

# 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



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

Tłumaczenie: Ryszard Górniewicz, Grzegorz Kowalczyk

ISBN: 978-83-246-7950-8

Original English language edition Copyright © 2013 by John Wiley & Sons, Inc., Hoboken, New Jersey. All rights reserved including the right of reproduction in whole or in part any form. This translation published by arrangement with Wiley Publishing, Inc.

Oryginalne angielskie wydanie © 2013 by John Wiley & Sons, Inc., Hoboken, New Jersey. Wszelkie prawa, włączając prawo do reprodukcji całości lub części w jakiegokolwiek formie, zarezerwowane. Tłumaczenie opublikowane na mocy porozumienia z Wiley Publishing, Inc.

Translation copyright © 2014 by Helion S.A.

Wiley, the Wiley logo, For Dummies, the Dummies Man logo, A Reference for the Rest of Us!, The Dummies Way, Dummies Daily, The Fun and Easy Way, Dummies.com, Making Everything Easier, and related trade dress are trademarks or registered trademarks of John Wiley and Sons, Inc. and/or its affiliates in the United States and/or other countries. Used under License.

Wiley, the Wiley logo, For Dummies, the Dummies Man logo, A Reference for the Rest of Us!, The Dummies Way, Dummies Daily, The Fun and Easy Way, Dummies.com, Making Everything Easier, i związana z tym szata graficzna są markami handlowymi John Wiley and Sons, Inc. i/lub firm stowarzyszonych w Stanach Zjednoczonych i/lub innych krajach. Wykorzystywane na podstawie licencji.

Polish language edition published by Wydawnictwo Helion.  
Copyright © 2014.

All rights reserved. No part of this book may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording or by any information storage retrieval system, without permission from the Publisher.

Wszelkie prawa zastrzeżone. Nieautoryzowane rozpowszechnianie całości lub fragmentu niniejszej publikacji w jakiegokolwiek postaci jest zabronione. Wykonywanie kopii metodą kserograficzną, fotograficzną, a także kopiowanie książki na nośniku filmowym, magnetycznym lub innym powoduje naruszenie praw autorskich niniejszej publikacji.

Autor oraz Wydawnictwo HELION dołożyli wszelkich starań, by zawarte w tej książce informacje były kompletne i rzetelne. Nie biorą jednak żadnej odpowiedzialności ani za ich wykorzystanie, ani za związane z tym ewentualne naruszenie praw patentowych lub autorskich. Autor oraz Wydawnictwo HELION nie ponoszą również żadnej odpowiedzialności za ewentualne szkody wynikłe z wykorzystania informacji zawartych w książce.

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](mailto: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.

- [Kup książkę](#)
- [Poleć książkę](#)
- [Oceń książkę](#)

- [Księgarnia internetowa](#)
- [Lubię to! » Nasza społeczność](#)

# Spis treści

|                                                             |           |
|-------------------------------------------------------------|-----------|
| <b>0 autorze .....</b>                                      | <b>15</b> |
| <b>Podziękowania autora .....</b>                           | <b>17</b> |
| <b>Wstęp .....</b>                                          | <b>19</b> |
| Czy ta książka jest dla Ciebie? .....                       | 19        |
| A więc chcesz być programistą... ..                         | 20        |
| Dlaczego warto? .....                                       | 20        |
| Co powinienś wiedzieć? .....                                | 21        |
| Obowiązkowy podrozdział o konwencjach typograficznych ..... | 22        |
| Sprawdź ustawienia zabezpieczeń .....                       | 22        |
| Jak podzielona jest książka? .....                          | 24        |
| Część I: Wstęp do programowania w VBA .....                 | 24        |
| Część II: Jak VBA współpracuje z Excelem? .....             | 24        |
| Część III: Podstawy programowania .....                     | 24        |
| Część IV: Komunikacja z użytkownikiem .....                 | 24        |
| Część V: Od teorii do praktyki .....                        | 24        |
| Część VI: Dekalogi .....                                    | 24        |
| Ikony używane w książce .....                               | 25        |
| Pobieranie plików z przykładami .....                       | 25        |
| Co dalej? .....                                             | 26        |
| <br><b>Część I: Wstęp do programowania w VBA .....</b>      | <b>27</b> |
| <b>Rozdział 1: Czym jest VBA? .....</b>                     | <b>29</b> |
| No dobrze, czym jest więc VBA? .....                        | 29        |
| Co można zrobić za pomocą VBA? .....                        | 30        |
| Wprowadzanie bloków tekstu .....                            | 31        |
| Automatyzacja często wykonywanego zadania .....             | 31        |
| Automatyzacja powtarzalnych operacji .....                  | 31        |
| Tworzenie własnego polecenia .....                          | 31        |
| Tworzenie własnego przycisku .....                          | 31        |
| Tworzenie własnych funkcji arkusza kalkulacyjnego .....     | 31        |
| Tworzenie własnych dodatków do Excela .....                 | 32        |
| Tworzenie kompletnych aplikacji opartych na makrach .....   | 32        |

|                                    |    |
|------------------------------------|----|
| Plusy i minusy języka VBA .....    | 32 |
| Plusy języka VBA .....             | 32 |
| Minusy języka VBA .....            | 33 |
| VBA w pigułce .....                | 33 |
| Wycieczka po wersjach Excela ..... | 35 |

### **Rozdział 2: Szybkie zanurzenie ..... 39**

|                                                   |    |
|---------------------------------------------------|----|
| Przygotowanie do pracy .....                      | 39 |
| Plan działania .....                              | 40 |
| Stawiamy pierwsze kroki .....                     | 40 |
| Rejestrowanie makra .....                         | 41 |
| Testowanie makra .....                            | 42 |
| Podgląd kodu makra .....                          | 42 |
| Modyfikacja makra .....                           | 44 |
| Zapisywanie skrósztytów zawierających makra ..... | 45 |
| Bezpieczeństwo makr .....                         | 45 |
| Więcej o makrze NameAndTime .....                 | 47 |

### **Część II: Jak VBA współpracuje z Excelem? ..... 49**

### **Rozdział 3: Praca w edytorze VBE ..... 51**

|                                                |    |
|------------------------------------------------|----|
| Czym jest Visual Basic Editor? .....           | 51 |
| Uruchamianie edytora VBE .....                 | 51 |
| Zapoznanie z komponentami edytora VBE .....    | 52 |
| Praca z oknem Project .....                    | 54 |
| Dodawanie nowego modułu VBA .....              | 55 |
| Usuwanie modułu VBA .....                      | 55 |
| Eksportowanie i importowanie obiektów .....    | 56 |
| Praca z oknem Code .....                       | 56 |
| Minimalizowanie i maksymalizowanie okien ..... | 56 |
| Tworzenie modułu .....                         | 57 |
| Wprowadzanie kodu VBA do modułu .....          | 58 |
| Bezpośrednie wprowadzanie kodu .....           | 58 |
| Używanie rejestratora makr .....               | 61 |
| Kopiowanie kodu VBA .....                      | 63 |
| Dostosowywanie środowiska VBA .....            | 63 |
| Karta Editor .....                             | 64 |
| Karta Editor Format .....                      | 66 |
| Karta General .....                            | 67 |
| Karta Docking .....                            | 68 |

### **Rozdział 4: Wprowadzenie do modelu obiektowego w Excelu ..... 69**

|                                         |    |
|-----------------------------------------|----|
| Czy Excel to obiekt? .....              | 70 |
| Wspinaczka po hierarchii obiektów ..... | 70 |
| Zapoznanie z kolekcjami .....           | 71 |
| Odwoływanie się do obiektów .....       | 71 |
| Nawigacja po hierarchii obiektów .....  | 72 |
| Upraszczanie odwołań do obiektów .....  | 73 |

|                                              |    |
|----------------------------------------------|----|
| Właściwości i metody obiektów .....          | 74 |
| Właściwości obiektów .....                   | 74 |
| Metody obiektów .....                        | 76 |
| Zdarzenia obiektów .....                     | 77 |
| Poszukiwanie dodatkowych informacji .....    | 78 |
| System pomocy VBA .....                      | 78 |
| Narzędzie Object Browser .....               | 79 |
| Automatyczna lista właściwości i metod ..... | 80 |

## **Rozdział 5: Procedury Sub i Function w języku VBA ..... 81**

|                                                                |    |
|----------------------------------------------------------------|----|
| Procedury Sub a funkcje .....                                  | 81 |
| Rzut oka na procedury Sub .....                                | 82 |
| Rzut oka na procedury Function .....                           | 82 |
| Nazwy procedur Sub i Function .....                            | 83 |
| Uruchamianie procedur Sub .....                                | 83 |
| Bezpośrednie uruchamianie procedur Sub .....                   | 85 |
| Uruchamianie procedur w oknie dialogowym Makro .....           | 85 |
| Uruchamianie makr za pomocą skrótów klawiszowych .....         | 86 |
| Uruchamianie procedur przy użyciu przycisków i kształtów ..... | 87 |
| Uruchamianie procedur z poziomu innych procedur .....          | 89 |
| Uruchamianie procedur Function .....                           | 89 |
| Wywoływanie funkcji z poziomu procedur Sub .....               | 90 |
| Wywoływanie funkcji z poziomu formuły arkusza .....            | 90 |

## **Rozdział 6: Używanie rejestratora makr ..... 93**

|                                                         |     |
|---------------------------------------------------------|-----|
| Czy to rzeczywistość, czy to VBA? .....                 | 93  |
| Podstawy rejestrowania makr .....                       | 93  |
| Przygotowania do rejestrowania makr .....               | 95  |
| Względne czy bezwzględne? .....                         | 96  |
| Rejestrowanie makr w trybie odwołań bezwzględnych ..... | 96  |
| Rejestrowanie makr w trybie odwołań względnych .....    | 97  |
| Co jest rejestrowane? .....                             | 98  |
| Opcje rejestratora makr .....                           | 100 |
| Nazwa makra .....                                       | 100 |
| Klawisz skrótu .....                                    | 100 |
| Przechowuj makro w .....                                | 101 |
| Opis .....                                              | 101 |
| Czy to coś jest wydajne? .....                          | 101 |

## **Część III: Podstawy programowania ..... 105**

### **Rozdział 7: Kluczowe elementy języka VBA ..... 107**

|                                                       |     |
|-------------------------------------------------------|-----|
| Stosowanie komentarzy w kodzie VBA .....              | 107 |
| Używanie zmiennych, stałych i typów danych .....      | 109 |
| Pojęcie zmiennej .....                                | 109 |
| Czym są typy danych w języku VBA? .....               | 110 |
| Deklarowanie zmiennych i określanie ich zasięgu ..... | 111 |

|                                        |     |
|----------------------------------------|-----|
| Stałe .....                            | 117 |
| Stałe predefiniowane .....             | 118 |
| Łańcuchy znaków .....                  | 118 |
| Daty i godziny .....                   | 119 |
| Instrukcje przypisania .....           | 120 |
| Przykłady instrukcji przypisania ..... | 120 |
| O znaku równości .....                 | 121 |
| Proste operatory .....                 | 121 |
| Praca z tablicami .....                | 123 |
| Deklarowanie tablic .....              | 123 |
| Tablice wielowymiarowe .....           | 124 |
| Tablice dynamiczne .....               | 124 |
| Stosowanie etykiet .....               | 125 |

## Rozdział 8: Praca z obiektami Range ..... 127

|                                               |     |
|-----------------------------------------------|-----|
| Szybka powtórka .....                         | 127 |
| Inne sposoby odwoływania się do zakresu ..... | 129 |
| Właściwość Cells .....                        | 129 |
| Właściwość Offset .....                       | 130 |
| Wybrane właściwości obiektu Range .....       | 131 |
| Właściwość Value .....                        | 131 |
| Właściwość Text .....                         | 132 |
| Właściwość Count .....                        | 133 |
| Właściwości Column i Row .....                | 133 |
| Właściwość Address .....                      | 133 |
| Właściwość HasFormula .....                   | 134 |
| Właściwość Font .....                         | 134 |
| Właściwość Interior .....                     | 136 |
| Właściwości Formula i FormulaLocal .....      | 136 |
| Właściwość NumberFormat .....                 | 137 |
| Wybrane metody obiektu Range .....            | 137 |
| Metoda Select .....                           | 137 |
| Metody Copy i Paste .....                     | 138 |
| Metoda Clear .....                            | 138 |
| Metoda Delete .....                           | 139 |

## Rozdział 9: Praca z funkcjami VBA i arkusza kalkulacyjnego ..... 141

|                                                                        |     |
|------------------------------------------------------------------------|-----|
| Co to jest funkcja? .....                                              | 141 |
| Stosowanie wbudowanych funkcji VBA .....                               | 142 |
| Przykłady funkcji VBA .....                                            | 142 |
| Funkcje VBA, które robią coś więcej niż tylko zwracanie wartości ..... | 144 |
| Odkrywanie funkcji VBA .....                                           | 144 |
| Użycie funkcji arkusza kalkulacyjnego w VBA .....                      | 145 |
| Przykłady funkcji arkusza kalkulacyjnego .....                         | 148 |
| Wprowadzanie funkcji arkusza kalkulacyjnego .....                      | 150 |
| Więcej o użyciu funkcji arkusza kalkulacyjnego .....                   | 151 |
| Użycie własnych funkcji .....                                          | 151 |

**Rozdział 10: Sterowanie przepływem i podejmowanie decyzji ..... 153**

|                                                      |     |
|------------------------------------------------------|-----|
| Zabierz się za przepływ, kolego .....                | 153 |
| Instrukcja Go To .....                               | 154 |
| Decyzje, decyzje .....                               | 155 |
| Struktura If-Then .....                              | 155 |
| Struktura Select Case .....                          | 159 |
| Entliczek, pętliczek — czyli jak używać pętli? ..... | 162 |
| Pętla For-Next .....                                 | 162 |
| Pętla Do-While .....                                 | 167 |
| Pętla Do-Until .....                                 | 168 |
| Użycie pętli For Each-Next z kolekcjami .....        | 168 |

**Rozdział 11: Automatyczne procedury i zdarzenia ..... 171**

|                                                     |     |
|-----------------------------------------------------|-----|
| Przygotowanie do wielkiego zdarzenia .....          | 171 |
| Czy zdarzenia są przydatne? .....                   | 173 |
| Programowanie procedur obsługi zdarzeń .....        | 173 |
| Gdzie jest umieszczony kod VBA? .....               | 174 |
| Tworzenie procedury obsługi zdarzenia .....         | 175 |
| Przykłady wprowadzające .....                       | 176 |
| Zdarzenie Open dla skoroszytu .....                 | 176 |
| Zdarzenie BeforeClose dla skoroszytu .....          | 179 |
| Zdarzenie BeforeSave dla skoroszytu .....           | 179 |
| Przykłady zdarzeń aktywacyjnych .....               | 180 |
| Zdarzenia aktywacji i dezaktywacji arkusza .....    | 180 |
| Zdarzenia aktywacji i dezaktywacji skoroszytu ..... | 181 |
| Zdarzenia aktywacji skoroszytu .....                | 182 |
| Inne zdarzenia dotyczące arkusza .....              | 183 |
| Zdarzenie BeforeDoubleClick .....                   | 183 |
| Zdarzenie BeforeRightClick .....                    | 184 |
| Zdarzenie Change .....                              | 184 |
| Zdarzenia niezwiązane z obiektami .....             | 186 |
| Zdarzenie OnTime .....                              | 186 |
| Zdarzenia naciśnięcia klawisza .....                | 188 |

**Rozdział 12: Techniki obsługi błędów ..... 191**

|                                                        |     |
|--------------------------------------------------------|-----|
| Rodzaje błędów .....                                   | 191 |
| Błędny przykład .....                                  | 192 |
| To makro nie jest idealne .....                        | 192 |
| Makro wciąż nie jest idealne .....                     | 193 |
| Czy teraz makro jest idealne? .....                    | 194 |
| Rezygnacja z ideału .....                              | 195 |
| Inny sposób obsługi błędów .....                       | 195 |
| Korekta procedury EnterSquareRoot .....                | 195 |
| O instrukcji On Error .....                            | 196 |
| Obsługa błędów — szczegółowe informacje .....          | 197 |
| Wznawianie wykonywania kodu po wystąpieniu błędu ..... | 197 |
| Obsługa błędów w pigułce .....                         | 199 |

|                                        |     |
|----------------------------------------|-----|
| Kiedy ignorować błędy? .....           | 199 |
| Rozpoznawanie określonych błędów ..... | 200 |
| Zamierzony błąd .....                  | 201 |

## **Rozdział 13: Dezynsekcja kodu, czyli jak walczyć z pluskwami ..... 203**

|                                                                                      |     |
|--------------------------------------------------------------------------------------|-----|
| Rodzaje pluskiew .....                                                               | 203 |
| Podstawy entomologii, czyli jak zidentyfikować pluskwę .....                         | 205 |
| Metody i techniki walki z pluskwami .....                                            | 205 |
| Przeglądanie kodu VBA .....                                                          | 206 |
| Umieszczanie funkcji MsgBox w kluczowych miejscach kodu .....                        | 206 |
| Umieszczanie polecenia Debug.Print w kluczowych miejscach kodu .....                 | 208 |
| Korzystanie z wbudowanych narzędzi Excela wspomagających odpluskwanie kodu VBA ..... | 209 |
| Kilka słów o debuggerze .....                                                        | 209 |
| Ustawianie punktów przerwań w kodzie programu .....                                  | 209 |
| Zastosowanie okna Watch .....                                                        | 212 |
| Zastosowanie okna Locals .....                                                       | 213 |
| Jak zredukować liczbę błędów w kodzie programu? .....                                | 215 |

## **Rozdział 14: Przykłady i techniki programowania w języku VBA ..... 217**

|                                                                                       |     |
|---------------------------------------------------------------------------------------|-----|
| Przetwarzanie zakresów komórek .....                                                  | 217 |
| Kopiowanie zakresów .....                                                             | 218 |
| Kopiowanie zakresu o zmiennej wielkości .....                                         | 219 |
| Zaznaczanie komórek do końca wiersza lub kolumny .....                                | 220 |
| Zaznaczanie całego wiersza lub całej kolumny .....                                    | 221 |
| Przenoszenie zakresów .....                                                           | 222 |
| Wydajne przetwarzanie komórek zaznaczonego zakresu przy użyciu pętli .....            | 222 |
| Wydajne przetwarzanie komórek zaznaczonego zakresu przy użyciu pętli (część II) ..... | 224 |
| Wprowadzanie wartości do komórki .....                                                | 225 |
| Określanie typu zaznaczonego zakresu .....                                            | 226 |
| Identyfikowanie zaznaczeń wielokrotnych .....                                         | 226 |
| Zmiana ustawień Excela .....                                                          | 227 |
| Zmiana ustawień logicznych (opcje typu Boolean) .....                                 | 227 |
| Zmiana innych opcji (typu non-Boolean) .....                                          | 228 |
| Praca z wykresami .....                                                               | 229 |
| Metoda AddChart kontra metoda AddChart2 .....                                         | 230 |
| Modyfikowanie typu wykresu .....                                                      | 231 |
| Przechodzenie w pętli przez elementy kolekcji ChartObjects .....                      | 232 |
| Modyfikowanie właściwości wykresu .....                                               | 232 |
| Zmiana formatowania wykresów .....                                                    | 233 |
| Jak przyspieszyć działanie kodu VBA? .....                                            | 234 |
| Wyłączanie aktualizacji ekranu .....                                                  | 234 |
| Wyłączenie automatycznego przeliczania skoroszytu .....                               | 235 |
| Wyłączanie irytujących ostrzeżeń .....                                                | 236 |
| Upraszczenie odwołań do obiektów .....                                                | 236 |
| Deklarowanie typów zmiennych .....                                                    | 237 |
| Zastosowanie struktury With-End With .....                                            | 238 |

## **Część IV: Komunikacja z użytkownikiem ..... 239**

### **Rozdział 15: Proste okna dialogowe .....241**

|                                                                    |     |
|--------------------------------------------------------------------|-----|
| Co zamiast formularzy UserForm? .....                              | 241 |
| Funkcja MsgBox .....                                               | 242 |
| Wyświetlanie prostych okien dialogowych .....                      | 243 |
| Pobieranie odpowiedzi z okna dialogowego .....                     | 243 |
| Dostosowywanie wyglądu okien dialogowych do własnych potrzeb ..... | 244 |
| Funkcja InputBox .....                                             | 247 |
| Składnia funkcji InputBox .....                                    | 248 |
| Przykład zastosowania funkcji InputBox .....                       | 248 |
| Inny rodzaj okna dialogowego InputBox .....                        | 249 |
| Metoda GetOpenFilename .....                                       | 250 |
| Składnia metody GetOpenFilename .....                              | 251 |
| Przykład zastosowania metody GetOpenFilename .....                 | 251 |
| Metoda GetSaveAsFilename .....                                     | 253 |
| Pobieranie nazwy folderu .....                                     | 254 |
| Wyświetlanie wbudowanych okien dialogowych programu Excel .....    | 254 |

### **Rozdział 16: Wprowadzenie do formularzy UserForm .....257**

|                                                                     |     |
|---------------------------------------------------------------------|-----|
| Kiedy używać formularzy UserForm? .....                             | 257 |
| Tworzenie formularzy UserForm — wprowadzenie .....                  | 258 |
| Praca z formularzami UserForm .....                                 | 259 |
| Wstawianie nowego formularza UserForm .....                         | 259 |
| Umieszczanie formantów na formularzu UserForm .....                 | 260 |
| Modyfikacja właściwości formantów formularza UserForm .....         | 261 |
| Przeglądanie okna Code formularza UserForm .....                    | 263 |
| Wyświetlanie formularzy UserForm .....                              | 263 |
| Pobieranie i wykorzystywanie informacji z formularzy UserForm ..... | 264 |
| Przykład tworzenia formularza UserForm .....                        | 264 |
| Tworzenie formularza UserForm .....                                 | 265 |
| Dodawanie przycisków poleceń (formanty CommandButton) .....         | 265 |
| Dodawanie przycisków opcji (formanty OptionButton) .....            | 267 |
| Dodawanie procedur obsługi zdarzeń .....                            | 268 |
| Tworzenie makra, które wyświetla formularz na ekranie .....         | 270 |
| Udostępnianie makra użytkownikowi .....                             | 271 |
| Testowanie działania makra .....                                    | 272 |

### **Rozdział 17: Praca z formantami formularza UserForm .....275**

|                                                           |     |
|-----------------------------------------------------------|-----|
| Rozpoczynamy pracę z formantami formularzy UserForm ..... | 275 |
| Dodawanie formantów .....                                 | 276 |
| Wprowadzenie do właściwości formantów .....               | 277 |
| Formanty okien dialogowych — szczegóły .....              | 278 |
| Formant CheckBox (pole wyboru) .....                      | 279 |
| Formant ComboBox (pole kombi) .....                       | 280 |
| Formant CommandButton (przycisk polecenia) .....          | 281 |
| Formant Frame (pole grupy) .....                          | 281 |
| Formant Image (pole obrazu) .....                         | 282 |

|                                                                            |     |
|----------------------------------------------------------------------------|-----|
| Formant Label (pole etykiety) .....                                        | 283 |
| Formant ListBox (pole listy) .....                                         | 283 |
| Formant MultiPage .....                                                    | 284 |
| Formant OptionButton (przycisk opcji) .....                                | 285 |
| Formant RefEdit (pole zakresu) .....                                       | 286 |
| Formant ScrollBar (pasek przewijania) .....                                | 286 |
| Formant SpinButton (pokrętło) .....                                        | 287 |
| Formant TabStrip (pole karty) .....                                        | 288 |
| Formant TextBox (pole tekstowe) .....                                      | 288 |
| Formant ToggleButton (przycisk przełącznika) .....                         | 289 |
| Praca z formantami w oknach dialogowych .....                              | 289 |
| Zmiana rozmiarów i przenoszenie formantów w inne miejsce .....             | 289 |
| Rozmieszczanie i wyrównywanie położenia formantów w oknie dialogowym ..... | 290 |
| Obsługa użytkowników preferujących korzystanie z klawiatury .....          | 291 |
| Testowanie formularzy UserForm .....                                       | 293 |
| Estetyka okien dialogowych .....                                           | 293 |

### **Rozdział 18: Techniki pracy z formularzami UserForm ..... 295**

|                                                                          |     |
|--------------------------------------------------------------------------|-----|
| Zastosowanie własnych okien dialogowych .....                            | 295 |
| Przykładowy formularz UserForm .....                                     | 296 |
| Tworzenie okna dialogowego .....                                         | 296 |
| Tworzenie kodu procedury wyświetlającej okno dialogowe .....             | 298 |
| Udostępnianie makra użytkownikowi .....                                  | 299 |
| Testowanie okna dialogowego .....                                        | 299 |
| Dodawanie procedur obsługi zdarzeń .....                                 | 300 |
| Sprawdzanie poprawności danych .....                                     | 302 |
| Teraz okno dialogowe działa tak, jak powinno! .....                      | 302 |
| Więcej przykładów formularzy UserForm .....                              | 302 |
| Zastosowanie formantów ListBox .....                                     | 303 |
| Zaznaczanie zakresów .....                                               | 307 |
| Praca z wieloma grupami formantów OptionButton .....                     | 309 |
| Zastosowanie formantów SpinButton oraz TextBox .....                     | 310 |
| Wykorzystywanie formularza UserForm jako wskaźnika postępu zadania ..... | 312 |
| Tworzenie niemodalnych okien dialogowych z wieloma kartami .....         | 315 |
| Wyświetlanie wykresów na formularzach UserForm .....                     | 317 |
| Lista kontrolna tworzenia i testowania okien dialogowych .....           | 318 |

### **Rozdział 19: Udostępnianie makr z poziomu interfejsu użytkownika ..... 321**

|                                                        |     |
|--------------------------------------------------------|-----|
| Dostosowywanie Wstążki .....                           | 321 |
| Ręczne dopasowywanie Wstążki do własnych potrzeb ..... | 322 |
| Dodawanie do Wstążki przycisku własnego makra .....    | 324 |
| Dostosowywanie Wstążki za pomocą kodu XML .....        | 324 |
| Dostosowywanie menu podręcznego .....                  | 329 |
| Rodzaje obiektów CommandBar .....                      | 329 |
| Wyświetlanie wszystkich menu podręcznych .....         | 329 |
| Odwołania do elementów kolekcji CommandBars .....      | 330 |
| Odwołania do formantów obiektu CommandBar .....        | 331 |
| Właściwości formantów obiektu CommandBar .....         | 332 |

|                                                                  |     |
|------------------------------------------------------------------|-----|
| Przykłady zastosowania VBA do modyfikacji menu podręcznego ..... | 334 |
| Resetowanie wszystkich wbudowanych menu podręcznych .....        | 334 |
| Dodawanie nowego elementu do menu podręcznego Cell .....         | 335 |
| Wyłączanie menu podręcznego .....                                | 337 |
| Tworzenie własnych pasków narzędzi .....                         | 338 |

## **Część V: Od teorii do praktyki ..... 341**

### **Rozdział 20: Jak tworzyć własne funkcje arkuszowe i jak przeżyć, aby o tym opowiedzieć? .....343**

|                                                     |     |
|-----------------------------------------------------|-----|
| Dlaczego tworzymy własne funkcje? .....             | 343 |
| Podstawowe informacje o funkcjach VBA .....         | 344 |
| Tworzenie funkcji .....                             | 345 |
| Praca z argumentami funkcji .....                   | 345 |
| Przykłady funkcji .....                             | 346 |
| Funkcje bezargumentowe .....                        | 346 |
| Funkcje jednoargumentowe .....                      | 346 |
| Funkcje z dwoma argumentami .....                   | 348 |
| Funkcje pobierające zakres jako argument .....      | 349 |
| Funkcje z argumentami opcjonalnymi .....            | 351 |
| Funkcje opakowujące .....                           | 353 |
| Funkcja NumberFormat .....                          | 353 |
| Funkcja ExtractElement .....                        | 354 |
| Funkcja SayIt .....                                 | 354 |
| Funkcja IsLike .....                                | 355 |
| Funkcje zwracające tablice .....                    | 355 |
| Zwracanie tablicy zawierającej nazwy miesięcy ..... | 355 |
| Zwracanie posortowanej listy .....                  | 356 |
| Okno dialogowe Wstawianie funkcji .....             | 358 |
| Wyświetlanie opisów funkcji .....                   | 358 |
| Opisy argumentów .....                              | 360 |

### **Rozdział 21: Tworzenie dodatków .....361**

|                                           |     |
|-------------------------------------------|-----|
| No dobrze... czym zatem są dodatki? ..... | 361 |
| Po co tworzy się dodatki? .....           | 362 |
| Praca z dodatkami .....                   | 363 |
| Podstawy tworzenia dodatków .....         | 364 |
| Tworzymy przykładowy dodatek .....        | 365 |
| Konfiguracja skoroszytu .....             | 365 |
| Testowanie skoroszytu .....               | 367 |
| Tworzenie opisów dodatku .....            | 368 |
| Ochrona kodu VBA .....                    | 369 |
| Tworzenie dodatku .....                   | 369 |
| Otwieranie dodatku .....                  | 369 |
| Dystrybucja dodatków .....                | 370 |
| Modyfikowanie dodatków .....              | 371 |

**Część VI: Dekalogi ..... 373**

**Rozdział 22: Dziesięć pytań na temat VBA (wraz z odpowiedziami) ..... 375**

**Rozdział 23: (Prawie) dziesięć źródeł informacji na temat Excela ..... 379**

|                                           |     |
|-------------------------------------------|-----|
| System pomocy języka VBA .....            | 379 |
| Wsparcie techniczne firmy Microsoft ..... | 380 |
| Inne strony internetowe .....             | 380 |
| Blogi poświęcone Excelowi .....           | 380 |
| Google .....                              | 381 |
| Bing .....                                | 381 |
| Lokalne grupy użytkowników .....          | 381 |
| Moje inne książki .....                   | 381 |

**Rozdział 24: Dziesięć rzeczy, które powinieneś robić w języku VBA  
i których nie powinieneś robić ..... 383**

|                                                                                                |     |
|------------------------------------------------------------------------------------------------|-----|
| Zawsze deklaruj wszystkie zmienne .....                                                        | 383 |
| Nigdy nie powinieneś mylić hasła chroniącego kod VBA z bezpieczeństwem aplikacji .....         | 384 |
| Zawsze staraj się wyczyścić i zoptymalizować kod aplikacji .....                               | 384 |
| Nigdy nie umieszczaj wszystkiego w jednej procedurze .....                                     | 385 |
| Zawsze powinieneś rozważyć zastosowanie innego oprogramowania .....                            | 385 |
| Nigdy nie zakładaj, że każdy użytkownik zezwala na uruchamianie makr .....                     | 386 |
| Zawsze staraj się eksperymentować z nowymi rozwiązaniami .....                                 | 386 |
| Nigdy z góry nie zakładaj, że Twój kod będzie poprawnie działał z innymi wersjami Excela ..... | 386 |
| Zawsze pamiętaj o użytkownikach Twojej aplikacji .....                                         | 387 |
| Nigdy nie zapominaj o tworzeniu kopii zapasowych .....                                         | 387 |

**Skorowidz ..... 389**

## 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.

**W**ierzę, 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 wstawiś 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.

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

|    | 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 |            |              |          |   |   |

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 użyciu 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[#A1:]")
```

## 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  $\downarrow$  (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 `OnError` 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 *Stale* 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.

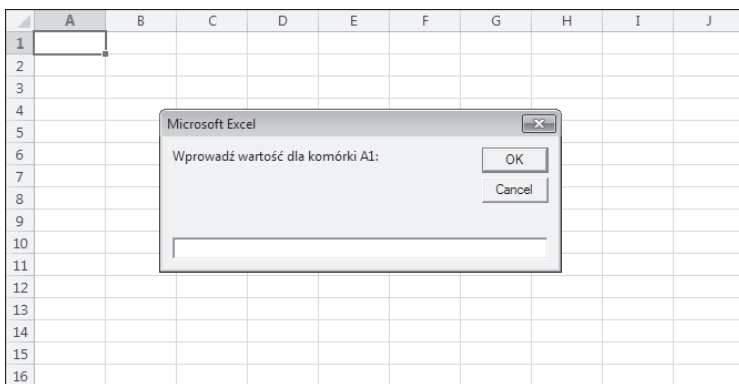
```
Sub SkipBlanks2()
    Dim WorkRange As Range
    Dim cell As Range
    Set WorkRange = Intersect(Selection, ActiveSheet.UsedRange)
    For Each cell In WorkRange
        If cell.Value > 0 Then
            cell.Font.Bold = True
        End If
    Next cell
End Sub
```

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 `Input-`  
`Box` do pobie-  
rania 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

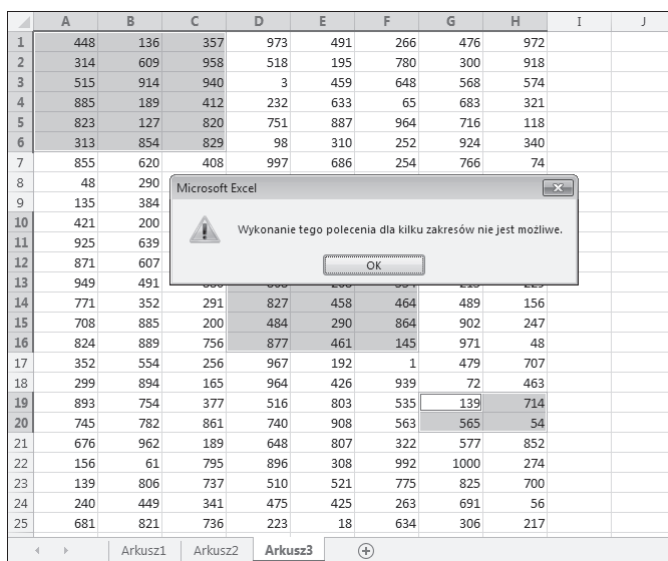
Jak pamiętasz, Excel pozwala na jednoczesne zaznaczanie wielu obiektów. Aby to zrobić, powinieneś podczas zaznaczania obiektów lub zakresów trzymać wciśnięty klawisz *Ctrl*. Zaznaczenia wielokrotne mogą być przyczyną problemów z wykonywaniem niektórych makr. Przykładowo nie możesz skopiować zakresu komórek, 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 nie-  
ciągłe zakresy  
komórek



## Zmiana ustawień Excela

Chyba najbardziej użytecznymi makrami są proste procedury, które zmieniają jedno lub kilka ustawień Excela. Jeśli na przykład dojdiesz 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 wyłączenia. 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 wyłączenie 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 zazaczyłem ten obszar arkusza. Następnie włączyłem rejestrator makr i dla tych trzech punktów danych utworzyłem prosty wykres kolumnowy. Później uśunałem 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 Excelsa 2010 są kompatybilne w przód (czyli zachowują zgodność z przyszłymi wersjami Excelsa; *forward compatibility*), a makra Excelsa 2013 zostały pozbawione kompatybilności wstecznej (*backward compatibility*), czyli nie zachowują zgodności z poprzednimi wersjami.

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

```
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
```

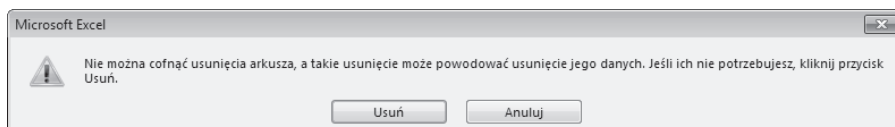


Jeżeli kod VBA Twojego makra wykorzystuje wyniki działania formuł arkuszowych, pamiętaj, że przełączenie Excela w tryb ręcznego przeliczania arkusza oznacza, iż wartości komórek nie zostaną zaktualizowane, aż do momentu, kiedy jawnie nie „poprosisz” Excela, aby to zrobił.

## 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
```

## Upraszczenie 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

---

## A

Add-In, 34  
aplikacja  
    bezpieczeństwo, 384  
    jako niezależny program, 33  
    oparta na makrach, 32  
arkusz, 34  
    aktywacja, 180  
    aktywny, 169  
    dodawanie, 61  
    nazwa, 169  
    okno kodu, 180  
    wykresu, 72  
Auto Data Tips, 65  
Auto Indent, 65  
Auto List Members, 64, 78, 80, 144, 150  
Auto Quick Info, 65  
Auto Syntax Check, 64  
automatyzacja, 30

## B

biblioteka obiektów, Patrz: obiekt biblioteka  
Bieżący obszar, 220  
blok  
    instrukcji jako komentarz, 108  
    tekstu, 31  
błąd, 76, 146  
    #WARTOŚĆ!, 352  
    czasu wykonania, 116  
    eliminowanie, 215  
    graniczny, 204  
    ignorowanie, 197, 199  
    kod, 144, 146  
    kompilacji, 60, 110  
    komunikat, 146, Patrz: komunikat o błędzie  
    numer, 200  
    obsługa, 192, 195, 196, 197, 199

poprawianie, Patrz: odpluskwianie  
programowania, 191, 203, 204, 205  
    kontekstem operacji, 204  
    logiczny, 204  
    typ danych, 204  
    warunki graniczne, 204  
    wersja programu, 204  
przechwytywanie, 195  
rozpoznawanie, 200  
składni, 64, 136, 204  
Subscript out of range, 376  
wykonania, 191, 196  
zakresu, 127  
zamierzony, 201  
breakpoint, Patrz: punkt przzerwania

## C

Chart, 34, 70, 71, 231, 232  
Charts, 72  
ciąg znaków, 142, 147, 249  
collection, Patrz: kolekcja  
Comment Block, 108  
Custom UI Editor for Microsoft Office, 325  
czas, 147, 186  
    systemowy, 156  
    zapis liczbowy, 187  
    zegarowy, 119  
czcionka, 134

## D

dane  
    kopiowanie, 185  
    poprawność, 185  
    tekstowe, 118  
    typ, 82, 107, 110, 237  
        błędy, 204  
    Boolean, 111, 134

dane

typ

- Byte, 111
- Currency, 111
- Date, 111, 119
- definiowanie, 95
- domyślny, 111
- Double, 111
- Integer, 111
- Long, 111
- Object, 111
- predefiniowany, 111
- przydział dynamiczny, 110
- Single, 111
- String, 111
- Variant, 111, 132, 134

wklejanie, 185

data, 119, 146, 147

- część, 146
- format, 120
- krótka, 120
- obliczenia, 146
- systemowa, 146
- zamiana na liczbę seryjną, 146

debugger, 216

debugowanie, Patrz: odpluskwanie

Default to Full Module View, 66

deklaracja, 58

Deweloper, 39

dodatek, 34, 48, 54, 361, 362, 363

- Analysis ToolPak, 208, 361, 363
- bezpieczeństwo, 369
- modyfikowanie, 371
- obiekt UserForm, 362
- opis, 368
- otwieranie, 363, 369
- Power Utility Pak, 361
- Solver, 361
- tworzenie, 32, 364, 365, 369
- udostępnianie, 370

Dostosowywanie Wstążki, 256

Drag-and-Drop Text Editing, 66

drzewo, 53, 54

## **E**

edytor VBE, Patrz: VBE

ekran

- aktualizacja, 235, 377
- wyłączenie aktualizacji, 234

element graficzny, 261

embedded chart, Patrz: wykres osadzony

etykieta, 125, 154

Excel

- ustawienia, 227, 228
- wersja, 35, 229, 387

## **F**

folder

- nazwa, 146
- zaufany, 23

formant, 259, 260, 275

ActiveX, 88

CheckBox, 261, 279

ComboBox, 261, 280, 303

CommandButton, 261, 265, 281

dodawanie, 276, 277

etykieta, 278

Frame, 261, 281, 292

grafika, 278

Image, 261, 282

jakoo kontener, 292

klawisz skrótu, 292

kolejność tabulacji, 291

Label, 261, 283, 287, 310

ListBox, 261, 283, 284, 303

MultiPage, 261, 284, 292, 294, 315, 316

nawigacja za pomocą klawiatury, 291

nazwa, 264, 278

obiektu CommandBar, 331, 332

OptionButton, 261, 267, 276, 280, 285, 309

pozycja w oknie, 278, 289, 290

RefEdit, 261, 286

rozmiar, 278, 290

ScrollBar, 261, 286

SpinButton, 261, 287, 310

TabStrip, 261, 288

TextBox, 261, 288, 296, 310

tło, 278

ToggleButton, 261, 289

wartość, 278

widoczność, 278

właściwość, 261, 262, 277, 278

Accelerator, 278, 279, 285

AutoSize, 278, 288

BackColor, 278

BackStyle, 278

BeginGroup, 332

- BorderStyle, 282
- BuiltIn, 332
- Cancel, 281
- Caption, 278, 282, 332
- ControlSource, 280, 284, 285, 287, 288
- Default, 281
- Enabled, 332
- FaceID, 332
- GroupName, 285
- Height, 278
- IntegralHeight, 284, 288
- LargeChange, 287
- Left, 278
- ListCount, 303
- ListIndex, 303, 305
- ListRows, 280
- ListStyle, 280, 284
- Max, 287
- MaxLength, 288
- metody, 303
- Min, 287
- MultiLine, 288
- MultiSelect, 284, 303, 306
- Name, 278
- OnAction, 332
- Picture, 278, 282
- PictureSizeMode, 282
- RowSource, 280, 284
- ScrollBars, 289
- Selected, 303
- SmallChange, 287
- Style, 280, 285
- TextAlign, 278, 289
- ToolTipText, 333
- Top, 278
- Value, 278, 280, 284, 285, 287, 303
- Visible, 278, 332
- Width, 278
- WordWrap, 289
- zmiana, 277
- zaznaczanie, 290
- format
  - XLA, 362
  - XLAM, 362
  - xlsm, 45, 376
  - XLSM, 362
  - xlsx, 45, 376
- formularz UserForm, Patrz: UserForm
- formuła, 134
  - nazwa, 44
  - odpowiednik angielski, 44
  - tablicowa, 355
- funkcja, 33, 58, 59, 81, 141, 344,
  - Patrz też: metoda
  - Abs, 146
  - argument, 82
  - argumenty, 345, 349
    - opcjonalne, 351
    - opis, 360
  - arkuszowa, 343, 358, 375
    - ograniczenia, 344
  - Array, 146
  - bezargumentowa, 346
  - Choose, 146
  - Chr, 146
  - CurDir, 146
  - Date, 142, 146
  - DateAdd, 146
  - DateDiff, 146
  - DatePart, 146
  - DateSerial, 146
  - DateValue, 146
  - Day, 146
  - Dir, 146
  - DŁ, 142
  - dwuargumentowa, 348
  - Err, 146
  - Error, 146
  - Exp, 146
  - FileLen, 143, 146
  - Fix, 146
  - Format, 146
  - GetOpenFilename, 250
  - GetSetting, 146
  - Hour, 146
  - InputBox, 144, 146, 149, 154, 195, 225, 242,
    - 247, 248, 249
    - argumenty, 248
    - pobranie liczby, 249
  - InStr, 146
  - InStrRev, 146
  - Int, 146
  - IsArray, 146
  - IsDate, 146
  - IsEmpty, 146
  - IsError, 146
  - IsMissing, 146

## funkcja

- IsNull, 147
- IsNumeric, 147, 193
- jednoargumentowa, 346
- LARGE, 148
- LBound, 147
- LCase, 147, 270
- Left, 147
- Len, 142, 147
- MAX, 148
- Mid, 147
- MIN, 148
- Minute, 147
- MOD, 122, 151
- Month, 147
- MonthName, 143
- MsgBox, 74, 142, 144, 147, 149, 206, 242, 247
  - argumenty, 242
  - przyciski, 245
- nazwa, 83, 112
- Now, 142, 147
- opakowująca, 353
- opis, 358
- pasywna, 344
- PMT, 148
- Proper, 270
- Replace, 147
- RGB, 135, 147
- Right, 147
- Rnd, 147
- Second, 147
- Shell, 144, 147
- Space, 147
- Split, 147
- Sqr, 147
- StrConv, 270
- String, 147
- Time, 142, 147
- Timer, 147
- TimeSerial, 147
- TimeValue, 147, 187
- Trim, 147
- tworzenie, 31, 345
- TypeName, 144, 147
- UBound, 147
- UCase, 147, 257, 270
- użytkownika, 141, 151
- Val, 147
- VLOOKUP, 149

## wbudowana

- języka VBA, 141, 142, 144
- podpowiedzi, 144
- programu Excel, 141, 145, 150, 151
- Weekday, 147
- własna, Patrz: funkcja użytkownika
- wykrywanie błędów, 352
- WYSZUKAJ.PIONOWO, 149
- wyświetlanie informacji, 65
- wywołanie z procedury Sub, 352
- wywoływanie, 82, 89, 90
- Year, 147
- Z.WIELKIEJ.LITERY, 270

**G**

- generator liczb pseudolosowych, 353
- godzina, 146
- długa, 120
- format, 119
- Graphical User Interface, Patrz: GUI
- GUI, 259

**H**

- hasło, 54, 362, 369, 376, 384

**I**

- identyfikator zadania, 144
- instrukcja
  - ElseIf, 157
  - Exit For, 163
  - Exit Sub, 154
  - GoTo, 125, 153, 154, 155
  - If-Then, Patrz: struktura If-Then
  - On Error, 195, 196, 197
  - On Error GoTo, 197
  - On Error Resume, 197, 198
  - On Error Resume Next, 189, 197, 199, 215
  - Option Explicit, 60, 64, 112
  - przypisania, 120
  - ReDim, 124
  - Resume, 197, 198
  - Resume Next, 197
  - Step, 163
  - warunkowa, 95
- IntelliSense, 64
- interfejs użytkownika graficzny, Patrz: GUI

**J**

## język

- makr, 30
- programowania, 30
- XML, 36
- XML, 325

**K**

## karta

- Deweloper, 363, Patrz: Deweloper
- DODATKI, 329
- Plik, 363
- zawierająca formanty, 261

## klawisz Esc, 219

## kod

- ANSI, 146
- spaghetti, 155
- kolekcja, 34, 71
- Addins, 361
- ChartObject, 232
- CommandBars, 330
- element, 71
- metoda, 77
- przeglądanie, 168
- Sheets, Patrz: Sheets
- zakresów, 226

## kolor, 135

- motywu, 135
- RGB, 147
- standardowy, 135
- TintAndShade, 135
- vbBlack, 135
- vbBlue, 135
- vbCyan, 135
- vbGreen, 135
- vbMagenta, 135
- vbRed, 135
- vbWhite, 135
- vbYellow, 135
- wypełnienia, 135

## kolumny ukrywanie, 61

## komentarz, 44, 107, 108, 216, 385

## komórka, 73

- adresu wprowadzanie, 261
- format, 185
- niepusta, 223
- pusta, 218

## wartości wprowadzanie, 225

## zaznaczanie, 219, 221

## całego wiersza, 221

## całej kolumny, 221

## do końca kolumny, 220, 377

## do końca wiersza, 220

## komunikat

## o błędzie, 146, 174

## wymagający potwierdzenia, 236

## komunikatem, 147

## kontener, 34

## kontrolka formularza, 87, 88

## kształt, 34, 84, 87, 88

**L**

## liczba

## całkowita, 110

## część całkowita, 146

## e, 146

## formatowanie, 132

## pseudolosowa, 353

## rzeczywista, 110

## wartość bezwzględna, 146

## lista, 261, 303

## element, 304, 306

## rozwijana, 261, 280

## sortowanie, 356

## logarytm naturalny, 146

## lokalizacja zaufana, 23, 46, 47

**Ł**

## łańcuch znaków, 110, 118, 119, 136, 142, 354

## o stałej długości, 119

## o zmiennej długości, 119

## porównywanie, 355

**M**

## makro, 30, 60, 82, Patrz też: procedura Sub,

## program

## bezpieczeństwo, 45, 47

## instrukcje nadmiarowe, 44

## klawisz skrótu, 100

## kod, 42

## lista, 63

## lokalizacja, 101

## modyfikacja, 44

makro  
nazwa, 100  
rejestrator, 58, 61, 82, 93, 95, 98, 375  
ograniczenia, 95  
opcje, 100  
wydajność, 101, 218  
rejestrowanie, 31, 41, 55, 88, 93, 95  
w trybie odwołań bezwzględnych, 96  
w trybie odwołań względnych, 97  
testowanie, 272  
ustawienia, 23, 46  
menu, 331  
podręczne, 329  
Cell, 335  
Excel 2003, 338  
Excel 2013, 336  
modyfikacja, 334  
resetowanie, 334  
wyłączanie, 337  
wyświetlanie, 329  
metoda, 35, 127, Patrz też: funkcja  
Add, 77  
AddChart, 230  
AddChart2, 229, 230  
Areas, 226  
argument, 76  
Cells, 129  
Clear, 138  
ClearContents, 76  
Copy, 138, 219  
Delete, 139  
End, 221  
ExecuteMso, 255  
Export, 318  
FileDialog, 242  
GetOpenFilename, 242, 251  
argumenty, 251  
GetSaveAsFilename, 242, 253  
InputBox, 242, 249  
Intersect, 224  
OnTime, 187, 188  
Paste, 138  
SaveCopyAs, 179  
Select, 137  
SpecialCells, 223, 224, 273  
miesiąc, 143, 147, 355  
model obiektowy, 34, 69, 110

moduł, 54, 154  
Code, 263  
dodawanie, 55, 94  
limit znaków, 58  
przewijanie w oknie, 66  
sekcja Declarations, 115  
tworzenie, 57, 58  
usuwanie, 55

## N

narzędzie Object Browser, Patrz: Object Browser

## O

obiekt, 34, 69  
ActiveChart, 231  
Addin, 70  
Add-In, Patrz: Add-In  
Application, 34, 70, 72  
biblioteka, 79  
Chart, Patrz: Chart  
ChartObject, 231  
CommandBar, 329, 330, 338  
formanty, 331, 332  
CommandBars, 255  
Comment, 70  
eksportowanie, 56  
Err, 200  
FileDialog, 254  
hierarchia, 34, 69  
Hyperlink, 70  
importowanie, 56  
kontener, Patrz: kontener  
metoda, 74, 76, 80, Patrz: metoda  
Name, 70  
numer indeksu, 72  
odwołanie, Patrz: odwołanie  
okno Code, 53  
PageSetup, 70  
PivotTable, 70, Patrz: PivotTable  
Range, 70, Patrz: Range  
Shape, 231  
Ten\_skoroszyt, 54  
UserForm, 259  
VBProject, 70  
Window, 70  
właściwość, Patrz: właściwość  
Workbook, Patrz: Workbook

- Worksheet, Patrz: Worksheet
  - WorksheetFunction, 70, 145
  - wskazywanie, 71
  - zakresu, Patrz: Range
  - zdarzenie, Patrz: zdarzenie
  - Object Browser, 78, 79
  - object-oriented programming, Patrz:
    - programowanie zorientowane obiektowo
  - obsługa techniczna, 33
  - odpluskwanie, 33, 54, 204, 205, 208, 273
    - metody, 205
    - narzędzia, 209
  - odwołanie
    - bezwzględne, 94, 96, 133
    - do obiektu, 236
    - do zakresu, 129, 130
    - jednoznaczne, Patrz: odwołanie pełne
    - pełne, 73, 74
    - upraszczanie, 73
    - w pełni kwalifikowane, Patrz: odwołanie pełne
    - względne, 94, 96, 97, 130
  - Office Compatibility Pack, 37
  - okno
    - dialogowe, 241, 253, 258
      - dostosowywanie, 244
      - pobieranie odpowiedzi, 243
      - użytkownika, Patrz: UserForm
      - wbudowane, 242, 254
      - Wstawianie funkcji, 358
      - wyświetlanie, 243
    - Immediate, 84
    - Properties, 261
    - Toolbox, 260
    - wprowadzania danych, 146
  - OOP, Patrz: programowanie zorientowane
    - obiekto
  - operator, 121
    - dodawania, 121
    - dzielenia, 121
    - dzielenia całkowitego, 121
    - konkatenacji ciągów znaków, 121, 122, 207
    - kropki, 72, 73
    - Like, 355
    - logiczny
      - alternatywy, 122
      - alternatywy wykluczającej, 122
      - And, 122
      - Eqv, 122
      - Imp, 122
      - implikacji, 122
      - koniunkcji, 122
      - negacji, 122
      - Not, 122
      - Or, 122
      - równoważności, 122
      - XoR, 122
    - logiczny, 122
    - mnożenia, 121
    - Mod, 121, 122, 151
    - modulo, 121
    - odejmowania, 121
    - potęgowania, 121
    - priorytet, 122
    - znaku równości, 109
  - Option Explicit, 215, 384
- ## P
- pasek
    - postępu zadania, Patrz: wskaźnik
    - postępu zadania
    - przewijania, 261, 286, 289
    - szybkiego dostępu, 272
    - umieszczanie procedur, 299, 328
  - pętla, 95, 162, Patrz też: struktura
    - Do-Until, 153, 154, 168
    - Do-While, 153, 154, 167
    - For Each-Next, 168, 222, 232
    - For-Next, 153, 154, 162, 232
      - czas wykonania, 165
      - z instrukcją Exit For, 163
      - z instrukcją Step, 163
      - zagnieżdżona, 165
  - pierwiastek kwadratowy, 147
  - PivotTable, 34
  - plik
    - liczba bajtów, 146
    - nazwa, 146, 250
    - PERSONAL.XLSB, 54, 101
    - ścieżka, 146
    - wielkość, 143
  - pluskwa, Patrz: błąd programowania
  - podprogram, 59
  - Pokaż podziały stron, 227
  - pokrętło, 261, 287, 310
  - pole
    - etykiety, 261, 283
    - grupy, 261, 281

## pole

- karty, 261, 288
- kombi, 261, 280
- listy, 261, 280, 283, 303
- obrazu, 261, 282
- opcji, 261, 285, 296
- strony, 261, 284
- tekstowe, 261, 288, 289, 296, 310
- wyboru, 261, 279
- zakresu, 261, 286

## polecenie

- Add Watch, 213
- Debug.Print, 208, 384
- DisplayAlerts, 236, 378
- MsgBox, 384
- On Error Resume Next, 223
- Print, 212
- Randomize, 353
- Set, 237

## procedura

- argumenty, 82
- dysfunkcyjna, 34
- Function, Patrz: funkcja
- obsługi błędów
  - wbudowana, 196, 197
  - własna, 196
- obsługi zdarzenia, 173, 268
  - aktywacja arkusza, 180
  - aktywacja skoroszytu, 181
  - Open, 176
  - tworzenie, 173, 175
- obsługujące zdarzenie, 300
- separator, 66
- Sub, 33, 44, 58, 59, 60, 81, 82, 173,
  - Patrz też: makro
  - argumenty, 85, 87
  - nazwa, 83, 100
  - skrót klawiszowy, 41, 47, 86, 87, 271
  - tworzenie, 84
  - uruchamianie, 83
  - uruchamianie bezpośrednie, 85
  - uruchamianie w oknie dialogowym
    - Makro, 85
  - uruchamianie z poziomu innych
    - procedur, 89
  - uruchamianie za pomocą przycisków
    - i kształtów, 87, 88
  - uruchamianie za pomocą skrótów
    - klawiszowych, 86

- wywołanie, Patrz: procedura Sub
- substandardowa, 33
- udostępnienie użytkownikowi, 299
- uruchamianie, 60
- wyświetlająca okno dialogowe, 298

## Procedure Separator, 66

- program, Patrz: makro
- wykonywalny, 147
- wykonywanie krokowe, 211, 212
- wymuszanie zatrzymania, 207

## programowanie

- przykłady, 217
- strukturalne, 154
- zorientowane obiektowo, 69

## projekt, 54

- przycisk, 84, 87, 331
  - na pasku narzędzi Szybki dostęp, 31, 84
  - na Wstążce, 31
  - opcji, 267
  - polecenia, 261, 281
  - poleceń, 265
  - przłącznika, 261, 289
  - tworzenie, 31
  - wstawianie, 87, 88
- pułapka, Patrz: punkt przzerwania
- punkt przzerwania, 210, 211, 352
  - usuwanie, 210
  - wstawianie, 209

## R

- Range, 34, 73, 127, 129, 138, 217, 226
  - metoda, 137
- rata pożyczki, 148
- rejestr Windows, 146
- rejestrator makr, Patrz: makro rejestrator
- Require Variable Declaration, 64
- Require Variable Definition, 112
- RibbonX, 321
- runtime error, Patrz: błąd czasu wykonania

## S

- Sheets, 72
- skoroszyt, 34, 54
  - dezaktywacja, 183
  - konwersja na plik dodatku, 48
  - kopia zapasowa, 179

makr osobistych, 47, 54, 101  
 otwarty, 201  
 przekształcanie na dodatek, 364, 367  
 testowanie, 367  
 tryb obliczania  
   automatyczny, 118  
   przełączanie, 228  
   ręczny, 118, 235, 377  
 XLSM, 361  
 zapisywanie, 45  
 zawierający makro, 45  
 skrót klawiszowy, 86  
 słowo kluczowe, 109, 112  
   Call, 89  
   Case, 159  
   Const, 117  
   Dim, 109, 113, 119, 123  
   End, 109  
   End Function, 82  
   End Sub, 82  
   End With, 103  
   For, 109  
   Function, 82  
   Next, 109  
   Preserve, 125  
   Print, 212  
   Private, 113  
   Public, 113, 115, 123  
   Static, 113  
   Stop, 210  
   Sub, 82, 109  
   With, 103, 109  
 stała, 107, 117, 244  
   predefiniowana, 118  
   vbNewLine, 207, 377  
   vbProperCase, 270  
   xlCalculationAutomatic, 235  
   xlCalculationManual, 118, 235  
   xlCalculationSemiautomatic, 118  
   xlDown, 377  
   xlToLeft, 377  
   xlToRight, 377  
   xlUp, 377  
   zasięg, 117  
 string, Patrz: łańcuch znaków  
 strona podgląd podziału, 227, 228

struktura, Patrz też: pętla  
   End If, 156  
   For Each-Next, 168, 222, 232  
   If-Then, 153, 155, 156, 157, 158, 199  
   If-Then-Else, 154, 155, 156, 157  
   Select Case, 153, 154, 159, 228  
     zagnieżdżona, 160  
   With-End With, 233, 238, 377  
 suwak, 261, 286  
 synteza mowy, 354  
 system pomocy, 53, 78, 379  
   formanty, 279  
   funkcje wbudowane, 144  
   zakres, 129

## T

tabela, 146, 147, 220  
   kopiowanie, 220  
   nazwa, 220  
   przecawna, 34  
   wiersz nagłówka, 220  
 tablica, 107, 123, 132  
   deklarowanie, 123  
   dynamiczna, 124  
   liczba elementów, 124  
   wielowymiarowa, 124  
 TintAndShade, 135  
 tryb Break, 211, 212, 214

## U

UserForm, 54, 241, 257, 295  
   lista kontrolna, 318  
   niemodalne, 315, 316  
   poprawność danych, 302  
   prowadnice, 276  
   testowanie, 293, 299, 318  
   tworzenie, 258, 259, 264, 265, 318  
   właściwości, 261, 262  
   wykres, 317  
   wyświetlanie, 263  
   wyświetlanie na ekranie, 270  
   z wieloma kartami, 315  
   zamienniki, 241  
 ustawienia regionalne, 132

**V**

VBA, 29  
  fundamenty, 33  
  kod, 53, 56, 57  
    kopiowanie, 63  
    lokalizacja, 173, 174  
    optymalizacja, 234, 377, 384  
    wcięcia, 58, 65, 161, 215, 384  
  moduł, Patrz: moduł  
  Project, 43  
  Project Explorer, Patrz: VBA Project  
  wady, 33  
  zalety, 32  
VBE, 33, 42, 51, 79  
  funkcje, 144  
  menu podręczne, 52  
  okno, 52  
    Code, 53, 56  
    dokowanie, 68  
    Immediate, 53, 54, 208, 211, 212  
    Locals, 214  
    Project, 53, 54  
    Watch, 212, 213  
  pasek  
    menu, 52  
    narzędzi Edit, 66  
    narzędzi Standard, 53  
  środowiska dostosowanie, 63  
  Tools Options, 63, 66, 67, 68  
  uruchamianie, 51  
  wygląd, 66  
Visual Basic for Applications, Patrz: VBA

**W**

wartość  
  False, 134  
  Null, 134  
  True, 134  
watch expression, Patrz: wyrażenie monitorujące  
węzeł  
  Forms, 54  
  Modules, 54  
wiersza ukrywanie, 61  
wirus, 22  
właściwość, 74, 80, 127  
  Accelerator, 268  
  Address, 131, 133

Cells, 129  
Color, 135  
Column, 133  
Columns, 133  
Count, 133  
CurrentRegion, 219, 221  
DisplayAlerts, 169  
EntireRow, 221  
Font, 134  
Formula, 136  
FormulaLocal, 136  
HasFormula, 134  
Interior, 136  
IsAddin, 361  
NumberFormat, 137  
Offset, 130  
Path, 143  
Row, 133  
Rows, 133  
Text, 132  
ThemeColor, 135  
UsedRange, 224  
Value, 131  
Visible, 169  
Workbook, 54, 70, 71, 75  
Worksheet, 34, 70, 71, 127, 138  
wrapper function, Patrz: funkcja opakowująca  
wskaźnik postępu zadania, 312  
Wstążka, 84, 255, 321, 333  
  dostosowywanie, 321, 324  
    za pomocą kodu XML, 324, 329  
wykres, 34, 72, 229, 230  
  aktywowanie, 233  
  formatowanie, 233  
  na UserForm, 317  
  osadzony, 231  
  przetwarzanie, 231  
  właściwości modyfikowanie, 232  
wyrażenie, 120, 147  
  monitorujące, 212, 213

**Z**

zabezpieczeń ustawienia, 23  
zakres, 34  
  cała kolumna, 128, 218  
  cały wiersz, 128, 218  
  komórek, 34, 137, 217, 261, 307  
    jako argument funkcji, 349

- kopiowanie, 218
- nazwa, 218
- nieciągły, 226
- przenoszenie, 222
- nazwa, 127
- nieciągły, 128
- o zmiennej wielkości kopiowanie, 219
- określanie typu, 226
- zaznaczenie wielokrotne, 226
- zdarzenie, 77, 84, 171, 173
  - Activate, 172
  - aktywacyjne, 180
  - BeforeClose, 172, 179
  - BeforeDoubleClick, 172, 183
  - BeforePrint, 172
  - BeforeRightClick, 172, 184
  - BeforeSave, 172, 179, 180
  - Change, 172, 184
  - Deactivate, 172
  - dotyczące
    - arkusza, 172, 180, 181, 183
    - skoroszytu, 172, 176, 179, 182
  - NewSheet, 172
  - niezwiązane z obiektami, 186, 188
  - OnKey, 189
  - OnTime, 186, 187, 188
  - Open, 172, 176
  - SelectionChange, 172
  - SheetActivate, 172
  - SheetBeforeDoubleClick, 172
  - SheetBeforeRightClick, 172
  - SheetChange, 172
  - SheetDeactivate, 172
  - SheetSelectionChange, 172
  - WindowActivate, 172
  - WindowDeactivate, 172
- zegar analogowy, 188
- zmienna, 35, 107, 130, 157
  - czas życia, 116
  - deklarowanie, 111, 112, 215, 237, 383
  - globalna, 117
  - licznikowa, 162
  - lokalna, 114, 116, 214
  - łańcuchowa, 119
  - nazwa, 72, 109
  - niezainicjowana, 146
  - o zasięgu
    - jednego modułu, 115, 117
    - jednej procedury, Patrz: zmienna lokalna
  - o zasięgu globalnym, Patrz: zmienna globalna
  - obiektowa, 237, 377
  - przypisywanie wartości, 95
  - publiczna, Patrz: zmienna globalna
  - statyczna, 116
  - tekstowa, 377
  - typ, Patrz: dane typ
  - usuwanie z pamięci, 116
  - zasięg, 113, 114
- znak
  - >=, 156
  - ", 121
  - #, Patrz: znak krzyżyka
  - &, 121, 331
  - \*, 121
  - /, 121
  - ^, 121
  - +, 121
  - apostrofu, 107
  - cudzysłowu, 108, 127, 136
  - cudzysłowu podwójnego, 136
  - dolara, 133
  - Esc, 219
  - kontynuacji wiersza, 59, 121, 156, 378
  - kropki, 72, 73, 119
  - krzyżyka, 119
  - łamania wiersza, 246
  - nawias, 122, 142
  - przecinka, 119
  - równości, 77, 109, 121
  - średnika, 77, 125
  - zapytania, 212



# PROGRAM PARTNERSKI

GRUPY WYDAWNICZEJ HELION



1. ZAREJESTRUJ SIĘ
2. PREZENTUJ KSIĄŻKI
3. ZBIERAJ PROWIZJĘ

Zmień swoją stronę WWW  
w działający bankomat!

**Dowiedz się więcej i dołącz już dzisiaj!**

<http://program-partnerski.helion.pl>

GRUPA WYDAWNICZA



**Helion SA**

# Nie takie programowanie straszne...

Większość użytkowników Excela nie zwraca 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 zagraasz.

- **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.

**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>.



## 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

PO ROZUM NA...

[www.dlabystzakow.pl](http://www.dlabystzakow.pl)

Zamówienia telefoniczne:



0 801 339900



0 601 339900

septem  
septem.pl

Sprawdź najnowsze promocje: <http://dlabystzakow.pl/promocje>  
Książki najchętniej czytane: <http://dlabystzakow.pl/bestsellery>  
Zamów informacje o nowościach: <http://dlabystzakow.pl/nawosci>

Hellon SA: ul. Kościuszki 1c, 44-100 Gliwice, tel.: 32 230 98 63  
e-mail: [rady@dlabystzakow.pl](mailto:rady@dlabystzakow.pl) <http://dlabystzakow.pl>

Cena 49,00 zł

ISBN 978-83-246-7950-8



9 788324 679508