

Microsoft Excel VBA and Macros

(Office 2021 and Microsoft 365)

Bill Jelen and Tracy Syrstad



Data sets and code files
on the web

FREE SAMPLE CHAPTER

SHARE WITH OTHERS



Microsoft Excel VBA and Macros (Office 2021 and Microsoft 365)

Bill Jelen
Tracy Syrstad

Microsoft Excel VBA and Macros (Office 2021 and Microsoft 365)

Published with the authorization of Microsoft Corporation by:
Pearson Education, Inc.
Copyright © 2022 by Pearson Education, Inc.

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/permissions.

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: 978-0-13-752152-4
ISBN-10: 0-13-752152-9

Library of Congress Control Number: 2022930486
ScoutAutomatedPrintCode

Trademarks

Microsoft and the trademarks listed at <http://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 are © 2022 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.

Special Sales

For information about buying this title in bulk quantities, or for special sales opportunities (which may include electronic versions; custom cover designs; and content particular to your business, training goals, marketing focus, or branding interests), please contact our corporate sales department at corpsales@pearsoned.com or (800) 382-3419.

For government sales inquiries, please contact governmentsales@pearsoned.com.

For questions about sales outside the U.S., please contact intlcs@pearson.com.

EDITOR-IN-CHIEF

Brett Bartow

EXECUTIVE EDITOR

Loretta Yates

SPONSORING EDITOR

Charvi Arora

DEVELOPMENT EDITOR

Songlin Qiu

MANAGING EDITOR

Sandra Schroeder

SENIOR PROJECT EDITOR

Tracey Croom

COPY EDITOR

Sarah Kearns

INDEXER

Timothy Wright

PROOFREADER

Donna E. Mulder

TECHNICAL EDITOR

Bob Umlas

COVER DESIGNER

Twist Creative, Seattle

COMPOSITOR

codeMantra

Pearson's Commitment to Diversity, Equity, and Inclusion

Pearson is dedicated to creating bias-free content that reflects the diversity of all learners. We embrace the many dimensions of diversity, including but not limited to race, ethnicity, gender, socioeconomic status, ability, age, sexual orientation, and religious or political beliefs.

Education is a powerful force for equity and change in our world. It has the potential to deliver opportunities that improve lives and enable economic mobility. As we work with authors to create content for every product and service, we acknowledge our responsibility to demonstrate inclusivity and incorporate diverse scholarship so that everyone can achieve their potential through learning. As the world's leading learning company, we have a duty to help drive change and live up to our purpose to help more people create a better life for themselves and to create a better world.

Our ambition is to purposefully contribute to a world where:

- Everyone has an equitable and lifelong opportunity to succeed through learning.
- Our educational products and services are inclusive and represent the rich diversity of learners.
- Our educational content accurately reflects the histories and experiences of the learners we serve.
- Our educational content prompts deeper discussions with learners and motivates them to expand their own learning (and worldview).

While we work hard to present unbiased content, we want to hear from you about any concerns or needs with this Pearson product so that we can investigate and address them.

- Please contact us with concerns about any potential bias at <https://www.pearson.com/report-bias.html>.

This page intentionally left blank

Dedication

*For Skipper Geanangel, Patricia Garick, Jim Lantz, Robert Mucci,
Bill & Jean Esposito. Thanks for launching a writing career.*

—Bill Jelen

To John. Giraffe.

—Tracy Syrstad

Contents at a Glance

	Acknowledgments	xxv
	About the Authors	xxvii
	Introduction	xxix
CHAPTER 1	Unleashing the power of Excel with VBA	1
CHAPTER 2	This sounds like BASIC, so why doesn't it look familiar?	27
CHAPTER 3	Referring to ranges	53
CHAPTER 4	Looping and flow control	69
CHAPTER 5	R1C1-style formulas	87
CHAPTER 6	Creating and manipulating names in VBA	97
CHAPTER 7	Event programming	111
CHAPTER 8	Arrays	125
CHAPTER 9	Creating classes and collections	133
CHAPTER 10	Userforms: An introduction	153
CHAPTER 11	Data mining with Advanced Filter	175
CHAPTER 12	Using VBA to create pivot tables	211
CHAPTER 13	Excel power	257
CHAPTER 14	Sample user-defined functions	285
CHAPTER 15	Creating charts	313
CHAPTER 16	Data visualizations and conditional formatting	339
CHAPTER 17	Dashboarding with sparklines in Excel	363
CHAPTER 18	Reading from the web using M and VBA	385
CHAPTER 19	Text file processing	413
CHAPTER 20	Automating Word	427
CHAPTER 21	Using Access as a back end to enhance multiuser access to data	447
CHAPTER 22	Advanced userform techniques	465
CHAPTER 23	The Windows Application Programming Interface (API)	491
CHAPTER 24	Handling errors	501
CHAPTER 25	Customizing the ribbon to run macros	517
CHAPTER 26	Creating Excel add-ins	539
CHAPTER 27	An introduction to creating Office add-ins	549
CHAPTER 28	What's new in Excel 365 and what's changed	571
	Index	579

Contents

<i>Acknowledgments</i>	xxv
<i>About the Authors</i>	xxvii
<i>Introduction</i>	xxix
Chapter 1 Unleashing the power of Excel with VBA	1
Barriers to entry	1
The macro recorder doesn't work!	2
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 box	9
Running a macro	10
Creating a macro button on the ribbon	10
Creating a macro button on the Quick Access Toolbar	11
Assigning a macro to a form control, text box, or shape.	12
Understanding the VB Editor	13
VB Editor settings.	14
The Project Explorer	14
The Properties window	15

Understanding shortcomings of the macro recorder	15
Recording the macro	17
Examining code in the Programming window	17
Running the macro on another day produces undesired results	19
Possible solution: Use relative references when recording.....	20
Never use AutoSum or Quick Analysis while recording a macro.....	24
Four tips for using the macro recorder	25
Next steps	26

Chapter 2 This sounds like BASIC, so why doesn't it look familiar? 27

Understanding the parts of VBA "speech"	28
VBA is not really hard	32
VBA Help files: Using F1 to find anything	32
Using Help topics.....	32
Examining recorded macro code: Using the VB Editor and Help	33
Optional parameters.....	34
Defined constants	35
Properties can return objects.....	38
Using debugging tools to figure out recorded code	38
Stepping through code	38
More debugging options: Breakpoints.....	40
Backing up or moving forward in code.....	40
Not stepping through each line of code.....	41
Querying anything while stepping through code.....	41
Using a watch to set a breakpoint.....	43
Using a watch on an object.....	44
Object Browser: The ultimate reference.....	45
Seven tips for cleaning up recorded code	45
Tip 1: Don't select anything.....	46
Tip 2: Use Cells(2, 5) because it's more convenient than Range("E2").....	47
Tip 3: Use more reliable ways to find the last row	47
Tip 4: Use variables to avoid hard-coding rows and formulas.....	49

Tip 5: Use R1C1 formulas that make your life easier.	49
Tip 6: Copy and paste in a single statement	49
Tip 7: Use With . . . End With to perform multiple actions	50
Next steps	52
Chapter 3 Referring to ranges	53
The Range object	54
Syntax for specifying a range.	54
Referencing named ranges.	55
Shortcut for referencing ranges	55
Referencing ranges in other sheets	55
Referencing a range relative to another range.	56
Using the Cells property to select a range.	57
Using the Offset property to refer to a range	58
Using the Resize property to change the size of a range.	60
Using the Columns and Rows properties to specify a range.	61
Using the Union method to join multiple ranges.	62
Using the Intersect method to create a new range from overlapping ranges.	62
Using the IsEmpty function to check whether a cell is empty	62
Using the CurrentRegion property to select a data range	63
Using the Areas collection to return a noncontiguous range	66
Referencing tables.	67
Next steps	68
Chapter 4 Looping and flow control	69
For . . . Next loops.	69
Using variables in the For statement.	72
Variations on the For . Next loop	72
Exiting a loop early after a condition is met	73
Nesting one loop inside another loop.	74
Do loops.	75
Using the While or Until clause in Do loops	77

The VBA loop: For Each.....	79
Object variables	79
Flow control: Using If...Then...Else and Select Case.....	81
Basic flow control: If...Then...Else.....	81
Using Select Case...End Select for multiple conditions	83
Next steps	86
Chapter 5 R1C1-style formulas	87
Toggling to R1C1-style references	88
Witnessing the miracle of Excel formulas.....	89
Entering a formula once and copying 1,000 times	89
The secret: It's not that amazing	90
Understanding the R1C1 reference style.....	91
Using R1C1 with relative references	91
Using R1C1 with absolute references	92
Using R1C1 with mixed references	93
Referring to entire columns or rows with R1C1 style.....	93
Replacing many A1 formulas with a single R1C1 formula.....	94
Remembering column numbers associated with column letters.....	95
Next steps	96
Chapter 6 Creating and manipulating names in VBA	97
Global versus local names.....	97
Adding names.....	98
Deleting names	100
Adding comments.....	100
Types of names.....	101
Formulas.....	101
Strings	101
Numbers.....	103
Tables.....	103
Using arrays in names.....	104
Reserved names	104
Hiding names	106

	Checking for the existence of a name	106
	Next steps	109
Chapter 7	Event programming	111
	Levels of events	111
	Using events	112
	Event parameters	112
	Enabling events	113
	Workbook events	113
	Workbook-level sheet events	115
	Worksheet events	116
	Chart events	118
	Embedded charts	118
	Embedded chart and chart sheet events	119
	Application-level events	120
	Next steps	124
Chapter 8	Arrays	125
	Declaring an array	125
	Declaring a multidimensional array	126
	Filling an array	127
	Retrieving data from an array	128
	Using arrays to speed up code	129
	Using dynamic arrays	130
	Passing an array	131
	Next steps	132
Chapter 9	Creating classes and collections	133
	Inserting a class module	133
	Trapping application and embedded chart events	134
	Application events	134
	Embedded chart events	136
	Creating a custom object	137
	Using a custom object	139

Using collections	140
Creating a collection	140
Creating a collection in a standard module	141
Creating a collection in a class module	142
Using dictionaries	145
Using user-defined types to create custom properties	148
Next steps	152
Chapter 10 Userforms: An introduction	153
Input boxes	153
Message boxes	154
Creating a userform	155
Calling and hiding a userform	156
Programming userforms	157
Userform events	157
Programming controls	158
Using basic form controls	159
Using labels, text boxes, and command buttons	159
Deciding whether to use list boxes or combo boxes in forms	162
Using the MultiSelect property of a list box	163
Adding option buttons to a userform	165
Adding graphics to a userform	167
Using a spin button on a userform	168
Using the MultiPage control to combine forms	169
Verifying field entry	171
Illegal window closing	172
Getting a file name	173
Next steps	174
Chapter 11 Data mining with Advanced Filter	175
Replacing a loop with AutoFilter	175
Using AutoFilter techniques	178
Selecting visible cells only	181

Advanced Filter—easier in VBA than in Excel	183
Using the Excel interface to build an advanced filter	184
Using Advanced Filter to extract a unique list of values	184
Extracting a unique list of values with the user interface	185
Extracting a unique list of values with VBA code	186
Getting unique combinations of two or more fields.	190
Using Advanced Filter with criteria ranges	191
Joining multiple criteria with a logical OR	192
Joining two criteria with a logical AND	193
Other slightly complex criteria ranges	193
The most complex criteria: Replacing the list of values with a condition created as the result of a formula.	193
Setting up a condition using computed criteria.	194
Using Filter In Place in Advanced Filter	201
Catching no records when using a filter in place	201
Showing all records after running a filter in place.	202
The real workhorse: x1FilterCopy with all records rather than unique records only	202
Copying all columns	202
Copying a subset of columns and reordering.	203
Excel in practice: Turning off a few drop-down menus in the AutoFilter	209
Next steps	210

Chapter 12 Using VBA to create pivot tables 211

Understanding how pivot tables evolved over various Excel versions	211
Building a pivot table in Excel VBA.	212
Defining the pivot cache	213
Creating and configuring the pivot table.	213
Adding fields to the data area	214
Learning why you cannot move or change part of a pivot report	217
Determining the size of a finished pivot table to convert the pivot table to values	217

Using advanced pivot table features	220
Using multiple value fields	220
Grouping daily dates to months, quarters, or years	221
Changing the calculation to show percentages	223
Eliminating blank cells in the Values area	226
Controlling the sort order with AutoSort	226
Replicating the report for every product	226
Filtering a data set	229
Manually filtering two or more items in a pivot field	229
Using the conceptual filters	230
Using the search filter	234
Setting up slicers to filter a pivot table	237
Setting up a timeline to filter an Excel pivot table	241
Formatting the intersection of values in a pivot table	243
Using the Data Model in Excel	244
Adding both tables to the Data Model	244
Creating a relationship between the two tables	245
Defining the pivot cache and building the pivot table	245
Adding model fields to the pivot table	246
Adding numeric fields to the Values area	246
Putting it all together	247
Using other pivot table features	249
Calculated data fields	249
Calculated items	250
Using ShowDetail to filter a record set	250
Changing the layout from the Design tab	250
Settings for the report layout	251
Suppressing subtotals for multiple row fields	252
Comparing VBA to TypeScript	253
Next steps	256
Chapter 13 Excel power	257
File operations	257
Listing files in a directory	257
Importing and deleting a CSV file	260
Reading a text file into memory and parsing	260

Combining and separating workbooks	261
Separating worksheets into workbooks	261
Combining workbooks	262
Copying data to separate worksheets without using Filter	263
Exporting data to an XML file	264
Placing a chart in a cell note	265
Tracking user changes	267
Techniques for VBA pros	268
Creating an Excel state class module	268
Drilling-down a pivot table	270
Filtering an OLAP pivot table by a list of items	271
Creating a custom sort order	273
Creating a cell progress indicator	274
Using a protected password box	275
Selecting with SpecialCells	277
Resetting a table's format	278
Using VBA Extensibility to add code to new workbooks	279
Converting a fixed-width report to a data set	280
Next steps	284

Chapter 14 Sample user-defined functions **285**

Creating user-defined functions	285
Building a simple custom function	286
Sharing UDFs	288
Useful custom Excel functions	288
Checking whether a workbook is open	288
Checking whether a sheet in an open workbook exists	289
Counting the number of workbooks in a directory	290
Retrieving the user ID	291
Retrieving date and time of last save	292
Retrieving permanent date and time	292
Validating an email address	293
Summing cells based on interior color	295
Counting unique values	296
Finding the first nonzero-length cell in a range	296

Substituting multiple characters	297
Retrieving numbers from mixed text	298
Converting week number into date	299
Sorting and concatenating	300
Sorting numeric and alpha characters	301
Searching for a string within text	303
Returning the addresses of duplicate maximum values	304
Returning a hyperlink address	305
Returning the column letter of a cell address	305
Using Select...Case on a worksheet	306
Creating LAMBDA functions	307
Building a simple LAMBDA function	307
Sharing LAMBDA functions	308
Useful LAMBDA functions	309
Next steps	311

Chapter 15 Creating charts 313

Using .AddChart2 to create a chart	314
Understanding chart styles	315
Formatting a chart	318
Referring to a specific chart	318
Specifying a chart title	319
Applying a chart color	320
Filtering a chart	322
Using SetElement to emulate changes from the plus icon	322
Using the Format tab to micromanage formatting options	327
Changing an object's fill	328
Formatting line settings	331
Creating a combo chart	331
Creating map charts	335
Creating waterfall charts	336
Exporting a chart as a graphic	337
Considering backward compatibility	337
Next steps	338

Chapter 16	Data visualizations and conditional formatting	339
	VBA methods and properties for data visualizations	340
	Adding data bars to a range.	342
	Adding color scales to a range	346
	Adding icon sets to a range	347
	Specifying an icon set.	348
	Specifying ranges for each icon	350
	Using visualization tricks	350
	Creating an icon set for a subset of a range.	351
	Using two colors of data bars in a range	352
	Using other conditional formatting methods.	355
	Formatting cells that are above or below average	355
	Formatting cells in the top 10 or bottom 5	355
	Formatting unique or duplicate cells.	356
	Formatting cells based on their value	358
	Formatting cells that contain text.	358
	Formatting cells that contain dates	359
	Formatting cells that contain blanks or errors	359
	Using a formula to determine which cells to format.	359
	Using the new NumberFormat property	361
	Next steps	362
Chapter 17	Dashboarding with sparklines in Excel	363
	Creating sparklines	363
	Scaling sparklines.	366
	Formatting sparklines.	369
	Using theme colors	369
	Using RGB colors	373
	Formatting sparkline elements	374
	Formatting win/loss charts.	377
	Creating a dashboard.	378
	Observations about sparklines	379
	Creating hundreds of individual sparklines in a dashboard	379
	Next steps	383

Chapter 18 Reading from the web using M and VBA	385
Get credentials for accessing an API	386
Build a query in Power Query using the M language to retrieve data from the web for one specific value	387
Refreshing the credentials after they expire	390
Building a custom function in Power Query	390
Using the new function in your code	393
Duplicating an existing query to make a new query.	393
Querying the list of songs on an album	395
Generalizing the queries using VBA	396
Simplifying the SearchArtist query to a single line of code	396
Simplifying the ArtistAlbums query	396
Simplifying the AlbumTracks query	397
Grouping queries to clean up the queries list	397
Planning the arrangement of query results on your dashboard	398
Using global variables and loops in M	402
Storing global variables in a Settings record in Power Query ...	402
Simple error handling using try with otherwise	403
Using If logic in M	403
Looping using List.Generate	404
Application.OnTime to periodically analyze data	406
Using Ready mode for scheduled procedures	407
Specifying a window of time for an update	407
Canceling a previously scheduled macro	408
Closing Excel cancels all pending scheduled macros	408
Scheduling a macro to run <i>x</i> minutes in the future	408
Scheduling a verbal reminder	409
Scheduling a macro to run every two minutes	410
Next steps	411
 Chapter 19 Text file processing	 413
Importing from text files	413
Importing text files with fewer than 1,048,576 rows	413
Dealing with text files with more than 1,048,576 rows	419

Writing text files.....	424
Next steps	425
Chapter 20 Automating Word	427
Using early binding to reference a Word object	427
Using late binding to reference a Word object	430
Using the New keyword to reference the Word application	430
Using the CreateObject function to create a new instance of an object	431
Using the GetObject function to reference an existing instance of Word.....	431
Using constant values.....	433
Using the Watches window to retrieve the real value of a constant.....	433
Using the Object Browser to retrieve the real value of a constant	433
Understanding Word's objects	435
The Document object	435
Controlling form fields in Word.....	443
Next steps	445
Chapter 21 Using Access as a back end to enhance multiuser access to data	447
ADO versus DAO	448
The tools of ADO	450
Adding a record to a database	452
Retrieving records from a database.....	453
Updating an existing record.....	455
Deleting records via ADO	458
Summarizing records via ADO.....	458
Other utilities via ADO.....	459
Checking for the existence of tables	460
Checking for the existence of a field	461
Adding a table on the fly.....	461
Adding a field on the fly.....	462

SQL Server examples	463
Next steps	464
Chapter 22 Advanced userform techniques	465
Using the UserForm toolbar in the design of controls on userforms.....	465
More userform controls.....	466
CheckBox controls	466
TabStrip controls	468
RefEdit controls.....	470
ToggleButton controls	471
Using a scrollbar as a slider to select values.....	472
Controls and collections	473
Modeless userforms	475
Using hyperlinks in userforms	476
Adding controls at runtime.....	477
Resizing the userform on the fly	479
Adding a control on the fly.....	479
Sizing on the fly	479
Adding other controls	480
Adding an image on the fly	480
Putting it all together	481
Adding help to a userform	483
Showing accelerator keys	483
Adding control tip text.....	484
Creating the tab order	484
Coloring the active control.....	485
Creating transparent forms	487
Next steps	489
Chapter 23 The Windows Application Programming Interface (API)	491
Understanding an API declaration.....	492
Using an API declaration	493
Making 32-bit- and 64-bit-compatible API declarations	493

API function examples	494
Retrieving the computer name	495
Checking whether an Excel file is open on a network	495
Retrieving display-resolution information	496
Customizing the About dialog box	497
Disabling the X for closing a userform	498
Creating a running timer	498
Playing sounds	499
Next steps	500
Chapter 24 Handling errors	501
What happens when an error occurs?	501
A misleading Debug error in userform code	503
Basic error handling with the On Error GoTo syntax	505
Generic error handlers	506
Handling errors by choosing to ignore them	506
Suppressing Excel warnings	508
Encountering errors on purpose	509
Training your clients	509
Errors that won't show up in Debug mode	510
Errors while developing versus errors months later	510
Runtime error 9: Subscript out of range	511
Runtime error 1004: Method range of object global failed	512
The ills of protecting code	513
More problems with passwords	515
Errors caused by different versions	515
Next steps	516
Chapter 25 Customizing the ribbon to run macros	517
Where to add code: The customui folder and file	518
Creating a tab and a group	519
Adding a control to a ribbon	520
Accessing the file structure	525
Understanding the RELS file	525

Renaming an Excel file and opening a workbook	526
Using images on buttons	527
Using Microsoft Office icons on a ribbon.....	527
Adding custom icon images to a ribbon.....	528
Troubleshooting error messages	529
The attribute " <i>Attribute Name</i> " on the element " <i>customui ribbon</i> " is not defined in the DTD/schema	530
Illegal qualified name character	530
Element " <i>customui Tag Name</i> " is unexpected according to content model of parent element " <i>customui Tag Name</i> "....	531
Found a problem with some content	531
Wrong number of arguments or invalid property assignment.....	532
Invalid file format or file extension.....	533
Nothing happens.....	533
Other ways to run a macro	533
Using a keyboard shortcut to run a macro.....	533
Attaching a macro to a command button	534
Attaching a macro to a shape	535
Attaching a macro to an ActiveX control	536
Running a macro from a hyperlink.....	537
Next steps	537

Chapter 26 Creating Excel add-ins **539**

Characteristics of standard add-ins	539
Converting an Excel workbook to an add-in.....	540
Using Save As to convert a file to an add-in	541
Using the VB Editor to convert a file to an add-in	542
Having a client install an add-in	543
Add-in security.....	544
Closing add-ins.....	545
Removing add-ins	545
Using a hidden workbook as an alternative to an add-in	545
Next steps	547

Chapter 27 An introduction to creating Office add-ins	549
Creating your first Office add-in—Hello World	550
Adding interactivity to an Office add-in	554
A basic introduction to HTML	557
Using tags	557
Adding buttons	557
Using CSS files	558
Using XML to define an Office add-in	558
Using JavaScript to add interactivity to an Office add-in	559
The structure of a function	560
Curly braces and spaces	560
Semicolons and line breaks	560
Comments	561
Variables	561
Strings	562
Arrays	562
JavaScript for loops	563
How to do an if statement in JavaScript	564
How to do a Select . . Case statement in JavaScript	564
How to use a For each . . next statement in JavaScript	565
Mathematical, logical, and assignment operators	566
Math functions in JavaScript	567
Writing to the content pane or task pane	569
JavaScript changes for working in an Office add-in	569
Next steps	570
Chapter 28 What's new in Excel 365 and what's changed	571
Office 365 subscription versus Excel 2021 perpetual	571
If it has changed in the front end, it has changed in VBA	571
The ribbon	572
Single-document interface	572
Modern array formulas	573
LAMBDA function	573
Quick Analysis tool	573
Charts	574

Pivot tables.....	574
Slicers.....	574
Icons.....	575
3D models.....	575
SmartArt.....	575
TypeScript.....	576
Learning the new objects and methods.....	576
Compatibility mode.....	576
Using the Version property.....	577
Using the Excel8CompatibilityMode property.....	577
Next steps.....	578
<i>Index</i>	579

Acknowledgments

Thanks to Tracy Syrstad for being a great coauthor.

Bob Umlas is the smartest Excel guy I know and is an awesome technical editor. At Pearson, Loretta Yates is an excellent acquisitions editor. Thanks to the Kughens for guiding this book through production. I updated this edition in residence at the Kola Mi Writing Camp. My sincere thanks to the staff there for keeping me on track.

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

Thank you to all the moderators at the MrExcel forum who keep the board organized, despite the best efforts of the spammers. Thank you to Joe4, RoryA, and Petersss for helping process all the forum's contact emails.

Programming is a constant learning experience, and I really appreciate the clients who have encouraged me to program outside my comfort zone so that my skills and knowledge have expanded. Thank you to Suat Özgür for helping me defeat some truly insidious programming puzzles.

Final Fantasy XIV has become my second home. I'd like to give a special thank you to my in-game friends who not only make gaming so much fun, but for also helping me find the confidence to dive head first into the unknown: War, Chraz, and Shabadoo. Thank you for sharing your love of gaming with me.

And last, but not least, thanks to Bill Jelen. His site, MrExcel.com, is a place where thousands come for help. It's also a place where I, and others like me, have an opportunity to learn from and assist others.

—Tracy

This page intentionally left blank

About the Authors



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 2,300 episodes of his daily video podcast, *Learn Excel from MrExcel*. He is the author of 65 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.

Tracy Syrstad is a Microsoft Excel developer and author of ten Excel books. She has been helping people with Microsoft Office issues since 1997, when she discovered free online forums where anyone could ask and answer questions. Tracy found out she enjoyed teaching others new skills, and when she began working as a developer, she was able to integrate the fun of teaching with one-on-one online desktop sharing sessions. Tracy lives on an acreage in eastern South Dakota with her husband, two cats, two horses, and a variety of wild foxes, squirrels, and rabbits.

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.
- Find out where to find 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 new 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 is perhaps 1% of the time where I will open an Excel workbook on an iPad. 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. Ozgur'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 tab 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. I have 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).

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," cracks the code on how to work efficiently with ranges and cells.

Chapter 4, "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 5, "R1C1-style formulas," covers, obviously, R1C1-style formulas. Chapter 6, "Creating and manipulating names in VBA," covers names. Chapter 7, "Event programming," includes some great tricks that use event programming. Chapter 8, "Arrays," covers arrays. Chapter 9, "Creating classes and collections," covers classes and collections. 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.

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 TypeScript application within Excel. Chapter 28, “What’s new in Excel 365 and what’s changed,” summarizes the changes in Excel 365.

Does this book teach Excel?

Microsoft believes that the ordinary Office customer touches only 10% of the features in Office. We realize that everyone reading this book is above average, and the visitors to MrExcel.com are a pretty smart audience. Even so, a poll of 8,000 MrExcel.com readers showed that only 42% of smarter-than-average users are using any 1 of the top 10 power features in Excel.

Bill regularly presents a Power Excel seminar for accountants. These are hard-core Excelers who use Excel 30 to 40 hours every week. Even so, two things come out in every seminar. First, half of the audience gasps when they see how quickly you can do tasks with a particular feature, such as automatic subtotals or pivot tables. Second, someone in the audience routinely trumps me. For example, someone asks a question, I answer, and someone in the second row raises a hand to give a better answer.

The point? Both the authors and the audience of this book know a lot about Excel. However, we assume that in any given chapter, maybe 58% of the people have not used pivot tables before and maybe even fewer have used the Top 10 Filter feature of pivot tables. With this in mind, before we show how to automate something in VBA, we briefly cover how to do the same task in the Excel interface. This book does not teach you how to make pivot tables, but it does alert you when you might need to explore a topic and learn more about it elsewhere.

Case study: Monthly accounting reports

This is a true story. Valerie is a business analyst in the accounting department of a medium-size 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 a well-deserved "atta-girl."

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 seventh edition of *VBA and Macros* is designed to work with Microsoft 365 features released up through August 2021. The previous editions of this book covered code for Excel 97 through Excel 2019. In 80% 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 monospace**—Indicates user input.

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

In addition to the case studies, you will see Notes, Tips, and Cautions.



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/ExcelVBAMacros365/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/ExcelVBAMacros365/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 <http://support.microsoft.com>.

Stay in touch

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

<http://twitter.com/MicrosoftPress>

<http://twitter.com/MrExcel>

This page intentionally left blank

Referring to ranges

In this chapter, you will:

- Learn how to reference the Range object
- Reference ranges in other sheets
- Reference a range relative to another range
- Use the Cells property to select a range
- Use the Offset property to refer to a range
- Use the Resize property to change the size of a range
- Use the Columns and Rows properties to specify a range
- Use the Union method to join multiple ranges
- Use the Intersect method to create a new range from overlapping ranges
- Use the IsEmpty function to check whether a cell is empty
- Use the CurrentRegion property to select a data range
- Use the SpecialCells property to interact with specific cells in a range
- Use the Areas collection to return a noncontiguous range
- Learn the syntax used for tables

A *range* can be a cell, a row, a column, or a grouping of any of these. The Range object is probably the most frequently used object in Excel VBA; after all, you're manipulating data on a sheet. Although a range can refer to any grouping of cells on a sheet, it can refer to only one sheet at a time. If you want to refer to ranges on multiple sheets, you must refer to each sheet separately.

This chapter shows you different ways of referring to ranges, such as specifying a row or column. You'll also find out how to manipulate cells based on the active cell and how to create a new range from overlapping ranges.

The Range object

The following is the Excel object hierarchy:

```
Application > Workbook > Worksheet > Range
```

The Range object is a property of the Worksheet object. This means it requires that a sheet be active or else it must reference a worksheet. Both of the following lines mean the same thing if Worksheets(1) is the active sheet:

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

There are several ways to refer to a Range object. Range("A1") is the most identifiable because that is how the macro recorder refers to it. However, all the following are equivalent when referring to cell D5:

```
Range("D5")  
[D5]  
Range("B3").Range("C3")  
Cells(5,4)  
Range("A1").Offset(4,3)  
Range("MyRange") 'assuming that D5 has a Name of MyRange
```

Which format you use depends on your needs. Keep reading. It will all make sense soon!

Syntax for specifying a range

The Range property has two acceptable syntaxes. To specify a rectangular range in the first syntax, specify the complete range reference just as you would in a formula in Excel:

```
Range("A1:B5")
```

In the alternative syntax, specify the upper-left corner and lower-right corner of the desired rectangular range. In this syntax, the equivalent statement might be this:

```
Range("A1", "B5")
```

For either corner, you can substitute a named range, the Cells property, or the ActiveCell property. The following line of code selects the rectangular range from A1 to the active cell:

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

The following statement selects from the active cell to five rows below the active cell and two columns to the right:

```
Range(ActiveCell, ActiveCell.Offset(5, 2)).Select
```

Referencing named ranges

You probably have already used named ranges on your worksheets and in formulas. You can also use them in VBA.

Use the following code to refer to the range "MyRange" in Sheet1:

```
Worksheets("Sheet1").Range("MyRange")
```

Notice the name of the range is in quotes—unlike the use of named ranges in formulas on the sheet itself. If you forget to put the name in quotes, Excel thinks you are referring to a variable in the program. One exception is if you use the shortcut syntax discussed in the next section. In that case, quotes aren't used.

Shortcut for referencing ranges

A shortcut is available when referencing ranges. The shortcut involves using square brackets, as shown in Table 3-1.

TABLE 3-1 Shortcuts for referencing ranges

Standard Method	Shortcut
<code>Range("D5")</code>	<code>[D5]</code>
<code>Range("A1:D5")</code>	<code>[A1:D5]</code>
<code>Range("A1:D5, G6:I17")</code>	<code>[A1:D5, G6:I17]</code>
<code>Range("MyRange")</code>	<code>[MyRange]</code>

Referencing ranges in other sheets

Switching between sheets by activating the needed sheet slows down your code. To avoid this, refer to a sheet that is not active by first referencing the `Worksheet` object:

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

This line of code references Sheet1 of the active workbook even if Sheet2 is the active sheet.

To reference a range in another workbook, include the `Workbook` object, the `Worksheet` object, and then the `Range` object:

```
Workbooks("InvoiceData.xlsx").Worksheets("Sheet1").Range("A1")
```

To use the `Range` property as an argument within another `Range` property, identify the range fully each time. For example, suppose that Sheet1 is your active sheet and you need to total data from Sheet2:

```
WorksheetFunction.Sum(Worksheets("Sheet2").Range(Range("A1"), _  
Range("A7")))
```


This line does not work. Why not? Although `Range("A1"), Range("A7")` is meant to refer to the sheet at the beginning of the code line (Sheet2), Excel does not assume that you want to carry the worksheet object reference over to these other Range objects; instead, Excel assumes that they refer to the active sheet, Sheet1. So, what do you do? Well, you could write this:

```
WorksheetFunction.Sum(Worksheets("Sheet2").Range(Worksheets("Sheet2"). _  
Range("A1"), Worksheets("Sheet2").Range("A7")))
```

However, not only is this a long line of code, but it is also difficult to read! Thankfully, there is a simpler way, using `With...End With`:

```
With Worksheets("Sheet2")  
WorksheetFunction.Sum(.Range(.Range("A1"), .Range("A7")))  
End With
```

Notice now there is a `.Range` in your code but without the preceding object reference. That's because `With Worksheets("Sheet2")` implies that the object of the range is that worksheet. Whenever Excel sees a period without an object reference directly to the left of it, it looks up the code for the closest `With` statement and uses that as the object reference.

Referencing a range relative to another range

Typically, the `Range` object is a property of a worksheet. It is also possible to have `Range` be the property of another range. In this case, the `Range` property is relative to the original range, which makes for unintuitive code. Consider this example:

```
Range("B5").Range("C3").Select
```

This code actually selects cell D7. Think about cell C3, which is located two rows below and two columns to the right of cell A1. The preceding line of code starts at cell B5. If we assume that B5 is in the A1 position, VBA finds the cell that would be in the C3 position relative to B5. In other words, VBA finds the cell that is two rows below and two columns to the right of B5, which is D7.

Again, I consider this coding style to be very unintuitive. This line of code mentions two addresses, and the actual cell selected is neither of these addresses! It seems misleading when you're trying to read this code.

You might consider using this syntax to refer to a cell relative to the active cell. For example, the following line of code activates the cell three rows down and four columns to the right of the currently active cell:

```
Selection.Range("E4").Select
```

I mention this syntax only because the macro recorder uses it. Recall that when you recorded a macro in Chapter 1, "Unleashing the power of Excel with VBA," with relative references on, the following line was recorded:

```
ActiveCell.Offset(0, 4).Range("A1").Select
```

This line found the cell four columns to the right of the active cell, and from there, it selected the cell that would correspond to A1. This is not the easiest way to write code, but it is the way the macro recorder does it.

Although a worksheet is usually the object of the Range property, occasionally, such as during recording, a range may be the property of a range.

Using the Cells property to select a range

The Cells property refers to all the cells of the specified Range object, which can be a worksheet or a range of cells. For example, this line selects all the cells of the active sheet:

```
Cells.Select
```

Using the Cells property with the Range object might seem redundant:

```
Range("A1:D5").Cells
```

This line refers to the original Range object. However, the Cells property has an Item property that makes the Cells property very useful. The Item property enables you to refer to a specific cell relative to the Range object.

The syntax for using the Item property with the Cells property is as follows:

```
Cells.Item(Row,Column)
```

You must use a numeric value for Row, but you may use the numeric value or string value for Column. Both of the following lines refer to cell C5:

```
Cells.Item(5,"C")  
Cells.Item(5,3)
```

Because the Item property is the default property of the Range object, you can shorten these lines as follows:

```
Cells(5,"C")  
Cells(5,3)
```

The ability to use numeric values for parameters is particularly useful if you need to loop through rows or columns. The macro recorder usually uses something like Range("A1").Select for a single cell and Range("A1:C5").Select for a range of cells. If you're learning to code only from the recorder, you might be tempted to write code like this:

```
FinalRow = Cells(Rows.Count, 1).End(xlUp).Row  
For i = 1 to FinalRow  
    Range("A" & i & ":E" & i).Font.Bold = True  
Next i
```

This little piece of code, which loops through rows and bolds the cells in columns A through E, is awkward to read and write. But how else can you do it? Like this:

```
FinalRow = Cells(Rows.Count, 1).End(xlUp).Row
For i = 1 to FinalRow
    Cells(i, "A").Resize(, 5).Font.Bold = True
Next i
```

Instead of trying to type the range address, the new code uses the `Cells` and `Resize` properties to find the required cell, based on the active cell. See the “Using the `Resize` property to change the size of a range” section later in this chapter for more information on the `Resize` property.

You can use the `Cells` properties for parameters in the `Range` property. The following refers to the range A1:E5:

```
Range(Cells(1,1), Cells(5,5))
```

This is particularly useful when you need to specify variables with a parameter, as in the previous looping example.

Using the `Offset` property to refer to a range

You’ve already seen a reference to `Offset` when you recorded a relative reference. `Offset` enables you to manipulate a cell based on the location of another cell, such as the active cell. Therefore, you do not need to know the address of the cell you want to manipulate.

The syntax for the `Offset` property is as follows:

```
Range.Offset(RowOffset, ColumnOffset)
```

For example, the following code affects cell F5 from cell A1:

```
Range("A1").Offset(RowOffset:=4, ColumnOffset:=5)
```

Or, shorter yet, you can write this:

```
Range("A1").Offset(4,5)
```

The count of the rows and columns starts at A1 but does not include A1.

If you need to go over only a row or a column, but not both, you don’t have to enter both the row and the column parameters. To refer to a cell one column over, use one of these lines:

```
Range("A1").Offset(ColumnOffset:=1)
Range("A1").Offset(,1)
```

Both of these lines have the same meaning, so the choice is yours. If you use the second line, make sure to include the comma so Excel knows that the 1 refers to the ColumnOffset argument. Referring to a cell one row up is similar:

```
Range("B2").Offset(RowOffset:=-1)
Range("B2").Offset(-1)
```

Once again, you can choose which one to use. It's a matter of the readability of the code.

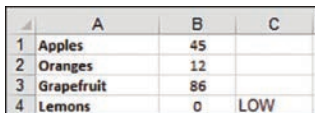
Suppose you have a list of produce in column A, with totals next to the produce items in column B. If you want to find any total equal to zero and place LOW in the cell next to it, do this:

```
Set Rng = Range("B1:B16").Find(What:="0", LookAt:=xlWhole, _
    LookIn:=xlValues)
Rng.Offset(, 1).Value = "LOW"
```

When used in a Sub and looping through a data set, it would look like this:

```
Sub FindLow()
    With Range("B1:B16")
        Set Rng = .Find(What:="0", LookAt:=xlWhole, LookIn:=xlValues)
        If Not Rng Is Nothing Then
            firstAddress = Rng.Address
            Do
                Rng.Offset(, 1).Value = "LOW"
                Set Rng = .FindNext(Rng)
            Loop While Not Rng Is Nothing And Rng.Address <> firstAddress
        End If
    End With
End Sub
```

The LOW totals are noted by the program, as shown in Figure 3-1.



	A	B	C
1	Apples	45	
2	Oranges	12	
3	Grapefruit	86	
4	Lemons	0	LOW

FIGURE 3-1 The code puts "LOW" next to the zeros in the data set.



Note Refer to the section "Object variables" in Chapter 4, "Looping and flow control," for more information on the Set statement.

Offsetting isn't only for single cells; you can use it with ranges. You can shift the focus of a range over in the same way you can shift the active cell. The following line refers to B2:D4 (see Figure 3-2):

```
Range("A1:C3").Offset(1,1)
```

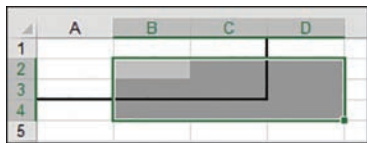


FIGURE 3-2 Offsetting the original range A1:C3 by one row and one column references a new range, B2:D4.

Using the `Resize` property to change the size of a range

The `Resize` property enables you to change the size of a range based on the location of the active cell. You can create a new range as needed. This is the syntax for the `Resize` property:

```
Range.Resize(RowSize, ColumnSize)
```

To reference the range B3:D13, use the following:

```
Range("B3").Resize(RowSize:=11, ColumnSize:=3)
```

Here's a simpler way to reference this range:

```
Range("B3").Resize(11, 3)
```

But what if you need to resize by only a row or a column—not both? You don't have to enter both the row and the column parameters.

To expand by two columns, use either of the following:

```
Range("B3").Resize(ColumnSize:=2)
```

or

```
Range("B3").Resize(,2)
```

Both lines mean the same thing. The choice is yours. If you use the second line, make sure to include the comma so Excel knows the 2 refers to the `ColumnSize` argument. Resizing just the rows is similar. You can use either of the following:

```
Range("B3").Resize(RowSize:=2)
```

or

```
Range("B3").Resize(2)
```

Once again, the choice is yours. It is a matter of the readability of the code.

From the list of produce, say that you want to find the zero totals and color the cells of the total and corresponding produce (see Figure 3-3). Here's what you do:

```
Set Rng = Range("B1:B16").Find(What:="0", LookAt:=xlWhole, _  
    LookIn:=xlValues)  
Rng.Offset(-1).Resize(2).Interior.ColorIndex = 15
```

	A	B
1	Apples	45
2	Oranges	12
3	Grapefruit	0
4	Lemons	26

FIGURE 3-3 You can resize a range to extend the selection.

Notice that the `Offset` property first moves the active cell over to the produce column. When you're resizing, the upper-left-corner cell must remain the same.

Resizing isn't only for single cells; you can use it to resize an existing range. For example, if you have a named range but need it and the column next to it, use this:

```
Range("Produce").Resize(,2)
```

Remember, the number you resize by is the total number of rows/columns you want to include.

Using the Columns and Rows properties to specify a range

The `Columns` and `Rows` properties refer to the columns and rows of a specified `Range` object, which can be a worksheet or a range of cells. They return a `Range` object referencing the rows or columns of the specified object.

You've seen the following line used, but what is it doing?

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

This line of code finds the last row in a sheet in which column A has a value and places the row number of that `Range` object into the variable called `FinalRow`. This can be useful when you need to loop through a sheet row by row; you will know exactly how many rows you need to go through.



Note Some properties of columns and rows require contiguous rows and columns in order to work properly. For example, if you were to use the following line of code, 9 would be the answer because only the first range would be evaluated:

```
Range("A1:B9, C10:D19").Rows.Count
```

However, if the ranges were grouped separately, the answer would be 19. Excel takes the top-left cell address, A1, and the bottom-right cell address, D19, and counts the rows in the range A1:D19:

```
Range("A1:B9", "C10:D19").Rows.Count
```

Using the Union method to join multiple ranges

The `Union` method enables you to join two or more noncontiguous ranges. It creates a temporary object of the multiple ranges, which enables you to affect them at the same time:

```
Application.Union(argument1, argument2, etc.)
```

The expression `Application` is not required. The following code joins two named ranges on the sheet, inserts the `=RAND()` formula, and bolds them:

```
Set UnionRange = Union(Range("Range1"), Range("Range2"))
With UnionRange
    .Formula = "=RAND()"
    .Font.Bold = True
End With
```

Using the Intersect method to create a new range from overlapping ranges

The `Intersect` method returns the cells that overlap between two or more ranges. If there is no overlap, an error is returned:

```
Application.Intersect(argument1, argument2, etc.)
```

The expression `Application` is not required. The following code colors the overlapping cells of the two ranges:

```
Set IntersectRange = Intersect(Range("Range1"), Range("Range2"))
IntersectRange.Interior.ColorIndex = 6
```

Using the IsEmpty function to check whether a cell is empty

The `IsEmpty` function returns a Boolean value that indicates whether a single cell is empty: `True` if empty, and `False` if not. The cell must truly be empty for the function to return `True`. If it contains even just a space that you cannot see, Excel does not consider the cell to be empty:

```
IsEmpty(Cell)
```

Say that you have several groups of data separated by a blank row. You want to make the separations a little more obvious. The following code goes down the data in column A. When it finds an empty cell in column A, it colors in the first four cells of that row (see Figure 3-4):

```
LastRow = Cells(Rows.Count, 1).End(xlUp).Row
For i = 1 To LastRow
    If IsEmpty(Cells(i, 1)) Then
        Cells(i, 1).Resize(1, 4).Interior.ColorIndex = 1
    End If
Next i
```

	A	B	C	D
1	Apples	Oranges	Grapefruit	Lemons
2	45	12	86	15
3	71%	53%	82%	52%
4				
5	Tomatoes	Cabbage	Lettuce	Green Peppers
6	58	24	31	0
7	30%	43%	68%	1%
8				
9	Potatoes	Yams	Onions	Garlic
10	10	61	26	29
11	18%	19%	22%	82%

FIGURE 3-4 You can make separations more obvious by using colored rows.

Using the CurrentRegion property to select a data range

CurrentRegion returns a Range object that represents a set of contiguous data. As long as the data is surrounded by one empty row and one empty column, you can select the data set by using CurrentRegion:

```
RangeObject.CurrentRegion
```

The following line selects A1:D3 because this is the contiguous range of cells around cell A1 (see Figure 3-5):

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

This is useful if you have a data set whose size is in constant flux.

	A	B	C	D	E
1	Apples	Oranges	Grapefruit	Lemons	
2	85	93	85	77	
3	6%	36%	80%	77%	
4					
5	Tomatoes	Cabbage	Lettuce	Green Peppers	
6	82	60	60	98	

FIGURE 3-5 You can use CurrentRegion to select a range of contiguous data around the active cell.

Case Study: Using the SpecialCells method to select specific cells

Even Excel power users might not have encountered the Go To Special dialog box. If you press the F5 key in an Excel worksheet, you get the normal Go To dialog box (see Figure 3-6). In the lower-left corner of this dialog box is a button labeled Special. Click this button to get to the super-powerful Go To Special dialog box (see Figure 3-7).

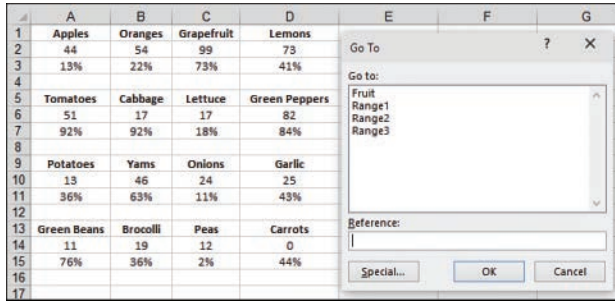


FIGURE 3-6 Although the Go To dialog box doesn't seem useful, click the Special button in the lower-left corner to specify what type of cells to select.

In the Excel interface, the Go To Special dialog box enables you to select only cells with formulas, only blank cells, or only the visible cells. Selecting only visible cells is excellent for grabbing the visible results of AutoFiltered data. If you already have a range highlighted, only cells within this range meeting the criteria will be selected. Make sure only one cell is selected to search the entire sheet.

To simulate the Go To Special dialog box in VBA, use the `SpecialCells` method. This enables you to act on cells that meet certain criteria, like this:

```
RangeObject.SpecialCells(Type, Value)
```

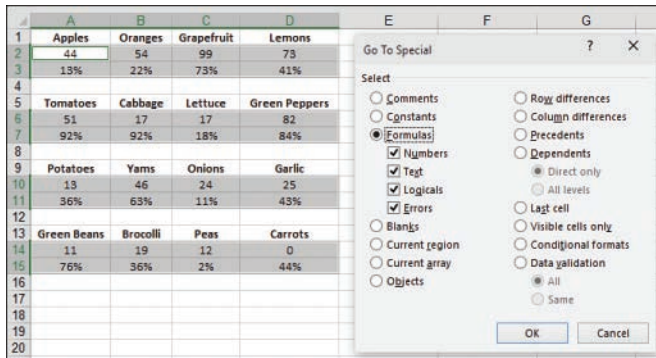


FIGURE 3-7 The Go To Special dialog box has many incredibly useful selection tools, such as one for selecting only the formulas on a sheet.

The `SpecialCells` method has two parameters: `Type` and `Value`. `Type` is one of the `xlCellType` constants:

```
xlCellTypeAllFormatConditions
xlCellTypeAllValidation
xlCellTypeBlanks
xlCellTypeComments
xlCellTypeConstants
xlCellTypeFormulas
xlCellTypeLastCell
xlCellTypeSameFormatConditions
xlCellTypeSameValidation
xlCellTypeVisible
```

Set one of the following optional `Value` constants if you use `xlCellTypeConstants` or `xlCellTypeFormulas`:

```
xlErrors
xlLogical
xlNumbers
xlTextValues
```

The following code returns all the ranges that have conditional formatting. It produces an error if there are no conditional formats and adds a border around each contiguous section it finds:

```
Set rngCond = ActiveSheet.Cells.SpecialCells(xlCellTypeAllFormatConditions)
If Not rngCond Is Nothing Then
    rngCond.BorderAround xlContinuous
End If
```

Have you ever had someone send you a worksheet without all the labels filled in? Some people think that the data shown in Figure 3-8 looks tidy. They enter the `Region` field only once for each region. This might look aesthetically pleasing, but it's impossible to sort.

	A	B	C
1	Region	Product	Sales
2	North	ABC	766,469
3		DEF	776,996
4		XYZ	832,414
5	East	ABC	703,255
6		DEF	891,799
7		XYZ	897,949

FIGURE 3-8 The blank cells in the `Region` column make it difficult to sort data sets such as this.

Using the `SpecialCells` method to select all the blanks in this range is one way to fill the blank region cells quickly using the region found above them:

```
Sub FillIn()
On Error Resume Next 'Need this because if there aren't any blank
'cells, the code will error
Range("A1").CurrentRegion.SpecialCells(xlCellTypeBlanks).FormulaR1C1 _
    = "=R[-1]C"
Range("A1").CurrentRegion.Value = Range("A1").CurrentRegion.Value
End Sub
```

In this code, `Range("A1").CurrentRegion` refers to the contiguous range of data in the report. The `SpecialCells` method returns just the blank cells in that range. This particular formula fills in all the blank cells with a formula that points to the cell above the blank cell. (You can read more about R1C1-Style Formulas in Chapter 5, "R1C1-style formulas.") The second line of code is a fast way to simulate using the Copy and Paste Special Values commands. Figure 3-9 shows the results.

	A	B	C
1	Region	Product	Sales
2	North	ABC	766,469
3	North	DEF	776,996
4	North	XYZ	832,414
5	East	ABC	703,255
6	East	DEF	891,799
7	East	XYZ	897,949

FIGURE 3-9 After the macro runs, the blank cells in the Region column have been filled with data.

Using the Areas collection to return a noncontiguous range

The `Areas` collection is a collection of noncontiguous ranges within a selection. It consists of individual `Range` objects representing contiguous ranges of cells within the selection. If a selection contains only one area, the `Areas` collection contains a single `Range` object that corresponds to that selection.

You might be tempted to loop through the rows in a sheet and check the properties of a cell in a row, such as its formatting (for example, font or fill) or whether the cell contains a formula or value. Then you could copy the row and paste it to another section. However, there is an easier way. In Figure 3-10, the user enters the values below each fruit and vegetable. The percentages are formulas. The following line of code selects the cells with numeric constants and copies them to another area:

```
Set NewDestination = ActiveSheet.Range("I1")
For Each Rng In Cells.SpecialCells(xlCellTypeConstants, 1).Areas
    Rng.Copy Destination:=NewDestination
    Set NewDestination = NewDestination.Offset(Rng.Rows.Count)
Next Rng
```

	A	B	C	D	E	F	G	H	I	J	K	L
1	Apples	Oranges	Grapefruit	Lemons					45	12	86	15
2	45	12	86	15					58	24	31	0
3	6%	65%	78%	45%					10	61	26	29
4									46	64	79	95
5	Tomatoes	Cabbage	Lettuce	Green Peppers								
6	58	24	31	0								
7	22%	31%	70%	65%								
8												
9	Potatoes	Yams	Onions	Garlic								
10	10	61	26	29								
11	18%	49%	57%	86%								
12												
13	Green Beans	Broccoli	Peas	Carrots								
14	46	64	79	95								
15	27%	56%	21%	42%								

FIGURE 3-10 The `Areas` collection makes it easier to manipulate noncontiguous ranges.

Referencing tables

A table is a special type of range that offers the convenience of referencing named ranges. However, tables are not created in the same manner as other ranges. For more information on how to create a named table, see Chapter 6, “Creating and manipulating names in VBA.”

Although you can reference a table by using `Worksheets(1).Range("Table1")`, you have access to more of the properties and methods that are unique to tables if you use the `ListObjects` object, like this:

```
Worksheets(1).ListObjects("Table1")
```

This opens the properties and methods of a table, but you can't use that line to select the table. To do that, you have to specify the part of the table you want to work with. To select the entire table, including the header and total rows, specify the `Range` property:

```
Worksheets(1).ListObjects("Table1").Range.Select
```

The table part properties include the following:

- `Range`—Returns the entire table.
- `DataBodyRange`—Returns the data part only.
- `HeaderRowRange`—Returns the header row only.
- `TotalRowRange`—Returns the total row only.

What I really like about coding with tables is the ease of referencing specific columns of a table. You don't have to know how many columns to move in from a starting position or the letter/number of the column, and you don't have to use a `FIND` function. Instead, you can use the header name of the column. For example, to select the data of the `Qty` column of the table, but not the header or total rows, do this:

```
Worksheets(1).ListObjects("Table1").ListColumns("Qty").DataBodyRange.Select
```



Note For more details on coding with tables, check out *Excel Tables: A Complete Guide for Creating, Using, and Automating Lists and Tables* by Zack Barresse and Kevin Jones (ISBN: 9781615470280).

Next steps

Referencing ranges is an essential part of programming in Excel. Now that you're getting an idea of how Excel works, it's time to learn about a fundamental component of any programming language: loops. If you have taken a programming class, you will be familiar with basic loop structures. VBA supports all the usual loops. Chapter 4 also describes a special loop, `For Each . . . Next`, which is unique to object-oriented programming such as VBA.

Index

Numerics

3D models, 575
24-hour time, 117–118

A

A1 addressing, 87, 89–91, 94–95
above/below average rules, 340, 355. *See also* conditional formatting

absolute references, R1C1 addressing and, 92–93

Access

creating a shared database, 449–450

database

adding records to a, 452–453
deleting records via ADO, 458
retrieving records from, 453–455
summarizing records via ADO, 458–459
updating an existing record, 455–458

MDB (multidimensional database) files, 447–448

reading from, 451

tables

adding fields on the fly, 462–463
adding on the fly, 461–462
checking for the existence of, 460
checking for the existence of a field, 461
operations, 451

ACE engine, 447–448

ActiveX controls

minimizing duplicate code, 147–148
running a macro with, 536

AddAboveAverage method, 355

.AddChart2 method, 313, 314–315, 318–319, 337–338

adding

color scales to a range, 346–347
data bars to a range, 342–346
icon sets to a range, 347–350
names, 98–100

add-ins, 539

Excel

characteristics of, 539–540
closing, 545

converting workbooks to, 540–541
creating with VB Editor, 542–543
hidden workbook as an alternative, 545–547
installing, 543–544
removing, 545
saving files as, 541–542
security, 544–545

Office, 549

adding interactivity, 554–557, 559–560
defining, 558–559
Hello World, 550, 551–554
JavaScript and, 569–570
writing to the content or task pane, 569

AddTop10 method, 355

AddUniqueValues method, 356–358

ADOs (ActiveX Data Objects), 448

adding tables on the fly, 461–462
checking for the existence of a field, 461
checking for the existence of a table, 460
connection, 450, 451
cursor, 450–451
deleting records from an Access database, 458
lock type, 451
record set, 450
summarizing records via, 458–459

Advanced Filter, 183

building, 184
criteria ranges and, 191–192, 193
extracting a unique list of values, 185
changing the list range to a single column, 185
copying the customer heading before filtering, 185–186

Filter In Place, 201

catching no records, 201–202
showing all records, 202
formula-based conditions, 194–195
returning above-average records, 200
using in the Excel user interface, 195–196
using with VBA, 196–200

joining multiple criteria

with a logical AND, 193
with a logical OR, 192

Advanced Filter

- replacing a list of values with a condition created from a formula, 193–194
 - retrieving unique combinations of two or more fields, 190
 - xlFilterCopy, 202
 - combining multiple techniques, 205–209
 - copying a subset of columns and reordering, 203–205
 - copying all columns, 202–203
 - alpha characters, sorting, 301–303
 - API (Application Programming Interface), 491. *See also* Spotify
 - declarations, 492
 - 32-bit- and 64-bit-compatible, 493–494
 - private, 492
 - using, 493
 - functions
 - checking whether an Excel file is open on a network, 495–496
 - creating a running timer, 498–499
 - customizing the About dialog box, 497
 - disabling the X for closing a user form, 498
 - playing sounds, 499
 - retrieving display-resolution information, 496–497
 - retrieving the computer name, 495
 - getting credentials for accessing, 386–387
 - applications, events, 120–124
 - setting up a class module, 134
 - trapping, 134–135
 - Archibald, R., 260
 - Areas collection, 66
 - arrays, 125, 145
 - declaring, 125–126
 - dynamic, 130–131
 - filling, 127–128
 - formulas, 573
 - JavaScript, 562–563
 - multidimensional, declaring, 126–127
 - names and, 104
 - passing, 131–132
 - retrieving data from, 128–129
 - speeding up code with, 129–130
 - assigning, macros
 - to a form control, 12–13
 - to an object, 13
 - AutoFilter
 - filtering
 - by color, 179
 - by icon, 179–180
 - replacing loops with, 175–178
 - selecting a dynamic date range, 180–181
 - selecting multiple items, 178–179
 - selecting using the Search box, 179
 - turning off drop-down menus, 209–210
 - AutoSort method, 226
 - Autosum button, 24–25, 26
- ## B
- backward compatibility, .AddChart2 method, 337–338
 - Barresse, Z., 263, 278
 - Excel Tables: A Complete Guide for Creating, Using, and Automating Lists and Tables*, 67
 - barriers to learning VBA, 2. *See also* learning VBA
 - BASIC, 2, 28
 - binding. *See* early binding; late binding, 499
 - blocks, With...End With, 50
 - bookmarks, 441–442
 - BookOpen() function, 288–289
 - borders, chart, 331
 - breakpoints, 40, 41
 - backing up or moving forward, 40
 - querying
 - by hovering, 42
 - using a Watches window, 42–43
 - using the Immediate window, 41–42
 - setting with watches, 43
 - Bricklin, D., 87
 - building
 - advanced filters, 184
 - Data Model, 247–249
 - LAMBDA functions, 307–308
 - pivot tables, 245–246, 255–256
 - UDFs (user-defined functions), 286–287
 - buttons
 - Autosum, 24–25, 26
 - command, 160, 161–162
 - command, running a macro with, 534–535
 - creating, 557–558
 - Help, 576
 - option
 - adding to a userform, 165–166
 - events, 166
 - spin, 168–169
 - toggle, 471–472
 - using images on, 527

C

- calculated fields, 249–250
- calculated items, 250
- .Calculation property, 223
- calculations, changing to show percentages, 223–225
- calling, userforms, 156
- canceling a scheduled macro, 408
- Case statement, complex expressions in, 84
- cell pointer, 20–21, 23
- cells
 - creating a progress indicator, 274–275
 - duplicate, marking, 356–358
 - empty, 62–63, 226
 - formatting, 243–244
 - based on their value, 358
 - that contain blanks or errors, 359
 - that contain dates, 359
 - that contain text, 358–359
 - using a formula, 359–361
 - notes, placing charts in, 265–267
 - returning the column letter of an address, 305–306
 - selecting visible only, 181–182
- Cells property, referencing ranges with, 57–58
- ChartFormat object, 327
- charts, 313, 574. *See also* sparklines
 - applying a color, 320–321
 - borders, 331
 - combo, 331–334
 - creating
 - macro recorder and, 318
 - using .AddChart2 method, 314–315
 - embedded, events, 118–120
 - events, 118, 136–137
 - exporting as a graphic, 337
 - filtering, 322
 - formatting, 327–328
 - line settings, 331
 - map, creating, 335
 - placing in a cell note, 265–267
 - referring to, 318–319
 - SetElement method, 323–327
 - sheet events, 119–120
 - specifying a title, 319–320
 - styles, 315–318
 - trendline, formatting, 331
 - waterfall, creating, 336–337
 - win/loss, formatting, 377–378
- check boxes, 165, 466–468
- class modules, 133
 - application events, trapping, 134–135
 - clsCtlColor, 485–486
 - collections, creating, 142–144
 - custom objects
 - creating, 137–139
 - using, 139–140
 - embedded chart events, trapping, 136–137
 - Excel state, 268–270
 - inserting, 133–134
 - minimizing duplicate code for ActiveX labels, 147–148
- clauses
 - Step, 72–73
 - Until, 77–79
 - While, 77–79
- cleaning up code, 45–52
- Close method, 436–437
- closing, add-ins, 545
- code. *See also* M; XML
 - adding to new workbooks, 279–280
 - breakpoints, 40
 - cleaning up, 45–52
 - comments, 18
 - duplicate, minimizing, 147–148
 - early binding, referencing a Word object, 427–430
 - examining, 17–19, 33–34
 - Help topics, 37
 - JavaScript, 560–561
 - late binding, referencing a Word object, 430
 - picture catalog userform, 481–483
 - pivot tables
 - building with a timeline, 243
 - building with two slicers, 238–240
 - creating a Data model, 247–249
 - creating a static summary from, 218–219
 - generating, 216–217
 - producing one report per product, 227–229
 - protecting, 513–514
 - speeding up with arrays, 129–130
 - Step Into feature, 38–40
 - stepping through, 40–41. *See also* breakpoints
 - streamlining, 46
 - TypeScript, creating a pivot table, 255
 - VBA, 27
- collection(s), 29, 140
 - Areas, 66
 - controls and, 473–475

collection(s)

- creating, 140–141
 - in a class module, 142–144
 - in a standard module, 141–142
- dictionaries and, 145
- FormatConditions, 340–341
- ColName() function, 305–306
- color(s)
 - adding to fields in a userform, 485–486
 - applying
 - to a chart, 320–321
 - to data bars, 343, 344–345, 352–354
 - to objects, 329
 - filtering by, 179
 - RGB, applying to sparklines, 373–374
 - scales, 339, 346–347
 - theme, applying to sparklines, 369–372
- Columns property, 61
- combining
 - forms, 169–171
 - workbooks, 262–263
- combo boxes, 162–163, 164–165
- combo charts, creating, 331–334
- command buttons, 160
 - events, 161–162
 - running a macro with, 534–535
- comments, 18
 - HTML, 557
 - JavaScript, 556, 561
 - names and, 100
 - XML, 519
- comparing, VBA and TypeScript, 253–255
- concatenation, 300–301
- conceptual filter, 230–231
- conditional formatting, 340–341
 - above/below average rules, 355
 - data bars, 352–354
 - icons sets, 351–352
 - marking unique or duplicate cells, 356–358
 - NumberFormat property, 361–362
 - top/bottom rules, 355–356
 - using a formula, 359–361
- conditions, 81
 - formula-based
 - returning above-average records, 200
 - using in the Excel user interface, 195–196
 - using with VBA, 196–200
 - formulas-based, 194–195
- configuration, pivot table, 213–214

- constants
 - Help topics, 35
 - icon sets, 348–349
 - retrieving the real value of
 - using the Object Browser, 433–434
 - using the Watches window, 433
 - SetElement method, 323–327
 - values, 433
 - xlColumnDataType, 416–417
- ContainsText() function, 303–304
- controls
 - accelerator keys, 483–484
 - adding
 - on the fly, 479
 - at runtime, 477–478
 - adding to a ribbon, 520
 - attributes, 521–525
 - bug fix when adding to a form, 159
 - check boxes, 466–468
 - collections and, 473–475
 - combo boxes, 162–163, 164–165
 - command buttons, 160, 161–162
 - early binding, New keyword, 430–431
 - frames, 474, 475
 - graphic, 167
 - labels, 159, 161–162
 - list boxes, 162–163
 - events, 164–165
 - multicolumn, 486–487
 - MultiSelect property, 163–164
 - MultiPage, 169–171, 468
 - option buttons, 165–166
 - ProgIds, 480
 - programming, 158–159
 - RefEdit, 470–471
 - renaming, 159
 - scrollbars, 472–473
 - spin buttons, 168–169
 - tab strips, 468–470
 - text boxes, 160, 161–162
 - tip text, adding to userforms, 484
 - toggle button, 471–472
- ConvertWeekDay() function, 299–300
- copy and paste, 49. *See also* xlFilterCopy
- Copy method, 49
- CopyFromRecordSet method, 453, 455
- Count function, 221
- COUNTIF function, 360

- counting, unique functions, 296
- CreateObject function, 431
- CreatePivotTable method, 213–214
- creating
 - cell progress indicator, 274–275
 - charts
 - combo, 331–334
 - map, 335
 - using .AddChart2 method, 314–315
 - waterfall, 336–337
 - collections, 140–141
 - in a class module, 142–144
 - in a standard module, 141–142
 - custom objects, 137–139
 - custom properties, 148–151
 - dashboards, 378, 379–383
 - Data Model, 247–249
 - macro button
 - on the Quick Access Toolbar, 11–12
 - on the ribbon, 10–11
 - new instances of objects, 431
 - pivot tables in VBA, 212
 - adding fields to the data area, 214–217
 - configuration, 213–214
 - defining the pivot cache, 213
 - shared Access database, 449–450
 - sparklines, 363–365, 379–383
 - UDFs (user-defined functions), 285
 - userforms, 155–156
- criteria
 - complex ranges, 193
 - logical AND, 193
 - logical OR, 192
 - ranges, 191–192
 - replacing a list of values with a condition created from a formula, 193–194
- CSS (Cascading Style Sheets), 558
- CSV files
 - importing and deleting, 260
 - opening, 417–419
- CurrentRegion property, selecting ranges with, 63
- custom objects
 - creating, 137–139
 - using, 139–140
- custom properties, creating, 148–151

D

- DAOs (data access objects), 448
- dashboards
 - creating, 378
 - creating individual sparklines in, 379–383
 - placing query results on, 398–401
- data bars, 339
 - adding to a range, 342–346
 - color, applying, 344–345
 - using two colors in a range, 352–354
- Data Model, 244
 - creating, 247–249
 - loading large files to, 423–424
 - tables
 - adding, 244–245
 - creating a relationship between, 245
- data sets, converting fixed-width reports to, 280–283
- data visualizations, 252–253, 339
 - above/below average rules, 340
 - color scales, 339
 - conditional formatting, 340–341
 - data bars, 339
 - highlight cells, 340
 - icon sets, 340
 - top/bottom rules, 340
- date(s)
 - converting week number to, 299–300
 - dynamic filters and, 180–181
 - grouping to months, quarters, or years, 221–223
 - of last save, retrieving, 292
 - retrieving, 292–293
- debugging tools
 - breakpoints, 40, 41
 - backing up or moving forward, 40
 - queries and, 41–43
 - Step Into feature, 38–41
- declaring
 - arrays, 125–127
 - UDTs (user-defined types), 149
- Delete method, 100
- deleting, names, 100
- delimited files, opening, 417–419
- Design tab, changing the layout, 250–251
- Developer tab
 - Add-ins group, 4
 - Code group, 4

Developer tab

- Controls group, 4
 - Disable All Macros with Notification setting, 8
 - displaying, 3–4
 - macro settings, 7–8
 - Modify group, 4
 - Relative References, 21–24
 - XML group, 4
 - dialog boxes
 - Advanced Filter, 184
 - File Open, 173–174
 - Go To Special, 64, 201
 - loops and, 182–183
 - Visible Cells Only option, 181–182
 - Name Manager, 97
 - Record Macro, filling out, 9–10
 - Show Values As tab of the Value Field Settings, 223
 - dictionaries, 145–146
 - DIM statement, 79–80
 - displaying, Developer tab, 3–4
 - DLLs (Dynamic Link Libraries), 491
 - Document object, 435–437
 - Do...loops, 75–77
 - stopping, 77
 - While and Until clauses, 77–79
 - drilling-down a pivot table, 270–271
 - duplicate
 - cells, marking, 356–358
 - code, minimizing, 147–148
 - values, 340
 - dynamic arrays, 130–131
 - dynamic filters, 180–181
- ## E
- early binding
 - New keyword, 430–431
 - referencing a Word object, 427–430
 - email, validating an address, 293–295
 - embedded charts, events, 118–120, 136–137
 - empty cells, 62–63, 226
 - enabling, events, 113
 - End+down arrow shortcut, 47–48
 - EndKey method, 437
 - error handling, 501–503
 - client training and, 509–510
 - custom ribbon and, 529–533
 - Debug mode, 510
 - encountering errors on purpose, 509
 - On Error GoTo syntax, 505–506
 - Excel versions and, 515–516
 - generic, 506
 - ignoring errors, 506–507
 - message boxes and, 509
 - misleading errors, 503–504
 - page setup, 507–508
 - Power Query, 403
 - protecting code and, 513–514
 - runtime error 9, 511–512
 - runtime error 1004, 512–513
 - suppressing Excel warnings, 508
 - events, 112
 - application, 120–124
 - setting up a class module, 134
 - trapping, 134–135
 - Change, 267
 - chart, 118
 - embedded, 118–120
 - trapping, 136–137
 - check box, 467–468
 - combo box, 164–165
 - command button, 161–162
 - enabling, 113
 - graphic control, 167
 - label, 161–162
 - levels of, 111–112
 - list box, 164–165
 - MultiPage, 171
 - option button, 166
 - parameters, 112–113
 - QueryClose, 172
 - RefEdit control, 470–471
 - scrollbar, 473
 - sheet
 - chart-level, 119–120
 - workbook-level, 115–116
 - spin button, 169
 - tab strip, 469–470
 - text box, 161–162
 - toggle button, 471–472
 - userform, 157–158
 - workbook, 113–115
 - worksheet, 116–117
 - Excel. *See also* ribbon
 - add-ins
 - closing, 553
 - converting workbooks to, 550–551
 - creating with VB Editor, 551–552
 - hidden workbook as an alternative, 554–555

- installing, 552
 - removing, 554
 - saving files as, 551
 - security, 553
 - Compatibility mode, 576
 - Developer tab
 - Add-ins group, 4
 - Code group, 4
 - Controls group, 4
 - Disable All Macros with Notification setting, 8
 - displaying, 3–4
 - macro settings, 7–8
 - Modify group, 4
 - XML group, 4
 - error handling, 501–503
 - Help button, 576
 - Point mode, 185
 - purchasing, 571
 - Quick Analysis tool, 573
 - Ready mode, 407
 - RELS file, 525–526
 - single-document interface, 572–573
 - updates, scheduling, 407
 - versions, 571
 - error handling and, 515–516
 - pivot table evolution, 211–212
 - .xslm files, 525
 - Excel8CompatibilityMode property, 577–578
 - Execute method, 458
 - Exists method, 146
 - Exit For statement, 73–74
 - exporting
 - charts as a graphic, 337
 - to an XML file, 264–265
 - expressions
 - Case statement and, 84
 - text, 83
 - watches and, 42–43
 - extracting a unique list of values
 - with the user interface, 185–186
 - with VBA code, 186–189
- F**
- fields
 - active, coloring, 485–486
 - adding on the fly, 462–463
 - adding to the pivot table data area, 214–217
 - calculated, 249–250
 - checking for the existence of via ADOs, 461
 - form, controlling in Word, 443–444
 - protected password box, 275–277
 - file
 - operations, 257
 - exporting data to an XML file, 264–265
 - importing and deleting a CSV file, 260
 - listing files in a directory, 257–260
 - reading a text file into memory and parsing, 260–261
 - types
 - .xslm, 5
 - macro-supported, 4–5
 - File Open dialog box, 173–174
 - Filter In Place, 201
 - catching no records, 201–202
 - showing all records, 202
 - filtering, 229
 - filters
 - chart, 322
 - conceptual, 230–231
 - manual, 229–230
 - OLAP pivot table, 271–273
 - search, 234–236
 - ShowDetail property and, 250
 - slicers, 237–241
 - Timelines, 241–243
 - types, 232–234
 - FirstNonZeroLength() function, 296–297
 - fixed-width reports, converting to a data set, 280–283
 - flow control. *See also* loops
 - conditions, 81
 - If-Else If-End If construct, 82–83
 - If-Then-Else construct, 81
 - If-Then-Else-End If construct, 82
 - If-Then-End If construct, 82
 - Select Case construct, 83–84
 - For...Each loops, 79
 - For...Next loops, 69–71
 - Step clause, 72–73
 - using variables in the For statement, 72
 - Format method, 327–328
 - Format tab, 327–328
 - Shape Fill drop down, 328–330
 - FormatConditions collection, 340–341
 - formatting. *See also* conditional formatting
 - cells, 243–244
 - based on their value, 358
 - that contain blanks or errors, 359

formatting

- that contain dates, 359
- that contain text, 358–359
- using a formula, 359–361
- charts, 327–328
- conditional, 340–341
- line settings, 331
- resetting on tables, 278–279
- sparklines, 369
 - elements, 374–377
 - using RGB colors, 373–374
 - using theme colors, 369–372
 - win/loss charts, 377–378
- forms, 160
 - combining, 169–171
 - controls. *See also* controls
 - assigning a macro to a, 12–13
 - bug fix when adding to a form, 159
 - combo boxes, 162–163
 - command buttons, 160
 - graphic, 167
 - labels, 159
 - list boxes, 162–163, 164–165
 - MultiPage, 169–171
 - option buttons, 166
 - programming, 158–159
 - renaming, 159
 - spin buttons, 168–169
 - text boxes, 160
 - fields, controlling in Word, 443–444
 - getting a file name, 173–174
 - illegal window closing, 172
 - retrieving information from, 160–161
 - transparent, setting up, 487–488
 - verifying field entry, 171
- Formula property, 352–354
- formula-based conditions, 194–195
 - returning above-average records, 200
 - using in the Excel user interface, 195–196
 - using with VBA, 196–200
- formulas
 - A1 addressing, 87, 89–91
 - array, 573
 - conditional formatting and, 359–361
 - controls, combo boxes, 164–165
 - naming, 101
 - R1C1 addressing, 49, 87, 90, 91
 - absolute references, 92–93
 - mixed references, 93
 - referring to entire columns or rows, 93
 - relative references, 91–92
 - remembering column numbers associated with column letters, 95–96
 - replacing A1 formulas with, 94–95
 - toggling to, 88
- frames, 474, 475
- Frankston, B., 87
- functions
 - API
 - checking whether an Excel file is open on a network, 495–496
 - creating a running timer, 498–499
 - customizing the About dialog box, 497
 - disabling the X for closing a user form, 498
 - playing sounds, 499
 - retrieving display-resolution information, 496–497
 - retrieving the computer name, 495
 - building in Power Query, 390–393
- Count, 221
- COUNTIF, 360
- CreateObject, 431
- GetObject, 431–432
- InputBox, 153–154
- IsEmpty, 62–63
- JavaScript, 560–561
- LAMBDA, 307, 573
 - building, 307–308
 - sharing, 308
 - SLUGIFY.PLUS(), 309–310
 - TOC2HTML(), 310–311
- MsgBox, 154–155
- NOW, 292
- recursive, 290
- RGB, 373–374
- SUM, 25
- user-defined, 285, 304–305
 - BookOpen(), 288–289
 - building, 286–287
 - ColName(), 305–306
 - ContainsText(), 303–304
 - ConvertWeekDay(), 299–300
 - creating, 285
 - DateTime(), 292–293
 - FirstNonZeroLength(), 296–297
 - GetAddress(), 305
 - IsValidEmail(), 293–295
 - LastSaved(), 292

- MSubstitute(), 297–298
- NumUniqueValues(), 296
- RetrieveNumbers(), 298–299
- sharing, 288
- SheetExists(), 289–291
- SortConcat(), 300–301
- SumColor(), 295
- WinUserName(), 291–292
- VLOOKUP, 107
- XLOOKUP, named ranges and, 107–109

G

- GetAddress() function, 305
- GetObject function, 431–432
- global
 - names, 97–98
 - variables, 402
- Go To Special dialog box, 64, 201
 - loops and, 182–183
 - Visible Cells Only option, 181–182
- González Ruiz, J. P., 268
- gradients, applying to objects, 330
- graphic controls, 167
- grouping, dates, 221–223

H

- Help topics, 32–34
 - code and, 37
 - constants, 35
 - optional parameters, 34–35
 - properties and, 38
 - ToolTips and, 42
- Hide method, 156–157
- hiding
 - names, 106
 - userforms, 156–157
- HomeKey method, 437
- HTML, 557
 - comments, 557
 - tags, 557
- hyperlinks
 - returning the address, 305
 - running a macro from, 537
 - userforms and, 476–477

I

- icon(s), 340, 575
 - filtering by, 179–180
 - plus sign, 322–323
 - Restore Down, 39
 - sets, adding to a range, 347–348, 351–352
 - constants, 348–349
 - specifying an icon set, 348–349
 - specifying ranges for each icon, 350
 - Stop Recording, 10
 - using on the ribbon, 527–529
- If statements, nesting, 84–86
- If-Else If-End If construct, 82–83
- If-Then-Else construct, 81
- If-Then-Else-End If construct, 82
- If-Then-End If construct, 82
- images, adding to userforms on the fly, 480–481
- Immediate window (VB Editor), 35, 41–42
- importing
 - CSV files, 260
 - text files, 413
- input boxes, 153–154
- InsertLines method, 280
- installing, add-ins, 543–544
- Intersect method, 62
- IsEmailValid() function, 293–295
- IsEmpty function, 62–63

J

- JavaScript
 - adding interactivity to Office add-ins, 559–560
 - arrays, 562–563
 - comments, 561
 - custom functions, 549
 - For each...next statement, 565–566
 - functions, 560–561
 - if statements, 564
 - for loops, 563
 - math functions, 567–568
 - Office add-ins and, 569–570
 - operators, 566–567
 - Select Case construct, 564–565
 - strings, 562
 - variables, 561

Jet engine

Jet engine, 448
joining, ranges, 62
Jones, K., 263
*Excel Tables: A Complete Guide for Creating, Using,
and Automating Lists and Tables*, 67

K

Kaji, M., 260
Kapor, M., 87
keyboard shortcuts. *See* shortcut(s)
keywords
 New, 430–431
 Preserve, 131
Klann, D., 275

L

labels, 159, 161–162
LAMBDA functions, 307, 573
 building, 307–308
 sharing, 308
 SLUGIFY.PLUS(), 309–310
 TOC2HTML(), 310–311
Lanzo, L., 264
late binding, 145, 430
layout
 pivot table, 250–251
 report, 251
learning VBA, 3
 barriers
 macro recorder, 2
 syntax, 2
line settings, trendline, 331
LineFormat object, 331
list boxes, 162–163
 events, 164–165
 multicolumn, 486–487
 MultiSelect property, 163–164
Load method, 156
loading large files to the Data Model, 423–424
local names, 97–98, 104
logical AND, 193
logical OR, 192
loops, 69
 Do, 75–77
 stopping, 77
 While and Until clauses, 77–79

 exiting early, 73–74
 For...Each, 79
 For...Next, 69–71
 Step clause, 72–73
 using variables in the For statement, 72
 Go To Special dialog box and, 182–183
 JavaScript, 563
 M, 402
 If-then logic, 403–404
 List.Generate, 404–406
 nesting, 74–75
 replacing with AutoFilter, 175–178
 While...Wend, 79
Lotus 1–3, 24, 87

M

M, 385
 global variables, 402
 loops, 402
 If-then logic, 403–404
 List.Generate, 404–406
 searching Spotify for an artist, 388–389
macro button, creating
 on the Quick Access Toolbar, 11–12
 on the ribbon, 10–11
macro recorder
 absolute references, 19–20
 charts and, 318
 relative references, 20–24
 shortcomings of, 15–16
 tips for using, 25–26
macros. *See also* code
 assigning
 to a form control, 12–13
 to an object, 13
 attaching
 to an ActiveX control, 536
 to a shape, 535
 canceling, 408
 copying to new workbooks, 279–280
 passwords and, 515
 recording, 8–9
 running, 10
 from a hyperlink, 537
 using a keyboard shortcut, 533–534
 scheduling, 407, 408–411
 searching Spotify database for an artist, 400

- security, 6
 - adding a trusted location, 6–7
 - Disable All Macros with Notification setting, 8
 - enabling outside trusted locations, 7–8
- supported file types, 4–5
- testing, 19, 22–23
- manual filter, 229–230
- map charts, creating, 335
- matrix, 126
- MDB (multidimensional database) files, 447–448
- message boxes, 154–155
- methods, 28
 - AddAboveAverage, 355
 - .AddChart2, 313, 318–319
 - backward compatibility, 337–338
 - creating charts with, 314–315
 - AddTop10, 355
 - AddUniqueValues, 356–358
 - AutoFilter, 176–177
 - AutoSort, 226
 - Close, 436–437
 - Copy, 49
 - CopyFromRecordSet, 453, 455
 - CreatePivotTable, 213–214
 - Delete, 100
 - EndKey, 437
 - Execute, 458
 - Exists, 146
 - Format, 327–328
 - Hide, 156–157
 - HomeKey, 437
 - InsertLines, 280
 - Intersect, 62
 - Load, 156
 - Modify, 343
 - OneColorGradient, 330
 - OnTime, 406–407
 - Open, 436
 - parameters, 29–31
 - .Patterned, 330
 - PresetGradient, 330
 - PresetTextured, 329
 - PrintOut, 437
 - SaveAs2, 436
 - Select, 38
 - SelectAll, 474

- SetElement, 322–327
 - .SetSourceData, 313
- SparklineGroups.Add, 363
- SpecialCells, 64–66, 201
 - selecting with, 277–278
- TwoColorGradient, 330
- TypeText, 437–438
- Union, 62
- UnselectAll, 474
- UserPicture, 329
- Miles, T., 261, 262
- mixed references, 93
- Moala, I., 277
- modeless userforms, 475–476
- Modify method, 343
- modules, 15
- MSubstitute() function, 297–298
- multicolumn list boxes, 486–487
- multidimensional arrays, declaring, 126–127
- MultiPage control, 169–171, 468

N

- Name Manager dialog box, 97
- named ranges, 55, 107–109
- names, 97
 - adding, 98–100
 - checking for the existence of, 106
 - comments, 100
 - deleting, 100
 - formula, 101
 - global, 97–98
 - hiding, 106
 - local, 97–98, 104
 - numbers and, 103
 - reserved, 104–105
 - string, 101–103
 - table, 103–104
 - types of, 101
 - using arrays in, 104
- nesting
 - If statements, 84–86
 - loops, 74–75
- New keyword, referencing a Word application, 430–431
- noncontiguous ranges, 66
- NOW function, 292
- NumberFormat property, 361–362

numbers

- numbers
 - naming, 103
 - retrieving from mixed text, 298–299
 - sorting, 301–303
- NumFilesInCurDir() function, 290–291
- NumUniqueValues() function, 296

O

- Object Browser, 45, 433–434
- object-oriented programming, 28
- object(s), 28
 - assigning a macro to an, 13
 - properties, 31
- objects, 49. *See also* class modules; DAOs (data access objects)
 - Chart, 318
 - ChartFormat, 327
 - color, applying, 329
 - creating new instances of, 431
 - custom
 - creating, 137–139
 - using, 139–140
 - gradients, applying, 330
 - LineFormat, 331
 - patterns, applying, 330
 - properties, 38, 137. *See also* properties
 - Range, 54
 - texture, applying, 329
 - variables, 79–81
 - watches, 44
- Word, 435
 - Document, 435–437
 - Range, 438–441
 - referencing via early binding, 427–430
 - referencing via late binding, 430
 - Selection, 437–438
- ObjectThemeColor property, 329
- Office 365, 314
- Office add-ins, 549, 550
 - adding interactivity, 554–557, 559–560
 - defining, 558–559
 - Hello World, 550, 551–554
 - JavaScript and, 569–570
 - writing to the content or task pane, 569
- OFFSET property, 177
- Offset property, referencing ranges with, 58–60
- OLAP pivot table, filtering by a list of items, 271–273
- Oliver, N., 257

- OneColorGradient method, 330
- OnTime method, 406–407
- Open method, 436
- opening
 - files in a text editor, 550
 - text files
 - delimited files, 417–419
 - fixed-width files, 413–417
- operators
 - JavaScript, 566–567
 - xlFilterIcon, 179–181
- option buttons
 - adding to a userform, 165–166
 - events, 166
- Ozgur, S. M., 549

P

- PageFields parameter, 221
- parameters, 29–31
 - event, 112–113
 - PageFields, 221
- parsing, text files, 260–261
- passing an array, 131–132
- password box, 275–277
- passwords
 - cracking, 514–515
 - macros and, 515
- .Patterned method, 330
- patterns, applying to objects, 330
- picture, filling objects with a, 329
- Pieterse, J. K., 491
- pivot table(s), 211, 574. *See also* Data Model
 - adding model fields to, 246
 - advanced features, 220
 - AutoSort option, 226
 - building, 245–246
 - calculated fields, 249–250
 - calculated items, 250
 - changing the calculation to show percentages, 223–225
 - counting the number of records, 221
 - creating in VBA, 212
 - adding fields to the data area, 214–217
 - configuration, 213–214
 - defining the pivot cache, 213
 - data visualizations, 252–253
 - defining, 220–221
 - determining the size of, 217–220
 - drilling-down, 270–271

- evolution over various Excel versions, 211–212
 - filtering data sets, 229
 - filters
 - conceptual, 230–231
 - manual, 229–230
 - search, 234–236
 - ShowDetail property and, 250
 - slicers, 237–241
 - types, 232–234
 - formatting the intersection of values in, 243–244
 - grouping daily dates to months, quarters, or years, 221–223
 - layout, 250–251
 - OLAP, filtering by a list of items, 271–273
 - producing a static summary, 218–220
 - reports, 217
 - layout, 251
 - multiple value fields, 220–221
 - replicating for every product, 226–229
 - suppressing subtotals for multiple row fields, 252
 - TypeScript and, 255–256
 - Values area
 - adding numeric fields, 246–247
 - eliminating blank cells, 226
 - Power Pivot, 244. *See also* pivot table(s)
 - Power Query. *See also* M
 - Advanced Editor, 387–388
 - building a custom function, 390–393
 - loading large text files to the Data Model, 423–424
 - M and, 385
 - queries
 - AlbumTracks, 395, 397
 - ArtistsAlbums, 394–395, 396–397
 - duplicating to make a new query, 393–395
 - error handling, 403
 - fnGetToken, 392
 - grouping, 397–398
 - placing results on your dashboard, 398–401
 - SearchArtist, 391, 393, 396
 - refreshing Spotify credentials after they expire, 390
 - searching Spotify database for an artist, 388–390
 - storing global variables in a Settings record, 402–403
 - Preserve keyword, 131
 - PresetGradient method, 330
 - PresetTextured method, 329
 - PrintOut method, 437
 - Priority property, 341
 - procedural programming, 28
 - VBA and, 29
 - programming languages. *See also* M
 - BASIC, 2, 28
 - M, 385
 - object-oriented, 28
 - procedural, 28
 - Programming window, VB Editor, 17–19
 - progress indicator, cell, 274–275
 - Project Explorer, 14–15
 - properties, 31, 137
 - .Calculation, 223
 - Cells, referencing ranges with, 57–58
 - ChartColor, 320–321
 - Columns, 61
 - CurrentRegion, selecting ranges with, 63
 - custom, creating, 148–151
 - Excel8CompatibilityMode, 577–578
 - Formula, 352–354
 - Help topics, 38
 - MultiSelect, 163–164
 - NumberFormat, 361–362
 - ObjectThemeColor, 329
 - OFFSET, 177
 - Offset, referencing ranges with, 58–60
 - Priority, 341
 - Range(), 47
 - Resize, 60–61
 - Rows, 61
 - ShowDetail, 250
 - TableRange2, 217–218
 - Type, 341
 - Version, 577
 - Properties window, VB Editor, 15
 - protected password box, 275–277
- ## Q
- QueryClose event, 172
 - querying, in Break mode
 - by hovering, 42
 - using a Watches window, 42–43
 - using the Immediate window, 41–42
 - Quick Access Toolbar, creating a macro button, 11–12
 - Quick Analysis tool, 24–25, 573
- ## R
- R1C1 addressing, 49, 87, 90, 91
 - absolute references, 92–93
 - mixed references, 93

R1C1 addressing

- referring to entire columns or rows, 93
 - relative references, 91–92
 - remembering column numbers associated with column letters, 95–96
 - replacing A1 formulas with, 94–95
 - toggling to, 88
- Range object, 54, 438–441
- Range() property, 47
- ranges, 53
- color scales, adding, 346–347
 - creating from overlapping ranges, 62
 - data bars
 - adding, 342–346
 - using two colors, 352–354
 - finding the first nonzero-length cell in, 296–297
 - icon sets, 347–348
 - specifying, 348–349
 - specifying for subset of a range, 351–352
 - VBA constants, 348–349
 - joining, 62
 - named, 55, 107–109
 - noncontiguous, 66
 - referencing
 - in other worksheets, 55–56
 - relative to another range, 56–57
 - shortcuts, 55
 - using Cells property, 57–58
 - using Columns and Rows properties, 61
 - using the CurrentRegion property, 63
 - using the Offset property, 58–60
 - resizing, 60–61
 - specifying for icons, 350
 - syntax, 54
 - tables, 67
 - worksheets and, 54
- reading
- from an Access database, 451
 - text files with more than 1,048, 576 rows, 420–423
- Record Macro dialog box, 9–10
- recording a macro, 8–9, 17
- absolute references, 19–20
 - Autosum and, 24–25
 - preparations, 16–17
 - relative references, 20–24
 - tips, 25–26
- records, counting, 221
- recursive functions, 290
- RefEdit control, 470–471
- references
- absolute, 19–20, 92–93
 - array, 104
 - mixed, 93
 - relative, 20–24, 91–92
- relationships, creating between tables, 245
- relative references, 20–24, 91–92
- RELS file, 525–526
- removing, add-ins, 545
- renaming, controls, 159
- reports
- fixed-width, converting to a data set, 280–283
 - pivot table, 217
 - layout, 251
 - multiple value fields, 220–221
 - replicating for every product, 226–229
 - suppressing subtotals for multiple row fields, 252
- reserved names, 104–105
- resetting, table format, 278–279
- Resize property, 60–61
- resizing
- ranges, 60–61
 - userforms, 479
- Restore Down icon, 39
- RetrieveNumbers() function, 298–299
- retrieving data
- from arrays, 128–129
 - date and time of last save, 292
 - file names, 173–174
 - from forms, 160–161
 - from mixed text, 298–299
 - permanent date and time, 292–293
 - user ID, 291–292
- returning
- addresses of duplicate maximum values, 304–305
 - hyperlink address, 305
- ReturnMax() function, 304–305
- RGB color, applying to sparklines, 373–374
- ribbon, 517, 572
- adding a control, 520
 - adding custom icons, 528–529
 - buttons, using images on, 527
 - control attributes, 521–525
 - error handling, 529–533
 - macro button, creating, 10–11
 - using Microsoft Office icons, 527–528

Rows property, 61

rules

- above/below average, 340
- top/bottom, 340

running a macro, 10

- with a command button, 534–535
- using a keyboard shortcut, 533–534

S

SaveAs2 method, 436

scaling, sparklines, 366–369

scheduling macros, 407, 408–411

ScrollBar control, 472–473

Search box, AutoFilter and, 179

search filter, 234–236

security, 6

- add-in, 544–545
- Disable All Macros with Notification setting, 8
- trusted locations
 - adding, 6–7
 - enabling macros outside trusted locations, 7–8

Select Case construct, 83–84

- in JavaScript, 564–565
- using on a worksheets, 306

Select method, 38

SelectAll method, 474

selecting, with SpecialCells, 277–278

Selection object, 437–438

SetElement method, 322–327

.SetSourceData method, 313

Shape Fill drop down, 329–330

shared Access database, creating, 449–450

sharing

- LAMBDA functions, 308
- UDFs (user-defined functions), 288

sheet events

- chart, 119–120
- workbook-level, 115–116

SheetExists() function, 289–290

shortcut(s)

- accelerator keys, 483–484
- Ctrl+T, 244
- End+down arrow, 47–48
- for referencing ranges, 55
- running a macro with, 533–534

Show Values As tab of the Value Field Settings dialog box, 223

ShowDetail property, 250

size of pivot tables, determining, 217–220

slicers, 237–243, 574

SLUGIFY.PLUS() function, 309–310

SmartArt, 575

Smith, C., 267

sorting, 300–301

- custom, 273–274
- numeric and alpha characters, 301–303

SparklineGroups.Add method, 363–365

sparklines, 363

- creating, 363–365
- dashboards
 - creating, 378
 - creating individual sparklines in, 379–383
- formatting, 369
 - elements, 374–377
 - using RGB colors, 373–374
 - using theme colors, 369–372
- scaling, 366–369
- win/loss charts, formatting, 377–378

SpecialCells method, 64–66, 201, 277–278

spin buttons, 168–169

Spotify. *See also* Power Query

- creating a developer account, 386–387
- querying the list of songs on an album, 395
- Search API, 388–390
- searching the database for an artist, 388–389

SQL Server, 463–464. *See also* Access

standard modules, 141–142. *See also* class modules

statements

- Case, complex expressions in, 84
- DIM, 79–80
- Exit For, 73–74
- For, 72
- If, nesting, 84–86
- Select Case, 83–84

Step Into feature, 38–40

Stop Recording icon, 10

streamlining code, 46

strings

- JavaScript, 562
- naming, 101–103
- searching for within text, 303–304

styles, chart, 315–318

substituting, multiple characters, 297–298

subtotals, suppressing for multiple row fields, 252

Sullivan, J., 271

SUM function

SUM function, 25
SumColor() function, 295

T

tab stops, setting on userforms, 484–485
tab strips, 468–470
tables. *See also* Access
 Access
 adding fields on the fly, 462–463
 adding on the fly, 461–462
 checking for the existence of, 460
 checking for the existence of a field, 461
 adding to the Data Model, 244–245
 creating a relationship between, 245
 naming, 103–104
 referencing, 67
 resetting the format, 278–279
 tblTransfer, 449
testing, macros, 19, 22–23
text boxes, 160, 161–162
text files
 importing, 413
 with more than 1,048, 576 rows, 419–420
 reading, 420–423
 using Power Query to load, 423–424
 opening
 delimited files, 417–419
 fixed-width files, 413–417
 reading into memory and parsing, 260–261
 writing, 424–425
Text Import Wizard, 35–37
texture, applying to objects, 329
theme colors, applying to sparklines, 369–372. *See also* color(s)
time. *See also* dates
 24-hour, 117–118
 of last save, retrieving, 292
 retrieving, 292–293
Timelines, 241–243
tip text, adding to userforms, 484
TOC2HTML() function, 310–311
ToggleButton control, 471–472
toolbars, UserForm, 465–466
ToolTips, 42
top/bottom rules, 340, 355–356. *See also* conditional formatting
tracking, user changes, 267–268

transparent forms, setting up, 487–488
trapping
 application events, 134–135
 embedded chart events, 136–137
trendline, formatting, 331
trusted locations, 6–7
Tufte, E., 363
TwoColorGradient method, 330
Type property, 341
TypeScript, 576
 comparing to VBA, 253–255
 creating a pivot table, 255–256
TypeText method, 437–438

U

UDFs (user-defined functions), 285
 BookOpen(), 288–290
 building, 286–287
 ColName(), 305–306
 ContainsText(), 303–304
 ConvertWeekDay(), 299–300
 creating, 285
 DateTime(), 292–293
 FirstNonZeroLength(), 296–297
 GetAddress(), 305
 IsEmailValid(), 293–295
 JavaScript, 549
 LastSaved(), 292
 MSubstitute(), 297–298
 NumFilesInCurDir(), 290–291
 NumUniqueValues(), 296
 RetrieveNumbers(), 298–299
 ReturnMax(), 304–305
 sharing, 288
 SortConcat(), 300–301
 SumColor(), 295
 WinUserName(), 291–292
UDTs (user-defined types)
 creating custom properties, 148–151
 declaring, 149
Union method, joining multiple ranges, 62
UnselectAll method, 474
Until clause, 77–79
updates, scheduling, 407
Urtis, T., 265, 270, 274
user ID, retrieving, 291–292
UserForm toolbar, 465–466

- userforms, 153. *See also* controls; forms
 - accelerator keys, 483–484
 - adding option buttons, 165–166
 - calling, 156
 - check boxes, 466–468
 - coloring the active field, 485–486
 - controls
 - adding at runtime, 477–478
 - adding on the fly, 479
 - collections and, 473–475
 - ProgIds, 480
 - creating, 155–156
 - disabling the X for closing, 498
 - events, 157–158
 - graphics, 167
 - hiding, 156–157
 - hyperlinks, 476–477
 - illegal window closing, 172
 - images, adding on the fly, 480–481
 - input boxes, 153–154
 - message boxes, 154–155
 - modeless, 475–476
 - picture catalog, 481–483
 - programming, 157
 - RefEdit control, 470–471
 - resizing on the fly, 479
 - scrollbars, 472–473
 - setting tab stops, 484–485
 - sizing on the fly, 479
 - tab strips, 468–470
 - tip text, adding, 484
 - transparent, 487–488
 - verifying field entry, 171
 - UserPicture method, 329
 - utilities
 - combining workbooks, 262–263
 - converting a fixed-width report to a data set, 280–283
 - copying data to separate worksheets without using Filter, 263–264
 - creating a cell progress indicator, 274–275
 - creating a custom sort order, 273–274
 - creating an Excel state class module, 268–270
 - drilling-down a pivot table, 270–271
 - exporting data to an XML file, 264–265
 - filtering an OLAP pivot table by a list of items, 271–273
 - importing and deleting a CSV file, 260
 - listing files in a directory, 257–260
 - placing a chart in a cell note, 265–267
 - reading a text file into memory and parsing, 260–261
 - resetting a table's format, 278–279
 - selecting with SpecialCells, 277–278
 - separating worksheets into workbooks, 261–262
 - tracking user changes, 267–268
 - using a protected password box, 275–277
 - using VBA Extensibility to add code to new workbooks, 279–280
- ## V
- validating, email addresses, 293–295
 - values
 - duplicate, minimizing, 340
 - unique, counting, 296
 - variables, 49
 - declaring, 14
 - global, 402
 - JavaScript, 561
 - object, 79–81
 - For statement, 72
 - Variant, 128
 - Variant variables, 128
 - VB Editor, 13–14. *See also* Object Browser
 - Break mode, 38–43. *See also* breakpoints
 - converting a file to an add-in, 542–543
 - Help and, 33–34
 - Immediate window, 35, 41–42
 - Programming window, 17–19
 - Project Explorer, 14–15
 - Properties window, 15
 - settings, 14
 - VBA, 31–32
 - barriers to learning
 - macro recorder, 2
 - syntax, 2
 - charts
 - applying a color, 320–321
 - borders, 331
 - combo, 331–334
 - filtering, 322
 - formatting, 327–328
 - line settings, 331
 - map, 335
 - referring to, 318–319
 - SetElement method, 322–327
 - specifying a title, 319–320
 - styles, 315–318

- trendline, 331
- waterfall, 336–337
- collections, 29
- comparing to TypeScript, 253–255
- constants, icon set, 348–349
- data bars, 339, 352–354
 - adding to a range, 342–346
 - color, applying, 344–345
 - using two colors in a range, 352–354
- data visualizations, 340–341
- error handling, 501–503
- Extensibility, 279–280
- formula-based conditions, 196–200
- icon sets, 347–348
 - creating for a subset of a range, 351–352
 - specifying ranges for each icon, 350
- learning, 3
- methods, 28, 29–31
- objects, 28, 31
- Office 365 and, 314
- pivot tables
 - adding fields to the data area, 214–217
 - adding model fields to, 246
 - adding numeric fields to the Values area, 246–247
 - advanced features, 220
 - AutoSort option, 226
 - calculated fields, 249–250
 - calculated items, 250
 - changing the calculation to show percentages, 223–225
 - conceptual filter, 230–231
 - configuration, 213–214
 - creating, 212
 - creating a Data model, 247–249
 - data visualizations, 252–253
 - defining, 220–221
 - defining the pivot cache, 213, 245–246
 - determining the size of, 217–220
 - eliminating blank cells in the Values area, 226
 - filter types, 232–234
 - filtering, 229
 - formatting the intersection of values in, 243–244
 - grouping daily dates to months, quarters, or years, 221–223
 - layout, 250–251
 - manual filter, 229–230
 - replicating the report for every product, 226–229

- reports, 251
- search filter, 234–236
- ShowDetail property, 250
- slicers, 237–241
- Timelines, 241–243
- procedural programming and, 29
- simplifying Power Query queries, 396
 - AlbumTracks, 397
 - ArtistsAlbums, 396–397
 - SearchArtist, 396
- syntax, 38
- verifying, field entry, 171
- Version property, 577
- Visual Basic, BASIC and, 2
- VLOOKUP function, 107

W

- watches, 42–43
 - on objects, 44
 - setting a break point, 43
- waterfall charts, creating, 336–337
- Wei, J., 273
- While clause, 77–79
- While.Wend loops, 79
- Windows API, 491. *See also* API (Application Programming Interface)
- win/loss charts, formatting, 377–378
- WinUserName() function, 291–292
- With...End With block, 50
- Word
 - bookmarks, 441–442
 - constants, retrieving the real value of, 433–434
 - using the Object Browser, 433–434
 - using the Watches window, 433
 - controlling form fields in, 443–444
- documents
 - closing, 436–437
 - creating, 435–436
 - opening, 436
 - printing, 437
 - saving changes to, 436
 - templates, 436
- objects, 435
 - Document, 435–437
 - Range, 438–441
 - Selection, 437–438

- referencing an existing instance of, 431–432
- referencing objects via early binding, 427–430
- referencing objects via late binding, 430
- workbooks, 2
 - adding code to, 279–280
 - checking for existing sheets in, 289–290
 - code, 546–547
 - combining, 262–263
 - converting to an add-in, 540–541
 - events, 113–115
 - open, checking for, 288–289
 - ribbon, 517–518
 - sheet events, 115–116
 - themes, 369
- worksheets
 - copying data to without using Filter, 263–264
 - events, 116–117
 - ranges and, 54, 55–56

- Select Case construct, 306
- separating into workbooks, 261–262
- writing, text files, 424–425

X-Y-Z

- xlFilterCopy, 202
 - combining multiple techniques, 205–209
 - copying a subset of columns and reordering, 203–205
 - copying all columns, 202–203
- xlFilterIcon operator, 179–181
- XLOOKUP function, named ranges, 107–109
- .xlsm file type, 5, 525
- XML
 - adding a control to a ribbon, 520
 - creating a tab and a group, 519
 - customui folder, 518–519
 - defining an Office add-in, 558–559
 - exporting data to a file, 264–265