

IDŹ DO

PRZYKŁADOWY ROZDZIAŁ



SPIS TREŚCI

KATALOG KSIĄŻEK

KATALOG ONLINE

ZAMÓW DRUKOWANY KATALOG

TWÓJ KOSZYK

DODAJ DO KOSZYKA

CENNIK I INFORMACJE

ZAMÓW INFORMACJE
O NOWOŚCIACH

ZAMÓW CENNIK

CZYTELNIA

FRAGMENTY KSIĄŻEK ONLINE

Excel 2007. Język VBA i makra. Rozwiązania w biznesie

Autor: Bill Jelen, Tracy Syrstad

Tłumaczenie: Radosław Meryk

ISBN: 978-83-246-1459-2

Tytuł oryginału: [VBA and Macros for Microsoft Office Excel 2007 \(Business Solutions\)](#)

Format: 170x230, stron: 720



Poznaj techniki tworzenia makr w Excelu

- Do czego można wykorzystać makra?
- W jaki sposób zaimplementować w VBA obsługę zdarzeń?
- Jak korzystać w makrach z zaawansowanych możliwości Excela?

Dla większości użytkowników praca z Excelem kojarzy się ze żmudnym wpisywaniem dziesiątek wartości, szukaniem właściwej funkcji i zastanawianiem się nad skonstruowaniem odpowiedniej formuły obliczeniowej. Makropolecenia i język VBA, za pomocą których można zdecydowanie przyspieszyć i usprawnić pracę z arkuszem kalkulacyjnym, nadal są stosunkowo rzadko wykorzystywane. Dlaczego? Użytkownicy próbujący tworzyć własne makra zwykle borykają się z problemami, których wyjaśnień nie znajdują w dokumentacji Excela, przez co wydają się nie do pokonania. Warto jednak poświęcić czas na opanowanie Rejestratora Makr i edytora VBA, ponieważ pozwolą usprawnić i przyspieszyć pracę z Excelem.

Jeśli poszukujesz książki, dzięki której makra i VBA w Excelu już nigdy nie będą dla Ciebie zagadką, sięgnij po „Excel 2007. Język VBA i makra. Rozwiązania w biznesie”. Znajdziesz w niej wszystkie informacje niezbędne do tego, aby tworzyć własne makra i programować w VBA. Dowiesz się, do czego można zastosować makra, jak je budować i modyfikować. Poznasz język VBA, nauczysz się konstruować rozbudowane raporty i formularze, obsługiwać zdarzenia i tworzyć tabele przestawne. Zaczyniesz wykorzystywać zaawansowane możliwości Excela, takie jak obsługa formatu XML, pobieranie danych z zewnętrznych źródeł, komunikacja z internetem i wykorzystanie Windows API. Przeczytasz także o wyszukiwaniu i usuwaniu błędów w aplikacjach VBA.

- Podstawowe elementy VBA
- Odwołania do zakresów
- Korzystanie z funkcji
- Formuły tablicowe
- Obsługa zdarzeń
- Tworzenie interfejsów użytkownika
- Wybieranie danych z arkuszy
- Tabele przestawne
- Operacje na plikach
- Pobieranie danych ze stron internetowych
- Obsługa formatu XML
- Korzystanie z Windows API
- Wykrywanie błędów

Twórz własne makra i przekonaj się, jak szybka może być praca w Excelu



Spis treści

Wprowadzenie	25
Korzystanie z języka VBA	25
Zawartość tej książki	25
Przyszłość języka VBA i windowsowych wersji Excela	28
Elementy specjalne i konwencje typograficzne	29
Pliki z kodem	30
Następne kroki	30
1 Uwolnij możliwości Excela, korzystając z VBA	31
Możliwości Excela	31
Podstawowe przeszkody	31
Rejestrator makr nie działa!	31
Visual Basic nie jest podobny do BASIC-a	32
Dobre wieści — nauczenie się języka VBA nie jest trudne	32
Doskonała wiadomość — Excel z językiem VBA jest wart wysiłków włożonych w jego naukę	33
Znajomość narzędzi — wstążka Deweloper	33
Bezpieczeństwo makr	34
Dodawanie zaufanej lokalizacji	35
Zastosowanie ustawień makr w celu zezwolenia na wykorzystanie makr poza zaufanymi lokalizacjami	36
Wykorzystanie opcji Wyłącz wszystkie makra i wyświetl powiadomienie	37
Przegląd wiadomości na temat rejestrowania, zapisywania i uruchamiania makr	38
Wypełnianie okna dialogowego Rejestrowanie makra	38
Uruchamianie makr	39
Tworzenie przycisku makra	40
Przypisywanie makra do formantu formularza, pola tekstowego lub figury	41
Nowe typy plików w Excelu 2007	42
Edytor Visual Basic	44
Ustawienia edytora VB	45
Eksplorator projektu	45
Okno Properties	47
Niedoskonałości rejestratora makr	47
Przygotowanie do rejestracji makra	47
Rejestrowanie makra	48
Analiza kodu w oknie programowania	49

Uruchomienie tego samego makra innego dnia generuje niewłaściwe wyniki	51
Możliwe rozwiązania: wykorzystywanie odwołań względnych podczas rejestrowania	52
Następne kroki: rozwiązaniem jest nauka języka VBA	56
2 Jeśli to jest BASIC, to dlaczego nie wygląda znajomo?	57
Nie rozumiem tego kodu	57
Części mowy języka VBA	58
Czy język VBA jest naprawdę taki trudny? Nie!	61
Pliki pomocy VBA — używanie klawisza F1 do wyszukiwania potrzebnych informacji	62
Korzystanie z tematów pomocy	63
Analiza kodu zarejestrowanego makra — korzystanie z edytora VB i systemu pomocy	65
Parametry opcjonalne	65
Zdefiniowane stałe	66
Właściwości mogą zwracać obiekty	71
Wykorzystanie narzędzi debugowania do analizy zarejestrowanego kodu	72
Wykonywanie kodu krok po kroku	72
Więcej opcji debugowania — pułapki	74
Cofanie się lub przesuwanie w przód w kodzie	75
Uruchamianie grupy instrukcji bez trybu krokowego	76
Zapytania podczas krokowego uruchamiania kodu	76
Wykorzystywanie czujek do ustawiania pułapek	81
Wykorzystanie czujki w odniesieniu do obiektu	81
Opis wszystkich obiektów, metod i właściwości	82
Pięć prostych wskazówek dotyczących usprawniania zarejestrowanego kodu	84
Wskazówka 1.: Nie należy niczego zaznaczać	85
Wskazówka 2.: Przeszukiwanie zakresu od dołu w celu odnalezienia ostatniego wiersza	86
Wskazówka 3.: Używanie zmiennych w celu uniknięcia „kodowania na sztywno” wierszy i formuł	87
Wskazówka 4.: Kopiowanie i wklejanie w pojedynczej instrukcji	87
Wskazówka 5.: Wykorzystywanie konstrukcji With...End With w przypadku wykonywania tych samych działań w odniesieniu do tej samej komórki lub zakresu komórek	88
Podsumowanie — usprawnienie zarejestrowanego kodu	88
Modyfikowanie zarejestrowanego kodu	88
Następne kroki	91
3 Odwoływanie się do zakresów	93
Obiekt Range	93
Wykorzystywanie lewego górnego i dolnego prawego narożnika zaznaczonego obszaru do określania zakresu	94

Zakresy identyfikowane przez nazwy	94
Skrótowny sposób odwoływania się do zakresów	95
Odwoływanie się do zakresów w innych arkuszach	95
Odwoływanie się do zakresu względem innego zakresu	96
Wykorzystywanie właściwości Cells do zaznaczania zakresu	97
Wykorzystanie właściwości Cells w odniesieniu do właściwości Range	98
Wykorzystywanie właściwości Offset do odwoływania się do zakresu	98
Wykorzystanie właściwości Resize do zmiany rozmiaru zakresu	100
Wykorzystanie właściwości Columns i Rows do definiowania zakresu	101
Wykorzystywanie metody Union do łączenia wielu zakresów	102
Wykorzystywanie metody Intersect do tworzenia nowego zakresu na podstawie zakresów nakładających się na siebie	102
Wykorzystanie funkcji ISEMPY do sprawdzania, czy komórka jest pusta	102
Wykorzystanie właściwości CurrentRegion do zaznaczania zakresu danych	103
Użycie metody SpecialCells do zaznaczania określonych komórek	104
Wykorzystanie kolekcji Areas do zwracania nieciągłego zakresu	106
Odwołania do tabel	106
Następne kroki	107
4 Funkcje definiowane przez użytkowników	109
Tworzenie funkcji	109
Funkcje użytkownika — przykład i objaśnienie	110
Współdzielenie funkcji użytkownika	111
Przydatne funkcje użytkownika w Excelu	112
Wyświetlanie w komórce nazwy bieżącego skoroszytu	112
Wyświetlanie w komórce nazwy bieżącego skoroszytu wraz ze ścieżką dostępu	113
Sprawdzenie, czy skoroszyt jest otwarty	113
Sprawdzenie, czy w otwartym skoroszytcie istnieje arkusz	113
Zliczanie liczby skoroszytów w katalogu	114
Odczytywanie zmiennej USERID	115
Odczytywanie daty i godziny ostatniego zapisania skoroszytu	116
Odczytywanie trwałej wartości daty i godziny	117
Sprawdzanie poprawności adresu e-mail	118
Sumowanie komórek na podstawie wewnętrznego koloru	119
Zliczanie unikatowych wartości	120
Usuwanie duplikatów z zakresu	121
Znalezienie w zakresie pierwszej komórki o nierównej długości	123
Zastępowanie wielu znaków	124
Odczytanie liczb z tekstu składającego się z liczb i liter	125

Konwersja numerów tygodni na daty	126
Rozdzielanie tekstu	126
Sortowanie z konkatenacją	127
Sortowanie cyfr i liter	129
Wyszukiwanie ciągu w tekście	130
Odwrocenie zawartości komórki	131
Więcej niż jedna wartość maksymalna	131
Zwracanie adresu hiperłącza	132
Zwrócenie litery kolumny na podstawie adresu komórki	133
Statyczne liczby losowe	133
Korzystanie z konstrukcji Select Case w arkuszu	134
Następne kroki	135
5 Pętle i sterowanie przepływem	137
Pętla For . . . Next	137
Korzystanie ze zmiennych w instrukcji For	140
Wariacje na temat pętli For . . . Next	141
Wcześniejsze zakończenie pętli w przypadku spełnienia warunku	142
Zagnieżdżanie pętli wewnątrz innej pętli	142
Pętla Do	143
Wykorzystanie klauzuli While lub Until wewnątrz pętli Do	147
Pętla While . . . Wend	148
Pętla języka VBA: For Each	149
Zmienne obiektowe	149
Przetwarzanie w pętli wszystkich plików w katalogu	151
Sterowanie przepływem: korzystanie z konstrukcji If . . . Then . . . Else i Select Case	152
Proste sterowanie przepływem: If . . . Then . . . Else	153
Warunki	153
Konstrukcja If . . . Then . . . End If	153
Decyzje typu albo — albo: If . . . Then . . . Else . . . End If	154
Wykorzystanie konstrukcji If . . . Else If . . . End If do sprawdzania wielu warunków	154
Wykorzystanie struktury Select Case . . . End Select do sprawdzania wielu warunków	155
Złożone wyrażenia w instrukcjach Case	156
Zagnieżdżanie instrukcji If	156
Następne kroki	158
6 Formuły w stylu W1K1	159
Odwołania do komórek: porównanie stylu A1 z W1K1	159
Przełączanie Excela w celu wyświetlania odwołań w stylu W1K1	160

Cudowna moc formuł Excela	161
Wprowadź formułę raz i skopiuj ją 1000 razy	161
Sekret? Nie ma w tym nic nadzwyczajnego	162
Wprowadzanie formuł w stylu A1 w porównaniu z wprowadzaniem formuł w stylu W1K1 w języku VBA	163
Objaśnienie stylu odwołań R1C1	164
Zastosowanie stylu W1K1 dla odwołań względnych	164
Zastosowanie stylu W1K1 dla odwołań bezwzględnych	165
Zastosowanie stylu W1K1 dla odwołań mieszanych	166
Odwoływanie się do całych kolumn lub wierszy z wykorzystaniem stylu W1K1	166
Zastępowanie wielu formuł A1 pojedynczą formułą W1K1	167
Zapamiętywanie numerów kolumn powiązanych z literami kolumn	168
Formatowanie warunkowe — obowiązkowy styl W1K1	169
Konfiguracja formatowania warunkowego z wykorzystaniem interfejsu użytkownika	170
Konfigurowanie formatów warunkowych w języku VBA	171
Identyfikacja wiersza z największą wartością w kolumnie G	173
Formuły tablicowe wymagają stylu W1K1	174
Następne kroki	175
7 Co nowego w Excelu 2007 i co się zmieniło?	177
Jeśli coś zmieniło się w warstwie frontonu, zmieniło się również w VBA	177
Wstążka	177
Wykresy	177
Tabele przestawne	178
Formatowanie warunkowe	178
Tabele	179
Sortowanie	179
SmartArt	180
Rejestrator makr nie rejestruje operacji, które rejestrował we wcześniejszych wersjach Excela	180
Nowe obiekty i metody	182
Tryb zgodności	183
Version	184
Excel8CompatibilityMode	184
Następne kroki	185
8 Definiowanie nazw i wykonywanie z nimi operacji za pomocą języka VBA	187
Nazwy w Excelu	187
Nazwy globalne a nazwy lokalne	187
Dodawanie nazw	188

Usuwanie nazw	190
Dodawanie komentarzy	191
Typy nazw	192
Formuły	192
Ciągi znaków	193
Liczby	194
Tabele	195
Wykorzystanie tablic w nazwach	195
Nazwy zarezerwowane	196
Ukrywanie nazw	197
Sprawdzanie, czy określona nazwa istnieje	198
Wykorzystanie zakresów identyfikowanych przez nazwy do wykonywania funkcji VLOOKUP	198
Następne kroki	200
9 Programowanie zdarzeń	201
Poziomy zdarzeń	201
Wykorzystywanie zdarzeń	202
Parametry zdarzeń	203
Uaktywnianie zdarzeń	203
Zdarzenia związane ze skoroszytem	203
Workbook_Activate()	203
Workbook_Deactivate()	204
Workbook_Open()	204
Workbook_BeforeSave(ByVal SaveAsUI As Boolean, Cancel As Boolean)	205
Workbook_BeforePrint(Cancel As Boolean)	205
Workbook_BeforeClose(Cancel As Boolean)	206
Workbook_NewSheet(ByVal Sh As Object)	207
Workbook_WindowResize(ByVal Wn As Window)	207
Workbook_WindowActivate(ByVal Wn As Window)	207
Workbook_WindowDeactivate(ByVal Wn As Window)	207
Workbook_AddInInstall()	208
Workbook_AddInUninstall	208
Workbook_SheetActivate(ByVal Sh As Object)	208
Workbook_SheetBeforeDoubleClick(ByVal Sh As Object, ByVal Target As Range, Cancel As Boolean)	208
Workbook_SheetBeforeRightClick(ByVal Sh As Object, ByVal Target As Range, Cancel As Boolean)	208
Workbook_SheetCalculate(ByVal Sh As Object)	209
Workbook_SheetChange(ByVal Sh As Object, ByVal Target As Range)	209
Workbook_Sync(ByVal SyncEventType As Office.MsoSyncEventType)	209
Workbook_SheetDeactivate(ByVal Sh As Object)	209

Workbook_SheetFollowHyperlink(ByVal Sh As Object, ByVal Target As Hyperlink)	209
Workbook_SheetSelectionChange(ByVal Sh As Object, ByVal Target As Range)	209
Workbook_PivotTableCloseConnection(ByVal Target As PivotTable)	210
Workbook_PivotTableOpenConnection(ByVal Target As PivotTable)	210
Workbook_RowsetComplete(ByVal Description As String, ByVal Sheet As String, ByVal Success As Boolean)	210
Zdarzenia związane z arkuszem	210
Worksheet_Activate()	210
Worksheet_Deactivate()	210
Worksheet_BeforeDoubleClick(ByVal Target As Range, Cancel As Boolean)	211
Worksheet_BeforeRightClick(ByVal Target As Range, Cancel As Boolean)	211
Worksheet_Calculate()	211
Worksheet_Change(ByVal Target As Range)	213
Worksheet_SelectionChange(ByVal Target As Range)	213
Worksheet_FollowHyperlink(ByVal Target As Hyperlink)	214
Szybkie wprowadzanie do komórki czasu w formacie militarnym	214
Zdarzenie dotyczące wykresów	215
Wykresy osadzone	215
Chart_Activate()	216
Chart_BeforeDoubleClick(ByVal ElementID As Long, ByVal Arg1 As Long, ByVal Arg2 As Long, Cancel As Boolean)	216
Chart_BeforeRightClick(Cancel As Boolean)	216
Chart_Calculate()	216
Chart_Deactivate()	216
Chart_MouseDown(ByVal Button As Long, ByVal Shift As Long, ByVal x As Long, ByVal y As Long)	217
Chart_MouseMove(ByVal Button As Long, ByVal Shift As Long, ByVal x As Long, ByVal y As Long)	217
Chart_MouseUp(ByVal Button As Long, ByVal Shift As Long, ByVal x As Long, ByVal y As Long)	217
Chart_Resize()	218
Chart_Select(ByVal ElementID As Long, ByVal Arg1 As Long, ByVal Arg2 As Long)	218
Chart_SeriesChange(ByVal SeriesIndex As Long, ByVal PointIndex As Long)	219
Chart_DragOver()	219
Chart_DragPlot()	219
Zdarzenia poziomu aplikacji	219
AppEvent_AfterCalculate()	220
AppEvent_NewWorkbook(ByVal Wb As Workbook)	220
AppEvent_SheetActivate (ByVal Sh As Object)	220
AppEvent_SheetBeforeDoubleClick(ByVal Sh As Object, ByVal Target As Range, Cancel As Boolean)	221
AppEvent_SheetBeforeRightClick(ByVal Sh As Object, ByVal Target As Range, Cancel As Boolean)	221

AppEvent_SheetCalculate(ByVal Sh As Object)	221
AppEvent_SheetChange(ByVal Sh As Object, ByVal Target As Range)	221
AppEvent_SheetDeactivate(ByVal Sh As Object)	221
AppEvent_SheetFollowHyperlink(ByVal Sh As Object, ByVal Target As Hyperlink)	221
AppEvent_SheetSelectionChange(ByVal Sh As Object, ByVal Target As Range)	221
AppEvent_WindowActivate(ByVal Wb As Workbook, ByVal Wn As Window)	222
AppEvent_WindowDeactivate(ByVal Wb As Workbook, ByVal Wn As Window)	222
AppEvent_WindowResize(ByVal Wb As Workbook, ByVal Wn As Window)	222
AppEvent_WorkbookActivate(ByVal Wb As Workbook)	222
AppEvent_WorkbookAddInInstall(ByVal Wb As Workbook)	222
AppEvent_WorkbookAddInUninstall(ByVal Wb As Workbook)	222
AppEvent_WorkbookBeforeClose(ByVal Wb As Workbook, Cancel As Boolean)	223
AppEvent_WorkbookBeforePrint(ByVal Wb As Workbook, Cancel As Boolean)	223
AppEvent_WorkbookBeforeSave(ByVal Wb As Workbook, ByVal SaveAsUI As Boolean, Cancel As Boolean)	223
AppEvent_WorkbookNewSheet(ByVal Wb As Workbook, ByVal Sh As Object)	223
AppEvent_WorkbookOpen(ByVal Wb As Workbook)	223
AppEvent_WorkbookPivotTableCloseConnection(ByVal Wb As Workbook, ByVal Target As PivotTable)	224
AppEvent_WorkbookPivotTableOpenConnection(ByVal Wb As Workbook, ByVal Target As PivotTable)	224
AppEvent_WorkbookRowsetComplete(ByVal Wb As Workbook, ByVal Description As String, ByVal Sheet As String, ByVal Success As Boolean)	224
AppEvent_WorkbookSync(ByVal Wb As Workbook, ByVal SyncEventType As Office.MsoSyncEventType)	224
Następne kroki	224
10 Obiekty UserForm — wprowadzenie	225
Metody interakcji z użytkownikami	225
Pola tekstowe	225
Okna informacyjne	226
Tworzenie obiektów UserForm	226
Wywoływanie i ukrywanie obiektów UserForm	228
Programowanie obiektów UserForm	228
Zdarzenia dotyczące obiektu UserForm	228
Programowanie formantów	230
Dodawanie formantów do istniejących formularzy	231
Wykorzystywanie podstawowych formantów formularzy	231
Wykorzystanie etykiet, pól tekstowych i przycisków poleceń	232
Decydowanie o użyciu w formularzach pól listy lub pól kombi	234
Dodawanie przycisków opcji w oknie UserForm	235
Dodawanie elementów graficznych na formularzach UserForm	238

Wykorzystanie formantu pokręta w oknach UserForm	239
Wykorzystanie formantu MultiPage do łączenia formularzy	240
Weryfikacja danych wprowadzanych w polach	243
Nieprawidłowe zamykanie okien	243
Pobieranie nazwy pliku	245
Następne kroki	246
11 Tworzenie wykresów	247
Obsługa wykresów w Excelu 2007	247
Kodowanie nowych własności obsługi wykresów w Excelu 2007	248
Odwoływanie się do wykresów i obiektów wykresów w kodzie VBA	249
Tworzenie wykresu	249
Określanie rozmiaru i lokalizacji wykresu	250
Odwoływanie się do specyficznego wykresu	251
Rejestrowanie poleceń z poziomu wstążek Układ lub Projektowanie	253
Określanie wbudowanego typu wykresu	253
Określanie szablonu typu wykresu	259
Zmiana układu lub stylu wykresu	259
Wykorzystanie obiektu SetElement do emulowania zmian na wstążce Układ	261
Zmiana tytułu wykresu za pomocą VBA	267
Emulowanie zmian na wstążce Formatowanie	267
Wykorzystanie metody Format w celu uzyskania dostępu do nowych opcji formatowania	268
Wykorzystanie okna Watch do wyświetlania ustawień obiektów	285
Wykorzystanie okna Watch do wyświetlania ustawień obrotu	288
Tworzenie zaawansowanych wykresów	289
Tworzenie rzeczywistych wykresów giełdowych typu Otwarcie-maks.-min.-zamknięcie	289
Tworzenie koszyków dla wykresu częstości	291
Tworzenie skumulowanego wykresu warstwowego	294
Eksportowanie wykresów jako obiektów graficznych	299
Tworzenie dynamicznych wykresów w formularzach UserForm	300
Tworzenie wykresów przestawnych	301
Następne kroki	304

12 Wykorzystanie polecenia Filtr zaawansowany do wydobywania danych	305
Korzystanie z polecenia Filtr zaawansowany jest łatwiejsze w VBA niż w Excelu	305
Wykorzystanie polecenia Filtr zaawansowany do wyodrębniania listy niepowtarzalnych wartości	306
Wyodrębnianie listy niepowtarzalnych wartości z poziomu interfejsu użytkownika	307
Wyodrębnienie listy niepowtarzalnych wartości za pomocą kodu VBA	308
Tworzenie niepowtarzalnych kombinacji dwóch lub większej liczby pól	312
Wykorzystanie polecenia Filtr zaawansowany z zakresem kryteriów	314
Łączenie wielu kryteriów z wykorzystaniem logicznego operatora OR	316
Łączenie wielu kryteriów z wykorzystaniem logicznego operatora AND	316
Inne, nieco bardziej złożone zakresy kryteriów	316
Najbardziej złożone kryteria — zastępowanie listy wartości przez warunek utworzony jako wynik formuły	317
Wykorzystanie w poleceniu Filtr zaawansowany opcji filtrowania na miejscu	324
Brak rekordów spełniających kryteria podczas wykorzystywania opcji filtrowania listy na miejscu	325
Wyświetlanie wszystkich rekordów po wykonaniu filtrowania listy na miejscu	326
Wykorzystanie filtrowania na miejscu z opcją wyświetlania tylko unikatowych rekordów	326
Prawdziwy „koń pociągowy”: xlFilterCopy z wszystkimi rekordami zamiast tylko niepowtarzalnych	326
Kopiowanie wszystkich kolumn	327
Kopiowanie podzbioru kolumn i zmiana ich kolejności	328
Wykorzystanie dwóch rodzajów filtrów zaawansowanych w celu utworzenia raportu dla każdego klienta	330
Wykorzystanie autofiltra	334
Włączanie funkcji Autofiltr w kodzie	335
Wyłączenie kilku list rozwijanych autofiltra	335
Filtrowanie kolumn z wykorzystaniem autofiltrów	336
Wybieranie za pomocą filtra wielu wartości	337
Wybieranie dynamicznego zakresu dat za pomocą autofiltrów	338
Filtrowanie na podstawie koloru lub ikony	339
Wykorzystanie autofiltra do skopiowania wszystkich rekordów z następnego tygodnia	341
Następne kroki	342
13 Wykorzystanie języka VBA do tworzenia tabel przestawnych	343
Wprowadzenie w tematykę tabel przestawnych	343
Wersje tabel przestawnych	344
Nowości w Excelu 2007	344

Tworzenie prostych tabel przestawnych w środowisku interfejsu użytkownika Excela	347
Nowe własności tabel przestawnych Excela 2007	349
Tworzenie tabel przestawnych w języku VBA Excela	351
Definiowanie bufora tabeli przestawnej	352
Tworzenie i konfigurowanie tabeli przestawnej	352
Obliczanie sumy zamiast zliczania wartości	353
Dlaczego nie można przesuwać lub modyfikować fragmentów raportu przestawnego?	356
Określanie rozmiaru zakończonej tabeli przestawnej	356
Tworzenie raportu z obrotów według produktu	359
Eliminowanie pustych komórek w obszarze wartości	361
Zapewnienie wykorzystania układu tabelarycznego	362
Zarządzanie porządkiem sortowania za pomocą opcji automatycznego sortowania	362
Zmiana domyślnego formatu liczb	362
Wyłączanie sum częściowych dla tabel z wieloma polami wierszy	363
Wyłączanie sumy końcowej dla wierszy	364
Rozwiązywanie dodatkowych problemów podczas tworzenia raportu w ostatecznej postaci	364
Utworzenie nowego skoroszytu do przechowywania raportu	364
Utworzenie zestawienia w pustym arkuszu	365
Wypełnianie tabeli w widoku konspektu	366
Ostateczne formatowanie	367
Dodanie sum częściowych	367
Ostateczna wersja raportu	369
Rozwiązywanie problemów dotyczących raportów z dwoma lub większą liczbą pól danych	372
Wylizane pola danych	374
Wylizane elementy	377
Podsumowania pól daty z wykorzystaniem grupowania	379
Wykorzystanie metody Group w kodzie VBA	380
Grupowanie według tygodnia	383
Mierzenie cyklu obsługi zamówień poprzez grupowanie dwóch pól daty	385
Zaawansowane techniki obsługi tabel przestawnych	388
Wykorzystanie funkcji Autopokazywania do tworzenia podsumowań poglądowych dla kierownictwa	388
Wykorzystanie właściwości ShowDetail do filtrowania zestawu rekordów	391
Tworzenie raportów dla każdego regionu lub modelu	393
Ręczne filtrowanie dwóch lub większej liczby elementów pola tabeli przestawnej	396
Ręczne zarządzanie porządkiem sortowania	397
Korzystanie z funkcji Suma, Średnia, Licznik, Min, Max i innych	398

Tworzenie raportów procentowych	399
Procent sumy końcowej	400
Procentowy wzrost w stosunku do poprzedniego miesiąca	400
Procent określonej pozycji	400
Suma narastająco	401
Wykorzystanie nowych własności tabel przestawnych w Excelu 2007	401
Korzystanie z nowych filtrów	402
Stosowanie stylu tabeli	404
Modyfikowanie układu tabeli przestawnej z poziomu wstążki Projektowanie	405
Wizualizacja danych	405
Następne kroki	408
14 Zaawansowane możliwości Excela	409
Operacje na plikach	409
Wyświetlanie listy plików w katalogu	409
Importowanie plików w formacie CSV	412
Wczytanie całego pliku CSV do pamięci w celu jego przetwarzania	413
Łączenie i rozdzielanie skoroszytów	414
Rozdzielanie arkuszy na osobne skoroszyty	414
Łączenie skoroszytów	415
Filtrowanie i kopiowanie danych do osobnych arkuszy	416
Eksport danych do Worda	417
Korzystanie z komentarzy w komórkach	418
Lista komentarzy	418
Zmiana rozmiaru komentarzy	420
Zmiana rozmiaru komentarzy poprzez wyśrodkowanie	421
Umieszczenie wykresu w komentarzu	422
Narzędzia, których celem jest zrobienie pozytywnego wrażenia na klientach	424
Wykorzystanie formatowania warunkowego do podświetlenia wybranej komórki	424
Wyróżnienie wybranej komórki bez użycia formatowania warunkowego	425
Niestandardowe transponowanie danych	427
Zaznaczanie (anulowanie zaznaczenia) nieciągłego zakresu komórek	429
Techniki dla ekspertów języka VBA	431
Rozwijane tabele przestawne	431
Szybka konfiguracja stron	433
Obliczanie czasu wykonania kodu	436
Niestandardowy porządek sortowania	437
Wskaźnik postępu wykonywania operacji w komórce	438
Chronione pole do wprowadzania hasła	439
Zmiana wielkości liter	441
Zaznaczanie komórek za pomocą metody SpecialCells	443
Menu prawego przycisku myszy dla obiektów ActiveX	443

Interesujące aplikacje	445
Historyczne kursy akcji (funduszy)	445
Wykorzystanie rozszerzalności języka VBA w celu dodawania kodu do nowych skoroszytów	446
Następne kroki	448
15 Wizualizacja danych i formatowanie warunkowe	449
Wprowadzenie do wizualizacji danych	449
Nowe metody i właściwości języka VBA służące do wizualizacji danych	451
Dodawanie pasków danych do zakresu	451
Wykorzystanie skali kolorów w zakresach	454
Wykorzystywanie zestawów ikon w zakresach	455
Określanie zestawu ikon	456
Określanie przedziałów dla każdej z ikon	456
Sztuczki wizualizacyjne	458
Tworzenie zestawu ikon dla podzbioru zakresu	458
Używanie dwóch kolorów pasków danych w zakresie	460
Wykorzystywanie innych metod formatowania warunkowego	463
Formatowanie komórek zawierających wartości powyżej lub poniżej średniej	463
Formatowanie komórek zawierających 10 pierwszych lub 5 ostatnich elementów	463
Formatowanie niepowtarzalnych wartości lub duplikatów	464
Formatowanie komórek na podstawie ich wartości	466
Formatowanie komórek zawierających tekst	466
Formatowanie komórek zawierających daty	467
Formatowanie komórek zawierających puste wartości lub błędy	467
Wykorzystanie formuł w celu określenia komórek do formatowania	467
Wykorzystanie nowej właściwości NumberFormat	469
Następne kroki	470
16 Czytanie informacji ze stron WWW i zapisywanie informacji do internetu	471
Pobieranie danych z internetu	471
Ręczne tworzenie kwerend webowych i odświeżanie ich za pomocą VBA	472
Wykorzystanie języka VBA do aktualizacji zdefiniowanych kwerend sieci Web	474
Tworzenie nowej kwerendy sieci Web za pomocą języka VBA	474
Wykorzystanie strumieni danych	477
Wykorzystanie metody Application.OnTime do okresowego analizowania danych	478
Zaplanowane procedury wymagają trybu Gotowy	479
Definiowanie okna czasowego dla aktualizacji	479
Anulowanie makra zaplanowanego wcześniej	479

Zamknięcie Excela powoduje anulowanie wszystkich oczekujących zaplanowanych makr	480
Planowanie uruchomienia makra za x minut w przyszłości	480
Zaplanowanie słownego przypomnienia	481
Planowanie uruchamiania makra co dwie minuty	482
Publikowanie danych na stronach WWW	483
Wykorzystanie języka VBA do tworzenia niestandardowych stron WWW	485
Wykorzystanie Excela w roli systemu zarządzania zawartością	486
Premia: FTP z Excela	490
Ufanie zawartości pochodzącej z internetu	490
Następne kroki	492
17 Obsługa XML w Excelu 2007	493
Czym jest XML?	493
Proste reguły języka XML	494
Uniwersalny format plików	495
XML jako nowy uniwersalny format plików	495
Alfabet języka XML	496
Wykorzystanie XML jako typu plików w aplikacjach Microsoft	498
W jaki sposób Excel 2007 zapisuje skróty w formacie XML?	499
Wykorzystanie danych XML z serwisu Amazon.com	500
Następne kroki	502
18 Automatyzacja Worda	503
Wczesne wiązanie	503
Błąd kompilacji: Nie można znaleźć obiektu lub biblioteki	506
Późne wiązanie	506
Tworzenie obiektów i odwoływanie się do nich	507
Słowo kluczowe New	507
Funkcja CreateObject	508
Funkcja GetObject	508
Wykorzystanie stałych	509
Wykorzystanie okna Watch do odczytywania rzeczywistych wartości stałych	510
Wykorzystanie przeglądarki obiektów do odczytywania rzeczywistych wartości stałych	510
Obiekty Worda	511
Obiekt Document	512
Obiekt Selection	514
Obiekt Range	515
Zakładki	519
Zarządzanie polami formularzy w Wordzie	521
Następne kroki	524

19 Tablice	525
Deklaracje tablic	525
Tablice wielowymiarowe	526
Wypełnianie tablic danymi	527
Opróżnianie tablic	528
Tablice mogą ułatwić operowanie danymi, ale czy to wszystko?	530
Tablice dynamiczne	531
Przekazywanie tablic jako argumentów	533
Następne kroki	533
20 Przetwarzanie plików tekstowych	535
Importowanie danych z plików tekstowych	535
Importowanie danych z plików tekstowych zawierających mniej niż 1 048 576 wierszy	535
Importowanie danych z plików tekstowych zawierających więcej niż 1 048 576 wierszy	543
Zapisywanie danych do plików tekstowych	547
Następne kroki	548
21 Wykorzystanie Accessa w celu usprawnienia dostępu do danych wielu użytkownikom jednocześnie	549
Modele ADO i DAO	550
Narzędzia modelu ADO	552
Wprowadzanie rekordów do bazy danych	554
Pobieranie rekordów z bazy danych	556
Aktualizacja istniejącego rekordu w bazie danych	558
Usuwanie rekordów w przypadku wykorzystania modelu ADO	560
Podsumowania danych za pośrednictwem obiektów ADO	561
Inne narzędzia dostępne dla modelu ADO	562
Sprawdzanie istnienia tabel	562
Sprawdzanie istnienia pola	563
Dodawanie tabeli „w locie”	564
Dodawanie pól „w locie”	565
Następne kroki	565
22 Tworzenie klas, rekordów i kolekcji	567
Wstawianie modułu klasy	567
Przechwytywanie zdarzeń dotyczących aplikacji i wbudowanych wykresów	568
Zdarzenia aplikacji	568
Zdarzenia wbudowanych wykresów	570

Tworzenie własnych obiektów	572
Korzystanie z własnych obiektów	572
Wykorzystanie procedur Property Let i Property Get do zarządzania sposobem, w jaki użytkownicy korzystają z własnych obiektów	574
Kolekcje	576
Tworzenie kolekcji w module standardowym	576
Tworzenie kolekcji w module klasy	578
Przyciski pomocy	580
Typy definiowane przez użytkowników	582
Następne kroki	585

23 Zaawansowane techniki wykorzystania obiektów UserForm 587

Korzystanie z paska narzędzi obiektu UserForm podczas projektowania formantów na formularzach	587
Więcej formantów obiektów UserForm	588
Pola wyboru	588
Zakładki TabStrip	589
RefEdit	591
Przyciski-przełączniki	593
Wykorzystanie paska przewijania jako suwaka do wybierania wartości	594
Formanty i kolekcje	596
Niemodalne obiekty UserForm	598
Korzystanie z hiperłączy w formularzach UserForm	599
Dodawanie formantów w czasie działania programu	600
Zmiana rozmiaru formantów „w locie”	602
Dodawanie formantów „w locie”	602
Zmiana rozmiaru formantów „w locie”	603
Dodawanie innych formantów	603
Dodawanie obrazów „w locie”	603
Ostateczna wersja katalogu produktów	604
Tworzenie systemu pomocy w formularzach UserForm	606
Wyświetlanie aktywnych klawiszy	606
Dodawanie etykietek ekranowych do formantów	607
Określanie kolejności dostępu	607
Kolorowanie aktywnego formantu	608
Wielokolumnowe pola list	609
Przezroczyste formularze	611
Następne kroki	612

24 Interfejs programowania aplikacji (API) systemu Windows	613
Czym jest Windows API?	613
Deklaracje API	614
Korzystanie z deklaracji API	615
Przykłady użycia API	615
Odczytywanie nazwy komputera	616
Sprawdzenie, czy w sieci jest otwarty plik Excela	616
Odczytywanie informacji o rozdzielczości ekranu	617
Niestandardowe okno dialogowe O programie	618
Blokowanie ikony X zamykającej okno UserForm	619
Dynamiczny zegar	620
Odtwarzanie dźwięków	621
Odczytywanie ścieżki do pliku	621
Więcej deklaracji API	625
Następne kroki	625
25 Obsługa błędów	627
Co się dzieje, kiedy wystąpi błąd?	627
Debugowanie błędów występujących podczas obsługi formularza użytkownika jest mylące	628
Podstawowa obsługa błędów za pomocą instrukcji On Error GoTo	631
Blok obsługi błędów ogólnego przeznaczenia	632
Obsługa błędów polegająca na ich ignorowaniu	633
Problemy z konfiguracją stron zwykle można zignorować	633
Blokowanie wyświetlania ostrzeżeń	634
Celowe prowokowanie błędów	635
Szkolenie użytkowników	636
Błędy wykryte w fazie projektowania a błędy wykryte miesiąc później	636
Błąd wykonania nr 9: indeks poza zakresem	637
Błąd wykonania nr 1004: niepowodzenie odwołania do zakresu globalnego obiektu	638
Problemy związane z zabezpieczaniem kodu	639
Łamanie haseł	639
Dodatkowe problemy z hasłami	640
Błędy powodowane przez różne wersje	640
Następne kroki	641
26 Dostosowywanie wstążki do uruchamiania makr	643
Stare odchodzi, nowe przychodzi	643
Gdzie wprowadzać kod: folder i plik customui	644

Tworzenie zakładek i grup	645
Dodawanie formantu na wstążce	646
Dostęp do struktury pliku	653
Struktura pliku RELS	654
Zmiana nazwy pliku Excela i otwarcie skoroszytu	655
RibbonCustomizer	655
Wykorzystywanie elementów graficznych na przyciskach	655
Ikony Microsoft Office	656
Tworzenie własnych ikon	657
Konwersja niestandardowych pasków narzędzi z Excela 2003 do Excela 2007	659
Rozwiązywanie problemów z wykorzystaniem komunikatów o błędach	660
Atrybut „Nazwa atrybutu” w elemencie „wstążka customui” nie został zdefiniowany w schemacie lub definicji DTD	661
Niedozwolony znak w nazwie kwalifikowanej	662
Element „nazwa znacznika customui” jest nieoczekiwany w odniesieniu do modelu zawartości elementu nadrzędnego „nazwa znacznika customui”	662
Excel znalazł zawartość, której nie można odczytać	663
Niewłaściwa liczba argumentów lub nieprawidłowe przypisanie właściwości	664
Nic się nie dzieje	664
Inne sposoby uruchamiania makr	664
Skróty klawiaturowe	665
Powiązanie makra z przyciskiem polecenia	666
Dowiązanie makr do formantów ActiveX	668
Uruchamianie makra za pośrednictwem hiperłącza	670
Następne kroki	671
27 Tworzenie dodatków	673
Charakterystyka standardowych dodatków	673
Konwersja skoroszytu Excela na dodatek	674
Wykorzystanie polecenia Zapisz jako w celu konwersji pliku na dodatek	675
Wykorzystanie edytora VB w celu konwersji pliku na dodatek	676
Instalacja dodatków	677
Standardowe dodatki nie są bezpieczne	678
Zamykanie dodatków	679
Usuwanie dodatków	679
Wykorzystanie ukrytych skoroszytów jako alternatywy dodatków	680
Wykorzystywanie ukrytego skoroszytu z kodem w celu przechowywania wszystkich makr i formularzy	680
Następne kroki	681
Skorowidz	683

Zaawansowane możliwości Excela

14

Jedną z najważniejszych zasad programisty, który chce odnieść sukces, jest unikanie marnowania czasu na dwukrotne pisanie tego samego kodu. Wszyscy programiści mają swoje fragmenty kodu — niewielkie lub nawet obszerne — które wykorzystują wielokrotnie. Inna zasada głosi, aby nigdy nie poświęcać ośmiu godzin na zrobienie czegoś, co można zrobić w dziesięć minut — właśnie o tym piszemy w tej książce.

W niniejszym rozdziale zamieszczono funkcje подарowane przez kilku zaawansowanych programistów Excela. Są to programy uznane przez nich za przydatne, którymi postanowili podzielić się z czytelnikami tej książki. Dzięki nim nie tylko można zaoszczędzić czas, ale również nauczyć się nowych sposobów rozwiązywania znanych problemów.

Różni programiści stosują odmienne style programowania, a my nie poprawialiśmy nadesłanych materiałów. Podczas przeglądania kodu można spotkać różne sposoby wykonywania tych samych zadań — np. odwoływania się do zakresów.

Operacje na plikach

Narzędzia zamieszczone w tym podrozdziale dotyczą obsługi plików w folderach. Umiejętność przeglądania w pętli listy plików w folderze bardzo się przydaje.

Wyświetlanie listy plików w katalogu

Nadesłał Nathan P. Oliver z Minneapolis w stanie Minnesota. Nathan jest konsultantem finansowym i twórcą aplikacji.

W TYM ROZDZIALE:

Operacje na plikach	409
Łączenie i rozdziałanie skoroszytów	414
Korzystanie z komentarzy w komórkach	418
Narzędzia, których celem jest zrobienie pozytywnego wrażenia na klientach	424
Techniki dla ekspertów języka VBA	431
Interesujące aplikacje	445
Następne kroki	448



Poniższy program zwraca nazwę, rozmiar i datę modyfikacji wszystkich plików w wybranym katalogu i jego podfolderach.

```

Sub ExcelFileSearch()
Dim srchExt As Variant, srchDir As Variant, i As Long, j As Long
Dim strName As String, varArr(1 To 1048576, 1 To 3) As Variant
Dim strFileFullName As String
Dim ws As Worksheet
Dim fso As Object

Let srchExt = Application.InputBox("Proszę wprowadzić rozszerzenie pliku",
↳"Prośba o dane")
If srchExt = False And Not TypeName(srchExt) = "String" Then
Exit Sub
End If

Let srchDir = BrowseForFolderShell
If srchDir = False And Not TypeName(srchDir) = "String" Then
Exit Sub
End If

Application.ScreenUpdating = False

Set ws = ThisWorkbook.Worksheets.Add(Sheets(1))
On Error Resume Next
Application.DisplayAlerts = False
ThisWorkbook.Worksheets("Wyniki wyszukiwania plików").Delete
Application.DisplayAlerts = True
On Error GoTo 0
ws.Name = "Wyniki wyszukiwania plików"

Let strName = Dir$(srchDir & "\*" & srchExt)
Do While strName <> vbNullString
Let i = i + 1
Let strFileFullName = srchDir & strName
Let varArr(i, 1) = strFileFullName
Let varArr(i, 2) = FileLen(strFileFullName) \ 1024
Let varArr(i, 3) = FileDateTime(strFileFullName)
Let strName = Dir$()
Loop

Set fso = CreateObject("Scripting.FileSystemObject")
Call recurseSubFolders(fso.GetFolder(srchDir), varArr(), i, CStr(srchExt))
Set fso = Nothing

ThisWorkbook.Windows(1).DisplayHeadings = False
With ws
If i > 0 Then
.Range("A2").Resize(i, UBound(varArr, 2)).Value = varArr
For j = 1 To i
.Hyperlinks.Add anchor:=.Cells(j + 1, 1), Address:=varArr(j, 1)
Next
End If

```

```

.Range(.Cells(1, 4), .Cells(1, .Columns.Count)).EntireColumn.Hidden = True
.Range(.Cells(.Rows.Count, 1).End(xlUp)(2), _
.Cells(.Rows.Count, 1)).EntireRow.Hidden = True
With .Range("A1:C1")
    .Value = Array("Pełna nazwa", "Kilobajłów", "Data ostatniej
↳modyfikacji")
    .Font.Underline = xlUnderlineStyleSingle
    .EntireColumn.AutoFit
    .HorizontalAlignment = xlCenter
End With
End With
Application.ScreenUpdating = True
End Sub

Private Sub recurseSubFolders(ByRef Folder As Object, _
    ByRef varArr() As Variant, _
    ByRef i As Long, _
    ByRef srchExt As String)
Dim SubFolder As Object
Dim strName As String, strFileFullName As String
For Each SubFolder In Folder.SubFolders
    Let strName = Dir$(SubFolder.Path & "\*" & srchExt)
    Do While strName <> vbNullString
        Let i = i + 1
        Let strFileFullName = SubFolder.Path & "\" & strName
        Let varArr(i, 1) = strFileFullName
        Let varArr(i, 2) = FileLen(strFileFullName) \ 1024
        Let varArr(i, 3) = FileDateTime(strFileFullName)
        Let strName = Dir$()
    Loop
    If i > 1048576 Then Exit Sub
    Call recurseSubFolders(SubFolder, varArr(), i, srchExt)
Next
End Sub

Private Function BrowseForFolderShell() As Variant
Dim objShell As Object, objFolder As Object
Set objShell = CreateObject("Shell.Application")
Set objFolder = objShell.BrowseForFolder(0, "Proszę wybrać folder", 0, "C:\")
If Not objFolder Is Nothing Then
    On Error Resume Next
    If IsError(objFolder.Items.Item.Path) Then
        BrowseForFolderShell = CStr(objFolder)
    Else
        On Error GoTo 0
        If Len(objFolder.Items.Item.Path) > 3 Then
            BrowseForFolderShell = objFolder.Items.Item.Path & _
                Application.PathSeparator
        Else
            BrowseForFolderShell = objFolder.Items.Item.Path
        End If
    End If
Else
End Function

```

```
        BrowseForFolderShell = False
    End If
    Set objFolder = Nothing: Set objShell = Nothing
End Function
```

Importowanie plików w formacie CSV

Nadesłał Masaru Kaji z Kobe-City w Japonii. Masaru pracuje jako konsultant w dziedzinie Excela za pośrednictwem witryny Excel Junk Room (www.puremis.net/excel/).

Program przyda się tym osobom, które często importują pliki w formacie danych rozdzielanych przecinkami (ang. *comma-separated variable* — CSV), a następnie muszą zadbać o ich usunięcie. Aplikacja błyskawicznie otwiera plik CSV w Excelu, po czym trwale usuwa oryginalny plik.

```
Option Base 1
```

```
Sub OpenLargeCSVFast()
    Dim buf(1 To 16384) As Variant
    Dim i As Long
    ' Tutaj należy zmienić lokalizację pliku i jego nazwę
    Const strFilePath As String = "C:\temp\Test.CSV"

    Dim strRenamedPath As String
    strRenamedPath = Split(strFilePath, ".")(0) & ".txt"

    With Application
        .ScreenUpdating = False
        .DisplayAlerts = False
    End With
    ' Konfiguracja tablicy dla struktury FieldInfo w celu otwarcia pliku CSV
    For i = 1 To 16384
        buf(i) = Array(i, 2)
    Next
    Name strFilePath As strRenamedPath
    Workbooks.OpenText Filename:=strRenamedPath, DataType:=xlDelimited, _
        Comma:=True, FieldInfo:=buf

    Erase buf
    ActiveSheet.UsedRange.Copy ThisWorkbook.Sheets(1).Range("A1")
    ActiveWorkbook.Close False
    Kill strRenamedPath
    With Application
        .ScreenUpdating = True
        .DisplayAlerts = True
    End With
End Sub
```

Wczytanie całego pliku CSV do pamięci w celu jego przetwarzania

Przesłane przez Suata Mehmeta Ozgurę z Istambułu w Turcji. Suat zajmuje się tworzeniem aplikacji w Excelu, Accessie i Visual Basicu dla firm MrExcel oraz TheOfficeExperts.

W tym przykładzie zastosowano inne podejście do czytania pliku tekstowego. Makro, zamiast odczytywania po jednym rekordzie, ładuje cały plik tekstowy do pamięci jako pojedynczą zmienną tekstową, a następnie przetwarza ciąg znaków na pojedyncze rekordy. Zaletą zastosowania tej metody jest fakt, iż wymaga tylko jednorazowego dostępu do dysku. Wszystkie pozostałe obliczenia są wykonywane bardzo szybko w pamięci.

```
Sub ReadTxtLines()
```

```
' Nie ma potrzeby instalacji biblioteki Scripting Runtime, ponieważ zastosowano późne wiązanie
```

```
Dim sht As Worksheet
```

```
Dim fso As Object
```

```
Dim fil As Object
```

```
Dim txt As Object
```

```
Dim strtxt As String
```

```
Dim tmpLoc As Long
```

```
' Praca w aktywnym arkuszu
```

```
Set sht = ActiveSheet
```

```
' Wyczyszczenie danych w arkuszu
```

```
sht.UsedRange.ClearContents
```

```
' Obiekt systemu plików potrzebny do zarządzania plikami
```

```
Set fso = CreateObject("Scripting.FileSystemObject")
```

```
' Plik, który chcemy otworzyć i odczytać
```

```
Set fil = fso.GetFile("c:\test.txt")
```

```
' Otwarcie pliku w postaci strumienia tekstowego
```

```
Set txt = fil.OpenAsTextStream(1)
```

```
' Odczytanie pliku i zapisanie go w zmiennej tekstowej
```

```
strtxt = txt.ReadAll
```

```
' Zamknięcie strumienia tekstowego i zwolnienie pliku. Nie będzie już dłużej potrzebny
```

```
txt.Close
```

```
' Wyszukanie pierwszego wystąpienia znaku przejścia do nowego wiersza
```

```
tmpLoc = InStr(1, strtxt, vbCrLf)
```

```
' Pętla do momentu, kiedy następny znak przejścia do nowego wiersza nie zostanie odnaleziony
```

```
Do Until tmpLoc = 0
```

```
' Wykorzystanie kolumny A i następnej pustej komórki w celu zapisania wiersza pliku tekstowego
```

```
sht.Cells(sht.Rows.Count, 1).End(xlUp).Offset(1).Value = _  
Left(strtxt, tmpLoc - 1)
```

```
' Usunięcie przetworzonego wiersza ze zmiennej, w której zapamiętano dołączony plik
```

```
strtxt = Right(strtxt, Len(strtxt) - tmpLoc - 1)
```



```

        ' Wyszukanie następnego wystąpienia znaku przejścia do nowego wiersza
        tmpLoc = InStr(1, strtxt, vbCrLf)
    Loop

    ' Ostatni wiersz, w którym są dane, ale nie ma znaku przejścia do nowego wiersza
    sht.Cells(sht.Rows.Count, 1).End(xlUp).Offset(1).Value = strtxt

    ' Zakończenie tej procedury zwolniłoby obiekt, ale do dobrego zwyczaju należy
    ' ustawienie wartości obiektu na Nothing
    Set fso = Nothing
End Sub

```

Łączenie i rozdzielanie skoroszytów

W kolejnych czterech narzędziach zademonstrowano sposoby łączenia arkuszy w pojedynczy skoroszyt lub rozdzielania skoroszytu na indywidualne arkusze albo dokumenty Worda.

Rozdzielanie arkuszy na osobne skoroszyty

Przesłane przez Tommy'ego Milea z Houston w stanie Texas.

Kod zamieszczony poniżej przegląda aktywny skoroszyt i zapisuje każdy arkusz w osobnym skoroszycie w tym samym folderze, w którym jest zapisany oryginalny skoroszyt. Nazwy nowych skoroszytów są wyznaczone na podstawie nazwy arkusza. Makro nadpisuje pliki bez wyświetlania pytania. Jak można zauważyć, użytkownik musi również zdecydować o tym, czy chce zapisać plik w formacie *.xism* (z obsługą makr), czy też *.xlsx* (wtedy makra będą usunięte). W poniższym makrze zamieszczono kod obsługi zarówno formatu *.xism*, jak i *.xlsx*, ale wiersze obsługujące format *.xlsx* ujęto w komentarz, by stały się nieaktywne.

```

Sub SplitWorkbook()

    Dim ws As Worksheet
    Dim DisplayStatusBar As Boolean

    DisplayStatusBar = Application.DisplayStatusBar
    Application.DisplayStatusBar = True
    Application.ScreenUpdating = False
    Application.DisplayAlerts = False

    For Each ws In ThisWorkbook.Sheets
        Dim NewFileName As String
        Application.StatusBar = "Pozostało arkuszy " & ThisWorkbook.Sheets.Count
        If ThisWorkbook.Sheets.Count <> 1 Then
            NewFileName = ThisWorkbook.Path & "\" & ws.Name & ".xism" 'Z obsługą makr
            ' NewFileName = ThisWorkbook.Path & "\" & ws.Name & ".xlsx" _
            ' Bez obsługi makr
            ws.Copy

```

```

        ActiveWorkbook.Sheets(1).Name = "Arkusz1"
        ActiveWorkbook.SaveAs Filename:=NewFileName, _
            FileFormat:=xlOpenXMLWorkbookMacroEnabled
    '   ActiveWorkbook.SaveAs Filename:=NewFileName, _
        FileFormat:=xlOpenXMLWorkbook
        ActiveWorkbook.Close SaveChanges:=False
    Else
        NewFileName = ThisWorkbook.Path & "\" & ws.Name & ".xlsm"
    '   NewFileName = ThisWorkbook.Path & "\" & ws.Name & ".xlsx"
        ws.Name = "Arkusz1"
    End If
Next

Application.DisplayAlerts = True
Application.StatusBar = False
Application.DisplayStatusBar = DisplayStatusBar
Application.ScreenUpdating = True
End Sub

```

Łączenie skoroszytów

Nadesłał Tommy Miles.

Makro zaprezentowane w tym przykładzie przegląda wszystkie pliki Excela w określonym katalogu i łączy je w pojedynczy skoroszyt. Nazwy arkuszy są wyznaczone na podstawie nazw źródłowych skoroszytów.

```

Sub CombineWorkbooks()
    Dim CurFile As String, DirLoc As String
    Dim DestWB As Workbook
    Dim ws As Object 'Zezwolenie na różne typy arkuszy

    DirLoc = ThisWorkbook.Path & "\tst\" 'Lokalizacja plików
    CurFile = Dir(DirLoc & "*.xls")

    Application.ScreenUpdating = False
    Application.EnableEvents = False

    Set DestWB = Workbooks.Add(xlWorksheet)

    Do While CurFile <> vbNullString
        Dim OrigWB As Workbook
        Set OrigWB = Workbooks.Open(Filename:=DirLoc & CurFile, ReadOnly:=True)

        ' Ograniczenie do poprawnych nazw arkuszy i usunięcie plików .xls*
        CurFile = Left(Left(CurFile, Len(CurFile) - 5), 29)

        For Each ws In OrigWB.Sheets
            ws.Copy After:=DestWB.Sheets(DestWB.Sheets.Count)

            If OrigWB.Sheets.Count > 1 Then
                DestWB.Sheets(DestWB.Sheets.Count).Name = CurFile & ws.Index
            End If
        Next ws
    Loop
End Sub

```

```

        Else
            DestWB.Sheets(DestWB.Sheets.Count).Name = CurFile
        End If
    Next

    OrigWB.Close SaveChanges:=False
    CurFile = Dir
Loop

Application.DisplayAlerts = False
    DestWB.Sheets(1).Delete
Application.DisplayAlerts = True

Application.ScreenUpdating = True
Application.EnableEvents = True

Set DestWB = Nothing
End Sub

```

Filtrowanie i kopiowanie danych do osobnych arkuszy

Nadesłał Dennis Wallentin z Ostersund w Szwecji. Dennis prezentuje wskazówki i sztuczki dotyczące Excela w witrynie www.xldennis.com.

Makro zaprezentowane w tym przykładzie wykorzystuje podaną kolumnę do filtrowania danych i kopiuje wyniki do nowych arkuszy w aktywnym skoroszytcie.

```

Sub Filter_NewSheet()
Dim wbBook As Workbook
Dim wsSheet As Worksheet
Dim rnStart As Range, rnData As Range
Dim i As Long

Set wbBook = ThisWorkbook
Set wsSheet = wbBook.Worksheets("Arkusz1")

With wsSheet
    'Sprawdzenie, czy pierwszy wiersz zawiera nagłówki
    Set rnStart = .Range("A2")
    Set rnData = .Range(.Range("A2"), .Cells(.Rows.Count, 3).End(xlUp))
End With

Application.ScreenUpdating = True

For i = 1 To 5
    'Filtrowanie danych z wykorzystaniem pierwszego kryterium
    rnStart.AutoFilter Field:=1, Criteria:="AA" & i
    'Skopiowanie odfiltrowanej listy
    rnData.SpecialCells(xlCellTypeVisible).Copy
    'Dodanie nowego arkusza do aktywnego skoroszytu
    Worksheets.Add Before:=wsSheet
    'Nadanie nazw nowym arkuszom

```

```

        ActiveSheet.Name = "AA" & i
        ' Wklejenie odfiltrowanej listy
        Range("A2").PasteSpecial xlPasteValues
    Next i

    ' Zresetowanie listy do jej pierwotnego statusu
    rnStart.AutoFilter Field:=1

    With Application
        ' Zresetowanie schowka
        .CutCopyMode = False
        .ScreenUpdating = False
    End With

End Sub

```

Eksport danych do Worda

Nadesłał Dennis Wallentin.

Program transferuje dane z Excela do pierwszej z tabel w dokumencie Worda. Wykorzystano w nim wczesne wiązanie, zatem w edytorze VBA trzeba zdefiniować referencję (za pomocą polecenia *Tools/References*) do biblioteki *Microsoft Word Object Library*.

```

Sub Export_Data_Word_Table()
Dim wdApp As Word.Application
Dim wdDoc As Word.Document
Dim wdCell As Word.Cell
Dim i As Long
Dim wbBook As Workbook
Dim wsSheet As Worksheet
Dim rnData As Range
Dim vaData As Variant

Set wbBook = ThisWorkbook
Set wsSheet = wbBook.Worksheets("Arkusze1")

With wsSheet
    Set rnData = .Range("A1:A10")
End With

' Dodanie wartości z zakresu do jednowymiarowej tablicy typu variant
vaData = rnData.Value

' Utworzenie egzemplarza nowego obiektu
Set wdApp = New Word.Application
' W tym przykładzie dokument docelowy jest zapisany w tym samym folderze, co skoroszyt
Set wdDoc = wdApp.Documents.Open(ThisWorkbook.Path & "\Test.docx")

' Zaimportowanie danych do pierwszego wiersza i do pierwszej kolumny tabeli składającej się z dziesięciu wierszy
For Each wdCell In wdDoc.Tables(1).Columns(1).Cells
    i = i + 1

```

```
wdCell.Range.Text = vaData(i, 1)
Next wdCell

' Zapisanie i zamknięcie dokumentu
With wdDoc
    .Save
    .Close
End With

' Zamknięcie ukrytej instancji programu Microsoft Word
wdApp.Quit
' Zwolnienie pamięci zajmowanej przez zewnętrzne zmienne
Set wdDoc = Nothing
Set wdApp = Nothing

MsgBox "Dane przetransferowano do pliku Test.docx.", vbInformation

End Sub
```

Korzystanie z komentarzy w komórkach

Komentarze do danych w komórkach często są niedocenianą własnością Excela. Cztery narzędzia zaprezentowane w tym podrozdziale pomogą czytelnikom we właściwym wykorzystaniu tego mechanizmu.

Lista komentarzy

Nadesłał Tommy Miles.

Excel pozwala użytkownikom na wyświetlanie komentarzy w skróconym widoku, ale nie precyzuje skróconego widoku arkusza, do którego dodano komentarz, a jedynie komórkę — co pokazano na rysunku 14.1. Makro zamieszczone poniżej umieszcza komentarze wraz z autorem i lokalizacją każdego z nich w nowym arkuszu w celu umożliwienia łatwego przeglądania, zapisywania lub drukowania. Uzyskane wyniki pokazano na rysunku 14.2.

Rysunek 14.1.
Excel wyświetla tylko adres komórki i umieszczony w niej komentarz

Komórka: C5
Komentarz: Bill Jelen: Nie obejmuje specjalnej wyprzedaży.
Komórka: D14
Komentarz: Bill Jelen: Dziękujemy za pobranie plików projektu.
Komórka: A27
Komentarz: Bill Jelen: Odwiedź witrynę MrExcel.com, aby uzyskać dostęp do ponad 70 000 artykułów na temat Excela.

Rysunek 14.2.
Lista wszystkich informacji
związanych z komentarzami

	A	B	C	D	E	F	G	H	I	J	K	L	M
1	Autor	Skoroszyt	Arkusze	Zakres	Komentarz								
2	Bill Jelen	Rozdział14.xlsm	ListaKomentarzy	\$C\$5	Nie obejmuje specjalnej wyprzedaży.								
3	Bill Jelen	Rozdział14.xlsm	ListaKomentarzy	\$D\$14	Dziękujemy za pobranie plików projektu.								
4	Bill Jelen	Rozdział14.xlsm	ListaKomentarzy	\$A\$27	Odwiedź witrynę MrExcel.com, aby uzyskać dostęp do ponad 70 000 artykułów na temat Excela.								

```

Sub ListComments()
    Dim wb As Workbook
    Dim ws As Worksheet

    Dim cmt As Comment

    Dim cmtCount As Long

    cmtCount = 2

    On Error Resume Next
        Set ws = ActiveSheet
        If ws Is Nothing Then Exit Sub
    On Error GoTo 0

    Application.ScreenUpdating = False

    Set wb = Workbooks.Add(xlWorksheet)

    With wb.Sheets(1)
        .Range("$A$1") = "Autor"
        .Range("$B$1") = "Skoroszyt"
        .Range("$C$1") = "Arkusze"
        .Range("$D$1") = "Zakres"
        .Range("$E$1") = "Komentarz"
    End With

    For Each cmt In ws.Comments
        With wb.Sheets(1)
            .Cells(cmtCount, 1) = cmt.author
            .Cells(cmtCount, 2) = cmt.Parent.Parent.Parent.Name
            .Cells(cmtCount, 3) = cmt.Parent.Parent.Name
            .Cells(cmtCount, 4) = cmt.Parent.Address
            .Cells(cmtCount, 5) = CleanComment(cmt.author, cmt.Text)
        End With

        cmtCount = cmtCount + 1
    Next

    wb.Sheets(1).UsedRange.WrapText = False

    Application.ScreenUpdating = True

    Set ws = Nothing
    Set wb = Nothing
End Sub

Private Function CleanComment(author As String, cmt As String) As String

```

```

Dim tmp As String

tmp = Application.WorksheetFunction.Substitute(cmt, author & ":", "")
tmp = Application.WorksheetFunction.Substitute(tmp, Chr(10), "")

CleanComment = tmp
End Function

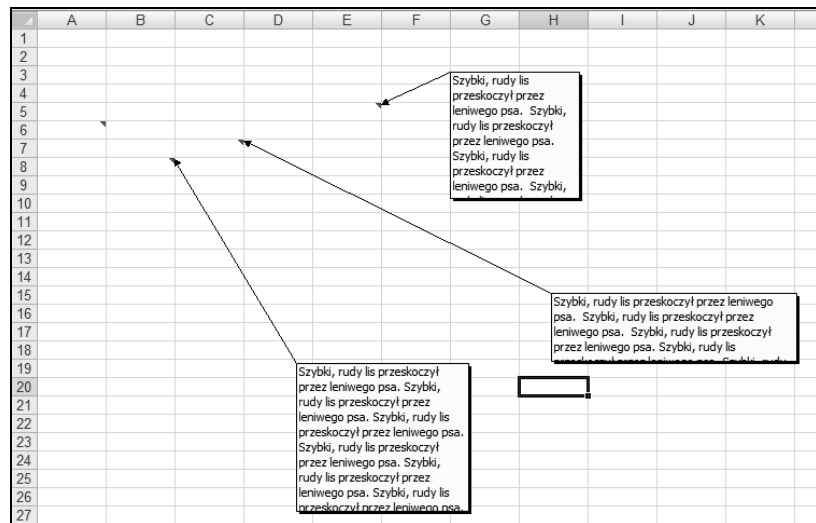
```

Zmiana rozmiaru komentarzy

Nadesłał Tom Urtis z San Francisco w stanie Kalifornia. Tom jest jednym z właścicieli Atlas Programming Management — firmy konsultingowej zajmującej się Exceliem w Bay Area.

Excel nie zmienia automatycznie ramek komentarzy w komórkach. Jeśli w arkuszu jest ich kilka, tak jak pokazano na rysunku 14.3, zmienianie ich rozmiaru po kolei może okazać się kłopotliwe. Makro zamieszczone w poniższym przykładzie zmienia rozmiar wszystkich ramek komentarzy w taki sposób, że po zaznaczeniu można przeglądać całą treść komentarza (rysunek 14.4).

Rysunek 14.3.
Domyślnie Excel nie dostosowuje rozmiaru ramek komentarzy w taki sposób, by był widoczny cały wprowadzony tekst



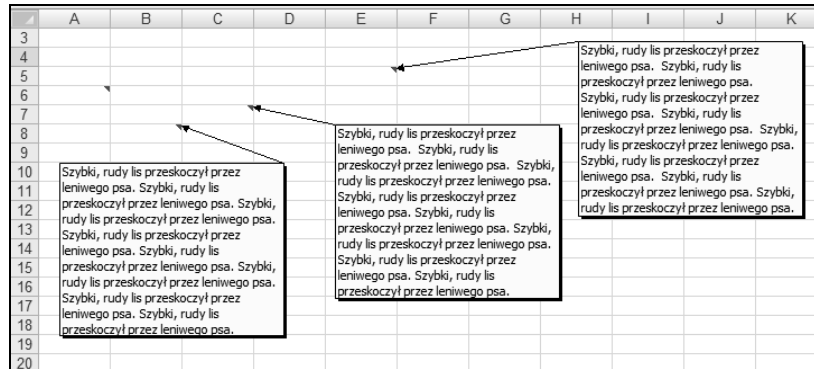
```

Sub CommentFitter1()
Application.ScreenUpdating = False
Dim x As Range, y As Long

For Each x In Cells.SpecialCells(xlCellTypeComments)
Select Case True
Case Len(x.NoteText) <> 0
With x.Comment
.Shape.TextFrame.AutoSize = True
If .Shape.Width > 250 Then

```

Rysunek 14.4.
Zmiana rozmiaru ramek komentarzy w celu dopasowania do objętości tekstu



```

        y = .Shape.Width * .Shape.Height
        .Shape.Width = 150
        .Shape.Height = (y / 200) * 1.3
    End If
End With
End Select
Next x
Application.ScreenUpdating = True
End Sub

```

Zmiana rozmiaru komentarzy poprzez wyśrodkowanie

Nadesłał Tom Urtis.

Makro pokazane w tym przykładzie zmienia rozmiar wszystkich ramek komentarzy poprzez wyśrodkowanie ich zawartości (rysunek 14.5).

Rysunek 14.5.
Wyśrodkowanie komentarzy w arkuszu



```

Sub CommentFitter2()
Application.ScreenUpdating = False
Dim x As Range, y As Long

For Each x In Cells.SpecialCells(xlCellTypeComments)
    Select Case True

```



```

Case Len(x.NoteText) <> 0
  With x.Comment
    .Shape.TextFrame.AutoSize = True
    If .Shape.Width > 250 Then
      y = .Shape.Width * .Shape.Height
      .Shape.ScaleHeight 0.9, msoFalse, msoScaleFromTopLeft
      .Shape.ScaleWidth 1#, msoFalse, msoScaleFromTopLeft
    End If
  End With
End Select
Next x
Application.ScreenUpdating = True
End Sub

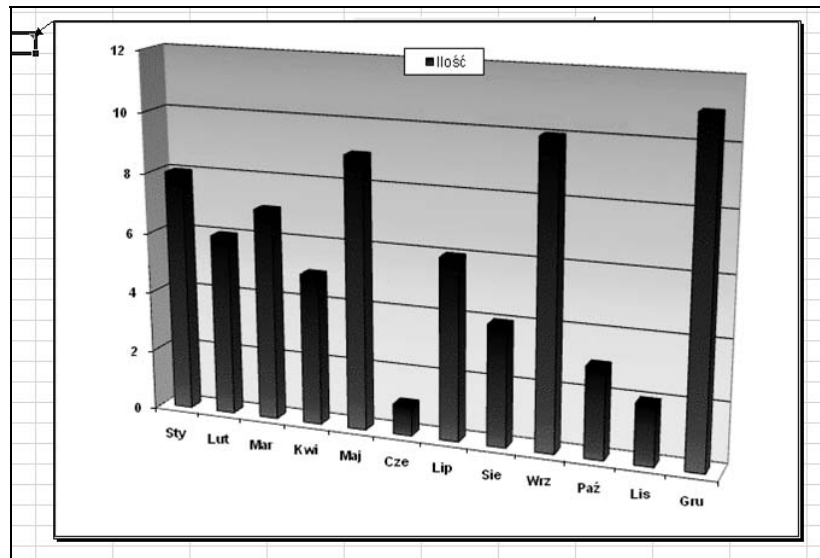
```

Umieszczenie wykresu w komentarzu

Nadesłał Tom Urtis.

Dynamiczny wykres nie może występować w formie kształtu. Można jednak utworzyć zrzut wykresu i załadować go do komentarza, co pokazano na rysunku 14.6.

Rysunek 14.6.
Umieszczenie wykresu
w komentarzu



Aby osiągnąć taki efekt ręcznie, wykonaj następujące czynności:

1. Utwórz i zapisz obraz, który ma się wyświetlić w komentarzu.
2. Utwórz komentarz, jeśli nie zrobiłeś tego wcześniej, i zaznacz komórkę, której dotyczy.
3. Na wstążce *Recenzja* wybierz polecenie *Edytuj komentarz* lub kliknij komórkę prawym przyciskiem myszy i wybierz polecenie *Edytuj komentarz*.

4. Kliknij prawym przyciskiem myszy obramowanie komentarza i wybierz polecenie *Formatuj komentarz*.
5. Zaznacz zakładkę *Kolory i linie* i kliknij strzałkę w dół należącą do *Kolor* w sekcji *Wypełnienie*.
6. Kliknij polecenie *Efekty wypełnienia*, wybierz zakładkę *Obraz*, a następnie kliknij przycisk *Wybierz obraz*.
7. Odszukaj żądany obraz, wybierz go i dwukrotnie kliknij *OK*.

Efekt posiadania dynamicznego wykresu w komentarzu można osiągnąć np. wtedy, kiedy kod jest częścią obsługi zdarzenia `SheetChange` po zmodyfikowaniu źródłowych danych wykresu. Wykresy prezentujące biznesowe dane są często aktualizowane, zatem może się przydać makro, które zapewnia aktualność komentarzy i pozwala na uniknięcie wykonywania tych samych czynności. Makro zamieszczone poniżej wykonuje właśnie takie czynności: modyfikuje ścieżkę dostępu do pliku, nazwę wykresu, arkusz docelowy, komórkę i rozmiar kształtu komentarza, w zależności od rozmiaru wykresu.

```
Sub PlaceGraph()
```

```
Dim x As String, z As Range
```

```
Application.ScreenUpdating = False
```

```
' Określenie tymczasowej lokalizacji, w której ma być zapisany obraz
```

```
x = "C:\XW MJGraph.gif"
```

```
' Określenie komórki, w której ma się znaleźć komentarz
```

```
Set z = Worksheets("WykresWKomentarzu").Range("A3")
```

```
' Usunięcie z komórki komentarzy, które były tam wcześniej
```

```
On Error Resume Next
```

```
z.Comment.Delete
```

```
On Error GoTo 0
```

```
' Zaznaczenie i wyeksportowanie wykresu
```

```
ActiveSheet.ChartObjects("Wykres 1").Activate
```

```
ActiveChart.Export x
```

```
' Dodanie nowego komentarza do komórki, ustawienie rozmiaru i wstawienie wykresu
```

```
With z.AddComment
```

```
With .Shape
```

```
.Height = 322
```

```
.Width = 465
```

```
.Fill.UserPicture x
```

```
End With
```

```
End With
```

```
' Usunięcie tymczasowego obrazu
```

```
Kill x
```

```
Range("A1").Activate
Application.ScreenUpdating = True
```

```
Set z = Nothing
End Sub
```

Narzędzia, których celem jest zrobienie pozytywnego wrażenia na klientach

Następne cztery narzędzia wzbudzą zachwyt i uznanie waszych klientów.

Wykorzystanie formatowania warunkowego do podświetlenia wybranej komórki

Nadesłał Ivan F. Moala z Auckland w Nowej Zelandii. Ivan jest autorem serwisu XcelFiles (www.xcelfiles.com); na stronie tego serwisu można się dowiedzieć, w jaki sposób w Excelu robi się rzeczy niewykonalne.

W tym przykładzie wykorzystano formatowanie warunkowe w celu podświetlenia wiersza i kolumny aktywnej komórki, aby ułatwić jej wizualną lokalizację (rysunek 14.7). **Ważne!** Nie należy stosować tej metody w arkuszach, w których wcześniej wykorzystano formatowanie warunkowe. Zdefiniowane wcześniej formaty warunkowe zostaną nadpisane. Program ten zeruje również schowek, dlatego nie ma możliwości jego użycia podczas operacji kopiowania, wycinania lub wklejania.

Rysunek 14.7.
Wykorzystanie formatowania warunkowego do podświetlenia wybranej komórki w tabeli

	A	B	C	D	E
4	Tomek	Wschodni	Kw.2	Kurtki	Czarne
5	Michał	Zachodni	Kw.2	Kurtki	Żółte
6	Jurek	Południowy	Kw.3	Kapelusze	Żółte
7	Natalia	Północny	Kw.3	Kapelusze	Czarne
8	Zofia	Wschodni	Kw.4	Kapelusze	Niebieskie
9	Wiliam	Zachodni	Kw.4	Buty	Czarne
10	Maria	Południowy	Kw.4	Buty	Czarne
11	Bogdan	Północny	Kw.1	Buty	Niebieskie
12	Bronek	Wschodni	Kw.1	Kapelusze	Czarne
13	Michał	Zachodni	Kw.2	Buty	Żółte
14	Jurek	Południowy	Kw.2	Buty	Żółte
15	Natalia	Północny	Kw.3	Buty	Czarne
16	Zofia	Wschodni	Kw.3	Kurtki	Niebieskie
17	Wiliam	Zachodni	Kw.4	Kurtki	Czarne
18	Maria	Południowy	Kw.4	Kurtki	Czarne
19	Bogdan	Północny	Kw.4	Kurtki	Niebieskie
20	Tomek	Wschodni	Kw.1	Kapelusze	Czarne
21	Michał	Zachodni	Kw.1	Kapelusze	Żółte
22	Tomek	Południowy	Kw.2	Kapelusze	Żółte
23	Michał	Północny	Kw.2	Buty	Czarne

```
Const iInternational As Integer = Not (0)

Private Sub Worksheet_SelectionChange(ByVal Target As Range)
Dim iColor As Integer
' // Obsługa błędu w sytuacji, gdy
' // użytkownik zaznaczy zakres komórek
On Error Resume Next
iColor = Target.Interior.ColorIndex
' // Pozostawienie włączonej obsługi błędów dla błędów przesunięcia wiersza

If iColor < 0 Then
    iColor = 36
Else
    iColor = iColor + 1
End If

' // Ten test jest potrzebny na wypadek, gdyby kolor czcionki był taki sam
If iColor = Target.Font.ColorIndex Then iColor = iColor + 1

Cells.FormatConditions.Delete

' // Naprzemiennie pokolorowanie wierszy
With Range("A" & Target.Row, Target.Address) 'Rows(Target.Row)
    .FormatConditions.Add Type:=2, Formula1:=iInternational 'Lub po prostu 1—"TRUE"
    .FormatConditions(1).Interior.ColorIndex = iColor
End With

' // Naprzemiennie pokolorowanie kolumn
With Range(Target.Offset(1 - Target.Row, 0).Address & ":" & _
    Target.Offset(-1, 0).Address)
    .FormatConditions.Add Type:=2, Formula1:=iInternational 'Lub po prostu 1—"TRUE"
    .FormatConditions(1).Interior.ColorIndex = iColor
End With

End Sub
```

Wyróżnienie wybranej komórki bez użycia formatowania warunkowego

Nadesłał Ivan F. Moala.

W tym przykładzie pokazano sposób wizualnego wyróżnienia aktywnej komórki bez użycia formatowania warunkowego w czasie, gdy użytkownik wykorzystuje klawisze strzałek do poruszania się po arkuszu.

Poniższy fragment kodu należy umieścić w standardowym module:

```
Dim strCol As String
Dim iCol As Integer
Dim dblRow As Double

Sub HighlightRight()
    HighLight 0, 1
```

```

End Sub

Sub HighlightLeft()
    HighLight 0, -1
End Sub

Sub HighlightUp()
    HighLight -1, 0, -1
End Sub

Sub HighlightDown()
    HighLight 1, 0, 1
End Sub

Sub HighLight(dblxRow As Double, iyCol As Integer, Optional dblZ As Double = 0)

On Error GoTo NoGo
strCol = Mid(ActiveCell.Offset(dblxRow, iyCol).Address, _
            InStr(ActiveCell.Offset(dblxRow, iyCol).Address, "$") + 1, _
            InStr(2, ActiveCell.Offset(dblxRow, iyCol).Address, "$") - 2)
iCol = ActiveCell.Column
dblRow = ActiveCell.Row

Application.ScreenUpdating = False

With Range(strCol & ":" & strCol & "," & dblRow + dblZ & ":" & dblRow + dblZ)
    .Select
    Application.ScreenUpdating = True
    .Item(dblRow + dblxRow).Activate
End With

NoGo:
End Sub

Sub ReSet() 'Ręczny reset
    Application.OnKey "{RIGHT}"
    Application.OnKey "{LEFT}"
    Application.OnKey "{UP}"
    Application.OnKey "{DOWN}"
End Sub

Poniższy fragment kodu należy umieścić w module ThisWorkbook:

Private Sub Workbook_Open()
    Application.OnKey "{RIGHT}", "HighlightRight"
    Application.OnKey "{LEFT}", "HighlightLeft"
    Application.OnKey "{UP}", "HighlightUp"
    Application.OnKey "{DOWN}", "HighlightDown"
    Application.OnKey "{DEL}", "DisableDelete"
End Sub

Private Sub Workbook_BeforeClose(Cancel As Boolean)
    Application.OnKey "{RIGHT}"
    Application.OnKey "{LEFT}"

```

```

Application.OnKey "{UP}"
Application.OnKey "{DOWN}"
Application.OnKey "{DEL}"
End Sub

```

Niestandardowe transponowanie danych

Nadesłał Masaru Kaji.

Mamy raport, w którym dane są skonfigurowane w wierszach (rysunek 14.8). Jednak potrzebny jest taki sposób formatowania danych, aby informacje o dacie i partii znalazły się w pojedynczym wierszu, natomiast dane dotyczące wartości i pozycji końcowych — w dalszych kolumnach (kolumny zawierające dane pozycji końcowych nie są widoczne na rysunku 14.9). Program zamieszczony poniżej wykonuje niestandardową transpozycję danych na podstawie podanej kolumny do postaci pokazanej na rysunku 14.9.

Rysunek 14.8.

W wyjściowym układzie danych w oddzielnych wierszach wyświetlają się podobne rekordy

	A	B	C	D	E
1	NazwaPozycji	DataPozycji	NrPartii	PozycjaKońcowa	Wartość
2	Ciepłne	2002-10-23	1	8	2,15
3	Ciepłne	2002-10-23	1	3	3,2
4	Ciepłne	2002-10-23	1	2	4,9
5	Ciepłne	2002-10-23	1	1	6,1
6	Ciepłne	2002-10-23	1	7	6,2
7	Ciepłne	2002-10-23	1	4	12,9
8	Ciepłne	2002-10-23	1	9	23
9	Ciepłne	2002-10-23	1	5	36
10	Ciepłne	2002-10-23	1	6	36,25
11	Ciepłne	2002-10-23	2	2	1,05
12	Ciepłne	2002-10-23	2	1	2,5
13	Ciepłne	2002-10-23	2	8	7,3
14	Ciepłne	2002-10-23	2	3	10,9
15	Ciepłne	2002-10-23	2	4	12,1
16	Ciepłne	2002-10-23	2	9	21,7

Rysunek 14.9.

Po przekształceniu pojedynczy wiersz zawiera wszystkie dane dotyczące określonej daty i partii

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
1	NazwaPozycji	DataPozycji	NrPartii	W1	W2	W3	W4	W5	W6	W7	W8	W9	W10	W11	W12
2	Ciepłne	2002-10-23	1	2,15	3,2	4,9	6,1	6,2	12,9	23	36	36,25			
3	Ciepłne	2002-10-23	2	1,05	2,5	7,3	10,9	12,1	21,7	33,25	43	43,25			
4	Ciepłne	2002-10-23	3	1,65	3,1	3,1	3,75	7,1	7,1	7,7	18,7	34	55,5		
5	Ciepłne	2002-10-23	4	1,1	2,75	4	9,5	14,3	25	37,75					
6	Ciepłne	2002-10-23	5	0,9	3,75	7,1	9	16	18,1	19,5	22,5	74,75			
7	Ciepłne	2002-10-23	6	1,6	3,4	5,2	7,8	8,2	9,4	11,5					
8	Ciepłne	2002-10-23	7	0,8	4,2	4,9	9,6	15	21,2	24,75	63,25				
9	Ciepłne	2002-10-23	8	0,7	6,2	8,4	10,3	10,6	12,3	28,75	31,75	52	76,75		
10	Ciepłne	2002-10-23	9	2,9	3,9	4,4	5,9	7	11,4	13,5	18,4	26,25	66,25		
11	Ciepłne	2002-10-24	1	1,4	3,85	6,2	8,1	10	12,3	17,2	27,5	37,5	55,5		
12	Ciepłne	2002-10-24	2	1,75	2,95	6	6,5	7,8	8,3	16,8					
13	Ciepłne	2002-10-24	3	1,15	5,4	8,7	9,9	10,9	11,8	13,3	17,1	24	37		
14	Ciepłne	2002-10-24	4	1,05	1,9	5,2	6,8	19,9							
15	Ciepłne	2002-10-24	5	2,5	3,15	3,15	4,2	6	12,2	12,3	19,9	23,2	25,25	42,25	150
16	Ciepłne	2002-10-24	6	2,4	2,95	4,4	6,5	8,7	14,2	22,9	22,9	25,75	51	58,25	59,5
17	Ciepłne	2002-10-24	7	1,2	3,35	6,3	9,5	11,3	12	14,4	36,25				
18	Ciepłne	2002-10-24	8	0,85	5	6,5	6,8	11,1	11,4	22,6					
19	Ciepłne	2002-10-24	9	2	3,3	4,5	6,8	8,6	9,7	20,5	30	58,5			
20	Ciepłne	2002-10-25	1	2,05	2,65	3,95	4,8	4,8	15,5	21	30	31,5	64,75	107,25	

```

Sub TransposeData()
Dim shOrg As Worksheet, shRes As Worksheet
Dim rngStart As Range, rngPaste As Range
Dim lngData As Long

Application.ScreenUpdating = False
On Error Resume Next
Application.DisplayAlerts = False
Sheets("WynikiTranspozycji").Delete
Application.DisplayAlerts = True
On Error GoTo 0

On Error GoTo terminate

Set shOrg = Sheets("DaneDoTranspozycji")
Set shRes = Sheets.Add(After:=shOrg)
shRes.Name = "WynikiTranspozycji"
With shOrg
    ' --Sortowanie
    .Cells.CurrentRegion.Sort Key1:=. [B2], Order1:=1, Key2:=. [C2], Order2:=1, _
        Key3:=. [E2], Order3:=1, Header:=xlYes
    ' --Skopiowanie tytułu
    .Rows(1).Copy shRes.Rows(1)
    ' --Ustawienie zakresu początkowego
    Set rngStart = . [C2]
    Do Until IsEmpty(rngStart)
        Set rngPaste = shRes.Cells(shRes.Rows.Count, 1).End(xlUp).Offset(1)
        lngData = GetNextRange(rngStart)
        rngStart.Offset(, -2).Resize(, 5).Copy rngPaste

        ' Skopiowanie wartości od W1 do W14
        rngStart.Offset(, 2).Resize(lngData).Copy
        rngPaste.Offset(, 5).PasteSpecial Paste:=xlAll, Operation:=xlNone, _
            SkipBlanks:=False, Transpose:=True
        ' Skopiowanie wartości W1PK do W14PK
        rngStart.Offset(, 1).Resize(lngData).Copy
        rngPaste.Offset(, 19).PasteSpecial Paste:=xlAll, Operation:=xlNone, _
            SkipBlanks:=False, Transpose:=True
        Set rngStart = rngStart.Offset(lngData)
    Loop
End With

Application.Goto shRes.[A1]
With shRes
    .Cells.Columns.AutoFit
    .Columns("D:E").Delete shift:=xlToLeft
End With

Application.ScreenUpdating = True
Application.CutCopyMode = False

If MsgBox("Czy chcesz usunąć wyjściowy arkusz danych?", 36) = 6 Then
    Application.DisplayAlerts = False

```

```
        Sheets("DaneDoTranspozycji").Delete
        Application.DisplayAlerts = True
    End If

    Set rngPaste = Nothing
    Set rngStart = Nothing
    Set shRes = Nothing

Exit Sub

terminate:
End Sub

Function GetNextRange(ByVal rngSt As Range) As Long
    Dim i As Long
    i = 0

    Do Until rngSt.Value <> rngSt.Offset(i).Value
        i = i + 1
    Loop

    GetNextRange = i
End Function
```

Zaznaczanie (anulowanie zaznaczenia) nieciągłego zakresu komórek

Nadesłał Tom Urtis.

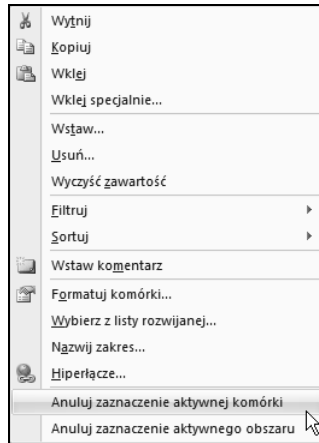
Standardowo usunięcie zaznaczenia pojedynczej komórki bądź zakresu komórek wymaga kliknięcia niezaznaczonej komórki w celu anulowania zaznaczenia wszystkich komórek, a następnie ponownego zaznaczenia nowego zakresu. Jest to niewygodne, gdy trzeba zaznaczyć wiele nieprzyległych do siebie komórek.

Makro zaprezentowane w tym punkcie dodaje do kontekstowego menu zaznaczenia dwie nowe opcje: *Anuluj zaznaczenie aktywnej komórki* oraz *Anuluj zaznaczenie aktywnego obszaru*. Aby zaznaczyć nieciągły zakres komórek, wystarczy przytrzymać klawisz *Ctrl*, kliknąć komórkę, której zaznaczenie chcemy anulować, zwolnić klawisz *Ctrl*, a następnie kliknąć prawym przyciskiem myszy komórkę, której zaznaczenie chcemy anulować. Wyświetli się menu kontekstowe pokazane na rysunku 14.10. Należy kliknąć pozycję menu, która powoduje anulowanie zaznaczenia pojedynczej aktywnej komórki lub ciągłego zaznaczonego obszaru, którego częścią jest ta komórka.

Poniższy fragment kodu należy umieścić w standardowym module:

```
Sub ModifyRightClick()
    ' Dodanie nowych opcji do menu dostępnego po kliknięciu prawym przyciskiem myszy
    Dim O1 As Object, O2 As Object
```


Rysunek 14.10.
 Procedura
 ModifyRightClick tworzy
 niestandardowe menu
 kontekstowe umożliwiające
 anulowanie zaznaczenia
 nieciągłego zakresu komórek



' Usunięcie opcji, jeśli już istnieją

```
On Error Resume Next
With CommandBars("Cell")
    .Controls("Anuluj zaznaczenie aktywnej komórki").Delete
    .Controls("Anuluj zaznaczenie aktywnego obszaru").Delete
End With
On Error GoTo 0
```

' Dodanie nowych opcji

```
Set O1 = CommandBars("Cell").Controls.Add

With O1
    .Caption = "Anuluj zaznaczenie aktywnej komórki"
    .OnAction = "DeselectActiveCell"
End With

Set O2 = CommandBars("Cell").Controls.Add

With O2
    .Caption = "Anuluj zaznaczenie aktywnego obszaru"
    .OnAction = "DeselectActiveArea"
End With

End Sub

Sub DeselectActiveCell()
Dim x As Range, y As Range

If Selection.Cells.Count > 1 Then
    For Each y In Selection.Cells
        If y.Address <> ActiveCell.Address Then
            If x Is Nothing Then
                Set x = y
            Else
                Set x = Application.Union(x, y)
            End If
        End If
    Next y
End Sub
```

```
        End If
    End If
Next y
If x.Cells.Count > 0 Then
    x.Select
End If
End If

End Sub

Sub DeselectActiveArea()
Dim x As Range, y As Range

If Selection.Areas.Count > 1 Then
    For Each y In Selection.Areas
        If Application.Intersect(ActiveCell, y) Is Nothing Then
            If x Is Nothing Then
                Set x = y
            Else
                Set x = Application.Union(x, y)
            End If
        End If
    Next y
    x.Select
End If
End Sub
```

Poniższe procedury należy dodać do modułu ThisWorkbook:

```
Private Sub Workbook_Activate()
ModifyRightClick
End Sub

Private Sub Workbook_Deactivate()
Application.CommandBars("Cell").Reset
End Sub
```

Techniki dla ekspertów języka VBA

Następne dziesięć przykładów sprawiło mnie w zachwyt. Programiści VBA należący do różnych społeczności w internecie ciągle wymyślają nowe sposoby wykonywania operacji szybciej lub lepiej. Kiedy ktoś opublikuje nowy kod, który jest zdecydowanie lepszy od najlepszego kodu wcześniej zaakceptowanego przez wszystkich, każdy na tym korzysta.

Rozwijane tabele przestawne

Nadesłał Tom Urtis.

Domyślne działanie tabel przestawnych po dwukrotnym kliknięciu sekcji danych polega na wstawieniu nowego arkusza i wyświetleniu w nim uszczegółowionych informacji. Poniższy przykład to dodana dla wygody opcja, pozwalająca na utrzymywanie uszczegółowionego zbioru rekordów w tym samym arkuszu, w którym znajduje się tabela przestawna (rysunek 14.11). Wstawione zbiory rekordów można następnie usunąć, jeśli trzeba. Aby skorzystać z tego makra, należy dwukrotnie kliknąć sekcję danych lub sekcję sum końcowych. Spowoduje to utworzenie uszczegółowionych zbiorów rekordów w następnym dostępnym wierszu tego samego arkusza. Aby usunąć dowolny z uszczegółowionych zbiorów rekordów, które utworzyliśmy wcześniej, wystarczy dwukrotnie kliknąć w dowolnym miejscu obszaru, który zajmuje.

Rysunek 14.11.
Wyświetlanie uszczegółowionych zbiorów rekordów w tym samym arkuszu, w którym znajduje się tabela przestawna

28	☒ Tomasz	Kw.2	10818	13047		
29	Tomasz Suma		10818	13047		
30	☒ William	Kw.4	3656			
31	William Suma		3656			
32	☒ Zofia	Kw.4	5278	6595	1803	86
33	Zofia Suma		5278	6595	1803	86
34	Suma końcowa		38022	37337	3564	14335
35						
36	Imię	Region	Kwartał	Towar	Kolor	Sprzedaż
37	Zofia	Wschodni	Kw.4	Kapelusze	Niebieski	86
38						

```
Private Sub Worksheet_BeforeDoubleClick(ByVal Target As Range, Cancel As
↳ Boolean)
Application.ScreenUpdating = False
Dim LPTR&

With ActiveSheet.PivotTables(1).DataBodyRange
LPTR = .Rows.Count + .Row - 1
End With

Dim PTT As Integer
On Error Resume Next
PTT = Target.PivotCell.PivotCellType
If Err.Number = 1004 Then
Err.Clear
If Not IsEmpty(Target) Then
If Target.Row > Range("A1").CurrentRegion.Rows.Count + 1 Then
Cancel = True
With Target.CurrentRegion
.Resize(.Rows.Count + 1).EntireRow.Delete
End With
End If
Else
Cancel = True
End If
Else
CS = ActiveSheet.Name
End If
Application.ScreenUpdating = True
End Sub
```

Szybka konfiguracja stron

Nadesłał Juan Pablo González Ruiz z Bogoty w Kolumbii. Juan Pablo jest programistą kreatora menu F&I i obsługuje wszystkie hiszpańskojęzyczne prośby dotyczące programowania w witrynie *MrExcel.com*.

Makro zaprezentowane w poniższym przykładzie porównuje różnice w czasie wykonania operacji modyfikacji marginesów z wartości domyślnej na 1,5 cala oraz stopki (nagłówek) na 1 cal w konfiguracji strony. Do utworzenia makra *Makro1* wykorzystano rejestrator makr. W kolejnych trzech makrach zaprezentowano sposób skrócenia czasu wykonywania zarejestrowanego kodu. Na rysunku 14.12 pokazano wyniki testu szybkości działania dla poszczególnych makr.

Rysunek 14.12.
Testy szybkości makr wykonujących konfigurację stron

	A	B	C	D
8	Makro1	Makro2	Makro3	Makro4
9	3,5702	0,8185	0,8149	0,1485
10	3,1218	0,8338	0,8257	0,1373
11	3,1133	0,8122	0,8165	0,1356
12	3,1374	0,8266	0,8296	0,1345
13	3,1579	0,8226	0,8228	0,1359
14	3,1158	0,8116	0,8205	0,1371
15	3,1363	0,8011	0,828	0,1373
16	3,1272	0,8111	0,8286	0,1343
17	3,1295	0,8093	0,8193	0,1334
18	3,1159	0,8106	0,8094	0,1334
19	3,1465	0,8026	0,8101	0,1335
20	3,1374	0,8269	0,8181	0,1385
21	3,1161	0,8274	0,8125	0,1367
22	3,1105	0,8249	0,8091	0,1373
23	3,1401	0,8227	0,8093	0,1381
24	3,1288	0,8065	0,825	0,1375
25	3,1908	0,8264	0,8264	0,1372
26	3,1788	0,8259	0,8223	0,1379
27	3,1602	0,8265	0,8248	0,1383
28	3,1184	0,8189	0,8324	0,1391
29	316%	82%	82%	14%
30	4	3	2	1

```
Sub Makro1()
'
' Macro1 Makro
' Makro zarejestrowano 3/28/2007
'
With ActiveSheet.PageSetup
    .PrintTitleRows = ""
    .PrintTitleColumns = ""
End With
ActiveSheet.PageSetup.PrintArea = ""
With ActiveSheet.PageSetup
    .LeftHeader = ""
    .CenterHeader = ""
    .RightHeader = ""
    .LeftFooter = ""
    .CenterFooter = ""
    .RightFooter = ""
End With
```

```

.LeftMargin = Application.InchesToPoints(1)
.RightMargin = Application.InchesToPoints(1)
.TopMargin = Application.InchesToPoints(1)
.BottomMargin = Application.InchesToPoints(1)
.HeaderMargin = Application.InchesToPoints(1)
.FooterMargin = Application.InchesToPoints(1)
.PrintHeadings = False
.PrintGridlines = False
.PrintComments = xlPrintNoComments
.PrintQuality = -3
.CenterHorizontally = False
.CenterVertically = False
.Orientation = xlPortrait
.Draft = False
.PaperSize = xlPaperLetter
.FirstPageNumber = 1
.Order = xlDownThenOver
.BlackAndWhite = False
.Zoom = False
.FitToPagesWide = 1
.FitToPagesTall = 1
.PrintErrors = xlPrintErrorsDisplayed
.OddAndEvenPagesHeaderFooter = False
.DifferentFirstPageHeaderFooter = False
.ScaleWithDocHeaderFooter = True
.AlignMarginsHeaderFooter = False
.EvenPage.LeftHeader.Text = ""
.EvenPage.CenterHeader.Text = ""
.EvenPage.RightHeader.Text = ""
.EvenPage.LeftFooter.Text = ""
.EvenPage.CenterFooter.Text = ""
.EvenPage.RightFooter.Text = ""
.FirstPage.LeftHeader.Text = ""
.FirstPage.CenterHeader.Text = ""
.FirstPage.RightHeader.Text = ""
.FirstPage.LeftFooter.Text = ""
.FirstPage.CenterFooter.Text = ""
.FirstPage.RightFooter.Text = ""

```

End With

End Sub

Rejestrator makr wykonuje mnóstwo nadmiarowych operacji wymagających dodatkowego czasu przetwarzania. W związku z tym faktem, a także z uwagi na to, że obiekt PageSetup jest jednym z wolniejszych obiektów, jeśli chodzi o aktualizację, uzyskane wyniki są takie, a nie inne. Oto poprawiona wersja (w której wykorzystano jedynie klawisz *Delete!*):

```
Sub Makro1_Wersja2()
```

```
With ActiveSheet.PageSetup
```

```

.LeftMargin = Application.InchesToPoints(1.5)
.RightMargin = Application.InchesToPoints(1.5)
.TopMargin = Application.InchesToPoints(1.5)
.BottomMargin = Application.InchesToPoints(1.5)

```

```

        .HeaderMargin = Application.InchesToPoints(1)
        .FooterMargin = Application.InchesToPoints(1)
    End With
End Sub

```

Powyższe makro działa szybciej niż *Makro1* (średnia redukcja czasu wykonywania sięga 70 % dla kilku prostych testów), ale makro to można usprawnić jeszcze bardziej. Jak wspomniano wcześniej, posługiwanie się obiektem PageSetup zajmuje dużo czasu, zatem jeśli zmniejszymy liczbę operacji, które musi wykonać kod VBA **oraz** dołączymy kilka funkcji IF w celu aktualizacji tylko tych właściwości, które wymagają modyfikacji, uzyskamy znacznie lepsze wyniki.

W przykładzie zamieszczonym poniżej zakodowano „na sztywno” wywołanie funkcji Application.InchesToPoints w celu ustawienia wartości w calach. Trzecia wersja makra *Makro1* ma następującą postać:

```

Sub Makro1_Wersja3()
    With ActiveSheet.PageSetup
        If .LeftMargin <> 108 Then .LeftMargin = 108
        If .RightMargin <> 108 Then .RightMargin = 108
        If .TopMargin <> 108 Then .TopMargin = 108
        If .BottomMargin <> 108 Then .BottomMargin = 108
        If .HeaderMargin <> 72 Then .HeaderMargin = 72
        If .FooterMargin <> 72 Then .FooterMargin = 72
    End With
End Sub

```

Jeśli dotąd nie zmieniliśmy wszystkich domyślnych marginesów, powinniśmy zauważyć różnicę.

Kolejna wersja makra pozwala na redukcję czasu wykonywania o ponad 95%! Makro wykorzystuje metodę języka XLM — PAGE.SETUP. Potrzebne parametry to left, right, top, bot, head_margin oraz foot_margin. Podaje się je w calach, a nie w punktach. Tak więc przy wykorzystaniu tych samych marginesów, które modyfikowaliśmy wcześniej, czwarta wersja makra *Makro1* ma następującą postać:

```

Sub Makro1_Wersja4()
    Dim St As String
    St = "PAGE.SETUP(, , " & _
        "1.5, 1.5, 1.5, 1.5" & _
        ", 0, False, False, False, 1, 1, True, 1, 1,False, , " & _
        "1, 1" & _
        ", False)"
    Application.ExecuteExcel4Macro St
End Sub

```

Linijki druga i czwarta instrukcji ustawiającej wartość zmiennej St odpowiadają wartościom wymienionych parametrów. Trzeba jednak zwrócić uwagę na kilka elementów. Po pierwsze, zaprezentowane makro wykorzystuje język XLM, którego obsługa jest w dalszym ciągu dostępna w Excelu w celu zachowania zgodności wstecz, ale nie wiemy, kiedy firma

Microsoft z niej zrezygnuje. Po drugie, należy zachować ostrożność przy ustawianiu parametrów funkcji PAGE.SETUP, jeśli bowiem jeden z nich będzie miał nieprawidłową wartość, funkcja PAGE.SETUP nie wykona się i nie wygeneruje błędu, co może spowodować, że ustawienia strony dla arkusza będą nieprawidłowe.

Obliczanie czasu wykonania kodu

Niektórzy czytelnicy pewnie zastanawiają się, w jaki sposób można obliczyć czas z dokładnością do tysięcznych sekundy, co pokazano wcześniej na rysunku 14.12.

Poniżej zaprezentowano kod służący do wygenerowania czasu wykonywania się makr w tym punkcie:

```
Public Declare Function QueryPerformanceFrequency _
    Lib "kernel32" (lpFrequency As Currency) As Long
Public Declare Function QueryPerformanceCounter _
    Lib "kernel32.dll" (lpPerformanceCount As Currency) As Long

Sub CalculateTime()
    Dim Ar(1 To 20, 1 To 4) As Currency, WS As Worksheet
    Dim n As Currency, str As Currency, fin As Currency
    Dim y As Currency

    Dim i As Long, j As Long

    Application.ScreenUpdating = False
    For i = 1 To 4
        For j = 1 To 20
            Set WS = ThisWorkbook.Sheets.Add
            WS.Range("A1").Value = 1
            QueryPerformanceFrequency y
            QueryPerformanceCounter str
            Select Case i
            Case 1: Makro1
            Case 2: Makro1_Wersja2
            Case 3: Makro1_Wersja3
            Case 4: Makro1_Wersja4
            End Select
            QueryPerformanceCounter fin
            Application.DisplayAlerts = False
            WS.Delete
            Application.DisplayAlerts = True
            n = (fin - str)
            Ar(j, i) = CCur(Format(n, "#####.#####") / y)
        Next j
    Next i
    With Range("A1").Resize(1, 4)
        .Value = Array("Makro1", "Makro2", "Makro3", "Makro4")
        .Font.Bold = True
    End With
    Range("A2").Resize(20, 4).Value = Ar
```

```

With Range("A22").Resize(1, 4)
    .FormulaR1C1 = "=AVERAGE(R2C:R21C)"
    .Offset(1).FormulaR1C1 = "=RANK(R22C,R22C1:R22C4,1)"
    .Resize(2).Font.Bold = True
End With
Application.ScreenUpdating = True
End Sub

```

Niestandardowy porządek sortowania

Nadesłał Wei Jiang z Shiyang City w Chinach. Jiang jest konsultantem w witrynie *MrExcel.com*.

Domyślnie Excel pozwala na sortowanie list numerycznie lub alfabetycznie, ale czasami chodzi nam o coś innego. Przykładowo klientowi mogą być potrzebne codzienne dane dotyczące sprzedaży posortowane zgodnie z domyślnym porządkiem obowiązującym na wydziale: torby, zegarki, portfele i wszystko inne. W przykładzie zamieszczonym w tym punkcie wykorzystano zdefiniowaną listę w celu posortowania zakresu danych zgodnie z niestandardowym porządkiem danego wydziału. Uzyskane wyniki pokazano na rysunku 14.13.

Rysunek 14.13.

Uruchomienie tego makra spowoduje, że lista w zakresie A:C najpierw zostanie posortowana według daty, a następnie zgodnie z listą definiującą niestandardowy porządek sortowania, umieszczoną w kolumnie I

	A	B	C	D	E	F	G	H	I	J
1	Data	Kategoria	Sprzedanych						Paski	
2	2007-01-01	Paski	15						Torebki	
3	2007-01-01	Torebki	23						Zegarki	
4	2007-01-01	Zegarki	42						Portfele	
5	2007-01-01	Portfele	17						Pozostałe	
6	2007-01-01	Pozostałe	36							
7	2007-01-02	Paski	17							
8	2007-01-02	Torebki	21							
9	2007-01-02	Zegarki	43							
10	2007-01-02	Portfele	18							
11	2007-01-02	Pozostałe	42							
12	2007-01-03	Paski	21							
13	2007-01-03	Torebki	20							
14	2007-01-03	Zegarki	35							
15	2007-01-03	Portfele	19							
16	2007-01-03	Pozostałe	45							

```
Sub CustomSort()
```

```
    ' Dodanie listy do zbioru zdefiniowanych list
```

```
    Application.AddCustomList ListArray:=Range("I1:I5")
```

```
    ' Pobranie numeru listy
```

```
    nIndex = Application.GetCustomListNum(Range("I1:I5").Value)
```

```
    ' Teraz można posortować zakres z wykorzystaniem niestandardowej listy
```

```
    ' Należy zwrócić uwagę, że jako numer listy niestandardowej trzeba wykorzystać wartość nIndex + 1,
```

```
    ' ponieważ pierwsza pozycja odpowiada porządkowi standardowemu
```

```
    Range("A2:C16").Sort Key1:=Range("B2"), Order1:=xlAscending, _
        Header:=xlNo, Orientation:=xlSortColumns, _
        OrderCustom:=nIndex + 1
```

```
    Range("A2:C16").Sort Key1:=Range("A2"), Order1:=xlAscending, _
        Header:=xlNo, Orientation:=xlSortColumns
```


¹ Na koniec należy usunąć niestandardową listę...

```
Application.DeleteCustomList nIndex
End Sub
```

Wskaźnik postępu wykonywania operacji w komórce

Nadesłał Tom Urtis.

Muszę przyznać, że nowe opcje formatowania warunkowego w Excelu, np. paski danych, są fantastyczne. Jednak nie pozwalają na uzyskanie takich efektów wizualnych, jakie pokazano na rysunku 14.14. Makro zaprezentowane w poniższym przykładzie tworzy wskaźnik postępu zadania w kolumnie C na podstawie wartości w kolumnach A i B.

Rysunek 14.14.
Wykorzystanie wskaźników w komórkach do prezentacji postępu wykonania zadania

	A	B	C
	Wykonany postep	Wymagany postep	Wizualna reprezentacja postępu w porównaniu z postępuem wymaganym
1			
2	11	15	██████████████████□□□□
3	14	20	██□□□□□
4	1	5	████□□□
5	4	10	██████□□□□□□
6	4	10	██████□□□□□□
7	10	10	██████████████████
8	8	10	██████████████□□
9		10	□□□□□□□□□□
10		10	□□□□□□□□□□
11		10	□□□□□□□□□□

```
Private Sub Worksheet_Change(ByVal Target As Range)
If Target.Column > 2 Or Target.Cells.Count > 1 Then Exit Sub
If Application.IsNumber(Target.Value) = False Then
Application.EnableEvents = False
Application.Undo
Application.EnableEvents = True
MsgBox "Proszę o wprowadzanie wyłącznie liczb."
Exit Sub
End If
Select Case Target.Column
Case 1
If Target.Value > Target.Offset(0, 1).Value Then
Application.EnableEvents = False
Application.Undo
Application.EnableEvents = True
MsgBox "Wartość w kolumnie A nie może być większa od wartości  
↳ w kolumnie B."
Exit Sub
End If
Case 2
If Target.Value < Target.Offset(0, -1).Value Then
Application.EnableEvents = False
Application.Undo
Application.EnableEvents = True
```

```

        MsgBox "Wartość w kolumnie B nie może być mniejsza " & _
            "niż wartość w kolumnie A."
    Exit Sub
End If
End Select
Dim x As Long
x = Target.Row
Dim z As String
z = Range("B" & x).Value - Range("A" & x).Value
With Range("C" & x)
    .Formula = "=IF(RC[-1]<=RC[-2],REPT(""n"",RC[-1]) _
        &REPT(""n"",RC[-2]-RC[-1]),REPT(""n"",RC[-2]) _
        &REPT(""o"",RC[-1]-RC[-2]))"
    .Value = .Value
    .Font.Name = "Wingdings"
    .Font.ColorIndex = 1
    .Font.Size = 10
    If Len(Range("A" & x)) <> 0 Then
        .Characters(1, (.Characters.Count - z)).Font.ColorIndex = 3
        .Characters(1, (.Characters.Count - z)).Font.Size = 12
    End If
End With
End Sub

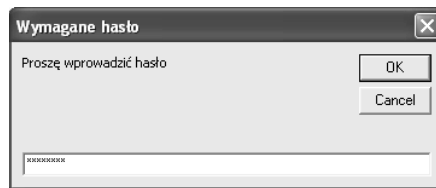
```

Chronione pole do wprowadzania hasła

Nadesłał Daniel Klann z Sydney w Australii. Daniel pracuje głównie w języku VBA w Excelu i Accessie, ale zdarza mu się programować w różnych językach. Prowadzi serwis WWW pod adresem www.danielklann.com.

Wykorzystanie pól tekstowych do wpisywania haseł ma zasadniczą wadę dotyczącą bezpieczeństwa. Wpisywane znaki można bez trudu podejrzeć. Poniższy program podczas wpisywania zamienia znaki na gwiazdki — dzięki temu pole tekstowe działa tak, jak klasyczne pole do wprowadzania haseł (rysunek 14.15).

Rysunek 14.15.
Wykorzystanie pól tekstowych jako narzędzi do bezpiecznego wprowadzania haseł



```

Private Declare Function CallNextHookEx Lib "user32" (ByVal hHook As Long, _
    ByVal ncode As Long, ByVal wParam As Long, lParam As Any) As Long

```

```

Private Declare Function GetModuleHandle Lib "kernel32" _
    Alias "GetModuleHandleA" (ByVal lpModuleName As String) As Long

```

```

Private Declare Function SetWindowsHookEx Lib "user32" _

```

```

Alias "SetWindowsHookExA" _
  (ByVal idHook As Long, ByVal lpfn As Long, _
  ByVal hmod As Long, ByVal dwThreadId As Long) As Long

Private Declare Function UnhookWindowsHookEx Lib "user32" _
  (ByVal hHook As Long) As Long

Private Declare Function SendDlgItemMessage Lib "user32" _
  Alias "SendDlgItemMessageA" _
  (ByVal hDlg As Long, _
  ByVal nIDDlgItem As Long, ByVal wParam As Long, _
  ByVal lParam As Long, ByVal lParam As Long) As Long

Private Declare Function GetClassName Lib "user32" _
  Alias "GetClassNameA" (ByVal hwnd As Long, _
  ByVal lpClassName As String, _
  ByVal nMaxCount As Long) As Long

Private Declare Function GetCurrentThreadId _
  Lib "kernel32" () As Long

' Stale używane w funkcjach API
Private Const EM_SETPASSWORDCHAR = &HCC
Private Const WH_CBT = 5
Private Const HCBT_ACTIVATE = 5
Private Const HC_ACTION = 0

Private hHook As Long

Public Function NewProc(ByVal lngCode As Long, _
  ByVal wParam As Long, ByVal lParam As Long) As Long
  DimRetVal
  Dim strClassName As String, lngBuffer As Long

  If lngCode < HC_ACTION Then
    NewProc = CallNextHookEx(hHook, lngCode, wParam, lParam)
    Exit Function
  End If

  strClassName = String$(256, " ")
  lngBuffer = 255

  If lngCode = HCBT_ACTIVATE Then ' Okno zostało uaktywnione

   RetVal = GetClassName(wParam, strClassName, lngBuffer)

    ' Sprawdzenie nazwy klasy dla pola tekstowego
    If Left$(strClassName, RetVal) = "#32770" Then
      ' Modyfikacja pola tekstowego w celu wyświetlenia znaku gwiazdki (*)
      ' Gwiazdkę można zastąpić dowolnym innym znakiem
      SendDlgItemMessage wParam, &H1324, EM_SETPASSWORDCHAR,
      ↪ Asc("*"), &H0
    End If
  End If

```

```

End If

' Ten wiersz zapewnia poprawne wywoływania innych
' zdefiniowanych haków
CallNextHookEx hHook, lngCode, wParam, lParam

End Function

Public Function InputBoxDK(Prompt, Optional Title, _
    Optional Default, Optional XPos, _
    Optional YPos, Optional HelpFile, Optional Context) As String
    Dim lngModHwnd As Long, lngThreadID As Long

    lngThreadID = GetCurrentThreadId
    lngModHwnd = GetModuleHandle(vbNullString)

    hHook = SetWindowsHookEx(WH_CBT, AddressOf NewProc, lngModHwnd,
        ↪lngThreadID)
    On Error Resume Next
    InputBoxDK = InputBox(Prompt, Title, Default, XPos, YPos, HelpFile,
        ↪Context)
    UnhookWindowsHookEx hHook

End Function

Sub PasswordBox()
If InputBoxDK("Proszę wprowadzić hasło", "Wymagane hasło") <> "hasło" Then
    MsgBox "Niestety, to nie było prawidłowe hasło."
Else
    MsgBox "Hasło prawidłowe! Zapraszamy."
End If
End Sub

```

Zmiana wielkości liter

Nadesłał Ivan F. Moala.

W Wordzie można zmienić wielkość liter w zaznaczonym tekście, funkcji tej wyraźnie brakuje w Excelu. Zaprezentowany program umożliwia użytkownikowi Excela zmianę wielkości liter w tekście dla dowolnego zaznaczonego zakresu (rysunek 14.16).

Rysunek 14.16.

Wykorzystanie tego makra umożliwia zmianę wielkości liter w poszczególnych słowach, tak jak w Wordzie



```

Sub TextCaseChange()
Dim RgText As Range
Dim oCell As Range
Dim Ans As String
Dim strTest As String
Dim sCap As Integer, _
    lCap As Integer, _
    i As Integer

' // Najpierw należy zaznaczyć zakres, którego będzie dotyczyła operacja!

Again:
Ans = Application.InputBox("[M]ałe litery" & vbCrLf & "[W]ielkie litery"
↳& vbCrLf & _
    "[J]ak w zdaniu" & vbCrLf & "[a]k w Nazwie Własnej" & vbCrLf & "[z]AMIANA
↳nA mAŁE/wIELKIE", _
    "Wpisz literę", Type:=2)

If Ans = "False" Then Exit Sub
If InStr(1, "MWJAZ", UCase(Ans), vbTextCompare) = 0 _
    Or Len(Ans) > 1 Then GoTo Again

On Error GoTo NoText
If Selection.Count = 1 Then
    Set RgText = Selection
Else
    Set RgText = Selection.SpecialCells(xlCellTypeConstants, 2)
End If
On Error GoTo 0

For Each oCell In RgText
    Select Case UCase(Ans)
        Case "M": oCell = LCase(oCell.Text)
        Case "W": oCell = UCase(oCell.Text)
        Case "J": oCell = UCase(Left(oCell.Text, 1)) & _
            LCase(Right(oCell.Text, Len(oCell.Text) - 1))
        Case "A": oCell = Application.WorksheetFunction.Proper(oCell.Text)
        Case "Z"
            lCap = oCell.Characters(1, 1).Font.Size
            sCap = Int(lCap * 0.85)
            ' Wszystko małymi literami
            oCell.Font.Size = sCap
            oCell.Value = UCase(oCell.Text)
            strTest = oCell.Value
            ' Wielkie litery dla pierwszej litery słów
            strTest = Application.Proper(strTest)
            For i = 1 To Len(strTest)
                If Mid(strTest, i, 1) = UCase(Mid(strTest, i, 1)) Then
                    oCell.Characters(i, 1).Font.Size = lCap
                End If
            Next i
    End Select
Next

```

```
Exit Sub
NoText:
MsgBox "W zaznaczonym obszarze nie ma tekstu @ " & Selection.Address

End Sub
```

Zaznaczanie komórek za pomocą metody SpecialCells

Nadesłał Ivan F. Moala.

Standardowo, jeśli w zakresie trzeba znaleźć określone wartości, tekst lub formuły, należy zaznaczyć zakres i sprawdzić każdą komórkę. W poniższym przykładzie pokazano sposób wykorzystania metody `SpecialCells` w celu zaznaczenia tylko pożądaných komórek. Konieczność sprawdzenia mniejszej liczby komórek powoduje przyspieszenie działania kodu.

```
Sub SpecialRange()
Dim TheRange As Range
Dim oCell As Range

Set TheRange = Range("A1:Z200").SpecialCells(
xlCellTypeConstants, xlTextValues)

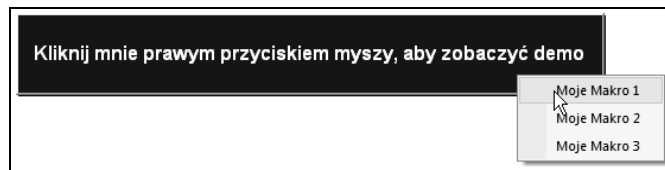
For Each oCell In TheRange
If oCell.Text = "Twój tekst" Then
MsgBox oCell.Address
MsgBox TheRange.Cells.Count
End If
Next oCell

End Sub
```

Menu prawego przycisku myszy dla obiektów ActiveX

W Excelu nie ma wbudowanego menu dla zdarzenia kliknięcia prawym przyciskiem myszy obiektów ActiveX w arkuszu. W tym punkcie zaprezentowano narzędzie służące do tego celu. Na rysunku 14.17 zaprezentowano użycie tego narzędzia dla przycisku polecenia. Właściwość `Take Focus on Click` przycisku polecenia należy ustawić na wartość `False`.

Rysunek 14.17.
Niestandardowe menu kontekstowe formantu ActiveX (dostępne po kliknięciu prawym przyciskiem myszy)



Poniższy fragment kodu należy umieścić w module `ThisWorkbook`:

```
Private Sub Workbook_Open()  
With Application  
    .CommandBars("Cell").Reset  
    .WindowState = xlMaximized  
    .Goto Sheet1.Range("A1"), True  
End With  
End Sub  
  
Private Sub Workbook_Activate()  
Application.CommandBars("Cell").Reset  
End Sub  
  
Private Sub Workbook_SheetBeforeRightClick(ByVal Sh As Object, _  
    ByVal Target As Range, Cancel As Boolean)  
Application.CommandBars("Cell").Reset  
End Sub  
  
Private Sub Workbook_Deactivate()  
Application.CommandBars("Cell").Reset  
End Sub  
  
Private Sub Workbook_BeforeClose(Cancel As Boolean)  
With Application  
    .CommandBars("Cell").Reset  
    .WindowState = xlMaximized  
    .Goto Sheet1.Range("A1"), True  
End With  
ThisWorkbook.Save  
End Sub
```

Poniższy fragment kodu należy umieścić w standardowym module:

```
Sub MyRightClickMenu()  
Application.CommandBars("Cell").Reset  
Dim cbc As CommandBarControl  
    For Each cbc In Application.CommandBars("cell").Controls  
        cbc.Visible = False  
    Next cbc  
With Application.CommandBars("Cell").Controls.Add(temporary:=True)  
    .Caption = "Moje makro 1"  
    .OnAction = "Test1"  
End With  
With Application.CommandBars("Cell").Controls.Add(temporary:=True)  
    .Caption = "Moje makro 2"  
    .OnAction = "Test2"  
End With  
With Application.CommandBars("Cell").Controls.Add(temporary:=True)  
    .Caption = "Moje makro 3"  
    .OnAction = "Test3"  
End With  
Application.CommandBars("Cell").ShowPopup  
End Sub
```

```

Sub Test1()
MsgBox "To jest makro Test1 z niestandardowego menu obiektu ActiveX " & _
    "dostępnego po kliknięciu prawym przyciskiem myszy.", , "Polecenie menu
    ↳ 'Moje makro 1'."
End Sub

Sub Test2()
MsgBox "To jest makro Test2 z niestandardowego menu obiektu ActiveX " & _
    "dostępnego po kliknięciu prawym przyciskiem myszy.", , "Polecenie menu
    ↳ 'Moje makro 2'."
End Sub

Sub Test3()
MsgBox "To jest makro Test3 z niestandardowego menu obiektu ActiveX " & _
    "dostępnego po kliknięciu prawym przyciskiem myszy.", , "Polecenie menu
    ↳ 'Moje makro 3'."
End Sub

```

Interesujące aplikacje

Ostatnie przykłady prezentują interesujące aplikacje, które mogą się przydać czytelnikom podczas realizacji własnych projektów.

Historyczne kursy akcji (funduszy)

Nadesłał Nathan P. Oliver.

Poniższe makro oblicza średnią z kursu podanej akcji lub wartość zamknięcia funduszu dla podanej daty (rysunek 14.18).

Rysunek 14.18.
Pobieranie informacji
giełdowych

	A	B	C
1	Symbol	Data	Średni/Zamknięcia
2	Dell	1994-01-12	25,3125
3	MSFT	2003-01-30	49,18
4	VFINX	2000-01-20	
5	INNDX	2003-01-06	
6	INSTX	2004-02-17	

```

Private Sub GetQuote()
Dim ie As Object, lCharPos As Long, sHTML As String
Dim HistDate As Date, HighVal As String, LowVal As String
Dim c1 As Range

```

```

Set c1 = ActiveCell
HistDate = c1(, 0)

```

```

If Intersect(c1, Range("C2:C" & Cells.Rows.Count)) Is Nothing Then
MsgBox "Należy zaznaczyć komórkę w kolumnie C."

```



```
Exit Sub
End If

If Not CBool(Len(cl, -1)) Or Not CBool(Len(cl, 0)) Then
    MsgBox "Należy podać symbol i datę."
    Exit Sub
End If

Set ie = CreateObject("InternetExplorer.Application")

With ie
    .Navigate _
        http://bigcharts.marketwatch.com/historical & _
        "/default.asp?detect=1&symbol=" & _
        & cl(, -1) & "&close_date=" & Month(HistDate) & "%2F" & _
        Day(HistDate) & "%2F" & Year(HistDate) & "&x=31&y=26"
    Do While .Busy And .ReadyState <> 4
        DoEvents
    Loop
    sHTML = .Document.body.innertext
    .Quit
End With

Set ie = Nothing

lCharPos = InStr(1, sHTML, "High:", vbTextCompare)
If lCharPos Then HighVal = Mid$(sHTML, lCharPos + 5, 15)

If Not Left$(HighVal, 3) = "n/a" Then
    lCharPos = InStr(1, sHTML, "Low:", vbTextCompare)
    If lCharPos Then LowVal = Mid$(sHTML, lCharPos + 4, 15)
    cl.Value = (Val(LowVal) + Val(HighVal)) / 2
Else: lCharPos = InStr(1, sHTML, "Closing Price:", vbTextCompare)
    cl.Value = Val(Mid$(sHTML, lCharPos + 14, 15))
End If

Set cl = Nothing
End Sub
```

Wykorzystanie rozszerzalności języka VBA w celu dodawania kodu do nowych skoroszytów

Mamy makro, które przenosi dane przeznaczone dla regionalnych kierowników do nowego skoroszytu. Co zrobić, jeśli jednocześnie trzeba skopiować makra do nowego skoroszytu? Można skorzystać z własności rozszerzalności języka Visual Basic for Application w celu zaimportowania modułów do skoroszytu lub zapisania do niego fragmentów kodu.

Aby wykorzystać dowolny z tych przykładów, należy najpierw otworzyć edytor VB, wybrać pozycję *References* z menu *Tools* i zaznaczyć referencję do biblioteki *Microsoft Visual Basic for Applications Extensibility 5.3*. Trzeba również włączyć opcję zaufania dostępowi do

obiektów VBA. W tym celu należy przejść do wstążki *Deweloper*, wybrać polecenie *Bezpieczeństwo makr* i zaznaczyć opcję *Ufaj dostępowi do modelu obiektowego projektu VBA*.

Najłatwiejszym sposobem zastosowania własności rozszerzalności języka VBA jest wyeksportowanie całego modułu lub formularza z bieżącego projektu i zaimportowanie go do nowego skoroszytu. Załóżmy, że mamy aplikację składającą się z wielu tysięcy wierszy kodu. Chcemy utworzyć nowy skoroszyt zawierający dane dla regionalnego kierownika i przekazać mu trzy makra pozwalające na niestandardowe formatowanie i drukowanie. Wszystkie te makra umieścimy w module o nazwie `modToRegion`. Makra z tego modułu wywołują formularz `frmRegion`. Poniższy kod przenosi ten kod z bieżącego skoroszytu do nowego skoroszytu:

```
Sub MoveDataAndMacro()
    Dim WSD as worksheet
    Set WSD = Worksheets("Raport")
    ' Skopiowanie raportu do nowego skoroszytu
    WSD.Copy
    ' Aktywnym skoroszytem jest teraz nowy skoroszyt
    ' Usunięcie starej kopii modułu z dysku C
    On Error Resume Next
    ' Usunięcie niepotrzebnych kopii z dysku twardego
    Kill ("C:\ModToRegion.bas")
    Kill ("C:\frmRegion.frm")
    On Error GoTo 0
    ' Wyeksportowanie modułu i formularza z tego skoroszytu
    ThisWorkbook.VBProject.VBComponents("ModToRegion").Export _
        ("C:\ModToRegion.bas")
    ThisWorkbook.VBProject.VBComponents("frmRegion").Export
    ↵("C:\frmRegion.frm")
    ' Zaimportowanie do nowego skoroszytu
    ActiveWorkbook.VBProject.VBComponents.Import ("C:\ModToRegion.bas")
    ActiveWorkbook.VBProject.VBComponents.Import ("C:\frmRegion.frm")
    On Error Resume Next
    Kill ("C:\ModToRegion.bas")
    Kill ("C:\frmRegion.bas")
    On Error GoTo 0
End Sub
```

Pokazana powyżej metoda zadziała, jeśli będzie trzeba przenieść moduły lub formularze użytkownika do nowego skoroszytu. Co jednak zrobić, jeżeli musimy wpisać kod w makrze *Workbook_Open* w module *ThisWorkbook*? Można skorzystać z dwóch narzędzi. Metoda *Lines* zwraca określony zbiór wierszy kodu z podanego modułu. Metoda *InsertLines* pozwala na wstawienie linii kodu do nowego modułu.

```
Sub MoveDataAndMacro()
    Dim WSD as worksheet
    Dim WBN as Workbook
    Dim WBCCodeMod1 As Object, WBCCodeMod2 As Object
    Set WSD = Worksheets("Raport")
```

OSTRZEŻENIE

Każde wywołanie metody `InsertLines` powinno wstawiać kompletne makro. Po każdym wywołaniu metody `InsertLines` Excel podejmuje próbę skompilowania kodu. W przypadku wstawienia wierszy, które nie mogą się poprawnie skompilować, Excel może się zawiesić, zgłaszając ogólny błąd zabezpieczeń (ang. *General Protection Fault* — GPF).

```
' Skopiowanie raportu do nowego skoroszytu
WSD.Copy
' Aktywnym skoroszytem jest teraz nowy skoroszyt
Set WBN = ActiveWorkbook
' Skopiowanie procedur obsługi zdarzeń poziomu skoroszytu
Set WBCodeMod1 = ThisWorkbook.VBProject.VBComponents("ThisWorkbook") _
.CodeModule
Set WBCodeMod2 = WBN.VBProject.VBComponents("ThisWorkbook").CodeModule
WBCodeMod2.InsertLines 1, WBCodeMod1.Lines(1, WBCodeMod1.CountOfLines)
End Sub
```

Następne kroki

Excel 2007 oferuje nowe, fantastyczne narzędzia wizualizacji danych, takie jak słupki danych, skale kolorów, zbiory ikon, oraz ulepszone reguły formatowania. W rozdziale 15., „Wizualizacja danych i formatowanie warunkowe”, nauczymy się, w jaki sposób można zautomatyzować nowe narzędzia i wykorzystać język VBA do wywoływania opcji niedostępnych w środowisku interfejsu użytkownika Excela.