

Maciej Gonet

# ZROZUMIEĆ EXCELA

## VBA – MAKRA I FUNKCJE



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.

Wszystkie znaki występujące w tekście są zastrzeżonymi znakami firmowymi bądź towarowymi ich właścicieli.

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.

Redaktor prowadzący: Małgorzata Kulik

Projekt okładki: Studio Gravite / Olsztyn  
Obarek, Pokoński, Pazdrijowski, Zaprucki

Grafika na okładce została wykorzystana za zgodą Shutterstock.com

Wydawnictwo HELION  
ul. Kościuszki 1c, 44-100 GLIWICE  
tel. 32 231 22 19, 32 230 98 63  
e-mail: [helion@helion.pl](mailto:helion@helion.pl)  
WWW: <http://helion.pl> (księgarnia internetowa, katalog książek)

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

ISBN: 978-83-283-5694-8

Copyright © Helion 2019

Printed in Poland.

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

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

# Spis treści

<b>Wstęp .....</b>	<b>15</b>
<b>Rozdział 1. Struktura i podstawowe konstrukcje języka VBA .....</b>	<b>23</b>
Podstawowe informacje o języku programowania VBA .....	23
Edytor VBA .....	24
Uruchamianie i testowanie kodu .....	26
Makra i funkcje .....	28
Struktura modułów w VBA .....	29
<i>Polecenia opcji .....</i>	<i>29</i>
<i>Deklaracje zmiennych i stałych globalnych .....</i>	<i>30</i>
<i>Definicje podprogramów .....</i>	<i>31</i>
Makropolecenie utworzone w wyniku rejestracji .....	32
Skoroszyt makr osobistych i folder AddIns .....	34
Uruchamianie makropoleceń .....	34
Wprowadzanie kodu VBA w edytorze .....	35
Stosowanie nazw w kodzie VBA .....	35
Prosta funkcja zdefiniowana w VBA .....	37
Stosowanie komentarzy .....	38
Zmienne — typy i deklaracje .....	39
Określanie typu zmiennej w czasie wykonania .....	41
Deklaracje stałych .....	43
Wyrażenia .....	44
<i>Podwójne znaczenie znaku równości .....</i>	<i>44</i>
<i>Lista operatorów w Visual Basicu .....</i>	<i>44</i>
<i>Specyfika użycia operatorów relacji do argumentów różnych typów .....</i>	<i>47</i>
<i>Użycie operatora dodawania do danych tekstowych .....</i>	<i>48</i>
<i>Interpretacja tekstu pustego i wartości pustej w VBA i w Excelu .....</i>	<i>49</i>
<i>Specyfika dzielenia całkowitego i operacji modulo .....</i>	<i>51</i>
<i>Użycie funkcji w wyrażeniach .....</i>	<i>51</i>

<b>Rozdział 2. Obiekty, ich właściwości i metody .....</b>	<b>57</b>
Obiekt Range i jego właściwości .....	57
<i>Właściwość Value</i> .....	57
<i>Właściwość Formula i właściwości pokrewne</i> .....	58
<i>Właściwość NumberFormat</i> .....	59
<i>Właściwość Text</i> .....	60
<i>Właściwość Count</i> .....	62
<i>Uproszczony sposób zapisu odwołania do zakresu</i> .....	62
Metody .....	62
Hierarchia obiektów i nazwy kodowe .....	64
Wymiana informacji między arkuszem a kodem VBA .....	65
<i>Komórka aktywna i komórki wybrane</i> .....	65
<i>Użycie instrukcji wiążącej With</i> .....	66
<i>Użycie adresów bez kwalifikatora arkusza</i> .....	67
Sposoby odwołania do zakresu komórek przez adres .....	68
<i>Ogólna postać definicji zakresu</i> .....	68
<i>Wykorzystanie nazw</i> .....	69
<i>Wykorzystanie właściwości Cells</i> .....	69
<i>Tworzenie odwołań pośrednich</i> .....	70
<i>Wykorzystanie właściwości Offset i Resize</i> .....	71
<i>Odwołania do sąsiednich komórek</i> .....	72
Zmienne reprezentujące obiekty .....	72
Unia zakresów i zakresy złożone .....	73
Odwołania do wskazanych wierszy i kolumn .....	74
<i>Kopiowanie zakresu nieciągłego z zachowaniem jego struktury</i> .....	75
Metoda Find obiektu Range — wyszukiwanie adresu komórki o określonej zawartości .....	76
Obiekt zakresu a jego wartość .....	79
<i>Jawne odwołanie do wartości zakresu</i> .....	79
<i>Pośrednie sposoby odwołania do wartości zakresu</i> .....	80
<i>Specyfika zakresów złożonych</i> .....	81
Użycie autofiltra .....	82
<i>Składnia i znaczenie parametrów</i> .....	82
<i>Filtr wykluczający trzy i więcej wartości</i> .....	84
<i>Odczyt ustawień filtra</i> .....	85
Użycie filtra zaawansowanego .....	86
Kontrola procesu modyfikacji danych w arkuszu .....	88
<b>Rozdział 3. Sterowanie wykonaniem kodu i komunikacja w języku VBA .....</b>	<b>91</b>
Sterowanie wykonaniem kodu .....	91
<i>Instrukcje warunkowe i instrukcje wyboru</i> .....	91
<i>Sterowanie przebiegiem kompilacji — kompilacja warunkowa</i> .....	96

<i>Instrukcje pętli</i> .....	98
<i>Awaryjne przerwanie wykonywania funkcji lub procedury</i> .....	103
<i>Instrukcje skoku</i> .....	103
Komunikacja programu w Visual Basicu z użytkownikiem .....	107
<i>Wyświetlanie okienka komunikatów</i> .....	107
<i>Wyświetlanie komunikatów na pasku stanu</i> .....	110
<i>Pobieranie informacji od użytkownika</i> .....	112
<b>Rozdział 4. Użycie tablic w Visual Basicu</b> .....	<b>117</b>
Deklaracje tablic .....	117
<i>Tablice statyczne</i> .....	117
<i>Tablice dynamiczne</i> .....	118
<i>Sprawdzanie zakresu indeksów tablicy</i> .....	119
Tablice w zmiennych typu Variant .....	119
<i>Odwzorowanie zakresu komórek w tablicy</i> .....	119
<i>Funkcja Array</i> .....	121
<i>Funkcja Split</i> .....	122
<i>Specyfika deklaracji zmiennych typu Variant i ich użycia jako tablic</i> .....	123
<i>Teksty w tablicach dynamicznych</i> .....	124
Podstawowe operacje .....	125
<i>Nadawanie wartości elementom tablicy</i> .....	125
<i>Kasowanie zawartości tablicy</i> .....	127
<i>Użycie operatora Not w odniesieniu do tablicy</i> .....	128
Zmiana struktury tablic .....	129
<i>Rozbudowa tablicy jednowymiarowej o drugi wymiar</i> .....	129
<i>Modyfikacja pierwszego wymiaru tablicy dwuwymiarowej</i> .....	129
Tablice ułatwiają wymianę informacji z arkuszem .....	130
<i>Tablice i zakresy jako alternatywne argumenty funkcji</i> .....	130
<i>Funkcje generujące tablice bazowe do użycia w arkuszu</i> .....	131
<i>Wykorzystanie arkuszowej funkcji Index w VBA</i> .....	132
<i>Wypełnianie zakresu zawartością tablicy</i> .....	132
<i>Odwzorowanie zakresu w zmiennej obiektowej</i> .....	133
Stałe tablicowe Excela w VBA .....	134
<i>Odczyt wartości stałych tablicowych w kodzie VBA</i> .....	134
<i>Zagnieżdżanie metody Evaluate</i> .....	135
<i>Zamiana zakresu na stałą tablicową</i> .....	135
<i>Zamiana tablicy VBA na stałą tablicową</i> .....	138
Porównywanie tablic w VBA .....	139

<b>Rozdział 5. Funkcje definiowane przez użytkownika .....</b>	<b>143</b>
Sposoby przekazywania argumentów funkcji .....	143
Opcjonalne parametry funkcji .....	145
Użycie tablicy parametrów .....	147
<i>Problem pustych argumentów .....</i>	<i>147</i>
<i>Problem tablic wśród argumentów ParamArray .....</i>	<i>148</i>
Deklaracja nagłówka funkcji o zmiennej liczbie parametrów .....	149
Przekazywanie zmiennej liczby parametrów pomiędzy funkcjami .....	151
<i>Przekazywanie parametrów przez wartość .....</i>	<i>151</i>
<i>Przekazywanie parametrów przez referencję .....</i>	<i>153</i>
Funkcja zwracająca wynik w postaci tablicy .....	156
Przeliczanie wartości funkcji .....	157
Wywołanie makroinstrukcji z kodu VBA .....	158
Funkcje użytkownika podobne do funkcji standardowych .....	159
<b>Rozdział 6. Wybrane zastosowania tablic .....</b>	<b>163</b>
Przekształcenie tablicy dwuwymiarowej w jednowymiarową .....	163
Wyszukiwanie danych w strukturach dwuwymiarowych .....	166
<i>Wyszukiwanie w zakresie .....</i>	<i>166</i>
<i>Wyszukiwanie w tablicy .....</i>	<i>168</i>
Funkcje składające dowolne dane w tablicę .....	169
<i>Połączenie danych w tablicę jednowymiarową przez kopiowanie elementów .....</i>	<i>169</i>
<i>Połączenie danych w tablicę jednowymiarową za pośrednictwem tekstu .....</i>	<i>170</i>
<i>Ustalenie orientacji tablicy lub przekształcenie w tablicę dwuwymiarową .....</i>	<i>172</i>
Usuwanie wybranego wiersza lub kolumny z tablicy .....	173
Sortowanie danych w obszarach i tablicach .....	175
<i>Czyszczenie danych pochodzących z arkusza .....</i>	<i>177</i>
Grupowanie arkuszy w skoroszycie .....	178
<b>Rozdział 7. Metoda Evaluate i nazwy arkuszowe .....</b>	<b>181</b>
Metoda Evaluate .....	181
<i>Użycie funkcji OBLICZ z odwołaniem .....</i>	<i>182</i>
<i>Specyfika i ograniczenia metody Evaluate .....</i>	<i>184</i>
<i>Warianty użycia metody Evaluate .....</i>	<i>186</i>
<i>Kwalifikowane wywołanie funkcji używających metody Evaluate .....</i>	<i>186</i>
<i>Zagnieżdżona metoda Evaluate .....</i>	<i>187</i>
<i>Dane grupowe jako argumenty funkcji OBLICZ .....</i>	<i>188</i>
<i>Szacowanie formuł z polskimi nazwami funkcji .....</i>	<i>189</i>
<i>Użycie funkcji OBLICZ w obliczeniach iteracyjnych .....</i>	<i>190</i>
<i>Metoda Evaluate rozszerza możliwości funkcji ADR.POŚR .....</i>	<i>191</i>

<i>Użycie metody Evaluate w kodzie VBA</i> .....	192
<i>Alternatywa dla funkcji LICZ.JEŻELI i SUMA.JEŻELI</i> .....	193
<i>Uproszczenie zapisu wyrażeń wykorzystujących metodę Evaluate</i> .....	195
Funkcja Eval do szacowania wyrażeń w VBA .....	196
Obiekt Names .....	198
<i>Definiowanie nazw arkuszowych w kodzie VBA</i> .....	198
<i>Nazwy odnoszące się do zamkniętych skoroszytów</i> .....	205
<i>Stosowanie metody Evaluate do nazw arkuszowych</i> .....	206
<i>Nadawanie nazw stałym i formułom w arkuszu</i> .....	207
<b>Rozdział 8. Obsługa wyjątków i zdarzeń, kontrola poprawności danych</b> .....	<b>209</b>
Obsługa błędów wykonania w kodzie VBA .....	209
<i>Procedury niestandardowej obsługi błędów</i> .....	209
<i>Kody błędów w arkuszu</i> .....	215
<i>Funkcje walidujące wartość wyrażenia</i> .....	217
Śledzenie wykonania kodu VBA — obiekt Debug .....	218
Identyfikacja miejsca, z którego wywołano makro lub funkcję .....	219
<i>Identyfikacja komórki, z której wywołano funkcję UDF</i> .....	220
Procedury obsługi zdarzeń .....	221
<i>Koncepcja zdarzeń</i> .....	221
<i>Włączanie zgody na iteracje przed otwarciem skoroszytu</i> .....	222
<i>Operacje na zakresie wskazanym myszką</i> .....	223
<i>Poprzednia zawartość komórek</i> .....	224
<i>Dziedzictwo przeszłości — właściwości OnDoubleClick</i> <i>    oraz OnEntry obiektu Application</i> .....	231
<i>Konsolidacja z automatyczną aktualizacją</i> .....	232
Kontrola poprawności danych .....	233
<i>Obiekt Range.Validation, jego metody i właściwości</i> .....	233
<i>Ograniczenia źródła listy rozwijanej</i> .....	236
<i>Ustalanie listy poprawności danych za pomocą formuły</i> .....	237
<i>Powiązanie listy poprawności danych z formatowaniem warunkowym</i> .....	238
<i>Wyświetlanie komunikatu o konieczności nowego wyboru w przypadku list zależnych</i> .....	240
<i>Modyfikacja sposobu wyświetlania listy rozwijanej</i> .....	241
<i>Ochrona listy rozwijanej przed nadpisaniem</i> .....	243
<i>Wprowadzanie danych z podpowiedzią</i> .....	244
<b>Rozdział 9. Adaptacja wybranych funkcji i metod VBA</b> <b>do użycia jako funkcji arkuszowych</b> .....	<b>247</b>
Funkcja Val .....	247
Metoda InputBox .....	249
Przekazywanie wartości argumentów do formuł nazwanych .....	252

Metoda ConvertFormula .....	253
Funkcja CallByName — wywoływanie metod przez tekst ich nazwy .....	254
Dostęp do stałych predefiniowanych w VBA .....	258
Odczyt nazwy lub numeru arkusza .....	260
Wyświetlanie tekstów formuł w arkuszu .....	261
Dodawanie i edycja komentarzy w komórkach arkusza .....	262
<i>Metoda NoteText (notatka tekstowa)</i> .....	262
<i>Właściwość Comment obiektu Range</i> .....	263
<i>Metoda AddComment obiektu Range</i> .....	263
<i>Metody ClearNotes i ClearComments z obiektu Range</i> .....	264
<i>Obiekt Comment i jego komponenty</i> .....	264
<i>Sposób wyświetlania komentarzy i znaczników komentarza</i> .....	266
<i>Kolekcja Comments</i> .....	267
<i>Metoda SpecialCells obiektu Range</i> .....	267
<i>Wstawianie i edycja komentarzy przez funkcje UDF</i> .....	269
Wartości w komórkach scalonych .....	270
Określanie formatu komórek w VBA .....	272
<i>Odczyt i zapis kodu formatu</i> .....	272
<i>Wykorzystanie informacji udostępnianych przez funkcję Format</i> .....	273
<i>Inne funkcje do formatowania w VBA</i> .....	274
<b>Rozdział 10. Wybrane problemy obliczeniowe .....</b>	<b>277</b>
Zaokrąglanie liczb .....	277
<i>Zaokrąglanie liczb z uwzględnieniem cyfr znaczących</i> .....	277
<i>Uwzględnienie zasady cyfry parzystej przy zaokrąglaniu</i> .....	278
Obliczanie wartości wielomianu .....	278
Obliczanie pierwiastków równania kwadratowego .....	279
Rozwiązywanie równań nieliniowych z wykorzystaniem metody GoalSeek .....	280
Całkowanie numeryczne .....	282
Wspomaganie obliczania szeregów .....	285
<i>Iloczyn pierwszych elementów tablicy</i> .....	285
Rozwiązanie równania różniczkowego metodą Rungego-Kutty .....	287
Wspomaganie wykonywania wykresów .....	288
<i>Wykresy funkcji opisanych wzorem</i> .....	288
<i>Korekta danych do wykresów funkcji nieciągłych</i> .....	289
<i>Wykresy funkcji nieciągłych — wykrywanie nieciągłości</i> .....	291
<i>Ukrywanie zawartości komórek w komentarzach</i> .....	293
Generowanie liczb pseudolosowych w VBA .....	294
Wybrane zagadnienia kombinatoryki .....	295
<i>Generowanie permutacji</i> .....	295
<i>Generowanie permutacji z powtórzeniami</i> .....	297



<i>Generowanie kombinacji</i> .....	300
<i>Planowanie serii rozgrywek sportowych</i> .....	301
Zwiększona dokładność obliczeń .....	303
Zamiana odwołań w wyrażeniu na wartości .....	307
<b>Rozdział 11. Przykłady zastosowań makroinstrukcji w chemii</b> .....	<b>309</b>
Obliczanie masy molowej .....	309
Przeliczanie stężeń roztworów .....	312
Modyfikacja wyglądu i zawartości obiektów na poziomie znaków .....	319
<i>Łączenie tekstów sformatowanych</i> .....	321
<i>Formatowanie wzorów chemicznych</i> .....	323
<i>Formatowanie wzorów chemicznych — inaczej</i> .....	325
Formatowanie tekstu za pomocą polecenia SendKeys .....	328
<i>Polecenie SendKeys</i> .....	328
<i>Wybrane skróty klawiaturowe, które działają w trybie edycji</i> .....	330
<i>Zastosowania polecenia SendKeys do formatowania komórki z tekstem</i> .....	332
<b>Rozdział 12. Alternatywne struktury danych: kolekcje i słowniki</b> .....	<b>335</b>
Kolekcje .....	335
<i>Tworzenie kolekcji i usuwanie jej elementów</i> .....	335
<i>Odczyt elementów kolekcji</i> .....	337
<i>Specyfika kolekcji w porównaniu z tablicami</i> .....	338
<i>Kolekcje jako argumenty procedur i funkcji oraz wynik funkcji</i> .....	339
<i>Ograniczenia kolekcji i sposoby ich obejścia</i> .....	340
Słowniki .....	341
<i>Tworzenie słownika</i> .....	341
<i>Odczyt i modyfikacja zapisów</i> .....	343
<i>Klucze mogą być obiektami zakresów (komórkami)</i> .....	345
<i>Różnice między kolekcją a słownikiem</i> .....	345
<i>Wykorzystanie słownika w praktyce</i> .....	346
<i>Kopiowanie słownika</i> .....	348
<b>Rozdział 13. Operacje na danych oznaczających datę i czas</b> .....	<b>351</b>
Podstawy operowania datami w VBA .....	352
<i>Rozpoznawanie dat w arkuszu za pomocą VBA</i> .....	353
<i>Interpretacja tekstu w arkuszu jako daty</i> .....	354
<i>Funkcja UDF do konwersji dat w formie tekstu</i> .....	355
Przegląd funkcji VBA do operacji na datach .....	356
<i>Funkcje Date, Time i Now</i> .....	356
<i>Funkcje DateValue i TimeValue</i> .....	356
<i>Funkcje DateSerial i TimeSerial</i> .....	356

<i>Funkcja IsDate</i> .....	356
<i>Funkcja DateAdd</i> .....	357
<i>Funkcja DatePart</i> .....	358
<i>Funkcja DateDiff</i> .....	359
<i>Funkcja FormatDateTime</i> .....	359
<i>Funkcja Format</i> .....	360
<i>Funkcja MonthName</i> .....	364
<i>Funkcja WeekdayName</i> .....	364
<i>Funkcja Timer</i> .....	364
<i>Metoda Wait</i> .....	365
<i>Metoda OnTime</i> .....	365
<i>Dokładny pomiar czasu</i> .....	366
<i>Ciągłe wyświetlanie czasu — stoper sekundowy</i> .....	369
<i>Ciągłe wyświetlanie czasu — inne rozwiązanie stopera</i> .....	371
<i>Niestandardowe minuty w indeksie górnym (VBA)</i> .....	374
<i>Właściwości Value i Value2 obiektu Range w rozpoznawaniu dat</i> .....	374
<i>Data i czas systemowy w kodach formatu</i> .....	375

## **Rozdział 14. Operacje z udziałem tekstów ..... 377**

<i>Funkcje do operacji na tekstach w VBA</i> .....	377
<i>Funkcje LTrim, RTrim i Trim (\$)</i> .....	377
<i>Funkcje Chr i ChrW (\$)</i> .....	377
<i>Funkcje Asc i AscW</i> .....	378
<i>Funkcje Hex i Oct (\$)</i> .....	378
<i>Funkcje LCase i UCase (\$)</i> .....	378
<i>Funkcje Left i Right (\$)</i> .....	378
<i>Funkcja Len</i> .....	379
<i>Funkcja Mid (\$)</i> .....	379
<i>Instrukcja Mid</i> .....	379
<i>Funkcje Space i String (\$)</i> .....	380
<i>Funkcja Format (\$)</i> .....	380
<i>Instrukcje LSet i RSet</i> .....	383
<i>Funkcje InStr i InStrRev</i> .....	383
<i>Funkcja StrComp</i> .....	384
<i>Funkcja StrConv</i> .....	385
<i>Funkcja StrReverse</i> .....	386
<i>Funkcja Replace</i> .....	386
<i>Funkcja Filter</i> .....	387
<i>Funkcja Join</i> .....	388

Łączenie tekstów z użyciem VBA .....	388
<i>Wykorzystanie operatora złączenia tekstów</i> .....	388
<i>Inny wariant z operatorem złączenia tekstów</i> .....	389
<i>Wykorzystanie funkcji Join</i> .....	390
<i>Funkcja Join w wersji minimum</i> .....	390
Operator Like — porównywanie tekstów .....	392
<i>Położenie pierwszej i ostatniej cyfry w tekście</i> .....	393
Podział tekstu w kolumnie .....	394
<i>Metoda TextToColumns</i> .....	394
<i>Metoda Parse</i> .....	397
<b>Rozdział 15. Operowanie kolorami i formatowanie warunkowe .....</b>	<b>399</b>
Sposób przedstawiania kolorów .....	399
Odczyt koloru tła lub czcionki bez użycia VBA .....	402
Wykorzystanie kolorów do oznaczania komórek w arkuszu .....	403
Rozjaśnianie i ściemnianie kolorów .....	404
Operacje na komórkach sformatowanych w określony sposób .....	405
<i>Sumowanie komórek wykorzystujących ten sam kolor czcionki</i> .....	405
<i>Bezpośrednie oznaczanie komórek numerami kolorów tła</i> .....	406
<i>Przyspieszenie reakcji na zmiany kolorów</i> <i>przez wykorzystanie niestandardowego zdarzenia</i> .....	409
Symulowana lista rozwijana .....	411
Formatowanie warunkowe w VBA .....	412
<i>Dostęp do definicji formatów warunkowych z poziomu VBA</i> .....	412
<i>Przeliczanie reguł formatowania warunkowego</i> .....	418
<i>Definiowanie reguł formatowania warunkowego za pomocą funkcji UDF</i> .....	419
<i>Kopiowanie formatowania warunkowego z zamianą na formatowanie stałe</i> .....	425
<i>Symulacja skali barw za pomocą VBA</i> .....	427
<i>Selektywne kopiowanie reguł formatowania warunkowego</i> .....	428
<b>Rozdział 16. Ograniczenia i możliwości funkcji UDF .....</b>	<b>429</b>
Funkcje UDF wywoływane bezpośrednio .....	429
<i>Modyfikacja parametrów wykresu</i> .....	431
Funkcje UDF wywoływane w sposób pośredni .....	432
<i>Ogólne wskazówki co do użycia metody Evaluate</i> .....	432
<i>Definiowanie nazw za pomocą funkcji UDF</i> .....	434
<i>Zmiana elementów formatowania innych komórek</i> .....	439
<i>Zmiana zawartości i koloru czcionki w innych komórkach</i> .....	441
<i>Usunięcie zawartości komórek</i> .....	443
<i>Wpis wartości do innej komórki</i> .....	444
<i>Tworzenie list rozwijanych</i> .....	445

Uruchamianie funkcji UDF za pomocą funkcji HIPERŁĄCZE .....	446
<i>Specyfika działania funkcji HIPERŁĄCZE</i> .....	446
<i>Użycie funkcji UDF w hiperłączy</i> .....	447
Wykonanie kodu VBA przy uaktywnieniu listy rozwijanej .....	449
Współdziałanie funkcji UDF z procedurami obsługi zdarzeń .....	450
<i>Kopiowanie komórek z pełnym formatowaniem</i> .....	451
<i>Funkcja wyszukiwania zwracająca wynik sformatowany</i> .....	453
<i>Rzeczywista długość tekstu</i> .....	455
<i>Formatowanie fragmentu tekstu</i> .....	458
<i>Funkcja łącząca teksty sformatowane</i> .....	460
<b>Rozdział 17. Ciekawe pomysły z użyciem VBA .....</b>	<b>465</b>
Monitorowanie zmian w komórkach .....	465
<i>Działanie jednorazowe</i> .....	465
<i>Działanie wielokrotne</i> .....	466
Odczyt danych z zamkniętego skoroszytu .....	466
<i>Rozwiązanie klasyczne</i> .....	467
<i>Rozwiązanie z wykorzystaniem ADO</i> .....	467
<i>Konstrukcja odwołań zewnętrznych</i> .....	469
<i>Wykorzystanie nazw arkuszowych</i> .....	472
Nadmierna objętość skoroszytu po usunięciu części danych .....	473
Wyznaczenie różnicy zakresów .....	475
Wyraźne zaznaczenie komórki aktywnej lub zakresu selekcji .....	476
Konwersja stylu liczb za pomocą narzędzi Visual Basic .....	477
<i>Metoda Range.Replace</i> .....	477
<i>Metoda Range.TextToColumns</i> .....	478
<i>Funkcja Replace</i> .....	479
<i>Funkcja WorksheetFunction.Substitute</i> .....	480
<i>Funkcja Val</i> .....	481
<i>Funkcja CDBl</i> .....	481
<i>Funkcje Str i CStr</i> .....	481
<i>Funkcja Format</i> .....	482
Obliczenia uwzględniające ukryte kolumny .....	483
<i>Rozwiązanie wykorzystujące właściwość Width</i> .....	483
<i>Przeliczanie formuł</i> .....	484
Ustalanie absolutnej wielkości komórki .....	485
Problemy z użyciem metody AutoFit .....	488
Przełączanie między alternatywnymi wynikami w komórce za pomocą klawiatury .....	489
Ochrona komórek przed przypadkową edycją .....	491

Powiększenie zaznaczonych komórek .....	492
Użycie schowka systemowego z poziomu VBA .....	494
<i>Obiekt pośredniczący DataObject</i> .....	494
<i>Wykorzystanie samego obiektu DataObject bez schowka</i> .....	497
<i>Zapis do schowka w Windows 8 i 10</i> .....	498
<i>Opróżnianie schowka</i> .....	500
<i>Użycie funkcji UDF</i> .....	501
<b>Rozdział 18. Graficzne elementy sterujące (kontrolki ekranowe, formanty) .....</b>	<b>503</b>
Rodzaje graficznych elementów sterujących i ich przeznaczenie .....	503
Dodawanie kontroltek (formantów) do arkusza .....	504
<i>Formanty formularza</i> .....	507
<i>Formanty ActiveX</i> .....	510
<i>Niestandardowe wykorzystanie pola listy</i> .....	516
<i>Zamienniki formantów pola tekstowego</i> .....	517
Obiekty graficzne w kodzie VBA .....	520
<i>Pola tekstowe i formanty formularza</i> .....	521
<i>Formanty ActiveX</i> .....	522
<i>Obiekty graficzne w funkcjach UDF</i> .....	523
Makroinstrukcje przypisywane do formantów w arkuszu .....	524
<i>Formanty formularza</i> .....	525
<i>Formanty ActiveX</i> .....	525
<b>Rozdział 19. Funkcje makr Excela w wersji 4.0 .....</b>	<b>529</b>
Geneza makr XLM .....	529
Podstawy użycia makr XLM w arkuszu .....	530
<i>Wstawianie arkuszy makr do skoroszytu</i> .....	530
<i>Dostęp do funkcji makr XLM</i> .....	530
<i>Wykorzystanie Międzynarodowego arkusza makr</i> .....	531
<i>Przeliczanie formuł w arkuszach makr</i> .....	532
<i>Wywołanie funkcji makr XLM w kodzie VBA</i> .....	533
<i>Bezpieczeństwo użycia makr XLM</i> .....	534
Odwołania do zakresów wielokomórkowych w funkcjach XLM .....	534
Przekazywanie parametrów w wywołaniu formuły nazwanej .....	536
<i>Jeden parametr</i> .....	536
<i>Dwa parametry</i> .....	537
Przykłady użycia funkcji makr XLM w arkuszu i w formułach nazwanych .....	538
Ukryta przestrzeń nazw .....	546

<b>Rozdział 20. Przegląd funkcji makr XLM .....</b>	<b>547</b>
Składnia funkcji makr Excela 4.0 i opis znaczenia parametrów .....	547
<i>ADR.TEKST = REFTEXT</i> <i>Kategoria: Wyszukiwania i adresu</i> .....	547
<i>DOKUMENTY = DOCUMENTS</i> <i>Kategoria: Informacyjne</i> .....	548
<i>FORMUŁA.TRYB.ADR = FORMULA.CONVERT</i> <i>Kategoria: Wyszukiwania i adresu</i> .....	549
<i>KOM.AKT = ACTIVE.CELL</i> <i>Kategoria: Informacyjne</i> .....	549
<i>NAZWY = NAMES</i> <i>Kategoria: Informacyjne</i> .....	550
<i>O.APLIKACJI = GET.WORKSPACE</i> <i>Kategoria: Informacyjne</i> .....	551
<i>O.DEFINICJI = GET.DEF</i> <i>Kategoria: Informacyjne</i> .....	558
<i>O.DOKUMENCIE = GET.DOCUMENT</i> <i>Kategoria: Informacyjne</i> .....	559
<i>O.FORMULE = GET.FORMULA</i> <i>Kategoria: Informacyjne</i> .....	566
<i>O.KOMÓRCE = GET.CELL</i> <i>Kategoria: Informacyjne</i> .....	567
<i>O.NAZWIE = GET.NAME</i> <i>Kategoria: Informacyjne</i> .....	572
<i>O.NOTATCE = GET.NOTE</i> <i>Kategoria: Informacyjne</i> .....	574
<i>O.OBIEKCIE = GET.OBJECT</i> <i>Kategoria: Informacyjne</i> .....	574
<i>O.OPCJACH.LISTY = OPTIONS.LISTS.GET</i> <i>Kategoria: Informacyjne</i> .....	582
<i>O.SKOROSZYCIE = GET.WORKBOOK</i> <i>Kategoria: Informacyjne</i> .....	582
<i>OKNA = WINDOWS</i> <i>Kategoria: Informacyjne</i> .....	585
<i>PLIKI = FILES</i> <i>Kategoria: Informacyjne</i> .....	586
<i>SZACUJ = EVALUATE</i> <i>Kategoria: Wyszukiwania i adresu</i> .....	586
<i>ZAZNACZENIE = SELECTION</i> <i>Kategoria: Informacyjne</i> .....	588
<b>Literatura cytowana i uzupełniająca .....</b>	<b>589</b>
<b>Skorowidz .....</b>	<b>590</b>

## Rozdział 5.

# Funkcje definiowane przez użytkownika

W zastosowaniach naukowych najczęściej wykorzystuje się Visual Basic do definiowania funkcji, które mają uzupełnić dostępny standardowo zestaw o funkcje realizujące specyficzne potrzeby obliczeniowe. Wstępne informacje o funkcjach podano w rozdziale 1., w podrozdziale „Makra i funkcje”, dalej w punkcie „Definicje podprogramów”, a prosty przykład funkcji zamieszczono w tym samym rozdziale, w podrozdziale „Prosta funkcja zdefiniowana w VBA”. Teraz deklarowanie i definiowanie funkcji zostanie omówione bardziej szczegółowo.

## Sposoby przekazywania argumentów funkcji

Jeśli argument funkcji (lub procedury **Sub**) jest zmienną, to może być przekazany do funkcji wywołującej dwoma sposobami: przez **odwołanie** (używa się również określeń referencja lub wskazanie) lub przez **wartość**. Pierwszy sposób jest w VBA domyślny, gdyż jest szybszy i bardziej efektywny, jeśli chodzi o gospodarkę pamięcią, ale nie zawsze może być użyty. Jeśli chcemy przekazać argument przez wartość, należy poprzedzić jego nazwę w deklaracji funkcji słowem kluczowym **ByVal**. Jeśli natomiast chcemy zaakcentować chęć posłużenia się odwołaniem, możemy analogicznie użyć słowa **ByRef**, jednak nie jest to konieczne, gdyż jest to ustawienie domyślne. W przypadku odwołania, do funkcji wywołującej przekazywany jest faktycznie adres zmiennej i w związku z tym wszystkie zmiany dokonane na tej zmiennej będą widoczne również po wyjściu z funkcji (ogólniej: procedury). W przeciwieństwie do tego, gdy argument jest przekazywany przez wartość, tworzona jest kopia argumentu o tej samej wartości; procedura operuje na tej kopii, a oryginalna zmienna zachowuje swoją niezmienną wartość.

Przeanalizujmy to na prostym przykładzie.

```
Sub Sumuj ()  
    Dim a  
    a = 5  
    Debug.Print Pomnóż(3, a) + a      ' z ByVal 20, bez ByVal 30  
    Debug.Print Pomnóż(5, a) + a      ' z ByVal 30, bez ByVal 150  
End Sub
```

```

Function Pomnóż(x, ByVal y)
    y = x * y
    Pomnóż = y
End Function

```

Ze względu na utrudnioną analizę kodu należy unikać sytuacji, gdy wywołanie funkcji powoduje „skutki uboczne”, jak w powyższym przykładzie zmianę wartości zmiennej *a* w razie pominięcia słowa **ByVal**.

Przy przekazywaniu argumentów przez odwołanie należy upewnić się, że typy powiązanych zmiennych są identyczne, w przeciwnym razie wystąpi błąd kompilacji *ByRef Argument Type Mismatch*. Jeżeli argumenty są przekazywane przez wartość, typ zmiennej zostanie automatycznie dopasowany, jeśli to będzie możliwe.

Należy zwrócić uwagę na specyfikę przekazywania argumentów, gdy są nimi odwołania do obiektów, najczęściej zakresów. Jeśli parametr procedury jest zadeklarowany jako **Range** lub **Object**, to nie ma znaczenia, czy użyjemy wcześniej **ByVal** lub **ByRef**. Nie można utworzyć lokalnej kopii obiektu w procedurze i w związku z tym operacje odnoszące się do obiektu są od razu widoczne w tym obiekcie. Natomiast jeśli parametr procedury jest zadeklarowany jako **Variant** lub inny typ standardowy, a jako argument przekazemy odwołanie do zakresu (lub innego obiektu), to sytuacja jest bardziej złożona. Jeśli użyjemy słowa **ByVal**, to jest tworzona lokalna zmienna o wartości pobranej z zakresu, a sam zakres nie jest modyfikowany. Jeśli jawnie lub przez domniemanie użyjemy słowa **ByRef**, a argument zostanie przekazany w formie — na przykład — **Range("C5")** lub **[C5]**, to wartość zakresu również tym razem zostanie tylko skopiowana do zmiennej lokalnej. Jedynie w przypadku, gdy parametr procedury jest zadeklarowany jako **Variant**, a argument zostanie przekazany w formie zmiennej obiektowej (przez nazwę), to wówczas faktycznie ta zmienna obiektowa będzie modyfikowana (czyli zachowanie będzie takie, jak przy użyciu deklaracji **As Range** albo **As Object**).

Poniższy listing ilustruje ten najbardziej nietypowy przypadek.

```

Private Sub ChangeValue(ByRef a) 'As Variant
    ' Jeśli zmienna a jest zadeklarowana jako wariant, to na ogół przekazywana jest wartość
    ' a obiekt tylko wtedy, gdy argumentem będzie nazwana zmienna obiektowa
    Debug.Print a
    a = 35
    Debug.Print a
    Debug.Print [A6] 'wartość komórki zmieniona, jeśli argumentem jest obiekt
End Sub

Sub TestChV()
    Dim b As Range
    Set b = [A6]
    ChangeValue b 'b -> argumentem jest obiekt, [A6], Range("A6") -> argumentem jest wartość
End Sub

```

Następna komplikacja dotyczy tablic. Całe tablice (deklarowane po słowach **Dim**, **ReDim** lub **Static**) muszą być przekazywane przez referencję, natomiast pojedyncze elementy tablicy można przekazywać przez wartość. Jeżeli chcemy przekazać taką tablicę do innej funkcji przez wartość (aby uniknąć modyfikacji oryginalnej tablicy), to mamy do wyboru dwa sposoby: albo w wywołaniu



nazwę tablicy umieścimy w nawiasach, albo utworzymy kopię oryginalnej tablicy w zmiennej typu **Variant** i tę kopię prześlemy jako argument do dalszego przetwarzania. Sytuację ilustruje poniższy listing:

```
Sub TestDim()
    Dim a(1 To 3) As Long
    a(1) = 1: a(2) = 4: a(3) = 9
    'Wariant 1
    Modyfikuj a           ' tablica a zostanie zmodyfikowana
    'Wariant 2
    ' Modyfikuj (a)      ' tablica a nie zostanie zmodyfikowana
    'Wariant 3
    ' Dim kopia
    ' kopia = a
    ' Modyfikuj kopia    ' tablica a nie zostanie zmodyfikowana
    Debug.Print a(1); a(2); a(3)
End Sub

Private Sub Modyfikuj(t)
    Dim u, v, i
    For i = LBound(t) To UBound(t)
        v = t(i)
        t(i) = v + u
        u = v
        Debug.Print t(i);
    Next i
    Debug.Print
End Sub
```

Słowa **ByVal** lub **ByRef** mogą być używane wraz ze słowem **Optional**, natomiast nie mogą być używane do tablic **ParamArray** (patrz podrozdział „Użycie tablicy parametrów”).

## Opcjonalne parametry funkcji

W deklaracji funkcji można określić zarówno typy parametrów, jak i typ wyniku. Jak w przypadku deklaracji zmiennych, służy do tego słowo kluczowe **As** lub znak deklaratywny. Brak deklaracji typu oznacza typ **Variant**.

Niektóre — lub nawet wszystkie — parametry funkcji można określić jako opcjonalne (nieobowiązkowe). Muszą one wystąpić jako ostatnie na liście parametrów. Nazwę takiego parametru w deklaracji poprzedzić trzeba słowem **Optional**. Jeżeli jest więcej niż jeden taki parametr, słowo to powinno się znaleźć przed każdym z nich. W deklaracji funkcji można podać jego wartość domyślną, która będzie użyta w razie braku argumentu w wywołaniu funkcji. Typ parametru opcjonalnego może być określony jawnie, może też pozostać nieokreślony (wtedy jest to typ **Variant**). Może to wyglądać na przykład tak:

```
Optional wsp As Double = 1
```

lub

```
Optional wsp# = 1
```

Jednak deklaracja

```
Optional wsp = 1#
```

ma inne znaczenie. W tym drugim przypadku parametr *wsp* jest typu **Variant**, a tylko jest inicjowana stałą typu **Double**. W wywołaniu lub treści funkcji można mu nadawać wartości innych typów.

Jeśli wartość domyślna parametru opcjonalnego nie została określona, a w wywołaniu wartość odpowiedniego argumentu nie została podana, to jeśli parametr był typu **Variant**, przyjmuje on specjalną wartość *Missing*, która nie może być użyta do obliczeń (ale może być zidentyfikowana — patrz niżej), a jeśli typ parametru był określony, przyjmuje on wartość, jaką domyślnie przyjmują zmienne określonego typu.

Parametr opcjonalny musi reprezentować pojedynczą wartość, na przykład nie może być jawną tablicą, ale może być zmienną typu **Variant** zawierającą tablicę.

Inaczej niż w przypadku funkcji Excela, pominięcie argumentu obowiązkowego w wywołaniu funkcji VBA zawsze prowadzi do błędu, natomiast w przypadku parametrów opcjonalnych nie ma znaczenia, czy pominięto tylko wartość argumentu, a pozostawiono separator, czy też pominięto argument wraz z separatorem. W obu przypadkach przyjmuje się wartość domyślną argumentu, a gdy ta nie została określona (ale typ parametru był określony) — wartość zerową lub pustą.

W kodzie funkcji można sprawdzać obecność lub brak argumentu opcjonalnego typu **Variant** za pomocą funkcji **IsMissing**. Funkcja ta zwraca **True**, jeśli argument pominięto i nie została określona wartość domyślna. Termin **IsMissing** należy rozumieć dosłownie — jeśli argument ma wartość **Null** lub **Empty**, to **IsMissing** zwraca **False**. W każdym innym przypadku funkcja również zwraca **False**. Także wtedy, gdy parametr miał określony typ, a argument pominięto (bo wtedy jest nadawana standardowa wartość domyślna — zerowa lub pusta). Parametru odpowiadającego pominiętemu argumentowi (dla którego funkcja **IsMissing** zwróciłaby **True**) nie można używać w obliczeniach, w razie użycia wyrażenie zwraca błąd. Można jednak przekazać wartość *Missing* do innej procedury lub nadać ją innej zmiennej pod warunkiem, że obie zmienne są typu **Variant**. Jest to specyficzna wartość, którą można odczytać tylko za pomocą funkcji **IsMissing**. Samo słowo *Missing* **nie jest** słowem kluczowym, nie ma więc możliwości nadania tej wartości przez przypisanie stałej, jak w przypadku wartości **Null** czy **Empty**. Parametrowi, któremu nie nadano wartości przy wywołaniu, można nadać wartość w treści procedury, ale wtedy — oczywiście — zmienna już nie będzie *Missing*. Oznacza to, że po nadaniu wartości nie można już sprawdzić, czy ta wartość została przekazana przy wywołaniu procedury, czy nadana w jej treści.

Alternatywą testu **IsMissing** jest test **IsError**. Brak argumentu generuje wartość błędu 448, któremu odpowiada opis *Named argument not found*. Opis ten nie jest w pełni adekwatny do sytuacji, ale czasem taki test może być wygodniejszy. Numer błędu odczytamy, stosując konwersję na tekst **CStr(zmienna)**. Wynikiem będzie tekst *Error 448*.

## Użycie tablicy parametrów

Inny sposób przekazania zmiennej liczby argumentów do funkcji polega na użyciu tablicy **ParamArray** jako parametru. Może być tylko jedna taka tablica i musi występować jako ostatnia na liście parametrów. Jeśli zadeklarowano użycie tablicy **ParamArray**, nie można używać parametrów opcjonalnych. Po jej nazwie należy umieścić parę pustych nawiasów. Jest to tablica złożona z elementów typu **Variant** (czyli typu danych nie należy określać). Dolny indeks tej tablicy zawsze jest równy 0, niezależnie od ewentualnego użycia dyrektywy **Option Base 1**. Górny indeks można sprawdzić w kodzie naszej funkcji, używając funkcji **UBound**. Jeśli do tablicy parametrów nie przekazano żadnych argumentów, funkcja **UBound** ma wartość  $-1$ .

Na liście argumentów przy wywołaniu funkcji każdy argument składający się na tablicę **ParamArray** musi wystąpić oddzielnie, oddzielony przecinkiem (w arkuszu średnikiem) od następnych. Stanowią one kolejne elementy tablicy. Nie można użyć zakresu i oczekiwać, że poszczególne jego elementy zostaną potraktowane jako kolejne argumenty. Zakres może być użyty, ale stanie się on pojedynczym argumentem i kod funkcji musi wydobyć z niego poszczególne wartości. Przyjmując, że tablica **ParamArray** ma nazwę *a*, za pomocą funkcji **IsMissing(a)** można sprawdzić, czy przy wywołaniu funkcji do tablicy **ParamArray** przekazano jakiegokolwiek argumenty; jeśli przekazano choć jeden, obecność argumentu o numerze *i* można sprawdzić funkcją **IsMissing(a(i))**. Alternatywnie, podobnie jak w przypadku argumentów opcjonalnych, można wykorzystać test **IsError**.

Przy wywołaniu funkcji z poziomu VBA nie można przekazywać argumentów składających się na tablicę **ParamArray** jak argumentów nazwanych, to znaczy, że na przykład nie można użyć konstrukcji `a(1) := 2`.

## Problem pustych argumentów

Używając funkcji z tablicą **ParamArray** należy zwrócić uwagę na użycie odwołań do **pustych komórek**, gdyż one również wywołują błąd. Normalnie funkcje agregujące radzą sobie dobrze z problemem pustych argumentów i pustych komórek. Można na przykład bezpiecznie napisać:

```
= ILOCZYN(3; ; 7)          lub          = ILOCZYN(3; pusta; 7)
```

gdzie *pusta* jest odwołaniem do pustej komórki. W obu przypadkach otrzymamy w wyniku 21.

Gdyby jednak użyć jako funkcji pośredniczącej funkcji UDF, która tylko pobierałaby argumenty i przekazywałaby je w formie tablicy do dalszego przetworzenia:

```
Function My0(ParamArray x())  
    My0 = x  
End Function
```

to takie wywołania:

```
= ILOCZYN(My0(3; ; 7))      lub          = ILOCZYN(My0(3; pusta; 7))
```

zakończą się błędem #ARG!. Błąd dotyczy całego wywołania funkcji *Wy0*, a nie tylko pojedynczych elementów tablicy, a więc nie można go zamaskować funkcją JEŻELI.BŁĄD.



Signalizowany tu problem dotyczy tylko tablicy **ParamArray**, a nie argumentów opcjonalnych, poprzedzonych słowem **Optional**.

Aby wyeliminować ten problem, należy w treści funkcji testować argumenty funkcją **IsMissing**, aby wykryć pominięte argumenty i funkcją **IsEmpty**, aby znaleźć odwołania do pustych komórek. Zamiast testu **IsMissing** można użyć testu **IsError**; wówczas zamaskujemy nie tylko pominięte argumenty, ale również te, które zawierają błędy. Wartości te wystarczy zastąpić wartością **Empty**, aby funkcje agregujące i niektóre tablicowe prawidłowo interpretowały te wywołania. Oto przykładowy kod:

```
Function Wy(ParamArray x())
    Dim i As Long
    For i = 0 To UBound(x)
        If IsMissing(x(i)) Or IsEmpty(x(i)) Then x(i) = Empty
    Next i
    Wy = x
End Function
```

Tu ciekawostka. Pusta komórka jest interpretowana w kodzie funkcji jako wartość **Empty**, ale jest to odwołanie (**TypeName = Range**), jednak trzeba jeszcze raz przypisać jej wartość **Empty**, aby usunąć ślad po tym odwołaniu (**TypeName = Empty**).

## Problem tablic wśród argumentów ParamArray

Takie rozwiązanie będzie wystarczające, jeśli argumenty przekazywane do tablicy będą tylko pojedynczymi wartościami. Jeśli będziemy chcieli przekazać również tablice i zakresy, to taki kod nie zda egzaminu, bo struktura tablicy wynikowej będzie „niestrawna” dla funkcji zewnętrznych. W takim przypadku należy zastosować kod, który „wydobędzie” wszystkie elementy składowe tablicy argumentów i utworzy z nich tablicę jednowymiarową, pomijając „po drodze” elementy puste.

```
Function Wy_(ParamArray x())
    Dim i As Long, eL, eLw, y()
    For Each eL In x
        If Not (IsMissing(eL) Or IsEmpty(eL)) Then
            If IsArray(eL) Then
                For Each eLw In eL
                    i = i + 1
                    ReDim Preserve y(1 To i)
                    y(i) = eLw
                Next eLw
            Else
                i = i + 1
                ReDim Preserve y(1 To i)
                y(i) = eL
            End If
        End If
    End If
End Function
```

```

Next e1
Wy_ = y
End Function

```

Jeśli chcemy uniknąć powtarzania niemal identycznego fragmentu kodu dla tablic i wartości skalar-nych, można zastosować „sztuczkę” z nadaniem pojedynczej wartości struktury tablicy, a następnie wykonać kod dla tablic. Odpowiednia instrukcja będzie miała postać:

```
If Not IsArray(e1) Then e1 = Array(e1)
```

Należy uwzględnić, że powyższy kod pomija elementy puste, więc liczba elementów w tablicy wynikowej może być mniejsza od liczby elementów danych. Jeśli zachowanie identycznej liczby danych jest istotne, należy zmodyfikować kod tak, żeby zamiast pomijania elementów pustych były w ich miejscach umieszczane wartości **Empty**. Jeśli liczba argumentów jest duża, każdorazowe rozszerzanie tablicy wynikowej o jeden element wydłuża czas wykonania funkcji. W takim przypadku warto rozważyć rozszerzanie tablicy wynikowej blokami o kilkadziesiąt czy kilkaset pozycji. Wydłuży to kod, bo trzeba za każdym razem sprawdzać, czy w tablicy jest jeszcze miejsce na nową wartość, ale przyspieszy wykonanie. Na końcu należy „odciąć” niewykorzystany fragment tablicy.

## Deklaracja nagłówka funkcji o zmiennej liczbie parametrów

Nagłówek funkcji wykorzystującej tablicę parametrów może wyglądać na przykład tak:

```
Function Wielomian1(x As Double, ParamArray wsp()) As Double
```

Funkcja ta ma obliczać wartość wielomianu o współczynnikach określonych przez elementy tablicy *wsp* dla argumentu *x*. Wywołanie tej funkcji w VBA może mieć postać:

```
Wielomian1(3.5,1,3,4,-2)
```

a w arkuszu:

```
Wielomian1(3,5;1;3;4;-2)
```

Funkcja obliczy wartość wielomianu  $x^3+3x^2+4x-2$  dla  $x = 3,5$ . Zakładamy, że kod funkcji został tak napisany, że *wsp*(0) jest współczynnikiem wielomianu przy najwyższej potęgze *x*. Kod funkcji musi uwzględniać to, że liczba współczynników, określająca zarazem stopień wielomianu, może być dowolna.

Jeżeli współczynniki wielomianu są umieszczone w komórkach arkusza z zakresu C7:F7, to wywołanie z kodu VBA może mieć postać:

```
Wielomian1(3.5, Range("C7"), Range("D7"), Range("E7"), Range("F7"))
```

lub

```
Wielomian1(3.5, [C7], [D7], [E7], [F7])
```

a w arkuszu:

**Wielomian1**(3,5;C7;D7;E7;F7)

Gdybyśmy chcieli wywoływać tę funkcję, podając zakres zawierający współczynniki, nagłówek musiałby wyglądać inaczej, na przykład tak:

**Function Wielomian2**(*x* As Double, *wsp*) As Double

W tym przypadku **wsp** jest zmienną typu **Variant**, pod którą przy wywołaniu funkcji można podstawić zakres lub tablicę, tak jak to opisano w rozdziale 4., w punkcie „Tablice w zmiennych typu Variant”. Wywołanie tej funkcji w sytuacji, gdy współczynniki wielomianu są umieszczone w komórkach arkusza z zakresu C7:F7, ma postać:

w kodzie VBA:

**Wielomian2**(3.5, Range("C7:F7"))      lub      **Wielomian2**(3.5, [C7:F7])

a w arkuszu:

**Wielomian2**(3,5;C7:F7)

Gdyby przy tej drugiej postaci funkcji trzeba było w wywołaniu podać współczynniki w formie liczbowej, wywołanie to mogłoby wyglądać tak:

w kodzie VBA:

**Wielomian2**(3.5,Array(1,3,4,-2))

a w arkuszu:

**Wielomian2**(3,5;{1;3;4;-2})      lub      **Wielomian2**(3,5;{1\3\4\ -2})

Współczynniki wielomianu zostały umieszczone w stałej tablicowej o strukturze wiersza lub tablicy. Który zapis odpowiada której strukturze, to zależy od używanej wersji Excela. W wersji VBA użyto funkcji **Array** (opisanej w rozdziale 4., w punkcie „Funkcja Array”), która zwraca zmienną typu **Variant** o strukturze jednowymiarowej (wierszowej) tablicy złożonej z elementów wymienionych w wywołaniu. Każdy z tych zapisów powinien pozwolić na poprawne obliczenie wartości wielomianu, ale czy tak faktycznie będzie, to już zależy od treści funkcji. Ten aspekt omówiono w rozdziale 10., w podrozdziale „Obliczanie wartości wielomianu”.



Nazwa funkcji WIELOMIAN jest w Excelu 2010+ wykorzystywana jako nazwa predefiniowana. Zatem, jeśli zdefiniowalibyśmy własną funkcję o takiej nazwie, nie mogłaby być ona użyta jako funkcja UDF.

# Przekazywanie zmiennej liczby parametrów pomiędzy funkcjami

## Przekazywanie parametrów przez wartość

Należy pamiętać, że wewnątrz funkcji, która została zdefiniowana z tablicą parametrów **ParamArray** *wsp*(), tablica ta jest jednowymiarową tablicą wierszową, zawierającą odczytane argumenty. I o ile wewnątrz tej funkcji do odczytania wartości argumentów możemy użyć na przykład instrukcji pętli **For Each ... Next** (o różnych rodzajach pętli pisałem w rozdziale 3., w podrozdziale „Sterowanie wykonaniem kodu”) w postaci:

```
Dim element
For Each element In wsp
    instrukcje
Next element
```

to w przypadku, gdy całą tablicę chcemy przekazać jako argument do innej funkcji, na przykład:

```
Wynik = MojaFunkcja(wsp)
```

to jeżeli *MojaFunkcja* oczekuje na argument typu **Variant**, nie można przekazać tablicy *wsp* przez referencję (wystąpi błąd kompilacji: *Invalid ParamArray use*), a jedynie przez wartość. Można to zrobić kilkoma sposobami. Jeśli deklaracja *MojejFunkcji* przewiduje przekazanie argumentu przez referencję, należy najpierw skopiować tablicę *wsp* do innej zmiennej typu **Variant** i dopiero tę zmienną przekazać do *MojejFunkcji*.

```
Dim lista
lista = wsp
Wynik = MojaFunkcja(lista)
```

Można również utworzyć kopię przez wykorzystanie tablicy **Array**:

```
Wynik = MojaFunkcja(Array(wsp)(0))
```

Dwa powyższe sposoby odnoszą się też do sytuacji, gdy *MojaFunkcja* jest funkcją arkuszową typu agregującego, akceptującą dowolną liczbę argumentów, udostępnioną przez obiekt **Worksheet** → **Function** lub **Application**. W tym przypadku jednak wszystkie argumenty muszą reprezentować pojedyncze wartości.

Jeżeli możemy zadeklarować *MojaFunkcję* z parametrem przekazywanym przez wartość

```
Function MojaFunkcja(ByVal arg)
```

to można przekazać tablicę *wsp* wprost

```
Wynik = MojaFunkcja(wsp)
```

Jeżeli natomiast *MojaFunkcja* również została zadeklarowana z tablicą **ParamArray** *wspn*(), można przekazać jej bezpośrednio tablicę *wsp* jako całość, ale należy mieć świadomość, że zostanie ona potraktowana jako pierwszy (a ściślej zerowy) element nowej tablicy parametrów. Dlatego w treści *MojejFunkcji* należy umieścić kod:

```

Dim element, lista
lista = wspn(0)
For Each element In lista
    instrukcje
Next element

```

Jeśli w łańcuchu argumentów przekazywanych z funkcji do funkcji tablica parametrów występowała kilkakrotnie, operację „rozpakowywania” argumentów należy powtórzyć. Odpowiedni przykład podano w rozdziale 6., w punkcie „Funkcje składające dowolne dane w tablicę”.

Oczywiście, należy pamiętać, że jeśli na liście argumentów *wsp* są argumenty grupowe (na przykład zakresy obejmujące kilka komórek), to nie zostaną one automatycznie „rozpakowane”. Trzeba to zrobić odpowiednim kodem w treści funkcji przyjmującej dane, na przykład takim, jaki podałem wcześniej, w punkcie „Problem tablic wśród argumentów ParamArray”.

Najbardziej skomplikowany jest przypadek, kiedy funkcja zadeklarowana z tablicą **ParamArray** jest funkcją predefiniowaną i nie ma dostępu do jej wnętrza albo z innych powodów nie można jej modyfikować. Argumenty do takiej funkcji muszą być przekazywane pojedynczo. Mogą to być elementy tablicy, ale nie można przekazać tej tablicy hurtem. Problemem, który trzeba rozwiązać, jest nieznaną z góry liczba argumentów do przekazania. Musimy zdecydować, ile tych argumentów może być maksymalnie. To zależy od przeznaczenia funkcji i programista musi podjąć tę decyzję. Funkcje arkuszowe, wywoływane w VBA za pomocą obiektu **WorksheetFunction**, przewidują przekazywanie do 30 argumentów. Wydaje się, że jest to liczba wystarczająca do większości zastosowań, w wielu przypadkach może wystarczyć mniej. W treści funkcji zewnętrznej (tej, z której chcemy przekazać tablicę parametrów) musimy zadeklarować tablicę typu **Variant** o stałej liczbie elementów i przepisać do niej argumenty wprowadzone przy wywołaniu. Problemem są nadmiarowe elementy, których nie wykorzystujemy. W wyniku użycia instrukcji **ReDim Preserve** są one inicjowane wartością **Empty**. Jeśli takie dane zostaną przekazane do funkcji wewnętrznej, będą interpretowane tak, jak wartości zerowe. W niektórych przypadkach może to być akceptowalne (np. w funkcjach wykonujących operacje sumowania), ale w innych (np. w funkcji obliczającej średnią albo funkcji, która ma określoną liczbę argumentów) już nie. Problem można rozwiązać, nadając zbędnym elementom tablicy wartość **Null**. Wartości te będą ignorowane przez funkcję wewnętrzną, tak jakby ich nie było. Poniżej przedstawiam listing funkcji, która pozwala wywoływać dowolną funkcję z zestawu **WorksheetFunction** przez podanie tekstu jej angielskiej nazwy jako pierwszego argumentu, a następnie pozostałych niezbędnych argumentów:

```

Function Use(fname As String, ParamArray Data())
    Dim d() As Variant, n As Long ' konieczne nawiasy przy deklaracji d()
    d = Data
    ReDim Preserve d(29)
    For n = UBound(Data) + 1 To 29
        d(n) = Null
    Next n
    Use = CallByName(WorksheetFunction, fname, VbMethod, _
        d(0), d(1), d(2), d(3), d(4), d(5), _
        d(6), d(7), d(8), d(9), d(10), d(11), _
        d(12), d(13), d(14), d(15), d(16), d(17), _
        d(18), d(19), d(20), d(21), d(22), d(23), _
        d(24), d(25), d(26), d(27), d(28), d(29))
End Function

```



Po przekazaniu wszystkich argumentów funkcji poszerzamy tablicę do wymaganego rozmiaru, uzupełniamy brakujące elementy wartościami **Null** i przekazujemy wszystkie elementy tablicy do funkcji **CallByName**, która umożliwia wywołanie metod obiektów VBA przez tekst nazwy (angielskiej). Opis tej funkcji podano w rozdziale 9., w podrozdziale „Funkcja CallByName — wywoływanie metod przez tekst ich nazwy”. W wywołaniu funkcji **CallByName** można użyć obiektu **WorksheetFunction** lub obiektu **Application**, przy czym jeśli obiektem jest **WorksheetFunction**, a użyto argumentu, który nie jest obsługiwany w tym trybie, funkcja automatycznie przestawia się w tryb z kwalifikatorem **Application**. Różnice wyjaśniono w rozdziale 1., w punkcie „Użycie funkcji w wyrażeniach”.

Podobny sposób można zastosować w przypadku funkcji UDF współpracujących z funkcjami arkuszowymi, gdy chcemy użyć funkcji jako argumentu, a równocześnie chcemy, aby funkcja arkuszowa odczytała to jako pominięcie argumentu. W takim przypadku należy przekazać wartość **Null**. Przykładem może być funkcja arkuszowa ADRES. Ostatnim (opcjonalnym) jej argumentem jest nazwa arkusza. Jeśli ten argument zostanie podany, funkcja generuje tekst adresu kwalifikowanego, na przykład Arkusz2!B5. Jeśli argument zostanie pominięty, otrzymamy tekst adresu bez kwalifikatora, na przykład B5. Jeśli jako argument zostanie przekazany tekst pusty, funkcja wygeneruje tekst adresu w postaci !B5. Taka postać adresu jest dopuszczalna tylko przy tworzeniu nazw, natomiast użyta bezpośrednio w arkuszu spowoduje błąd. Aby tego błędu uniknąć, a równocześnie nie pisać wariantowych wywołań funkcji ADRES, można (w sytuacji, gdy nie chcemy adresu kwalifikowanego) nadać funkcji UDF wartość **Null**. Podobną metodę można zastosować w przypadku dwóch ostatnich argumentów funkcji PRZESUNIĘCIE i generalnie tych funkcji, które mają na końcu więcej niż jeden argument opcjonalny. Funkcje zwykłe, które mają tylko jeden taki argument, traktują wartość **Null** jak zero.

## Przekazywanie parametrów przez referencję

Omówione dotychczas sposoby przekazywania parametrów pomiędzy funkcjami sprowadzały się do przekazywania wartości. Mechanizm zastosowany w przypadku tablicy **ParamArray**, choć w zasadzie powinien działać przez referencję, zawiera jednak ograniczenia, które utrudniają napisanie kodu w taki sposób, aby funkcje zewnętrzne mogły zmodyfikować zawartość argumentów przekazywanych w tablicy **ParamArray**.

Należy również wziąć pod uwagę, czy argumentami przekazanymi poprzez tablicę **ParamArray** są tablice, czy zakresy komórek. Zaczniemy od analizy poniższych przykładów, które zilustrują możliwe przypadki.

### Przykład 5.1.

W tym przykładzie dane są odwołaniami. W pierwszym kodzie użyto zmiennej typu **Variant**.

```
Sub AddOne1 (ParamArray rg ())
    Dim e
    For Each e In rg
        e = e + 1
        Debug.Print e;
    Next e
```

```

    Debug.Print
    Debug.Print Join(rg, " ") 'wartości niezmienione
End Sub
Sub TestAddOne1()
    AddOne1 [A1], [B1], [C1]
End Sub

```

Ze względu na użycie zmiennej **e** typu **Variant**, wartości **e** modyfikowane są tylko lokalnie, oryginalne wartości w komórkach nie zmieniają się. W drugim kodzie występuje jawne odwołanie do wartości obiektów zakresów, dzięki temu wartości w komórkach zostały zmienione.

```

Sub AddOne2(ParamArray rg())
    Dim e 'musi być Variant
    For Each e In rg
        e.Value = e.Value + 1
        Debug.Print e;
    Next e
    Debug.Print
    Debug.Print Join(rg, "; ") 'wartości zmienione
End Sub
Sub TestAddOne2()
    AddOne2 [A31], [B31], [C31]
End Sub

```

## Przykład 5.2.

Kolejne dwa warianty kodu będą dotyczyły tablic jako argumentów.

```

Sub AddOne3(ParamArray tb())
    Dim e 'musi być Variant
    For Each e In tb
        e = e + 1
        Debug.Print e;
    Next e
    Debug.Print
    Debug.Print Join(tb, "; ") 'wartości niezmienione
End Sub
Sub TestAddOne3()
    Dim tabl
    tabl = Array(3, 7, 10)
    AddOne3 tabl(0), tabl(1), tabl(2)
    Debug.Print Join(tabl, "; ") 'wartości niezmienione
End Sub

```

W tym przypadku, podobnie jak w pierwszym wariantcie kodu w przykładzie 5.1, w zmiennej **e** tworzone są tylko lokalne kopie elementów tablicy; w efekcie oryginalna tablica pozostaje bez zmian.

```

Sub AddOne4(ParamArray tb())
    Dim i As Long
    For i = 0 To UBound(tb)
        tb(i) = tb(i) + 1
        Debug.Print tb(i);
    Next i
    Debug.Print
    Debug.Print Join(tb, "; ") 'wartości zmienione
End Sub

```

```

Sub TestAddOne4()
    Dim tabl
    tabl = Array(3, 7, 10)
    AddOne4 tabl(0), tabl(1), tabl(2)
    Debug.Print Join(tabl, "; ")      'wartości zmienione
End Sub

```

W czwartym wariantcie kodu odwołujemy się indywidualnie do poszczególnych elementów tablicy ze wskazaniem jej nazwy. W tym przypadku wywołanie przez referencję działa zgodnie z założeniem.

### Przykład 5.3.

Z bardziej złożonym przypadkiem będziemy mieć do czynienia, gdy chcemy wykorzystać uniwersalną procedurę pośredniczącą zadeklarowaną z tablicą **ParamArray**. Proszę przeanalizować poniższy przykładowy kod:

```

Sub ModyfikujTablice()
    'modyfikuje elementy tablic a(), b() i c()
    Dim a, b, c
    a = Array(12, 1)      'Array(12, 1, 3)
    b = Array(5, -7)
    c = Array(10, 8, -4)
    Modyfikuj a, b, c
    Debug.Print "a(): "; Join(a) 'pokaż zawartość tablic
    Debug.Print "b(): "; Join(b)
    Debug.Print "c(): "; Join(c)
End Sub

Sub Modyfikuj(ParamArray y())
    'uniwersalna procedura modyfikująca
    Dim i As Long, x
    x = y 'nie można bezpośrednio użyć y
    'podejmij działanie w zależności od liczby elementów w pierwszym argumencie y(0)
    If UBound(y(0)) > 1 Then
        Dodaj x, 1
    Else
        Dodaj x, 2
    End If
    'kopiuj wyniki do tablicy ParamArray
    ' y = x      'to nie działa na tablice zewnętrzne, tylko lokalnie
    For i = 0 To UBound(y)
        y(i) = x(i)
    Next i      'to działa na tablice zewnętrzne
End Sub

Sub Dodaj(y, k As Long)
    'inkrementuje o k wszystkie zagnieżdżone elementy y
    Dim i As Long, j As Long
    For i = LBound(y) To UBound(y)
        For j = LBound(y(i)) To UBound(y(i))
            y(i)(j) = y(i)(j) + k
        Next j, i
    Next i
End Sub

```

Źródłem problemu jest niemożność bezpośredniego przekazania przez referencję tablicy **ParamArray** w procedurze **Modyfikuj**. Przez podstawienie  $x = y$  tworzymy kopię tablicy  $y$ , ale już nie przez referencję, tylko przez wartość. Procedura **Dodaj** działa na tej kopii i wszelkie zmiany są wprowadzane do zmiennej  $x$ , a nie do  $y$ . Zwykle przypisanie zwrotne całej tablicy  $y = x$  znowu odbywałoby się przez wartość i tablica  $y$  miałyby nowe wartości tylko wewnątrz procedury **Modyfikuj**, a po jej opuszczeniu tablice  $a$ ,  $b$  i  $c$ , przekazane w tablicy **ParamArray**  $y$  miałyby wartości niezmienione. Obejście tego problemu polega na przypisaniu elementów tablicy  $x$  do  $y$  pojedynczo, w pętli. W tym przypadku nowe wartości  $x$  są kopiowane pod stare adresy tablic  $a$ ,  $b$  i  $c$ , czyli przez referencję.

## Funkcja zwracająca wynik w postaci tablicy

Zazwyczaj funkcja zwraca w wyniku jedną wartość, jednak nic nie stoi na przeszkodzie, aby napisać taką, która będzie zwracać tablicę wartości. W deklaracji takiej funkcji można nie podawać typu wyniku (zastosowany zostanie typ **Variant**) albo po nazwie typu wyniku dać parę pustych nawiasów, na przykład zapis **As Double()** oznacza, że wynik będzie tablicą złożoną z liczb typu **Double**. W kodzie funkcji należy tę tablicę utworzyć, a w końcu przypisać ją do nazwy funkcji. Jeśli typ funkcji nie jest zadeklarowany (lub jest zadeklarowany jako tablica elementów typu **Variant**), można też przypisać wynik do nazwy funkcji bezpośrednio, używając funkcji **Array** lub wyrażenia zwracającego wynik w postaci tablicy. Można modyfikować rozmiary wynikowej tablicy przez użycie instrukcji **ReDim** do nazwy funkcji, ale nie można nadawać wartości pojedynczo, gdyż każde wywołanie nazwy funkcji, po którym następują nawiasy, jest traktowane jak wywołanie rekurencyjne.

```
Sub TestArr1()  
    Debug.Print Join(Arr1)  
End Sub  
Function Arr1() As Variant()  
    Arr1 = Array(3.5, 1.5, 5, 3, 4)  
    ReDim Preserve Arr1(2) '0 To  
End Function
```

Aby wyświetlić wyniki takiej funkcji w arkuszu kalkulacyjnym, trzeba skonstruować odpowiednie wyrażenie tablicowe i zatwierdzić je klawiszami **Ctrl+Shift+Enter** lub odwoływać się do poszczególnych składowych wyniku za pośrednictwem funkcji arkusza **INDEKS**. Można również użyć wywołania takiej funkcji w wyrażeniu na analogicznych zasadach jak w przypadku funkcji standardowych. Pamiętać też należy, że jeśli funkcja daje w wyniku tablicę jednowymiarową, to wartości wynikowe będą tworzyły wiersz. Aby uzyskać kolumnę, trzeba umieścić wyniki w tablicy dwuwymiarowej, w której drugi indeks jest zawsze równy 1 albo użyć funkcji **Array** wraz z funkcją **WorksheetFunction.Transpose**. W tym drugim przypadku wynik musi być typu **Variant**.

Jeśli rozmiary tablicy zwracanej przez funkcję są określone w sposób sztywny w treści funkcji, użytkownik musi zadbać o zaznaczenie odpowiednio dużego zakresu na wynik. Jakie są konsekwencje niedopasowania rozmiarów zakresu i tablicy, opisałem w rozdziale 4., w punkcie „Wypełnianie zakresu zawartością tablicy”. Możliwe jest również napisanie funkcji UDF, która uwzględni wielkość zakresu zaznaczonego przez użytkownika. Odczytanie wielkości tego zakresu w treści

funkcji jest możliwe dzięki właściwości **Application.Caller**, o której napisałem więcej w rozdziale 8., w punkcie „Identyfikacja komórki, z której wywołano funkcję UDF”. Właściwość ta zwraca obiekt zakresu, z którego wywołano funkcję UDF. Jeśli jest to obiekt wielokomórkowy, można odczytać jego rozmiary poprzez odczyt właściwości **Application.Caller.Rows.Count** i **Application.Caller.Columns.Count**. Jeśli — przykładowo — funkcja ma zwrócić tablicę jednowymiarową, można sprawdzić w treści funkcji, czy użytkownik zaznaczył zakres wierszowy, czy kolumnowy i dostosować do tego orientację tablicy.

```
Function Wektor(Optional pocz As Long = 1, Optional kon)
    Dim Tabl(), i As Long, wk As Long
    Dim wiersze As Long, kolumny As Long
    wiersze = Application.Caller.Rows.Count
    kolumny = Application.Caller.Columns.Count
    wk = Application.Max(wiersze, kolumny)
    ReDim Tabl(1 To wk)
    If IsMissing(kon) Then
        kon = pocz + wk - 1
    Else
        wk = Application.Min(Abs(kon - pocz) + 1, wk)
    End If
    For i = 1 To wk
        Tabl(i) = pocz + Sgn(kon - pocz)*(i - 1)
    Next i
    For i = wk + 1 To UBound(Tabl)
        Tabl(i) = ""
    Next i
    If kolumny > wiersze Then
        Wektor = Tabl
    Else
        Wektor = Application.Transpose(Tabl)
    End If
End Function
```

Przykładowa funkcja wypełnia zaznaczony zakres komórek kolejnymi liczbami całkowitymi od argumentu **pocz** aż do wartości **kon** (jeśli została podana) lub wypełnienia wiersza lub kolumny, w zależności od zaznaczonego zakresu. Jeśli podano **kon < pocz**, komórki są wypełniane malejąco. Jeśli zaznaczono zbyt duży zakres, nadwyżkowe komórki są wypełniane tekstem pustym, a nie kodami błędu, jak w przypadku funkcji arkuszowych.

## Przeliczanie wartości funkcji

Gdy jest włączony tryb automatycznego przeliczania formuł w arkuszu (ustawienie domyślne), każda zmiana wartości w arkuszu wywołuje przeliczenie wszystkich komórek zawierających formuły zależne od komórek, które uległy zmianie. W przypadku funkcji arkuszowych sprawa nie budzi większych wątpliwości, gdyż wszystkie argumenty funkcji muszą być jawnie podane. Niektóre funkcje mają status ulotnych (są przeliczane przy każdym przeliczeniu arkusza). Inaczej rzecz się ma w przypadku funkcji UDF, których treść zapisano w VBA. Tu w treści funkcji mogą wystąpić odwołania do komórek lub nazw, które nie są wyraźnie wymienione na liście argumentów. W takiej sytuacji przy przeliczaniu arkusza Excel „nie wie”, czy taka funkcja powinna

być przeliczona. Jednym z możliwych rozwiązań tego problemu jest umieszczenie na początku kodu funkcji dyspozycji:

```
Application.Volatile
```

Spowoduje to przeliczenie funkcji przy każdym przeliczeniu arkusza. Jest to jednak rozwiązanie niezbyt efektywne, bo może powodować wielokrotne zbędne przeliczanie. Lepiej zadbać o to, by wszystkie argumenty były jawnie wskazane w nagłówku funkcji. Wtedy łatwo ustalić, kiedy funkcja musi być przeliczona.

Jeśli w treści funkcji UDF nie użyto polecenia **Volatile**, funkcja jest przeliczana „na życzenie” dopiero po naciśnięciu klawiszy *Ctrl+Alt+F9* lub po edycji komórki z tą funkcją. Jeżeli natomiast je zastosowano, do przeliczenia wystarczy użycie klawisza *F9* lub edycja dowolnej komórki.

Sprawa jawnej specyfikacji argumentów nabiera szczególnego znaczenia, gdy funkcja ma być użyta w obliczeniach iteracyjnych. Od wersji 2003 zasady przeliczania arkusza zmieniły się i wszystkie argumenty funkcji, które podlegają zmianie, powinny być jawnie wymienione, w przeciwnym razie nawet w przypadku użycia polecenia **Application.Volatile** funkcja jest przeliczana tylko jeden raz, a nie przy każdej iteracji.

Przykład takiej sytuacji omówiono w rozdziale 7., w podrozdziale „Metoda Evaluate”.

## Wywołanie makroinstrukcji z kodu VBA

To, co napisałem powyżej o funkcjach UDF, odnosi się też w dużym stopniu do makroinstrukcji. Zasadnicza różnica polega na tym, że makroinstrukcje mogą zmieniać stan arkusza (wpisywać dane, zmieniać format komórek itp.), a funkcje UDF, w zasadzie, nie mogą — mają tylko zwrócić wynik obliczeń do miejsca wywołania (o wyjątkach od tej reguły piszę w rozdziale 16.). Ponadto makroinstrukcje, które zawierają parametry wywołania, mogą być wywołane wyłącznie z poziomu kodu VBA. Z poziomu arkusza można wywoływać tylko makroinstrukcje bez parametrów. Nie dotyczy to natomiast tych, które są wywoływane automatycznie do obsługi zdarzeń, ale one mają stały zestaw parametrów.

Wywołanie makroinstrukcji z poziomu kodu VBA może być zrealizowane kilkoma sposobami: albo przez wpisanie nazwy makroinstrukcji, a następnie wartości argumentów wywołania (jeżeli występują), albo przez użycie słowa **Call** poprzedzającego nazwę makroinstrukcji, albo za pomocą metody **Run**. Jeżeli użyto słowa **Call**, argumenty wywołania makra muszą być umieszczone w nawiasach; jeżeli użyto tylko nazwy makra bez **Call**, należy też pominąć nawiasy. Słowo **Call** może być też formalnie zastosowane do wywołania funkcji, jednak wówczas nie można przekazać wyniku funkcji — takie wywołanie może mieć sens tylko w szczególnych przypadkach. Inny sposób wywołania makra to użycie konstrukcji:

```
Application.Run "nazwa_makra", arg_1, arg_2, ...
```

Nazwa makroinstrukcji jest podawana jako tekst w cudzysłowie, następnie podaje się argumenty wywołania oddzielone przecinkami. W analogiczny sposób można również wywołać funkcję:

```
wynik = Application.Run("nazwa_funkcji", arg_1, arg_2, ...)
```

Istotne jest, że za pomocą metody **Run** można wywoływać tylko funkcje zdefiniowane w kodzie VBA, a nie funkcje wewnętrzne Visual Basic.

Makroinstrukcje można wywoływać również pośrednio, za pomocą metody **Evaluate**, zarówno z kodu, jak i z arkusza. Wywołuje się je podobnie jak funkcje. Ponieważ makroinstrukcja nie zwraca wyniku, nie należy używać nawiasów otaczających tekstowy argument metody, natomiast nawiasy są potrzebne wewnątrz argumentu tekstowego, niezależnie od tego, czy makro ma być wywołane z argumentami, czy bez. (O użyciu tej metody piszę w rozdziale 7., w podrozdziale „Metoda Evaluate” i w rozdziale 16., w podrozdziale „Funkcje UDF wywoływane w sposób pośredni”.) Niestety, w tym przypadku obowiązują ograniczenia i makra wywoływane w ten sposób nie mogą wykonywać wszystkich zmian w arkuszu, ale jest to pewien sposób na obejście problemu przekazywania argumentów do makroinstrukcji wywoływanej z poziomu arkusza, gdyż tu można je przekazać podobnie jak do funkcji.

## Funkcje użytkownika podobne do funkcji standardowych

Korzystając z funkcji standardowych dostępnych w arkuszu kalkulacyjnym, nie zastanawiamy się zwykle nad tym, że funkcja, która na przykład wymaga podania argumentu w postaci liczby rzeczywistej, może w istocie przyjąć jako argument stałą liczbową, stałą tablicową, adres pojedynczej komórki, wyrażenie dające w wyniku liczbę, nazwy reprezentujące wymienione wcześniej rodzaje danych, ale również argument tekstowy, który ma postać liczby (liczbotekst) i zostanie skonwertowany na liczbę. Argument może także przybrać formę tablicy, jeżeli funkcja zostanie użyta w odwołaniu tablicowym. Ta tablica również może mieć postać stałej tablicowej, zakresu komórek, wyrażenia dającego w wyniku zakres komórek lub tablicę wartości oraz nazwy reprezentującej jedną z wymienionych wcześniej postaci. Dla przykładu można spróbować wywołać w arkuszu funkcję LN z różnego rodzaju argumentami:

- = LN(2)
- = LN({2})
- = LN("2")
- = LN(C3), przy założeniu, że w komórce C3 umieszczono liczbę 2
- = LN(G2+1), przy założeniu, że w komórce G2 umieszczono liczbę 1
- = LN(*arg*), przy założeniu, że nazwa *arg* została wcześniej zdefiniowana i reprezentuje jeden z typów danych wymienionych powyżej.

Można również użyć wywołania tablicowego — zaznaczywszy wcześniej trzy sąsiednie komórki (poziomo), trzeba wprowadzić do nich jedno z poniższych wyrażeń tablicowych i zatwierdzić przez **Ctrl+Shift+Enter**:

- = LN({2;5;8,5}) (w Excelu 2010+ wprowadzamy = LN({2\5\8,5}))
- = LN(C3:E3), przy założeniu, że w zakresie komórek C3:E3 umieszczono liczby, na przykład 2; 5 i 8,5

- = LN(C3:E3+1), przy założeniu, że w zakresie komórek C3:E3 umieszczono odpowiednie liczby
- = LN(*arg\_tabl*), przy założeniu, że nazwa *arg\_tabl* została wcześniej zdefiniowana i reprezentuje jeden z typów danych tablicowych wymienionych powyżej.

Niestety, nie wszystkie funkcje predefiniowane w Excelu akceptują tak różnorodny typ argumentów. Łatwo sprawdzić, że na przykład funkcje pochodzące z dodatku *Analysis ToolPak*, mimo że od wersji 2007 dodatek ten jest już zintegrowany z arkuszem, nie akceptują argumentów w postaci odwołań do zakresów wielokomórkowych.

Możemy to sprawdzić na przykładzie jednej z funkcji z tego dodatku: SQRTPI (od wersji 2010 występuje pod spolszczoną nazwą PIERW.PI). Funkcja ta oblicza pierwiastek z argumentu pomnożonego przez liczbę  $\pi$ . W razie użycia jej z argumentem w postaci tablicy funkcja może zwracać kod błędu #ARG!. Konkretny wynik zależy od wersji Excela i postaci argumentu. Przykładowo, Excel 2000 i 2003 nie akceptuje żadnych argumentów tablicowych, wersje 2007+ akceptują stałe tablicowe, ale nie przyjmują zakresów.

Jeżeli piszemy własną funkcję w VBA i chcemy, aby zachowywała się podobnie jak funkcje standardowe, musimy sami zadbać o to, aby funkcja akceptowała różne rodzaje danych. Jeżeli typ argumentu zostanie zadeklarowany jako **Single**, **Double** lub typ całkowitoliczbowy, w wywołaniu funkcji może wystąpić dowolny rodzaj danych, odnoszący się do pojedynczej wartości (z wyjątkiem stałej tablicowej). Jeżeli typ argumentu zostanie zadeklarowany jako **Object** lub **Range**, w wywołaniu funkcji może wystąpić tylko adres komórki lub nazwa tej komórki. Dotyczy to sytuacji, gdy w treści funkcji argument jest traktowany jak pojedyncza wartość.

Jeżeli chcemy, aby własna funkcja mogła być używana w odwołaniu tablicowym, akceptowała dane tablicowe i zwracała wynik w postaci tablicy, musimy w treści funkcji sprawdzić, jaki rodzaj danych reprezentuje argument i stosownie do tego zorganizować obliczenia. Załóżmy, że mamy przykładową funkcję SQRTPI, która pobiera jedną liczbę jako argument i zwraca jedną liczbę jako wynik. Chcemy zmodyfikować ją tak, aby mogła akceptować również argument w postaci tablicy i zwracać wynik w postaci tablicy o takich samych rozmiarach.

Taka funkcja musi przyjmować argument typu **Variant** i sama zwracać wynik tego samego typu. W treści funkcji należy sprawdzić, czy argument jest pojedynczą liczbą, czy tablicą — w tym drugim przypadku trzeba jeszcze odróżnić zakres komórek od zwykłej tablicy. Do sprawdzenia, czy mamy do czynienia z tablicą, służy funkcja **IsArray**, natomiast za pomocą **IsObject** sprawdzimy, czy mamy do czynienia z zakresem. W różny sposób sprawdza się rozmiary zwykłej tablicy (służą do tego funkcje **LBound** i **UBound**) i zakresu (właściwości **Rows.Count** i **Columns.Count**). Osobna obsługa zakresów i tablic jest uzasadniona, jeśli funkcja ma odwoływać się do innych niż wartość właściwości zakresu. W tym przypadku potrzebna jest tylko wartość, więc można skopiować wartość zakresu do tablicy, stosując zwykłe przypisanie  $y = x$ . Jeżeli  $y$  jest zmienną typu **Variant**, a  $x$  zakresem, to taka operacja spowoduje, że  $y$  będzie przechowywać wartości zawarte w zakresie  $x$ , nie będąc obiektem.



Musimy jeszcze sprawdzić, czy argument tablicowy jest jednowymiarowy, czy dwuwymiarowy — od tego zależy sposób odwoływania się do jego elementów. Jeśli pierwotnym argumentem był zakres, to tablica będzie zawsze obiektem dwuwymiarowym, natomiast jeśli pierwotnym argumentem było wyrażenie zwracające tablicę lub stała tablicowa, to tablica zawierająca jeden wiersz jest obiektem jednowymiarowym, a inne tablice (w tym jednokolumnowa) mają strukturę dwuwymiarową.

Nie ma możliwości bezpośredniego sprawdzenia, czy tablica jest jednowymiarowa, czy dwuwymiarowa. Można to zrobić pośrednio, próbując odczytać drugi wymiar tablicy i sprawdzając, czy wystąpi błąd, czy nie. Metodę tę wykorzystano za pośrednictwem zmiennej *TwoDim*, która przyjmuje wartość logiczną **True**, jeśli sprawdzany argument jest tablicą dwuwymiarową.

Korzystnie będzie oddzielić część ogólną procedury, która będzie wspólna dla różnych funkcji, od definicji samej funkcji poddawanej „opakowaniu”. Aby to było możliwe, nazwa funkcji będzie przekazywana do dalszej obróbki w postaci tekstowej i po podstawieniu pojedynczej wartości argumentu wartość funkcji będzie obliczana za pomocą metody **Evaluate** (por. podrozdział „Metoda Evaluate” na początku rozdziału 7. i podrozdział „Funkcje UDF wywoływane w sposób pośredni” w rozdziale 16.).

Ponieważ funkcja, którą chcemy zmodyfikować, pochodzi z dodatku *Analysis ToolPak*, sposób odwołania do niej zależy od wersji arkusza, w której chcemy uruchomić funkcję. Jak już pisałem w rozdziale 1., w podrozdziale „Wyrażenia”, w wersjach wcześniejszych — do 2003 — można nazwę funkcji poprzedzić kwalifikatorem [*atpvbaen.xls*], oznaczającym nazwę pliku biblioteki funkcji, którą trzeba wcześniej zainstalować; można również użyć samej nazwy funkcji. Od wersji 2007 nazwę funkcji przy bezpośrednim wywołaniu należy poprzedzić słowem **WorksheetFunction** lub **Application**, jednak w omawianym przykładzie, gdzie wywołanie jest pośrednie, a funkcja w postaci tekstowej jest obliczana za pomocą metody **Evaluate**, wystarczy sama nazwa funkcji.

Całość można zapisać w następujący sposób:

```
Function MySqrtPi(x)
    MySqrtPi = MyFunction("SqrtPi(x)", x) ' albo [atpvbaen.xls].
End Function

Function FVal(FName As String, x)
    If IsError(x) Then
        FVal = x
    Else
        If IsNumeric(x) Then x = Str(x) Else x = Chr(34) & x & Chr(34)
        FVal = Application.ThisCell.Parent.Evaluate(Replace(FName, "x", x))
    End If
End Function

Function MyFunction(FName As String, x)
    Dim i, j, y, TwoDim As Boolean
    If IsArray(x) Then
        y = x ' kopia x jako tablica
        TwoDim = False
        On Error Resume Next
```

```
TwoDim = UBound(y, 2) > 0
On Error GoTo 0
For i = LBound(y) To UBound(y)
    If TwoDim Then
        For j = LBound(y, 2) To UBound(y, 2)
            y(i, j) = FVal(FName, x(i, j))
        Next j
    Else : y(i) = FVal(FName, x(i))
    End If
Next i
MyFunction = y
Else
    MyFunction = FVal(FName, x)
End If
End Function
```

Po zaproponowanej modyfikacji funkcje użytkownika będą działać bardzo podobnie do funkcji predefiniowanych Excela, z jedną różnicą. Gdy w zwykłym wywołaniu argumentem funkcji jest zakres, przypisanie faktycznego argumentu odbywa się na zasadzie rzutu prostokątnego (dokładniej o tym piszę w [A7]). Po zaproponowanej przeze mnie modyfikacji, w takim przypadku faktycznym argumentem będzie pierwsza wartość z zakresu, tak jakby użyto formuły tablicowej. Natomiast gdy argumentem jest wyrażenie, na przykład **zakres\*1**, obowiązuje zasada rzutu prostokątnego, gdyż Excel wyznacza faktyczną wartość argumentu zanim zostanie on przekazany do funkcji.



# Skorowidz

## A

absolutna wielkość komórki, 485  
ActiveCell, 65, 72  
ADO, ActiveX Data Objects, 467  
AdvancedFilter, 86  
aktualizacja automatyczna, 232  
Analysis ToolPak, 56  
API, Application Programming Interface, 498  
Application, 65, 66, 217  
argumenty, 29  
    funkcji  
        O.APLIKACJI, 551  
        O.DOKUMENCIE, 559  
        O.KOMÓRCE, 567  
        O.OBIEKCIE, 574  
        O.SKOROSZYCIE, 582  
        OBLICZ, 188  
        Split, 123  
    puste, 147  
arkusz  
    wymiana informacji z kodem, 65  
arkusze makr, 529, 530  
    przeliczanie formuł, 532  
arkuszowe stałe tablicowe, 204  
AutoFilter, 82, 86  
AutoFilterMode, 85  
automatyzacja przeliczania, 443  
awaryjne przerwanie, 103

## B

bezpieczeństwo użycia makr XLM, 534  
biblioteka Microsoft Script Control, 196  
błąd, 209, 216  
    #ADR!, 36  
    #LICZBA!, 47

#NAZWA?, 436

    Type mismatch, 47, 48  
breakpoint, 26  
ByRef, 143  
ByVal, 143

## C

Call, 158  
Caller, 219  
całkowanie numeryczne, 282  
Cells, 69  
Chart, 57  
cofanie zmian, 228  
Collection, 335  
Comment, 264  
Comments, 267  
Count, 62  
CurrentRegion, 82  
czas, 351  
    ciągłe wyświetlanie, 369, 371  
    dokładny pomiar, 366  
    niestandardowe minuty, 374  
    systemowy, 375  
czyszczenie danych, 177

## D

dane grupowe, 188  
data, 351, 352  
    jako tekst, 354  
    rozpoznawanie, 353  
    systemowa, 375  
DataObject, 494, 497  
Debug, 27, 218

definiowanie  
 funkcji, 37  
 nazw, 434  
 podprogramów, 31  
 reguł formatowania warunkowego, 419

deklarowanie, 39  
 kolekcji, 335  
 nagłówka funkcji, 149  
 słownika, 341  
 stałych, 43  
 tablic, 117  
 typu, 39  
 zmiennych, 30  
 zmiennych typu Variant, 123

Dictionary, 342

DisplayStatusBar, 111

dodatek Analysis ToolPak, 56

dokładność zwiększona obliczeń, 303

dokładny pomiar czasu, 366

dostęp  
 do funkcji makr XLM, 530  
 do stałych predefiniowanych, 258

dyrektywa Option Explicit, 41

dyrektywy kompilacji warunkowej, 97

dzielenie całkowite, 51

## E

edytor VBA, 24

EnableEvents, 88

etykieta, 103, 507, 511

expression, 71

## F

filtr  
 odczyt ustawień, 85  
 wykluczający, 84  
 zaawansowany, 86

Find, 76

folder AddIns, 34

formant  
 „lupa”, 517  
 ActiveX, 504, 510, 522, 525  
 etykieta, 511  
 obraz, 515  
 pasek przewijania, 514, 526  
 pole kombi, 514  
 pole listy, 512

pole tekstowe, 513

pole wyboru, 512

przycisk opcji, 512

przycisk pokrętła, 513

przycisk polecenia, 511

przycisk przełącznika, 516

formularza, 504–507, 521, 525

etykieta, 507

identyfikacja, 522

pasek przewijania, 509

pokrętło, 508

pole grupy, 510

pole kombi, 509

pole listy, 508

pole wyboru, 507

przycisk, 507

przycisk opcji, 507

znacznik wyboru, 525

## formanty

dodawanie, 504

makroinstrukcje, 524

zamienniki, 517

## format

.xls, 34

.xlsm, 34

## formaty komórek, 272

w VBA, 272

## FormatConditions, 412

## formatowanie, 274

fragmentu tekstu, 458

komórki z tekstem, 332

stałe, 425

tekstu, 328

warunkowe, 238, 399, 412  
 kopiowanie, 425  
 przeliczanie reguł, 418  
 selektywne kopiowanie reguł, 428

wzorów chemicznych, 323, 325

## Formuła, 58

## formuły, 207, 237

## funkcja, 28, 29, 37

ACTIVE.CELL, 549

ADR.POŚR, 191

ADR.TEKST, 547

ADRES, 476

Aggregate, 55

Array, 121, 126

Asc, 378

## funkcja

- AscW, 378
- CallByName, 153, 254, 404
- CDate, 352
- CDbl, 183, 481
- Choose, 95
- Chr, 377
- ChrW (\$), 377
- CopyNestedDict, 349
- CStr, 481
- CVErr, 216
- Date, 356
- DateAdd, 357
- DateDiff, 359
- DatePart, 358
- DateSerial, 356
- DateValue, 352, 356
- DOCUMENTS, 548
- DOKUMENTY, 548
- Eval, 196
- EVALUATE, 586
- FILES, 586
- Filter, 387
- Format (\$), 380
- Format, 273, 360, 482
- FormatDateTime, 359
- FORMULA.CONVERT, 549
- FORMUŁA.TRYB.ADR, 549
- GET.CELL, 567
- GET.DEF, 558
- GET.DOCUMENT, 559
- GET.FORMULA, 566
- GET.NAME, 572
- GET.NOTE, 574
- GET.OBJECT, 574
- GET.WORKBOOK, 582
- GET.WORKSPACE, 551
- GetTickCount, 368
- Hex, 378
- HIPERŁĄCZE, 446, 447
- IIf, 94
- INDEKS, 173
- Index, 132
- INDIRECT, 435
- InputBox, 114, 164
- InStr, 383
- InStrRev, 383
- IntCpy, 445
- Intersect, 224
- IsDate, 356
- IsEmpty, 49
- IsMissing, 146
- IsNumeric, 55, 375
- JEŻELI.BŁĄD, 171, 217, 444
- JEŻELI.ND, 217
- Join, 170
- Join, 388, 390
- KOM.AKT, 549
- KOMÓRKA, 476, 483
- LCase, 378
- Left, 378
- Len, 379
- LICZ.JEŻELI, 193
- LICZBA, 248
- LTrim, 377
- MACIERZ.ILOCZYN, 285
- Mid (\$), 379
- Mid, 379
- MOD, 51
- Modif, 194
- MonthName, 364
- MouseOver, 449
- MsgBox, 107, 108, 109
- NAMES, 550
- NAZWY, 550
- Now, 356
- O.APLIKACJI, 551
- O.DEFINICJI, 558
- O.DOKUMENCIE, 559
- O.FORMULE, 566
- O.KOMÓRCE, 567
- O.NAZWIE, 572
- O.NOTATCE, 574
- O.OBIEKCIE, 574
- O.OPCJACH.LISTY, 582
- O.SKOROSZYCIE, 582
- OBLICZ, 182, 188–191
- Oblicz, 440
- Oct (\$), 378
- OKNA, 585
- OnTime, 365
- OPTIONS.LISTS.GET, 582
- PLIKI, 586
- QBColor, 402
- Rank, 55
- REFTEXT, 547

- Replace, 183, 386, 479
- Right (\$), 378
- RTrim, 377
- SELECTION, 588
- Space, 380
- Split, 122, 170
- Str, 192, 481
- StrComp, 384
- String (\$), 380
- StrReverse, 386
- Subtotal, 55
- Sum\_If, 194
- SUMA.JEŻELI, 193
- SUMA.WARUNKÓW, 285
- Switch, 95
- SZACUJ, 586
- Time, 356
- Timer, 364
- TimeSerial, 356
- TimeValue, 356
- TRANSPONUJ, 171, 188
- Transpose, 129, 516
- Trim (\$), 377
- TypeName, 42
- UCase (\$), 378
- UDF do konwersji dat, 355
- UDF, 220, 237
- Val, 247, 481
- VarType, 42
- Wait, 365
- WeekdayName, 364
- WINDOWS, 585
- WorksheetFunction.Substitute, 480
- WYBIERZ, 241
- WYSZUKAJ.PIONOWO, 453
- ZAZNACZENIE, 588
- funkcje, 28, 29, 37
  - arkuszowe, 53, 55
  - do formatowania, 274, 275
  - do operacji na tekstach, 377
  - generujące tablice, 131
  - informacyjne, 53
  - kwalifikowane wywołanie, 186
  - makr, 529
    - Excela 4.0, 547
    - XML, 530, 534, 538, 547
  - matematyczne, 52
  - nieciągłe, 291
  - o zmiennej liczbie parametrów, 149
  - parametry opcjonalne, 145
  - polskie nazwy, 189
  - przekazywanie zmiennej liczby parametrów, 151
  - przeliczanie wartości, 157
  - tekstowe, 52, 53
  - użytkownika, UDF, 24, 29, 143, 159, 247
    - definiowanie nazw, 434
    - definiowanie reguł formatowania warunkowego, 419
    - długość tekstu, 455
    - edycja komentarzy, 269
    - formatowanie fragmentu tekstu, 458
    - formatowanie komórek, 439
    - kopiowanie komórek, 451
    - łączenie sformatowanych tekstów, 460
    - modyfikacja parametrów wykresu, 431
    - monitorowanie zmian w komórkach, 465
    - możliwości, 429
    - nadawanie nazwy zakresowi, 439
    - obiekty graficzne, 523
    - obsługa zdarzeń, 450
    - odczyt ze skoroszytu, 466
    - ograniczenia, 429
    - przeliczanie formuł, 484
    - schowek systemowy, 501
    - uruchamianie, 446
    - wspomagane procedurami obsługi zdarzeń, 424
    - wstawianie komentarzy, 269
    - wynik sformatowany, 453
    - wyszukiwanie, 453
    - wywoływane pośrednio, 440, 422, 432
    - wywoływane bezpośrednio, 420, 429
    - zmiana zawartości komórki, 441, 443
  - walidujące, 217
  - wewnętrzne, 55
  - zwracające wynik w postaci tablicy, 156

## G

- generowanie
  - kombinacji, 300
  - liczb pseudolosowych, 294
  - permutacji, 295
  - permutacji z powtórzeniami, 297
- graficzne elementy sterujące, *Patrz* formanty
- grupowanie arkuszy, 178

**H**

hierarchia obiektów, 64  
hiperłącze  
użycie funkcji UDF, 447

**I**

identyfikacja formantu formularza, 522  
iloczyn  
cyfr w liczbie, 382  
elementów tablicy, 285  
informacje od użytkownika, 112  
inicjowanie zmiennych, 50  
instrukcja  
For Each ... Next, 286  
GoSub ... Return, 104  
GoTo, 104  
If ... Then ... Else, 91  
LSet, 383  
On Error, 174, 209, 471  
On Error GoTo, 212  
On Error Resume Next, 167, 214  
Option Compare, 30  
Option Explicit, 30  
Print, 27  
Resume, 210  
RSet, 383  
Select Case, 94, 106, 215  
SendKeys, 328, 329, 332  
With, 66, 106  
With Selection, 493  
instrukcje  
pętli, 98, 105  
pętli programowej, 91  
przypisania, 44  
skoku, 91, 103–106  
warunkowe, 91  
wyboru, 91, 94 106, 215

**J**

jawne odwołanie, 79

**K**

kasowanie tła komórki, 407  
klasa CommandBars, 409  
klucze, 345

kody

błędów, 215  
dwubajtowe, 326

kolekcja, 335

Comments, 267  
FormatConditions, 412  
Names, 546

kolekcje

a słowniki, 345  
a tablice, 338  
deklarowanie, 335  
jako argumenty, 339  
odczyt elementów, 337  
ograniczenia, 340  
tworzenie, 335  
usuwanie elementów, 335

kolory, 399

czcionki, 405, 441  
odczyt, 402  
oznaczanie komórek, 403, 406  
reakcja na zmiany, 409  
rozjaśnianie, 404  
sumowanie komórek, 405  
symulacja skali barw, 427  
system RGB, 399  
ściemnianie, 404  
tła, 402

kombinacje, 300

kombinatoryka, 295

komentarz, 38

dodawanie, 262  
edycja, 262, 269  
w komórce, 262  
wstawianie, 269  
znaczniki, 266

komórki

aktywne, 65, 476  
kasowanie tła, 407  
kolor czcionki, 441  
monitorowanie zmian w, 465  
ochrona, 491  
określanie formatu, 272  
powiększenie, 492  
scalone, 270  
ukrywanie zawartości, 293  
ustalanie absolutnej wielkości, 485  
usuwanie zawartości, 443  
wybrane, 65  
zmiana zawartości, 441



kompilacja warunkowa, 96  
 dyrektywy, 97  
 stałe, 96

komunikacja programu, 107

komunikaty MsgBox, 28

kontrola poprawności danych, 209, 233

kontrolki, *Patrz* formanty

konwersja  
 dat, 355  
 na tekst, 48  
 stylu liczb, 477

kopiowanie  
 elementów, 169  
 formatowania warunkowego, 425  
 graficzne, 518  
 komórek, 451  
 reguł formatowania warunkowego, 428  
 słownika, 348  
 zakresu nieciągłego, 75

kwifikator  
 Application, 55  
 arkusza, 67  
 ThisWorkbook, 436  
 WorksheetFunction, 54, 55

kwifikowane wywołanie funkcji, 186

## L

liczby  
 obliczanie iloczynu cyfr, 382  
 pseudolosowe, 294

lista poprawności danych, 237, 238

listy  
 rozwijane, 236, 241–445, 449  
 ochrona, 243  
 ograniczenia źródła, 236  
 symulowane, 411  
 wprowadzanie danych, 244  
 wyświetlanie, 241  
 zależne, 240

## Ł

łączenie  
 danych tekstowych, 48  
 tekstów, 388  
 tekstów sformatowanych, 321, 460

## M

makra XLM, 529, 538, 547  
 bezpieczeństwo użycia, 534

makro, 28  
 rejestracja, 32  
 uruchamianie, 34

makroinstrukcje, *Patrz* podprogram

makropolecenie, *Patrz* makro

masa molowa, 309

Menedżer nazw, 252, 437

metoda, 57, 62  
 Add, 234, 437  
 AddComment, 263  
 AdvancedFilter, 87  
 Application.ExecuteExcel4Macro, 533  
 AutoFilter, 83  
 AutoFit, 488  
 ClearComments, 264  
 ClearNotes, 264  
 ConvertFormula, 253  
 Delete, 320  
 Err.Raise, 213  
 Evaluate, 98, 135, 137, 181, 184, 186, 191–195,  
 206, 432, 437  
 Evaluate zagnieżdżona, 187  
 FillAcrossSheets, 179  
 Find, 76, 77, 166  
 GetFormat, 497  
 GoalSeek, 280  
 InputBox, 28, 112, 240, 249  
 Merge, 271  
 Modify, 234  
 Names.Add, 198, 203, 435  
 NoteText, 262  
 OnKey, 489  
 Parse, 397  
 PopUp, 109  
 Range.Replace, 477  
 Range.TextToColumns, 478  
 Rungego-Kutty, 287  
 SpecialCells, 267, 268, 269  
 Text, 264  
 TextToColumns, 394  
 UnMerge, 271

metody, 57, 62  
 obiektu Range, 63

Międzynarodowy arkusz makr, 531

moduł

- objektowy, 29
- standardowy, 29
- ThisWorkbook, 67

modyfikacja

- danych, 88
- obiektów, 319
- parametrów wykresu, 431
- zdefiniowanych nazw, 203

monitorowanie zmian w komórkach, 465

MsgBox, 28

## N

Names, 198

narzędzie „Aparat fotograficzny”, 518

nazwy, 35

- arkuszowe, 181, 199, 202, 204, 206, 472
- definiowanie, 434
- formuł, 207
- stałych, 207
- zakresu, 201, 439

notatka tekstowa, 262

NumberFormat, 59

## O

obiekt, 57

- „Pole tekstowe”, 517
- Application, 217
- Characters, 324
- Comment, 264, 270
  - metoda Text, 264
  - właściwość Author, 266
- DataObject, 494
- Debug, 218
- Err, 212
- Names, 198
- Range, 57
- Range.Validation, 233, 449
- zakresu, 79

obiekty, 57

- graficzne, 520
- hierarchia, 64
- modyfikacja wyglądu, 319
- modyfikacja zawartości, 319
- zmiennie, 72

obliczanie

- masy molowej, 309
- pierwiastków równania kwadratowego, 279
- szeregów, 285
- wartości wielomianu, 278
- iteracyjne, 190

obraz, 515

obsługa

- wyjątków, 209
- zdarzeń, 221, 424

ochrona komórek, 491

odczyt

- danych ze skoroszytu, 466
- elementów kolekcji, 337
- kodu formatu, 272
- koloru tła, 402
- nazwy, 260
- nazwy zakresu, 201
- numeru arkusza, 260
- wartości stałych tablicowych, 134

odwołania

- do kolumn, 74
- do sąsiednich komórek, 72
- do wartości zakresu, 79, 80
  - przez adres, 68
- do wierszy, 74
- do zakresów wielokomórkowych, 534
- pośrednie, 70
- zewnętrzne, 205, 469

Offset, 75

okienko komunikatów, 107

okno Immediate, 27

określanie typu zmiennej, 41

operacja modulo, 51

operacje

- na danych, 351
  - na datach, 356
- operator, 44, 46
- &, 48
  - dodawania, 48
  - Like, 392
  - łączenia tekstów, 388, 389
  - Not, 128

operatory relacji, 47

opróżnianie schowka, 500

Optional, 145

oznaczanie komórek, 403

**P**

ParamArray, 148  
 parametr, 29  
     conversion  
         wartości, 385  
     format  
         dopuszczalne elementy, 362  
         wartości, 361  
 parametry  
     metody Add, 234  
     metody Modify, 234  
 pasek  
     narzędzi Debug, 27  
     przewijania, 509, 514, 526  
     stanu, 110  
 permutacje, 295  
     z powtórzeniami, 297  
 pętla  
     Do ... Loop, 102, 106  
     For ... Next, 98, 105  
     For Each, 249  
     For Each ... Next, 100, 105  
     While ... Wend, 103, 106  
 planowanie serii, 301  
 podprogramy, 24, 31  
     przypisywane do formantów, 524  
     typu funkcji, *Function*, 28  
     typu makropolecień, *Sub*, 28, 31  
     wywołanie, 158  
     zastosowania w chemii, 309  
 podział tekstu, 394  
     na kolumny, 395  
 pokrętko, 508  
 pole  
     grupy, 510  
     kombi, 509, 514  
     listy, 508, 512, 516  
     tekstowe, 513, 517, 521  
     wyboru, 507, 512  
 polecenia opcji, 29  
 polecenie, *Patrz* instrukcja  
 porównywanie tekstów, 392  
 procedura  
     GetSystemTime, 367, 368  
     OnUpdate, 410  
 procedury obsługi zdarzeń, 221, 424, 450  
 przechowywanie wartości, 224, 226, 227

przekazywanie argumentów  
     do formuł nazwanych, 252, 536  
     przez odwołanie, 143  
     przez referencję, 153  
     przez wartość, 143, 151  
 przeliczanie  
     formuł, 484, 532  
     stężeń roztworów, 312  
 przełączanie między wynikami, 489  
 przestrzeń nazw ukryta, 546  
 przycisk, 507  
     opcji, 507, 512  
     pokrętkła, 513  
     polecenia, 511  
     przełącznika, 516  
 pułapka, 26

**R**

Range, 57  
     metoda, 62  
         AddComment, 263  
         ClearComments, 264  
         ClearNotes, 264  
         Find, 76  
         SpecialCells, 267  
     odwołanie do zakresu, 62  
     właściwość  
         Comment, 263  
         Count, 62  
         Formuła, 58  
         Name, 201  
         NumberFormat, 59  
         Text, 60  
         Value, 57, 374  
 reguły formatowania warunkowego, 418  
 Rejestrator Makropolecień, 32  
 relacja równości, 47  
 RGB, 399  
 rozpoznawanie dat, 353, 374  
 roztwory  
     przeliczanie stężeń, 312  
 rozwiązywanie równań  
     nieliniowych, 280  
     różniczkowych, 287  
 równania  
     kwadratowe, 279  
     nieliniowe, 280  
     różniczkowe, 287

**S**

schowek systemowy, 494  
 opróżnianie, 500  
 użycie, 494  
 użycie funkcji UDF, 501  
 zapis, 498

Selection, 66, 164

serie, 301

skoroszyt  
 arkusze makr, 530  
 makr osobistych, 34  
 nadmierna objętość, 473  
 odczyt danych, 466

skrót klawiaturowy, 330  
 Alt+F11, 33  
 Ctrl+Alt+F9, 539  
 Ctrl+End, 474  
 Ctrl+F11, 530  
 Ctrl+F8, 27  
 Ctrl+Home, 333  
 Ctrl+n, 33  
 Shift+F10, 178

słowniki, 335, 341  
 a kolekcje, 345  
 deklarowanie, 341  
 kopiowanie, 348  
 modyfikacja, 343  
 odczyt, 343  
 tworzenie, 341  
 zastosowania, 346

słowo kluczowe  
 As, 39  
 Const, 30, 43  
 Dim, 30, 39  
 End Sub, 31  
 Function, 28  
 New, 340  
 Print, 27  
 Private, 30  
 Public, 30  
 Sub, 28, 31  
 WorksheetFunction, 53

sortowanie danych, 175

stałe, 43  
 globalne, 30  
 kompilacji warunkowej, 96  
 predefiniowane, 258  
 tablicowe, 134, 135, 204

sterowanie przebiegiem kompilacji, 96

stoper sekundowy, 369

stosowanie  
 komentarzy, 38  
 nazw, 35

struktura  
 modułów, 29  
 obiektów, 64

struktury danych  
 kolekcje, 335  
 słowniki, 335, 341

suma cyfr w liczbie, 382

sumowanie komórek, 405

symulacja skali barw, 427

system  
 kontroli błędów, 209  
 RGB, 399

szacowanie formuły, 189, 307

**Ś**

śledzenie wykonania kodu, 218

**T**

tablica TDW, 189

tablice, 117  
 bazowe, 131  
 dwuwymiarowe, 163  
 dynamiczne, 118, 124  
 jako argumenty, 130  
 jednowymiarowe, 129, 163  
 kasowanie zawartości, 127  
 kopiowanie elementów, 169  
 modyfikacja pierwszego wymiaru, 129  
 nadawanie wartości elementom, 125  
 operator Not, 128  
 parametrów, 147  
 porównywanie, 139  
 przypisywane do zakresu, 132  
 sortowanie danych, 175  
 statyczne, 117  
 ustalenie orientacji, 172  
 usuwanie wiersza, 173  
 zamiana na stałą tablicową, 138  
 zmiana struktury, 129

teksty  
 formatowanie fragmentów, 458  
 funkcje, 377

jako data, 354  
 łączenie, 321, 388  
 podział w kolumnie, 394  
 porównywanie, 392  
 puste, 49  
 rzeczywista długość, 455  
 treść definicji nazwy, 435  
 testowanie kodu, 26  
 Text, 60  
 tworzenie  
   kolekcji, 335  
   list rozwijanych, 445  
   nazw arkuszowych, 199  
   odwołań pośrednich, 70  
   słownika, 341  
 typ Variant, 41, 119, 123  
 typy, 39  
   komórek, 268  
   walidacji, 235  
   zmiennych, 40

## U

UDF, User-Defined Functions, 24  
 ukryta  
   kolumna, 483  
   przestrzeń nazw, 546  
   zawartość komórek, 293  
 unia zakresów, 73  
 UniCode, 326  
 uruchamianie  
   kodu, 26  
   makropoleceń, 34  
 usuwanie zawartości komórek, 443  
 użycie makr XLM, 530, 538

## V

VBA, Visual Basic for Applications, 23  
 VBE, Visual Basic Editor, 24

## W

walidacja, 235  
 wartość pusta Empty, 49  
 wersje Excela, 98, 414, 417  
 wielkość absolutna komórki, 485  
 wielomian, 278

właściwość, 57  
 Address, 220  
 Application.Caller, 187  
 Author, 266  
 Caller, 219, 220  
 Cells, 69  
 Characters, 319, 325  
 CodeName, 260  
 Color, 402  
 ColorIndex, 402  
 ColumnWidth, 485  
 Comment, 263  
 CompareMode, 345, 346  
 Count, 62  
 DirectPrecedents, 308  
 DisplayCommentIndicator, 267  
 DisplayStatusBar, 111  
 EnableEvents, 88  
 Formula, 58, 189, 471  
 FormulaLocal, 189, 442  
 ID, 229  
 Left, 242  
 MaxChange, 281  
 MergeArea, 271, 272  
 MergeCells, 271, 272  
 Name, 201, 260  
 NumberFormat, 59  
 NumberFormatLocal, 272  
 obiektu Validation, 233  
 Offset, 71, 75, 282  
 OnDoubleClick, 231  
 OnEntry, 231  
 Precedents, 308  
 RefersTo, 203  
 RefersToLocal, 205  
 RefersToR1C1, 203  
 RefersToRange, 202  
 Resize, 71  
 RowHeight, 485  
 Rows, 271  
 Selection, 164  
 Text, 60  
 ThisCell, 220  
 TintAndShade, 404  
 Value, 57, 207, 345, 374  
 Width, 242, 483  
 włączanie iteracji, 222  
 Workbook, 57

Worksheet, 57  
 WorksheetFunction, 152, 153  
 Worksheets, 179  
 wprowadzanie danych, 244  
 wykresy  
   funkcji, 288  
   funkcji nieciągłych, 289, 291  
   modyfikacja parametrów, 431  
 wykrywanie nieciągłości, 291  
 wyrażenia, 44  
 wyrażenie expression, 71  
 wyszukiwanie  
   adresu komórki, 76  
   danych, 166  
   w tablicy, 168  
   w zakresie, 166  
 wyświetlanie  
   komentarzy, 266  
   komunikatów, 110  
   listy rozwijanej, 241  
   tekstów formuł, 261  
 wywołanie  
   funkcji makr XLM, 533  
   funkcji UDF, 220  
   makroinstrukcji, 158  
   pośrednie, 440, 442  
   rekurencyjne, 32  
   metod, 254  
 wyznaczenie różnicy zakresów, 475  
 wzory chemiczne  
   formatowanie, 323, 325

## Z

zagnieżdżanie metody Evaluate, 135, 187  
 zakres, 68  
   indeksów tablicy, 119  
   nazwa, 439  
   nieciągły, 75  
   odczyt nazwy, 201

scalony, 270  
 selekcji, 476  
 wielokomórkowy, 534  
 wskazany myszką, 223  
 wyznaczenie różnicy, 475  
 zamiana na stałą tablicową, 135  
 złożony, 73, 81  
 zaokrąglanie liczb  
   cyfry parzyste, 278  
   cyfry znaczące, 277  
 zapis  
   do schowka, 498  
   kodu formatu, 272  
 zaznaczenie komórki aktywnej, 476  
 zdarzenia, 209, 221  
   funkcje UDF, 450  
   procedury, 450  
 zdarzenie  
   BeforeDoubleClick, 219  
   BeforeRightClick, 293, 493  
   Calculate, 222, 458  
   Change, 219, 224  
   OnUpdate, 409  
   SelectionChange, 406, 411, 418  
 zmiana kolorów, 409  
 zmienne, 39  
   globalne, 30, 224–227  
   obiektywne, 72, 133  
   typu Variant, 119, 123  
 znacznik  
   wyboru, 525  
   komentarza, 266  
 znak  
   apostrofu, 50  
   dwukropka, 35  
   dzielnej, 51  
   dzielnika, 51  
   podkreślenia, 35, 54  
   równości, 44  
 znaki specjalne, 329

# PROGRAM PARTNERSKI

— GRUPY 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  
**Helion** 



## Programowanie i makra w Excelu? Nic strasznego!

- Poznaj Visual Basic for Applications (VBA)
- Naucz się tworzyć makra i własne funkcje
- Zrozum Excela

Na rynku nie brakuje książek opisujących obsługę i zastosowania arkusza kalkulacyjnego MS Excel, żadna jednak nie wprowadzi Cię w tę tematykę tak skutecznie jak ta! Omiń rafa i białe plamy dokumentacji, skorzystaj z doświadczenia autora i śmiało wkrocz w świat niesamowitych możliwości Excela.

Poznaj konstrukcje języka VBA i naucz się przeprowadzać obliczenia za jego pomocą. Odkryj zastosowania formantów, zapanuj nad danymi opisującymi datę i czas, dowiedz się, jak radzić sobie z tekstami, wykorzystaj funkcje definiowane przez użytkownika. Przekonaj się też, do czego mogą Ci się przydać makra.

- Struktura i konstrukcje VBA
- Obiekty, właściwości i metody
- Instrukcje warunkowe i komunikacja
- Definiowanie i używanie tablic oraz funkcji
- Obsługa wyjątków i zdarzeń
- Zaawansowane struktury danych
- Operacje na datach, czasach i tekstach
- Operowanie kolorami i formatowanie warunkowe
- Graficzne elementy sterujące
- Funkcje makr programu Excel

**Dowiedz się, jak wykorzystać Excela do zautomatyzowania swojej pracy!**

	<i>Sprawdź nasze szkolenia!</i>	<b>KOD KORZYŚCI</b> Sięgnij po więcej ►	
 <b>helion.pl</b>		ISBN 978-83-283-5694-8	
 <b>HELION SA</b> ul. Kościuszki 1c 44-100 Gilwice tel.: 32 230 98 63 helion@helion.pl	<b>AKADEMIA IT &amp; BUSINESS</b>		
<b>WWW.SZKOLENIA.HELION.PL</b>		9 788328 356948	
<b>INFORMATYKA W NAJLEPSZYM WYDANIU</b>		Cena: 99,00 zł	