и дважды щелкните мышью по узлу **ThisWorkbook** (ЭтаКнига) на панели **Project Explorer** (Обозреватель проектов).

3. В раскрывающемся списке слева выберите пункт **Workbook**. Это модуль книги.

4. В раскрывающемся списке справа выберите пункт **BeforeClose**. Таким образом вы добавите обработчик событий, код которого показан ниже.

```
Private Sub Workbook_BeforeClose(Cancel As
Boolean)

' Сюда помещаются инструкции (Шаг 5).

End Sub
```

Обратите внимание на то, что ключевое слово **Me** используется для ссылки на объект книги, а также на свойство **Saved** и метод **Save**.

5. Добавьте инструкции, предотвращающие закрытие книги и проверяющие, была ли она сохранена перед закрытием.

```
If IsEmpty(Range("B4").Value) Then
MsgBox "Cannot Close: Items Are Untotaled"
Cancel = True
ElseIf Me. Saved = False Then
Me. Save
End If
```

6. В раскрывающемся списке справа выберите пункт **BeforePrint**, чтобы добавить обработчик событий, код которого приведен ниже.

```
Private Sub Workbook_BeforePrint(Cancel As
Boolean)

' Сюда помещаются инструкции (Шаг 7).

End Sub
```

7. Добавьте инструкции, которые предотвратят печать книги.

```
If IsEmpty(Range("B4").Value) Then
MsgBox "Cannot Print: Items Are Untotaled"
Cancel = True
End If
```

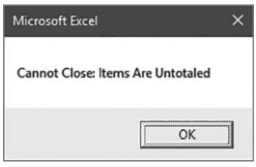
8. В раскрывающемся списке справа выберите пункт **BeforeSave**, чтобы добавить обработчик событий, код которого показан ниже.

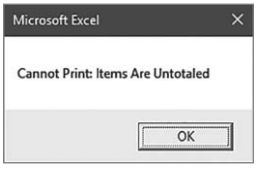
```
Private Sub Workbook_BeforeSave _
(ByVal SaveAsUI As Boolean, Cancel As Boolean)

' Сюда помещаются инструкции (Шаг 9).

End Sub
```

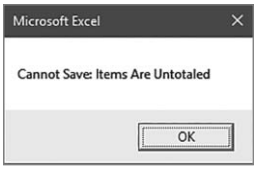
9. Добавьте инструкции, которые предотвратят сохранение книги.

```
If IsEmpty(Range("B4").Value) Then
MsgBox "Cannot Save: Items Are Untotaled"
Cancel = True
End If
```

10. Попробуйте закрыть, напечатать или сохранить книгу. Вы увидите, что каждая из операций будет прервана.







11. Для расчета конечного значения в ячейку B4 добавьте формулу **=SUM(B1:B3)**, затем закройте книгу.

Выявление изменений книги

Каждый раз, когда пользователь выбирает ячейку или диапазон ячеек на листе Excel, возникает событие **SelectionChange**, на которое реагирует обработчик событий **Worksheet_SelectionChange()**. Ему передается единственный аргумент **Range**, определяющий измененную ячейку или диапазон ячеек.

Убедитесь в том, что событие **Change** обеспечивает нужную функциональность, требуемую вашей процедурой.

Когда пользователь изменяет содержимое ячейки Excel, возникает событие **Change**, на которое реагирует обработчик событий **Worksheet_Change()**. Ему передается единственный аргумент **Range**, определяющий измененную ячейку или диапазон. Событие **Change** возникает тогда, когда пользователь или процедура VBA добавляет, изменяет или удаляет значение в ячейке или диапазоне. Событие также возникает в том случае, когда пользователь добавляет или удаляет форматирование ячейки или диапазона ячеек. Однако существует ряд изменений, которые не вызывают это событие:

- объединение ячеек с учетом того, что часть данных может быть удалена;
- добавление форматирования, если оно не вставлено в ячейку или диапазон;
- добавление, изменение или удаление комментария к ячейке.

Обработчик событий **Worksheet_Change()** может использоваться для проверки значений, введенных в диапазон ячеек:

SheetChange.xlsm

1. Дважды щелкните мышью по узлу **Sheet1** (Лист1) на панели **Project Explorer** (Обозреватель проектов), затем из раскрывающегося списка слева выберите пункт **Worksheet**. Это модуль книги.

2. Из раскрывающегося списка справа выберите пункт **SelectionChange**. Таким образом вы добавите обработчик событий, код которого показан ниже.

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

' Сюда помещаются инструкции (Шаг 3).

End Sub
```

3 Добавьте код для выделения текущей строки и столбца.

```
Cells.Interior.ColorIndex = xlNone
Target.EntireRow.Interior.ColorIndex = 20
Target.EntireColumn.Interior.ColorIndex = 20
```

4 Из раскрывающегося списка справа выберите пункт **Change**, чтобы добавить обработчик событий, код которого показан ниже.

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

' Сюда помещаются инструкции (Шаги 5–6).

End Sub
```

5 Добавьте инструкции, которые объявят две переменные и инициализируют одну переменную с определенным диапазоном, требующим проверки.

```
Dim cell As Range
Dim ValidationRange As Range
Set ValidationRange = Range("B1: B3")
```

Больше о требованиях к **Set** см. в главе 3.

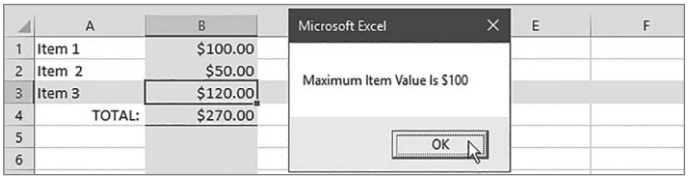
6 Добавьте проверку, которая будет проводиться только в том случае, если ячейка находится в указанном диапазоне.

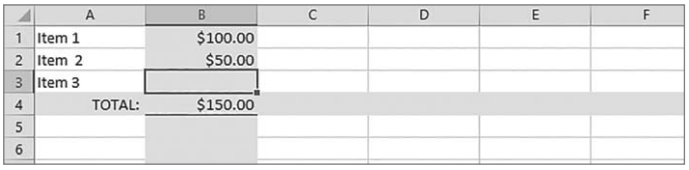
```
If Intersect(ValidationRange, Target) Is Nothing
Then
Exit Sub
Else
  For Each cell In Intersect(ValidationRange, Target)
  If Target.Value > 100 Then
  Target.Select
  MsgBox "Maximum Item Value Is $100"
  Target.ClearContents
  End If
  Next cell
End If
```

Если пользователь выбирает диапазон ячеек, то нужно использовать цикл.

7 Выберите любую ячейку или диапазон ячеек для выделения строки и столбца, затем введите недопустимое значение в любую

из ячеек диапазона. Появится ошибка проверки.





При закрытии диалогового окна код удаляет содержимое, и конечная сумма пересчитывается по формуле **SUM** из ячейки B4.

Обработка изменений книги

Когда пользователь дважды щелкает по ячейке на листе Excel, возникает событие **BeforeDoubleClick**, на которое реагирует обработчик событий **Worksheet_BeforeDoubleClick()**. Ему передается аргумент **Range**, идентифицирующий ячейку, а также **логический** аргумент, определяющий состояние отмены как **False**. Двойной щелчок по ячейке активирует режим редактирования ячейки, если, конечно, данный параметр не отключен. Если значение **логического** аргумента будет **True**, то режим редактирования ячейки выключится. Для изменения свойства **Style** аргумента **Range** вы можете дважды щелкнуть по ячейке.

Чтобы отключить режим редактирования ячеек, выберите команду меню **File** ⇨ **Options** ⇨ **Advanced** (Файл ⇨ Параметры ⇨ Дополнительно) и сбросьте флажок **Allow editing directly in cells** (Разрешить редактирование в ячейках).

Щелкая правой кнопкой мышью по ячейке на листе Excel, пользователь активирует событие **BeforeRightClick**, на которое реагирует обработчик событий **Worksheet_BeforeRightClick()**. Ему передается аргумент **Range**, идентифицирующий ячейку, а также логический аргумент, определяющий состояние отмены как **False**. Щелчок правой кнопкой мыши по ячейке вызывает контекстное меню, однако, если значение **логического** аргумента будет **True**, то контекстное меню станет недоступно. Для изменения свойства **Style** аргумента **Range** щелкните правой кнопкой мыши по ячейке:

SheetClick.xlsm

1. На панели **Project Explorer** (Обозреватель проектов) дважды щелкните мышью по узлу **Sheet1** (Лист1), затем в раскрывающемся списке в левой части окна для ввода кода выберите пункт **Worksheet**, чтобы выбрать модуль рабочего листа.

2. В раскрывающемся списке в правой части окна для ввода кода выберите пункт **BeforeDoubleClick** — вы добавите в код обработчик событий, показанный ниже.

```
Private Sub Worksheet_BeforeDoubleClick _
(ByVal Target As Range, Cancel As Boolean)

' Сюда помещаются инструкции (Шаги 3–4).

End Sub
```

3. Добавьте код для изменения оформления выбранной ячейки.

```
If(Target.Style = "Normal") Then
Target.Style = "Bad"
Else
Target.Style = "Normal"
End If
```

4. Затем добавьте код для отключения режима редактирования ячейки и вывода подтверждения.

```
Cancel = True
MsgBox "Cell Edit Mode Is Disabled"
```

5. В правом раскрывающемся списке окна для ввода кода выберите пункт **BeforeRightClick**, чтобы добавить обработчик событий, код которого показан ниже.

```
Private Sub Worksheet_BeforeRightClick _
(ByVal Target As Range, Cancel As Boolean)

' Сюда помещаются инструкции (Шаги 6–7).

End Sub
```

Шаги 3 и 6 в этом примере изменяют форматирование выбранной ячейки, причем шаг 6 предусматривает укороченный вариант кода. Больше дополнительной информации о встроенной функции **IIf()** см. в главе 5.

6 Добавьте код для изменения форматирования выбранной ячейки.

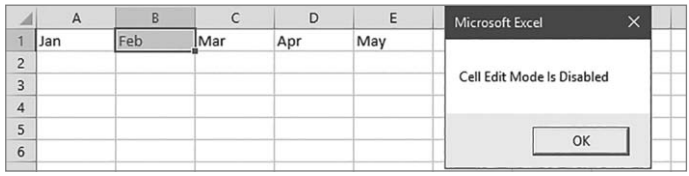
```
Target.Style = _
IIf(Target.Style = "Normal", "Good", "Normal")
```

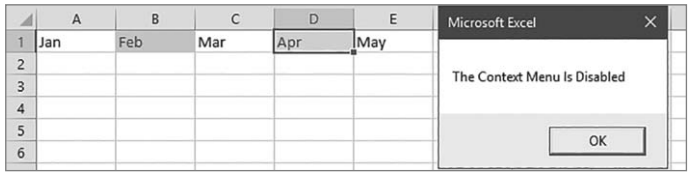
7 Затем добавьте код для отключения контекстного меню и вывода подтверждения.

```
Cancel = True
MsgBox "The Context Menu Is Disabled"
```

8 Дважды щелкните мышью, а затем щелкните правой кнопкой мыши по любой ячейке для изменения ее оформления и отображения подтверждения.

В случае если после щелчка правой кнопкой мыши не появляется контекстное меню, вы можете открыть его с помощью сочетания клавиш **Shift + F10**.





Перехват нажатий клавиш

Программа Excel постоянно перехватывает нажатия клавиш и вызывает событие **OnKey** для нажатий клавиш, которые отличаются от букв и цифр. Это событие уровня программы, поэтому код макроса должен располагаться в модуле **General**.

Чтобы назначить код макроса клавише, свойство **Application.OnKey** должно содержать код клавиши из числа показанных в таблице ниже и имя подпрограммы. Все это должно быть записано строками, разделенными запятыми.

Также можно назначить и сочетание клавиш, добавив клавиши **Shift**, **Ctrl** или **Alt**.

Пользователю важно понимать, что макрос, назначенный свойству **Application.OnKey**, будет применяться во всех открытых книгах Excel.

Чтобы вернуть клавиатуре состояние по умолчанию, рекомендуется присвоить программе свойство **Application.OnKey** без имени подпрограммы. Обычно этот код — часть обработчика событий **Workbook_BeforeClose()**.

Код	Клавиша
{BS}	Backspace
{BREAK}	Break
{CAPSLOCK}	Caps Lock
{DEL}	Delete
{DOWN}	↓
{END}	End
~	Enter
{ENTER}	Enter (на числовом блоке)
{ESC}	Escape
{HOME}	Home
{INS}	Insert
{LEFT}	←
{NUMLOCK}	Num Lock
{PGDN}	Page Down
{PGUP}	Page Up
{RIGHT}	→
{SCROLLLOCK}	Scroll Lock
{TAB}	Tab
{UP}	↑
{F1} — {F15}	F1 — F15
+	Shift
^	Ctrl
%	Alt

1. Начните модуль VBA с подпрограммы, которая назначит сочетание клавиш **Ctrl+Tab** для запуска другой подпрограммы.

```
Sub Start_OnKey()

Application.OnKey "^{TAB}", "HiLite"

End Sub
```

2. Теперь добавьте назначенную подпрограмму, которая заполнит выбранную ячейку и отобразит сообщение.

```
Private Sub HiLite()

On Error Resume Next

ActiveCell.Interior.Color = vbRed
MsgBox "OnKey Is Active!"

End Sub
```

Вы можете назначить сочетание клавиш для запуска макроса с помощью свойства **Application.OnKey**, однако намного проще сделать это через диалоговое окно **Macro Options** (Параметры макроса).

Добавьте средство обработки ошибок в любую подпрограмму, назначенную для выполнения **Application.OnKey**.

KeyListener.xlsm

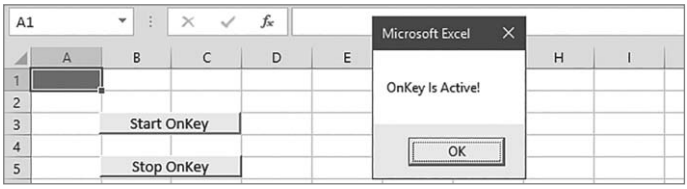
Обычно код, возвращающий клавиатуру к состоянию по умолчанию, располагается в обработчике событий **Workbook_BeforeClose()**, однако в данном случае в качестве примера код был назначен кнопке.

3. Затем добавьте подпрограмму, которая вернет клавиатуру к состоянию по умолчанию.

```
Sub Stop_OnKey()

Application.OnKey "^{TAB}"

End Sub
```

4. На рабочий лист добавьте две кнопки для запуска подпрограмм из шагов 1 и 3.

5. Нажмите кнопку для применения назначения, а затем выберите любую ячейку и нажмите сочетание клавиш **Ctrl+Tab** для выполнения макроса.



6. Нажмите другую кнопку, чтобы вернуть клавиатуру к состоянию по умолчанию.

Отслеживание времени

Программа Excel следит за ходом системных часов и может вызывать событие **OnTime** в указанное время. Это событие уровня программы, поэтому код макроса должен располагаться в модуле **General**.

Чтобы назначить код макроса таймеру, значение времени и имя подпрограммы нужно присвоить в строковом виде (в кавычках) свойству **Application.OnTime**. Код может включать в себя длительность задержки и логическое значение для составления графика таймера. Конечное значение **True** (обычно установлено по умолчанию) запускает новый таймер, в то время как конечное значение **False** отменяет запланированный. Время дня может быть указано как строковый аргумент в виде часов, минут и секунд, разделенных двоеточием, для встроенной функции

TimeValue(). Например, функция **TimeValue("12:00:00")** используется для обозначения полудня. Чтобы запланировать таймер на определенное время относительно текущего времени, просто добавьте нужное значение времени к текущему: например, **Now + TimeValue ("01:00:00")**, чтобы запланировать таймер на час вперед. Конкретные дата и время могут быть указаны путем объединения значений даты и времени: например, полдень 1 января 2018 года — это **DateSerial(2018, 1, 1) + TimeValue("12:00:00")**.

С помощью функции **TimeValue()** вы можете вводить время как в 24-часовом, так и в 12-часовом формате, то есть оба варианта — 14:30 и 2:30 — допустимы.

Таймер можно запускать повторно, в обратном порядке вызывая подпрограмму, в которой указан код макроса. В данном случае необходимо использовать код отмены таймера, потому что иначе он будет работать даже после закрытия книги. Сделать это очень просто: присвойте назначенное значение времени и имя подпрограммы свойству **Application.OnTime** с конечным значением **False**. Обычно этот код — часть обработчика события **Workbook_BeforeClose()**:

1. Начните модуль VBA с кода, который объявит глобальную переменную для хранения даты и времени.

```
Dim interval As Date
```

2. Теперь добавьте подпрограмму, которая отобразит текущее время в ячейке A1.

```
Sub Tick()

On Error Resume Next

Range("A1").Value = Time
' Statements to be inserted here (Step 3).

End Sub
```

3. Затем добавьте инструкции, которые вставят таймер с интервалом в одну секунду.

```
interval = Now + TimeValue("00:00:01")
Application.OnTime interval, "Tick", Null, True
```

4. Добавьте подпрограмму, которая отменит таймер из предыдущего шага.

С помощью необязательного периода задержки можно указать самое позднее допустимое время, в которое подпрограмма должна начать выполняться, если Excel не запустит ее в указанное время. В этом примере необязательный период задержки определен как **Null**, что обозначает его отсутствие.

```
Sub Stop_Tick()

On Error Resume Next
Application.OnTime interval, "Tick", Null, False

End Sub
```

5 Добавьте подпрограмму, которая установит таймер на пять секунд.

```
Sub Set_Alarm()

Application.OnTime Now + TimeValue("00:00:05"),
"Alarm"

End Sub
```

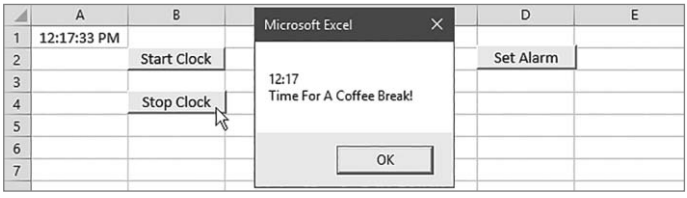
Обычно код отмены таймера часов располагается в обработчике событий **Workbook_BeforeClose()**, однако в данном случае в качестве примера код был назначен кнопке.

6 Добавьте подпрограмму, назначенную таймером из предыдущего шага.

```
Private Sub Alarm()

MsgBox Format(Time, "h: mm") & vbNewLine & _
"Time For A Coffee Break!"

End Sub
```

7 На рабочий лист добавьте три кнопки для выполнения подпрограмм из шагов 1, 3 и 5.

8 Нажмите кнопку для управления часами и будильником.



Заключение

- Программа Excel умеет распознавать события, вызванные действиями пользователя.
- Обработчики событий реагируют на каждое из них.
- Существует несколько видов событий, а их обработчики должны быть созданы в соответствующем модуле.
- При открытии книги происходят события **Open** и **Activate**.
- Добавление нового листа в книгу вызывает событие **NewSheet**, а активация листа — событие **SheetActivate**.
- Закрытие книги вызывает событие **BeforeClose**, а сохранение книги — событие **BeforeSave**.
- Печать книги вызывает событие **BeforePrint**.
- Выбор ячейки листа вызывает событие **SelectionChange**.
- Добавление, редактирование или удаление ячейки или диапазона ячеек вызывает событие **Change**. Также оно вызывается при добавлении или удалении форматирования ячейки или диапазона ячеек.
- Объединение ячеек, а также добавление, редактирование или удаление комментариев к ячейкам не вызывает событие **Change**.
- Двойной щелчок мышью по ячейке вызывает событие **BeforeDoubleClick**.
- Щелчок правой кнопкой мыши по ячейке вызывает событие **BeforeRightClick**.

- В модуле **General** могут отображаться только обработчики событий **Application.OnKey** и **Application.OnTime**.

- Свойство **Application.OnKey** определяет сочетание клавиш и подпрограмму в виде строк, разделенных запятыми.

- Свойство **Application.OnTime** определяет значение времени и строку имени подпрограммы.

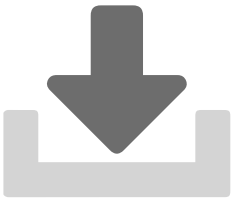
- Изменение конечного значения свойства **Application.OnTime** на **False** используется для отмены запланированного таймера.

- Таймер может запускаться повторно, в обратном порядке вызывая подпрограмму, где указан код макроса.

9 Отображение диалоговых окон

В этой главе вы узнаете, как с помощью диалоговых окон можно использовать макросы VBA для взаимодействия с пользователями.

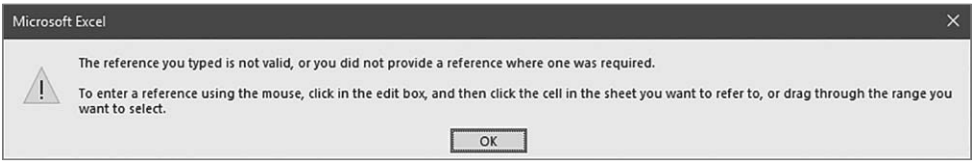
Запрос ввода



Функция **InputBox()** всегда возвращает **строковое** значение и не выполняет проверку.

С помощью диалогового окна ввода пользователь может самостоятельно ввести данные. Для этой цели в VBA есть функция **InputBox()**, которой в качестве первого аргумента нужно указать строку запроса. Вы можете добавить второй строковый аргумент, в котором указать название диалогового окна, а также третий аргумент с текстом для поля ввода. После нажатия кнопки **OK** содержимое поля ввода возвращается в процедуру как **строковый** тип данных. Вы можете присвоить его и переменной, однако все числовые данные должны быть преобразованы до того, как будут использованы в процедуре.

В Excel также есть функция **Application.InputBox()**, с помощью которой можно указать тип данных, которые необходимо возвратить. Вы можете указать диапазон ячеек, и у них будет автоматически проверяться тип данных. Если тип данных будет неверным, то Excel выдаст сообщение об ошибке.



Функция **Application.InputBox()** принимает такие же аргументы, что и функция **InputBox()**, однако вы можете указать необязательный аргумент для указания типа возвращаемых данных. Сделать это можно с помощью числовых кодов, указанных в таблице напротив. Аргументы должны быть указаны как поименованные аргументы **Prompt:**, **Title:**, **Default:** и **Type:**, однако обычно требуется только первый вариант.

Код	Тип данных
0	Формула
1	Номер
2	Строка
4	Логическое значение
8	Диапазон
16	Ошибка
64	Массив

Если в диалоговом окне ввода нажать кнопку **Cancel** (Отмена), то у процедуры логическое значение будет **False**. Если тип данных не **Boolean**, то появится ошибка, так что процедура должна включать в себя средство обработки ошибок.

GetInput.xlsm

1 Начните модуль VBA с подпрограммы, которая объявит три переменные.

```
Sub Total()

Dim selector As Range
Dim cell As Range
Dim result As Double

' Сюда помещаются инструкции (Шаги 2–4).

End Sub
```

2 Теперь добавьте обработку ошибок для кнопки **Cancel** (Отмена).

```
On Error Resume Next
```

Обратите внимание, что по умолчанию за ввод отвечает **selection.Address**.

3 Затем запросите ввод данных типа **Range**.

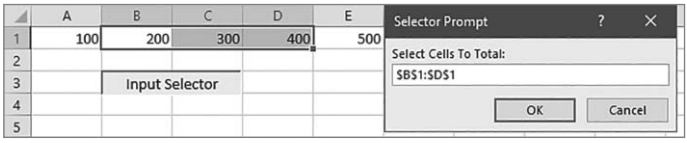
```
Set selector = Application.InputBox(_
Prompt:= "Select Cells To Total", _
Title:= "Selector Prompt", _
Default:= selection.Address, Type:= 8)
```

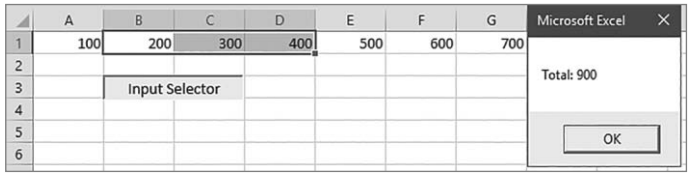
4 Проверьте, действительно ли вы запросили ввод данных, повторите диапазон и сложите выбранные значения ячеек.

```
If Not selector Is Nothing Then
  For Each cell In selector
  result = result + cell.Value
  Next cell
  MsgBox "Total: " & result
End If
```

Если вы решите опустить проверку наличия ввода, то проверка выдаст ошибку.

5 На рабочий лист добавьте кнопку для запуска макроса, щелкните по ней мышью. Вы увидите введенные данные и итоговый результат.





Отображение сообщений

Пользователь может изменить возможности диалогового окна **MsgBox** с помощью аргумента, указанного в круглых скобках после строки сообщения. С помощью аргумента можно указать, какой значок будет отображаться в диалоговом окне.

Константа кнопки	Значение кнопки
vbOkOnly	0
vbOkCancel	1
vbAbortRetryIgnore	2
vbYesNoCancel	3
vbYesNo	4
vbRetryCancel	5

Сочетание клавиш для вызова диалогового окна можно указать с помощью константы или соответствующего значения, указанного в таблице рядом. Например, если вы хотите, чтобы там отображались кнопки **Yes** (Да), **No** (Нет) и **Cancel** (Отмена), то используйте константу **vbYesNoCancel** или значение **3**.

Константа значка	Значение
vbCritical	16
vbQuestion	32
vbExclamation	48
vbInformation	64

Значок в диалоговом окне также можно указать с помощью константы или соответствующего значения, указанного в таблице рядом. Например, если вы хотите, чтобы в диалоговом окне отображался значок вопросительного знака, то используйте константу **vbQuestion** или ее значение **32**.

При вызове диалогового окна **MsgBox** всегда указывайте соответствующий значок — так пользователю будет проще понять, о чем сообщение.

Сочетание клавиш и значок могут быть указаны с помощью оператора сложения **+**. Например, если

вы захотите отобразить кнопки **Yes** (Да), **No** (Нет), **Cancel** (Отмена) и значок вопросительного знака, используйте **vbYesNoCancel + vbQuestion** или сумму их значений — в данном случае это будет **35** (3+32).

При нажатии кнопки в диалоговом окне **MsgBox** происходит возврат числового значения в процедуру. Вы можете присвоить это значение переменной и проверить ее значение, чтобы установить намерение пользователя.

Кнопка	Возвращаемое значение
OK	1
Cancel (Отмена)	2
Abort (Прервать)	3
Retry (Повторить)	4
Ignore (Игнорировать)	5
Yes (Да)	6
No (Нет)	

1 Начните модуль VBA с подпрограммы, которая объявит одну переменную.

```
Sub Question()
Dim intent As Integer

' Сюда помещаются инструкции (Шаги 2–4).

End Sub
```

ShowMessage.xlsm

2 Запросите вводимое количество решений из диалогового окна **MsgBox**, в котором будут кнопки **Yes** (Да), **No** (Нет), **Cancel** (Отмена) и значок вопросительного знака.

```
intent = MsgBox("Do You Wish To Proceed?", 35)
```

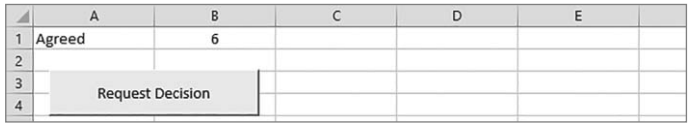
В целях краткости используйте числовые значения для сочетания кнопок и значков.

3 Проверьте возвращенное значение и отображение строки ответа.

```
Select Case intent
Case 2
Range("A1").Value = "Canceled"
Case 6
Range("A1").Value = "Agreed"
Case 7
Range("A1").Value = "Refused"
End Select
```

4. Наконец, отобразите возвращенное значение.
Range("B1").Value = intent

5. На рабочий лист добавьте кнопку и выполните макрос. В каждом случае вы увидите строку ответа и возвращаемое значение.



При закрытии диалогового окна **MsgBox** нажатием кнопки ✕ в процедуру возвращается числовое значение **2**, как и при нажатии кнопки **Cancel** (Отмена).

Импортирование файлов

Процедура VBA с помощью функции **Application.GetOpenFilename()** позволяет пользователю самому выбирать файл для импорта данных. В таком случае открывается системное диалоговое окно, где пользователь может перейти в папку и выбрать файл. Обычно это файл, содержащий разделенные запятыми значения (CSV), которые юзер может импортировать на рабочий лист в виде таблицы.

Файлы данных, в которых перечислены значения, разделенные запятыми, обычно имеют тип файла *.csv* или *.txt*.

Функция **Application.GetOpenFilename()** принимает аргументы для определения названия диалогового окна и типов файлов, которые будут видны при его открытии. Данные должны быть указаны как поименованные аргументы **Title:** и **FileFilter:**, хотя они не обязательны.

После того как пользователь с помощью диалогового окна выбрал нужный файл, функция **Application.GetOpenFilename()** возвращает путь к файлу, который можно назначить переменной. В дальнейшем этот путь можно использовать с функцией **QueryTables**.

Add() для импорта данных. Нужно указать аргументы как поименованные аргументы **Connection:** для описания типа и пути, а также **Destination:**, чтобы определить область на листе, в которой будет создана таблица. Возвращаемая функцией **QueryTables.Add()** таблица имеет свойства **AdjustColumnWidth** и **TextFileCommaDelimiter**, которым можно присвоить логическое значение **True**, а также метод **Refresh**, который необходимо вызвать для отображения таблицы:

FileOpen.xlsm

1 Начните модуль VBA с подпрограммы, которая объявит две переменные.

```
Sub FileOpen()

Dim path As Variant
Dim table As Variant

' Сюда помещаются инструкции (Шаги 2 и 3).

End Sub
```

2 Теперь добавьте код, который присвоит путь к выбранному файлу в качестве значения первой переменной.

```
path = Application.GetOpenFilename(_
Title:= "Select A File To Import", _
FileFilter:= "Comma Separated Files, *.csv, _
                        Text Files, *.txt")
```

3 Затем добавьте проверку условия для отображения сообщения в случае, если пользователь не выбирает файл.

```
If path = False Then
MsgBox "No File Selected!", 16
Else

' Сюда помещаются инструкции(Шаг 4).

End If
```

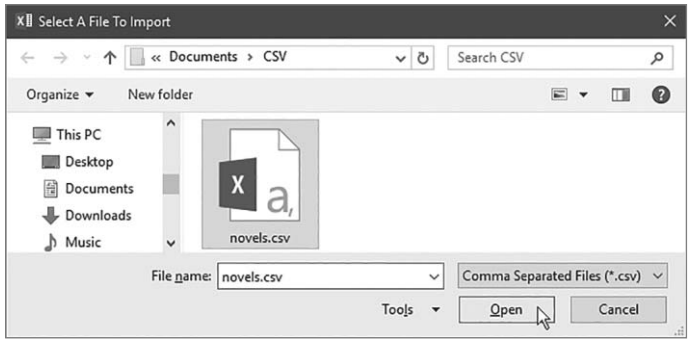
Обратите внимание на то, что присвоенное параметру **Connection:** значение состоит из двух частей между точками с запятой: **TEXT** и путь к файлу.

4 Добавьте код для отображения импортированных файлов в таблицу на рабочем листе.

```
Set table = ActiveSheet.QueryTables.Add(_
Connection:= "TEXT;" & path, Destination:=
                        Range("A1"))
```

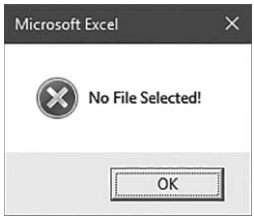
```
table.AdjustColumnWidth = True
table.TextFileCommaDelimiter = True
table.Refresh
```

5 На рабочий лист добавьте кнопку для запуска макроса, затем выберите файл. На рабочем листе появится таблица.



	A	B	C	D	E	F
1	Series #	Title	ISBN	Publication Date		
2	1	Killing Floor	0-515-12344-7	Mar-97		
3	2	Die Trying	0-399-14379-3	Jul-98		
4	3	Tripwire	0-515-14307-3	Jul-99	Select File	
5	4	Running Blind	978-0-515-14350-8	Apr-00		
6	5	Echo Burning	0-515-13331-0	Apr-01		
7	6	Without Fail	978-0-515-14431-4	Apr-02		
8	7	Persuader	978-0-440-24598-8	Apr-03		
9	8	The Enemy	0-553-81585-7	Apr-04		
10	9	One Shot	0-385-33668-3	Apr-05		

В этом примере сообщение отображается в том случае, если пользователь нажимает кнопку **Cancel** (Отмена) или ✕ для закрытия диалогового окна.



Сохранение файлов

Процедура VBA позволяет пользователю сохранять файл с любым именем, а также в любом расположении. Это происходит благодаря функции **Application.GetSaveAsFilename()**, которая создает системный диалог, где пользователь может выбрать папку и название файла.

Функция **Application.GetSaveAsFilename()** принимает аргументы для определения названия диалогового окна и типов файлов, видных при его открытии. Процедура может указать имя, которое отобразится в поле диалогового окна **File name** (Имя файла). Данные должны быть указаны как поименованные аргументы **Title:**, **FileFilte:** и **InitialFilename:**, хотя каждый из них необязателен.

Книги Excel можно сохранять в различных форматах, а не только с расширением *.xlsx* и *.xlsm*.

Когда пользователь выбрал расположение файла, функция **Application.GetSaveAsFilename()** возвратит адрес пути к файлу, который должен быть назначен переменной. С помощью метода **ActiveWorkbook.SaveAs** показанный ниже путь может использоваться для сохранения книги. А с методом **ActiveWorkbook.SaveCopyAs** вы можете сохранить копию книги. Допустим, вам надо сделать это после импорта данных из файла CSV (который был описан в предыдущем примере):

1. Начните модуль VBA с подпрограммы, которая объявит одну переменную.

```
Sub FileSave()

Dim path As Variant

' Сюда помещаются инструкции (Шаги 2 и 3).

End Sub
```

2. Теперь добавьте код, который присвоит выбранному файлу путь к переменной.

```
path = Application.GetSaveAsFilename(_
Title:= "Select A File Location", _
InitialFilename:= "Novels", _
FileFilter:= "Excel Files, *.xlsx, _
        Macro Enabled Workbook, *.xlsm")
```

3. Затем добавьте проверку условия для отображения сообщения в том случае, если пользователь не выберет место сохранения.

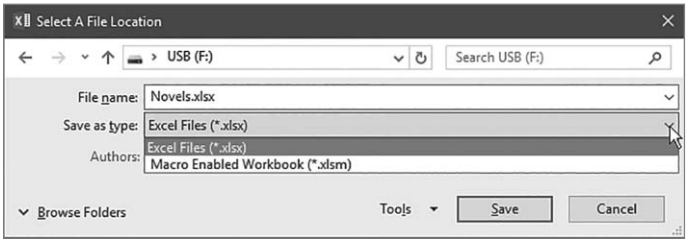
```
If path = False Then
MsgBox "No Location Selected!", 16
Else

' Сюда помещаются инструкции (Шаг 4).

End If
```

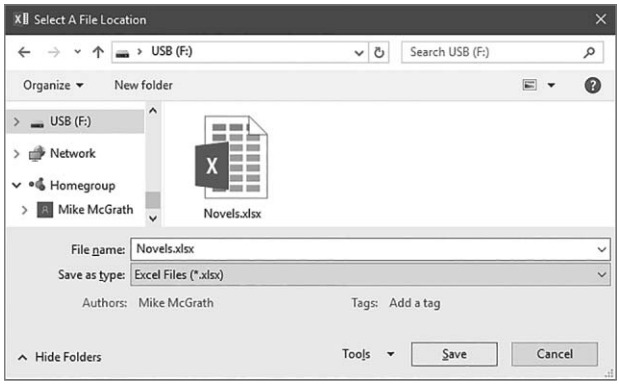
4. Добавьте код для сохранения копии рабочего листа в файл в выбранном месте.

```
ActiveWorkbook.SaveCopyAs path
```

5 На рабочий лист добавьте кнопку для выполнения макроса, запустите его, а затем выберите расположение и сохраните копию файла.

	A	B	C	D	E	F
1	Series #	Title	ISBN	Publication Date		
2	1	Killing Floor	0-515-12344-7	Mar-97		
3	2	Die Trying	0-399-14379-3	Jul-98		
4	3	Tripwire	0-515-14307-3	Jul-99	Select File	
5	4	Running Blind	978-0-515-14350-8	Apr-00		
6	5	Echo Burning	0-515-13331-0	Apr-01		
7	6	Without Fail	978-0-515-14431-4	Apr-02	Save File	
8	7	Persuader	978-0-440-24598-8	Apr-03		





В этом примере сообщение отображается в том случае, если пользователь нажимает кнопку **Cancel** (Отмена) или ✕ для закрытия диалогового окна.

Создание форм

Когда на рабочем листе есть таблица с множеством столбцов, для ее редактирования может понадобиться неоднократная прокрутка. В данном случае лучше всего использовать форму данных Excel. Она представляет собой диалоговое окно со строкой информации о диапазоне или таблице, ширина которой ограничена 32 столбцами. Excel автоматически создает форму данных и отображает заголовки столбцов в виде меток. Рядом с каждой меткой находится доступное для редактирования поле, в котором расположены данные столбца. Форма данных имеет

полосу прокрутки и кнопки для перемещения между строками данных в редактируемых полях. Также есть кнопки для добавления или удаления строк данных и поиска данных.

Вас может удивить то, что на ленте Excel нет кнопки **Form** (Форма), однако ее можно добавить на панель быстрого доступа самостоятельно:

1. Выберите диапазон или таблицу и убедитесь в том, что у каждого столбца есть заголовок, который будет использоваться как метка формы данных.

Убедитесь в том, что в выбранном вами диапазоне нет пустых строк.

2. Нажмите кнопку в виде стрелки на панели быстрого доступа и выберите пункт **More Commands** (Дополнительные команды), чтобы открыть диалоговое окно **Options** (Параметры).

3. В раскрывающемся списке **Choose commands from** (Выбрать команды) выберите пункт **Commands Not in the Ribbon** (Команды нет на ленте).

4. Пролистайте вниз панель слева и выберите пункт **Form** (Форма).

5. Нажмите кнопку **Add** (Добавить), чтобы добавить элемент на правую панель.

6. Нажмите кнопку **OK** для закрытия диалогового окна **Options** (Параметры). Вы увидите, что кнопка **Form** (Форма) появилась на панели быстрого доступа.

Если вы используете процедуру VBA, то не обязательно добавлять кнопку на панель быстрого доступа.

Теперь пользователь может нажать кнопку **Form** (Форма), и тогда он увидит форму данных, в которой отображаются метки заголовков и содержимое ячеек. В качестве альтернативы вы можете вставить кнопку для вызова процедуры на рабочий лист с помощью метода **Activesheet.ShowDataForm**:

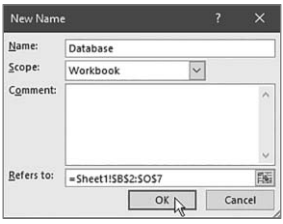
DataForm.xlsm

1 На рабочем листе выберите диапазон ячеек и убедитесь, что каждый столбец имеет заголовок, который используется как метка формы данных.

	A	B	C	D	E	F	G
1							
2		User Name	First Name	Last Name	Display Name	Job Title	Office
3		chris@contoso.com	Chris	Green	Chris Green	IT Manager	123451
4		ben@contoso.com	Ben	Andrews	Ben Andrews	IT Manager	123452
5		david@contoso.com	David	Longmuir	David Longmuir	IT Manager	123453
6		cynthia@contoso.com	Cynthia	Carey	Cynthia Carey	IT Manager	123454
7		melissa@contoso.com	Melissa	MacBeth	Melissa MacBeth	IT Manager	123455
8							

Указанный диапазон будет расширяться автоматически в процессе добавления новых данных в форму.

2 В группе **Defined Names** (Определенные имена) вкладки **Formulas** (Формулы) нажмите кнопку **Define Name** (Задать имя), чтобы открыть диалоговое окно **New Name** (Создание имени).



3 Присвойте диапазону имя **Database**, а затем нажмите кнопку **OK**.

4 Откройте редактор Visual Basic и создайте подпрограмму.

```
Sub FormOpen()

Activesheet.ShowDataForm

End Sub
```

5 На рабочий лист добавьте кнопку и назначьте ей макрос, а затем нажмите на нее. Вы увидите форму данных.

	A	B	C	D	E	F	G
1							
2		User Name	First Name	Last Name	Display Name	Job Title	Office
3		chris@contoso.com	Chris	Green	Chris Green	IT Manager	123451
4		ben@contoso.com	Ben	Andrews	Ben Andrews	IT Manager	123452
5		david@contoso.com	David	Longmuir	David Longmuir	IT Manager	123453
6		cynthia@contoso.com	Cynthia	Carey	Cynthia Carey	IT Manager	123454
7		melissa@contoso.com	Melissa	MacBeth	Melissa MacBeth	IT Manager	123455
8							
9			Show Data Form				
10							

Sheet1

Выполнение команд на ленте

Когда пользователь выбирает элемент, инструменты с ленты Excel выполняют команды; процедуры тоже могут выполнять команды. К примеру, диалоговое окно **Go To** (Перейти) приводит в действие команду **Goto**, и процедура начинает выполнять ее:

```
Application.Goto Reference:=Range("A1: A10")
```

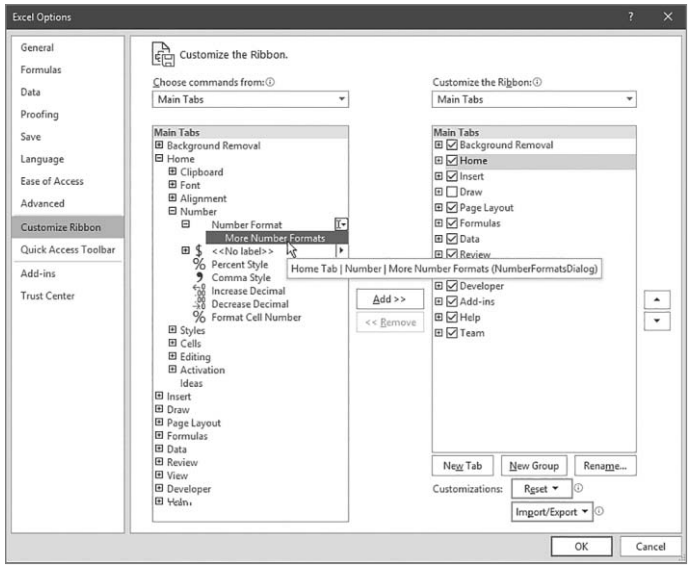
Инструкция выполняет команду без создания диалогового окна. С помощью инструкции **Application.CommandBars.ExecuteMso** вы можете создать диалоговое окно — укажите его имя в круглых скобках:

```
Application.CommandBars.ExecuteMso("GoTo")
```

Имена диалоговых окон вы можете узнать в окне **Options** (Параметры):

1. Выберите команду меню **File** ⇨ **Options** ⇨ **Customize Ribbon** (Файл ⇨ Параметры ⇨ Настроить ленту), а затем в раскрывающемся списке **Choose commands from** (Выбрать команды) выберите пункт **Main Tabs** (Основные вкладки).
2. Теперь разверните узел **Home** ⇨ **Number** ⇨ **Number Format** (Главная ⇨ Число ⇨ Числовой формат).
3. Установите указатель мыши на пункт **More Number Formats** (Другие числовые форматы), чтобы увидеть имя диалогового окна — **NumberFormatsDialog**.

Обратите внимание на то, что буква «T» в команде строчная, а в названии диалогового окна — заглавная.



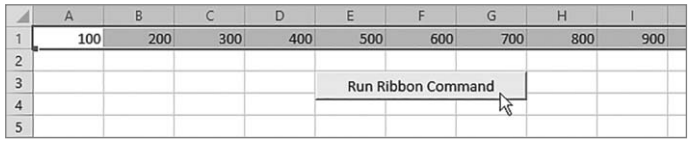
Эта процедура должна включать в себя обработку ошибок — таким образом вы сможете избежать ошибки Excel, возникающей при выборе элемента, отличного от ячейки листа. Как пример можно привести диаграмму.

4 Откройте редактор Visual Basic, затем создайте подпрограмму.

```
Sub RunRibbon()

On Error Resume Next
Application.CommandBars.ExecuteMso _
("NumberFormatsDialog")

End Sub
```

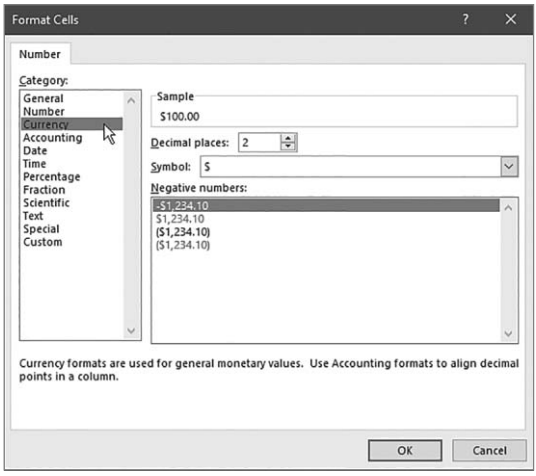
5 На рабочий лист добавьте кнопку и назначьте ей этот макрос.

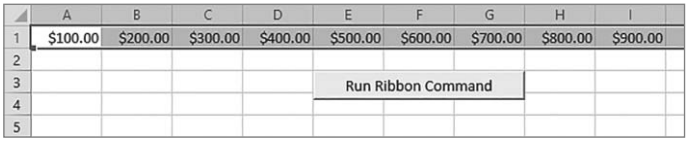
6 Выберите ячейки с числовыми значениями для форматирования, а затем нажмите кнопку для выполнения процедуры.



7 Выберите нужный формат и нажмите **OK**, чтобы закрыть диалоговое окно. Вы увидите,

что к выбранным ячейкам применился формат.





Из этого диалогового окна в процедуру не возвращается никакое значение.

Заключение

- Функция **InputBox()** помогает формировать запрос пользовательского ввода, однако она всегда возвращает данные типа **String**.
- Функция **Application.InputBox()** может использоваться для запроса пользовательского ввода и возврата данных без изменения их типа.
- С помощью функции **Application.InputBox()** проверка типа данных осуществляется автоматически.
- Сочетания клавиш и значки диалогового окна **MsgBox** можно указать как с помощью констант VBA, так и с помощью числовых значений.
- Сочетание клавиш и значка диалогового окна **MsgBox** можно указать с помощью оператора сложения **+** или сложения всех числовых значений.

- Любая кнопка диалогового окна **MsgBox** при нажатии возвращает в процедуру числовое значение.

- Функция **Application.GetOpenFilename()** позволяет пользователю самому выбрать файл и возвращает его расположение.

- Содержащиеся в файле и разделенные запятыми значения можно импортировать в таблицу с помощью функции **QueryTables.Add()**.

- Функция **Application.GetSaveAsFilename()** позволяет выбрать расположение файла и возвращает его расположение.

- С помощью метода **ActiveWorkbook.SaveAs** можно сохранить книгу, а с помощью **ActiveWorkbook.SaveCopyAs** — ее копию.

- Форма данных Excel особенно полезна при редактировании диапазона ячеек или таблицы, ширина которой ограничена 32 столбцами.

- Для редактирования данных с помощью формы данных пользователь может добавить кнопку **Form** (Форма) на панель быстрого доступа.

- Присвоение имени диапазону или таблице базы данных позволяет VBA работать с формами данных с помощью метода **Activesheet.ShowDataForm**.

- Процедура VBA может выполнять некоторые команды ленты, например, **Application.Goto**.

- В процедуре VBA можно создавать диалоговое окно — для этого укажите его название в инструкции **Application.CommandBars.ExecuteMso**.

10 Добавление пользовательских форм

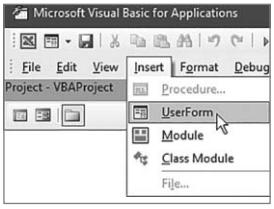
В этой главе вы узнаете, как можно использовать форму макроса VBA для взаимодействия с пользователем.

Вставка пользовательских форм

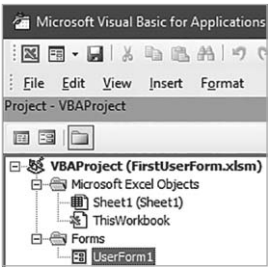
Редактор Visual Basic упрощает создание пользовательских диалоговых окон, позволяя добавить одну или несколько пользовательских форм для взаимодействия с книгой:

FirstUserForm.xlsm

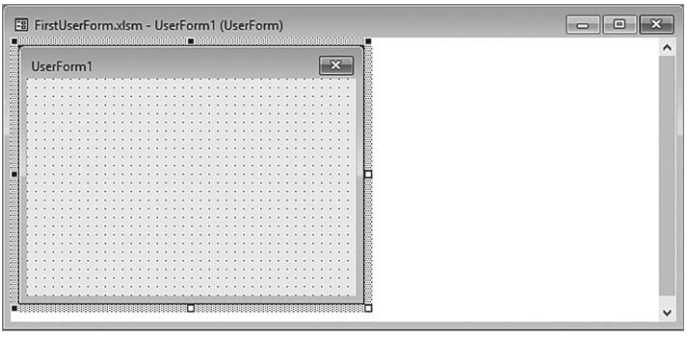
1. Откройте редактор Visual Basic, затем выберите команду меню **Insert ⇨ UserForm** (Вставка ⇨ Пользовательская форма).



2. Убедитесь, что на панели **Project Explorer** (Обозреватель проектов) в проект добавилась папка **Forms** с узлом пользовательской формы под именем **UserForm1**.



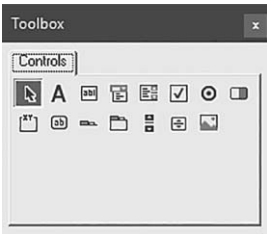
3. Дважды щелкните мышью по этому узлу — в окне конструктора откроется пустое диалоговое окно.



Окно конструктора — это пространство, где вы создаете пользовательские диалоговые окна путем добавления визуальных элементов управления формы на пустую пользовательскую форму.

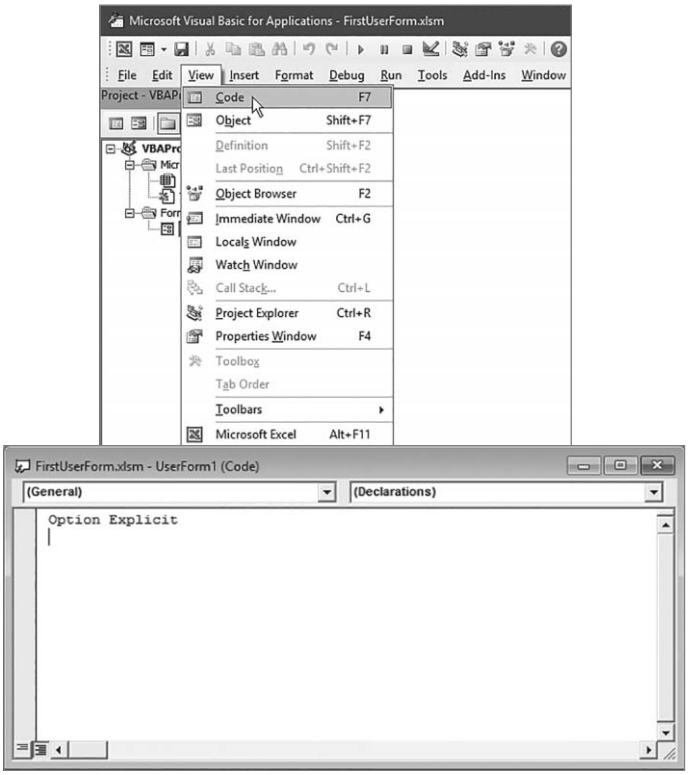
При открытии окна конструктора панель **Toolbox** (Инструменты) появится автоматически.

4 Выберите команду меню **View ⇨ Toolbox** (Вид ⇨ Панель элементов). Вы увидите перечень элементов управления формы, которые можно добавить на пустую пользовательскую форму.



При наведении курсора на элемент управления формы на панели **Toolbox** (Инструменты) появится подсказка с именем элемента.

5 Выберите команду меню **View ⇨ Code** (Вид ⇨ Код). Появится окно, в котором вы можете добавить функциональный код для пользовательской формы.

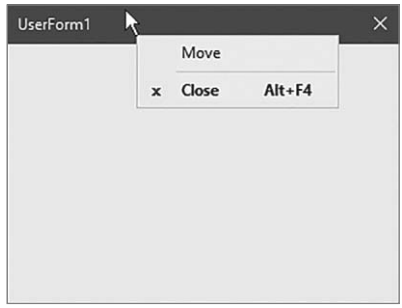


Вы можете изменять размер окон для просмотра проекта в окнах конструктора и кода.

6 Нажмите кнопку **Run** (Выполнить) или **F5**, чтобы вернуться к Excel. Появится пользовательская форма. Пока вы не добавите элементы управления формы и функциональный код, от них будет мало толку: вы сможете только перемещать окно или

Заголовок в верхней части пользовательской формы обычно такой же, как и имя его узла, однако вы всегда можете изменить его.

закрыть его, но у вас не получится изменить его размер или свернуть.

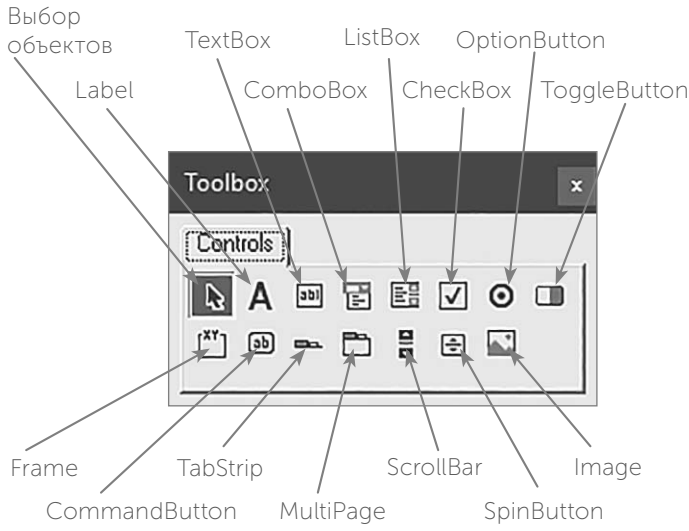


7 Нажмите кнопку × на пользовательской форме, чтобы закрыть ее и вернуться в редактор Visual Basic.

Добавление элементов управления формы

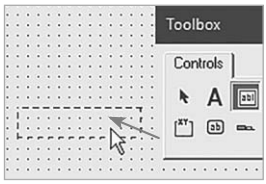
На панели **Toolbox** (Инструменты) редактора Visual Basic можно найти все элементы управления формы, доступные для добавления в ваш проект. Вы можете потянуть за угол панели и изменить ее размер. Доступные элементы управления формы перечислены на рисунке ниже:

Кнопка в виде стрелки на панели **Toolbox** (Инструменты) возвращает указатель мыши в режим работы по умолчанию, чтобы вы смогли выбирать объекты в окне конструктора.

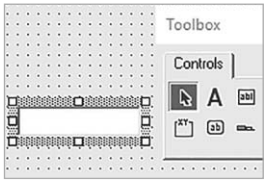


Существуют два способа добавления элементов управления формы:

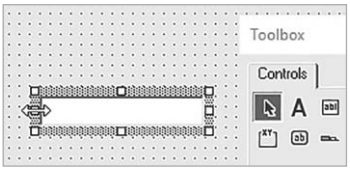
1. На панели **Toolbox** (Инструменты) выберите элемент управления формы, затем щелкните мышью в пользовательской форме. Размер элемента будет установлен по умолчанию.



2. После добавления элемента управления формы вы увидите на его границах маркеры — это означает, что элемент выделен.



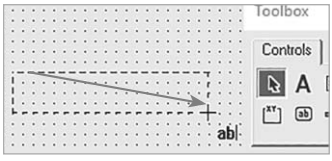
3. Нажав и удерживая кнопку мыши на любом маркере, перетащите его сторону, чтобы изменить размер элемента.



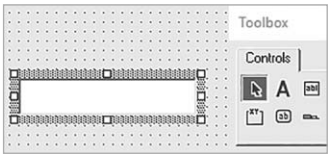
Или…

1. Выберите элемент управления формы на панели **Toolbox** (Инструменты).

2. Нажав и удерживая кнопку мыши в пользовательской форме, перетащите мышь, чтобы добавить элемент управления формы желаемого размера.



3. Отпустите кнопку мыши. Вы увидите, что пока элемент выбран, он отображается с маркерами.

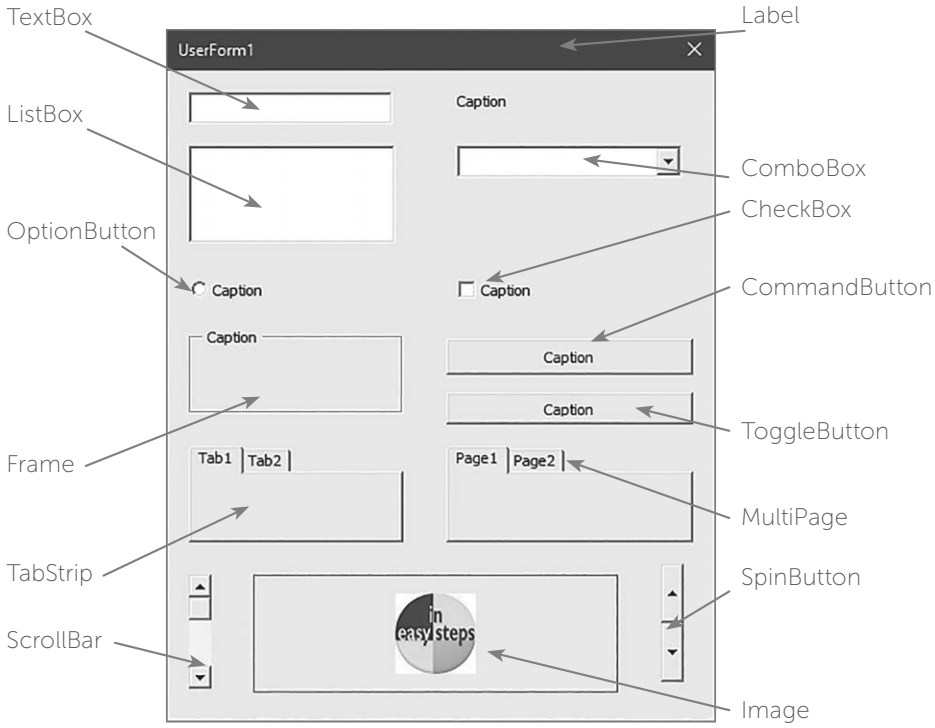


FormControls.xlsm

Чтобы переместить элемент управления формы, нажав и удерживая на нем кнопку мыши, перетащите мышь в сторону.

Когда вы выбираете элемент на панели **Toolbox** (Инструменты), указатель мыши меняет свой вид — именно таким способом программа показывает, что мышь «занята» элементом управления формы. После добавления элемента в пользовательскую форму или нажатия кнопки со стрелкой на панели **Toolbox** (Инструменты) указатель мыши возвращается в состояние по умолчанию.

Как только элемент управления формы перестает быть выбранным, маркеры пропадают. Когда вы нажимаете кнопку **Run** (Выполнить), элемент управления формы отображается как компонент Windows. В пользовательской форме, показанной ниже, добавлено по одному из всех элементов управления формы:



Сравнение элементов формы

Элементы управления формы позволяют пользователю легко работать с данными рабочего листа и кодом VBA. Ниже дано краткое описание всех элементов управления формы и их применение:

- **Label** отображает текст, который пользователь не может редактировать. Предоставляет инструкции или определяет назначение поля.

Элемент управления формы **TextBox** может также предоставлять информацию только для чтения.

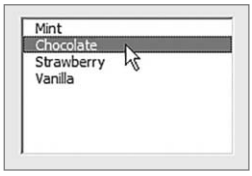
- **TextBox** — прямоугольное окно, в котором пользователь может просматривать, вводить или редактировать текст.

- **ListBox** отображает список из одной или нескольких строк (пунктов) текста, которые может выбрать пользователь. Существуют три вида списков:

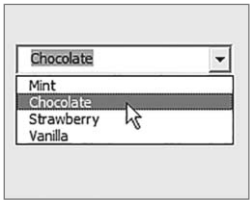
  1. **Список, допускающий единственный выбор,** позволяет выбрать только один пункт из списка.

  2. **Список с множественным выбором** позволяет выбрать один или несколько связанных пунктов из списка.

  3. **Список со связанным выбором** позволяет выбрать один или несколько пунктов из любого места списка.



- **ComboBox** объединяет в себе свойства элементов **TextBox** и **ListBox** и представляет собой раскрывающийся список, из которого пользователь может выбрать один элемент.



- **ScrollBar** позволяет прокручивать диапазон значений, когда пользователь перетаскивает ползунок прокрутки или нажимает стрелки прокрутки.

- **SpinButton** изменяет значения, такие как время, дата или число. Когда пользователь нажимает кнопку вверх, значение увеличивается, а при нажатии кнопки вниз уменьшается.

- **Frame** — прямоугольный элемент управления формы с дополнительной меткой, который визуально группирует связанные элементы управления.

- **OptionButton** позволяет выбирать один элемент из списка взаимоисключающих параметров, который отображается в элементе **Frame**.

В целях экономии места для больших списков рекомендуется использовать элемент управления формы **ComboBox**, а не **ListBox**.

Элемент управления формы **OptionButton** также известен как «переключатель».

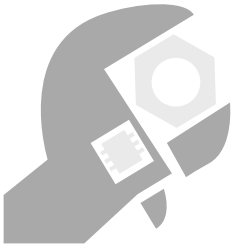
OptionButton — это переключатель, который может быть установлен только в одно положение. При выборе показанный ниже переключатель блокирует остальные положения (варианты).

- **CheckBox** позволяет выбрать один или несколько элементов из списка параметров, который отображается в элементе **Frame**. При выборе показанный ниже элемент управления формы не влияет на остальные флажки (варианты). Когда пользователь сбрасывает флажок, этот параметр деактивируется.

Элемент управления формы **CommandButton** также известен как «кнопка».

- **CommandButton** (кнопка) выполняет процедуру, когда пользователь нажимает кнопку.
- **ToggleButton** переключает состояния. Нажатие этой кнопки меняет состояние элемента на противоположное.
- **MultiPage** предоставляет возможность создавать отдельные диалоговые окна со вкладками-страницами с отдельными элементами управления формы. При выборе вкладки отображаются только элементы управления формы на соответствующей странице и скрываются остальные.
- **TabStrip** предоставляет возможность перемещаться между данными в одном и том же элементе управления формы. Не содержит отдельных страниц.
- **Image** вставляет изображение, например, BMP-, JPEG- или GIF-формата.

Изменение свойств



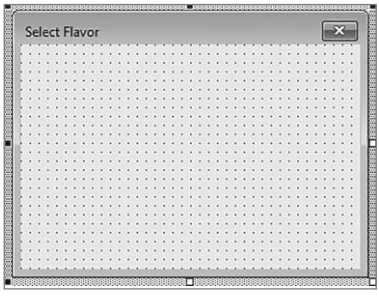
Каждый элемент управления формы, доступный для добавления в пользовательскую форму, имеет большое количество свойств, определяющих его оформление и поведение. В редакторе Visual Basic есть панель **Properties** (Свойства), на которой вы можете посмотреть и изменить значение свойств для элемента, выбранного в окне конструктора. На этой панели есть вкладки, где все свойства указаны в алфавитном порядке, а также поделены на категории. Для каждого свойства указано его имя и текущее значение. Пользователь всегда может изменить значение свойства на панели **Properties** (Свойства). Например, для элементов, имеющих свойство **Caption**, можно ввести новое значение заголовка следующим образом:

Properties.xlsm

1. Создайте пользовательскую форму и перейдите в окно конструктора.

2. Чтобы открыть панель **Properties** (Свойства), выберите команду меню **View ⇨ Properties Window** (Вид ⇨ Окно свойств) или нажмите клавишу **F4**.

3. Выберите свойство **Caption** и в качестве значения напишите текст **Select Flavor**, чтобы изменить заголовок окна пользовательской формы.



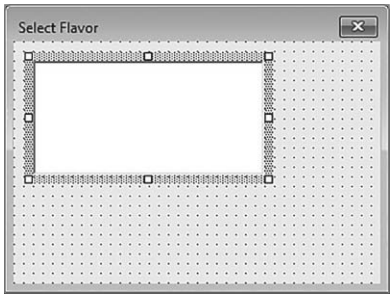
Некоторые свойства позволяют запустить диалоговое окно, в котором можно поменять их значения. Свойство **Font** открывает диалоговое окно, где можно выбрать шрифт, а свойство **Picture** открывает диалоговое окно **Load Picture** (Загрузить изображение).

Текст заголовка пользовательской формы изменится по мере ввода нового текста.

Свойства можно изменить и с помощью макроса VBA. Например, элементы, в которых есть списки, могут быть заполнены при первом открытии пользовательской формы путем добавления инструкций для события **Initialize**.

Если пользователь выбирает несколько элементов, то на панели **Properties** (Свойства) будут отображены лишь общие свойства элементов управления формы.

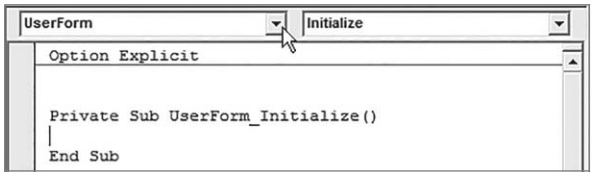
4 В пользовательскую форму добавьте элемент **ListBox**. Изначально списку будет присвоено имя по умолчанию — **ListBox1**.



Чтобы получить доступ к коду пользовательской формы, на панели **Project Explorer** (Обозреватель проектов) выберите узел **UserForm**.

5 В меню выберите команду **View ⇨ Code** (Вид ⇨ Код), чтобы открыть окно для ввода кода.

6 В раскрывающемся списке редактора слева выберите элемент **UserForm** и элемент **Initialize** из раскрывающегося списка справа. Таким образом вы добавите обработчик событий **UserForm_Initialize()**.



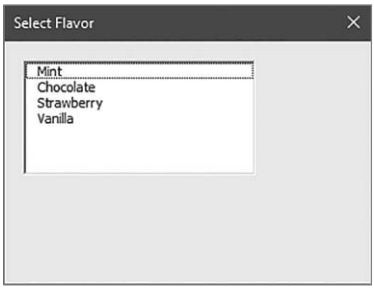
Не вводите несколько инструкций **ListBox1.AddItem** с помощью ключевого слова **With**. Также не забывайте о том, что перед каждым оператором **AddItem** нужно ставить точку.

7 Теперь добавьте показанные ниже инструкции для заполнения элемента **ListBox** четырьмя строками текста.

```
With ListBox1
.AddItem "Mint"
.AddItem "Chocolate"
.AddItem "Strawberry"
```

```
.AddItem "Vanilla"
End With
```

8 Нажмите кнопку **Run** (Выполнить). Появится окно диалоговой формы с вашим заголовком и списком элементов.



Присваивание имен элементам формы

Когда вы добавляете элементы управления формы, они получают имя по умолчанию вместе с порядковым номером. Например, первый добавленный элемент **CommandButton** будет иметь имя **CommandButton1**, второй — **CommandButton2** и так далее. Однако намного лучше присваивать каждому элементу значимые имена, которые легко распознаются в коде VBA. Например, флажок устанавливается с помощью данного кода:

```
CheckBox2.Value = True
```

Однако, если вы присвоите элементу значимое имя, например **chkAsia**, строка станет намного понятнее:

```
chkAsia.Value = True
```

Теперь имя элемента описывает его предназначение, а префикс — тип элемента управления формы. В таблице ниже указаны рекомендуемые префиксы для каждого элемента вместе с примером:

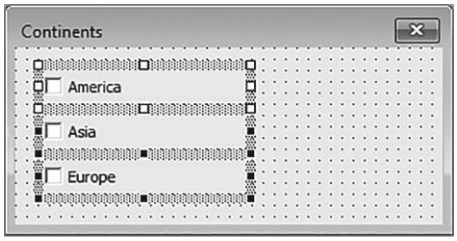
Элемент управления формы	Префикс	Пример имени
CheckBox	chk	chkAsia
ComboBox	cbo	cboFlavors
CommandButton	cmd	cmdAgree
Frame	fra	fraTypes
Image	img	imgLogo
Label	lbl	lblName
ListBox	lst	lstCities
MultiPage	mul	mulPages
OptionButton	opt	optMint
ScrollBar	scr	scrMore
SpinButton	spn	spnQuantity
TabStrip	tab	tabTabs
TextBox	txt	txtName
ToggleButton	tog	togStatus
UserForm	frm	frmMain

Если вы создаете программу с несколькими пользовательскими формами, то лучше присваивать им значимые имена.

Naming.xlsm

1. В пользовательскую форму добавьте три элемента управления формы **CheckBox**.

2. Откройте панели **Properties** (Свойства), затем измените форму и свойство **Caption** каждого элемента управления формы **CheckBox** вот так:



Для выравнивания флажков выделите все три элемента управления формы **CheckBox**, а затем настройте значение их свойства **Left**.

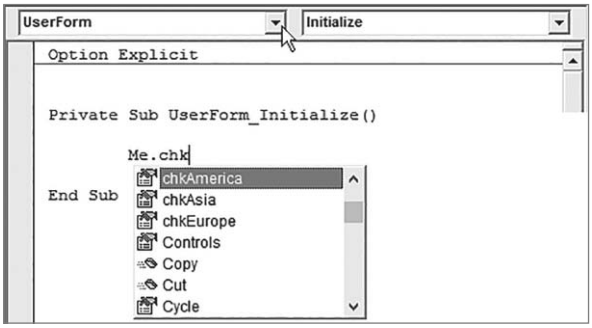
3. Измените свойство **Name** каждого элемента управления формы **CheckBox** на значимые имена, используя префикс **chk**.

4. Выберите команду меню **View ⇨ Code** (Вид ⇨ Код), чтобы открыть окно с кодом пользовательской формы.

5. В раскрывающемся списке слева выберите пункт **UserForm**, а из раскрывающегося списка справа — пункт **Initialize**. Вы

увидите обработчик событий **UserForm_Initialize()**.

6 Внутри кода блока обработчика событий напишите **Me.chk**. Появится подсказка IntelliSense, в которой будут указаны все допустимые элементы с префиксом **chk**.



7 Выберите второй элемент, затем добавьте код: **Me.chkAsia.Value = True**

Ключевое слово **Me** относится к пользовательской форме, поэтому вы можете ввести имя формы и поставить точку. IntelliSense отображает список всех свойств и элементов управления формы, однако с префиксом он сокращается.

Отображение форм

Чтобы отобразить пользовательскую форму, следует создать процедуру в модуле **General**. В таком случае метод **Show** будет работать корректно. К имени, присвоенному форме в коде, прибавляется точка: например, пользовательская форма с именем **MyForm** будет называться **MyForm.Show**. Когда пользовательская форма отображена, она по умолчанию «модальная». Это означает, что если пользователь хочет вернуться к работе с листом, то сначала ему нужно ее закрыть. Вы можете отключить эту функцию, добавив константу **vbModeless** при вызове метода **Show**: например, **MyForm.Show vbModeless**.

Не вставляйте процедуру для отображения формы в код модуля **UserForm**, потому что процедура должна находиться в модуле **General**.

Можно предположить, что при закрытии пользовательской формы используется команда **Close**, однако на самом деле применяется метод **Unload**. Его нужно вставлять перед именем формы, например, **Unload MyForm**. В таком случае пользовательская функция выводится из памяти компьютера. Например, **Load MyForm** загрузит форму в память компьютера,

Не вставляйте процедуру для отображения формы в код модуля **UserForm**, потому что процедура должна находиться в модуле **General**.

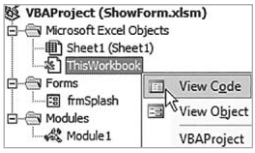
но не отобразит ее, если пользователь будет вызывать метод **Show**. Мы рекомендуем использовать его только в том случае, если пользовательская форма сложная и ее инициализация занимает много времени.

Намного полезнее вызвать метод **Hide**, поскольку тогда пользовательская форма не будет отображаться, но сохранится в оперативной памяти компьютера. Например, с **MyForm.Hide** код имеет доступ к свойствам элементов управления формы.

Пользовательскую форму можно открыть в ответ на событие: например, событие **Click** кнопки на рабочем листе или событие **Open** рабочей книги. Пользователь может использовать форму как экранзаставку при первом открытии книги:

ShowForm.xlsm

Вы можете щелкнуть правой кнопкой мыши по узлу на панели **Project Explorer** (Обозреватель проектов) и выбрать команду **View Code** (Посмотреть код) в контекстном меню. Откроется окно с кодом модуля.



1. Добавьте несколько элементов управления в пользовательскую форму, таких как **Image** и **Label**

2. Измените имя формы на **frmSplash**.

3. Выберите команду меню **View ⇨ Code** (Вид ⇨ Код), чтобы открыть окно с кодом пользовательской формы и добавить таймер.

```
Private Sub UserForm_Initialize()

Application.OnTime Now + _
TimeSerial(0, 0, 5), "CloseSplash"

End Sub
```

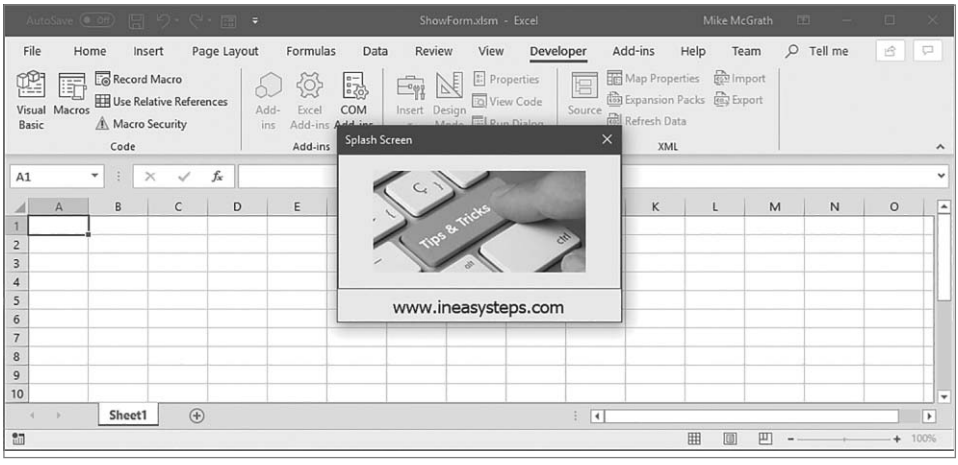
4 Перейдите в модуль узла **ThisWorkbook** (ЭтаКнига) и добавьте процедуру, показывающую пользовательскую форму.

```
Private Sub Workbook_Open()

frmSplash.Show

End Sub
```

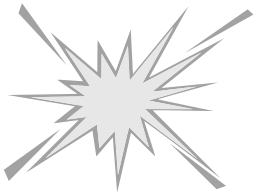
5 Добавьте в проект модуль **General** и показанную ниже процедуру, чтобы закрыть пользовательскую форму.

```
Private Sub CloseSplash()

Unload frmSplash

End Sub
```

6 Сохраните рабочую книгу, затем снова откройте ее. При открытии на пять секунд появится пользовательская форма прямо по центру окна Excel.



Обработка событий формы

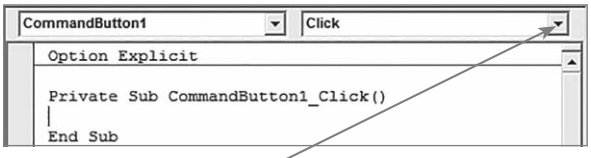


Пользовательская форма и каждый элемент, содержащийся в ней, распознают события, вызываемые действиями пользователя. Вы можете создать код макроса VBA для реакции на любое из событий. Например, когда пользователь нажимает кнопку **CommandButton**, вызывается событие **Click**, на которое вы можете создать ответ в обработчике событий.

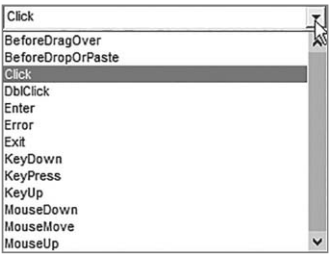
Каждый элемент управления формы вызывает несколько событий, которые можно просмотреть в окне редактора Visual Basic:

1. Добавьте элемент управления **CommandButton** в пользовательскую форму.

2. В окне конструктора дважды щелкните мышью по элементу управления формы **CommandButton**. Откроется окно с кодом с текстовым курсором внутри блока кода созданного обработчика событий **Click()**.



3. Откройте раскрывающийся список справа.



4. Прокрутите список вниз и просмотрите все события для этого элемента управления формы.

Некоторые действия могут вызывать несколько событий. Убедитесь, что вы добавили в код необходимую процедуру обработчика событий.

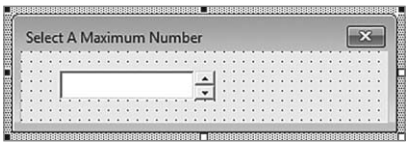
Этот метод можно использовать и для обнаружения событий, вызываемых любым элементом управления формы. Вы можете выбрать любое событие

в раскрывающемся списке, чтобы вставить блок кода обработчика событий.

Один элемент управления формы, которому всегда требуется код обработчика событий, — это **SpinButton**, представляющий собой полосу прокрутки с двумя кнопками со стрелками вверх и вниз, которые пользователи могут нажимать. При каждом нажатии на одну из кнопок запускается событие **Change** элемента управления формы **SpinButton**. Чтобы от него была польза, добавьте в обработчик событий код реакции в ответ на действия пользователя.

FormEvents.xlsm

1 В пользовательскую форму добавьте элементы управления **TextBox** и **SpinButton**, затем назовите их **txtNum** и **spnNum**, соответственно.



2 Выберите элемент управления формы **SpinButton**, затем откройте панель **Properties** (Свойства) и присвойте его свойству **Value** значение 1, а свойству **Min** — значение **1**.

Вы также можете настроить эти свойства с помощью инструкций **spnNum.Value = 1** и **spnNum.Min = 1**.

3 Дважды щелкните мышью по элементу управления формы **SpinButton**. Откроется окно с кодом элемента, в который добавьте показанные ниже инструкции обработчика событий, чтобы реагировать каждый раз, когда пользователь будет нажимать на кнопки вверх и вниз.

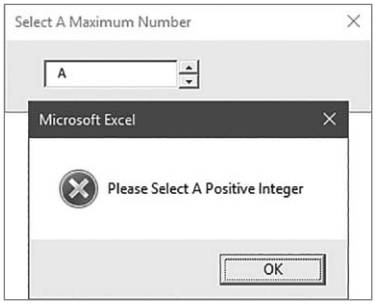
```
Private Sub spnNum_Change()

txtNum.Value = spnNum.Value

End Sub
```

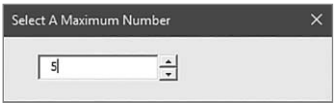
4 Теперь добавьте показанный ниже код обработчика событий, чтобы реагировать каждый раз, когда пользователь будет вводить значение в элемент **TextBox**.

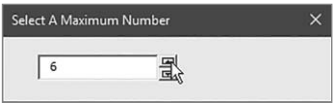
В этом примере значения в элементах **SpinButton** и **TextBox** синхронизируются и меняются, если пользователь просто очищает поле **TextBox**. Если ввести недопустимое значение, появится диалоговое окно, а затем в текстовом поле снова отобразится предыдущее значение.

```
Private Sub txtNum_Change()

If txtNum.Value = "" Then
Exit Sub
ElseIf IsNumeric(txtNum.Value) _
  And(txtNum.Value > 0) Then
spnNum.Value = txtNum.Value
Else
MsgBox "Please Select A Positive Integer", 16
txtNum.Value = spnNum.Value
End If

End Sub
```

5 Нажмите кнопку **Run** (Выполнить) и измените содержимое текстового поля **TextBox** напрямую или с помощью полосы прокрутки **SpinButton**.







Использование списков

Работа метода **AddItem** показана в разделе «Изменение свойств» этой главы.

Элементы **ListBox** и **ComboBox** отличаются от остальных элементов управления формы, так как могут принимать несколько значений. Обычно это осуществляется в коде VBA с помощью метода **AddItem**. В качестве альтернативного варианта можно назначить массив свойству **List** элемента **ListBox** или **ComboBox**.

Каждый элемент в списке имеет порядковый номер, поэтому вы можете выбрать его путем присвоения порядкового номера свойству **ListIndex**. Таким образом, данное свойство содержит номер нужного элемента списка.

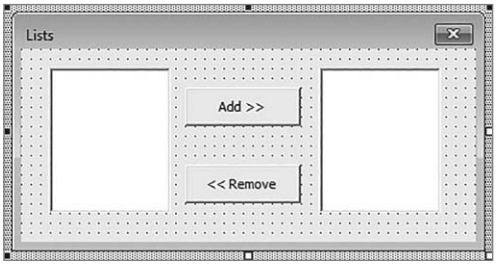
С помощью метода **RemoveItem** можно удалить существующий элемент из списка, указав свойство **ListIndex**. При попытке удалить элемент из пустого списка может возникнуть ошибка. Чтобы этого избежать, убедитесь, что пустых списков нет.

FormList.xlsm

1 В первой строке рабочего листа напишите то, что изображено ниже, чтобы потом эту информацию можно было передать в элемент управления формы **ListBox**.

	A	B	C	D	E	F	
1	Damsons	Elderberries	Figs	Grapes			
2							

2 Откройте редактор Visual Basic и добавьте два элемента **ListBox** и два элемента **CommandButton** в пользовательскую форму.



3 Откройте панель **Properties** (Свойства) и измените значение свойства **Caption** каждого элемента управления формы **CommandButton**, чтобы оно было практически таким же, как на изображении выше.

4 На панели **Properties** (Свойства) присвойте элементам управления формы следующие имена: **lstFruit**, **lstBasket**, **cmdAdd** и **cmdRemove**, соответственно.

5 Откройте окно со связанным кодом формы и начните ее обработчик событий **Initialize()** с заполнения **ListBox** тремя пунктами.

```
Private Sub UserForm_Initialize()
lstFruit.List = Array("Apples", "Bananas", "Cherries")

' Сюда помещаются инструкции (Шаг 6).

End Sub
```

Свойство **CurrentRegion** представляет собой диапазон, ограниченный сочетанием пустых строк или пустых столбцов.

6 Добавьте код, в котором объявляются две переменные и добавляется несколько пунктов в элемент управления формы **ListBox.**

```
Dim cell As Range
Dim rng As Range
Set rng = Range("A1").CurrentRegion
For Each cell In rng
lstFruit.AddItem cell.Value
Next cell
```

7 Добавьте обработчик событий **Click()** для элемента **CommandButton**, чтобы переместить пункты из списка **ListBox.**

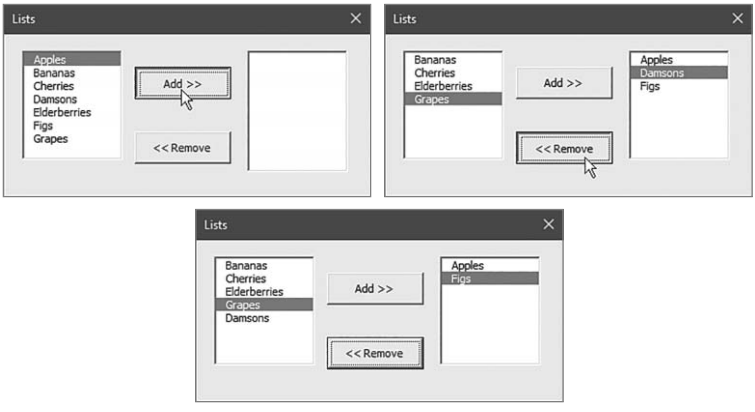
```
Private Sub cmdAdd_Click()
If lstFruit.Value <> "" Then
lstBasket.AddItem lstFruit.Value
lstFruit.RemoveItem listFruit.ListIndex
End If
End Sub
```

8 Добавьте обработчик событий **Click()** для элемента **CommandButton**, чтобы вернуть элементы в список **ListBox.**

```
Private Sub cmdRemove_Click()
If lstBasket.Value <> "" Then
lstFruit.AddItem lstBasket.Value
lstBasket.RemoveItem listBasket.ListIndex
End If
End Sub
```

Пустые двойные кавычки "" — это пустой список.

9 Выполните макрос и щелкайте мышью по кнопкам **CommandButton** для управления списками.



Заключение

- **UserForm** — это настраиваемое диалоговое окно, с помощью которого пользователь может взаимодействовать с книгами Excel.

- Чтобы форма не была бесполезной, функциональный код нужно добавлять в программную часть **UserForm**.

- Визуальные элементы управления формы, доступные для использования в пользовательской форме, находятся на панели **Toolbox** (Инструменты) редактора Visual Basic.

- Элементы управления формы добавляются в пользовательскую форму в **окне конструктора**. Размер элементов управления формы можно изменять, перетаскивая маркеры по краям.

- Элемент управления формы **ListBox** допускает единственный, множественный и связанный выбор элементов из списка.

- Элемент управления формы **OptionButton** (Переключатель) позволяет пользователю выбрать один элемент из списка, а элемент **CheckBox** (Флажок) — несколько элементов.

- На панели **Properties** (Свойства) все свойства выбранного элемента управления формы перечислены в алфавитном порядке, а также разделены по категориям.

- Значения свойств элемента управления формы можно изменить на панели **Properties** (Свойства) во время работы с формой или в коде во время выполнения макроса.

- Рекомендуется присваивать элементам управления формы значимые и понятные имена, поскольку так проще распознать их в коде VBA.

- Пользовательская форма открывается при вызове метода **Show** и закрывается при вызове метода **Unload**.

- По умолчанию все пользовательские формы модальные, поэтому, если у юзера открыта пользовательская форма, то он не сможет вернуться к работе с листом.

- Процедуры обработчика событий могут отзываться на события пользовательской формы. Элементу управления формы **SpinButton** (Полоса прокрутки) обязательно требуется обработчик событий.

- В отличие от других элементов управления формы, **ListBox** и **ComboBox** принимают несколько значений.

- Списки могут быть заполнены пунктами с помощью метода **AddItem** или путем присвоения массива значений их свойству **List**.

- Пункты списка можно удалить с помощью метода **RemoveItem**. А выделить их — с помощью свойства **ListIndex**.

11 Разработка приложений

В этой главе вы узнаете о расширенных функциях, которые используются при разработке приложений Excel с макросами VBA.

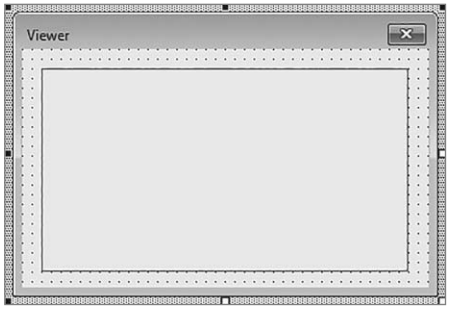
Игнорирование режимов

Как мы выяснили ранее, пользовательские диалоговые окна **UserForm** по умолчанию являются модальными, так что перед тем, как перейти к взаимодействию с рабочим листом, необходимо их закрыть. Если вы этого не хотите, добавьте константу **vbModeless** после вызова метода **Show**.

«Немодальные» пользовательские формы могут быть полезны для отображения или управления некоторыми данными рабочего листа. С их помощью можно также отображать дополнительную информацию, например, изображения из выбранного фрагмента рабочего листа:

Modeless.xlsm

1. В пользовательскую форму добавьте элемент управления формы **Image**, затем присвойте ему имя **imgPhoto**, а пользовательской форме — имя **frmViewer**.



Этому обработчику событий передаются два аргумента — рабочий лист, в котором было изменено выделение, и текущее выделение.

2. На панели **Project Explorer** (Обозреватель проектов) дважды щелкните мышью по узлу **ThisWorkbook** (ЭтаКнига).

3. Из раскрывающегося списка слева выберите пункт **Workbook**, а из списка справа — **SheetSelectionChange**. Таким образом вы добавите обработчик событий, код которого показан ниже.

```
Private Sub Workbook_SheetSelectionChange _
    (ByVal Sh As Object, ByVal Target As Range)
```

```
' Сюда помещаются инструкции (Шаги 4–7).

End Sub
```

4 Добавьте код, предназначенный для обработки ошибок.

```
On Error Resume Next
```

5 Добавьте объявления трех переменных.

```
Dim make As String
Dim model As String
Dim path As String
```

Код **ActiveWorkbook.path** возвращает расположение файла Excel в системе. В этом примере он используется для построения пути к файлу, располагающемся в папке *images*.

6 Теперь добавьте код для инициализации каждой переменной.

```
make = Sh. Cells(Target.Row, 1).Value
model = Sh. Cells(Target.Row, 2).Value
path = ActiveWorkbook.path & _
  "\images\" & model & ".jpg"
```

7 Добавьте код для настройки свойств **Caption** и **Picture** пользовательской формы.

```
frmViewer.Caption = make & " " & model
frmViewer.imgPhoto.Picture = LoadPicture(path)
```

8 В проект добавьте модуль **General**, а затем подпрограмму макроса, которая отобразит пользовательскую форму и активирует обработчик ошибок **Workbook_SheetSelectionChange()**.

```
Sub Viewer()

rmViewer.Show vbModeless

Range("A2").Select

End Sub
```

Обратите внимание, как в данном примере используется функция **LoadPicture()** для назначения изображения элементу управления формы.

9 На рабочий лист добавьте текст и кнопку для выполнения макроса, затем откройте пользовательскую форму и выделяйте строки. При выборе строки название кнопки поменяется и появится соответствующее изображение.

Имя файла каждого изображения соответствует тексту во втором столбце. Все изображения имеют расширение *.jpg*.

	A	B
1	Make	Model
2	Ford	Escape
3	Jeep	Wrangler
4	Toyota	Highlander
5	Porsche	Macan
6	GMC	Acadia
7	Chevrolet	Silverado
8	Buick	Enclave
9	Subaru	Outback
10	Volkswagen	Tiguan

	A	B
1	Make	Model
2	Ford	Escape
3	Jeep	Wrangler
4	Toyota	Highlander
5	Porsche	Macan
6	GMC	Acadia
7	Chevrolet	Silverado
8	Buick	Enclave
9	Subaru	Outback
10	Volkswagen	Tiguan

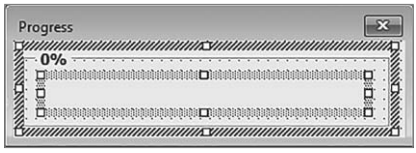
Индикация прогресса

Если выполнение макроса занимает много времени, лучшим решением будет индикатор выполнения — так пользователь сможет отслеживать, выполняет ли Excel код. Обычно это графическая полоса, показывающая процент выполненной работы на текущий момент. Создать индикатор выполнения можно с помощью элемента управления формы **Label** в качестве графической полосы и заголовка **Frame**, который будет отображать процент выполнения:

ProgressBar.xlsm

1. В окно **UserForm** добавьте элементы управления формы **Label** и **Frame**.

2. Пользовательскую форму назовите **frmProgress**, а элементы управления формы — **fraBar** и **lblBar** соответственно.



3. Выберите команду меню **View ⇨ Code** (Вид ⇨ Код), чтобы открыть окно со связанным кодом формы, затем добавьте

показанный ниже код обработчика событий. Таким образом вы установите цвет и размер элемента управления формы **Label** при первом открытии пользовательской формы.

```
Private Sub UserForm_Initialize()

lblBar.BackColor = vbRed
lblBar.Width = 0

End Sub
```

4 На страницу со связанным кодом добавьте показанную ниже подпрограмму, чтобы развернуть элемент управления формы **Label** и отобразить процент выполненной задачи с помощью соответствующего аргумента.

```
Public Sub ReportProgress(perCent As Double)

lblBar.Width = perCent *(fraBar.Width — 10)
fraBar.Caption = Format(perCent, "0%")
Repaint

End Sub
```

Встроенная функция **Format()** изменяет значение аргумента на процентное значение, а метод **Repaint** изменяет оформление элемента управления формы **Label**.

5 В проект добавьте модуль **General**, а в него подпрограмму, которая вставит в ячейки значения.

```
Sub FillCells()

' Сюда помещаются инструкции (Шаги 6–9).

End Sub
```

6 Добавьте инструкции, которые объявят четыре переменные.

```
Dim num As Integer
Dim row As Integer
Dim col As Integer
Dim perCent As Double
```

7 Теперь добавьте инструкции, которые очистят рабочий лист и откроют пользовательскую форму.

```
ActiveSheet.Cells.Clear
frmProgress.Show vbModeless
```

В этом примере пользовательская форма является немодальной, поэтому пользователь может взаимодействовать с рабочим листом, не закрывая при этом пользовательскую форму.

Чтобы вызвать глобальную подпрограмму (**Public**) модуля формы из модуля **General**, укажите префикс имени подпрограммы рядом с именем формы.

8 Добавьте вложенный цикл, который заполнит пять тысяч ячеек значением переменной цикла при каждой итерации.

```
num = 1
For row = 1 To 500
For col = 1 To 10
ActiveSheet.Cells(row, col) = num
num = num + 1
Next col

' Сюда помещаются инструкции (Шаг 10).

Next row
```

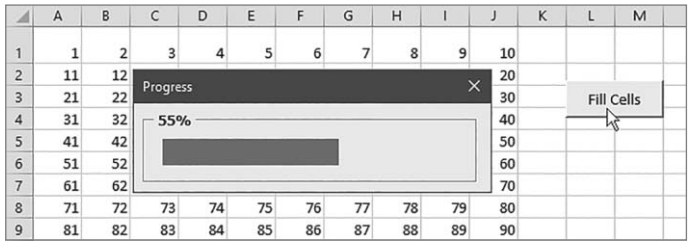
9 Добавьте код, который закроет пользовательскую форму после ее завершения.

```
Unload frmProgress
```

10 Добавьте инструкции, которые посчитают и отобразят процент выполненной задачи.

```
perCent = num / 5000
frmProgress.ReportProgress(perCent)
```

11 На рабочий лист добавьте кнопку для выполнения макроса. Ячейки начнут заполняться и появится индикатор выполнения, отображающий состояние задачи.

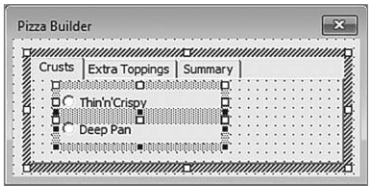


Управление элементами MultiPage

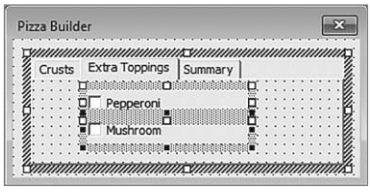
Для пользовательских диалоговых окон, в которых есть несколько элементов управления формы, лучше создать вкладки с помощью элемента **MultiPage**. Каждую группу элементов управления формы можно распределить по отдельным «страницам», благодаря чему пользователь сможет отслеживать отдельные этапы процесса:

1. На пользовательскую форму добавьте элемент управления формы **MultiPage**, затем правой кнопкой мыши щелкните по вкладке и выберите пункт **New Page** (Новая страница). Таким образом будет добавлена третья вкладка.

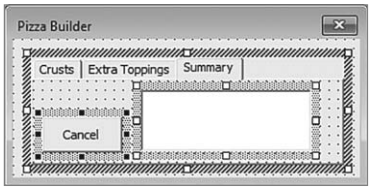
2. На первую страницу добавьте два элемента управления формы **OptionButton** и назовите их **optThn** и **optDpp**.



3. На вторую страницу также добавьте два элемента управления формы **CheckBox** и назовите их **chkPep** и **chkMsh**.



4. На третью страницу добавьте элементы управления формы **CommandButton** и **ListBox** и назовите их **cmdCan** и **lstSum**.



5. Измените свойство **Caption** каждого элемента управления формы согласно изображениям сверху.

6. На открывшейся странице со связанным кодом формы выберите команду меню **View ⇨ Code** (Вид ⇨ Код), затем добавьте показанную ниже подпрограмму для вывода состояний всех элементов управления формы.

Обратите внимание, что условие **If** может быть записано на одной строке при условии, если оператор **Then** имеет только одну инструкцию или несколько, разделенных двоеточием.

```
Sub Summarize()

lstSum.Clear

If optThn.Value = True Then lstSum.AddItem
optThn.Caption
If optDpp.Value = True Then lstSum.AddItem
optDpp.Caption
If chkPep.Value = True Then lstSum.AddItem
chkPep.Caption
If chkMsh.Value = True Then lstSum.AddItem
chkMsh.Caption

End Sub
```

7 Затем добавьте два обработчика событий, которые среагируют в том случае, если пользователь выберет элемент управления формы **OptionButton** на первой странице.

```
Private Sub optThn_Click()
Summarize
End Sub

Private Sub optDpp_Click()
Summarize
End Sub
```

При двойном щелчке мышью по элементу управления формы в окне конструктора в код добавляется обработчик событий **Click** этого элемента.

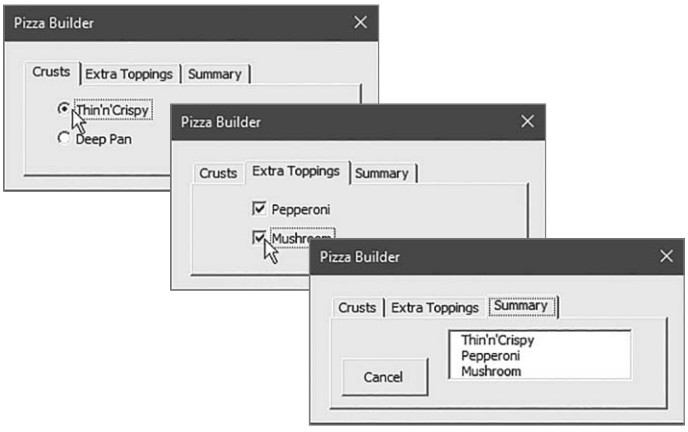
8 Теперь добавьте два обработчика событий, которые среагируют в том случае, если пользователь выберет элемент управления формы **CheckBox** на второй странице.

```
Private Sub chkPep_Click()
Summarize
End Sub

Private Sub chkMsh_Click()
Summarize
End Sub
```

9 Теперь добавьте два обработчика событий, которые среагируют в том случае, если пользователь выберет элемент управления формы **CommandButton** на третьей странице.

```
Private Sub cmdCan_Click()

lstSum.Clear
optThn.Value = False
optDpp.Value = False
chkPep.Value = False
chkMsh.Value = False

End Sub
```

10 Выполните макрос, затем выберите элементы управления формы на первых двух страницах. Вы увидите, что на третьей странице появятся результаты состояний всех элементов управления формы.



Создание вкладок с данными

В отличие от элемента **MultiPage**, представляющего собой разные элементы управления формы на каждой из «страниц», объект **TabStrip** — это те же элементы управления формы на каждой странице. Обычно представляемые данные связаны — например, сведения о продажах из разных регионов:

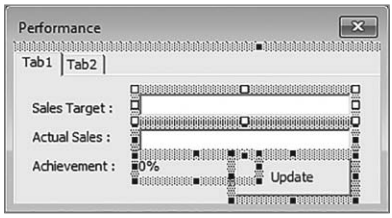
	A	B	C	D	E	
1	Sales	Region 1	Region 2			
2	Target :	$75,000.00	$85,000.00			
3	Actual :	$68,000.00	$81,500.00			
4						

Объект **MultiPage** имеет свойство **Pages**, то есть массив страниц элемента управления формы с нулевой базой. Оно имеет собственное свойство **Controls** — это массив элементов управления формы с нулевой базой. Их можно использовать для ссылки на любой элемент, расположенный на любой странице, например, на заголовок первого элемента управления формы **OptionButton**, расположенного на первой странице в этом примере: ***MultiPageName.*** **Pages (0).Controls(0). Caption**.

DataTabs.xlsm

1. Разместите элемент управления формы **TabStrip** в пользовательской форме. По умолчанию он будет содержать две вкладки.

2. Добавьте две описательные метки, а также два элемента управления формы **TextBox**, и элементы **Label** и **CommandButton** как показано на изображении ниже.



3. Присвойте форме имя **tabPrf**, а элементам управления формы имена **txtTgt**, **txtSls**, **lblPct** и **cmdUpdate** соответственно.

4. Выберите команду меню **View ⇨ Code** (Вид ⇨ Код), чтобы открыть окно со связанным кодом, затем добавьте показанную ниже подпрограмму для настройки элементов управления формы.

```
Sub Populate()

Select Case tabPrf.SelectedItem.Index
Case 0
txtTgt.Value = Range("B2").Value
txtSls.Value = Range("B3").Value
Case 1
txtTgt.Value = Range("C2").Value
txtSls.Value = Range("C3").Value
End Select
lblPct.Caption = Format(txtSls.Value / txtTgt.Value, "0%")

End Sub
```

5. Добавьте обработчик событий, который создаст заголовки вкладок.

```
Private Sub UserForm_Initialize()

tabPrf.Tabs(0).Caption = Range("B1").Value
```


Вкладки — это массив с нулевой базой, так что на нее можно ссылаться по номеру: **SelectedItem.Index**.

На панели **Properties** (Свойства) отображаются свойства **TabStrip**, среди которых нет пункта **Caption** для создания заголовков отдельных вкладок. Их можно настроить в коде.

```
tabPrf.Tabs(1).Caption = Range("C1").Value

End Sub
```

6 Добавьте обработчик событий для повторного заполнения элементов управления формы, когда пользователь выбирает другую вкладку.

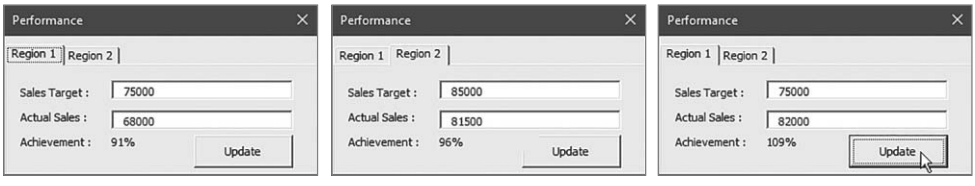
```
Private Sub tabPrf_Change()
Populate
End Sub
```

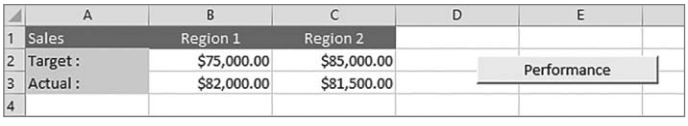
7 Добавьте обработчик событий для обновления рабочего листа и формы после того как пользователь изменит значение в форме.

```
Private Sub cmdUpdate_Click()

Select Case tabPrf.SelectedItem.Index
Case 0
Range("B2").Value = txtTgt.Value
Range("B3").Value = txtSls.Value
Case 1
Range("C2").Value = txtTgt.Value
Range("C3").Value = txtSls.Value
End Select
lblPct.Caption = Format(txtSls.Value / txtTgt.Value,
"0%")

End Sub
```

Для добавления новых вкладок щелкните правой кнопкой мыши по вкладкам в окне конструктора и выберите пункт **New page** (Новая страница) в контекстном меню. Каждая новая вкладка будет иметь идентичные элементы управления формы.

8 На рабочий лист добавьте кнопку для выполнения макроса, затем используйте вкладки пользовательской формы для изменений данных рабочего листа.





Отображение диаграмм

В VBA нет специального элемента управления формы для диаграмм Excel, однако вы можете экспортировать диаграмму как изображение, а затем загрузить его с помощью объекта **Image**. Если пользовательская форма немодальная, то изображение диаграммы может обновляться при изменении данных, представленных на диаграмме:

1. На рабочем листе **Sheet1** (Лист1) выберите данные, а затем создайте встроенную диаграмму. После этого переместите диаграмму на рабочий лист **Sheet2** (Лист2).

2. В пользовательскую форму добавьте элемент **Image** и переименуйте его как **imgChart**. Пользовательской форме присвойте имя **frmInfo**.

3. Выберите команду меню **View ⇨ Code** (Вид ⇨ Код), чтобы открыть окно со связанным кодом, затем добавьте показанную ниже глобальную подпрограмму с областью видимости **Public**, которая объявит две переменные.

```
Public Sub LoadChart()

Dim info As Chart
Dim path As String

' Сюда помещаются инструкции (Шаги 4–6).

End Sub
```

Объект **ChartObjects** — это коллекция встроенных объектов диаграммы.

4. Добавьте код, который присвоит объекту диаграммы переменную.

```
Set info = Sheets("Sheet2").ChartObjects(1).Chart
```

5. Затем добавьте код, чтобы указать имя файла, который будет располагаться во вложенной папке *images*.

```
path = ActiveWorkbook.path & "\images\chart.jpg"
```

6. Теперь добавьте код для загрузки файла изображения, а затем его отображения в пользовательской форме.

```
info.Export path
imgChart.Picture = LoadPicture(path)
```

Путь в примере предполагает, что папка *images* будет располагаться в одном каталоге с файлом Excel.

7. Добавьте обработчик событий для отображения изображения диаграммы при первом открытии пользовательской формы.

```
Private Sub UserForm_Initialize()

LoadChart

End Sub
```

8. На панели **Project Explorer** (Обозреватель проектов) дважды щелкните мышью по узлу **Sheet1** (Лист1), чтобы открыть код модуля, затем добавьте показанный ниже обработчик событий для обновления изображения диаграммы, которое появляется при первом открытии пользовательской формы.

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

If Not frmInfo.Visible Then Exit Sub

frmInfo.LoadChart

End Sub
```

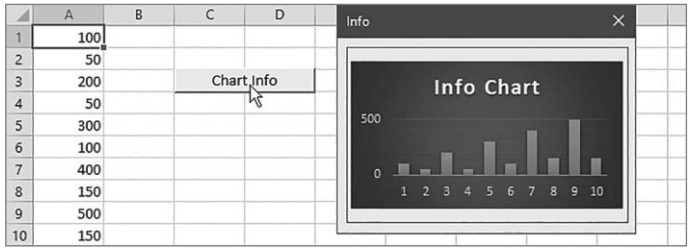
Обратите внимание, что условие **If** может быть записано на одной строке при условии, если оператор **Then** имеет только одну инструкцию или несколько, разделенных двоеточием.

9. Добавьте код модуля VBA в проект и напишите подпрограмму, которая откроет пользовательскую форму немодально.

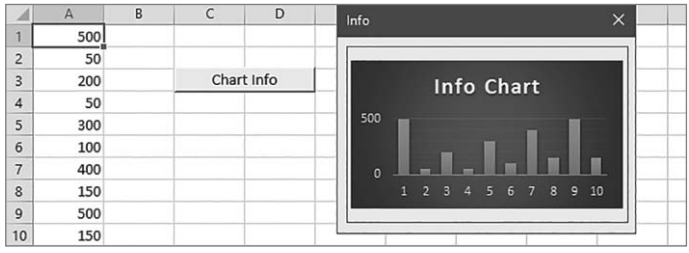
```
Sub ChartInfo()
frmInfo.Show vbModeless
End Sub
```

10. На рабочий лист добавьте кнопку для выполнения макроса. Нажмите ее, после чего отобразится диаграмма.

Объект **Chart** имеет свойство **ChartType**, с помощью которого можно указать тип диаграммы Excel. Например, в шаге 4 добавьте строку **info. ChartType = xlPie**, чтобы изменить диаграмму на круговую. На панели **Object Browser** (Обозреватель объектов) найдите класс **XLChartType**, в котором перечислены все возможные типы диаграмм.



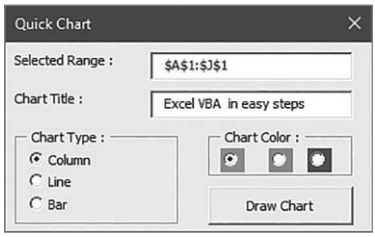
11 На рабочем листе измените данные диаграммы, затем нажмите клавишу **Enter**. Вы увидите, что данные на диаграмме изменятся.



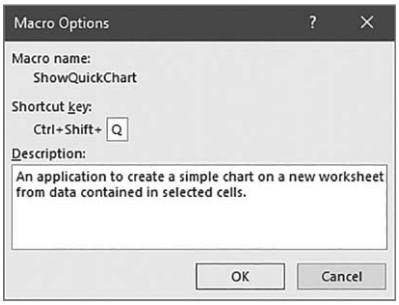
Создание надстроек

Если вы создали программу VBA для распространения другим лицам, то можете преобразовать файл книги в **надстройку** Excel. Создание надстройки позволяет защитить код VBA паролем. В таком случае при использовании надстройки окно книги не будет отображаться, однако пользователь может получить доступ к коду макроса, чтобы запустить программу:

1 Создайте и внимательно проверьте программу на наличие ошибок. В данном случае это программа для быстрого создания простой диаграммы из данных, содержащихся в выбранных ячейках листа.

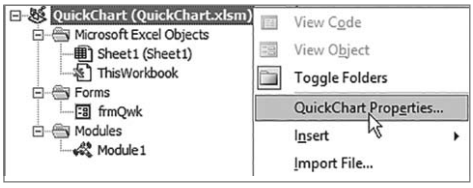


2 Выберите команду меню **Developer ⇨ Macros ⇨ Options** (Разработчик ⇨ Макросы ⇨ Параметры) и назначьте сочетания клавиш, если хотите защитить программу паролем.

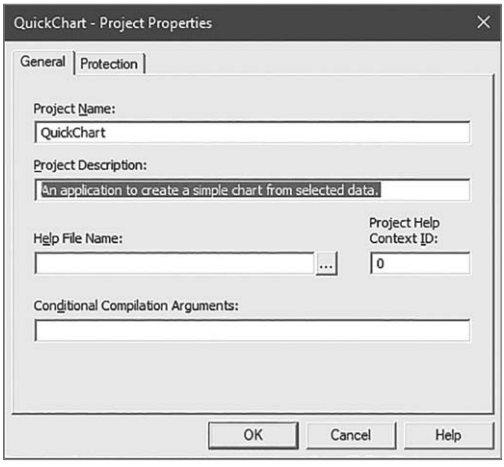


Без сочетания клавиш пользователю будет сложнее запустить защищенную надстройку Excel.

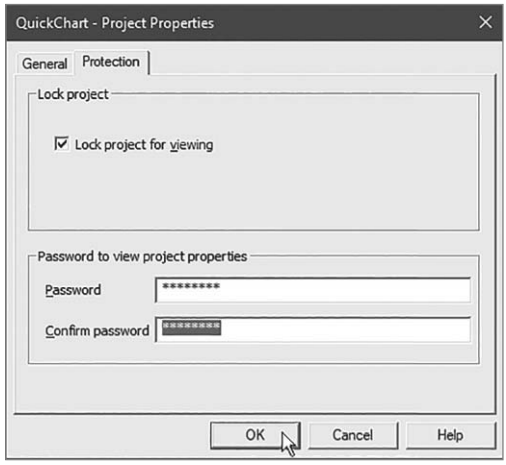
3 Откройте редактор Visual Basic, затем на панели **Project Explorer** (Обозреватель проектов) дважды щелкните мышью по узлу проекта и выберите пункт **Properties** (Свойства) в контекстном меню.



4 Перейдите на вкладку **General** (Общее), затем измените дефолтное имя проекта на название своей программы и напишите краткое описание — для чего нужна данная программа.



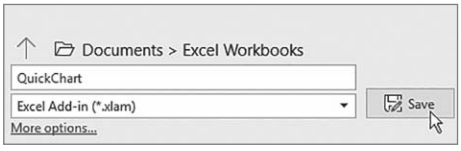
5 Если вы захотите установить пароль для программы, то перейдите на вкладку **Protection** (Защита) и установите флажок **Lock project for viewing** (Заблокировать проект для просмотра). Дважды введите пароль и нажмите кнопку **OK**.



Блокирование проекта не дает гарантию абсолютной защиты документа.

6 Вернитесь в Excel, затем выберите команду меню **File ⇨ Save As** (Файл ⇨ Сохранить как) и в раскрывающемся списке выберите тип файла **Excel Add-in (*.xlam)** (Надстройка Excel).

7 Выберите расположение, в котором вы хотите создать надстройку Excel, а затем нажмите кнопку **Save** (Сохранить).



QuickChart.xlam

Обычно Excel предлагает сохранять надстройки в каталоге по пути **C:\Users\ *username*\AppData\ Roaming\ Microsoft\ Addins**, однако вы можете сохранять их где угодно.

Установка надстроек

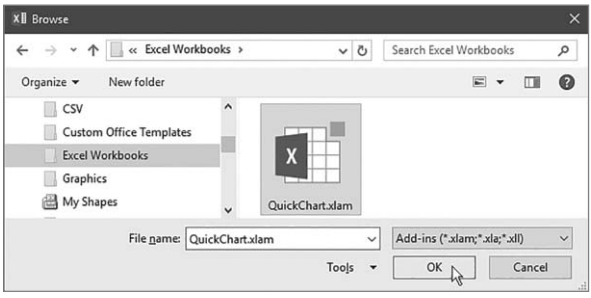
Создав надстройку согласно плану, описанному в предыдущем разделе, ее можно распространить другим лицам. Они могут установить вашу надстройку в программу Excel согласно инструкциям по ее использованию. Если она защищена паролем и пользователи его не знают, то они не смогут просматривать или редактировать код макроса. Позволит запустить макрос установка надстроек:

QuickData.xlsm

1. Откройте Excel, затем выберите команду меню **File** ⇨ **Options** (Файл ⇨ Параметры) и выберите элемент **Adds-ins** (Надстройки) на панели слева.

2. В самом низу панели справа в раскрывающемся списке **Manage** (Управление) выберите пункт **Excel Add-ins** (Надстройки Excel), а затем нажмите кнопку **Go** (Перейти). Откроется диалоговое окно **Add-ins** (Надстройки).

3. Нажмите кнопку **Browse** (Обзор), перейдите в папку с файлом надстройки и нажмите кнопку **OK**, чтобы установить надстройку.



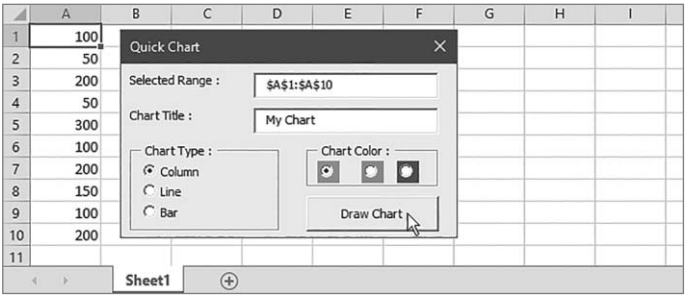
Некоторые надстройки могут отображаться в диалоговом окне надстроек Excel, однако они не будут активны, пока вы не установите флажок напротив нужной надстройки.

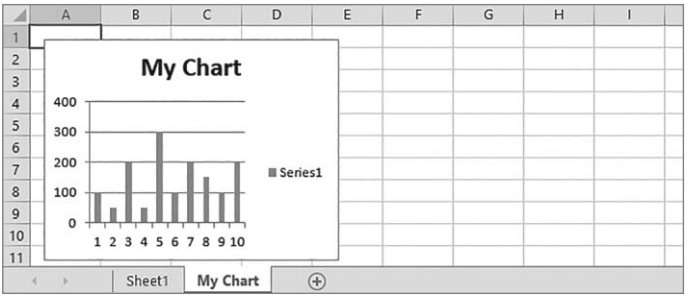
Вы можете удалить надстройку, переименовав или удалив файл *.xlam*, а также сбросив флажок в диалоговом окне **Adds-ins** (Надстройки). Появится диалоговое окно с запросом, точно ли вы хотите удалить надстройку из списка.

При создании надстройки нужно назначить сочетание клавиш — тогда пользователь сможет запустить макрос. В ином случае рабочий лист будет для него недоступен.

4 Надстройка Excel появится в списке диалогового окна **Adds-ins** (Надстройки). Проверьте, установлен ли флажок, обозначающий, что надстройка установлена.

5 Используйте назначенное надстройке сочетание клавиш и запустите макрос. В нашем примере используется сочетание **Ctrl + Shift + Q**.





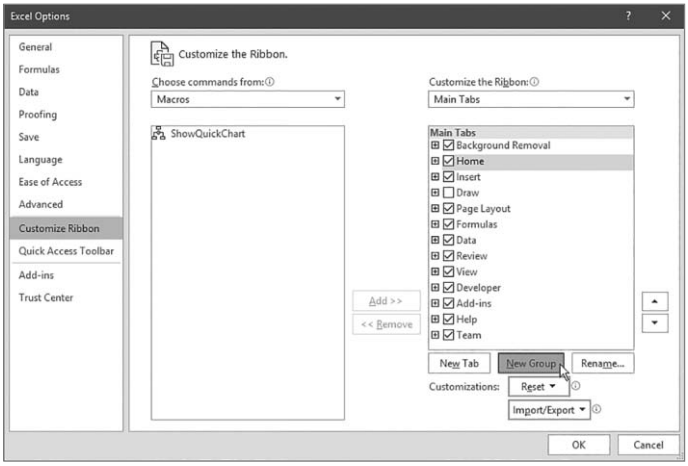
Добавление кнопок на ленту программы

При использовании незащищенной паролем надстройки пользователь может добавить на ленту кнопку для выполнения макроса:

1. Откройте Excel, затем выберите команду меню **File ⇨ Options** (Файл ⇨ Параметры). Откроется диалоговое окно **Excel Options** (Параметры Excel). На панели слева выберите пункт **Customize Ribbon** (Настройка ленты).

QuickData.xlsm

2. В раскрывающемся списке **Choose commands from** (Выбрать команды) выберите пункт **Macros** (Макросы). Появится список макросов.



3. На панели справа выберите пункт **Home** (Главная), затем нажмите кнопку **New group** (Создать группу).

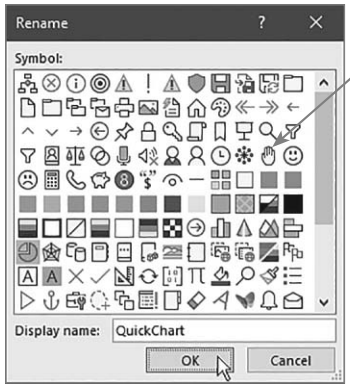
4. Нажмите кнопку **Rename** (Переименовать) и введите любое имя группы (мы использовали имя **In Easy Steps**) в открывшемся диалоговом окне. Затем нажмите кнопку **OK**.

Макрокоманда не появится в списке, если макрос защищен паролем. Пользователь не сможет добавить кнопку для макроса на свою ленту, однако он все еще может использовать сочетания клавиш.

5 На правой панели выберите новую группу и макрос на панели в центре (**ShowQuickChart**), затем нажмите кнопку **Add** (Добавить). В группу добавится макрокоманда.

6 На правой панели выберите макрокоманду, затем нажмите кнопку **Rename** (Переименовать), чтобы заново открыть диалоговое окно.

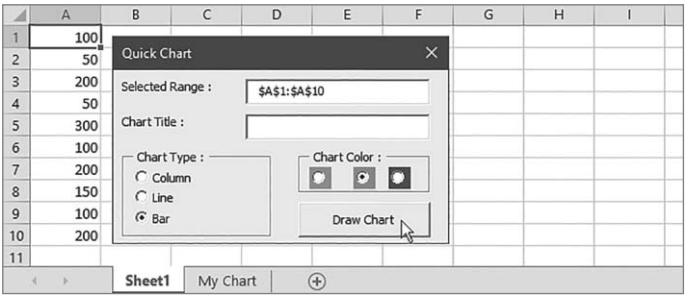
7 Измените отображаемое имя кнопки на любое другое, затем выберите значок кнопки.



8 Нажмите кнопку **OK** для закрытия диалогового окна **Rename** (Переименовать).

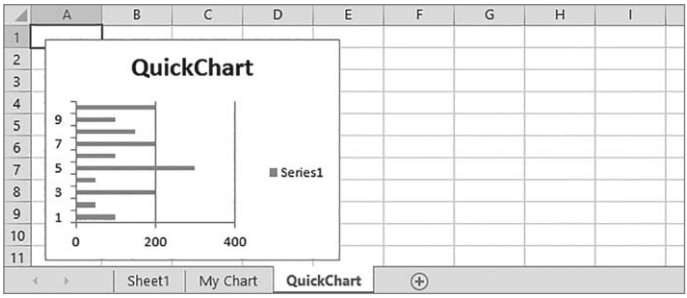
9 Нажмите кнопку **OK** для закрытия диалогового окна **Excel Options** (Параметры Excel). На вкладке **Home** (Главная) в новой группе появится кнопка.

10 Добавьте кнопку для выполнения макроса.



Вы можете в любой момент вернуться в диалоговое окно **Rename** (Переименовать) и изменить имя или значок кнопки.

Этот макрос заблокирован для выбора заголовка диаграмм в случае, если пользователь оставляет поле элемента **TextBox** пустым. Все примеры, приведенные в этой книге, доступны для скачивания по ссылке: **www.ineasysteps.com/resource-centre/ downloads**



Заключение

- Использование константы **vbModeless** позволяет продолжать работу с листом без закрытия пользовательской формы.

- Немодальная пользовательская форма может быть полезна для отображения информации текущего рабочего листа.

- Элемент управления формы **Label** позволяет визуально отмечать ход выполнения задачи, а с помощью элемента управления формы **Frame** можно отображать числовое значение выполнения задачи.

- Метод **Repaint** можно использовать для изменения оформления элемента управления формы.

- Для создания вкладок в пользовательских диалоговых окнах с несколькими элементами управления формы лучше использовать объект **MultiPage**.

- Каждая страница объекта **MultiPage** — это отдельные элементы управления формы, благодаря чему намного проще отслеживать отдельные этапы процесса.

- Каждая страница объекта **TabStrip** представляет одинаковые элементы управления формы для отображения различных данных.

- В диалоговом окне пользовательской формы Excel нет элемента управления формы для работы с диаграммами Excel.

- Диаграмму можно экспортировать в изображение, а затем загрузить его в элемент управления формы **Image** для отображения в пользовательском диалоговом окне.

- Немодальная пользовательская форма с диаграммой может динамически обновляться при изменении данных формы.

- Программа Excel VBA может быть распространена путем предоставления общего доступа к копии книги или с помощью **надстроек**.

- Дополнительные файлы **надстроек** Excel можно защитить паролем.

- Пользователь не видит рабочую книгу, однако может получить доступ к коду макроса для запуска программы.

- При создании **надстройки** следует присвоить ей сочетание клавиш, чтобы пользователь мог выполнить макрос.

- Перед запуском кода макроса надстройки пользователь должен установить ее в свою программу Excel.

- Пользователь может добавить кнопку для запуска незащищенной паролем **надстройки** в пользовательскую группу на **ленте** Excel.

Алфавитный указатель

И

К

Л

М

Н

О

П

Р

С

Т

У

Ф

Ц

Ч

Э

Производственно-практическое издание

EXCEL ДЛЯ ВСЕХ

МакГрат Майк

EXCEL VBA

СТАНЬ ПРОДВИНУТЫМ ПОЛЬЗОВАТЕЛЕМ ЗА НЕДЕЛЮ

Главный редактор *Р. Фасхутдинов*
Руководитель направления *В. Обручев*
Ответственный редактор *Е. Истомина*
Литературный редактор *Н. Домнина*
Младший редактор *А. Захарова*
Художественный редактор *А. Шуклин*
Компьютерная верстка *Э. Брегис*
Корректор *Л. Макарова*

Страна происхождения: Российская Федерация
Шығарылған елі: Ресей Федерациясы

В оформлении обложки использованы фотографии:
Pataradon Luangtongkum, Good dreams - Studio / Shutterstock.com
Используется по лицензии от Shutterstock.com

ООО «Издательство «Эксмо»
123308, Россия, город Москва, улица Зорге, дом 1, строение 1, этаж 20, каб. 2013.
Тел.: 8 (495) 411-68-86.
Home page: www.eksmo.ru E-mail: info@eksmo.ru
Өндіруші: «ЭКСМО» АҚБ Баспасы,
123308, Ресей, қала Мәскеу, Зорге көшесі, 1 үй, 1 ғимарат, 20 қабат, офис 2013 ж.
Тел.: 8 (495) 411-68-86.
Home page: www.eksmo.ru E-mail: info@eksmo.ru.
Тауар белгісі: «Эксмо»
Интернет-магазин : www.book24.ru

Интернет-магазин : www.book24.kz
Интернет-дукен : www.book24.kz
Импортёр в Республику Казахстан ТОО «РДЦ-Алматы».
Қазақстан Республикасындағы импорттаушы «РДЦ-Алматы» ЖШС.
Дистрибьютор и представитель по приему претензий на продукцию,
в Республике Казахстан: ТОО «РДЦ-Алматы»
Қазақстан Республикасында дистрибьютор және өнім бойынша арыз-талаптарды
қабылдаушының өкілі «РДЦ-Алматы» ЖШС,
Алматы қ., Домбровский көш., 3«а», литер Б, офис 1.
Тел.: 8 (727) 251-59-90/91/92; E-mail: RDC-Almaty@eksmo.kz
Өнімнің жарамдылық мерзімі шектелмеген.
Сертификация туралы ақпарат сайтта: www.eksmo.ru/certification

Сведения о подтверждении соответствия издания согласно законодательству РФ
о техническом регулировании можно получить на сайте Издательства «Эксмо»
www.eksmo.ru/certification
Өндірген мемлекет: Ресей. Сертификация қарастырылмаған

Дата изготовления / Подписано в печать 18.11.2021.
Формат 70x100 1/16. Печать офсетная. Усл. печ. л. 19,44.
Тираж экз. Заказ

ISBN 978-5-04-121944-4

Москва. ООО «Торговый Дом «Эксмо»
Адрес: 123308, г. Москва, ул. Зорге, д.1, строение 1.
Телефон: +7 (495) 411-50-74. **E-mail:** reception@eksmo-sale.ru

По вопросам приобретения книг «Эксмо» зарубежными оптовыми покупателями обращаться в отдел зарубежных продаж ТД «Эксмо»
E-mail: **international@eksmo-sale.ru**

International Sales: International wholesale customers should contact Foreign Sales Department of Trading House «Eksmo» for their orders.
international@eksmo-sale.ru

По вопросам заказа книг корпоративным клиентам, в том числе в специальном оформлении, обращаться по тел.: +7 (495) 411-68-59, доб. 2261.
E-mail: **ivanova.ey@eksmo.ru**

Оптовая торговля бумажно-беловыми и канцелярскими товарами для школы и офиса «Канц-Эксмо»:
Компания «Канц-Эксмо»: 142702, Московская обл., Ленинский р-н, г. Видное-2, Белокаменное ш., д. 1, а/я 5. Тел./факс: +7 (495) 745-28-87 (многоканальный).
e-mail: **kanc@eksmo-sale.ru**, сайт: www.**kanc-eksmo.ru**

Филиал «Торгового Дома «Эксмо» в Нижнем Новгороде
Адрес: 603094, г. Нижний Новгород, улица Карпинского, д. 29, бизнес-парк «Грин Плаза»
Телефон: +7 (831) 216-15-91 (92, 93, 94). **E-mail**: reception@eksmonn.ru

Филиал ООО «Издательство «Эксмо» в г. Санкт-Петербурге
Адрес: 192029, г. Санкт-Петербург, пр. Обуховской обороны, д. 84, лит. «Е»
Телефон: +7 (812) 365-46-03 / 04. **E-mail**: server@szko.ru

Филиал ООО «Издательство «Эксмо» в г. Екатеринбурге
Адрес: 620024, г. Екатеринбург, ул. Новинская, д. 2щ
Телефон: +7 (343) 272-72-01 (02/03/04/05/06/08)

Филиал ООО «Издательство «Эксмо» в г. Самаре
Адрес: 443052, г. Самара, пр-т Кирова, д. 75/1, лит. «Е»
Телефон: +7 (846) 207-55-50. **E-mail**: RDC-samara@mail.ru

Филиал ООО «Издательство «Эксмо» в г. Ростове-на-Дону
Адрес: 344023, г. Ростов-на-Дону, ул. Страны Советов, 44А
Телефон: +7(863) 303-62-10. **E-mail**: info@rnd.eksmo.ru

Филиал ООО «Издательство «Эксмо» в г. Новосибирске
Адрес: 630015, г. Новосибирск, Комбинатский пер., д. 3
Телефон: +7(383) 289-91-42. E-mail: eksmo-nsk@yandex.ru

Обособленное подразделение в г. Хабаровске
Фактический адрес: 680000, г. Хабаровск, ул. Фрунзе, 22, оф. 703
Почтовый адрес: 680020, г. Хабаровск, А/Я 1006
Телефон: (4212) 910-120, 910-211. **E-mail**: eksmo-khv@mail.ru

Филиал ООО «Издательство «Эксмо» в г. Тюмени
Центр оптово-розничных продаж Cash&Carry в г. Тюмени
Адрес: 625022, г. Тюмень, ул. Пермякова, 1а, 2 этаж. ТЦ «Перестрой-ка»
Ежедневно с 9.00 до 20.00. Телефон: 8 (3452) 21-53-96

Республика Беларусь: ООО «ЭКСМО АСТ Си энд Си»
Центр оптово-розничных продаж Cash&Carry в г. Минске
Адрес: 220014, Республика Беларусь, г. Минск, проспект Жукова, 44, пом. 1-17, ТЦ «Outleto»
Телефон: +375 17 251-40-23; +375 44 581-81-92
Режим работы: с 10.00 до 22.00. **E-mail:** exmoast@yandex.by

Казахстан: «РДЦ Алматы»
Адрес: 050039, г. Алматы, ул. Домбровского, 3А
Телефон: +7 (727) 251-58-12, 251-59-90 (91,92,99). E-mail: RDC-Almaty@eksmo.kz

Украина: ООО «Форс Украина»
Адрес: 04073, г. Киев, ул. Вербовая, 17а
Телефон: +38 (044) 290-99-44, (067) 536-33-22. **E-mail**: sales@forsukraine.com

Полный ассортимент продукции ООО «Издательство «Эксмо» можно приобрести в книжных магазинах «Читай-город» и заказать в интернет-магазине: www.chitai-gorod.ru.
Телефон единой справочной службы: 8 (800) 444-8-444. Звонок по России бесплатный.

Интернет-магазин ООО «Издательство «Эксмо»
www.book24.ru
Розничная продажа книг с доставкой по всему миру.
Тел.: +7 (495) 745-89-14. E-mail: **imarket@eksmo-sale.ru**

ЛУЧШИЕ КНИГИ О БИЗНЕСЕ С ЛОГОТИПОМ ВАШЕЙ КОМПАНИИ? ЛЕГКО!

Удивить своих клиентов, бизнес-партнеров, сделать памятный подарок сотрудникам и рассказать о своей компании читателям бизнес-литературы? Приглашаем стать партнерами выпуска актуальных и популярных книг. О вашей компании узнает наиболее активная аудитория.

ПАРТНЕРСКИЕ ОПЦИИ:

- Специальный тираж уже существующих книг с логотипом вашей компании.
- Размещение логотипа на супер-обложке для малых тиражей (от 30 штук).
- Поддержка выхода новинки, которая ранее не была доступна читателям (50 книг в подарок).

ПАРТНЕРСКИЕ ВОЗМОЖНОСТИ:

- Рекламная полоса о вашей компании внутри книги.
- Вступительное слово в книге от первых лиц компании-партнера.
- Обращение первых лиц на суперобложке.
- Отзыв на обороте обложки вложение информационных материалов о вашей компании (закладки, листовки, мини-буклеты).

У вас есть возможность обсудить свои пожелания с менеджерами корпоративных продаж. Как?

Звоните:
+7 495 411 68 59, доб. 2261

Заходите на сайт:
eksmo.ru/b2b

ПУСТЬ EXCEL СТАНЕТ ПОНЯТНЕЕ!

В этом самоучителе для начинающих просто и доступно рассказывается о том, как научиться эффективно использовать программу Microsoft Excel, программируя макросы VBA (Visual Basic для приложений).

ИЗ ЭТОЙ КНИГИ ВЫ УЗНАЕТЕ, КАК:

- Писать и изменять собственные макросы Excel
- Использовать операторы, процедуры VBA для автоматизации повторяющихся задач, а также функций и команд вне Excel
- Создавать, распространять свои программы Excel VBA и удивлять коллег!

БЕСПЛАТНЫЕ ПРИМЕРЫ ДЛЯ СКАЧИВАНИЯ ВНУТРИ

ISBN 978-5-04-121944-4

9 785041 219444 >

БОМБОРА
ИЗДАТЕЛЬСТВО

БОМБОРА – лидер на рынке полезных и вдохновляющих книг.
Мы любим книги и создаем их, чтобы вы могли творить, открывать мир, пробовать новое, расти. Быть счастливыми. Быть на волне.

bomborabooks bombora.ru