

Microsoft Excel VBA and Macros

Your guide to efficient automation

Tracy Syrstad and Bill Jelen



Data sets and code files
on the web

FREE SAMPLE CHAPTER



Microsoft Excel VBA and Macros: Your guide to efficient automation

Tracy Syrstad
Bill Jelen

Microsoft Excel VBA and Macros: Your guide to efficient automation

Published with the authorization of Microsoft Corporation by:

Pearson Education, Inc.

Copyright © 2026 by Pearson Education, Inc.

Hoboken, New Jersey

All rights reserved. This publication is protected by copyright, and permission must be obtained from the publisher prior to any prohibited reproduction, storage in a retrieval system, or transmission in any form or by any means, electronic, mechanical, photocopying, recording, or likewise. For information regarding permissions, request forms, and the appropriate contacts within the Pearson Education Global Rights & Permissions Department, please visit www.pearson.com/global-permission-granting.html.

No patent liability is assumed with respect to the use of the information contained herein. Although every precaution has been taken in the preparation of this book, the publisher and author assume no responsibility for errors or omissions. Nor is any liability assumed for damages resulting from the use of the information contained herein.

ISBN-13: 978-0-13-541023-3

ISBN-10: 0-13-541023-1

Library of Congress Control Number: 2025943857

\$PrintCode

Trademarks

Microsoft and the trademarks listed at www.microsoft.com on the “Trademarks” webpage are trademarks of the Microsoft group of companies. All other marks are property of their respective owners. Figures 18-1 and 18-2 © 2025 Spotify AB

Warning and Disclaimer

Every effort has been made to make this book as complete and as accurate as possible, but no warranty or fitness is implied. The information provided is on an “as is” basis. The author, the publisher, and Microsoft Corporation shall have neither liability nor responsibility to any person or entity with respect to any loss or damages arising from the information contained in this book or from the use of the programs accompanying it.

EXECUTIVE EDITOR

Loretta Yates

ACQUISITIONS EDITOR

Shourav Bose

DEVELOPMENT EDITOR

Rick Kughen

MANAGING EDITOR

Sandra Schroeder

SENIOR PROJECT EDITOR

Tracey Croom

COPY EDITOR

Rick Kughen

INDEXER

Rachel Kuhn

PROOFREADER

Donna E. Mulder

TECHNICAL EDITOR

Bob Umlas

COVER DESIGNER

Twist Creative, Seattle

COMPOSITOR

codeMantra

COVER ILLUSTRATION

ImageFlow/Shutterstock

Dedication

For Tracy Syrstad. Thanks for being a great coauthor for 20 years!

—Bill Jelen

To the clients who made this book possible—Marlee Jo J., Dale W., Eddie G., and all the others over the years—thank you for the challenging projects, sharp questions, and unexpected curveballs. You didn't just keep me on my toes—you made me better.

—Tracy Syrstad

Contents at a Glance

	Acknowledgments	xxiii
	About the Authors	xxv
	Introduction	xxvii
CHAPTER 1	Unleashing the power of Excel with VBA	1
CHAPTER 2	This sounds like BASIC, so why doesn't it look familiar?	25
CHAPTER 3	Referring to ranges, names, and tables	53
CHAPTER 4	Laying the groundwork with variables and structures	83
CHAPTER 5	Looping and flow control	99
CHAPTER 6	R1C1 style formulas	117
CHAPTER 7	Event programming	127
CHAPTER 8	Arrays	141
CHAPTER 9	Creating custom objects and collections	153
CHAPTER 10	Userforms: An introduction	175
CHAPTER 11	Data mining with Advanced Filter	197
CHAPTER 12	Using VBA to create pivot tables	239
CHAPTER 13	Excel power	293
CHAPTER 14	Sample user-defined functions	331
CHAPTER 15	Creating charts	357
CHAPTER 16	Data visualizations and conditional formatting	383
CHAPTER 17	Dashboarding with sparklines in Excel	407
CHAPTER 18	Reading from the web using M and VBA	429
CHAPTER 19	Text file processing	459
CHAPTER 20	Automating Word	473
CHAPTER 21	Using Access as a back end to enhance multiuser access to data	497
CHAPTER 22	Advanced userform techniques	519
CHAPTER 23	The Windows Application Programming Interface (API)	543
CHAPTER 24	Handling errors	553
CHAPTER 25	Customizing the ribbon to run macros	573
CHAPTER 26	Creating Excel add-ins	597
CHAPTER 27	An introduction to creating Office Add-ins	607
	Index	637

Contents

<i>Acknowledgments</i>	xxiii
<i>About the Authors</i>	xxv
<i>Introduction</i>	xxvii

Chapter 1	Unleashing the power of Excel with VBA	1
Barriers to entry		1
The macro recorder doesn't work!		1
No one person on the Excel team is focused on the macro recorder		2
Visual Basic is not like BASIC		2
Good news: Climbing the learning curve is easy		3
Great news: Excel with VBA is worth the effort		3
Knowing your tools: The Developer tab		3
Understanding which file types allow macros		4
Macro security		6
Adding a trusted location		6
Using macro settings to enable macros in workbooks outside trusted locations		7
Using Disable All Macros With Notification		8
Overview of recording, storing, and running a macro		8
Filling out the Record Macro dialog		8
Running a macro		10
Understanding the VB Editor		10
VB Editor settings		10
The Project Explorer		11
The Properties window		12
Understanding the shortcomings of the macro recorder		12
Recording the macro		14
Examining code in the Programming window		14
Running the macro on another day produces undesired results ...		16
Possible solution: Use relative references when recording		17

Never use AutoSum or Quick Analysis while recording a macro . . .	21
Four tips for using the macro recorder	22
Next steps	23
 Chapter 2 This sounds like BASIC, so why doesn't it look familiar?	 25
Understanding the parts of VBA "speech"	26
VBA is not really hard	30
VBA Help files: Using F1 to find anything	30
Using Help topics	30
Examining recorded macro code: Using the VB Editor and Help	31
Optional parameters	32
Defined constants	33
Properties can return objects	36
Using debugging tools to figure out recorded code	36
Stepping through code	36
More debugging options: Breakpoints	38
Backing up or moving forward in code	38
Not stepping through each line of code	39
Querying anything while stepping through code	39
Using a watch to set a breakpoint	42
Using a watch on an object	42
Object Browser: The ultimate reference	43
Seven tips for cleaning up recorded code	45
Tip 1: Don't select anything	45
Tip 2: Use Cells(2, 5) because it's more convenient than Range("E2")	46
Tip 3: Use more reliable ways to find the last row	46
Tip 4: Use variables to avoid hard-coding rows and formulas.	48
Tip 5: Use R1C1 formulas that make your life easier.	48
Tip 6: Copy and paste in a single statement	49
Tip 7: Use With . . . End With to perform multiple actions	49
Next steps	52

Chapter 3	Referring to ranges, names, and tables	53
	The Range object	54
	Syntax for specifying a range	54
	Referencing named ranges	55
	Referencing a range relative to another range	55
	Using the Cells property to select a range	56
	Using the Offset property to refer to a range	57
	Using the Resize property to change the size of a range	59
	Using the Columns and Rows properties to specify a range	61
	Using the Union method to join multiple ranges	61
	Using the Intersect method to create a new range from overlapping ranges	62
	Using the IsEmpty function to check whether a cell is empty	63
	Using the CurrentRegion property to select a data range	64
	Using the Areas collection to return a noncontiguous range	67
	Referencing ranges in other sheets	68
	Working with defined names	69
	Global versus local names	69
	Creating defined names	70
	Managing defined names	75
	Deleting names	78
	Reading names	78
	Referencing tables, a.k.a. ListObjects	80
	Adding a ListObject	80
	Working with ListObjects	81
	Next steps	82
Chapter 4	Laying the groundwork with variables and structures	83
	Declaring and assigning variables	83
	Variable data types	84
	Variable lifetime and scope	85
	Declaring constants for safety and clarity	86
	Using Set to assign object variables	86

Releasing object variables	88
Controlling argument passing with ByVal and ByRef	88
Why object variables are different	89
Managing code variations with compiler directives	91
Understanding Subs and Functions	93
Using Enum to create custom constants	94
Declaring parameters using an Enum	95
Using With statements to streamline code	96
Nesting With...End With statements	96
Switching objects within With...End With blocks	97
Next steps	97
Chapter 5 Looping and flow control	99
For...Next loops	100
Using variables in the For statement	102
Variations on the For...Next loop	103
Exiting a loop early after a condition is met	104
Nesting one loop inside another loop	104
Do loops	105
Using the While or Until clause in Do loops	108
The VBA loop: For Each	110
Flow control: Using If...Then...Else and Select Case	110
Basic flow control: If...Then...Else	111
Using Select Case...End Select for multiple conditions	113
Next steps	116
Chapter 6 R1C1 style formulas	117
Toggling to R1C1 style references	118
Witnessing the miracle of Excel formulas	119
Entering a formula once and copying 1,000 times	119
The secret: It's not that amazing	120
Understanding the R1C1 reference style	121
Using R1C1 with relative references	121
Using R1C1 with absolute references	122

Using R1C1 with mixed references	123
Referring to entire columns or rows with R1C1 style	123
Replacing many A1 formulas with a single R1C1 formula	123
Remembering column numbers associated with column letters ..	125
Next steps	126
Chapter 7 Event programming	127
Levels of events	127
Using events	128
Event parameters	128
Disabling and enabling events	129
Workbook events	129
Workbook-level sheet events	131
Worksheet events	132
Chart events	134
Embedded charts	134
Embedded chart and chart sheet events	135
Application-level events	136
Next steps	140
Chapter 8 Arrays	141
Declaring an array	141
Declaring a multidimensional array	142
Filling an array	144
Using the Array function	144
Filling from a sheet or listobject	145
Using the Split function	146
Working with data in an array	146
Updating existing array fields	147
Using a second array to hold results	147
Extracting a single element from a 2D array	149
Using dynamic arrays	150
Transposing an array	151
Passing an array	151
Next steps	152

Chapter 9	Creating custom objects and collections	153
	Inserting a class module	153
	Creating a custom object	154
	Adding properties	156
	Adding methods	160
	Using a custom object	161
	Using collections	163
	Declare and initialize a collection	163
	Add and retrieve an item to/from a collection	163
	Checking for an item in a collection	166
	Using dictionaries	169
	Using user-defined types to create custom properties	172
	Next steps	174
 Chapter 10	 Userforms: An introduction	 175
	Input boxes	175
	Message boxes	176
	Creating a userform	177
	Calling and hiding a userform	178
	Programming userforms	179
	Userform events	179
	Programming controls	180
	Using basic form controls	181
	Using labels, text boxes, and command buttons	182
	Deciding whether to use list boxes or combo boxes in forms	184
	Using the MultiSelect property of a list box	185
	Adding option buttons to a userform	187
	Adding graphics to a userform	188
	Using a spin button on a userform	189
	Using the MultiPage control to combine forms	191
	Verifying field entry	193
	Illegal window closing	193
	Getting a file name	194
	Next steps	196

Chapter 11 Data mining with Advanced Filter	197
Replacing a loop with AutoFilter.	197
Using AutoFilter techniques	200
Selecting visible cells only	204
Advanced Filter—easier in VBA than in Excel.	204
Using the Excel interface to build an advanced filter	205
Using Advanced Filter to extract a unique list of values	205
Extracting a unique list of values with the user interface	206
Extracting a unique list of values with VBA code	207
Getting unique combinations of two or more fields.	212
Using Advanced Filter with criteria ranges	214
Joining multiple criteria with a logical OR	215
Joining two criteria with a logical AND	216
Other slightly complex criteria ranges	216
The most complex criteria: Replacing the list of values with a condition created as the result of a formula	216
Setting up a condition using computed criteria	218
Using Filter In Place in Advanced Filter.	224
Catching no records when using a filter in place	224
Showing all records after running a filter in place.	225
The real workhorse: <code>xlFilterCopy</code> with all records rather than unique records only	225
Copying all columns	226
Copying a subset of columns and reordering	226
Excel in practice: Turning off a few dropdown menus in the AutoFilter	233
Sorting data.	234
Using <code>Range.Sort</code>	234
Using <code>Worksheet.Sort</code>	235
Using <code>ListObject.Sort</code>	236
Next steps	237
 Chapter 12 Using VBA to create pivot tables	 239
Building a pivot table in Excel VBA	240
ManualUpdate versus RefreshTable versus PivotCache.Refresh	240
Defining the pivot cache	241

Creating and configuring the pivot table	243
Adding multiple pivot tables with the same data source	248
Learning why you cannot move or change part of a pivot report	248
Determining the size of a finished pivot table to convert the pivot table to values	248
Using advanced pivot table features.....	251
Adding fields to the filters area, aka PageFields.....	251
Counting the number of records	252
Grouping daily dates to months, quarters, or years	252
Changing the calculation to show percentages	254
Eliminating blank cells in the Values area	256
Controlling the sort order with AutoSort.....	257
Replicating the report for every product.....	257
Filtering a data set.....	261
Manually filtering two or more items in a pivot field	261
Using the conceptual filters	263
Using the search filter.....	266
Setting up slicers to filter a pivot table	269
Setting up a timeline to filter an Excel pivot table	276
Formatting the intersection of values in a pivot table.....	278
Using the Data Model in Excel.....	278
Adding both tables to the Data Model.....	279
Creating a relationship between the two tables	280
Defining the pivot cache and building the pivot table	281
Adding model fields to the pivot table	281
Adding numeric fields to the Values area	281
Putting it all together	283
Using other pivot table features.....	284
Calculated data fields.....	284
Calculated items	285
Using ShowDetail to filter a record set	286
Changing the layout from the Design tab	286
Settings for the report layout.....	287
Office Scripts for pivot tables.....	288
Next steps	291

Chapter 13 Excel power	293
File operations	293
Listing files in a directory	293
Importing and deleting a CSV file	296
Reading a text file into memory and parsing	296
Checking cloud-based file paths	297
Combining and separating workbooks	298
Separating worksheets into workbooks	298
Combining workbooks	299
Copying data to separate worksheets without using Filter	300
Exporting data to an XML file	301
Placing a chart in a cell note	302
Tracking user changes	304
Using a settings table	304
Techniques for VBA pros	306
Creating an Excel state class module	306
Filtering an OLAP pivot table by a list of items	308
Creating a custom sort order	310
Creating a cell progress indicator	310
Using a protected password box	312
Selecting with SpecialCells	314
Resetting a table's format	314
Using VBA Extensibility to add code to new workbooks	315
Converting a fixed-width report to a data set	317
Leveling up: Real project issues, real solutions	320
Filtering out large numbers of rows efficiently	320
Importing daily report data into an existing table	320
Transposing array data before writing to the worksheet	321
Tracking dynamic columns	321
Converting a data set with multi-row records into a clean data set	322
Using templates to generate reports	323
Copying existing charts to a new workbook	323
Reducing API access token requests	326
Generating reports in Word	327

Supporting users with different feature needs.	328
Next steps	329
Chapter 14 Sample user-defined functions	331
Creating user-defined functions	331
Building a simple custom function	332
Sharing UDFs	334
Useful custom Excel functions	334
Checking whether a workbook is open	334
Checking whether a sheet in an open workbook exists	335
Counting the number of workbooks in a directory	336
Retrieving the user ID	337
Retrieving the date and time of the last save	338
Retrieving permanent date and time	339
Validating an email address	339
Summing cells based on interior color	341
Finding the first nonzero-length cell in a range	342
Substituting multiple characters	343
Sorting and concatenating	344
Sorting numeric and alpha characters	346
Searching for a string within text	348
Returning the addresses of duplicate maximum values	348
Returning a hyperlink address	349
Creating LAMBDA functions	350
Building a simple LAMBDA function	350
Sharing LAMBDA functions	352
Useful LAMBDA functions	352
Next steps	356
Chapter 15 Creating charts	357
Using .AddChart2 to create a chart	358
Understanding chart styles	360
Formatting a chart	363
Referring to a specific chart	363

Specifying a chart title	364
Applying a chart color	365
Filtering a chart.....	366
Using SetElement to emulate changes from the plus icon	367
Using the Format tab to micromanage formatting options.....	372
Changing an object's fill	373
Formatting line settings	376
Creating a combo chart	376
Creating map charts	380
Creating waterfall charts.....	381
Exporting a chart as a graphic.....	382
Next steps	382
Chapter 16 Data visualizations and conditional formatting	383
VBA methods and properties for data visualizations	385
Adding data bars to a range.....	386
Adding color scales to a range	390
Adding icon sets to a range	391
Specifying an icon set	392
Specifying ranges for each icon	393
Using visualization tricks.....	394
Creating an icon set for a subset of a range.....	394
Using two colors of data bars in a range	396
Using other conditional formatting methods.....	398
Formatting cells that are above or below average	398
Formatting cells in the top 10 or bottom 5	399
Formatting unique or duplicate cells.....	400
Formatting cells based on their value	401
Formatting cells that contain text	402
Formatting cells that contain dates	402
Formatting cells that contain blanks or errors	402
Using a formula to determine which cells to format	403
Using the NumberFormat property	404
Next steps	405

Chapter 17 Dashboarding with sparklines in Excel	407
Creating sparklines	408
Scaling sparklines	410
Formatting sparklines.....	413
Using theme colors	413
Using RGB colors	417
Formatting sparkline elements	418
Formatting win/loss charts.....	421
Creating a dashboard	422
Observations about sparklines	423
Creating hundreds of individual sparklines in a dashboard	424
Next steps	428
 Chapter 18 Reading from the web using M and VBA	 429
Get credentials for accessing an API	430
Build a query in Power Query using the M language to retrieve data from the web for a specific value	432
Refreshing the credentials after they expire	436
Building a custom function in Power Query	436
Using the new function in your code.....	439
Duplicating an existing query to make a new query	440
Querying the list of songs on an album	441
Generalizing the queries using VBA	442
Simplifying the SearchArtist query to a single line of code	442
Simplifying the ArtistAlbums query.....	443
Simplifying the AlbumTracks query	443
Grouping queries to clean up the queries list.....	444
Planning the arrangement of query results on your dashboard	445
Using global variables and loops in M	449
Storing global variables in a Settings record in Power Query	449
Simple error handling using try with otherwise.....	450
Using If logic in M	450
Looping using List.Generate.....	451
Application.OnTime to periodically analyze data	453
Using Ready mode for scheduled procedures	454

Specifying a window of time for an update.....	455
Canceling a previously scheduled macro.....	455
Closing Excel cancels all pending scheduled macros.....	456
Scheduling a macro to run x minutes in the future.....	456
Scheduling a verbal reminder.....	456
Scheduling a macro to run every two minutes.....	457
Next steps	458
Chapter 19 Text file processing	459
Importing from text files.....	459
Importing text files with fewer than 1,048,576 rows.....	459
Dealing with text files with more than 1,048,576 rows.....	466
Writing text files.....	471
Next steps	471
Chapter 20 Automating Word	473
Using early binding to reference a Word object.....	473
Using late binding to reference a Word object.....	476
Using the New keyword to reference the Word application.....	477
Using the CreateObject function to create a new instance of an object.....	478
Using the GetObject function to reference an existing instance of Word	478
Using constant values.....	479
Using the Watches window to retrieve the real value of a constant	480
Using the Object Browser to retrieve the real value of a constant	480
Using early and late binding at the same time.....	481
Coding for flexible binding.....	481
Handling constants with flexible binding.....	482
Understanding Word's objects	484
The Document object	484
Controlling form fields in Word.....	492
Next steps	495

Chapter 21 Using Access as a back end to enhance multiuser access to data	497
Understanding Access database formats and compatibility	497
ADO versus DAO	498
The tools of ADO	501
Connecting to a database	503
Adding a record to a database	503
Retrieving records from a database	504
Updating an existing record	506
Deleting records via ADO	509
Summarizing records via ADO	510
Other utilities via ADO	512
Checking for the existence of tables	512
Checking for the existence of a field	513
Adding a table on the fly	514
Adding a field on the fly	515
Connecting to SQL Server from Excel	515
Next steps	517
 Chapter 22 Advanced userform techniques	 519
Using the UserForm toolbar in the design of controls on userforms.	519
More userform controls	520
CheckBox controls	520
TabStrip controls	522
RefEdit controls	523
ToggleButton controls	525
Using a scrollbar as a slider to select values	525
Controls and collections	527
Modeless userforms	529
Using hyperlinks in userforms	529
Adding controls at runtime	530
Resizing the userform on the fly	532
Adding a control on the fly	532
Sizing on the fly	533

Adding other controls	533
Adding an image on the fly	534
Putting it all together	534
Adding help to a userform	536
Showing accelerator keys	536
Adding control tip text	537
Creating the tab order	537
Coloring the active control	538
Creating transparent forms	541
Next steps	542

Chapter 23 The Windows Application Programming Interface (API) 543

Understanding an API declaration	544
Using an API declaration	545
Making 32-bit- and 64-bit-compatible API declarations	545
API function examples	546
Retrieving the computer name	547
Checking whether an Excel file is open on a network	547
Retrieving display-resolution information	548
Customizing the About dialog box	549
Disabling the X for closing a userform	550
Creating a running timer	551
Playing sounds	551
Next steps	552

Chapter 24 Handling errors 553

What happens when an error occurs?	553
A misleading Debug error in userform code	555
Basic error handling with the On Error GoTo syntax	557
Generic error handlers	558
Handling errors by choosing to ignore them	560
Suppressing Excel warnings	562
Encountering errors on purpose	562

Helping Others Help You (When You're Not There)	564
Errors that won't show up in Debug mode	564
Errors while developing versus errors months later	565
Run-Time Error 9: Subscript out of range	565
Run-Time Error 1004: Method range of object global failed	567
The ills of protecting code	568
More problems with passwords	569
Errors caused by different versions	570
Next steps	571

Chapter 25 Customizing the ribbon to run macros 573

Where to add code: The customui folder and file	574
Creating a tab and a group.	575
Adding a control to a ribbon	576
Accessing the file structure.	581
Understanding the RELS file	581
Renaming an Excel file and opening a workbook	582
Using images on buttons	583
Using Microsoft Office icons on a ribbon.	583
Adding custom icon images to a ribbon.	584
Troubleshooting error messages	586
The attribute " <i>Attribute Name</i> " on the element " <i>customui ribbon</i> " is not defined in the DTD/schema.	586
Illegal qualified name character	586
Element " <i>customui Tag Name</i> " is unexpected according to content model of parent element " <i>customui Tag Name</i> ".	587
Found a problem with some content	588
Wrong number of arguments or invalid property assignment.	589
Invalid file format or file extension.	589
Nothing happens.	590
Other ways to run a macro	590
Using a keyboard shortcut to run a macro.	590
Adding a macro button on a built-in ribbon.	590
Creating a macro button on the Quick Access Toolbar	591

Attaching a macro to a command button	592
Attaching a macro to a shape	594
Running a macro from a hyperlink	595
Next steps	596
Chapter 26 Creating Excel add-ins	597
Characteristics of standard add-ins	597
Converting an Excel workbook to an add-in	598
Using Save As to convert a file to an add-in	599
Using the VB Editor to convert a file to an add-in	600
Having a client install an add-in	601
Add-in security	602
Closing add-ins	603
Removing add-ins	603
Using a hidden workbook as an alternative to an add-in	605
Next steps	606
Chapter 27 An introduction to creating Office Add-ins	607
Creating your first Office Add-in—Hello World	608
Adding interactivity to an Office Add-in	612
A basic introduction to HTML	615
Using tags	615
Adding buttons	616
Using CSS files	616
Using XML to define an Office Add-in	617
Using JavaScript to add interactivity to an Office Add-in	617
The structure of a function	618
Curly braces and spaces	618
Semicolons and line breaks	619
Comments	619
Variables	619
Strings	621
Arrays	621
JavaScript for loops	622

How to do an if statement in JavaScript	623
How to do a Select . . Case statement in JavaScript	623
How to use a For each . . next statement in JavaScript	625
Mathematical, logical, and assignment operators.	625
Math functions in JavaScript.	627
Writing to the content pane or task pane	628
JavaScript changes for working in an Office Add-in.	629
Introduction to Office Scripts in Excel Online.	630
The Automate ribbon tab	631
Recording a macro	632
Editing and managing scripts.	633
Running and debugging scripts	634
Next steps	635
<i>Index</i>	637

Acknowledgments

They say it takes an army to get a celebrity ready for the red carpet—stylists, coaches, and a whole lot of behind-the-scenes magic. Writing this book wasn't so different. I had my own team making sure the seams didn't show.

To Loretta: The calm in the storm, the keeper of continuity, and the person who managed to steer this whole thing without ever once raising her voice—at least not where I could hear it.

To Bill Jelen: Thank you for opening the door, showing me the path, and walking it with me. You've made this journey possible.

To Robert L., for keeping me fueled with cookies; to Chris H., for giving me the stage to share my VBA bag of tricks; to Geoff W., for tossing me puzzles that demanded real out-of-the-box thinking; and to Ash—technically, a client but really a collaborator—for helping me think in versions, not just answers. Your clarity cut through my fog every time. Consulting has introduced me to people across every kind of industry, and it's the variety—the weird bugs, the clever fixes, the unexpected questions—that make this work so much fun.

To Icarus, MeltedCake, and Chilly: You reminded me that “It wasn't a phase, Mom” can be a way of life. Thanks for the chaos, the loyalty, and the perfectly timed distractions.

To Jareth, Luna, Sam, and Anita: Your daily antics are worth the 2 a.m. feedings and zoomies.

And to John: Thank you for being my steady in the storm, from the moment I left the corporate grind and started building this strange, wonderful path of my own.

—Tracy

At Pearson, Loretta Yates is an excellent executive editor. Thanks to Rick Kughen for guiding this book through production.

Along the way, I've learned a lot about VBA programming from the awesome community at the MrExcel.com message board. VoG, Richard Schollar, and Jon von der Heyden all stand out as having contributed posts that led to ideas in this book. Thanks to Pam Gensel for Excel macro lesson #1. Mala Singh taught me about creating charts in VBA. Suat Özgür keeps me current on new VBA trends and contributed many ideas to Chapter 18.

My family was incredibly supportive during this time. Thanks to Mary Ellen Jelen.

—Bill

This page intentionally left blank

About the Authors

Tracy Syrstad is the author of 11 Excel books and a veteran Excel developer with decades of experience building custom Excel solutions for businesses and individuals. Since 1997, she's helped clients streamline workflows, automate reporting, and tame messy data using the full power of Excel and VBA. Her writing reflects the same approach she brings to her consulting—clear, practical, and focused on helping real people solve real problems.



Bill Jelen, Excel MVP and the host of MrExcel.com, has been using spreadsheets since 1985, and he launched the MrExcel.com website in 1998. Bill was a regular guest on *Call for Help* with Leo Laporte and has produced more than 2,500 episodes of his video podcast, *Learn Excel from MrExcel*. He is the author of 71 books about Microsoft Excel and writes the monthly Excel column for *Strategic*

Finance magazine. Before founding MrExcel.com, Bill spent 12 years in the trenches—working as a financial analyst for finance, marketing, accounting, and operations departments of a \$500 million public company. He lives in Merritt Island, Florida, with his wife, Mary Ellen.

This page intentionally left blank

Introduction

In this Introduction, you will:

- Find out what is in this book.
- Have a peek at the future of VBA and Windows versions of Excel.
- Learn about special elements and typographical conventions in this book.
- Learn where to find the code files for this book.

As corporate IT departments have found themselves with long backlogs of requests, Excel users have discovered that they can produce the reports needed to run their businesses themselves using the macro language *Visual Basic for Applications* (VBA). VBA enables you to achieve tremendous efficiencies in your day-to-day use of Excel. VBA helps you figure out how to import data and produce reports in Excel so that you don't have to wait for the IT department to help you.

Is TypeScript a threat to VBA?

Your first questions are likely: "Should I invest time in learning VBA? How long will Microsoft support VBA? Will the TypeScript language released for Excel Online replace VBA?"

Your investments in VBA will serve you well until at least 2049. The last macro language change—from XLM to VBA—happened in 1993. XLM is still supported in Excel to this day. That was a case where VBA was better than XLM, but XLM is still supported 28 years later. Microsoft introduced TypeScript for Excel Online in February 2020. I expect that they will continue to support VBA in the Windows and Mac versions of Excel for the next 28 years.

In the Excel universe today, there are versions of Excel running in Windows, in macOS, on mobile phones powered by Android and iOS, and in modern browsers using Excel Online. In my world, I use Excel 99% of the time on a Windows computer. There's perhaps 1% of the time I use Excel Online. But if you are in a mobile environment where you are using Excel in a browser, then the TypeScript UDFs might be appropriate for you.

For an introduction to TypeScript UDFs in Excel, read Suat M. Özgür's *Excel Custom Functions Straight to the Point* (ISBN 978-1-61547-259-8).

However, TypeScript performance is still horrible. If you don't need your macros to run in Excel Online, the VBA version of your macro will run eight times more quickly than

the TypeScript version. For people who plan to run Excel only on the Mac or Windows platforms, VBA will be your go-to macro language for another decade.

The threat to Excel VBA is the new Excel Power Query tools found in the Get & Transform group of the Data tab in Excel for Windows. If you are writing macros to clean imported data, you should consider cleaning the data once with Power Query and then refreshing the query each day. Bill has a lot of Power Query workflows set up that would have previously required VBA. For a primer on Power Query, check out *Master Your Data with Excel and Power BI: Leveraging Power Query to Get & Transform Your Task Flow* by Ken Puls and Miguel Escobar (ISBN 978-1-61547-058-7).

Is Python a threat to VBA?

Python in Excel is not made to automate chores like VBA. It doesn't control the workbook, respond to events, or handle formatting and file tasks. Instead, it's a powerful alternative to analyzing worksheet data—ideal for advanced calculations, reshaping data with pandas, or creating custom visuals. Think of it as a next-gen formula engine, not a macro tool.

Python runs in Excel as a special formula type using the `=PY()` function. You write Python code directly in a cell, and Excel sends that code to a sandbox-secure cloud environment to execute. The result is returned to the worksheet.

The function supports libraries like pandas, matplotlib, NumPy, and seaborn. This makes it a great option for complex analysis, but Python can't modify the workbook or sheet structure or trigger actions. For now, that means VBA is still your go-to for automating tasks, controlling Excel behavior, and building interactive tools.

What is in this book?

You have taken the right step by purchasing this book. We can help you reduce the learning curve so that you can write your own VBA macros and put an end to the burden of generating reports manually.

Reducing the learning curve

This Introduction provides a case study about the power of macros. Chapter 1, "Unleashing the power of Excel with VBA," introduces the tools and confirms what you probably already know: The macro recorder does not work reliably. Chapter 2, "This sounds like BASIC, so why doesn't it look familiar?" helps you understand the crazy syntax of VBA. Chapter 3, "Referring to ranges, names, and tables," cracks the code on how to work efficiently with ranges, cells, defined names, and tables.

Chapter 4, “Laying the groundwork with variables and structures,” lays the groundwork for writing flexible, maintainable VBA code by introducing key concepts like procedures, variables, object references, and code structure tools such as `With` blocks and compiler directives. These building blocks will help you understand—and eventually master—the code you’ll encounter throughout the book.

Chapter 5, “Looping and flow control,” covers the power of looping using VBA. The case study in this chapter demonstrates creating a program to produce a department report and then wrapping that report routine in a loop to produce 46 reports.

Chapter 6, “R1C1 style formulas,” covers, obviously, R1C1 style formulas. Chapter 7, “Event programming,” shows how user interaction and automatic Excel functionality, such as calculations, can be used to trigger procedures. Chapter 8, “Arrays,” covers arrays. Chapter 9, “Creating custom objects and collections,” covers creating custom objects and collections, great tools for organizing data. Chapter 10, “Userforms: An introduction,” introduces custom dialog boxes that you can use to collect information from a human using Excel.

Excel VBA power

Chapters 11, “Data mining with Advanced Filter,” and 12, “Using VBA to create pivot tables,” provide an in-depth look at Filter, Advanced Filter, and pivot tables. Report automation tools rely heavily on these concepts. Chapters 13, “Excel power,” and 14, “Sample user-defined functions,” include dozens of code samples designed to exhibit the power of Excel VBA and custom functions. Chapter 13 includes the section, “Leveling up: Real project issues, real solutions,” which features various project issues I run into and how I resolve them.

Chapters 15, “Creating charts,” through 20, “Automating Word,” handle charting, data visualizations, web queries, sparklines, and automating Word.

Techie stuff needed to produce applications

Chapter 21, “Using Access as a back end to enhance multiuser access to data,” handles reading and writing to Access databases and SQL Server. The techniques for using Access databases enable you to build an application with the multiuser features of Access while keeping the friendly front end of Excel.

Chapter 22, “Advanced userform techniques,” shows you how to go further with userforms. Chapter 23, “The Windows Application Programming Interface (API),” teaches some tricky ways to achieve tasks using the Windows API. Chapters 24, “Handling errors,” through 26, “Creating Excel add-ins,” deal with error handling, custom menus, and add-ins. Chapter 27, “An introduction to creating Office Add-ins,” provides a brief introduction to building your own Office Script application within Excel.

Does this book teach Excel?

Not exactly. We assume you already use Excel and are ready to go further with VBA. While we don't walk through every feature, we do point out powerful tools, such as pivot tables and Power Query, when they matter to the code. If you haven't used them before, you'll know what to explore.

Bill regularly presents a Power Excel seminar for accountants—people who use Excel 30 to 40 hours a week. Yet two things happen at every seminar. First, half the room gasps when they see how fast a feature like automatic subtotals or pivot tables can be. Second, someone always trumps Bill. He'll answer a question, and someone in the second row chimes in with a better solution.

The point? Both the authors and readers of this book know Excel—but we still assume that in any given chapter, many readers haven't used pivot tables, and even fewer have tried features like the Top 10 Filter. So, before we automate something in VBA, we'll briefly show how to do it manually in Excel. This isn't a how-to on pivot tables, but it will let you know when it's time to look one up.

Case study: Monthly accounting reports

This is a true story. Valerie is a business analyst in the accounting department of a medium-sized corporation. Her company recently installed an overbudget \$16 million enterprise resource planning (ERP) system. As the project ground to a close, there were no resources left in the IT budget to produce the monthly report that this corporation used to summarize each department.

However, Valerie had been close enough to the implementation to think of a way to produce the report herself. She understood that she could export general ledger data from the ERP system to a text file with comma-separated values. Using Excel, Valerie was able to import the general ledger data from the ERP system into Excel.

Creating the report was not easy. As in many other companies, there were exceptions in the data. Valerie knew that certain accounts in one particular cost center needed to be reclassified as expenses. She knew that other accounts needed to be excluded from the report entirely. Working carefully in Excel, Valerie made these adjustments. She created one pivot table to produce the first summary section of the report. She cut the pivot table results and pasted them into a blank worksheet. Then she created a new pivot table report for the second section of the summary. After about three hours, she had imported the data, produced five pivot tables, arranged them in a summary, and neatly formatted the report in color.

Becoming the hero

Valerie handed the report to her manager. The manager had just heard from the IT department that it would be months before they could get around to producing “that convoluted report.” When Valerie created the Excel report, she became the instant hero of the day. In three hours, Valerie had managed to do the impossible. Valerie was on cloud nine after some well-deserved recognition.

More cheers

The next day, Valerie’s manager attended the monthly department meeting. When the department managers started complaining that they could not get the report from the ERP system, this manager pulled out his department’s report and placed it on the table. The other managers were amazed. How was he able to produce this report? Everyone was relieved to hear that someone had cracked the code. The company president asked Valerie’s manager if he could have the report produced for each department.

Cheers turn to dread

You can probably see what’s coming. This particular company had 46 departments. That means 46 one-page summaries had to be produced once a month. Each report required importing data from the ERP system, backing out certain accounts, producing five pivot tables, and then formatting the reports in color. It had taken Valerie three hours to produce the first report, but after she got into the swing of things, she could produce the 46 reports in 40 hours. Even after she reduced her time per report, though, this is horrible. Valerie had a job to do before she became responsible for spending 40 hours a month producing these reports in Excel.

VBA to the rescue

Valerie found Bill’s company, MrExcel Consulting, and explained her situation. In the course of about a week, Bill was able to produce a series of macros in Visual Basic that did all the mundane tasks. For example, the macros imported the data, backed out certain accounts, made five pivot tables, and applied the color formatting. From start to finish, the entire 40-hour manual process was reduced to two button clicks and about 4 minutes.

Right now, either you or someone in your company is probably stuck doing manual tasks in Excel that can be automated with VBA. We are confident that we can walk into any company that has 20 or more Excel users and find a case just as amazing as Valerie’s.

Versions of Excel

This eighth edition of *VBA and Macros* is designed to work with Microsoft 365 features released up through July 2025. The previous editions of this book covered code for Excel 97 through Excel 365 (2021). In 80 percent of the chapters, the code today is identical to the code in previous versions.

Differences for Mac users

Although Excel for Windows and Excel for the Mac are similar in terms of user interface, there are a number of differences when you compare the VBA environment. Certainly, nothing in Chapter 23 that uses the Windows API will work on the Mac. That said, the overall concepts discussed in this book apply to the Mac. You can find a general list of differences as they apply to the Mac at <http://www.mrexcel.com/macvba.html>. The VBA Editor for the Mac does not let you design UserForms (Chapter 10). It also has a bug that makes it difficult to create event handler macros (Chapter 7). Excel throws an error when you try to select from the drop-downs at the top of the Code window. You have to first copy and paste an empty event procedure; then the drop-downs will work.

Special elements and typographical conventions

The following typographical conventions are used in this book:

- *Italic*—Indicates new terms when they are defined, special emphasis, non-English words or phrases, and letters or words used as words.
- Monospace—Indicates parts of VBA code, such as object or method names.
- **Bold**—Indicates user input.

In addition to these typographical conventions, there are several special elements. Many chapters have at least one case study that presents a real-world solution to common problems. The case study also demonstrates practical applications of the topics discussed in the chapter.

In addition to the case studies, you will see Level Up, Note, Tip, and Caution sidebars.



Level Up!

Level Up sidebars offer deeper insights or clever techniques that go beyond the basics. They're optional side explorations—perfect if you're ready to stretch your skills a bit further than what the current section assumes.



Note Notes provide additional information outside the main thread of the chapter discussion that might be useful for you to know.



Tip Tips provide quick workarounds and time-saving techniques to help you work more efficiently.



Caution Cautions warn about potential pitfalls you might encounter. Pay attention to the Cautions; they alert you to problems that might otherwise cause you hours of frustration.

About the companion content

As a thank-you for buying this book, we have put together a set of 50 Excel workbooks that demonstrate the concepts included in this book. This set of files includes all the code from the book, sample data, and additional notes from the authors.

To download the code files, visit this book's webpage at MicrosoftPressStore.com/XLVBAuto/downloads.

Errata, updates, and book support

We've made every effort to ensure the accuracy of this book and its companion content. Any errors that have been reported since this book was published are listed at

MicrosoftPressStore.com/XLVBAuto/errata.

If you find an error that is not already listed, you can report it to us through the same page.

For additional book support and information, please visit MicrosoftPressStore.com/Support.

Please note that product support for Microsoft software and hardware is not offered through the previous addresses. For help with Microsoft software or hardware, go to support.microsoft.com.

Stay in touch

Let's keep the conversation going! We're on X:

x.com/MicrosoftPress

x.com/MrExcel

bsky.app/profile/tsyrstad.bsky.social

This sounds like BASIC, so why doesn't it look familiar?

In this chapter, you will:

- Find out how VBA is different from BASIC
- Understand the parts of VBA “speech”
- Find out that learning VBA is not really hard
- Examine recorded macro code using the VB Editor and Help
- Use debugging tools to figure out recorded code
- Get to know the Object Browser
- Learn seven tips for cleaning up recorded code

As mentioned in Chapter 1, “Unleashing the power of Excel with VBA,” if you have taken a class in a procedural language such as BASIC or COBOL, you might be confused when you look at VBA code. Even though VBA stands for *Visual Basic for Applications*, it is an *object-oriented* version of BASIC. Here is a bit of recorded VBA code:

```
Selection.End(xlDown).Select
Range("A11").Select
ActiveCell.FormulaR1C1 = "Total"
Range("E11").Select
Selection.FormulaR1C1 = _
    "=SUM(R[-9]C:R[-1]C)"
Selection.AutoFill _
    Destination:=Range("E11:G11"), _
    Type:=xlFillDefault
```

This code likely makes no sense to anyone who knows only procedural languages. Unfortunately, your first introduction to programming in school (assuming that you are more than 40 years old) would have been a procedural language.

Here is a section of code written in the BASIC language:

```
For x = 0 to 9
  Print Rpt$(" ",x)
  Print "*"
Next x
```

If you run this code, you get a pyramid of asterisks on your screen:

```
*
 *
  *
   *
    *
   *
  *
 *
*
 *
  *
```

If you have ever been in a procedural programming class, you can probably look at the code and figure out what is going on because procedural languages are more English-like than object-oriented languages. The statement `Print "Hello World"` follows the verb–object format, which is how you would generally talk. Let’s step away from programming for a second and look at a concrete example.

Understanding the parts of VBA “speech”

If you were going to write code for instructions to play soccer using BASIC, the instruction to kick a ball would look something like this:

```
"Kick the Ball"
```

Hey, this is how you talk! It makes sense. You have a verb (*kick*) and then a noun (*ball*). The BASIC code in the preceding section has a verb (`Print`) and a noun (the asterisk, `*`). Life is good.

Here is the problem: VBA doesn’t work like this. In fact, no object-oriented language works like this. In an object-oriented language, the objects (nouns) are most important, hence the name, “object-oriented.” If you were going to write code for instructions to play soccer with VBA, the basic structure would be as follows:

```
Ball.Kick
```

You have a noun (`Ball`), which comes first. In VBA, this is an *object*. Then, you have the verb (`Kick`), which comes next. In VBA, this is a *method*.

The basic structure of VBA is a bunch of lines of code with this syntax:

```
Object.Method
```

Needless to say, this is not English. If you took a romance language in high school, you will remember that those languages use a “noun–adjective” construct. However, no one uses “noun–verb” to tell someone to do something:

Water.Drink
Food.Eat
Gir1.Kiss

That is why VBA is confusing to someone who previously took a procedural programming class.

Let’s carry the analogy a bit further. Imagine that you walk onto a grassy field, and there are five balls in front of you: a soccer ball, basketball, baseball, bowling ball, and tennis ball. You want to instruct a kid on your soccer team to “kick the soccer ball.”

If you tell them to kick the ball (or Ba11.Kick), you really aren’t sure which one of the five balls they will kick. Maybe they will kick the one closest to them, which could be a problem if they are standing in front of the bowling ball.

For almost any noun or object in VBA, there is a collection of that object. Think about Excel. If you can have one row, you can have a bunch of rows. If you can have one cell, you can have a bunch of cells. If you can have one worksheet, you can have a bunch of worksheets. The only difference between an object and a collection is that you add an *s* to the name of the object:

Row becomes Rows.

Cell becomes Cells.

Ba11 becomes Ba11s.

When you refer to something that is a collection, you have to tell the programming language to which item you are referring. There are a couple of ways to do this. You can refer to an item by using a number. For example, if the soccer ball is the second ball, you might say this:

Ba11s(2).Kick

This works fine, but it could be a dangerous way to program. For example, it might work on Tuesday. However, if you get to the field on Wednesday and someone has rearranged the balls, Ba11s(2).Kick might be a painful exercise.

A much safer way to go is to use a name for the object in a collection. You can say the following:

Ba11s("Soccer").Kick

With this method, you always know that it will be the soccer ball that is being kicked.

So far, so good. You know that a ball will be kicked, and you know that it will be a soccer ball. For most of the verbs or methods in Excel VBA, there are *parameters* that tell *how* to do the action. These parameters act as adverbs. You might want the soccer ball to be kicked to the left and with a hard force.

In this case, the method would have a number of parameters that tell how the program should perform the method:

```
Balls("Soccer").Kick Direction:=Left, Force:=Hard
```

When you are looking at VBA code, the colon–equal sign combination (:=) indicates that you are looking at the parameters of how the verb should be performed.

Sometimes, a method will have a list of 10 parameters, some of which are optional. For example, if the Kick method has an Elevation parameter, you would have this line of code:

```
Balls("Soccer").Kick Direction:=Left, Force:=Hard, Elevation:=High
```

Here is the confusing part: Every method has a default order for its parameters. If you are not a conscientious programmer and happen to know the order of the parameters, you can leave off the parameter names. The following code is equivalent to the previous line of code:

```
Balls("Soccer").Kick Left, Hard, High
```

This throws a monkey wrench into our understanding. Without :=, it is not obvious that you have parameters. Unless you know the parameter order, you might not understand what is being said. It is pretty easy with Left, Hard, and High, but when you have parameters like the following:

```
ActiveSheet.Shapes.AddShape Type:=1, Left:=10, Top:=20, Width:=100, Height:=200
```

It gets confusing if you instead have this:

```
ActiveSheet.Shapes.AddShape 1, 10, 20, 100, 200
```

The preceding line is valid code. However, unless you know that the default order of the parameters for this Add method is Type, Left, Top, Width, Height, this code does not make sense. The default order for any particular method is the order of the parameters, as shown in the Help topic for that method.

To make life more confusing, you are allowed to start specifying parameters in their default order without naming them, and then you can switch to naming parameters when you hit one that does not match the default order. If you want to kick the ball to the left and high but do not care about the force (that is, you are willing to accept the default force), the following two statements are equivalent:

```
Balls("Soccer").Kick Direction:=Left, Elevation:=High  
Balls("Soccer").Kick Left, Elevation:=High
```

However, keep in mind that as soon as you start naming parameters, they have to be named for the remainder of that line of code.

Some methods simply act on their own. To simulate pressing the F9 key, you use this code:

```
Application.Calculate
```

Other methods perform an action and create something. For example, you can add a worksheet by using the following:

```
Worksheets.Add Before:=Worksheets(1)
```

However, because `Worksheets.Add` creates a new object, you can assign the results of this method to a variable. In this case, you must surround the parameters with parentheses:

```
Set MyWorksheet = Worksheets.Add(Before:=Worksheets(1))
```



Note Don't worry if the use of `Set` isn't clear yet. This command and other foundational concepts are covered in more detail in Chapter 4, "Laying the groundwork with variables and structures."

One final bit of grammar is necessary: adjectives. Just as adjectives describe a noun, *properties* describe an object. Because you are an Excel fan, let's switch from the soccer analogy to an Excel analogy. There is an object to describe the active cell. Fortunately, it has a very intuitive name:

```
ActiveCell
```

Suppose you want to change the color of the active cell to red. There is a property called `Interior.Color` for a cell that uses a complex series of codes. However, you can turn a cell to red by using this code:

```
ActiveCell.Interior.Color = 255
```

You can see how this can be confusing. Again, there is the *noun-dot-something* construct, but this time, it is `Object.Property` rather than `Object.Method`. How you tell them apart is quite subtle: There is no colon before the equal sign. A property is almost always set equal to something, or perhaps the value of a property is assigned to something else.

To make this cell color the same as cell A1, you might say this:

```
ActiveCell.Interior.Color = Range("A1").Interior.Color
```

`Interior.Color` is a property. Actually, `Interior` is a property of the `Range` object, and `Color` is a property of the `Interior` property. By changing the value of a property, you can make things look different. It is kind of bizarre: Change an adjective, and you are actually doing something to the cell. Humans would say, "Color the cell red," whereas VBA says this:

```
ActiveCell.Interior.Color = 255
```

Table 2-1 summarizes the VBA "parts of speech."

TABLE 2-1 Parts of the VBA programming language

VBA Component	Analogous To	Notes
Object	Noun	Examples include cell or sheet.
Collection	Plural noun	Usually specifies which object: <code>Worksheets(1)</code> .
Method	Verb	Appears as <code>Object.Method</code> .
Parameter	Adverb	Lists parameters after the method. Separate the parameter name from its value with <code>:</code> .
Property	Adjective	You can set a property (for example, <code>ActiveCell.Height=10</code>) or store the value of a property (for example, <code>x = ActiveCell.Height</code>).

VBA is not really hard

Knowing whether you are dealing with properties or methods helps you set up the correct syntax for your code. Don't worry if it all seems confusing right now. When you are writing VBA code from scratch, it is tough to know whether the process of changing a cell to yellow requires a verb or an adjective. Is it a method or a property?

This is where the macro recorder is especially helpful. When you don't know how to code something, you record a short little macro, look at the recorded code, and figure out what is going on.

VBA Help files: Using F1 to find anything

Excel VBA Help is an amazing feature, provided that you are connected to the Internet. If you are going to write VBA macros, you absolutely *must* have access to the VBA Help topics installed. Follow these steps to see how easy it is to get help in VBA:

1. Open Excel and switch to the VB Editor by pressing Alt+F11. From the Insert menu, select Module.
2. Type these three lines of code:

```
Sub Test()  
    MsgBox "Hello World!"  
End Sub
```
3. Click inside the word `MsgBox`.
4. With the cursor in the word `MsgBox`, press F1. If you can reach the Internet, you will see the Help topic for the `MsgBox` function.

Using Help topics


If you request help on a function or method, the Help topic walks you through the various available arguments. If you browse to the bottom of a Help topic, you can see a great resource: code samples under the Example heading (see Figure 2-1).

It is possible to select the code, copy it to the Clipboard by clicking the Copy shortcut in the top-right corner of the code box, and then paste it into a module by pressing Ctrl+V.

After you record a macro, if there are objects or methods about which you are unsure, you can get help by inserting the cursor in any keyword and pressing F1.

Example

This example uses the `MsgBox` function to display a critical-error message in a dialog box with **Yes** and **No** buttons. The **No** button is specified as the default response. The value returned by the `MsgBox` function depends on the button chosen by the user. This example assumes that `DEMO.HLP` is a Help file that contains a topic with a Help context number equal to `1000`.



```
VB Copy

Dim Msg, Style, Title, Help, Ctxt, Response, MyString
Msg = "Do you want to continue ?" ' Define message.
Style = vbYesNo Or vbCritical Or vbDefaultButton2 ' Define buttons.
Title = "MsgBox Demonstration" ' Define title.
Help = "DEMO.HLP" ' Define Help file.
Ctxt = 1000 ' Define topic context.
' Display message.
Response = MsgBox(Msg, Style, Title, Help, Ctxt)
If Response = vbYes Then ' User chose Yes.
    MyString = "Yes" ' Perform some action.
Else ' User chose No.
    MyString = "No" ' Perform some action.
End If
```

FIGURE 2-1 Most Help topics include code samples.

Examining recorded macro code: Using the VB Editor and Help

Let's take a look at the code you recorded in Chapter 1 to see whether it makes more sense now that you know about objects, properties, and methods. You can also see whether it's possible to correct the errors created by the macro recorder.

Figure 2-2 shows the first code that Excel recorded in the example from Chapter 1.

Now that you understand the concept of `Noun.Verb` or `Object.Method`, consider the first line of code that reads `Workbooks.OpenText`. In this case, `Workbooks` is a collection object, and `OpenText` is a method. Click the word `OpenText` and press F1 for an explanation of the `OpenText` method (see Figure 2-3).

The Help file confirms that `OpenText` is a method, or an action word. The default order for all the arguments that can be used with `OpenText` appears in a Parameters table. Notice that only one argument is required: `Filename`. All the other arguments are listed as optional.

```

(Sub) ImportInvoice()
'
' ImportInvoice Macro
' Import Invoice.txt. Add Total Row. Format.
'
' Keyboard Shortcut: Ctrl+I
'
Workbooks.OpenText Filename:="G:\2016VBA\SampleFiles\invoice.txt", Origin:= _
437, StartRow:=1, DataType:=xlDelimited, TextQualifier:=xlDoubleQuote, _
ConsecutiveDelimiter:=False, Tab:=False, Semicolon:=False, Comma:=True, _
, Space:=False, Other:=False, FieldInfo:=Array(Array(1, 3), Array(2, 1), _
Array(3, 1), Array(4, 1), Array(5, 1), Array(6, 1), Array(7, 1)), TrailingMinusNumbers _
:=True
Selection.End(xlDown).Select
Range("A1").Select
ActiveCell.FormulaR1C1 = "Total"
Range("E1").Select
Selection.FormulaR1C1 = "=SUM(R[-9]C:R[-1]C)"
Selection.AutoFill Destination:=Range("E1:G11"), Type:=xlFillDefault
Range("E1:G11").Select
Rows("1:1").Select
Selection.Font.Bold = True
Rows("11:11").Select
Selection.Font.Bold = True
Selection.CurrentRegion.Select
Selection.Columns.AutoFit
End Sub

```

FIGURE 2-2 Here is the recorded code from the example in Chapter 1.

Parameters			
<i>Filename</i>	Required	String	Specifies the file name of the text file to be opened and parsed.
<i>Origin</i>	Optional	Variant	Specifies the origin of the text file. Can be one of the following XIPlatform constants: xlMacintosh , xlWindows , or xlMSDOS . Additionally, this could be an integer representing the code page number of the desired code page. For example, "1256" would specify that the encoding of the source text file is Arabic (Windows). If this argument is omitted, the method uses the current setting of the File Origin option in the Text Import Wizard .
<i>StartRow</i>	Optional	Variant	The row number at which to start parsing text. The default value is 1.
<i>DataType</i>	Optional	Variant	Specifies the column format of the data in the file. Can be one of the following XITextParsingType constants: xlDelimited or xlFixedWidth . If this argument is not specified, Microsoft Excel attempts to determine the column format when it opens the file.
<i>TextQualifier</i>	Optional	XITextQualifier	Specifies the text qualifier.
<i>ConsecutiveDelimiter</i>	Optional	Variant	True to have consecutive delimiters considered one delimiter. The default is False .

FIGURE 2-3 This shows part of the Help topic for the `OpenText` method.

Optional parameters

The Help file can tell you what happens if you skip an optional parameter. For `StartRow`, the Help file indicates that the default value is 1. If you leave out the `StartRow` parameter, Excel starts importing at row 1. This is fairly safe.

Now look at the Help file note about `Origin`. If this argument is omitted, you inherit whatever value was used for `Origin` the last time someone used this feature in Excel on this computer. That is a recipe for disaster. For example, your code might work 98 percent of the time. However, immediately after someone imports an Arabic file, Excel remembers the setting for Arabic and thereafter assumes that this is what your macro wants if you don't explicitly code this parameter.

Defined constants

Look at the Help file entry for `DataType` in Figure 2-3, which says it can be one of these constants: `xlDelimited` or `xlFixedWidth`. The Help file says these are the valid `xlTextParsingType` constants that are predefined in Excel VBA. In the VB Editor, press `Ctrl+G` to bring up the Immediate window. In the Immediate window, type this line and press `Enter`:

```
Print xlFixedWidth
```

The answer appears in the Immediate window. `xlFixedWidth` is the equivalent of saying 2 (see Figure 2-4). In the Immediate window, type **Print xlDelimited**, which is really the same as typing 1. Microsoft correctly assumes that it is easier for someone to read code that uses the somewhat English-like `xlDelimited` term than to read 1.

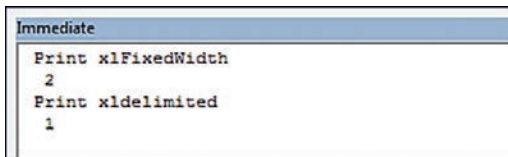


FIGURE 2-4 In the Immediate window of the VB Editor, you can query to see the true value of constants such as `xlFixedWidth`.




If you were an evil programmer, you could certainly memorize all these constants and write code using the numeric equivalents of the constants. However, the programming gods (and the next person who has to look at your code) will curse you for doing so.

In most cases, the Help file either specifically calls out the valid values of the constants or offers a hyperlink that opens the Help topic, showing the complete enumeration and the valid values for the constants (see Figure 2-5).

If you read the Help topic on `OpenText`, you can surmise that it is basically the equivalent of opening a file using the Text Import Wizard. In step 1 of the wizard, you normally choose either `Delimited` or `Fixed Width`. You also specify the file origin and at which row to start. This first step of the wizard is handled by these parameters of the `OpenText` method:

```
Origin:=437  
StartRow:=1  
DataType:=xlDelimited
```

XlColumnDataType enumeration (Excel)

06/08/2017 • 2 minutes to read •   

Specifies how a column is to be parsed.

Name	Value	Description
<code>xlDMYFormat</code>	4	DMY date format.
<code>xlDYMFormat</code>	7	DYM date format.
<code>xlEMDFormat</code>	10	EMD date format.
<code>xlGeneralFormat</code>	1	General.
<code>xlMDYFormat</code>	3	MDY date format.
<code>xlMYDFormat</code>	6	MYD date format.
<code>xlSkipColumn</code>	9	Column is not parsed.
<code>xlTextFormat</code>	2	Text.
<code>xlYDMFormat</code>	8	YDM date format.
<code>xlYMDFormat</code>	5	YMD date format.

FIGURE 2-5 Click the hyperlink to see all the possible constant values. Here, the 10 possible `xlColumnDataType` constants are revealed in a new Help topic.

Step 2 of the Text Import Wizard enables you to specify that your fields be delimited by commas. Because you do not want to treat two commas as a single comma, the Treat Consecutive Delimiters As One checkbox should not be selected. Sometimes, a field may contain a comma, such as "XYZ, Inc." In this case, the field should have quotes around the value, as specified in the Text Qualifier box. This second step of the wizard is handled by the following parameters of the `OpenText` method:

```
TextQualifier:=xlDoubleQuote
ConsecutiveDelimiter:=False
Tab:=False
Semicolon:=False
Comma:=True
Space:=False
Other:=False
```

Step 3 of the wizard is where you actually identify the field types. In this case, you leave all fields as General except for the first field, which is marked as a date in MDY (Month, Day, Year) format. This is represented in code by the `FieldInfo` parameter.

The third step of the Text Import Wizard is fairly complex. The entire `FieldInfo` parameter of the `OpenText` method duplicates the choices made in this step of the wizard. If you happen to click the Advanced button on the third step of the wizard, you have an opportunity to specify something other than the default decimal and thousands separators, as well as the setting Trailing Minus For Negative Numbers.



Note Note that the macro recorder does not write code for `DecimalSeparator` or `ThousandsSeparator` unless you change these from the defaults. The macro recorder does, however, always record the `TrailingMinusNumbers` parameter.

Remember that every action you perform in Excel while recording a macro gets translated to VBA code. In the case of many dialogs, the settings you do not change are often recorded along with the items you do change. When you click OK to close the dialog, the macro recorder often records all the current settings from the dialog in the macro.

Here is another example. The next line of code in the macro is this:

```
Selection.End(xlDown).Select
```

You can click to get help for three topics in this line of code: `Selection`, `End`, and `Select`. Assuming that `Selection` and `Select` are somewhat self-explanatory, click in the word `End` and press F1 for Help.

This Help topic says that `End` is a property. It returns a `Range` object that is equivalent to pressing End+up arrow or End+down arrow in the Excel interface (see Figure 2-6). If you click the blue hyperlink for `xlDirection`, you see the valid parameters that can be passed to the `End` function.

Range.End property (Excel)

Article • 02/07/2022 • 6 contributors [Feedback](#)

In this article

- [Syntax](#)
- [Parameters](#)
- [Example](#)

Returns a **Range** object that represents the cell at the end of the region that contains the source range. Equivalent to pressing END+UP ARROW, END+DOWN ARROW, END+LEFT ARROW, or END+RIGHT ARROW. Read-only **Range** object.

Syntax

expression.**End** (*Direction*)

expression A variable that represents a **Range** object.

Parameters

[Expand table](#)

Name	Required/Optional	Data type	Description
<i>Direction</i>	Required	XlDirection	The direction in which to move.

FIGURE 2-6 The correct Help topic for the `End` property.

Properties can return objects

In VBA, some properties give you simple values—like a number, a string, or True/False. But other properties give you something more powerful—another object. This is called *returning an object*. For example, the `End(xlDown)` property doesn't just give you a value—it gives you a Range object that represents a specific cell. Once you have that object, you can do things with it, like select it, read its contents, or format it.

Consider the line of code currently under examination:

```
Selection.End(xlDown).Select
```

The `End` keyword is a property, but from the Help file, you see that it returns a Range object. You then call `Select` on that returned object. In the end, you're calling a method (`Select`) on the result of a property (`End`). You'll do this often in VBA when a property gives you an object you can immediately act on.

Selection is actually a property of Application

`Selection` might look like an object, but if you bring up the help topic for `Selection`, you'll see that it is actually a property. In reality, the proper code would be `Application.Selection`. However, when you are writing code within Excel, VBA assumes you are referring to the Excel object model, so you can leave off the `Application` prefix. If you were to automate Excel from another program, like Word, you'd need to include an object reference before the `Selection` property, such as `ExcelApp.Selection`.

`Selection` can return different types of objects. It returns the object of whatever is selected, such as cells, charts, etc.

Using debugging tools to figure out recorded code

The following sections introduce some awesome debugging tools that are available in the VB Editor. These tools are excellent for helping you see what a recorded macro code is doing.

Stepping through code

Generally, a macro runs quickly: You start it, and less than a second later, it's done. If something goes wrong, you don't have an opportunity to figure out what the macro is doing. However, using Excel's Step Into feature makes it possible to run one line of code at a time.

To use this feature, make sure your cursor is in the procedure you want to run, such as the `ImportInvoice` procedure, and then from the menu, select `Debug | Step Into`, as shown in Figure 2-7. Alternatively, you can press F8.

The VB Editor is now in Break mode. The line about to be executed is highlighted in yellow, with a yellow arrow in the margin before the code (see Figure 2-8).

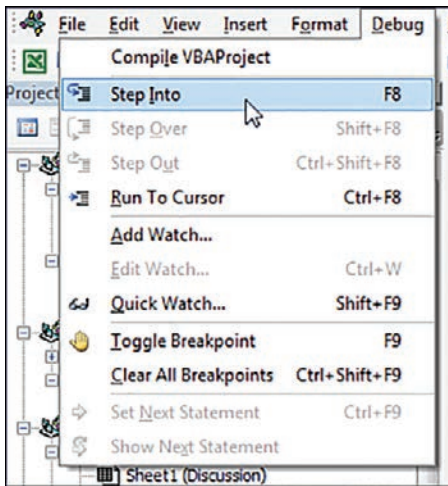


FIGURE 2-7 You can use the Step Into feature to run a single line of code at a time.

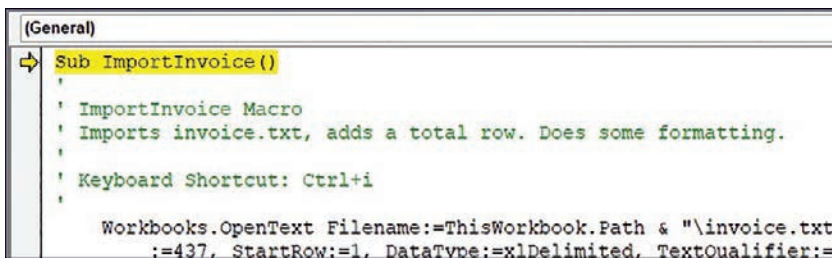


FIGURE 2-8 The first line of the macro is about to run.

In this case, the next line to be executed is the `Sub ImportInvoice()` line. This basically says, “You are about to start running this procedure.” Press the F8 key to execute the line in yellow and move to the next line of code. The long code for `OpenText` is then highlighted. Press F8 to run this line of code. When you see that `Selection.End(xlDown).Select` is highlighted, you know that Visual Basic has finished running the `OpenText` command. At this point, you can press Alt+Tab to switch to Excel and see that the `Invoice.txt` file has been parsed into Excel. Note that A1 is selected.



Tip If you have a wide monitor, you can use the Restore Down icon at the top right of the VBA window to arrange the window so that you can see both the VBA window and the Excel window. (Restore Down is the two-tiled window icon between the Minimize “dash” and the Close Window X icon at the top of every maximized window).

This is also a great trick to use while recording new code. You can actually watch the code appear as you do things in Excel.

Switch back to the VB Editor by pressing Alt+Tab. The next line about to be executed is `Selection.End(xlDown).Select`. Press F8 to run this code. Switch to Excel to see that the last cell in your data set is selected.

Press F8 again to run the `Range("A11").Select` line. If you switch to Excel by pressing Alt+Tab, you see that this is where the macro starts to have problems. Instead of moving to the first blank row, the program moves to the wrong row.

Now that you have identified the problem area, you can stop the code execution by using the Reset command. You can start the Reset command either by selecting Run | Reset or by clicking the Reset button on the toolbar (the small blue square next to the Run and Pause icons). After clicking Reset, you should return to Excel and undo anything done by the partially completed macro. In this case, you need to close the Invoice.txt file without saving.

More debugging options: Breakpoints

If you have hundreds of lines of code, you might not want to step through each line one at a time. If you have a general idea that a problem is happening in one particular section of the program, you can set a breakpoint. You can then have the code start to run, but the macro pauses just before it executes the breakpoint line of code.

To set a breakpoint, click in the gray margin area to the left of the line of code on which you want to break. A large maroon dot appears next to this code, and the line of code is highlighted in brown (see Figure 2-9). (If you don't see the margin area, go to Tools | Options | Editor Format and select the Margin Indicator Bar checkbox.) Or select a line of code and press F9 to toggle a breakpoint on or off.

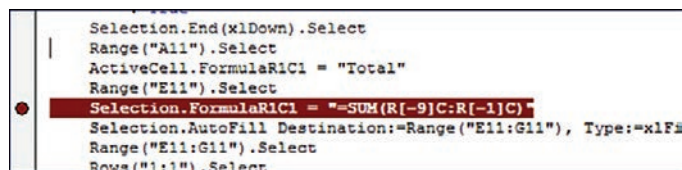


FIGURE 2-9 The large maroon dot signifies a breakpoint.

Next, from the Visual Basic menu, select Run | Run Sub/UserForm or press F5. The program executes but stops just before running the line in the breakpoint. The VB Editor shows the breakpoint line highlighted in yellow. You can now press F8 to begin stepping through the code.

After you have finished debugging your code, remove the breakpoints by clicking the dark brown dot in the margin next to each breakpoint to toggle it off. Alternatively, you can select Debug | Clear All Breakpoints or press Ctrl+Shift+F9 to clear all breakpoints that you set in the project.

Backing up or moving forward in code

When you are stepping through code, you might want to jump over some lines of code, or you might have corrected some lines of code that you want to run again. This is easy to do when you are working in Break mode. One favorite method is to use the mouse to grab the yellow arrow. The cursor changes

to a three-arrow icon, which enables you to move the next line up or down. Drag the yellow line to whichever line you want to execute next. The other option is to right-click the line to which you want to jump and then select Set Next Statement.

Not stepping through each line of code

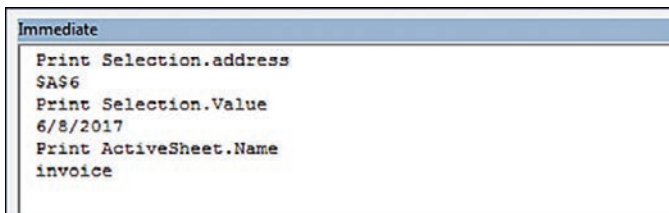
When you are stepping through code, you might want to run a section of code without stepping through each line, such as when you get to a loop. You might want VBA to run through the loop 100 times so you can step through the lines after the loop. It is particularly monotonous to press the F8 key hundreds of times to step through a loop. Instead, click the cursor on the line you want to step to and then press Ctrl+F8 or select Debug | Run To Cursor. This command is also available in the right-click menu.

Querying anything while stepping through code

Even though variables have not yet been discussed, you can query the value of anything while in Break mode. However, keep in mind that the macro recorder never records a variable.

Using the Immediate window

Press Ctrl+G to display the Immediate window in the VB Editor. While the macro is in Break mode, you can ask the VB Editor to tell you the currently selected cell, the name of the active sheet, or the value of any variable. Figure 2-10 shows several examples of queries typed into the Immediate window.



```
Immediate
Print Selection.address
$A$6
Print Selection.Value
6/8/2017
Print ActiveSheet.Name
invoice
```

FIGURE 2-10 Queries that can be typed into the Immediate window while a macro is in Break mode, shown along with their answers.



Tip Instead of typing Print, you can type a question mark: ? Selection.Address. Read the question mark as, "What is."

When invoked with Ctrl+G, the Immediate window usually appears at the bottom of the code window. You can use the resize handle, which is located above the Immediate title bar, to make the window larger or smaller.

There is a scrollbar on the side of the Immediate window that you can use to scroll backward or forward through past entries.

It is not necessary to run queries only at the bottom of the Immediate window. For example, if you have just run a line of code, type `?Selection.Address` in the Immediate window to ensure that this line of code worked.

Next, press the F8 key to run the next line of code. Instead of retyping the same query, in the Immediate window, click anywhere in the line that contains the last query and press Enter. The Immediate window runs this query again, displays the results on the next line, and pushes the old results farther down the window.

You also can use this method to change the query by clicking to the right of the word `Address` in the Immediate window. Press the Backspace key to erase the word `Address` and instead type `Columns.Count`. Press Enter, and the Immediate window shows the number of columns in the selection.

This is an excellent technique to use when you are trying to figure out a sticky bit of code. For example, you can query the name of the active sheet (`?ActiveSheet.Name`), the selection (`?Selection.Address`), the active cell (`?ActiveCell.Address`), the formula in the active cell (`?ActiveCell.Formula`), the value of the active cell (`?ActiveCell.Value` or `?ActiveCell` because `Value` is the default property of a cell), and so on.

To dismiss the Immediate window, click the X in its upper-right corner.



Note Ctrl+G does not toggle the window off. Use the X at the top right of the Immediate window to close it.



Tip The Immediate window is a very useful tool, and I always have it and the Watches window open. If your monitor is big enough, I highly recommend this setup. If you have multiple monitors, you can move these windows to another monitor.

Querying by hovering

In many instances, you can hover the cursor over an expression in code and then wait a second for a tooltip to show the current value of the expression. This is incredibly helpful when you get to looping in Chapter 5, “Looping and flow control.” It also comes in handy with recorded code. Note that the expression that you hover over does not have to be in the line of code just executed. In Figure 2-11, Visual Basic just selected E11, making E11 the active cell. If you hover the cursor over `ActiveCell.FormulaR1C1`, you see a tooltip showing that the formula in the active cell is `=SUM(R[-9]C:R[-1]C)`.

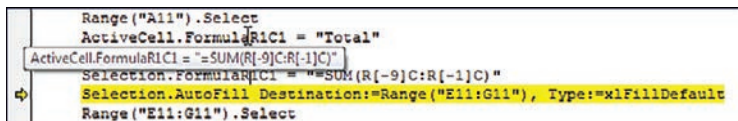


FIGURE 2-11 Hover the mouse cursor over any expression for a few seconds, and a tooltip shows the current value of the expression.

Sometimes, the VBA window seems to not respond to hovering. Because some expressions are not supposed to show values, it is difficult to tell whether VBA is not displaying a value on purpose or whether you are in the buggy “not responding” mode. Try hovering over something that you know should respond, such as a variable. If you get no response, hover, click into the variable, and continue to hover. This tends to wake Excel from its stupor, and hovering works again.

Are you impressed yet? This chapter started with a complaint that VBA doesn’t seem much like BASIC. However, by now, you have to admit that the Visual Basic environment is great to work in and that the debugging tools are excellent.

Querying by using a Watches window

In Visual Basic, a watch is not something you wear on your wrist; instead, it allows you to watch the value of any expression while you step through code. Let’s say that in the current example, you want to watch to see what is selected as the code runs. You can do this by setting up a watch for `Selection.Address`.

From the Debug menu, select Add Watch. In the Add Watch dialog, enter **Selection.Address** in the Expression text box and click OK (see Figure 2-12).

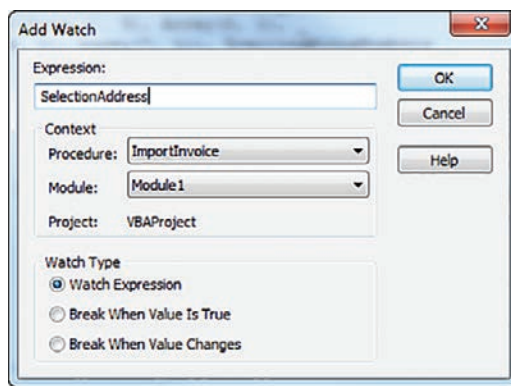


FIGURE 2-12 Setting up a watch to see the address of the current selection.

A Watches window is added to the busy Visual Basic window, usually at the bottom of the code window. When you start stepping through the code, it imports the file and then selects the last row with data. The Watches window confirms that `Selection.Address` is `A18` (see Figure 2-13).

Watches			
Expression	Value	Type	Context
Selection.Address	"\$A\$18"	Variant/String	Module1.ImportInvoice

FIGURE 2-13 Without having to hover or type in the Immediate window, you can always see the value of watched expressions.

Press the F8 key to run the code to the line after `Rows("1:1").Select`. The Watches window is updated to show that the current address of the `Selection` is now `$1:$1`.

In the Watches window, the value column is read/write (where possible)! You can type a new value here and see it change on the worksheet. For example, if your watch expression is `Selection.Value`, you can click on the value and enter a new one.

Using a watch to set a breakpoint

Right-click the `Selection.Address` expression in the Watches window and select `Edit Watch`. In the Watch Type section of the `Edit Watch` dialog, select `Break When Value Changes`. Click `OK`.

The glasses icon to the left of the expression changes to a hand with a triangle icon. You can now press `F5` to run the code. The macro starts running lines of code until something new is selected. This is very powerful. Instead of having to step through each line of code, you can now conveniently have the macro stop only when something important has happened. You also can set up a watch to stop when the value of a particular variable changes.

Using a watch on an object

In the preceding example, you watched a specific property: `Selection.Address`. It is also possible to watch an object such as `Selection`. In Figure 2-14, when a watch has been set up on `Selection`, you get the glasses icon and a + icon.

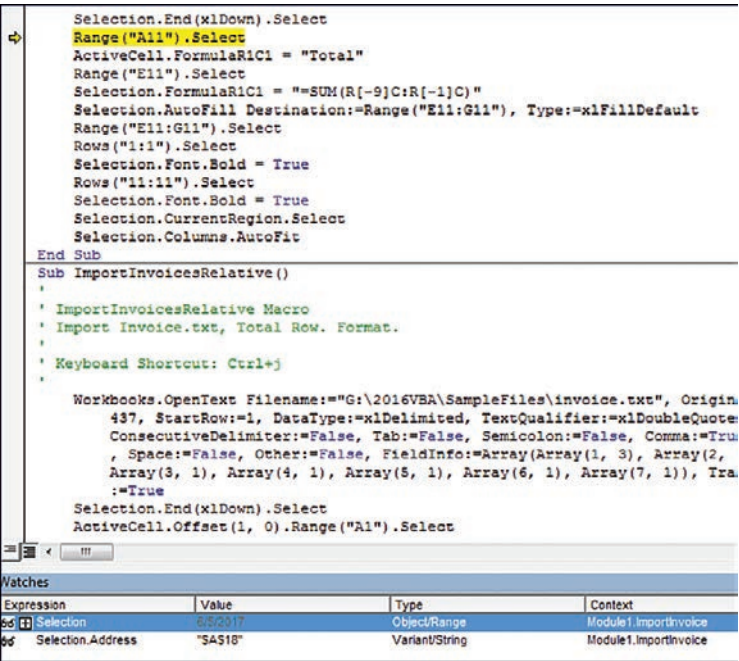
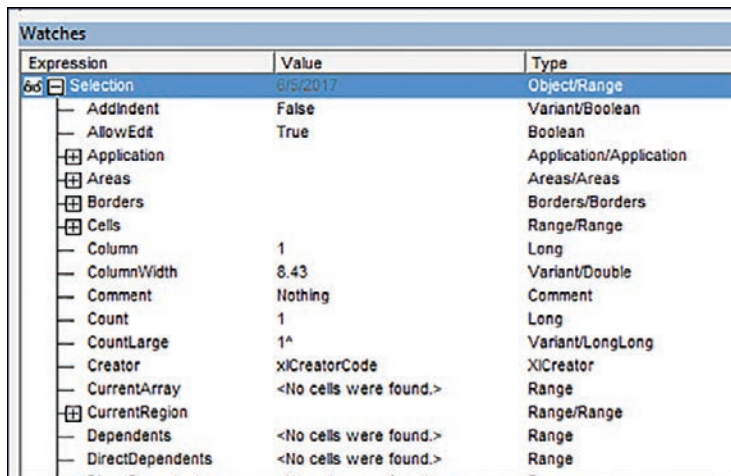


FIGURE 2-14 Setting a watch on an object gives you a + icon next to the glasses.

By clicking the + icon, you can see all the properties associated with `Selection`. When you look at Figure 2-15, you can see more than you ever wanted to know about `Selection`! There are properties

you probably never realized are available. You can see that the `AddIndent` property is set to `False`, and the `AllowEdit` property is set to `True`. There are useful properties further down in the list, such as the `Formula` of the selection.

In this `Watches` window, some entries can be expanded. For example, the `Borders` collection has a `+` next to it, which means you can click any `+` icon to see more details.



Expression	Value	Type
Selection	8/5/2017	Object/Range
AddIndent	False	Variant/Boolean
AllowEdit	True	Boolean
Application		Application/Application
Areas		Areas/Areas
Borders		Borders/Borders
Cells		Range/Range
Column	1	Long
ColumnWidth	8.43	Variant/Double
Comment	Nothing	Comment
Count	1	Long
CountLarge	1^	Variant/LongLong
Creator	xlCreatorCode	XlCreator
CurrentArray	<No cells were found.>	Range
CurrentRegion		Range/Range
Dependents	<No cells were found.>	Range
DirectDependents	<No cells were found.>	Range

FIGURE 2-15 Clicking the `+` icon shows a plethora of properties and their current values.

Object Browser: The ultimate reference

In the VB Editor, press `F2` to open the Object Browser, which lets you browse and search the entire Excel object library. The built-in Object Browser is always available; you simply press the `F2` key. The next few pages show you how to use it.

By default, the Object Browser opens where the code window normally appears. However, you can resize the window and reposition it anywhere you like, including on another monitor.

The topmost dropdown currently shows `<All Libraries>`. There are entries in this dropdown for Excel, Office, VBA, and each workbook that you have open, plus additional entries for anything you check in `Tools | References`. For now, go to the dropdown and select `Excel`.

In the bottom-left window of the Object Browser is a list of all classes available for Excel (see Figure 2-16). Click the `Application` class in the left window. The right window adjusts to show all properties and methods that apply to the `Application` object. Click something in the right window, such as `ActiveCell`. The bottom window of the Object Browser tells you that `ActiveCell` is a property that returns a range. It also tells you that `ActiveCell` is read-only (an alert that you cannot assign an address to `ActiveCell` to move the cell pointer).

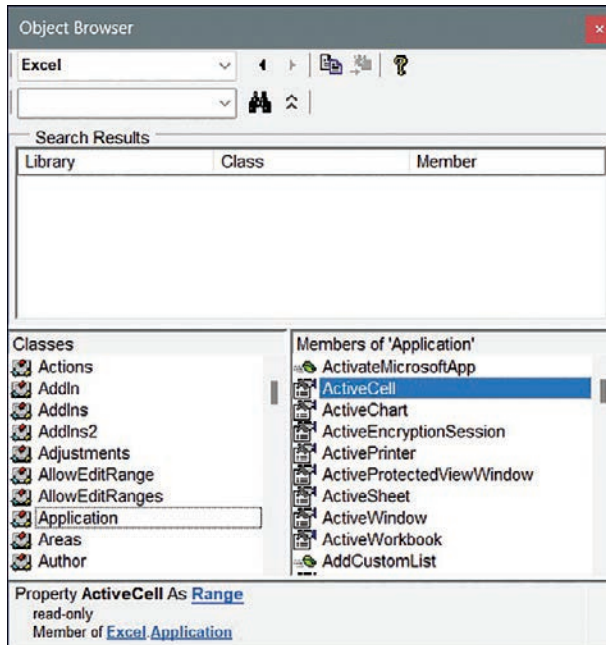


FIGURE 2-16 The Object Browser lets you explore objects, properties, methods, constants, and enumerations available in the VBA environment.

You have learned from the Object Browser that `ActiveCell` returns a range. When you click the hyperlink for `Range` in the bottom window, the `Classes` and `Members` windows update to show you all the properties and methods that apply to `Range` objects and, hence, to the `ActiveCell` property. Click any property or method and then click the yellow question mark near the top of the Object Browser to go to the online Help topic for that property or method.

Type any term in the text box in the Search field (next to the binoculars) and press Enter to find all matching members of the Excel library. Methods appear as green books with speed lines. Properties appear as index cards, each with a hand pointing to it.

The search capabilities and hyperlinks available in the Object Browser make it much more valuable than an alphabetic printed listing of all the information. Learn to make use of the Object Browser in the VBA window by pressing F2. To close the Object Browser and return to your code window, click the X in the upper-right corner.



Tip If you've maximized the Object Browser and are having trouble returning it to a floating window, try this: Drag it to the left or right edge of the VB Editor to dock it there. Once docked, move it back to the center of the code window—at that point, it should return to a resizable floating window.

Seven tips for cleaning up recorded code

Chapter 1 gave you four tips for recording code. So far, this chapter has covered how to understand the recorded code, how to access VBA help for any word, and how to use the excellent VBA debugging tools to step through your code. The remainder of this chapter presents seven tips to use when cleaning up recorded code.

Tip 1: Don't select anything

Nothing screams "recorded code" more than having code that selects things before acting on them. This makes sense in a way: In the Excel interface, you have to select row 1 before you can make it bold.

However, this is rarely done in VBA. There are a couple of exceptions to this rule. For example, you need to select a point on a chart before you can change its properties.

To streamline the code the macro recorder gives you, in many cases, you can remove the part of the code that performs the selection. The following two lines are macro recorder code before it has been streamlined:

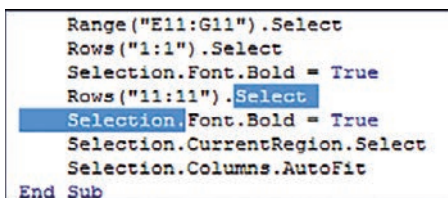
```
Cells.Select  
Selection.Columns.AutoFit
```

You can streamline the recorded code so it looks like this:

```
Cells.Columns.AutoFit
```

There are a couple of advantages to doing this streamlining. First, there will be half as many lines of code in your program. Second, the program will run faster because Excel does not have to redraw the screen after the lines that perform the selection.

After recording code, you can do this streamlining by highlighting the code from before the word `Select` at the end of one line all the way to the dot after the word `Selection` on the next line. Then, press Delete (see Figures 2-17 and 2-18).

A screenshot of a VBA code editor window showing a macro. The code is as follows:

```
Range("E11:G11").Select  
Rows("1:1").Select  
Selection.Font.Bold = True  
Rows("11:11").Select  
Selection.Font.Bold = True  
Selection.CurrentRegion.Select  
Selection.Columns.AutoFit  
End Sub
```


In the image, the text `Selection` on the line `Selection.Font.Bold = True` is highlighted in blue. The text `Selection` on the line `Selection.CurrentRegion.Select` is also highlighted in blue. The text `Selection` on the line `Selection.Columns.AutoFit` is also highlighted in blue. The text `Selection` on the line `Selection.Font.Bold = True` is also highlighted in blue. The text `Selection` on the line `Selection.CurrentRegion.Select` is also highlighted in blue. The text `Selection` on the line `Selection.Columns.AutoFit` is also highlighted in blue.

FIGURE 2-17 Select the part of the code highlighted here...


```

Selection.End(xlDown).Select
Selection.Offset(1, 0).Select
Range("A1").FormulaR1C1 = "Total"
Range("E11").FormulaR1C1 = "=SUM(R[-9]C:R[-1]C)"
Range("E11").AutoFill Destination:=Range("E11:G11"), Type:=xlFillDefault
Rows("1:1").Font.Bold = True
Rows("11:11").Font.Bold = True
Range("A1").CurrentRegion.Columns.AutoFit
End Sub

```

FIGURE 2-18 ...and press the Delete key. This is Cleaning Up Recorded Macros 101.

Tip 2: Use Cells(2, 5) because it's more convenient than Range("E2")

The macro recorder uses the Range() property frequently. If you follow the macro recorder's example, you will find yourself building a lot of complicated code. For example, if you have the row number for the total row stored in a variable TotalRow, you might try to build this code:

```
Range("E" & TotalRow).Formula = "=SUM(E2:E" & TotalRow-1 & ")"
```

In this code, you are using concatenation to join the letter *E* with the current value of the TotalRow variable. This works, but eventually, you have to refer to a range where the column is stored in a variable. Say that FinalCol is 10, which indicates column J. The column in the Range property must always be a letter, so you have to do something like this:

```
FinalColLetter = MID("ABCDEFGHIJKLMNOPQRSTUVWXYZ", FinalCol, 1)
Range(FinalColLetter & "2").Select
```

Alternatively, perhaps you could do something like this:

```
FinalColLetter = CHR(64 + FinalCol)
Range(FinalColLetter & "2").Select
```

These approaches work for the first 26 columns but fail for the remaining 99.85 percent of the columns.

You could start to write 10-line functions to calculate that the column letter for column 15896 is WMJ, but it is not necessary. Instead of using Range("WMJ17"), you can use the Cells(Row, Column) syntax.

Chapter 3, "Referring to ranges, names, and tables," covers this topic in complete detail. However, for now, you need to understand that Range("E10") and Cells(10, 5) both point to the cell at the intersection of the fifth column and the tenth row. Chapter 3 also shows you how to use .Resize to point to a rectangular range. Cells(11, 5).Resize(1, 3) is E11:G11.

Tip 3: Use more reliable ways to find the last row

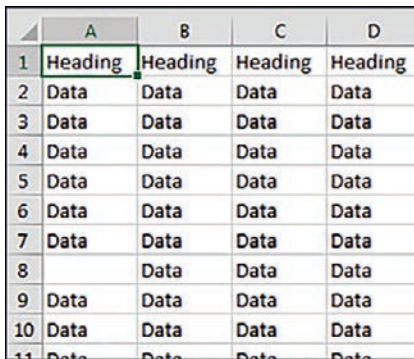
It is difficult to trust data from just anywhere. If you are analyzing data in Excel, remember that the data can come from who-knows-what system written who-knows-how-long-ago. The universal truth is that eventually, some clerk will find a way to break the source system and enter a record without an invoice

number. Maybe it will take a power failure to do it, but invariably, you cannot count on having every cell filled in.

This is a problem when you're using the End+down arrow shortcut. This key combination does not take you to the last row with data in the worksheet. It takes you to the last row with data in the current range. In Figure 2-19, pressing End+down arrow would move the cursor to cell A7 rather than the true last row with data.

One better solution is to start at the bottom of the worksheet and look for the first non-blank cell by using this:

```
FinalRow = Cells(Rows.Count, 1).End(xlUp).Row
```



	A	B	C	D
1	Heading	Heading	Heading	Heading
2	Data	Data	Data	Data
3	Data	Data	Data	Data
4	Data	Data	Data	Data
5	Data	Data	Data	Data
6	Data	Data	Data	Data
7	Data	Data	Data	Data
8		Data	Data	Data
9	Data	Data	Data	Data
10	Data	Data	Data	Data
11	Data	Data	Data	Data

FIGURE 2-19 End+down arrow fails in the user interface if a record is missing a value. Similarly, End(xlDown) fails in Excel VBA.

This method could fail if the very last record happens to contain the blank row. If the data is dense enough that there will always be a diagonal path of non-blank cells to the last row, you could use this:

```
FinalRow = Cells(1,1).CurrentRegion.Rows.Count
```

If you are sure that there are not any notes or stray activated cells below the data set, you might try this:

```
FinalRow = Cells(1, 1).SpecialCells(xlLastCell).Row
```

The xlLastCell property is often wrong. Say that you have data in A1:F500. If you accidentally press Ctrl+down arrow from A500, you will arrive at A1048576. If you then apply Bold to the empty cell, it becomes activated. Or, if you type **Total** and then clear the cell, it becomes activated. At this point, xlLastCell will refer to F1048576.

Another method is to use the Find method:

```
FinalRow = Cells.Find("*", SearchOrder:=xlByRows, _  
    SearchDirection:=xlPrevious).Row
```

You will have to choose from these various methods based on the nature of your data set. If you are not sure, you could loop through all the columns. If you are expecting seven columns of data, you could use this code:

```
FinalRow = 0
For i = 1 to 7
    ThisFinal = Cells(Rows.Count, i).End(xlUp).Row
    If ThisFinal > FinalRow then FinalRow = ThisFinal
Next i
```



Note ListObjects—Excel tables created by selecting Insert | Table—have their own set of properties that make some things, such as finding the last row used in the table, a lot easier. See “Referencing tables aka ListObjects” in Chapter 3 for more information.

Tip 4: Use variables to avoid hard-coding rows and formulas

The macro recorder never records a variable. Variables are easy to use, but just as in BASIC, a variable can remember a value. Variables are discussed in more detail in Chapter 4, “Laying the groundwork with variables and structures.”

It is recommended that you set the last row that contains data to a variable. Be sure to use meaningful variable names such as `FinalRow`:

```
FinalRow = Cells(Rows.Count, 1).End(xlUp).Row
```

When you know the row number of the last record, put the word *Total* in column A of the next row:

```
Cells(FinalRow + 1, 1).Value = "Total"
```

You can even use the variable when building this formula, which totals everything from E2 to the `FinalRow` of E:

```
Cells(FinalRow + 1, 5).Formula = "=SUM(E2:E" & FinalRow & ")"
```

Tip 5: Use R1C1 formulas that make your life easier

The macro recorder often writes formulas in an arcane R1C1 style. However, most people change the code back to use a regular A1-style formula. After reading Chapter 6, “R1C1 style formulas,” you’ll understand that there are times when you can build an R1C1 formula that is much simpler than the corresponding A1-style formula. By using an R1C1 formula, you can add totals to all three cells in the total row with the following:

```
Cells(FinalRow+1, 5).Resize(1, 3).FormulaR1C1 = "=SUM(R2C:R[-1]C)"
```

Tip 6: Copy and paste in a single statement

Recorded code is notorious for copying a range, selecting another range, and then doing an `ActiveSheet.Paste`. The `Copy` method, as it applies to a range, is actually much more powerful. You can specify what to copy and also specify the destination in one statement.

Here's the recorded code:

```
Range("E14").Select
Selection.Copy
Range("F14:G14").Select
ActiveSheet.Paste
```

Here's better code:

```
Range("E14").Copy Destination:=Range("F14:G14")
```

Tip 7: Use `With...End With` to perform multiple actions

If you are making the total row bold with double underline, a larger font, and a special color, you might get recorded code like this:

```
Range("A14:G14").Select
Selection.Font.Bold = True
Selection.Font.Size = 12
Selection.Font.ColorIndex = 5
Selection.Font.Underline = xlUnderlineStyleDoubleAccounting
```

For four of these lines of code, VBA must resolve the expression `Selection.Font`. Because you have four lines that all refer to the same object, you can name the object once at the top of a `With` block. Inside the `With...End With` block, everything that starts with a period is assumed to refer to the `With` object:

```
With Range("A14:G14").Font
    .Bold = True
    .Size = 12
    .ColorIndex = 5
    .Underline = xlUnderlineStyleDoubleAccounting
End With
```

See Chapter 4 for more information on using `With...End With` blocks.

Case study: Putting it all together—Fixing the recorded code

Using the seven tips discussed in the preceding section, you can convert the recorded code from Chapter 1 into efficient, professional-looking code. Here is the code as recorded by the macro recorder at the end of Chapter 1:

```
Sub FormatInvoice3()  
'ImportInvoice Macro  
Workbooks.OpenText Filename:="C:\Data\invoice.txt", Origin:=437, _  
    StartRow:=1, DataType:=xlDelimited, TextQualifier:=xlDoubleQuote, _  
    ConsecutiveDelimiter:=False, Tab:=False, Semicolon:=False, _  
    Comma:=True, Space:=False, Other:=False, FieldInfo:=Array( _  
    Array(1, 3), Array(2, 1), Array(3, 1), Array(4, 1), _  
    Array(5, 1), Array(6, 1), Array(7, 1)), TrailingMinusNumbers:=True  
Selection.End(xlDown).Select  
ActiveCell.Offset(1, 0).Range("A1").Select  
ActiveCell.FormulaR1C1 = "Total"  
ActiveCell.Offset(0, 4).Range("A1").Select  
Selection.FormulaR1C1 = "=SUM(R2C:R[-1]C)"  
Selection.AutoFill Destination:=ActiveCell.Range("A1:C1"), Type:= _  
    xlFillDefault  
ActiveCell.Range("A1:C1").Select  
ActiveCell.Rows("1:1").EntireRow.Select  
ActiveCell.Activate  
Selection.Font.Bold = True  
Application.Goto Reference:="R1C1:R1C7"  
Selection.Font.Bold = True  
Selection.CurrentRegion.Select  
Selection.Columns.AutoFit  
End Sub
```

Follow these steps to clean up the recorded macro code:

1. Leave the `Workbooks.OpenText` lines alone; they are fine as recorded.
2. Note that the following line of code attempts to locate the final row of data so that the program knows where to enter the total row:

```
Selection.End(xlDown).Select
```
3. You do not need to select anything to find the last row. It also helps to assign the row number of the final row and the total row to a variable so that they can be used later. To handle the unexpected case in which a single cell in column A is blank, start at the bottom of the worksheet and go up to find the last-used row:

```
'Find the last row with data. This might change every day  
FinalRow = Cells(Rows.Count, 1).End(xlUp).Row  
TotalRow = FinalRow + 1
```

Note that these lines of code enter the word `Total` in column A of the total row:

```
ActiveCell.Offset(1, 0).Range("A1").Select  
ActiveCell.FormulaR1C1 = "Total"
```

Better code uses the `TotalRow` variable to locate where to enter the word `Total`. Again, there is no need to select the cell before entering the label:

```
'Build a Total row below this
Cells(TotalRow,1).Value = "Total"
```

4. Note that these lines of code enter the `Total` formula in column E and copy it to the next two columns:

```
ActiveCell.Offset(0, 4).Range("A1").Select
Selection.FormulaR1C1 = "=SUM(R2C:R[-1]C)"
Selection.AutoFill Destination:=ActiveCell.Range("A1:C1"), Type:= _
    x1FillDefault
ActiveCell.Range("A1:C1").Select
```

There is no reason to do all this selecting. The following line enters the formula in three cells:

```
Cells(TotalRow,5).Resize(1, 3).FormulaR1C1 = "=SUM(R2C:R[-1]C)"
```

(The `R1C1` style of formulas is discussed in Chapter 6.)

5. Note that the macro recorder selects a range and then applies formatting:

```
ActiveCell.Rows("1:1").EntireRow.Select
ActiveCell.Activate
Selection.Font.Bold = True
Application.Goto Reference:="R1C1:R1C7"
Selection.Font.Bold = True
```

There is no reason to select before applying the formatting. The preceding five lines can be simplified to the two lines below. These two lines perform the same action and do it much more quickly:

```
Cells(1, 1).Resize(1, 7).Font.Bold = True
Cells(TotalRow, 1).Resize(1, 7).Font.Bold = True
```

6. Note that the macro recorder selects all cells before doing the `AutoFit` command:

```
Selection.CurrentRegion.Select
Selection.Columns.AutoFit
```

There is no need to select the cells before doing the `AutoFit`:

```
Cells(1, 1).Resize(TotalRow, 7).Columns.AutoFit
```

(The `Resize` method is discussed in Chapter 3.)

7. Note that the macro recorder adds a short description to the top of each macro:

```
'ImportInvoice Macro
```

You have changed the recorded macro code into something that will actually work, so you should feel free to add your name as author to the description and mention what the macro does:

```
'Written by Bill Jelen. Import invoice.txt and add totals.
```

Here is the final macro with a declaration of variables (see Chapter 4), with all the changes discussed above:

```
Sub FormatInvoiceFixed()
'Written by Bill Jelen. Import invoice.txt and add totals.
Dim TotalRow as Long, FinalRow as Long
Workbooks.OpenText Filename:="C:\Data\invoice.txt", Origin:=437, _
    StartRow:=1, DataType:=xlDelimited, TextQualifier:=xlDoubleQuote, _
    ConsecutiveDelimiter:=False, Tab:=False, Semicolon:=False, _
    Comma:=True, Space:=False, Other:=False, FieldInfo:=Array( _
    Array(1, 3), Array(2, 1), Array(3, 1), Array(4, 1), _
    Array(5, 1), Array(6, 1), Array(7, 1))
FinalRow = Cells(Rows.Count, 1).End(xlUp).Row
TotalRow = FinalRow + 1
Cells(TotalRow, 1).Value = "Total"
Cells(TotalRow, 5).Resize(1, 3).FormulaR1C1 = "=SUM(R2C:R[-1]C)"
Cells(TotalRow, 1).Resize(1, 7).Font.Bold = True
Cells(1, 1).Resize(1, 7).Font.Bold = True
Cells(1, 1).Resize(TotalRow, 7).Columns.AutoFit
End Sub
```

Next steps

By now, you should know how to record a macro. You should also be able to use Help and debugging to figure out how code works. This chapter provides seven tools for making the recorded code look like professional code.

The next chapters go into more detail about referring to ranges, looping, and the crazy but useful R1C1 style of formulas that the macro recorder loves to use.

Index

Symbols

symbol, 91–92, 482
[...last] entry, 95

Numbers

32-bit Excel, 337, 497, 498, 545
64-bit Excel, 312, 497, 498, 545

A

A1-style formulas

- formula duplication, 119–120
- R1C1 vs., 48, 117, 120–121
- relative references in, 72
- replaced with R1C1, 123–125

About dialog box, 549–550

above-average records, finding, 224

Absolute mode, 17, 21, 72

absolute references

- mixed with relative, 123
- R1C1 style, 122

ACCDB files, 497, 498

accelerator keys, 536–537

Access

- ACE engine, 497, 502, 503
- adding records to, 502
- connection strings for, 503
- creating shared databases, 500–501
- Excel compatibility and, 497–498
- large file use in, 469
- macros/forms database, 604
- reading/writing from, 499, 502
- self-updating workbooks, 511
- SQL generation, 505
- summarizing records with, 510
- VBA language used by, 473
- when to adopt, 497
- work with, 312

access tokens, 326–327, 434, 439

ACE (Access Connectivity Engine), 497, 502, 503

activating cells, 47

ActiveCell, 43–44

ActiveSheet.Paste, 49

ActiveWorkbook, 70, 71, 180

ActiveX Data Objects (ADO)

- ActiveX control buttons, 592–593
- DAO and, 498
- database connection via, 503, 515
- record deletion with, 509–510
- summarizing records with, 510
- tools, 501–502
- updates to, 511
- updating records with, 506–509

.AddChart2 method, 358–360, 361, 363, 564

AddFields method, 243–244

add-ins, Excel

- alternatives to, 605
- client installation of, 601–602
- client uses of, 597
- closing, 603
- converting workbooks to, 598–599
- installing/uninstalling, 597, 602
- Office Add-ins vs., 597
- removing, 603
- Save As for, 599–600
- security of, 602–603
- standard, 597–598
- UDF sharing via, 334
- updating, 602
- VB Editor to convert files to, 600
- version updates, 604
- visibility of, 598, 601

Add-ins, Office

- adding interactivity to, 612–615, 617–630
- Add-ins group to add, 4
- components of, 607
- CSS for, 616
- editing installed, 612
- Excel add-ins vs., 607

- how to create, 608–612
- HTML code in, 615
- initializing, 629
- JavaScript compatibility, 629
- program for writing, 608
- TypeScript vs. Office Scripts vs., 631
- use of, 605
- user access to multiple, 611
- web testing of, 608
- writing to content/task pane, 628
- writing to/reading from sheets, 629–630
- XML to define, 617
- Add-ins group, VBA, 4
- Add method
 - collection, 163–164
 - dictionary, 169
 - ranges, 72
 - sparkline, 408
- addressing
 - A1 style, 117
 - cells, 54, 55
 - R1C1 style, 117, 118
 - ranges, 57
- Advanced Edit, Office Scripts, 633
- Advanced Editor, 432, 435
- Advanced Filter
 - benefits of, 197
 - criteria, 205, 207
 - criteria ranges with, 214–224, 227, 230
 - dialog setup, 214
 - Excel interface to build, 205
 - extracting results to new locations in, 77
 - extracting unique values with, 205–213
 - Filter In Place in, 224–225
 - List Ranges in, 205, 207
 - output/input ranges, 208, 212, 224, 227
 - report production case study, 229–233
 - three basic actions in, 205, 207
 - unwanted row removal in, 320
 - using multiple filters, 206
 - VBA vs. Excel, 204
 - xlFilterCopy for, 225–229
- AdventureWorks, 515
- alerts
 - formula fail indicators, 20
 - read-only, 43
 - state class module to call, 306
 - suppressing Excel, 562
 - user alert/response code, 557
 - alert statements, JavaScript, 629
- analyzing data, xxviii, 423, 453–458
- AND criteria range, 216, 218
- Andreu, Fernando, 583
- AnyDesk, 564
- API, JavaScript, 618, 629
- API, Office Add-ins, 631
- API, Windows
 - 64-bit code modification in, 312
 - compiling for Windows versions, 92
 - credential refreshes, 436, 439
 - credentials for accessing, 430–431, 434
 - declarations, 543
 - declaration use, 545
 - duplicating existing queries, 440
 - Excel compatibility with, 545–546
 - functions and their use, 546–552
 - Mac use and, xxxii
 - M to call, 429
 - reducing access token requests, 326–327, 439
 - return data parameters, 434
 - streamlining queries, 442–443
 - understanding declarations, 544
 - USERID function, 337
- `Application.EnableEvents = False`, 134
- `Application.Index` function, 149
- `Application.InputBox`, 176
- `Application.OnTime`, 453–458
- applications
 - accessing currently running, 477
 - application-level events, 127, 136–140, 154
 - designing VBA, 635
 - distributing as an add-in, 597
 - hierarchy of objects and, 54
 - Office Add-ins as, 607, 608
 - pausing, 564–565
 - production of, xxix
- Archibald, Rory, 296
- Areas collection, 67–68
- arguments
 - default order of, 31
 - lookup column specs in, 305
 - multidimensional array, 143
 - named arguments and slicers, 270
 - passing arguments, 88–89
 - standard arguments for ribbon procedures, 577
 - syntax of, 332
 - wrong argument number errors, 589
- Array function, 144–145, 243

arrays

- adding rows to tables, 81–82
- applying math functions to, 628
- declaring, 141–144
- defined, 141
- dynamic, 150–151
- extracting elements from, 149, 317
- filling, 144–146
- how to work with, 146–149
- JavaScript, 621–622
- looping through, 146
- matrices as, 143
- multidimensional, 142–144
- object variables as, 89
- placing loops in, 99
- processed into collections, 165
- reading names with, 79
- referencing, 72
- resizing, 144, 150, 165
- size of, 150
- sorted numeric/alpha character, 346
- stored in names, 75
- transposing, 151, 321
- updating fields in, 147
- use of multiple, 147–149

assigning variables, 83–85

Assign Macro dialog, 447, 593, 594

Atlas Programming Management, 302

AutoFilter

- basic techniques, 200–204
- dynamic date ranges, 203–204
- dynamic filter criteria options, 203
- filtering by color, 201–202
- hiding rows with, 197–198
- isolating records with, 198
- multi-item selections with, 200
- replacing loops with, 197–200
- Search box selections and, 201
- top/bottom value filtration, 200–201
- turning off menus in, 233–234
- visible cell selection, 204

AutoFit command, 51

Automate tab, 631–632

Automate Work button, 631

automation

- loops in arrays for, 99
- mundane task, 105–107
- Office Scripts, 597, 631
- pivot table, 288

VBA to automate Word, 327

web query, 429

Word, 473–495

Autoshapes, 594

AutoSort, 257

AutoSum button, 20, 21–22, 23

averages

- finding above-average cells, 224
- formatting cells above/below average, 398–399
- See also top/bottom value filtration

B

backgrounds

- Form Control button, 592
- object fill patterns, 375
- sparkline, 423, 425
- theme colors for, 413, 415

Barresse, Zack, 82, 300

BASIC

- code written in, 26
- loops in, 99
- syntax of, 1
- VBA and, xxviii
- VBA vs., 1, 2, 25

Bevel, 372

binary number format, 416

binary workbooks, 5

blank cell formatting, 402–403

blank rows, 47

blank Word documents, 485

blocks, nesting, 97

block scoping, 619, 620–621

bookmarks, 327, 491–492

Boolean data type, 84

Boolean formula, 214

Boolean function, 93, 620

Boolean values, 63, 80, 335

borders, chart, 376

branching statements, 353

Break mode, 37, 38, 39, 554

breakpoints, 38, 42

Bricklin, Dan, 117

BubbleSort, 345

buttons

- adding macros to command buttons, 592–593
- adding option buttons, 187–188
- adding ribbon controls, 576–580
- closing forms with, 181

buttons

- disabling the X close button, 550
- HTML, 616
- icons and, 176
- images on, 583
- label, text box, and command buttons, 182–184
- macro buttons on built-in ribbons, 590–591
- QAT macro buttons, 591–592
- spin buttons, 189–191
- toggle buttons, 525, 541
- userform, 175
- ByRef, 88–90, 151
- ByVal, 88–90

C

- cache, pivot, 240–242
- calculated data fields, 284–285
- calculated items, 285–286
- .Calculation options, 254–256
- calling userforms, 178–179
- Case statements, 113–114
- CBool function, 335
- C drive, 6
- cell progress indicators, 310–312
- cells
 - activating, 47
 - addressing, 54, 55
 - character substitutions in, 343–344
 - checking emptiness of, 63–64
 - eliminating blank pivot table, 256
 - empty, 63, 64, 66, 147
 - finding first nonzero-length, 341–342
 - formatting blank/error, 402–403
 - formatting date-containing, 402
 - formatting duplicate/unique, 400–401
 - formatting text-containing, 402
 - highlighting for data visualization, 384
 - interior color summons for, 341–342
 - offsetting, 58
 - placing charts in cell notes, 302–304
 - sparkline, 423
 - SpecialCells selections, 314
 - unknown addresses, 57
 - value-based formatting for, 401–402
 - working only with visible, 204
- Cells object, 143
- Cells property, 56–57
- Change events, 133, 304, 526
- character sorting, 346–348
- character substitutions, 343–344
- ChartObjects, 357, 360, 364
- charts
 - .AddChart2 to create, 358–360
 - benefits of, 357
 - chart events, 127, 134–136, 154
 - chart sheet events, 135
 - chart update options, 302–304
 - color applied to, 365–366
 - combo charts, 376–379
 - copied into new workbooks, 323–326
 - data bars, 383
 - Excel version compatibility, 360, 363
 - exported as graphics, 380, 382
 - filtering, 366–367
 - formatting, 358, 363–376
 - inserted into Word, 327
 - legends, 372, 379
 - line settings, 376
 - map charts, 380
 - modifying preexisting, 363
 - names/titles for, 363, 364
 - old chart types, 360, 363
 - pie charts, 326
 - sparklines, 407
 - styles of, 360–363
 - templates for, 358
 - updating, 358
 - waterfall charts, 381–382
 - win/loss charts, 407, 421–422
 - See also data visualization; sparklines
- chart sheets, 357, 364
- ChartStyle property, 360
- Chart Styles gallery, 360
- CheckBox controls, 520–521
- circular references, 20
- class modules
 - accessing custom objects in, 161
 - adding properties to, 156
 - creating, 136
 - custom object creation in, 153
 - event access via, 127, 134, 137
 - Excel state class modules, 306–308
 - extracting data from reports with, 317
 - function of, 11
 - inserting new, 153–154, 155
 - managing userform collections via, 527–528
 - read-only collections with, 90
 - trapping chart events in, 154

- cleaning data, xxviii, 432
- Clear Contents, 63, 66
- client cursor, ADO, 501
- closing Excel, 456
- closing userforms, 550
- closing Word documents, 486
- cloud-based files, 297–298, 486
- COBOL, 25, 459, 460
- code
 - additions with VBA Extensibility, 315–316
 - case sensitivity of ribbon, 573
 - checking for "unexpecteds" in, 568
 - cleaning up, 96
 - clear, maintainable, 507
 - Excel action code, 16
 - flexible binding, 481–483
 - forgotten closing > errors, 586–587
 - indentation for readable, 100
 - JavaScript syntax, 618–619
 - modifying ribbon, 573
 - navigating through, 38–39
 - Programming window displays, 14–16
 - protected code, 568, 602–603
 - recorded macro code, 15
 - running loops of, 39
 - speed of unformatted, 146
 - stepping through, 36–38, 564
 - streamlining M queries, 434
 - variation management, 91
- Code Editor, 633–634
- code execution
 - breakpoints, 38
 - stepping through, 36–38
 - stopping, 38
- Code group, 4
- CodeName, 566–567
- Code window, xxxii
- coercionType property, 613, 629, 630
- collections
 - adding keys to, 166–167
 - adding/retrieving to/from, 163–164
 - arrays and, 151
 - benefits of, 169
 - checking for items in, 166–167
 - collection object components, 163
 - creating read-only, 90–91
 - custom objects in, 163, 164–166, 323
 - declaring/initializing, 163
 - definition of, 163
 - dictionaries vs., 168–169, 170
 - extracting data from reports with, 317
 - looping through controls with, 495
 - noncontiguous range, 67–68
 - objects and, 27
 - passed into procedures, 90
 - processing arrays into, 165
 - SERIES formula for, 323
 - UDTs and, 172
 - userform control collections, 527–529, 532
 - VBA syntax and, 30
- colons, 28, 133, 157, 557
- color
 - active userform field color, 538–539
 - applying chart color, 365–366, 372
 - AutoFilter to filter by, 200–202
 - changing cell, 29
 - changing object fill, 373–375
 - color scales, 383, 384, 385, 390–391
 - data bar, 383, 387, 396–398
 - data visualization and, 288
 - either/or decision colors, 112
 - filtering cells by, 201–202
 - For...Next loops, 102
 - Notepad ++ for add-in, 574, 608
 - pivot table fill color, 278
 - Range.Sort to sort by, 235
 - RGB colors, 417
 - row color, 60
 - slicer color, 269, 271
 - sparkline formatting, 413–418, 420, 421
 - summing cells based on interior, 341
 - Theme colors, 413
 - VBA charts and use of, 358
 - waterfall chart, 381–382
 - XML syntax, 574
- Column fields, 240
- ColumnOffset argument, 57, 58
- columns
 - array, 143, 151, 152, 321
 - column A class value, 113
 - copying/reordering subset of, 226–228
 - editing list ranges in, 206–207
 - Enums for positioning, 95
 - filtering, 201, 366
 - grand total column, 287
 - grouping dates to reduce, 254
 - lookup column, 305
 - number of slicer, 271

columns

- numbers vs. letters for, 56, 125
- pivot table, 243
- pivot table slicer columns, 269
- properties of contiguous, 61
- R1C1 references to, 117, 123
- Range() property, 46
- referencing table columns, 81
- referring to ranges of, 58
- resizing ranges by, 59
- sorting/concatenating, 344–346
- spark columns, 407
- Text to Columns utility, 297
- tracking dynamic, 321–322
- waterfall chart, 381
- writing columns for large files, 468
- ColumnSize argument, 59, 60
- Columns property, 61
- COM add-ins, 597
- combo boxes, 184–185, 210
- combo charts, 376–379
- comma-delimited files, 296, 463–466
- comma-delimited lines, 146, 297, 343
- command bars, 573
- command buttons, 182–184, 592–594
- comments
 - adding name, 76
 - changing, 14
 - With...End With blocks and, 97
 - HTML, 615
 - JavaScript syntax for, 614, 619
 - shortcut key, 14
- compact reports, 287
- comparative analyses, 423, 427
- compilers/compiling
 - compiler directives, 91–92, 328–329, 482
 - error catching by, 83, 84
 - errors in, 155, 570
 - following line insertion, 316
 - late binding and, 328, 476
 - Word constants, 479
- computer name retrieval, 547
- concatenation, 46, 344–346
- conceptual filters, 263–265
- conditional formatting
 - blank/error cell formatting, 402
 - data visualization via, 383
 - Excel options for, 311
 - finding records without, 202
 - formatting above/below average cells, 398–399
 - formatting top/bottom cells, 399–400
 - formatting unique/duplicate cells, 400–401
 - Formula property for, 397
 - method for adding, 385
 - NumberFormat property, 404–405
 - selecting visible cells with, 204
 - summing cells colored by, 341
 - use of multiple, 385
 - value-based formatting, 401
 - xlExpression type, 403
- Conditional Formatting menu, 383, 384, 394
- conditions
 - computed criteria to set up, 218
 - conditional formatting conditions, 385
 - data bar, 386
 - formulas and, 403
 - If...ElseIf...End If for multiple, 112
 - Select Case...End Select for multiple, 112
 - selecting formula-based, 216–224
 - testing, 111, 112
 - using, 111
- cone charts, 363
- connections, ADO, 501, 502
- connection strings, 503
- ConnectionStrings.com, 503
- constants
 - chart element constants, 368
 - constant value retrieval, 480–481
 - declaring, 85, 86
 - defined, 33–35
 - displaying, 86
 - flexible binding and, 482–483
 - icon sets and VBA, 392
 - JavaScript, 620
 - values of, 33
 - variables and, 86
 - Word-specific, 479–481
- Const values, 85, 94, 329
- content pane, writing to, 628
- controls
 - added to ribbon, 576–580
 - adding tip text to, 537
 - Developer tab, 4
 - modifying Word, 492
 - userform, 128, 180–193, 520–527, 530–533
- Copy method, 49
- counter variables, 100, 103
- CreateObject function, 477, 478, 485

CreatePivotTable method, 243
 criteria, 215–216
 CSS (Cascading Style Sheets), 597, 608, 609, 616
 Ctrl+* shortcut, 23
 Ctrl+A, 105, 106
 curly braces ({}), 618
 CurrentRegion property, 64, 110, 199, 204, 290
 cursor, ADO, 501, 502
 custom functions. See user-defined functions (UDFs)
 customized QAT, 591–592
 customizing ribbon code, 574–575
 custom objects

- About dialog box customization, 549–550
- adding methods to, 160–161
- adding properties to, 156–159
- checking collections for, 167
- collections to hold, 164–166
- dictionary storage for, 171–172
- existing keys to update, 168–169
- new class modules for, 153
- properties/methods of, 154–155
- using, 161–162

custom properties, 172–174
 custom sort orders, 310
 customui folder, 574–575, 581, 583, 586
 customui Tag Name error, 587
 Cut/Copy mode, 124

D

dashboards

- arranging query results on, 445–448
- creating, 422–423
- sparklines in, 407, 424–428

data

- analysis with Python, xxviii
- Application.OnTime to analyze, 453–458
- cleaning, 432
- copies to separate worksheets, 300–301
- field-specific actions required by, 459
- formatting, 149
- pivot cache, 240, 242
- pivot tables to summarize, 239
- Power Query to get/transform, 432
- protected from editing, 91
- sorting, 234–237
- sparklines to condense, 422–423
- web query returns, 434
- writing to add-in file data, 598

Data Access Objects (DAO), 497, 498, 499
 data bars

- added to ranges, 386–389
- data visualization via, 288, 383
- two colors of, 396–398

databases

- Access, 497
- adding fields on the fly, 515
- adding records to, 503–504
- adding tables to, 514
- checking for fields in, 513
- checking for tables in, 512
- connection strings, 503
- creating shared, 500–501
- database engines, 497–498
- database formatting, 106
- deleting records on, 509–510
- library types, 498
- looping multiple records in, 513
- multiuser access to, 500–501
- objects for accessing data in, 502
- raw data access, 158
- record retrievals, 504–506
- storing data in, 497
- summarizing records, 510–511
- updating existing records, 506–509
- See also Access

DataBodyRange property, 145
 Data fields, 240, 252, 285
 Data Model

- distinct value counts with, 252
- loading large files to, 470
- pivot table creation with, 241, 278–284
- pivot table vs. definitions in, 282
- slicers and, 269

Data Preview window, 460, 464, 465

Data Model

datasets

- AutoFilter actions on, 198
- charting summaries from, 326
- converting fixed-width reports to, 317–319
- converting multi-row record, 322–323
- creating dynamic, 73
- drill down queries into, 286
- extracting unique values from, 205–213
- filtering pivot table, 261–278
- finding the last row in, 46–48
- flexible code for resizing, 210
- highlighting highest-value row in, 404
- loops for large, 99

datasets

- navigation tips, 22–23
- pivot table replication for products in, 257–261
- pivot table results as, 245
- resizing large, 320
- selecting total, 23
- sorting of, 234–237
- tables for large, 80
- data sources
 - chart, 358
 - chart imports with differing, 323–324
 - external, for pivot tables, 281
 - live charts and changing, 303
 - multiple pivot tables for single, 248
 - OLAP, 257, 269
 - slicer cache, 269
 - sparkline, 408, 424
- Data tab, xxviii, 77, 206, 384, 400, 432, 470
- data types
 - arrays as containing multiple, 142
 - constants and, 86
 - conversions in Power Query, 452
 - naming custom objects, 159
 - parameters/variables, 89
 - private variable prefixes, 156
 - specifying array, 150
 - table relationships and, 280
 - user-defined types (UDTs), 172–174
 - variable, 84
 - Word automation and, 327–328
- data visualization
 - above/below average cell formats, 398–399
 - color scales, 390–391
 - data bars, 386–389, 396–398
 - icons, 288
 - icon sets, 391–396
 - pivot tables, 288
 - report enhancement via, 383
 - top/bottom cell formats, 399–400
 - VBA methods/properties for, 385
 - See also charts
- Data worksheet, 409, 424, 563, 604
- Date Filters, 263
- dates
 - custom object methods, 160
 - dynamic date ranges, 203–204
 - formatting cells with, 402
 - pivot table date filters, 264
 - pivot tables to group, 252–254
 - retrieving date/time of saves, 338
 - retrieving permanent date/time, 339
 - Timeline slicers, 276–278
 - userform spin buttons, 189–191
- DAX functions, 282
- debugging
 - compiler errors, 155
 - database record update code, 507
 - errors and choice to debug, 553–554
 - Office Scripts, 634–635
 - protected code and inability to, 568
 - userform controls, 181
 - userform debugging errors, 555–557
 - VB Editor tools for, 36–43
 - waterfall charts, 382
- Debug mode, 38, 41, 92, 155, 554, 555, 556, 564, 568, 603
- DecimalSeparator, 35
- declarations, API, 544
- declaring arrays, 141–142
- declaring parameters, 95
- declaring variables, 52, 83–85
- defaults
 - argument order, 31
 - calculated field, 285
 - default file types, 5
 - default parameter order, 28
 - initializing UDT values to, 172
 - Microsoft security default, 6
 - property values, 477
 - variable declaration, 84
- defined constants, 33–35
- defined names
 - creating, 70–75
 - definition of, 53
 - deleting, 78
 - global vs. local, 69
 - hiding/unhiding, 76
 - INDIRECT function and, 73
 - managing, 75–78
 - reading, 78
 - renaming, 75–76
 - reserving, 77–78
 - tables and, 80
- deletions
 - Access record, 502
 - accidental worksheet, 565
 - ADO to delete records, 509–510
 - backward For...Next for, 103
 - comma-separated value (CSV) file, 296
 - Kill statement for, 471

- name, 78
- pivot table, 249
- removing Excel add-ins, 603
- slicer cache deletion, 275
- delimited data, 146
- delimited files, 463–466
- designing charts, 360
- Design tab, 286–287, 361, 493, 505
- Design view, 11, 179
- developer accounts, 430
- Developer tab
 - accessing, 3–4
 - formatting buttons on, 593–594
 - icons on, 4
 - Macro Security, 6, 7
 - Record Macro dialog, 8–9, 460
 - Stop Recording icon, 9
 - trusting access to VBA in, 315
 - Use Relative References, 18
 - Word, 492
- dictionaries
 - benefits of, 169
 - collections vs., 168–169, 170
 - custom objects in, 171–172
 - how to use, 169–172
 - properties/methods of, 169
 - user-defined types (UDTs) and, 172
- Dim statements, 141, 357
- directives, compiler, 91–92, 328–329, 482
- directories
 - counting workbooks in, 336–337
 - listing files in, 293–295
- Disable VBA Macros Except Digitally Signed Macros, 7
- Disable VBA Macros With Notification, 7, 8
- Disable VBA Macros Without Notification, 7
- display-resolution information, 548–549
- distribution, late binding for, 481
- divide-by-zero error, 558
- Document object, 473, 484–486
- document.write statements, 629
- DoEvents line, 564, 565
- Do loops, 105–109
- Double data type, 84
- DTD/schema, 586
- duplicates, highlighting, 384, 385, 400
- duration of variables, 85
- dynamic arrays, 150–151, 321
- dynamic columns, 321–322
- dynamic cursors, ADO, 501, 502

- dynamic date ranges, 203–204
- Dynamic Filters, 203–204
- dynamic formulas, 73
- Dynamic Link Libraries (DLLs), 543
- dynamic variables in JavaScript, 620

E

- early binding
 - Access file projects, 499
 - data exports to XML, 301
 - Word projects, 473–476, 479, 480–483, 484
- Edit mode, 206
- either/or decisions, 112
- ElseIf statements, 113
- email address validation, 339–341
- embedded charts, 134–136, 357, 360, 364
- Enable VBA Macros setting, 7
- encryption keys, 569
- end+down arrow shortcut, 46–47
- End If statements, 112, 114
- End property, 35
- enumeration, 94–95
- Enum statements, 94–95, 152, 322
- EOF function, 108, 508–509
- equals sign, 73, 74, 75, 78
- ERP (enterprise resource planning) system, xxx
- Err.Number, 167, 350, 563
- Err object, 558, 563
- errors
 - API use and, 544
 - basic handling of, 557–558
 - case sensitivity, 586
 - checking for collection object, 167
 - cleaning recorded code, 31, 45–49
 - compiler, 155
 - custom ribbon errors, 586–590
 - custom toolbar to troubleshoot, 573
 - Debug mode invisibility of, 564
 - delayed arrival of, 565–568
 - detecting causes for, 554
 - encountered on purpose, 562–563
 - entering formulas as text, 462
 - On Error GoTo syntax to handle, 557–558
 - error trap to anticipate runtime, 224–225
 - Excel version compatibility, 570
 - formatting cells with, 402–403
 - formatting non-existent element, 372
 - generic handlers for, 558–559

- hiding all fields error, 262
- ignoring, 560, 562
- illegal window closing, 193–194
- imports over 1,048,576 rows, 466
- inevitability of, 553
- late binding and, 476
- locating origins of, 559–560, 563
- looping, 557
- macro recorder, 1–2, 12–14, 16, 23, 31
- macros and repetition of, 14
- missing userform control, 178
- misspellings in forms, 184
- M language error handling, 450
- New and late binding, 482
- page setup problems, 561–562
- protected code with, 568, 603
- RefEdit control errors, 524
- reliance on, 125
- syntax, 470
- troubleshooting tips, 559–560, 564, 573
- user assistance with, 564
- user entry verification, 193
- userform debugging errors, 555–557
- variable declaration errors, 85
- variable declaration to catch, 83
- workbook opening, 582
- Escobar, Miguel, xxviii
- Esoteric Software, 568, 569
- Evaluate method, 54
- events
 - application-level events, 127, 136–140
 - change, 303, 304
 - chart events, 127, 134–135
 - disabling/enabling, 129
 - event parameters, 128
 - levels of, 127–128
 - procedures triggered by, 127
 - Reset button and pausing of, 555
 - scrollbar event handlers, 526–527
 - tracking binary, 421
 - trapping, 154
 - userform, 179–193
 - using, 128–129
 - workbook events, 127, 129–131
 - workbook-level sheet events, 127, 131–132
 - worksheet events, 127, 132–134
- Excel
 - 32-bit, 337, 497, 498
 - 64-bit, 312, 497, 498
 - Access compatibility by version, 498
 - Access connection from, 503
 - accessing Excel file structure, 581
 - accessing Word from, 473
 - add-ins, 597–606
 - Advanced Filter use in, 197, 204
 - AutoFilter, 197
 - building an advanced filter in, 205
 - built-in functions, 331
 - checking for open files on networks, 547–548
 - closing, 456
 - code for actions in, 16
 - connecting to SQL Server from, 515–516
 - connection strings, 503
 - consulting on, 635
 - controlling Word from, 327
 - Data Model in, 278–279
 - empty cells in, 63, 64, 66, 147
 - encryption models, 569
 - ERP system in, xxx
 - errors, 554
 - filtering tools, 200–204
 - Filter In Place in, 224
 - formula-based conditions in, 218–219
 - formula duplication in, 119–121
 - grand total column, 287
 - learning, xxx
 - legacy versions of, 5, 77, 569, 574
 - manual tasks in, xxx, xxxi
 - modifying ribbon in, 573
 - object hierarchy, 54
 - Office Add-in interaction with, 629
 - OS platform compatibility, 571
 - pivot table and version compatibility, 239–240, 242
 - R1C1 and A1 options in, 117
 - registering Office Add-ins with, 607
 - relative references in, 71
 - RELS file in, 581–582, 588–589
 - renaming files in, 582–583
 - Search box, 266
 - security settings, 6
 - state class module, 306–308
 - substitutions, 343
 - summary queries from, 510–511
 - suppressing warnings by, 562
 - SWITCH function, 355
 - unzipping zipped files, 573
 - update scheduling, 455

- VBA language for, 473
- VBA support for, xxvii, xxix, 1, 3
- versions 97 to 365, xxxii
- Excel 5, 239
- Excel 97, 239, 569, 570
- Excel 97-2003 Workbook (.xls), 5
- Excel 365, 278, 331, 352, 569, 570
- Excel 2000, 239
- Excel 2007, 240, 242, 254, 385, 545, 546, 575
- Excel 2010, 240, 545, 546
- Excel 2013, 242, 360, 574
- Excel 2016, 360
- Excel Binary Workbook (.xlsb), 5
- Excel Custom Functions Straight to the Point, xxvii
- Excel JavaScript UDFs Straight to the Point, 607
- exceljunkie.wordpress.com, 296
- Excel Macro-Enabled Workbook (.xlsm), 5, 299
- Excel Online
 - automated collaboration in, 497
 - JavaScript UDFs for, 607
 - macros in, xxvii
 - Office Scripts for, 597
 - pivot table automation, 288
 - TypeScript performance in, xxvii
 - VBA support for, 630
 - See also Office Scripts
- Excel Tables: A Complete Guide for Creating, Using, and Automating Lists and Tables, 82, 300
- Excel Workbook (.xlsx), 4
- exp.com, 308

F

- FieldInfo parameter, 34, 463, 465
- fields
 - added to pivot table filter area, 251–252
 - added to pivot tables, 243–244
 - adding model fields to pivot tables, 281
 - calculated data fields, 284–285
 - checking for existing, 513
 - combining multi-row, 322
 - counting values in, 252
 - database updates and clarity of, 507
 - defined constants for, 34
 - direct property assignment, 244
 - field codes, 481
 - field type identification, 34
 - filtering unique combinations of multiple, 212–213
 - fixed-width file field markers, 460–463
 - grouping dates in, 252
 - manual filtering in pivot, 261–262
 - removing pivot table, 245
 - updating array, 147
 - userform field handling, 177
- File menu, 118
- file name specification, 194–196
- File Open dialog, 194
- files
 - accessing Excel file structure, 581
 - changing default types of, 5
 - chart graphic exports and types of, 380, 382
 - checking for open files on networks, 547–548
 - comma-delimited, 297
 - comma-separated value (CSV) files, 296
 - converted to add-ins, 599–600
 - counting workbook, 336–337
 - customui files, 574–575
 - detecting cloud-hosted, 297–298
 - file extension errors, 589
 - invalid file format errors, 589
 - listed in directories, 293–295
 - macro allowance and file type, 4–5
 - Office Add-in manifest files, 608
 - reading/parsing text files, 296–297
 - See also text files
- FileSystemObject, 336
- Filter formula, 300
- Filter In Place, 224–225
- filters
 - filtering charts, 366–367
 - unwanted row removal, 320
 - virtual date filters, 402
- filters, pivot table
 - adding fields to, 251–252
 - conceptual filters, 263–265
 - manual, 261–262
 - OLAP, 308–310
 - pivot table section names, 240
 - ShowDetail to filter record sets, 286
 - slicers to filter, 269–270
 - Timeline slicers, 276–278
 - top/bottom value filtration, 267–269
- fixed-width file fields, 460–461
- fixed-width reports, 317–319, 459
- fixed-width text files, 459–463
- flexible binding, 481–483
- flow control, 110–115

folders

folders

- counting workbooks in, 336–337
- customui folder, 574–575
- listing files in, 293–295

fonts

- chart title, 364
- color searches for, 201–202
- sparkline, 415, 423

For Each loop, 110

For Each...Next, 99

For each...next statements in JavaScript, 625

For loops, 367

for loops, JavaScript, 622–623

FormatConditions, 385

Format Control dialog, 593

Format tab, 368, 372–373

formatting

- above/below average cells, 398–399
- AutoFilter for, 198–200
- charts, 358, 363–376
- data, 149
- data bars, 386, 396
- data visualization options, 385
- duplicate cells, 400–401
- invalid file format errors, 589
- pivot tables, 246–247
- pivot table values, 278
- ranges in Word, 489–491
- resetting table formats, 314–315
- sparklines, 413–422
- unformatted code speed, 146
- unique cells, 400–401
- visible vs. hidden rows, 197
- See also conditional formatting

Form control buttons, 592–593

forms

- Access database for, 604
- form controls, 12, 592–593
- user entry to, 184
- Word form fields, 492–495

Forms module, 11

Formula property, 396, 397, 443

formulas

- A1-style formulas, 48, 119–120
- adding names to, 73
- data bar, 396
- dynamic, 73
- formatting decisions via, 403

LAMBDA, 350

named ranges in, 55

pivot table calculated field, 284–285

Power Query inside VBA, 470

R1C1 style. *see* R1C1 style

reading names with, 78

record selection based on, 216–224

referencing, 72

variables when building, 48

For...Next loops, 100–103, 451, 452

For statement

- For...Next loops, 100
- reversing order of, 103
- variables in, 102

Frame tool, 187–188

Frankston, Bob, 117

FreeFile function, 467

Function procedure, 88

functions

- API, 544, 546–552
- branching, 353
- built-in, 331
- calls, 93
- creating custom Power Query, 436–440
- creating custom VBA, 331–333
- database connection, 503
- Help on, 30
- JavaScript math functions, 627–628
- main subs to call, 559, 563
- private, 158, 160
- recursive, 336
- structure of JavaScript, 618
- subs and, 93
- troubleshooting tips, 559–560
- useful custom, 334–350
- See also user-defined functions (UDFs); specific function by name

G

general protection faults (GPFs), 316

Get & Transform, xxviii, 470

GetMeasure, 282

GetObject function, 478, 485

Get property, 158, 159, 160

GetValueFromName function, 79

GitHub, 582

Global Const., 74

globally unique identifier (GUID), 610, 611, 617

- global names
 - creating, 70
 - deleting, 78
 - local vs., 69
 - renaming, 76
- global variable initialization, 166
- global variables in M, 449–453
- Glow, 372
- González Ruiz, Juan Pablo, 306
- Go To Special dialog, 65, 66, 224
- gradients
 - chart formatting, 375
 - data bar, 386, 388
 - data visualization via, 383
 - object fill, 373–375
 - sparkline shading, 416
- grand total column, 287
- graphics
 - exporting charts as, 380, 382
 - userform, 188–189
- grouping dates/times, 252–254
- grouping userform controls, 527–529
- GroupName property, 187

H

- headings
 - copying, 206
 - tracking dynamic column, 322
- Hello World Office Add-in, 608–612, 622
- help
 - added to userforms, 536–540
 - Help file optional parameters, 32–33
 - Office Scripts, 634
 - OpenText method Help topic, 32
 - VBA Help files, 30–31
- hidden names, 76–77
- hidden pivot table fields, 261–262
- hidden rows, 197, 204
- Hide method, 178
- Hide Rows command, 204
- hiding userforms, 178–179
- hierarchy of objects, 54
- Ho, Don, 574
- hovering, querying by, 40–41
- HTML, 597, 607, 615–616, 617, 618, 629
- Hungarian notation, 156
- hyperlinks
 - formatting sparkline, 414
 - hyperlink address returns, 349–350

- running macros from, 595–596
- userforms with, 529–530

I

- icons
 - data visualization via, 288
 - filtering by, 202
 - icon sets, 383, 384, 391–396, 397
 - macro buttons on built-in ribbons, 590–591
 - Microsoft Office icon library, 583
 - Office Add-in, 617
 - reversing order of, 394
 - ribbon code with icons, 583–584
 - specifying ranges for, 393
 - user interface, 176
- ID retrieval, 337–338
- If...ElseIf...End If, 112, 450
- If statements
 - loop exits in, 107
 - M language and, 450–451
 - nesting, 114–115
 - testing conditions and, 111
- if statements, Javascript, 623
- If...Then...Else, 110–112, 355
- If...Then...Else...End If, 112
- If...Then...End If, 111
- If...Then statements, 91, 356
- ignoring errors, 560, 562
- images/pictures
 - added to userforms, 534
 - buttons with, 583
 - changing fill for, 374
 - custom ribbon images, 584–585
 - HTML tags for, 615
 - Office Add-in, 617
 - See also icons
- Immediate window, 33, 39–41, 92, 126, 361, 477, 555, 557, 600, 603, 605
- importing data
 - cleaning and, xxviii
 - CSV file imports, 296
 - daily data into tables, 320–321
 - text file imports, 459–470
 - Text Import Wizard, 33, 34
 - VBA for, xxvii
 - XML imports, 4
- Improv, 239
- indentation, 97, 100, 114, 352
- INDIRECT function, 73

InitializeAppEvent procedure

- InitializeAppEvent procedure, 136
- inner loops, 105
- InputBox, 93, 175–176
- input boxes, password protected, 312–314
- input range, Advanced Filter, 208, 212
- Insert | Table, 48, 53
- Insert Lines method, 316
- Insert menu, 4, 11, 30
- Integer data type, 84, 332
- Intersect method, 62–63
- invalid file format errors, 589
- invalid property assignments, 589
- IsEmpty function, 63, 64, 66, 147
- IT department, xxx, 3
- Item property
 - Cells property and, 56
 - dictionary, 169, 170, 172
 - retrieving data from collections with, 164
- lvy charting engines, 381

J

- JavaScript
 - alert, 629
 - API, 607, 618, 629
 - arrays, 621–622
 - case sensitivity of, 574
 - code changes for Office Add-ins, 629–630
 - comment syntax, 614, 619
 - document.write, 629
 - For each...next statements, 625
 - function structure, 618
 - if statements, 623
 - linking buttons to functions in, 616
 - for loops, 622–623
 - math functions in, 627–628
 - Office Add-ins interactivity with, 597, 607, 612–615, 617–630
 - operators, 625–627
 - Select...Case statements, 623–625
 - strings, 621
 - syntax of, 618–619
 - variables, 619–621
- Jet engine, 498
- Jiang, Wei, 310
- JKP Application Development Services, 543
- Jones, Kevin, 82, 300
- JS commands
 - alert, 629
 - document.write, 629
- JSON format, 434

K

- Kaji, Masaru, 296
- Kapor, Mitch, 117
- keys
 - checking for existing, 167, 172
 - collection, 164
 - updating custom objects with, 168–169
 - value queries via, 169
- Key value, 164
- Kill statement, 471, 560
- Klann, Daniel, 312

L

- label controls, 182–184
- Label Filters, 263
- LAMBDA functions
 - building, 350–352
 - converting Select Case to, 355–356
 - custom and macro-free, 350
 - Excel VBA and, 3
 - finding nonzero-length cells via, 341–342
 - limitations of, 350
 - reading tips, 352
 - sharing, 352
 - UDFs and, 331, 350
- languages
 - object-oriented, 25–26
 - procedural, 25–26, 27
 - VBA, 26–30
 - See also BASIC; M code; XML
- Lanzo, Livio, 301
- last rows, 46–48
- late binding
 - Access distribution with, 499
 - dictionary declarations with, 170
 - error avoidance via, 570
 - third-party QuickBooks access with, 328
 - Word projects, 473, 479, 480–483, 484
- legends, chart, 372, 379
- Len, 58, 64, 147
- Let expression, 433, 436, 619
- Let function, 293, 351
- libraries
 - =PY() function support for, xxviii
 - API vs. reference libraries, 543
 - database access, 498
 - early/late binding and, 328
 - external, and missing references, 570

- Word object libraries, 475, 476
- Word reference libraries, 480
- lifetime, variable, 85
- light spectrum, 417
- line breaks, JavaScript, 619
- line settings, chart, 376
- LinkSource argument, 80
- list boxes
 - combo boxes vs., 184–185
 - graphics for, 188–189
 - MultiSelect property of, 185–187
 - populating userform, 210–212
 - setting up multicolumn, 539–540
- List.Generate in M, 451–453
- listobjects
 - adding a, 80
 - arrays in, 148, 152
 - conditional formatting in, 383
 - data use in, 326
 - extracting values with, 208
 - filling arrays from, 145–146
 - methods for, 72
 - positioning on the worksheet, 234
 - properties of, 48
 - referencing, 80
 - sorting data with, 234
 - sparkline data source as, 408
 - working with, 81–82
- ListObject.Sort, 234, 236–237
- List Range, 205, 206
- lists
 - custom sort order for, 310
 - filtering OLAP pivot tables by, 308–310
- live charts, 302–304
- local names
 - adding comments to, 76
 - creating, 70
 - deleting, 78
 - global vs., 69
 - renaming, 76
 - reserved, 77–78
- lock types, 501
- log files, 337
- Logical AND criteria, 216, 218
- Logical OR criteria, 215–216, 218
- Long data type, 84
- LoopCounter, 354
- loops
 - AutoFilter to replace, 197–200

- basic constructs of, 99
- column filtering with, 367
- dictionary looping, 172
- Do loops, 105–109
- For Each loop, 110
- errors in, 557
- exiting, 104, 107
- files in folders for, 293–295
- If...Then...Else, 110–112
- importance of, 99
- JavaScript for loops, 622–623
- List.Generate in M, 451–453
- looping collections, 165
- looping database records, 513
- looping through arrays, 146
- manual pivot field filtering with, 261–262, 263
- memory arrays of, 57
- M language, 449–453
- nesting, 104
- For...Next loops, 100–102
- numeric values for, 56
- preventing endless, 129
- querying by hovering, 40
- rerunning, 39
- row by row, 61
- TOC2HTML, 354
- While...Wend loops, 109
- Lotus 1-2-3, 1–2, 117, 120
- lower bound (LBound), 142, 148

M

- m_ prefix, 156
- Mac
 - control buttons, 592
 - Excel compatibility with, xxxii, 571
- Macro dialog
 - changing settings in, 14
 - Developer tab, 4
 - new macro additions in, 447
 - running macros from, 10
- macro recorder
 - absolute actions and, 17
 - chart creation, 363
 - loading large files with, 470
- moving beyond, 432
 - Office Scripts, 631, 632–633
 - opening delimited files with, 464
 - opening fixed-width files with, 460

- printer settings and, 561
- R1C1 for, 125
- Range object references, 54
- relative references in, 72
- Word, 484
- See also recorded macros
- macros
 - Access database for, 604
 - Advanced Filter for, 197, 204, 210
 - attached to shapes, 594–595
 - benefits of automated, xxxi
 - break mode for, 554
 - breakpoints to pause, 38
 - canceling scheduled, 455
 - chart update, 303
 - cleaning recorded, 45–52
 - command button attachments for, 592–594
 - copied to new workbooks, 315
 - customization for running, 9
 - Debug mode error omission, 564
 - disabling, 7–8
 - encryption keys and slowed, 569–570
 - error handling for, 557, 564
 - Excel, 2
 - Excel Online, xxvii
 - file operation, 293–298
 - file types allowing, 4–5
 - flawed recorder for, 1–3, 12–14, 16, 23
 - grouping dates in, 254
 - how to run, 10
 - hyperlinks to run, 595–596
 - keyboard shortcuts to run, 590
 - macro-enabled workbooks, 5
 - pausing, 564
 - perfecting, 21
 - playing back, 4
 - procedure-scheduling, 453–458
 - Python and, xxviii
 - QAT macro buttons, 591–592
 - R1C1 for writing, 117
 - recording, 8–9
 - reducing mundane tasks via, 106
 - running, 9, 10
 - running multiple, 554
 - scheduling, 455, 456–458
 - security concerns, 5–8
 - stepping through, 36–38
 - storing, 9
 - testing, 16, 19
 - writing VBA, xxviii
- Macro Security, 4, 6, 7, 315
- mail merges, 327
- main subs, 559, 560
- manifest files, 608, 610
- ManualUpdate, 240, 241
- map charts, 380
- Master Your Data with Excel and Power BI: Leveraging Power Query to Get & Transform Your Task Flow, xxviii
- Material, 372
- math functions, JavaScript, 627–628
- matplotlib, xxviii
- matrices, 143
- MAX function, 349
- maximum values, duplicate, 348–349
- M code
 - building queries using, 432–434
 - custom functions in, 437
 - definition of, 432
 - error handling, 450
 - global variables and loops in, 449–453
 - If logic in, 450–451
 - let expression, 436
 - List.Generate in M, 451–453
 - loading large files with, 470
 - streamlining code, 434
 - unwanted row removal via, 320
 - web queries via, 429
 - See also Power Query
- MDB files, 497, 498
- Measures dialog, 282
- Melissa virus, 6
- memory
 - arrays, 57, 146, 150
 - caching access tokens in, 326–327
 - global variable initialization and, 166
 - large data sets into, 320
 - reading/parsing text files in, 296–297, 323, 459
 - unloading userforms from, 178
- menus, turning off dropdown, 233–234
- methods
 - collection object methods, 163
 - custom object, 154–155, 160–161
 - data bar, 386
 - data visualization, 385
 - default parameter order, 28
 - dictionary, 169
 - Help on, 30–31
 - parameters for, 27–28
 - slicer/slicer cache, 271
 - storing multiple, 110

- table, 81
 - userform control, 180
 - VBA syntax and, 26, 27–28, 30
 - See also specific method by name
 - Microsoft
 - default security settings, 6
 - Office.js library, 612
 - Outlook, 6, 473, 617
 - pivot table rollout, 239
 - Power Query, 429
 - R1C1 and A1 options, 117
 - spreadsheet programs, 1–2
 - See also Windows; Word
 - Microsoft 365, 631
 - Microsoft ADO Library 2.8, 499
 - Microsoft AdventureWorks, 515
 - Microsoft Excel objects module, 11
 - Microsoft Excel VBA and Macros, version 8
 - content overview, xxviii–xxix
 - elements/conventions in, xxxii–xxxiii
 - Excel overview in, xxx
 - online resources/support for, xxxiii
 - Microsoft Office
 - icon library, 583–584
 - Office 2010, 476
 - Office 2013, 569
 - Office icon library, 583–584
 - Office.js API, 597, 607, 612
 - Office RibbonX Editor tool, 582
 - professional version, 500
 - VBA control of, 473
 - Microsoft Scripting Runtime, 336
 - Microsoft Teams, 297
 - Microsoft Word. See Word
 - Miles, Tommy, 298, 299
 - Moala, Ivan F., 314
 - mobile environments, xxvii
 - modal userforms, 529
 - modeless userforms, 529
 - ModelMeasures.Add, 282
 - Modern Excel files, 297
 - modules
 - adding userform, 177
 - calling userforms from, 178
 - class, 90, 127, 134
 - closes and insertion of new, 19
 - editing, 12
 - events in, 127, 128
 - inserting, 11
 - isolating code in, 570
 - private module-level variables, 156
 - Project Explorer window, 11
 - referencing custom objects from, 161
 - UDF entry into standard, 332
 - VBA Extensibility to export, 315
 - MoveNext method, 513
 - MrExcel.com, xxxi, 118, 301, 352, 635–636
 - MsgBox function, 93, 175, 176–177
 - multidimensional arrays, 142–144
 - MultiPage control, 191–193
 - multiplication tables, 123–124
 - MultiSelect property, 185–187, 210, 212
 - mundane tasks, 105–107
 - myList, 90
- ## N
- named arguments, 270
 - Name Manager dialog, 69, 70, 72, 73, 75, 76, 350, 351
 - Name property, 76, 155, 172, 177, 181, 244, 285, 566
 - names
 - array storage via, 75
 - chart, 363, 364
 - class module, 153, 156
 - CodeName vs. sheet names, 566–567
 - collection, 27
 - computer name retrieval, 547
 - converted to objects, 54
 - custom object property, 159
 - custom slicer cache, 274–275
 - defined names, 69–80
 - definition of, 53
 - With...End With for, 49
 - Enum, 95
 - file name userforms, 194–196
 - function, 93
 - global vs. local, 69
 - hiding/unhiding, 76
 - macro, 8, 13
 - new, sheet-based workbook, 298
 - numbers stored as, 75
 - object, 27
 - object use in, 70
 - option button, 187–188
 - parameter, 93
 - pivot table section, 240
 - private variable prefixes, 156
 - range, 54–55

- renaming Excel files, 582–583
- storing formulas in, 73
- string, 73–74
- underscore prefix, 69
- user-defined type (UDT), 172
- userform, 177, 179
- userform control, 181
- variable, 48
- navigation
 - ADO tools for, 501, 502
 - bookmarks to navigate Word, 491
 - in-code, 38–39
 - final row/column, 22
 - keyboard shortcuts for, 633
 - macro recorder absolute actions and, 17
 - modeless userforms for, 529
 - Office Scripts, 633
- nesting
 - If statements, 114–115
 - loops, 104–105
 - statements, 96
- networks, checking for files on, 547–548
- New keyword, 473, 477, 482
- New Script button, 631
- Next statement, 101, 557
- noncontiguous ranges, 61, 62, 67–68
- Notepad, 574, 608, 609
- Notepad ++, 574, 608
- note shapes, 302
- NOW function, 339, 551
- NumberFormat property, 404–405
- numbers
 - binary storage of, 416
 - changing variables to, 620
 - column numbers/letters, 125
 - declaring data types with, 86
 - enumerations for, 94
 - highlighting duplicate, 385
 - number format for data fields, 244
 - number names, 75
 - numbers and letters in names, 77
 - numeric fields in Values, 281–282
 - reading names with, 78
 - referencing, 72
 - sorting numeric/alpha characters, 346–348
 - sparklines with positive/negative, 418
 - text conversion in Power Query, 452
- NumPy, xxviii

O

- Object Browser, 43–44, 473, 474, 480–481
- Object data type, 84
- object models, 473
- object-oriented languages
 - loop unique to, 99, 110
 - vs. procedural, 25–26
 - VBA as, 26–30
- objects
 - accessing, 87
 - arrays and, 89
 - attaching macros to, 594–595
 - available JavaScript, 629
 - browsing library of, 43–44
 - changing fill for, 373–375
 - collections and, 27
 - creating, 11, 478
 - creating custom, 153, 154–161, 317, 323
 - defined, 26
 - defined names and, 70, 75
 - early binding to reference Word, 473–476
 - With...End With to name, 49
 - Err object, 558, 563
 - events available for, 128
 - Help on, 31
 - hierarchy of, 54
 - late binding to reference Word, 476–477
 - naming, 27
 - properties of custom, 156–159
 - properties returning objects, 36
 - properties to describe, 29
 - Range object, 53
 - showing relations between, 97
 - slicer cache, 271
 - sound objects, 551–552
 - userform controls as, 180
 - validating existence of, 566, 567
 - VBA control and object models, 473
 - VBA syntax and, 25, 26, 30
 - watches on, 42–43
 - Word, 484–492
 - Worksheet object, 68
- object variables, 86–90, 110, 208, 386, 475, 477
- Office. See Microsoft Office
- Office Scripts
 - accessing, 631
 - add-ins and, 597
 - availability of, 631

- editing/managing scripts, 633–634
- Excel Online use of, 630
- pivot tables with, 288–291
- recording macros in, 631, 632–633
- running/debugging, 634–635
- TypeScript as base for, 631
- Offset property, 17, 55, 57–59, 199
- OLAP data sources, 257, 269
- OLAP pivot table filtering, 308–310
- Oliver, Nathan P., 293
- one-dimensional arrays, 142, 143, 146
- OneDrive, 289, 297, 486, 597, 630, 631
- On Error GoTo syntax, 557–558, 561
- On Error Resume Next, 66, 471, 479, 561
- OnTime method, 453, 454, 455
- OpenText method, 31, 32, 33, 34, 37, 321, 460, 467
- operators, JavaScript, 625–627
- optimistic lock types, 501, 502
- optional parameters, 32–33
- Option Base statement, 142
- option buttons, 187–188
- options, predefined, 94
- OR criteria range, 215–216, 218
- Orientation property, 244, 245
- outer loops, 105
- outline report layouts, 287
- Outlook, 6, 473, 617
- output range, Advanced Filter, 208, 212, 224
- Output tab in Office Scripts, 634
- overlapping ranges, 62–63
- Özgür, Suat M., xxvii, 352, 607

P

- Page fields, 240, 251–252
- page setup problems, 561–562
- pandas, xxviii
- paragraphs, formatting Word, 489–491
- parameters
 - Cells property for, 57
 - colon as, 28
 - Enums to declare, 95
 - event, 128
 - GetValueFromName, 79
 - LAMBDA function, 351
 - naming, 93
 - numeric values for, 56
 - optional, 32–33
 - ordering, 28

- passing variables as, 88–89
- SERIES formulas for, 324
- sparkline, 408
- VBA syntax and, 30
- web query, 434
- parentheses, 93
- parsing text files, 296–297
- passwords
 - Excel add-in security and, 602–603
 - passworded chart sheets, 359
 - passworded input boxes, 312–314
 - recovery plans and, 569
 - VBA password weaknesses, 569
 - See also security
- pattern, color and, 375
- Peltier, Jon, 382, 564
- PeltierTech.com, 382
- percentage calculations, 254–256
- Personal.xlsb, 9, 334, 352
- pessimistic lock types, 502
- pictures. See images/pictures
- pie charts, 326
- Pieterse, Jan Karel, 543
- PivotCache.Refresh, 240
- PivotCaches.Add, 242
- PivotFilters.Add2, 242
- pivot tables
 - adding model fields to, 281
 - advanced features, 251–261
 - attached to existing slicers, 275
 - benefits of, 239, 291
 - calculated data fields, 284–285
 - calculated items, 285–286
 - changing layout of, 286–287
 - changing sections of, 248
 - code storage, 239
 - conceptual filters, 263–265
 - controlling subtotals, 245–246
 - converted to values, 248–251
 - counting records in, 252
 - creating/configuring, 243–244, 281
 - Data Model to create, 278–284
 - data visualization options, 288
 - deleting, 249
 - eliminating blank cells in, 256
 - Excel version compatibility, 239–240, 242
 - filtering datasets in, 261–278
 - formatting, 246–247
 - formatting value intersections, 278

pivot tables

- grouping dates in, 252–254
 - large file data sources for, 470
 - layout settings, 287
 - learning, xxx
 - limitations of, 248
 - manual production of, xxx
 - Office Scripts for, 288–291
 - percentage calculations in, 254–256
 - pivot cache definition, 240–242, 281
 - pivot table events, 127, 129, 131
 - removing fields from, 245
 - report replication with, 257–261
 - same data source for multiple, 248
 - search filter, 266
 - section names, 240
 - size of finished, 248–251
 - slicers to filter, 269–274
 - Timeline slicers for, 276–278
 - top/bottom value filtration, 267–269
- pixels, 417
- Point mode, 206
- Power Automate, 289, 631
- Power Pivot, 241, 278, 282, 423, 470
- PowerPoint, 473
- Power Query
- Application.OnTime, 453–458
 - benefits of, 429
 - building queries using M, 432–434
 - caching access tokens in, 327
 - custom function creation, 436–440
 - dashboard query arrangement, 445–448
 - data privacy queries, 435
 - duplicating existing queries, 440–441
 - learning, xxx
 - loading large files with, 470
 - procedure scheduling, 453–458
 - storing global variables in, 449–450
 - streamlining, 442–443
 - syntax in VBA, 470
 - text/number conversions in, 452
 - unwanted row removal in, 320
 - VBA threat of, xxviii
 - VBA to generalize queries, 442–448
 - web queries via, 429
- Power Query Editor, 432, 437, 439, 445
- PowerQueryFormatter.com, 434
- predefined options, 94
- Preserve keyword, 150, 151
- Print_Area listing, 77–78
- printer settings, 561
- printing Word documents, 486
- privacy, 435
- private variables
- assigning properties to, 157
 - naming tips, 156
- Problems tab in Office Scripts, 634
- procedural languages, 25
- procedures
- API, 544
 - Collection passed to, 90
 - custom object, 154
 - declaring object variables in, 87
 - endless loops for, 129
 - passing arguments to, 88
 - passing arrays into, 151–152
 - passing values to, 128
 - preventing called, 134
 - private procedure-level variables, 156
 - scheduling, 455
 - standard arguments for ribbon, 577
 - triggered by events, 127, 128
- productivity increases, xxx, 1, 293
- programming
- advanced userform, 175
 - data access for programmers, 158
 - early binding for, 481
 - events, 127
 - expanding skills in, 2
 - JavaScript, 612
 - loops in, 99
 - procedural languages, 25
 - streamlining orientation in, 293
 - userform controls, 180–181
 - userforms, 179–180
 - VBA vs. BASIC, 2
- Programming Management, 302
- Programming window
- code displays in, 11, 14–16
 - VB Editor, 10
- progress indicators, 310–312
- Project Explorer, 10, 11, 153, 177, 181, 566, 605
- properties
- assigning pivot table field, 244
 - assigning private variable, 157
 - chart, 360
 - collection object properties, 163
 - contiguous column/row, 61
 - custom object, 156–159, 161

- data bar, 386
- data visualization, 385
- dictionary, 169
- icon set, 391
- invalid property assignments, 589
- late binding and specifying, 477
- object-describing, 29
- passed to objects, 159
- productivity increases, 154–155
- returning objects, 36
- Selection, 42
- storing multiple, 110
- table, 81
- UDTs to create custom, 172–174
- userform control, 178, 180
- VBA syntax and, 30
- viewing add-in, 601

Properties window, 10, 12, 177, 178, 181, 189, 540, 566, 594

Property Get procedures, 158–159

Property Let procedures, 157

Property Set procedures, 159

protected code, 568

Puls, Ken, xxviii

=PY() function, xxviii

pyramid charts, 363

Python, xxviii

Q

queries

- access token requests, 326
- changing, 40
- database retrieval queries, 504–505
- hovering for, 40–41
- summarizing with ADO, 510–511
- variable value queries, 39
- web. see web queries

Quick Access Toolbar (QAT), 432, 457, 591–592

Quick Analysis, 21–22

QuickBooks, 92, 328–329

R

R1C1 style

- A1 vs., 48, 117, 120–121
- benefits of, 117, 120
- numbers, 125
- reference style, 72, 121–125
- replacing A1 with, 123–125
- switching to, 118

Range object

- Cells property to select, 56–57
- columns and rows of, 61
- frequency of use, 53
- properties and methods of, 43–44, 55
- properties returning, 36
- referring to, 54, 57–61
- syntax of, 54
- using named, 73
- Word, 487–491
- worksheet property of, 54, 55

Range property, 46, 55, 57, 68, 81, 145

ranges

- added to Data Model, 280
- adding color scales to, 390–391
- adding data bars to, 386–389
- adding icon sets to, 391–394
- adding named, 70–71
- contiguous, 61
- defined, 53
- determining pivot table, 248–251
- editing in Advanced Filter, 206
- finding nonzero-length cells in, 341–342
- hierarchy of objects and, 54
- highlighting first unique value in, 403
- icon sets for range subsets, 394–396
- joining multiple, 61–62
- macro recorder and, 46
- named, 55
- new range from overlapping, 62–63
- non-absolute references for, 71–72
- noncontiguous, 61, 62, 67–68
- object variables for input/output, 208
- offsetting, 57–59, 199–200
- Range.Sort for data sorting, 234–235
- reading names with, 78
- RefEdit to select, 523
- referencing non-active sheet, 68, 71
- referencing table ranges, 81
- resizing, 59–60
- sorting, 345, 347
- Subscript out of range error, 565–566
- syntax of, 54
- Word, 487–491

Range.Sort, 234–235

readability, indentation for code, 97, 100

reading names, 78

reading text files, 296–297, 459–470

read-only locks, 502

Ready mode

- Ready mode, 454
- Record Actions button, 631, 632
- recorded macros
 - Absolute mode for, 17
 - alternate methods for, 23
 - AutoSum/Quick Analysis and, 21–22, 23
 - cleaning up, 45–49
 - code comprehension via, 30
 - code for, 15
 - correcting flaws in, 3, 45–52
 - current settings and, 35
 - debugging tools, 36–43
 - flaws in, 1–2, 12–14, 16, 23
 - how to record, 8–9, 12–13, 22–23
 - loops unrecorded by, 110
 - loop support for, 116
 - Office Scripts pivot table options, 289–290
 - recognizing failures in, 21
 - reducing mundane tasks via, 106
 - Relative mode for, 17–21, 22
 - Search box selections and, 201
 - streamlining and, 45
 - VB Editor to review code of, 31–32
 - window arrangement for, 37
 - See also macro recorder
- Record Macro dialog, 8–9, 13, 14, 125, 460, 590
- records
 - adding database, 503–504
 - Advanced Filter to extract, 204, 214
 - counting pivot table, 252
 - deleting, 103–104
 - deleting database, 509–510
 - editing Access database, 502
 - filtering and catching no, 224–225
 - filtering by top/bottom, 200–201
 - finding above-average, 224
 - isolated with AutoFilter, 198
 - looping multiple database, 513
 - multi-row record conversions, 322–323
 - record number and text file imports, 459, 466, 468
 - retrieving database, 504–506
 - ShowDetail to filter record sets, 286
 - summarized with ADO, 510–511
 - updating existing database, 506–509
 - xlFilterCopy for all, 225–229
- recordsets, 498, 501, 502, 509
- recursion, 352–353, 354
- recursive functions, 336
- ReDim Preserve, 321
- ReDim statements, 147
- RefEdit controls, 523–525
- RefersTo value, 70, 75, 78, 79
- RefreshTable, 240
- refresh tokens, 434, 439
- relationships
 - Excel file, 581–582, 588–589, 590
 - table, 280
- Relative mode, 17–18
- relative references
 - correct range and, 71–72
 - mixed with absolute references, 123
 - Offset property and, 57
 - R1C1 style, 121–122
 - rearranging into databases, 105
 - recording macros with, 17–21, 22
 - SUM function and, 21
- RELS file, 581–582, 588–589, 590
- renaming names, 75–76
- report production
 - advanced filters for, 229–233
 - automating, 3
 - comparative analyses, 423, 427
 - data visualization for, 383
 - date grouping in, 254
 - duplicating reports, 3
 - extracting data from reports, 317–319
 - field selection for, 227
 - hidden template sheets for, 229
 - importing daily data to tables, 320–321
 - joining multiple ranges for, 62
 - layout settings, 287
 - manual, xxviii
 - percentages in, 254
 - pivot tables for, 239, 241, 251, 267–269
 - recorded macros for, 14
 - record extraction with Advanced Filter, 204
 - replication and, 257–261
 - templates for, 323
 - VBA for, xxvii, xxviii, 233
 - web query report, 429
 - Word for, 327–328
 - xlFilterCopy for, 225–229
- Require Variable Declaration setting, 11, 84
- reserved names, 77–78
- Reset button, 38, 554, 555
- Reset command, 38
- resetting table formats, 314–315
- reshaping data, xxviii

resizing

- arrays, 150
- large data sets, 320
- pivot table slicers, 269
- Properties window, 594
- Resize property, 59–60
- row/column ranges, 59–60
- sparkline cells, 423
- userforms, 177, 532
- windows, 39

resolution information, 548–549

Restore Down icon, 37

Resume Next, 66, 167, 557, 558, 560, 561, 563

revenue-based sorting, 257

RGB function, 417

ribbon code

- adding ribbon controls, 576–580
- case sensitivity of, 573, 586
- creating tabs/groups, 575–576
- customizing, 573, 574–575, 583
- custom ribbon errors, 586–590
- custom ribbon images, 584–585
- legacy Excel version compatibility, 575
- macro buttons on built-in ribbons, 590–591
- using Office icons in, 583–584
- XML as, 573

Row fields, 240

rows

- adding table rows, 81
- array, 143, 151, 165, 321
- criteria ranges on, 214
- declaring variables for, 84
- filtering, 366
- filters for visible, 204
- finding the last, 46–48, 50
- hidden by AutoFilter, 197
- highlighting highest-value row, 404
- imports over 1,048,576 rows, 459, 466, 468
- looping through, 56
- pivot table, 243
- properties of contiguous, 61
- R1C1 references to, 117, 123
- reading text files row by row, 466–468
- referring to ranges of, 58
- removing unwanted, 320
- resizing ranges by, 59
- variables in the last, 48
- working with visible, 204

Rows property, 61

Run Dialog button, 4

running timers, 551

Run-Time Error 9: Subscript out of range, 565–566

Run-Time Error 1004

- choices for resolving, 553
- delayed arrival of, 567–568
- error trap to anticipate, 224–225
- pivot table changes and, 248
- R1C1 reference style and, 125

runtime logic, 329

runtime userform controls, 530–532

S

safeData, 91

Save As for add-ins, 599–600

Save command, 485

saves

- API use and, 544
- data in add-in files, 598
- flaws in saved macros, 16–17
- macro-enabled files, 299
- new file saves, 5
- personal workbook, 9
- retrieving date/time of, 338
- userforms for file saves, 196
- Word document, 485

scheduling procedures, 453–458

scope of variables, 85

scrollbar sliders, 525–527

seaborn, xxviii

Search box, 201, 266

search filter, 266

security

- Excel add-in security, 602–603
- Excel encryption keys, 569
- macros concerns, 5–8
- Power Query and data privacy, 435
- protecting data from editing, 91
- recovery plans and, 569
- source code protections, 569
- Trusted Locations list, 6
- VBA code protections, 568–569
- VBA Extensibility additions and, 315

Select Case construct, 113–115, 176, 355

Select..Case statements in JavaScript, 623–625

Selection.Address, 39, 40, 41, 42

Selection property, 36, 38, 42

selections

selections

- Advanced Filter for, 204, 214
- AutoFilter for multi-item, 200
- Cells property to select ranges, 56–57
- code cleanup and, 45
- CurrentRegion property for, 64
- formula-based record, 216–224
- search box, 201
- selections in Word, 486–487
- SpecialCells, 65–67, 314
- See also Advanced Filter
- Selection.Value, 42, 107
- SELECT query, 501, 504, 511
- semicolons (;), 619
- SERIES formula, 323, 324
- server cursor, ADO, 501
- Set command, 29, 63, 86–88, 93
- SetElement method, 367–372
- settings sheets, 304–306
- SHA512 algorithm, 569, 570
- Shadow, 372
- shapes, attaching macros to, 594–595
- SharePoint, 297, 597
- SheetChange events, 129, 137, 303
- sheets. See worksheets
- shortcut keys
 - comments for, 14
 - Ctrl+A, 105
 - End+down arrow shortcut, 46–47
 - reusing, 9, 13
 - running macros via, 590
- .ShowAllItems, 262–263
- ShowDetail property, 286
- ShowPages, 240, 251, 257
- Show Report Filter, 257
- single-document interface (SDI), 574
- slicers
 - attaching pivot tables to existing, 275
 - choosing items in, 272
 - custom names for, 274–275
 - modifying existing, 274–275
 - named arguments, 270
 - pivot table filtration with, 269–274
 - slicer cache deletion, 275
 - slicer caches, 269–271, 275, 276
 - Timeline slicers, 276–278
- sliders, scrollbar, 525–527
- SLUGIFY.PLUS, 352–354
- SmartArt graphics, 383

Smith, Chris “Smitty”, 304

sorting

- concatenation and, 344–346
- custom sort orders, 310
- data, 234–237
- numeric and alpha characters, 346–348
- tables, 383

sounds, playing, 551–552

SourceType, 80

spaces in JavaScript, 618

sparklines

- best practices tips on, 423
- data visualization via, 383
- formatting, 413–422
- how to create, 408–410
- scaling, 410–413, 419
- theme colors, 413–417
- types of, 407

Speak On Enter, 459

SpecialCells, 47, 65–67, 224–225, 300, 314

speed

- arrays to increase, 141, 146
- data type and, 84
- dynamic arrays and, 151
- encryption key slowdowns, 569–570
- global variable initialization and, 166
- looping in memory and, 57
- macros, xxvii
- unformatted code, 146
- VBA and increased, 3, 233

spin buttons, 189–191, 525

Split function, 146

Spotify, 429, 430, 434

spreadsheets

- A1 styling for, 117, 120
- charts and, 357
- macro-running shapes in, 447
- matrices as, 143
- R1C1 style for, 118, 120
- Ready mode for, 454
- spreadsheet programs, 1–2

SQL queries, 506, 511

SQL Server, 503, 515–516

SQL statements, 502

SQL string, 504, 505

SSL security protocols, 608

startFromScratch, 575, 586

static Add-ins, 612

static cursors, ADO, 501

- Step clause, 103
- Stop Recording icon, 9, 14, 125
- storage
 - custom object, 323
 - database, 497
 - global variable, in Power Query, 449–450
 - macros, 9
 - Office Scripts script, 289
 - pivot table code, 239
 - storage in names vs. cells, 74
 - storing formulas in names, 73
 - UDF, 334
- streamlining
 - M query code, 434
 - recorded macros, 45
 - With statements for, 68, 96
 - web queries with VBA, 442–443
- String data type, 84
- strings
 - adding string names, 73–74
 - collection keys as, 164
 - connection strings, 503
 - converting, 54
 - delimited, 146
 - input box returns, 175, 176
 - JavaScript, 621
 - pivot cache, 242
 - reading names with, 78
 - referencing, 72
 - searching text for, 348
 - sparkline parameters as, 408
 - stored in names, 74
 - variables changed to numbers from, 620
- structures
 - accessing Excel file structure, 581
 - data sorting and, 234–237
 - JavaScript function structure, 618
 - protection of data, 91
- Style property, 360
- Sub procedure, 88, 93
- subroutines
 - advanced filters to initialize, 220
 - branching functions and, 353
 - defined, 93
 - function calls via, 559
 - main subs, 559, 560
 - values and, 154
- substitutions, 343–344

- subtotals, controlling pivot table, 245–246
- Sullivan, Jerry, 308
- SUM function, 21, 23, 282
- SUMIF formula, 209, 210
- summarized data, 326
- SWITCH function, 355–356

T

- tables
 - added on the fly, 514
 - added to Data Model, 279
 - adding fields to, 502
 - arrays in, 148
 - building multiplication, 123–124
 - chart creation from, 366
 - checking for existing, 511
 - conditional formatting in, 383
 - creating relationships between, 280
 - data exports to XML, 301
 - defined, 53
 - defined names and, 80
 - extracting values with Advanced Filter in, 207
 - filling arrays from, 145–146
 - finding the last row in, 48
 - importing daily data to, 320–321
 - ListObject additions, 80
 - pivot table code storage in, 239
 - positioning on the worksheet, 234
 - referencing, 72, 80
 - resetting formats for, 314–315
 - settings, 304
 - userform, 177
 - working with, 81–82
- TabOrientation property, 522
- TabStrip controls, 522–523
- tabular report layouts, 287
- tags, HTML, 615
- Task Manager, 477
- task pane, writing to, 628
- templates
 - chart, 358
 - class modules as, 154
 - generating reports via, 323
 - hidden, for reports, 229
 - UDF storage in, 334
- temporary variables, 167, 169
- test expressions, 113

testing

testing

- conditions, 111
- items in a collection, 167
- macros, 16, 19
- multiple conditions, 112
- Office Add-ins, 608
- using up free token quota in, 326

Test procedure, 88

text

- column data types and, 280
- control tip text, 537
- Enums as non-supportive of, 94
- fixed-width files, 459–463
- formatting cells with, 402
- highlighting duplicate text cells, 385
- input box data entry of, 176
- inserted in Word, 486
- numbers converted to, 452
- Office Add-in, 609
- pivot table additions of, 281
- rendering URLs from, 352, 353
- searching for strings in, 348
- sparkline, 423
- SpecialCells to select, 314
- specifying text fields, 462
- spin button controls excluding, 189
- Word reports with, 7

text box controls, 182–184

Text Edit mode, 593

text files

- add-in version numbers in, 604
- delimited, 459
- generated in Word, 327–328
- importing from, 459–470
- opening delimited files, 463–466
- Power Query to load large, 470
- reading into memory and parsing, 296–297
- reading one row at a time, 466–468
- row count and importing, 459, 466, 468
- writing, 471

Text format, 462

Text Import Wizard, 33, 34

Text to Columns, 297, 467

texture, color and, 373, 374

third-party system integration, 328

ThisWorkbook module, 127

ThousandsSeparator, 35

Timeline slicers, 276–278

timers, running, 551

times, grouping, 252–254

TOC2HTML, 354–355

ToggleButton controls, 525

tooltips

- adding controls on the fly and, 533
- compiler errors and missing, 155
- early binding for, 481
- late binding and, 476, 479
- Office Scripts, 634
- Worksheet object, 154

top/bottom value filtration

- AutoFilter for, 200–201
- data visualization options, 384
- formatting top/bottom cells, 399–400
- pivot table, 267–269

tracking user changes, 304

TrailingMinusNumbers parameter, 35

transparent userforms, 541–542

transposing arrays, 151, 321

Trim, 58, 64, 147

Trusted Locations list, 6, 604, 610

Tufte, Professor Edward, 407

two-dimensional arrays, 143, 149, 321

Type parameter, 65

Type property, 385, 486

TypeScript, xxvii, 631

U

underscore prefix, 69, 95, 610

uninitializing global variables, 166

Union method, 61, 63

Unique Records Only, 207, 225

Until clause, 108–109

Unviewable+, 568, 569

unzipping Excel files, 573

updates

- custom object, 168–169
- debugging database update code, 507
- Excel add-in, 602, 604
- live chart updates, 303
- Office Add-in, 617
- Office Scripts, 630
- scheduling Excel, 455
- self-updating workbooks, 511

upper bound (UBound), 142, 144, 148

Upstart Multiplan, 117

URL handling, 297–298

Urtis, Tom, 302, 310

- user-defined functions (UDFs)
 - added to add-ins, 598
 - creating, 331–333
 - JavaScript custom functions, 607
 - overview of useful, 334–350
 - productivity increase via, 331
 - sharing, 334
 - user-defined types (UDTs), 153, 172–174
 - Use Relative References setting, 22
 - userforms
 - Access calls with, 503
 - adding controls on the fly, 532–533
 - adding help to, 536–539
 - adding images to, 534
 - adding option buttons to, 187–188
 - adding properties to, 178
 - advanced programming, 175
 - basic, 177
 - basic control use, 181–193
 - benefits of, 175
 - calling/hiding, 178–179
 - combo/list boxes in, 184–185
 - complex criteria in, 217, 219
 - control collections, 527–529
 - controls toolbox, 178
 - creating, 175, 177–178
 - debugging, 181, 555–557
 - disabling the X for closing, 550
 - entry verifications, 193
 - graphics for, 188–189
 - Hide, 178
 - hyperlinks in, 529–530
 - illegal window closing, 193–194
 - InputBox function, 175–176
 - label, text box, and command buttons, 182–184
 - logic application to multiple, 527
 - Load, 178
 - Me, 178–179
 - modal and modeless, 529
 - MsgBox, 176–177
 - multipage, 177, 191–193
 - naming, 177, 179, 181
 - operating systems and, xxxii
 - populating list boxes on, 210–212
 - programming, 179–180
 - programming controls for, 180–181
 - resizing, 177
 - runtime controls, 530–532
 - Show, 178
 - sizing on the fly, 532, 533
 - tab ordering, 537
 - transparent, 541–542
 - Unload, 178
 - userform controls, 520–527
 - userform events, 128, 179–180
 - UserForm toolbar, 519–520
 - VBA Extensibility to export, 315
 - UserForm toolbar, 519–520
 - USERID function, 337
 - user interface
 - chart elements beyond, 358
 - customizing ribbon code, 574–575
 - extracting unique values with, 206–207
 - formula-based conditions in Excel, 218–219
 - Office Add-in, 617
 - removing menus in, 233–234
 - sounds in, 551–552
 - users
 - accidental worksheet deletion by, 565
 - add-ins installed by, 601–602
 - button/icon prompts for, 175, 176
 - configuring prompts for, 175–176
 - differing feature needs of, 328–329
 - error identification by, 564
 - illegal window closing by, 193–194
 - information requests from, 182–183
 - multiuser database access, 500–501
 - Office Scripts availability for, 631
 - tracking changes by, 304
 - userform controls in response to, 181
 - userforms to interact with, 175, 177–178
 - user ID retrieval, 337–338
 - verifying entry from, 193
 - Windows API access for, 543
 - workbook rights for, 337
- ## V
- validating email addresses, 339–341
 - Value Filters, 263
 - value-for-value substitutions, 343–344
 - Value parameter, 65
 - Value property, 40, 146, 281–282
 - values
 - assigned to properties, 157
 - AutoFilter for top/bottom, 200–201
 - chart cell, 359
 - checking collections for, 167

values

- combo/list boxes for, 184
- constants to lock down, 86
- converting names to, 54
- converting pivot table to, 248–251
- counting values in fields, 252
- custom function, 332
- data bar min/max, 387, 389
- default property, 477
- duplicate maximum value returns, 348–349
- extracted from arrays, 149
- extracted from listobjects, 208
- extracted via Advanced Filter, 205–213
- formatting cells based on, 401–402
- highlighting duplicate, 384, 385, 400–401
- highlighting first unique, 403
- highlighting highest-value row, 404
- icon set, 394
- intersection of pivot table, 278
- keys to check for, 169
- Let function to assign, 351
- names to store, 74
- numeric, 56
- passed to procedures, 128
- percentage calculations, 254–256
- pivot table value filters, 264
- reading name, 78
- replaced with formula-based conditions, 216–224
- scrollbar sliders to select, 525–527
- string, 73
- texture, 374
- updating collection, 168
- user-defined, 172, 331
- using constant, 67
- value queries, 39
- variables as remembering, 48
- variables to hold, 83
- Watches window, 41
- Word-specific constant, 479–481
- Values area, 286
- variables
 - arrays and, 141, 142
 - assigning, 83–85
 - caution when using, 102
 - constants and, 86
 - counter variables, 100, 103
 - data types, 84
 - declaration of, 11, 52, 83–85, 619
 - declared as collection, 163
 - declaring Word, 475
 - definition of, 83
 - global, in M language, 449–453
 - JavaScript, 619–621
 - lifetime and scope of, 85
 - naming, 48, 79
 - non-recording of, 39, 48
 - object variables, 86–88, 110, 208
 - private, 156, 157
 - specifying parameter, 57, 88–89
 - For statement, 102
 - string, 242
 - temporary, 167, 169
 - uninitializing global, 166
 - value queries, 39
 - watching value changes, 42
- variations, managing, 91–92
- VBA (Visual Basic for Applications)
 - Advanced Filter in, 197, 204
 - arrays in, 142
 - automation via, xxxi, 3, 327
 - BASIC vs., 25
 - charts in, 357, 361–363, 382
 - cloud saving by, 486
 - code protections in, 568–569
 - combo charts in, 377
 - consistency from, 23
 - database access libraries, 498
 - data visualization in, 383, 385, 396
 - debugging tools, 36–43
 - declaring variables, 619
 - defined constants, 33–35
 - drill down queries in, 286
 - For Each loop, 110
 - errors in, 553
 - Excel Online and, 630
 - extracting unique values with, 207–212
 - Filter In Place in, 224
 - formula-based conditions with, 219–223
 - generalizing queries with, 442–448
 - Help files, 30
 - If logic in, 450–451
 - increased productivity via, 1, 3, 233
 - LAMBDA functions and, 350
 - learning barriers, 1
 - loop support by, 99
 - manual test of, xxx

- M equivalents of, 449
- Office Add-ins and, 607
- pivot table use in, 239, 240–251
- Power Query syntax in, 470
- pro techniques for, 306–319
- R1C1 style and, 117
- reading/writing from text files, 459
- relative references in, 72
- RGB function, 418
- search filter, 266
- Select Case construct, 355
- sparkline formatting in, 413
- SQL string creation, 505
- strings in, 621
- syntax of, xxviii, 1, 26–30, 31
- table relationships in, 280
- threats to, xxvii–xxviii
- turning off menus in, 233–234
- TypeScript and, xxvii
- understanding language of, 26–30
- unwanted row removal in, 320
- VB as foundation for, 631
- wild cards, 201
- Windows/Mac differences, xxxii
- VBA Extensibility, 315–316
- VBA Function procedures, 331
- VB Editor
 - adding object libraries through, 474
 - code review in, 19, 31–32
 - constants in, 86
 - converting files to add-ins with, 600
 - database access via, 499
 - debugging tools, 36–43
 - Immediate window, 39–40, 477
 - Mac vs. Windows, xxxii
 - new modules in, 332
 - Object Browser, 43–44
 - opening, 14
 - overview of, 10–11
 - Properties window in, 594
 - settings, 11
 - userform controls, 175
 - userform insertion in, 177
 - UserForm toolbar, 519–520
 - Word object model in, 484
 - XML export setup in, 301
- verbal reminder scheduling, 456–457
- videoconferencing, 564
- View Code, 11, 14

- visible vs. hidden pivot table fields, 261–262
- visible vs. hidden rows, 204, 261–262
- VisiCalc, 117, 120
- Visual Basic .NET, 597
- Visual C++, 597
- visuals, custom, xxviii

W

- W3Schools.com, 511
- Watches window
 - arrays in, 143–144, 145
 - breakpoint setting via, 42
 - constant value retrieval with, 480
 - objects in, 42–43
 - querying via, 41–42
 - usefulness of, 40
- waterfall charts, 381–382
- web connectivity, 608
- web queries
 - access vs. refresh tokens, 434
 - credential refreshes, 436, 439
 - dashboard arrangement of, 445–448
 - duplicating existing queries, 440–441
 - grouping, 444
 - Let expression for, 436
 - parameters for, 434
 - Power Query for, 429, 432–434
 - starting new, 437
 - streamlining, 442–443
 - VBA to generalize, 442–448
- While clause, 108–109
- While...Wend loops, 109
- wild cards, 201
- Windows
 - control buttons, 592
 - Excel versions for, xxxii
 - functionality via API, 543–552
 - Notepad, 574, 608
 - platform compatibility checks, 571
 - See also API, Windows; Microsoft
- windows
 - arrangement of, 37
 - maximized vs. floating, 44
 - resizing, 39
- win/loss charts, 407, 421–422, 425
- With...End With
 - commas/indentation for, 97
 - names with, 49

- nested, 96, 97
- streamlining code with, 68, 96
- With statements
 - creating Sort objects with, 235
 - streamlining code with, 49, 96
- Word
 - automating, 473–495
 - bookmarks in, 327
 - constants specific to, 479
 - controlling form fields in, 492–495
 - CreateObject for, 478
 - documents, 484–486
 - Excel access to, 473
 - generating reports via, 327–328
 - GetObject to reference, 478–479
 - icon galleries in, 584
 - multiple instances of, 477
 - New keyword to reference, 477
 - objects, 484–492
 - opening documents in, 485
 - selections in, 486–487
 - Word-specific constants, 479–481
- Word 365 object library, 474, 476
- WordPad, 574
- Workbook_Open event, 127, 129, 135, 136, 316, 512, 597, 604, 605
- workbooks
 - checking openness of, 334–335
 - combining/separating, 298–302
 - converted to add-ins, 598–599
 - copying existing charts into, 323–326
 - detecting cloud-hosted, 297–298
 - editing, 12
 - functions to open, 559
 - global names, 69
 - hidden, as add-in alternative, 605
 - hierarchy of objects and, 54
 - list of open, 11
 - macro-enabled, 5, 8
 - modifying, xxviii
 - multiple users for, 328–329
 - opening, 582–583
 - recording macros in, 13
 - referencing ranges in, 68
 - retrieving date/time of saves, 338
 - ribbon tab visibility for, 574
 - sheet searches in, 335–336
 - single-worksheet, 230
 - templates for, 323
 - UDF storage in, 334
 - user rights to, 337
 - VBA Extensibility to add code to, 315–316
 - workbook events, 127, 129–131
 - workbook-level sheet events, 127, 131–132
- Workbooks object, 473
- Workbooks.OpenText method, 31, 459, 464, 466
- Worksheet object, 154
- worksheets
 - accidental deletion of, 565
 - adding, 29
 - chart sheets, 357, 359
 - checking workbooks for, 335–336, 563
 - CodeName vs. sheet names, 566–567
 - copying data to separate, 300–301
 - criteria ranges on, 214
 - editing, 12
 - Enum for heading, 95
 - filling arrays from, 145–146
 - form controls for, 592
 - hierarchy of objects and, 54
 - large row count, 469
 - listobject positioning on, 234
 - local names, 69
 - named ranges in, 55
 - Office Add-in writing to/reading from, 629–630
 - properties/methods in, 154
 - Range objects as property of, 54, 55
 - referencing ranges in, 68
 - separated into workbooks, 298–299
 - settings sheets, 304–306
 - sparklines on, 423
 - UDF use on, 331, 332
 - worksheet events, 127, 129, 131, 132–134
 - Worksheet.Sort for data sorting, 234, 235–236
- Worksheets collection, 163
- Worksheet.Sort, 234, 235–236
- writing text files, 471

X

- xIExpression, 402, 403
- xIFilterCopy, 225–229
- XLM macro language, xxvii
- .xlsb file (macro-enabled), 5, 299
- XML
 - adding ribbon controls, 576–580
 - case sensitivity of, 573, 574, 586, 617