W prostocie tkwi siła

Wydanie III

Excel 2013 PL Programowanie w VBA

DLA BYSTRZAKÓW™

Dowiedz się, jak:

- używać niezbędnych narzędzi i operacji języka VBA
- optymalnie wykorzystywać rejestrator makr w Excelu
- radzić sobie z błędami w kodzie i je eliminować
- personalizować funkcje arkusza i rozwijać aplikacje przyjazne dla użytkowników

septem
septem.pl

John Walkenbach

Tytuł oryginalny: Excel® VBA Programming For Dummies®, 3rd Edition

Tłumaczenie: Ryszard Górnowicz, Grzegorz Kowalczyk

ISBN: 978-83-246-7950-8

Drogi Czytelniku!
Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres
http://dlabystrzakow.pl/user/opinie/e13pvb
Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Wydawnictwo HELION
ul. Kościuszki 1c, 44-100 Gliwice
tel. 32 231 22 19, 32 230 98 63
e-mail: dlabystrzakow@dlabystrzakow.pl
WWW: http://dlabystrzakow.pl

Pliki z przykładami omawianymi w książce można znaleźć pod adresem:
ftp://ftp.helion.pl/przyklady/e13pvb.zip

Printed in Poland.

Spis treści

Część IV: Komunikacja z użytkownikiem 239

Rozdział 14

Przykłady i techniki programowania w języku VBA

W tym rozdziale:

- poznasz szereg przykładów technik programowania w języku VBA,
- dowiesz się, jak możesz przyspieszyć działanie kodu VBA w Twojej aplikacji.

Wierzę, że nauka programowania jest znacznie szybsza i zdecydowanie bardziej efektywna, kiedy pracujemy na konkretnych przykładach. Dobrze opracowany przykład o wiele lepiej objaśnia zagadnienie niż najbardziej rozbudowany i szczegółowy, ale teoretyczny opis. Ponieważ czytasz tę książkę, prawdopodobnie zgadzasz się ze mną w tej materii. W tym rozdziale znajdziesz szereg przykładów demonstrujących użyteczne, praktyczne techniki programowania w języku VBA.

Przykłady omawiane w tym rozdziale zostały podzielone na kilka kategorii. Oto one.

- ✓ Praca z zakresami.
- ✓ Modyfikacja ustawień Excela.
- ✓ Praca z wykresami.
- ✓ Przyspieszanie i optymalizacja działania kodu VBA.

Niektóre z prezentowanych przykładów będziesz mógł od razu wykorzystać w swoich aplikacjach, jednak w większości przypadków będą wymagały pewnego dostosowania do Twoich aplikacji.

Przetwarzanie zakresów komórek

Większość zadań, z jakimi będziesz się stykał, programując w języku VBA, będzie zapewne wymagała mniejszego bądź większego przetwarzania zakresów komórek (aby odświeżyć sobie wiadomości na temat obiektu `Range`, powinieneś zajrzeć do rozdziału 8.). Kiedy pracujesz z obiektami `Range`, powinieneś pamiętać o następujących sprawach.

- ✓ Kod VBA *nie musi zaznaczać* danego zakresu, aby go przetwarzać.
- ✓ Jeżeli kod VBA zaznacza wybrany zakres, przechowujący go skoroszyt musi być aktywny.

- ✓ Rejestrator makr nie zawsze będzie w stanie wygenerować optymalny kod VBA. Bardzo często jednak możesz za jego pomocą utworzyć bazowe makro i potem odpowiednio zmodyfikować kod tak, aby stał się bardziej efektywny.
- ✓ Zazwyczaj bardzo dobrym rozwiązaniem jest nadawanie nazw zakresom komórek wykorzystywanym w kodzie VBA. Przykładowo polecenie `Range("Total")`jest znacznie lepszym rozwiązaniem niż `Range("D45")`. Jeśli w tym drugim przypadku później wstawisz dodatkowy wiersz powyżej wiersza `45`, to żeby wszystko działało poprawnie, będziesz musiał zmodyfikować makro tak, aby korzystało z nowego, poprawnego adresu komórki, a ta po wykonaniu takiej operacji będzie miała inny adres (`D46`). Aby nadać nazwę wybranemu zakresowi komórek, powinieneś przejść na kartę *FORMUŁY* i wybrać polecenie *Definiuj nazwę*, znajdujące się w grupie poleceń *Nazwy zdefiniowane*.
- ✓ Kiedy tworzysz makro, które będzie przetwarzało zakres komórek zaznaczony przez użytkownika, pamiętaj, że użytkownik może zaznaczyć kilka całych kolumn czy wierszy. W większości przypadków z pewnością nie będziesz chciał, aby makro w pętli przechodziło w takiej sytuacji przez wszystkie zaznaczone komórki (włącznie z pustymi), co mogłoby zająć bardzo wiele czasu. Dobre makro powinno odszukać i przetwarzać tylko komórki, które nie są puste.
- ✓ Excel pozwala na jednoczesne zaznaczanie wielu zakresów komórek. Aby to zrobić, powinieneś zaznaczyć pierwszy zakres komórek, potem wcisnąć i przytrzymać klawisz *Ctrl*, i zaznaczać kolejne zakresy komórek przy użyciu myszy. Kod Twojej aplikacji powinien być przygotowany na takie sytuacje i podejmować odpowiednie akcje.

Skoroszyty z wybranymi przykładami omawianymi w tym rozdziale znajdziesz na stronie internetowej naszej książki.

Jeżeli chcesz samodzielnie wpisywać kod omawianych przykładów, przejdź do edytora VBE, naciskając kombinację klawiszy *lewyAlt+F11*, a następnie wstaw nowy moduł VBA i wpisz kod prezentowanych procedur. Upewnij się, że Twój skoroszyt jest poprawnie skonfigurowany. Jeżeli na przykład kod danego przykładu odwołuje się do arkuszy o nazwach `Arkusz1` i `Arkusz2`, upewnij się, że takie arkusze istnieją w Twoim skoroszycie.

Kopiowanie zakresów

Kopiowanie zakresów komórek może śmiało pretendować do miana jednej z najczęściej wykonywanych operacji w Excelu. Kiedy włączysz rejestrator makr i skopiujesz zakres komórek o adresie `A1:A5` do zakresu `B1:B5`, otrzymasz następujące makro.

```
Sub CopyRange()
    Range("A1:A5").Select
    Selection.Copy
    Range("B1").Select
    ActiveSheet.Paste
    Application.CutCopyMode = False
End Sub
```

Zwróć uwagę na ostatnie polecenie. Zostało ono wygenerowane przez naciśnięcie klawisza *Esc* po skopiowaniu zakresu komórek, co spowodowało usunięcie przerywanej linii, reprezentującej na arkuszu obramowanie kopiowanego zakresu.

Przedstawione makro działa poprawnie, ale zakresy komórek można kopiować w znacznie bardziej efektywny sposób. Identyczny rezultat możesz osiągnąć za pomocą procedury, która składa się z tylko jednego wiersza polecenia i nie zaznacza żadnych komórek (dzięki czemu nie wymaga ustawiania właściwości `CutCopyMode` na wartość *False*).

```
Sub CopyRange2()
    Range("A1:A5").Copy Range("B1")
End Sub
```

Procedura przedstawiona powyżej korzysta z tego, że metoda `Copy` może używać argumentu wywołania reprezentującego miejsce docelowe kopiowanego zakresu. Informacje o tym znalazłem w pomocy systemowej VBA. Powyższy przykład doskonale również ilustruje fakt, że rejestrator makr nie zawsze generuje najbardziej efektywny kod.

Kopiowanie zakresu o zmiennej wielkości

W wielu przypadkach konieczne jest skopiowanie zakresu komórek, dla którego dokładna liczba wierszy i kolumn określających jego wielkość nie jest z góry znana. Przykładowo możesz dysponować skoroszytem śledzącym tygodniową sprzedaż, w którym liczba wierszy zmienia się każdego tygodnia po wprowadzeniu nowych danych.

Na rysunku 14.1 przedstawiam przykład często spotykanego typu arkusza. Znajdujący się w nim zakres komórek składa się z kilku wierszy, których liczba zmienia się każdego dnia. Ponieważ nie wiesz, jaki jest rozmiar zakresu w danej chwili, musisz utworzyć kod, który będzie działał bez używania adresu zakresu kopiowanych komórek.

	A	B	C	D	E
1	Data	Liczba sztuk	Kwota		
2	2013-08-04	181	6 697,00		
3	2013-08-05	174	5 742,00		
4	2013-08-06	201	7 437,00		
5	2013-08-07	229	8 473,00		
6	2013-08-08	203	6 496,00		
7	2013-08-09	229	7 328,00		
8	2013-08-10	213	8 094,00		
9	2013-08-11	219	8 322,00		
10	2013-08-12	236	7 780,00		
11	2013-08-13	261	8 613,00		
12	2013-08-14	262	8 646,00		
13					
14					
15					

Arkusz1 Arkusz2

Rysunek 14.1. Przykład zakresu, który może składać się z dowolnej liczby wierszy

Makro przedstawione poniżej ilustruje sposób kopiowania zakresu komórek z arkusza `Arkusz1` do arkusza `Arkusz2` (począwszy od komórki `A1`). Makro wykorzystuje właściwość `CurrentRegion`, która zwraca obiekt `Range` odpowiadający blokowi komórek otaczających określoną komórkę (w tym przypadku o adresie `A1`).

```
Sub CopyCurrentRegion()
    Range("A1").CurrentRegion.Copy
    Sheets("Arkusz2").Select
    Range("A1").Select
    ActiveSheet.Paste
    Sheets("Arkusz1").Select
    Application.CutCopyMode = False
End Sub
```

Zastosowanie właściwości `CurrentRegion` jest równoważne z przejściem na kartę *NARZĘDZIA GŁÓWNE* i wybraniem polecenia *Znajdź i zaznacz/Przejdź do — specjalnie*, znajdującego się w grupie opcji *Edytowanie*, i następnie zaznaczeniem opcji *Bieżący obszar*. Aby przekonać się, jak to działa, podczas wykonywania tych poleceń powinieneś użyć rejestratora makr. Zazwyczaj wartość właściwości `CurrentRegion` reprezentuje prostokątny blok komórek otoczony przez puste wiersze i kolumny.

Oczywiście, możesz zoptymalizować kod makra przedstawionego powyżej i nie zaznaczać obszaru docelowego dla kopiowanych komórek. Makro przedstawione poniżej korzysta z faktu, że metoda `Copy` może używać argumentu wywołania reprezentującego miejsce docelowe kopiowanego zakresu.

```
Sub CopyCurrentRegion2()
    Range("A1").CurrentRegion.Copy _
        Sheets("Arkusz2").Range("A1")
End Sub
```

Jeżeli zakres komórek, który chcesz skopiować, jest tabelą (zdefiniowaną przy użycia polecenia *WSTAWIANIE/Tabele/Tabela*), całe zadanie będzie jeszcze łatwiejsze. Każda tabela posiada swoją nazwę (na przykład `Tabela1`) i automatycznie rozszerza się w miarę dodawania nowych wierszy.

```
Sub CopyTable()
    Range("Tabela1").Copy Sheets("Arkusz2").Range("A1")
End Sub
```

Jeżeli spróbujesz wykonać procedurę przedstawioną powyżej, przekonasz się, że wiersz nagłówka tabeli nie jest kopiowany, ponieważ obiekt `Tabela1` nie obejmuje tego wiersza. Jeśli chcesz, aby wiersz nagłówka również był kopiowany, powinieneś zmienić odwołanie do tabeli tak, jak to zostało przedstawione poniżej.

```
Range("Tabela1[#All]")
```

Zaznaczanie komórek do końca wiersza lub kolumny

Prawdopodobnie bardzo często używasz kombinacji klawiszy, takich jak *Ctrl*+*Shift*+→, czy *Ctrl*+*Shift*+↓, do zaznaczania zakresów składających się ze wszystkich komórek, od komórki aktywnej, aż do końca kolumny czy wiersza. Nie jest więc chyba zaskoczeniem, że możesz napisać makro, które będzie zaznaczać komórki w podobny sposób.

Do zaznaczania całego bloku komórek możesz użyć właściwości `CurrentRegion`. Ale co powinieneś zrobić, jeżeli chcesz zaznaczyć, powiedzmy, tylko jedną kolumnę z tego bloku komórek? Na szczęście, VBA pozwala na wykonywanie takich operacji. Procedura, której kod przedstawiam poniżej, zaznacza zakres komórek, począwszy od bieżącej aktywnej komórki w dół kolumny, aż do komórki znajdującej się o jeden wiersz powyżej pierwszej pustej komórki tej kolumny. Po zaznaczeniu zakresu możesz przetwarzać go w dowolny sposób — kopiować komórki, przenosić je w inne miejsce arkusza, zmieniać formatowanie i tak dalej.

```
Sub SelectDown()
    Range(ActiveCell, ActiveCell.End(xlDown)).Select
End Sub
```

Oczywiście, taki sam obszar możesz również zaznaczyć ręcznie. Aby to zrobić, powinieneś zaznaczyć pierwszą komórkę, następnie wcisnąć i przytrzymać klawisz *Shift*, nacisnąć klawisz *End* i wreszcie nacisnąć klawisz ↓ (strzałka w dół).

W przykładzie wykorzystuję metodę `End` obiektu `ActiveCell`, która zwraca obiekt typu `Range`. Metoda `End` pobiera jeden argument określający kierunek, w którym zostanie wykonane zaznaczenie. Argumentami tej metody może być dowolna ze stałych przedstawionych poniżej:

- ✓ `xlUp`,
- ✓ `xlDown`,
- ✓ `xlToLeft`,
- ✓ `xlToRight`.

Pamiętaj, że zaznaczanie zakresu nie jest potrzebne do jego przetwarzania. Makro przedstawione poniżej zmienia czcionkę w komórkach zmiennego zakresu (pojedyncza kolumna) na pogrubioną bez uprzedniego zaznaczenia zakresu.

```
Sub MakeBold()
    Range(ActiveCell, ActiveCell.End(xlDown)) _
        .Font.Bold = True
End Sub
```

Zaznaczanie całego wiersza lub całej kolumny

Procedura przedstawiona poniżej ilustruje sposób zaznaczania kolumny, w której znajduje się aktywna komórka. Makro wykorzystuje właściwość `EntireColumn`, która zwraca obiekt typu `Range` reprezentujący całą kolumnę.

```
Sub SelectColumn()
    ActiveCell.EntireColumn.Select
End Sub
```

Jak pewnie się spodziewasz, w języku VBA dostępna jest również właściwość `EntireRow`, która zwraca obiekt typu `Range` reprezentujący cały wiersz.

Przenoszenie zakresów

Zazwyczaj, aby przenieść zakres komórek, zaznaczasz go, wycinasz do schowka systemowego i następnie wklejasz w inne miejsce. Jeżeli użyjesz rejestratora makr do zapisania takiej operacji, przekonasz się, że wygenerowany zostanie kod podobny do przedstawionego poniżej.

```
Sub MoveRange()
    Range("A1:C6").Select
    Selection.Cut
    Range("A10").Select
    ActiveSheet.Paste
End Sub
```

Podobnie jak podczas kopiowania komórek, takie rozwiązanie nie jest najbardziej efektywnym sposobem przenoszenia zakresu komórek w inne miejsce. W praktyce taką operację możesz wykonać za pomocą procedury składającej się z jednego wiersza kodu, co prezentuję poniżej.

```
Sub MoveRange2()
    Range("A1:C6").Cut Range("A10")
End Sub
```

Makro przedstawione powyżej korzysta z faktu, że metoda `Cut` może używać argumentu wywołania reprezentującego miejsce docelowe przenoszonego zakresu. Zwróć również uwagę na fakt, że podczas przenoszenia żaden zakres komórek nie jest zaznaczany. Wskaźnik aktywnej komórki przez cały czas pozostaje w tym samym miejscu arkusza.

Wydajne przetwarzanie komórek zaznaczonego zakresu przy użyciu pętli

Jednym z zadań często wykonywanych przez makra jest sprawdzanie poszczególnych komórek zakresu i wykonywanie określonych operacji na podstawie ich zawartości. Takie makra zazwyczaj wykorzystują pętlę `For-Next`, za pomocą której przetwarzane są komórki zakresu.

Przykład przedstawiony niżej ilustruje sposób przechodzenia kolejno przez wszystkie komórki danego zakresu. W naszym przypadku przetwarzany jest aktualnie zaznaczony zakres komórek. Zmienna obiektowa o nazwie `Cell` reprezentuje aktualnie przetwarzaną komórkę. W pętli `For Each-Next` znajduje się jedno polecenie, które sprawdza aktualnie przetwarzaną komórkę i zmienia jej czcionkę na pogrubioną, jeżeli wartość przechowywana w komórce jest dodatnia.

```
Sub ProcessCells()
    Dim Cell As Range
    For Each Cell In Selection
        If Cell.Value > 0 Then Cell.Font.Bold = True
    Next Cell
End Sub
```

Taka procedura działa poprawnie, ale co się stanie, jeżeli użytkownik zaznaczy całą kolumnę lub cały wiersz? To wcale nie jest takie nieprawdopodobne, bo przecież Excel pozwala na wykonywanie operacji na całych wierszach i kolumnach. W takiej sytuacji wykonanie makra może zająć naprawdę dużo czasu, ponieważ nasza pętla przetwarza każdą komórkę zaznaczonego zakresu, a łącznie z pustymi w jednej kolumnie komórek jest aż 1 048 576… Aby zatem nasze makro było bardziej wydajne, musimy je tak zmodyfikować, żeby przetwarzane były tylko i wyłącznie komórki, które nie są puste.

Procedura przedstawiona poniżej przetwarza wyłącznie niepuste komórki zaznaczonego zakresu dzięki zastosowaniu metody `SpecialCells` (więcej szczegółowych informacji na temat tej metody znajdziesz w pomocy systemowej VBA). Nasza procedura za pomocą polecenia `Set` tworzy dwa obiekty typu `Range`: pierwszy z nich to podzakres komórek zakresu wejściowego, zawierający wyłącznie komórki z wartościami stałymi (na przykład teksty, wartości liczbowe, literały i tak dalej), a drugi składa się z komórek zawierających formuły. Procedura przetwarza tylko komórki należące do tych podzakresów, co w efekcie powoduje pominięcie przetwarzania wszystkich pozostałych, pustych komórek zakresu wejściowego. Sprytne, prawda?

```
Sub SkipBlanks()
    Dim ConstantCells As Range
    Dim FormulaCells As Range
    Dim cell As Range
'   Ignoruj błędy
    On Error Resume Next

'   Przetwarzaj komórki zawierające wartości stałe
    Set ConstantCells = Selection _
        .SpecialCells(xlConstants)
    For Each cell In ConstantCells
        If cell.Value > 0 Then
            cell.Font.Bold = True
        End If
    Next cell

'   Przetwarzaj komórki zawierające formuły
    Set FormulaCells = Selection _
        .SpecialCells(xlFormulas)
    For Each cell In FormulaCells
        If cell.Value > 0 Then
            cell.Font.Bold = True
        End If
    Next cell
End Sub
```

Procedura `SkipBlanks` działa tak samo szybko, niezależnie od tego, jaki zakres komórek zaznaczyłeś. Możesz na przykład zaznaczyć zakres składający się z kilku komórek, zaznaczyć wszystkie kolumny w danym zakresie albo wszystkie wiersze w danym zakresie, albo nawet cały arkusz. Jak widać, jest to ogromne usprawnienie w stosunku do oryginalnej procedury `ProcessCells`, którą omówiłem nieco wcześniej.

Zwróć uwagę, że w kodzie procedury użyliśmy polecenia:

```
On Error Resume Next
```

Polecenie to powoduje, że Excel po prostu ignoruje błędy (inaczej mówiąc, jeżeli próba wykonania danego polecenia kończy się błędem, Excel ignoruje ten błąd i po prostu przechodzi do kolejnego polecenia; więcej szczegółowych informacji na temat obsługi błędów znajdziesz w rozdziale 12.). W naszym przypadku zastosowanie polecenia `On Error` jest konieczne, ponieważ metoda `SpecialCells` generuje błąd, gdy żadna komórka nie spełnia podanego kryterium.

Zastosowanie metody `SpecialCells` jest równoważne z przejściem na kartę *NARZĘDZIA GŁÓWNE*, wybraniem polecenia *Znajdź i zaznacz/Przejdź do — specjalnie*, znajdującego się w grupie opcji *Edytowanie*, i następnie zaznaczeniem opcji *Stałe* lub *Formuły*. Aby przekonać się, jak to działa, podczas wykonywania tych poleceń powinieneś użyć rejestratora makr i zaznaczać różne opcje.

Wydajne przetwarzanie komórek zaznaczonego zakresu przy użyciu pętli (część II)

A teraz ciąg dalszy naszej opowieści. W tym punkcie przedstawię inny sposób efektywnego przetwarzania komórek znajdujących się w zaznaczonym zakresie. Tym razem procedura będzie korzystała z właściwości `UsedRange`, która zwraca obiekt typu `Range`, reprezentujący używany zakres arkusza. Procedura korzysta również z metody `Intersect`, która zwraca obiekt typu `Range` zawierający komórki będące częścią wspólną dwóch zakresów.

Poniżej przedstawiam zmodyfikowaną wersję procedury `SkipBlanks`, omawianej w poprzednim punkcie.

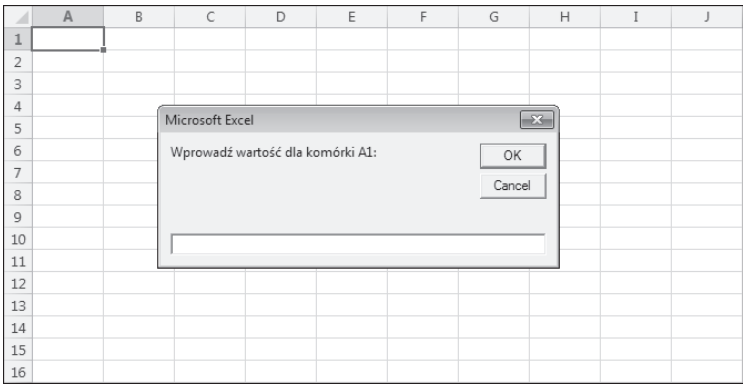
```
Sub SkipBlanks2()
    Dim WorkRange As Range
    Dim cell As Range
    Set WorkRange = Intersect(Selection, ActiveSheet.UsedRange)
    For Each cell In WorkRange
        If cell.Value > 0 Then
            cell.Font.Bold = True
        End If
    Next cell
End Sub
```

Zmienna obiektowa `WorkRange` zawiera komórki, które są częścią wspólną zakresu zaznaczonego przez użytkownika oraz zakresu używanych komórek arkusza. Jeśli zatem użytkownik zaznaczy całą kolumnę, zmienna `WorkRange` będzie zawierała tylko komórki, które znajdują się jednocześnie w zaznaczonej kolumnie i w używanym zakresie arkusza. Jak widać, jest to bardzo szybka i efektywna metoda pozwalająca na uniknięcie przetwarzania komórek znajdujących się poza zakresem używanych komórek arkusza.

Wprowadzanie wartości do komórki

Na rysunku 14.2 pokazuję, w jaki sposób możesz użyć funkcji `InputBox` języka VBA do pobierania od użytkownika wartości, która następnie może zostać zapisana w wybranej komórce. W procedurze przedstawionej poniżej demonstruję, jak poprosić użytkownika o podanie wartości i wstawić ją do komórki `A1` aktywnego arkusza (i to wszystko za pomocą jednego polecenia).

```
Sub GetValue()
    Range("A1").Value = InputBox( _
        "Wprowadź wartość dla komórki A1:")
End Sub
```



Rysunek 14.2. Zastosowanie funkcji InputBox do pobierania wartości od użytkownika

Przedstawiona procedura może jednak sprawiać pewien problem. Jeżeli użytkownik naciśnie w oknie dialogowym przycisk *Cancel*, procedura usunie wszelkie dane już znajdujące się w komórce `A1`, a takie zachowanie nie jest zbyt dobrą praktyką programistyczną. Naciśnięcie przycisku *Cancel* powinno po prostu usuwać z ekranu okno dialogowe bez wykonywania żadnych dodatkowych operacji.

Makro przedstawione poniżej ilustruje znacznie lepsze podejście do takiego zagadnienia i do zapamiętania wartości wprowadzonej przez użytkownika wykorzystuje zmienną *x*. Jeżeli zmienna zawiera coś innego niż pusty ciąg znaków (czyli kiedy użytkownik wprowadził jakąś wartość), wartość zmiennej jest zapisywana w komórce `A1`. W przeciwnym przypadku procedura kończy działanie, nie wykonując żadnych dodatkowych operacji.

```
Sub GetValue2()
    Dim x as Variant
    x = InputBox("Wprowadź wartość dla komórki A1:")
    If x <> "" Then Range("A1").Value = x
End Sub
```

Zmienna *x* została zdefiniowana jako zmienna typu `Variant`, ponieważ jej wartością może być wartość albo pusty ciąg znaków (jeżeli użytkownik naciśnie przycisk *Cancel*).

Określanie typu zaznaczonego zakresu

Jeżeli zadaniem Twojego makra będzie przetwarzanie zaznaczonego zakresu, takie makro musi mieć zdolność sprawdzenia, czy przed jego wywołaniem użytkownik rzeczywiście zaznaczył zakres komórek. W przeciwnym razie, jeżeli przed wywołaniem makra zaznaczony zostanie inny obiekt (na przykład wykres lub kształt), próba wykonania makra najprawdopodobniej zakończy się niepowodzeniem. Polecenie przedstawione poniżej wykorzystuje funkcję `TypeName` języka VBA do wyświetlania na ekranie typu aktualnie zaznaczonego obiektu.

```
MsgBox TypeName(Selection)
```

Jeżeli aktualnie zaznaczony jest obiekt typu `Range`, wykonanie takiego polecenia spowoduje wyświetlenie słowa *Range*. Jeśli Twoje makro działa tylko z zakresami komórek, możesz użyć polecenia `If` do sprawdzenia, czy aktualnie zaznaczony obiekt to zakres (obiekt typu `Range`). Procedura przedstawiona poniżej sprawdza typ zaznaczonego obiektu i jeżeli nie jest to obiekt typu `Range`, na ekranie wyświetlany jest odpowiedni komunikat i procedura kończy działanie.

```
Sub CheckSelection()
    If TypeName(Selection) <> "Range" Then
        MsgBox "Zaznacz zakres komórek."
        Exit Sub
    End If
'   ... [Tutaj wstaw dalszą część kodu procedury]
End Sub
```

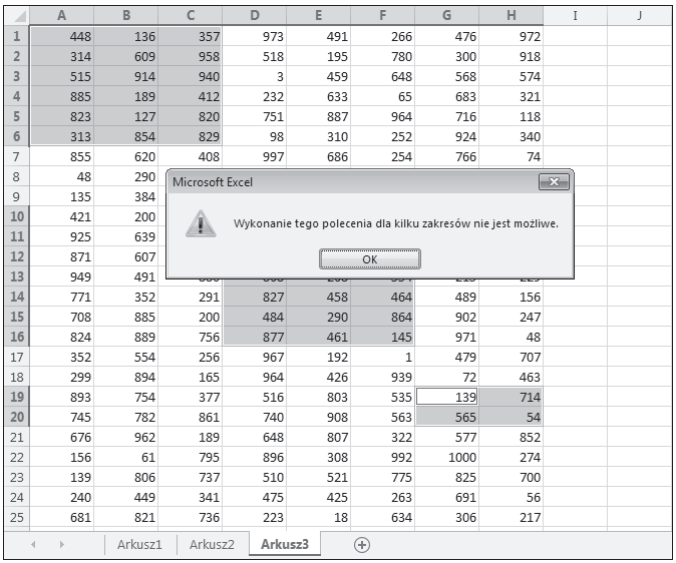
Identyfikowanie zaznaczeń wielokrotnych

Jak pamiętasz, Excel pozwala na jednoczesne zaznaczanie wielu obiektów. Aby to zrobić, powinieneś podczas zaznaczania obiektów lub zakresów trzymać wciśnięty klawisz *Ctrl*. Zaznaczenia wielokrotne mogą być przyczyną problemów z wykonywaniem niektórych makr. Przykładowo nie możesz skopiować zakresu komórek, który został utworzony poprzez wiele zaznaczeń nieciągłych zakresów komórek. Jeżeli spróbujesz wykonać taką operację, Excel wyświetli na ekranie komunikat przedstawiony na rysunku 14.3.

Makro przedstawione niżej pokazuje, w jaki sposób możesz sprawdzić, czy użytkownik dokonał zaznaczenia wielokrotnego, i na tej podstawie wykonać odpowiednią operację.

```
Sub MultipleSelection()
    If Selection.Areas.Count > 1 Then
        MsgBox "Zaznaczenia wielokrotne nie są dozwolone."
        Exit Sub
    End If
'   ... [Tutaj wstaw dalszą część kodu procedury]
End Sub
```

Przedstawiona procedura wykorzystuje metodę `Areas`, która zwraca kolekcję wszystkich zakresów w danym zaznaczeniu. Właściwość `Count` zwraca liczbę obiektów tej kolekcji.



Rysunek 14.3. Excel nie lubi, kiedy próbujesz kopiować nieciągłe zakresy komórek

Zmiana ustawień Excela

Chyba najbardziej użytecznymi makrami są proste procedury, które zmieniają jedno lub kilka ustawień Excela. Jeśli na przykład dojdziesz do wniosku, że bardzo często przywołujesz na ekran okno opcji programu Excel i zmieniasz jakieś ustawienie, taka operacja z pewnością będzie bardzo dobrym kandydatem do napisania makra oszczędzającego Twój czas i ułatwiającego zmianę takiego ustawienia.

W tym podrozdziale zaprezentuję dwa przykłady procedur, które pokazują, w jaki sposób można zmieniać ustawienia Excela. Ogólne zasady działania tych procedur możesz z powodzeniem zastosować do napisania własnych makr zmieniających inne ustawienia Excela.

Zmiana ustawień logicznych (opcje typu Boolean)

Podobnie jak wyłącznik światła, opcje logiczne (typu *Boolean*) mogą być albo włączone, albo wyłączone. Możesz na przykład utworzyć makro, które będzie włączało lub wyłączało wyświetlanie podziału arkusza na strony. Kiedy wydrukujesz arkusz (lub skorzystasz z trybu podglądu wydruku), Excel wyświetla na arkuszu przerywane linie reprezentujące miejsca podziału arkusza na strony wydruku. Niektórych użytkowników (włącznie z autorem tej książki) takie zachowanie Excela irytuje. Niestety, jedynym sposobem wyłączenia wyświetlania podziału arkusza na strony jest otwarcie okna dialogowego *Opcje programu Excel*, przejście na kartę *Zaawansowane,* a następnie przewijanie zawartości tej karty, aż do odszukania opcji *Pokaż podziały stron*. Jeżeli podczas wyłączania tej opcji korzystałeś z rejestratora makr, przekonasz się, że Excel generuje poniższy kod.

```
ActiveSheet.DisplayPageBreaks = False
```

Z drugiej strony, jeżeli podczas rejestrowania makra podziały stron nie są widoczne, Excel generuje taki kod.

```
ActiveSheet.DisplayPageBreaks = True
```

Takie informacje mogą doprowadzić do wniosku, że będziesz musiał napisać aż dwa makra — jedno do włączania podglądu podziału stron, a drugie do jego wyłączania. Na szczęście, to nieprawda. Procedura przedstawiona poniżej wykorzystuje operator `Not` do zmiany wartości logicznej *True* na *False* i odwrotnie. Wykonanie procedury `TogglePageBreaks` to prosty sposób na cykliczne włączanie i wyłączanie podglądu podziału stron arkusza.

```
Sub TogglePageBreaks()
    On Error Resume Next
    ActiveSheet.DisplayPageBreaks = Not _
        ActiveSheet.DisplayPageBreaks
End Sub
```

Pierwsze polecenie informuje Excel, że powinien ignorować ewentualne błędy. Przykładowo podziały stron nie są wyświetlane na arkuszach wykresów. Kiedy wprowadzisz takie polecenie i spróbujesz wykonać tę procedurę dla arkusza wykresu, na ekranie nie pojawi się komunikat o błędzie.

Techniki użytej w procedurze `TogglePageBreaks` możesz używać do zmiany dowolnych opcji logicznych (czyli takich, których wartościami są *True* albo *False*).

Zmiana innych opcji (typu non-Boolean)

Do zmiany opcji, które nie są typu logicznego, możesz używać konstrukcji `Select Case`. W przykładzie przedstawionym poniżej zmieniam tryb przeliczania skoroszytu z ręcznego na automatyczny i odwrotnie, i nakazuję wyświetlenie na ekranie komunikatu opisującego aktualny tryb przeliczania arkusza.

```
Sub ToggleCalcMode()
    Select Case Application.Calculation
        Case xlManual
            Application.Calculation = xlCalculationAutomatic
            MsgBox "Automatyczne przeliczanie skoroszytu"
        Case xlAutomatic
            Application.Calculation = xlCalculationManual
            MsgBox "Ręczne przeliczanie skoroszytu"
    End Select
End Sub
```

Techniki użytej w procedurze `ToggleCalcMode` możesz używać do zmiany dowolnych opcji, które nie posiadają wartości logicznych.

Praca z wykresami

Wykresy w Excelu są wręcz przeładowane najróżniejszymi obiektami, stąd ich przetwarzanie za pomocą kodu VBA może być niezłym wyzwaniem.

Uruchomiłem Excel 2013, w zakresie komórek `A1:A3` wpisałem kilka liczb i zaznaczyłem ten obszar arkusza. Następnie włączyłem rejestrator makr i dla tych trzech punktów danych utworzyłem prosty wykres kolumnowy. Później usunąłem wyświetlanie siatki wykresu i zmieniłem jego tytuł. Oto zarejestrowane makro.

```
Sub Macro1()
'    Zarejestrowane w Excelu 2013
    ActiveSheet.Shapes.AddChart2(201, xlColumnClustered).Select
    ActiveChart.SetSourceData Source:=Range("Arkusz1!$A$1:$A$3")
    ActiveChart.SetElement (msoElementPrimaryValueGridLinesNone)
    ActiveChart.ChartTitle.Select
    ActiveChart.ChartTitle.Text = "To jest mój wykres"
End Sub
```

Kiedy zobaczyłem ten kod, byłem nieco zaskoczony, ponieważ nigdy wcześniej nie słyszałem o metodzie `AddChart2`. Okazało się, że metoda `AddChart2` to nowość, która została wprowadzona w Excelu 2013. Jeżeli wykonasz podobną operację z rejestrowaniem makra w Excelu 2010, wynik będzie następujący.

```
Sub Macro1()
'    Zarejestrowane w Excelu 2010
    ActiveSheet.Shapes.AddChart.Select
    ActiveChart.ChartType = xlColumnClustered
    ActiveChart.SetSourceData Source:=Range("Arkusz1!$A$1:$A$3")
    ActiveChart.Axes(xlValue).MajorGridlines.Select
    Selection.Delete
    ActiveChart.SetElement (msoElementChartTitleAboveChart)
    ActiveChart.ChartTitle.Text = "To jest mój wykres"
End Sub
```

Co to wszystko oznacza? Ano tyle, że makra zarejestrowane w Excelu 2013 po prostu nie będą działały w Excelu 2010, ale makra rejestrowane w Excelu 2010 *działają* w Excelu 2013. Innymi słowy, makra Excela 2010 są kompatybilne w przód (czyli zachowują zgodność z przyszłymi wersjami Excela; *forward compatibility*), a makra Excela 2013 zostały pozbawione kompatybilności wstecznej (*backward compatibility*), czyli nie zachowują zgodności z poprzednimi wersjami.

Przeciętny użytkownik Excela 2013 prawdopodobnie nie wie nic na temat kompatybilności makr w odniesieniu do tworzenia wykresów. Jeżeli jednak udostępnisz takie makro komuś, kto używa starszej wersji Excela, bardzo szybko się o tym dowiesz. Wnioski? Gdy używasz rejestratora makr do tworzenia makr przetwarzających wykresy, powinieneś przetestować takie makra na wszystkich wersjach Excela, które będą wykorzystywane do uruchamiania takiego makra.

Metoda AddChart kontra metoda AddChart2

Poniżej przedstawiam oficjalną składnię metody `AddChart` (metoda jest kompatybilna z Excelem 2007 i wersjami późniejszymi).

```
.AddChart(Type, Left, Top, Width, Height)
```

A oto składnia metody `AddChart2` (która jest kompatybilna wyłącznie z Excelem 2013).

```
.AddChart2 (Style, XlChartType, Left, Top, Width, Height, NewLayout)
```

Jak widać, metoda `AddChart2` pobiera kilka dodatkowych argumentów, które określają styl wykresu, typ wykresu oraz jego układ. Z drugiej strony, metoda `AddChart` tworzy po prostu pusty wykres, a wszystkie detale muszą być zdefiniowane za pomocą dodatkowych poleceń.

Analiza zarejestrowanego kodu ujawnia kilka rzeczy, które mogą być pomocne podczas tworzenia własnych makr przetwarzających wykresy. Jeżeli jesteś ciekawy, rzuć okiem na zmodyfikowaną ręcznie procedurę, której zadaniem jest utworzenie wykresu na bazie zaznaczonego zakresu komórek.

```
Sub CreateAChart()
    Dim ChartData As Range
    Dim ChartShape As Shape
    Dim NewChart As Chart

'   Tworzenie zmiennych obiektowych
    Set ChartData = ActiveWindow.RangeSelection
    Set ChartShape = ActiveSheet.Shapes.AddChart
    Set NewChart = ChartShape.Chart

    With NewChart
        .ChartType = xlColumnClustered
        .SetSourceData Source:=Range(ChartData.Address)
        .SetElement (msoElementLegendRight)
        .SetElement (msoElementChartTitleAboveChart)
        .ChartTitle.Text = "To jest mój wykres"
    End With
End Sub
```

To makro jest kompatybilne z Excelem 2007 i wersjami późniejszymi. Makro tworzy grupowany wykres kolumnowy wraz z legendą i tytułem. Jest to podstawowa wersja makra, która w łatwy sposób może być dostosowana do Twoich indywidualnych wymagań. Jednym ze sposobów może być rejestrowanie makra podczas modyfikowania wykresu i następnie używanie takiego kodu jako wzorca w swoich procedurach.

Swoją drogą, dalej w tym rozdziale omówię konstrukcję `With End-With`, która znakomicie ułatwia pracę z obiektami, oszczędza sporo „stukania w klawiaturę” i znakomicie przyczynia się do zwiększenia przejrzystości kodu.

Jeżeli musisz napisać makro VBA, którego zadaniem będzie przetwarzanie wykresów, musisz zapoznać się z kilkoma ważnymi określeniami. *Wykres osadzony* (*embedded chart*) na arkuszu to obiekt typu `ChartObject`. Obiekt `ChartObject` możesz aktywować podobnie jak aktywujesz arkusz. Polecenie przedstawione poniżej aktywuje obiekt `ChartObject` o nazwie `Wykres 1`.

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

Po aktywowaniu danego wykresu możesz się do niego odwoływać w kodzie VBA za pomocą obiektu `ActiveChart`. Jeżeli wykres znajduje się na osobnym arkuszu wykresu, staje się wykresem aktywnym w chwili, kiedy aktywujesz arkusz wykresu.

Obiekt `ChartObject` jest również obiektem typu `Shape`, co może być nieco mylące. W rzeczywistości, kiedy Twój kod VBA tworzy wykres, cała operacja rozpoczyna się od utworzenia nowego obiektu `Shape` (kształt). Wykres możesz również aktywować poprzez zaznaczenie obiektu `Shape` przechowującego wykres.

```
ActiveSheet.Shapes("Wykres 1").Select
```

W moich programach wolę używać obiektu `ChartObject`, dzięki czemu nie mam żadnych wątpliwości, że pracuję z wykresami.

Kiedy klikasz wykres osadzony lewym przyciskiem myszy, Excel zaznacza obiekt znajdujący się *wewnątrz* obiektu `ChartObject`. Jeżeli chcesz zaznaczyć sam obiekt `ChartObject`, powinieneś klikając wykres, trzymać wciśnięty klawisz *Ctrl*.

Modyfikowanie typu wykresu

A teraz przeczytasz zdanie, które może Cię nieco zdezorientować: obiekty `ChartObject` spełniają rolę kontenerów dla obiektów `Chart`. Jeśli masz jakieś wątpliwości, powinieneś to zdanie kilka razy spokojnie przeczytać i wtedy na pewno wszystko stanie się jasne.

Aby zmodyfikować wykres za pomocą VBA, nie musisz tego wykresu aktywować. Metoda `Chart` może zwracać wykres przechowywany w kontenerze `ChartObject`. Nadal niejasne? Procedury przedstawione poniżej dają taki sam efekt — zmieniają typ wykresu o nazwie *Wykres 1* na wykres powierzchniowy. Pierwsza procedura najpierw aktywuje wykres i następnie pracuje z aktywnym wykresem. Druga procedura nie aktywuje wykresu, a zamiast tego wykorzystuje właściwość `Chart`, która zwraca obiekt `Chart` zawarty w kontenerze `ChartObject`.

```
Sub ModifyChart1()
    ActiveSheet.ChartObjects("Wykres 1").Activate
    ActiveChart.Type = xlArea
End Sub

Sub ModifyChart2()
    ActiveSheet.ChartObjects("Wykres 1").Chart.Type = xlArea
End Sub
```

Przechodzenie w pętli przez elementy kolekcji ChartObjects

Procedura przedstawiona poniżej wprowadza zmiany do wszystkich wykresów osadzonych na aktywnym arkuszu. Procedura wykorzystuje pętlę `For Each-Next` do przechodzenia kolejno przez wszystkie obiekty kolekcji `ChartObjects` i dla każdego obiektu `Chart` zmienia jego właściwość `Type`.

```
Sub ChartType()
    Dim cht As ChartObject
    For Each cht In ActiveSheet.ChartObjects
        cht.Chart.Type = xlArea
    Next cht
End Sub
```

Makro przedstawione poniżej wykonuje taką samą operację, ale na wszystkich arkuszach wykresów w aktywnym skoroszycie.

```
Sub ChartType2()
    Dim cht As Chart
    For Each cht In ActiveWorkbook.Charts
        cht.Type = xlArea
    Next cht
End Sub
```

Modyfikowanie właściwości wykresu

Procedura przedstawiona niżej zmienia czcionkę legendy wykresu dla wszystkich wykresów osadzonych na aktywnym arkuszu. Makro wykorzystuje pętlę `For-Next` do przetwarzania wszystkich obiektów `ChartObject`.

```
Sub LegendMod()
    Dim chtObj As ChartObject
    For Each chtObj In ActiveSheet.ChartObjects
    With chtObj.Chart.Legend.Font
        .Name = "Calibri"
        .FontStyle = "Bold"
        .Size = 12
    End With
    Next chtObj
End Sub
```

Zwróć uwagę na fakt, że obiekt `Font` jest zawarty w obiekcie `Legend`, który jest zawarty w obiekcie `Chart`, który z kolei jest zawarty w kolekcji `ChartObjects`. Czy teraz rozumiesz, dlaczego to wszystko jest nazywane *hierarchią obiektów*?

Zmiana formatowania wykresów

Ten przykład odnosi się do kilku różnych typów formatowania aktywnego wykresu. Utworzyłem to makro, rejestrując moje poczynania podczas formatowania wykresu. Następnie oczyściłem nieco uzyskany kod poprzez usunięcie zbędnych wierszy.

```
Sub ChartMods()
    ActiveChart.Type = xlArea
    ActiveChart.ChartArea.Font.Name = "Calibri"
    ActiveChart.ChartArea.Font.FontStyle = "Regular"
    ActiveChart.ChartArea.Font.Size = 9
    ActiveChart.PlotArea.Interior.ColorIndex = xlNone
    ActiveChart.Axes(xlValue).TickLabels.Font.Bold = True
    ActiveChart.Axes(xlCategory).TickLabels.Font.Bold =
        True
    ActiveChart.Legend.Position = xlBottom
End Sub
```

Przed wykonaniem tego makra musisz aktywować wykres. Wykresy osadzone możesz aktywować poprzez ich kliknięcie lewym przyciskiem myszy. Aby aktywować wykres na arkuszu wykresu, kliknij kartę arkusza.

Aby upewnić się, że wykres jest zaznaczony, możesz w kodzie procedury umieścić polecenie, które będzie sprawdzało, czy wykres jest aktywny. Poniżej znajdziesz kod zmodyfikowanej procedury, która — jeżeli wykres nie jest aktywny — wyświetla na ekranie odpowiedni komunikat i kończy działanie.

```
Sub ChartMods2()
    If ActiveChart Is Nothing Then
        MsgBox "Aktywuj wykres!"
        Exit Sub
    End If
    ActiveChart.Type = xlArea
    ActiveChart.ChartArea.Font.Name = "Calibri"
    ActiveChart.ChartArea.Font.FontStyle = "Regular"
    ActiveChart.ChartArea.Font.Size = 9
    ActiveChart.PlotArea.Interior.ColorIndex = xlNone
    ActiveChart.Axes(xlValue).TickLabels.Font.Bold = True
    ActiveChart.Axes(xlCategory).TickLabels.Font.Bold =
        True
    ActiveChart.Legend.Position = xlBottom
End Sub
```

Poniżej znajdziesz kolejną wersję procedury, która wykorzystuje konstrukcję `With-End With` do zaoszczędzenia „klepania” w klawiaturę i (co ważniejsze) zwiększenia optymalności i przejrzystości kodu. I znowu wyskakujemy nieco przed orkiestrę, ale jeżeli chcesz, możesz już teraz przeskoczyć parę stron do przodu i przeczytać opis polecenia `With-End With`.

```
Sub ChartMods3()
    If ActiveChart Is Nothing Then
        MsgBox "Aktywuj wykres!"
        Exit Sub
    End If
```

```
    With ActiveChart
        .Type = xlArea
        .ChartArea.Font.Name = "Calibri"
        .ChartArea.Font.FontStyle = "Regular"
        .ChartArea.Font.Size = 9
        .PlotArea.Interior.ColorIndex = xlNone
        .Axes(xlValue).TickLabels.Font.Bold = True
        .Axes(xlCategory).TickLabels.Font.Bold = True
        .Legend.Position = xlBottom
    End With
End Sub
```

No cóż… w zakresie zastosowania VBA do przetwarzania wykresów udało Ci się w tym rozdziale jedynie nieco „liznąć" podstawowe elementy tego rozbudowanego zagadnienia. Temat jest niezwykle szeroki, ale mam nadzieję, że to, czego dowiedziałeś się w tym rozdziale, pobudziło Twoją ciekawość i nakierowało poszukiwania we właściwym kierunku.

Jak przyspieszyć działanie kodu VBA?

VBA jest szybki, ale nie zawsze wystarczająco szybki (inna sprawa, że programy komputerowe nigdy nie są wystarczająco szybkie, przynajmniej w opinii większości użytkowników). W tym podrozdziale pokażę kilka trików i sztuczek, które będziesz mógł wykorzystać do przyspieszenia działania swoich makr.

Wyłączanie aktualizacji ekranu

Kiedy uruchomisz makro, możesz wygodnie wyciągnąć się na fotelu i ze spokojem obserwować na ekranie jego postępy. Choć takie postępowanie może być do pewnego czasu ciekawe, to jednak, kiedy makro zostanie już napisane i przetestowane, wyświetlanie bieżących wyników działania może być irytujące i niepotrzebnie zwalniać działanie makra. Na szczęście, Excel pozwala na wyłączenie aktualizacji ekranu na czas działania makra, co może znacząco przyspieszyć jego działanie. Aby wyłączyć aktualizację ekranu, powinieneś użyć polecenia:

```
Application.ScreenUpdating = False
```

Jeżeli chcesz, aby użytkownicy widzieli, co się dzieje na ekranie podczas działania makra, powinieneś włączyć aktualizację ekranu za pomocą polecenia:

```
Application.ScreenUpdating = True
```

Aby zademonstrować różnicę w szybkości działania, powinieneś uruchomić makro przedstawione poniżej, którego zadaniem jest wypełnianie liczbami dużego zakresu komórek.

```
Sub FillRange()
    Dim r as Long, c As Long
    Dim Number as Long
```

```
    Number = 0
    For r = 1 To 50
        For c = 1 To 50
            Number = Number + 1
            Cells(r, c).Select
            Cells(r, c).Value = Number
        Next c
    Next r
End Sub
```

Procedura zaznacza każdą komórkę zakresu i wpisuje do niej kolejną liczbę. Teraz na początku procedury wstaw polecenie przedstawione poniżej i ponownie uruchom procedurę.

```
Application.ScreenUpdating = False
```

Z pewnością zauważyłeś, że zakres został wypełniony znacznie szybciej, a rezultaty nie były widoczne na ekranie, aż do zakończenia działania procedury i automatycznego przywrócenia aktualizacji ekranu.

Kiedy pracujesz nad testowaniem procedury i wyszukiwaniem błędów w kodzie, działanie programu może nagle zostać przerwane, bez automatycznego przywrócenia aktualizacji ekranu (tak, też mi się to zdarza…). W takiej sytuacji okno Excela pozostaje „martwe" i wydaje się, że program nie reaguje na Twoje prośby i groźby. Rozwiązanie tego problemu jest proste — przejdź do okna edytora VBE i w oknie *Immediate* wpisz polecenie:

```
Application.ScreenUpdating = True
```

Wyłączenie automatycznego przeliczania skoroszytu

Załóżmy, że masz skoroszyt zawierający wiele złożonych formuł. Możesz znacząco przyspieszyć działanie makra, jeżeli na czas jego realizacji przełączysz Excel w tryb ręcznego przeliczania skoroszytu. Kiedy makro zakończy działanie, powinieneś ponownie przełączyć Excel w tryb automatycznego przeliczania skoroszytu.

Polecenie przedstawione poniżej przełącza Excel w tryb ręcznego przeliczania skoroszytu.

```
Application.Calculation = xlCalculationManual
```

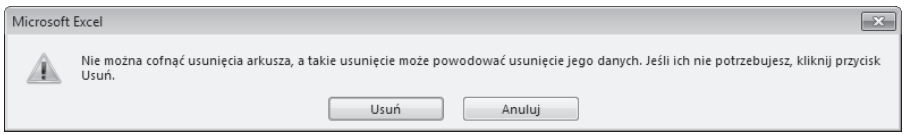
Aby przywrócić tryb automatycznego przeliczania skoroszytu, użyj polecenia:

```
Application.Calculation = xlCalculationAutomatic
```

Jeżeli kod VBA Twojego makra wykorzystuje wyniki działania formuł arkuszowych, pamiętaj, że przełączenie Excela w tryb ręcznego przeliczania arkusza oznacza, iż wartości komórek nie zostaną zaktualizowane, aż do momentu, kiedy jawnie nie „poprosisz" Excela, aby to zrobił.

Wyłączanie irytujących ostrzeżeń

Jak wiesz, makra mogą automatycznie wykonywać całe mnóstwo różnych operacji. W wielu przypadkach możesz po prostu uruchomić makro i spokojnie wybrać się do kuchni i zaparzyć filiżankę swojej ulubionej kawy. Jednak niektóre operacje wykonywane przez Excel mogą spowodować wyświetlenie na ekranie komunikatu, którego potwierdzenie wymaga interakcji ze strony użytkownika. Jeśli na przykład Twoje makro próbuje usunąć arkusz, na którym znajdują się niepuste komórki, działanie makra zostanie automatycznie zatrzymane, na ekranie pojawi się komunikat przedstawiony na rysunku 14.4 i Excel będzie oczekiwał na Twoją reakcję. Obecność tego typu komunikatów oznacza, że nie możesz pozostawić Excela bez nadzoru na czas działania makra… dopóki nie poznasz pewnego triku.



Rysunek 14.4. Możesz nakazać Excelowi zawieszenie wyświetlania takich komunikatów podczas działania makra

Oto cała sztuczka: aby uniknąć wyświetlania takich komunikatów z ostrzeżeniami, w kodzie procedury VBA umieść polecenie:

```
Application.DisplayAlerts = False
```

Excel wykonuje domyślne operacje dla tego typu komunikatów. Podczas usuwania arkusza domyślną operacją jest `Delete` (co właśnie przed chwilą zobaczyłeś). Jeżeli nie jesteś pewien, jaka operacja jest domyślna, przeprowadź test i przekonaj się sam.

Kiedy procedura kończy działanie, Excel automatycznie nada właściwości `DisplayAlerts` wartość *True* (czyli przywróci jej normalny stan). Jeżeli chcesz przywrócić wyświetlanie komunikatów przed zakończeniem działania procedury, powinieneś użyć w kodzie polecenia:

```
Application.DisplayAlerts = True
```

Upraszczanie odwołań do obiektów

Jak już sam zdążyłeś się zorientować, odwołania do obiektów mogą być bardzo rozbudowane. Przykładowo pełne, kwalifikowane odwołanie do obiektu `Range` może wyglądać następująco.

```
Workbooks("MójSkoroszyt.xlsx").Worksheets("Arkusz1") _
    .Range("StawkaProwizji")
```

Jeżeli Twoje makro często korzysta z takiego zakresu, powinieneś rozważyć utworzenie zmiennej obiektowej za pomocą polecenia `Set`. Przykładowo polecenie przedstawione poniżej przypisuje obiekt `Range` do zmiennej obiektowej o nazwie `Rate`.

```
Set Rate = Workbooks("MójSkoroszyt.xlsx").Worksheets("Arkusz1") _
    .Range("StawkaProwizji")
```

Po zdefiniowaniu zmiennej obiektowej możesz zamiast długiego odwołania używać nowo utworzonej zmiennej obiektowej. Aby na przykład zmienić wartość komórki o nazwie `StawkaProwizji`, możesz użyć polecenia:

```
Rate.Value = .085
```

Jak widać, jest to znacznie łatwiejsze do wpisania (i zrozumienia) niż to samo polecenie w pełnej postaci.

```
Workbooks("MójSkoroszyt.xlsx").Worksheets("Arkusz1") _
    .Range("StawkaProwizji").Value = .085
```

Oprócz upraszczania kodu, zastosowanie zmiennych obiektowych powoduje również znaczne zwiększenie szybkości działania kodu Twojego makra. Wiele razy widziałem już makra, które po utworzeniu zmiennych obiektowych zwiększyły szybkość działania nawet dwukrotnie.

Deklarowanie typów zmiennych

Zazwyczaj nie musisz się martwić o typ danych, który przypisujesz do zmiennej. Excel potrafi się tym doskonale zająć. Jeżeli masz zmienną o nazwie *MyVar*, możesz do niej przypisać dowolną liczbę, a później, w dalszej części procedury, możesz do tej samej zmiennej przypisać na przykład ciąg tekstu.

Jeżeli chcesz, aby Twoje procedury VBA działały tak szybko, jak to tylko możliwe (i aby przy okazji uniknąć kilku potencjalnych i naprawdę paskudnych problemów), powinieneś zawsze poinformować Excel o tym, jakie typy danych będą przypisywane do poszczególnych zmiennych. Takie postępowanie jest nazywane *deklarowaniem typów zmiennych* (więcej szczegółowych informacji na ten temat znajdziesz w rozdziale 7.). Powinieneś jak najszybciej wyrobić sobie nawyk deklarowania wszystkich zmiennych, których używasz w swoich programach.

Ogólnie rzecz biorąc, powinieneś zawsze używać takich typów danych, które wystarczą do obsługi Twoich danych przy wykorzystaniu jak najmniejszej liczby bajtów pamięci. Kiedy VBA przetwarza dane, szybkość działania programu zależy od liczby bajtów, jakie VBA ma do „przerobienia". Innymi słowy, im mniej bajtów zajmują dane, tym szybciej VBA może je przetwarzać. Wyjątkiem od tej reguły są dane typu `Integer` — jeżeli szybkość działania programu jest czynnikiem krytycznym, powinieneś zawsze stosować dane typu `Long`.

Jeżeli używasz zmiennych obiektowych (takich jakie opisywałem w poprzednim podrozdziale), możesz zadeklarować taką zmienną jako zmienną określonego typu obiektowego. A oto przykład takiej deklaracji.

```
Dim Rate as Range
Set Rate = Workbooks("MójSkoroszyt.xlsx").Worksheets("Arkusz1") _
    .Range("StawkaProwizji")
```

Zastosowanie struktury With-End With

Czy chcesz ustawić szereg właściwości wybranego obiektu? Twój kod będzie działał znacznie szybciej, gdy użyjesz struktury `With-End With`. Dodatkową zaletą zastosowania tej struktury jest znaczne zwiększenie czytelności kodu.

Fragment kodu przedstawiony poniżej nie wykorzystuje struktury `With-End With`.

```
Selection.HorizontalAlignment = xlCenter
Selection.VerticalAlignment = xlCenter
Selection.WrapText = True
Selection.Orientation = 0
Selection.ShrinkToFit = False
Selection.MergeCells = False
```

A teraz ten sam fragment kodu, ale zapisany z użyciem struktury `With-End With`.

```
With Selection
    .HorizontalAlignment = xlCenter
    .VerticalAlignment = xlCenter
    .WrapText = True
    .Orientation = 0
    .ShrinkToFit = False
    .MergeCells = False
End With
```

Jeżeli przedstawiona struktura wydaje Ci się znajoma, jest tak prawdopodobnie dlatego, że rejestrator makr używa struktury `With-End With` w każdej sytuacji, w której jest to możliwe, a poza tym struktura taka pojawiła się już wcześniej w kilku przykładach w tym rozdziale.

Skorowidz

E

F

G

H

I

J

K

L

Ł

M

N

O

P

R

S

T

U

V

W

Z

Nie takie programowanie straszne...

Większość użytkowników Excela nie zawraca sobie głowy programowaniem w VBA. Twoje zainteresowanie tym tematem zdecydowanie plasuje Cię w elitarnej grupie. Nawet jeśli nie masz najmniejszego pojęcia, o co chodzi w programowaniu, z tym podręcznikiem w mgnieniu oka poszerzysz możliwości najpopularniejszego na świecie arkusza kalkulacyjnego. Ta niezawodna książka jest napisana prostym językiem i zawiera mnóstwo konkretnych informacji. Dzięki nim zwiększysz swoją konkurencyjność na rynku, poprawisz produktywność i odkryjesz, że praca z Excelem może być ekscytującą przygodą. Spraw, by Excel zatańczył, jak mu zagrasz.

- ***VBA w akcji* — poznaj język VBA i etapy tworzenia programu w Excelu.**
- ***Magiczne makra* — automatyzuj częste zadania, personalizuj przyciski, polecenia i funkcje arkusza kalkulacyjnego za pomocą makr VBA.**
- ***Podstawy programowania* — wykonuj operacje na komórkach, kontroluj pracę programu, automatyzuj procedury oraz zdarzenia i eliminuj błędy w kodzie.**
- ***Komunikacja z użytkownikiem* — odkryj triki i techniki prostego personalizowania okien dialogowych i formularzy użytkownika (UserForm).**
- ***Użyj mocy* — wykorzystaj swoje umiejętności, by projektować programy przyjazne dla użytkowników, tworzyć dodatki i nawiązywać interakcje z innymi aplikacjami pakietu Office.**

W książce znajdziesz:

- kilka wartych zapamiętania faktów z historii Excela
- wszystko na temat procedur Sub i Function języka VBA
- informację, jak zdobyć przewagę dzięki nauce VBA
- różnice pomiędzy błędami programowania a standardowymi błędami
- wskazówki, jak personalizować interfejs użytkownika w Excelu
- techniki wyszukiwania i usuwania błędów
- sposoby na zwiększenie możliwości formuł za pomocą własnych funkcji arkusza kalkulacyjnego
- zakazy i nakazy języka VBA

John Walkenbach jest znany wśród swoich fanów jako Mr Spreadsheet. Napisał ponad 60 książek i 300 artykułów poświęconych informatyce, w tym przede wszystkim Excelowi. Stworzył wielokrotnie nagradzany Power Utility Pak oraz kilka innych dodatków do Excela o wszechstronnym zastosowaniu. Walkenbach prowadzi własną stronę internetową pod adresem *http://spreadsheetpage.com*.

Cena 49,00 zł

ISBN 978-83-246-7950-8

9 788324 679508