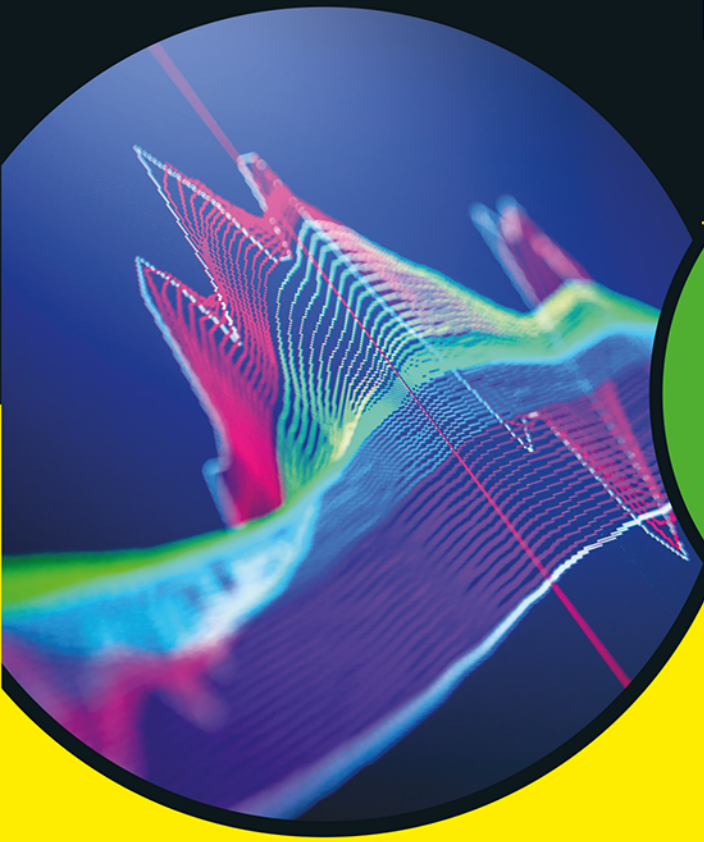
W PROSTOCIE TKWI SIŁA

wydanie V

Excel Programowanie w VBA

dla bystrzaków



Programuj w VBA
i rozszerzaj możliwości
Excela 2013, 2016 i 2019

Twórz własne aplikacje
i dodatki do programu Excel

Pracuj efektywniej
dzięki językowi VBA

Michael Alexander
John Walkenbach

septem
septem.pl
Helion

Tytuł oryginału: Excel VBA Programming For Dummies, 5th Edition

Tłumaczenie: Grzegorz Kowalczyk z wykorzystaniem fragmentów książki "Excel 2013 PL. Programowanie w VBA dla bystrzaków" w przekładzie Ryszarda Górnowicza i Andrzeja Watraka

ISBN: 978-83-283-6511-7

Drogi Czytelniku!
Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres
http://dlabystrzakow.pl/user/opinie/epvbb5
Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Pliki z przykładami omawianymi w książce można znaleźć pod adresem:
ftp://ftp.helion.pl/przyklady/epvbb5.zip

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

Printed in Poland.

Spis treści

W TYM ROZDZIALE:

- dowiesz się, kiedy, dlaczego i jak używać komentarzy w kodzie,
- nauczysz się korzystać ze zmiennych i stałych,
- poznasz sposób określania typu zmiennych VBA, których używasz,
- zapoznasz się z tablicami,
- dowiesz się, do czego czasami mogą być przydatne etykiety w procedurach.

Rozdział 7

Kluczowe elementy języka VBA

Jako że VBA jest prawdziwym, żywym językiem programowania, posiada wiele elementów charakterystycznych dla wszystkich języków programowania. W tym rozdziale poznasz niektóre takie elementy, czyli komentarze, zmienne, stałe, typy danych, tablice i inne dobrodziejstwa VBA. Jeżeli programowałeś już w innych językach, niektóre części materiału mogą wyglądać znajomo. Jeżeli w programowaniu jesteś żółtodziobem, czas zakasać rękawy i wziąć się do roboty.

Stosowanie komentarzy w kodzie VBA

Komentarz jest najprostszym rodzajem instrukcji języka VBA. Komentarze mogą zawierać dowolną treść, ponieważ VBA całkowicie je ignoruje. Komentarze możesz wstawiać, aby przypomnieć sobie później, dlaczego coś zrobiłeś, lub objaśnić jakieś szczególnie sprytne instrukcje, które napisałeś.

Stosuj często komentarze z dokładnymi opisami tego, co robi dany kod. Nie zawsze jest to oczywiste po przeczytaniu samych instrukcji. Często to, co dziś jest doskonale zrozumiałe, jutro może przysporzyć Ci bólu głowy. Już ja coś o tym wiem.

Komentarze rozpoczyna się od znaku apostrofu (`'`). VBA ignoruje wszelki tekst, jaki się znajdzie za tym znakiem w tym samym wierszu kodu. Komentarz może zajmować cały wiersz, ale równie dobrze może być umieszczony pod koniec wiersza kodu. W poniższym przykładzie demonstruję procedurę VBA z czterema komentarzami.

```
Sub FormatCells()
'   Jeśli nie jest zaznaczony zakres – zakończ działanie
    If TypeName(Selection) <> "Range" Then
       MsgBox "Zaznacz zakres."
       Exit Sub
    End If
'   Formatowanie komórek
    With Selection
       .HorizontalAlignment = xlRight
       .WrapText = False    ' bez zawijania tekstu
       .MergeCells = False ' bez łączenia komórek
    End With
End Sub
```

Apostrof oznacza, że jest to komentarz, niemniej jednak istnieje jeden wyjątek od tej reguły: apostrof znajdujący się wewnątrz frazy ujętej w znaki cudzysłowu nie jest interpretowany jako oznaczenie komentarza. Tak więc w poniższej instrukcji, mimo że zawiera znak apostrofu, nie ma komentarza.

```
Msg = "Can't continue!"
```

W trakcie pisania kodu może się zdarzyć, że będziesz chciał przetestować procedurę, *wyłączając* chwilowo pewną instrukcję lub grupę instrukcji. *Mógłbyś* — oczywiście — usunąć te instrukcje i później wprowadzić je ponownie, ale to strata czasu. Lepszym rozwiązaniem jest dodanie apostrofów tak, by zamienić te instrukcje w komentarze. Wykonując procedurę, VBA zignoruje wszelkie instrukcje zaczynające się od apostrofu. Aby reaktywować „zakomentowane" instrukcje, wystarczy usunąć apostrofy.

Oto szybki sposób na przekształcenie bloku instrukcji w komentarze. W edytorze VBE wybierz polecenie *View/Toolbars/Edit*, aby wyświetlić pasek narzędzi *Edit*. Następnie zaznacz wybrany blok instrukcji i naciśnij przycisk *Comment Block*. Aby usunąć apostrofy, zaznacz instrukcje i naciśnij przycisk *Uncomment Block*.

Po pewnym czasie każdy programista wykształca swój własny styl komentowania. Aby jednak komentarze były użyteczne, powinny zawierać informacje, które nie są oczywiste już po przeczytaniu samych instrukcji.

Poniższe wskazówki mogą Ci podpowiedzieć, jak efektywnie posługiwać się komentarzami.

- Identyfikuj się jako autor. Może się to przydać, gdy dostaniesz awans i ktoś, kto przejmie Twoje stanowisko, będzie mieć pytania.
- Opisz zwięźle cel każdej procedury Sub lub funkcji, jaką napiszesz.

- Używaj komentarzy, by śledzić zmiany, jakie wprowadzasz do procedury.
- Stosuj komentarze do oznaczania niestandardowych lub rzadkich zastosowań funkcji i konstrukcji.
- Opisuj w komentarzach przeznaczenie zastosowanych zmiennych szczególnie wtedy, kiedy ich nazwy są mało znaczące.
- Oznaczaj komentarzem wszelkiego rodzaju obejścia problemów w Excelu, jakie wypracujesz.
- Komentarze pisz równolegle z tworzonym kodem, nie zostawiaj tego zadania na koniec.
- W zależności od atmosfery panującej w pracy, możesz pomyśleć o umieszczeniu w komentarzach kilku zabawnych uwag. Osoba, która przejmie Twoją pracę, gdy już awansujesz, na pewno doceni poczucie humoru.

Używanie zmiennych, stałych i typów danych

Głównym zadaniem języka VBA jest przetwarzanie danych. VBA przechowuje dane w pamięci komputera, a ostatecznie mogą one (ale nie muszą) zostać zapisane na dysku twardym. Niektóre dane, takie jak zakresy komórek, przechowywane są w obiektach, podczas gdy nośnikami innych danych mogą być zmienne, które tworzysz.

Pojęcie zmiennej

Zmienna to po prostu posiadająca nazwę lokalizacja w pamięci komputera, używana przez program do przechowywania danych. Zmienne możesz nazywać w niemal dowolny sposób, więc powinieneś nadawać im możliwie jak najbardziej opisowe nazwy. Wartości przypisujemy do zmiennych za pomocą operatora znaku równości (więcej informacji na ten temat znajdziesz w podrozdziale „Instrukcje przypisania").

Oto kilka przykładów przypisywania wartości do zmiennych. Zauważ, że w ostatnim przykładzie użyte zostały dwie zmienne.

```
x = 1
InterestRate = 0.075
LoanPayoffAmount = 243089
DataEntered = False
x = x + 1
UserName = "Jan Nowak"
Date_Started = #3/14/2013#
MyNum = YourNum * 1.25
```

Nazwy zmiennych w języku VBA muszą spełniać kilka wymogów.

- Możesz używać liter, cyfr i niektórych znaków przestankowych, przy czym pierwszym znakiem musi być litera.
- VBA nie rozróżnia wielkości liter.
- W nazwach zmiennych nie możesz używać spacji, kropek i operatorów matematycznych.
- Nazwy zmiennych nie mogą zawierać następujących znaków: *#*, *$*, *%*, *&*, *!*.
- Nazwy zmiennych nie mogą być dłuższe niż 255 znaków. W praktyce jednak nikt nawet nie zbliża się do osiągnięcia tego limitu.

Aby nazwy zmiennych były czytelniejsze, programiści zazwyczaj stosują kombinacje małych i wielkich liter (na przykład `InterestRate`) lub znaki podkreślenia (`interest_rate`).

W języku VBA istnieje wiele zastrzeżonych słów kluczowych, których nie można używać jako nazw zmiennych lub procedur. Do słów kluczowych należą między innymi `Sub`, `Dim`, `With`, `End`, `Next` i `For`. Próba użycia któregoś z nich jako zmiennej może zakończyć się błędem kompilacji (w efekcie procedury nie da się uruchomić); jeżeli próba wykonania instrukcji przypisania spowoduje pojawienie się błędu, dokładnie sprawdź składnię tej instrukcji i upewnij się, że nazwa zmiennej nie jest zarezerwowanym słowem kluczowym. Aby to szybko zrobić, zaznacz wybraną nazwę i naciśnij klawisz *F1*. Jeżeli nazwa zmiennej jest słowem zastrzeżonym, będzie ona miała wpis w systemie Pomocy.

Język VBA pozwala tworzyć zmienne o nazwach pokrywających się z nazwami występującymi w modelu obiektowym Excela, takimi jak `Workbook` czy też `Range`. Takie praktyki z oczywistych powodów mogą się jednak przyczyniać do częstszych pomyłek. Oto przykład w pełni poprawnego, ale jednocześnie bardzo mylącego makra, w którym zadeklarowano `Range` jako nazwę zmiennej oraz wykonano operacje na komórce o nazwie `Range` i arkuszu noszącym również nazwę `Range`.

```
Sub RangeConfusion()
   Dim Range As Double
   Range = Sheets("Range").Range("Range").Value
   MsgBox Range
End Sub
```

Spróbuj więc oprzeć się pokusie deklarowania zmiennych o nazwach `Workbook` lub `Range`. Zamiast tego możesz użyć czegoś w stylu `MyWorkbook` lub `MyRange`.

Czym są typy danych w języku VBA?

Gdy mówimy o językach programowania, określenie *typ danych* odnosi się do sposobu, w jaki program przechowuje dane w pamięci komputera — na przykład jako liczby całkowite, liczby rzeczywiste czy ciągi znaków. Chociaż VBA potrafi uporać się z takimi szczegółami automatycznie, nie odbywa się to bez ponoszenia kosztów (w końcu nie istnieje przecież nic takiego jak bezpłatny lunch!). Pozwalając VBA dynamicznie przydzielać typy danych, godzisz się na wolniejsze wykonywanie programu i mało wydajne wykorzystanie zasobów pamięci. W przypadku niewielkich aplikacji nie stanowi to z reguły większego problemu. Projektując jednak duże lub złożone aplikacje, które mogą działać wolno lub wymagać optymalnego wykorzystania każdego wolnego bajta pamięci, musisz się dobrze orientować w typach danych.

VBA automatycznie określa typy danych, co ułatwia życie programistom. Nie wszystkie języki programowania zapewniają taki luksus. Niektóre języki programowania są *ściśle typowane*, co wymusza na programiście jawne deklarowanie typu danych każdej zmiennej, jakiej użyje.

VBA nie narzuca konieczności deklarowania używanych zmiennych, niemniej jednak jest to zdecydowanie dobra praktyka. Dalej w tym rozdziale dowiesz się, dlaczego tak jest.

VBA posiada szeroki wachlarz predefiniowanych typów danych. Tabela 7.1 stanowi zestawienie najczęściej używanych typów danych obsługiwanych w VBA.

TABELA 7.1. Wbudowane typy danych języka VBA

Typ danych	Rozmiar w bajtach	Zakres wartości
`Byte`	1	Od 0 do 255
`Boolean`	2	`True` lub `False`
`Integer`	2	Od –32 768 do 32 767
`Long`	4	Od –2 147 483 648 do 2 147 483 647
`Single`	4	Od –3,40E38 do –1,40E-45 dla wartości ujemnych; od 1,40E-45 do 3,40E38 dla wartości dodatnich
`Double`	8	Od –1,79E308 do –4,94E-324 dla wartości ujemnych; od 4,94E-324 do 1,79E308 dla wartości dodatnich
`Currency`	8	Od –922 337 203 685 477 do 922 337 203 685 477
`Date`	8	Od 1 stycznia 100 roku do 31 grudnia 9999 roku
`Object`	4	Dowolne odwołanie do obiektu
`String`	1 na każdy znak	Zmienny
`Variant`	zmienny	Zmienny

Generalnie rzecz ujmując, najlepiej stosować takie typy danych, które zajmują jak najmniejszą liczbę bajtów, a jednocześnie mieszczą się w nich wszystkie dane, które mają być przechowywane w danych zmiennych.

Wyjątkiem od „reguły najmniejszej liczby bajtów" jest typ `Long`. Większość programistów używa typu `Long` w miejsce typu `Integer`, gdyż pozwala to nieco poprawić wydajność programu. W niewielkich aplikacjach różnice pomiędzy typami `Long` a `Integer` są jednak praktycznie niezauważalne.

Deklarowanie zmiennych i określanie ich zasięgu

Jeżeli przeczytałeś poprzednie podrozdziały, dysponujesz już pewną wiedzą na temat zmiennych i ich typów. W tej sekcji dowiesz się, jak zadeklarować zmienną o określonym typie danych.

Jeżeli jawnie nie zadeklarujesz typu zmiennej używanej w instrukcjach VBA, domyślnie przyjęty zostanie typ `Variant`. Dane typu `Variant` są jak kameleon — zmieniają swój typ w zależności od tego, co z nimi zrobisz. Jeżeli zmienna typu `Variant` zawiera tekst, który wygląda jak liczba (na przykład `"143"`), można jej używać zarówno w operacjach na ciągach znaków, jak również w obliczeniach matematycznych — VBA automatycznie konwertuje dane z jednego typu na drugi. Pozostawienie językowi VBA odpowiedzialności za obsługę typów danych może się wydawać dużym ułatwieniem — pamiętaj jednak, że odbywa się to kosztem zmniejszonej prędkości działania i zwiększonej zajętości pamięci.

Deklarowanie zmiennych przed użyciem w procedurze jest praktyką ze wszech miar godną polecenia. Deklarowanie oznacza wskazanie typu danych dla poszczególnych zmiennych, co znakomicie przyspiesza szybkość działania makra oraz zapewnia bardziej wydajne wykorzystanie pamięci. Typ `Variant` (domyślny typ danych) wymaga od VBA ciągłych operacji porównywania typów danych i rezerwowania większych zasobów pamięci, niż to naprawdę konieczne. Kiedy określimy typ zmiennej, VBA nie musi go dodatkowo sprawdzać i może zarezerwować dla zmiennej tylko tyle pamięci, ile jest naprawdę potrzebne.

Aby zmusić się do deklarowania zmiennych w swoich programach, powinieneś umieścić poniższe dwa słowa kluczowe na początku modułu VBA:

```
Option Explicit
```

Gdy kod modułu rozpoczyna się od takiej instrukcji, każda próba uruchomienia procedury zawierającej niezadeklarowane zmienne zakończy się niepowodzeniem.

Instrukcji `Option Explicit` powinieneś użyć tylko raz — na początku modułu, przed deklaracjami jakichkolwiek procedur. Nie zapominaj, że opcja `Option Explicit` obowiązuje tylko w obrębie modułu, w którym się znajduje. Jeżeli w projekcie znajduje się więcej niż jeden moduł, instrukcję `Option Explicit` powinieneś umieścić w każdym z nich.

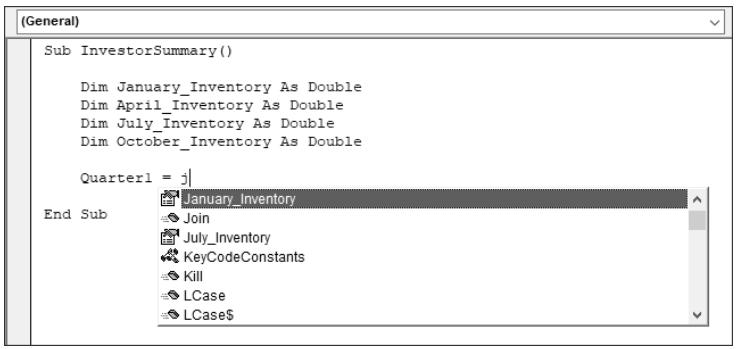
Załóżmy, że używasz niezadeklarowanej zmiennej (czyli zmiennej typu `Variant`) o nazwie `CurrentRate`. W pewnym miejscu zamieszczasz w swoim programie następującą instrukcję:

```
CurentRate = .075
```

Nazwa zmiennej zawiera literówkę (brakuje jednej litery *r*), co może być trudne do zauważenia. Jeżeli nie zwrócisz na ten błąd uwagi, Excel zinterpretuje to jako użycie innej, nowej zmiennej, rezultatem czego będą najprawdopodobniej błędne wyniki działania programu. Jeżeli na początku modułu umieścisz polecenie `Option Explicit`, wymuszając jednocześnie zadeklarowanie zmiennej `CurrentRate`, Excel wygeneruje błąd, jeżeli napotka niepoprawnie zapisaną nazwę takiej zmiennej.

Aby instrukcja `Option Explicit` była wstawiana automatycznie na początku każdego nowo dodanego modułu VBA, wystarczy włączyć opcję *Require Variable Definition* (wymuszaj deklarację zmiennych). Znajdziesz ją na karcie *Editor* (edytor) okna dialogowego *Options* (opcje — użyj polecenia *Tools/Options* w VBA). Szczerze radzę, byś tak zrobił.

Jeśli zadeklarujesz zmienne, będziesz mógł również skorzystać z ułatwienia, dzięki któremu zaoszczędzisz sobie nieco pisania. Wprowadź po prostu pierwsze dwie lub trzy litery nazwy zmiennej, a następnie naciśnij *Ctrl+Spacja*. Jeżeli wybór jest jednoznaczny, VBE uzupełni wpis za Ciebie. Jeżeli nie, wyświetlona zostanie lista pasujących słów, z której będziesz mógł wybrać odpowiednie. Dodam jeszcze, że funkcjonalność ta działa również w przypadku słów kluczowych i nazw funkcji. Na rysunku 7.1 przedstawiam przykład takiej automatycznej listy.



RYSUNEK 7.1. Naciśnięcie klawiszy Ctrl+spacja powoduje wyświetlenie listy nazw zmiennych, funkcji i słów kluczowych

Znasz już zalety deklarowania zmiennych, ale nadal nie wiesz, *jak* to się robi. Najczęściej stosowanym sposobem jest użycie instrukcji ze słowem kluczowym `Dim`. Oto kilka przykładów deklarowania zmiennych.

```
Dim YourName As String
Dim January_Inventory As Double
Dim AmountDue As Double
Dim RowNumber As Long
Dim X
```

Pierwsze cztery zmienne zostały zadeklarowane jako określony typ danych. Ostatnia zmienna, `X`, nie została zadeklarowana jako określony typ danych, jest więc traktowana jako `Variant` (może być dowolnego typu).

Poza `Dim` w VBA występują jeszcze trzy inne słowa kluczowe używane przy deklarowaniu zmiennych:

- `Static`,
- `Public`,
- `Private`.

Więcej szczegółowych informacji na temat słów kluczowych `Dim`, `Static`, `Public` i `Private` znajdziesz w dalszej części tego rozdziału, ale najpierw muszę omówić dwa inne, bardzo ważne tematy, czyli zasięg zmiennych i czas życia zmiennych.

Skoroszyt może zawierać dowolną liczbę modułów VBA, a każdy moduł VBA może zawierać dowolną liczbę procedur `Sub` i `Function`. *Zasięg zmiennej* określa, które moduły i procedury mogą używać danej zmiennej. Szczegółowe informacje znajdziesz w tabeli 7.2.

TABELA 7.2. Rodzaje zasięgu zmiennych

Zasięg	Sposób deklarowania zmiennej
Tylko dana procedura	Przy użyciu instrukcji `Dim` lub `Static` w obrębie procedury, w której używana jest zmienna.
Tylko dany moduł	Przy użyciu instrukcji `Dim` lub `Private` przed pierwszą instrukcją `Sub` lub `Function` w module.
Wszystkie procedury we wszystkich modułach	Przy użyciu instrukcji `Public` przed pierwszą instrukcją `Sub` lub `Function` w module.

Pogubiłeś się? Czytaj dalej, a znajdziesz kilka przykładów, dzięki którym wszystko stanie się jasne.

Zmienne o zasięgu jednej procedury

Najniższym poziomem zasięgu zmiennej jest poziom procedury. (Procedurami są procedury `Sub` i `Fuction`). Zmienne o tak zadeklarowanym zasięgu, inaczej **zmienne lokalne**, mogą być używane tylko w obrębie procedury, w której zostały zadeklarowane. Gdy kończy się procedura, zmienna przestaje istnieć (zostanie unicestwiona), a Excel zwalnia zajmowaną przez nią pamięć. Gdy teraz ponownie uruchamiasz procedurę, zmienna powraca do życia, ale jej poprzednia wartość została utracona.

Standardowym sposobem deklarowania zmiennych lokalnych jest użycie instrukcji `Dim`. Słowo `Dim` (*ciemny*) nie odnosi się do zdolności intelektualnych twórców VBA, a raczej jest to stary termin programistyczny będący skrótem od *dimension* (*rozmiar*) i oznacza po prostu rezerwowanie miejsca w pamięci dla danej zmiennej.

Poniższy przykład zawiera kilka zmiennych o zasięgu jednej procedury, zadeklarowanych przy użyciu słowa kluczowego `Dim`.

```
Sub MySub()
   Dim x As Integer
   Dim First As Long
   Dim InterestRate As Single
   Dim TodaysDate As Date
   Dim UserName As String
   Dim MyValue
'   ... [Tu znajduje się kod procedury] ...
End Sub
```

Zauważ, że w powyższym przykładzie ostatnia deklaracja `Dim` nie określa typu zmiennej `MyValue` — deklarowana jest jedynie sama zmienna. W rezultacie typem danych zmiennej `MyValue` jest `Variant`.

W przeciwieństwie do niektórych języków VBA nie pozwala na deklarowanie grupy zmiennych jako określonego typu danych poprzez oddzielenie kolejnych nazw zmiennych przecinkami. Choć przykładowo poniższa instrukcja jest całkowicie poprawna, *nie deklaruje* wszystkich wymienionych zmiennych jako `Integer`.

```
Dim i, j, k As Integer
```

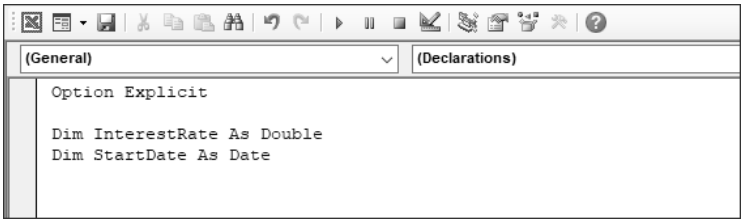
W przykładzie tym jedynie zmienna `k` zostanie zadeklarowana jako `Integer`, podczas gdy pozostałym zmiennym przydzielony zostanie domyślny typ danych (`Variant`).

Gdy zadeklarujesz zmienną o zasięgu jednej procedury, w pozostałych procedurach tego samego modułu możesz używać zmiennych o identycznej nazwie, lecz każda z tych instancji zmiennej będzie unikatowa w obrębie procedury, w której została zadeklarowana. Ogólnie rzecz ujmując, zmienne deklarowane na poziomie procedury są najbardziej wydajne, ponieważ VBA zwalnia zajmowaną przez nie pamięć w momencie, gdy procedura kończy działanie.

Zmienne o zasięgu jednego modułu

Czasami możesz potrzebować zmiennej, która byłaby dostępna z poziomu wszystkich procedur należących do modułu. W takim przypadku wystarczy zadeklarować zmienną (za pomocą `Dim` lub `Private`) przed pierwszym wystąpieniem instrukcji `Sub` lub `Function` — poza wszelkimi procedurami. Robi się to w sekcji *Declarations* (deklaracje), znajdującej się na początku modułu. (W sekcji tej umieszcza się również instrukcję `Option Explicit`).

Na rysunku 7.2 pokazujemy, jak możesz się upewnić, że znajdujesz się w sekcji *Declarations*. Aby to zrobić, użyj listy rozwijanej znajdującej się po prawej stronie i przejdź bezpośrednio do sekcji *Declarations* (to nie *Eurobusiness*, nie przechodzisz przez *Start* i nie zgarniasz 200 dolarów).



RYSUNEK 7.2. Każdy moduł VBA zawiera sekcję Declarations, która znajduje się przed instrukcjami wszelkich procedur Sub i Function

Dla przykładu załóżmy, że chcesz tak zadeklarować zmienną `CurrentValue`, aby była dostępna we wszystkich procedurach danego modułu. W tym przypadku wystarczy umieścić instrukcję `Dim` w sekcji *Declarations*:

```
Dim CurrentValue As Double
```

Gdy zamieścisz taką deklarację (robiąc to jednocześnie we właściwym miejscu), będziesz mógł używać zmiennej `CurrentValue` w każdej procedurze należącej do modułu, a wartość tej zmiennej będzie zachowywana również pomiędzy poszczególnymi procedurami.

Zmienne publiczne (o zasięgu globalnym)

Jeżeli chcesz zapewnić dostęp do zmiennej z poziomu wszystkich procedur należących do wszystkich modułów danego skoroszytu, zadeklaruj ją na poziomie modułu (w sekcji *Declarations*), używając słowa kluczowego `Public`. Oto przykład takiej deklaracji.

```
Public CurrentRate As Long
```

Po użyciu słowa kluczowego `Public` zmienna `CurrentRate` jest dostępna we wszystkich modułach skoroszytu — nie tylko w tym, w którym została zadeklarowana. Instrukcja ta musi się znaleźć przed pierwszą instrukcją procedury `Sub` lub `Function` w module.

SPRAWY TECHNICZNE

Jeżeli chcesz udostępnić zmienną również w modułach innych skoroszytów, musisz ją zadeklarować jako `Public` i ustanowić odwołanie do skoroszytu, w którym zawarta została deklaracja tej zmiennej. Odwołanie możesz ustanowić przy użyciu polecenia *Tools/References* (narzędzia/odwołania) w edytorze VBE. W praktyce, zmienne współdzielone pomiędzy skoroszytami wykorzystywane są niezmiernie rzadko. Odkąd programuję w VBA, ani razu nie użyłem takiej zmiennej. Tak czy inaczej myślę, że warto wiedzieć o takiej możliwości. Kto wie, czy takie pytanie nie padnie w *Milionerach*?

Zmienne statyczne

Standardowo, kiedy zakończy się wykonywanie procedury, wartości wszystkich zawartych w niej zmiennych zostają zresetowane. Wyjątek stanowią *zmienne statyczne*, które zachowują swoje wartości również po zakończeniu wykonywania procedury. Zmienne statyczne deklaruje się w obrębie procedury. Zmienna taka może być użyteczna, jeśli chcesz na bieżąco znać liczbę wykonań danej procedury. Możesz wówczas zadeklarować zmienną statyczną i zwiększać jej wartość przy każdym wywołaniu procedury.

Jak pokazano w poniższym listingu, zmienne statyczne deklaruje się za pomocą słowa kluczowego `Static`.

```
Sub MySub()
   Static Counter As Integer
   Dim Msg As String
   Counter = Counter + 1
   Msg = "Liczba wywołań procedury: " & Counter
   MsgBox Msg
End Sub
```

Powyższy program na bieżąco śledzi liczbę wywołań procedury i wyświetla tę liczbę w oknie komunikatu. Wartość zmiennej `Counter` nie jest resetowana po dotarciu do końca procedury, lecz dopiero po zamknięciu i ponownym otwarciu skoroszytu.

Pomimo że wartość zmiennej zadeklarowanej jako `Static` jest przechowywana również po zakończeniu procedury, zmienna taka nie jest dostępna z poziomu innych procedur. W poprzednim przykładzie zmienna `Counter` i przechowywana przez nią wartość są dostępne tylko w obrębie procedury `MySub`. Innymi słowy, `Counter` jest zmienną lokalną.

Czas życia zmiennych

Nic nie jest wieczne, dotyczy to również zmiennych. Zasięg zmiennej nie tylko określa, gdzie można jej używać, lecz również determinuje okoliczności, w których zmienna usuwana jest z pamięci komputera.

Pamięć komputera można oczyścić ze wszystkich zmiennych na trzy sposoby.

- Naciskając przycisk *Reset* (mały, kwadratowy, niebieski przycisk znajdujący się na pasku narzędzi *Standard* edytora VBE).
- Naciskając przycisk *End*, gdy pojawi się okno dialogowe z komunikatem błędu czasu wykonania (*runtime error*).
- Umieszczając instrukcję `End` w dowolnym miejscu procedury. Instrukcja ta nie jest tożsama z instrukcjami `End Sub` i `End Function`.

W przeciwnym wypadku jedynie zmienne o zasięgu procedury (zmienne lokalne) zostaną usunięte z pamięci w momencie, gdy instrukcje danego makra dobiegną końca. Zmienne statyczne, zmienne o zasięgu jednego modułu oraz zmienne globalne (publiczne) będą zachowywać swoje wartości pomiędzy kolejnymi wywołaniami procedury.


OSTRZEŻENIE

Gdy używasz zmiennych o zasięgu modułu lub globalnym, upewniaj się, że w danym momencie przechowują takie wartości, jakich oczekujesz. Nigdy nie możesz mieć pewności, czy w wyniku zaistnienia jednej z wyżej opisanych sytuacji zmienne te nie utraciły przechowywanych wartości.

Stałe

Wartość zmiennej może się zmieniać w trakcie działania programu. Z reguły tak się dzieje, dlatego też nazywane są *zmiennymi*. Czasami jednak trzeba odwołać się do wartości liczbowej lub tekstowej, która nigdy się nie zmienia. W takim przypadku najlepiej sprawdzi się *stała* — element posiadający nazwę, którego wartość się nie zmienia.

Jak pokazano w następnym przykładzie, stałe deklaruje się za pomocą słowa kluczowego `Const`. Deklaracja stałej nadaje jej jednocześnie określoną wartość.

```
Const NumQuarters As Integer = 4
Const Rate = .0725, Period = 12
Const ModName As String = "Makra Budżetu"
Public Const AppName As String = "Aplikacja Budżet"
```


WSKAZÓWKA

Używanie stałych w miejsce wpisywanych na sztywno wartości liczbowych i tekstowych stanowi doskonałą praktykę programistyczną. Jeżeli w procedurze kilkakrotnie odnosisz się do jakiejś określonej wartości (takiej jak na przykład stopa procentowa), lepiej będzie zadeklarować tę wartość jako stałą i posługiwać się nazwą stałej, niż każdorazowo wpisywać określoną liczbę. Dzięki temu kod będzie czytelniejszy i łatwiej będzie go zmodyfikować. Gdy zmieni się stopa procentowa, będziesz musiał zmienić nie wszystkie, a tylko jedną instrukcję.


ZAPAMIĘTAJ

Stałe mogą się różnić zasięgiem, podobnie jak zmienne. Miej na uwadze poniższe wskazówki.

- Jeżeli stała ma być dostępna jedynie z poziomu danej procedury, zadeklaruj ją za instrukcją `Sub` lub `Function` tej procedury.
- Jeżeli stała ma być dostępna z poziomu wszystkich procedur danego modułu, zadeklaruj ją w sekcji *Declarations* tego modułu.
- Jeżeli stała ma być dostępna z poziomu wszystkich modułów danego skoroszytu, zadeklaruj ją w sekcji *Declarations* dowolnego modułu, używając słowa kluczowego `Public`.

W przeciwieństwie do zmiennej wartość przypisana stałej nie ulega zmianie. Próba użycia instrukcji VBA do zmiany wartości stałej zakończy się zgłoszeniem błędu. Nie jest to zbyt zaskakujące, gdy uwzględnimy fakt, że wartość stałej musi pozostać stała. Jeżeli w trakcie wykonywania programu trzeba zmienić wartość stałej, oznacza to, że właściwie powinieneś był użyć zmiennej.

Stałe predefiniowane

Excel i VBA zawierają wiele predefiniowanych stałych, których można używać bez konieczności ich deklarowania. Rejestrator makr zazwyczaj operuje nie na konkretnych wartościach, lecz właśnie na takich stałych. Na ogół nie trzeba też znać wartości tych stałych, aby z nich korzystać. W następującym prostym przykładzie wykorzystano predefiniowaną stałą `xlCalculationManual`, aby zmienić wartość właściwości `Calculation` obiektu `Application`. Innymi słowy, tryb przeliczania skoroszytu został zmieniony na ręczny.

```
Sub CalcManual()
   Application.Calculation = xlCalculationManual
End Sub
```

Jeżeli zajrzysz do systemu pomocy Excela, znajdziesz tam następujące informacje:

Nazwa stałej	Wartość	Opis
`xlCalculation ↪Automatic`	–4105	Włącza tryb automatycznego przeliczania skoroszytu.
`xlCalculation ↪Manual`	–4135	Włącza tryb ręcznego przeliczania skoroszytu, które następuje na polecenie użytkownika.
`xlCalculation ↪Semiautomatic`	2	Włącza tryb półautomatycznego przeliczania skoroszytu, w którym skoroszyt jest przeliczany automatycznie, ale zmiany w tabelach są ignorowane.

Rzeczywistą wartością stałej `xlCalculationManual` jest więc `–4135`. Z oczywistych powodów dużo łatwiej posługiwać się nazwą zmiennej, niż próbować zapamiętywać tak dziwną wartość. Jak pokazano w tym przykładzie, wiele predefiniowanych stałych to po prostu arbitralne liczby o specjalnym znaczeniu dla języka VBA.

Aby poznać rzeczywistą wartość predefiniowanej zmiennej, użyj okna *Immediate* w edytorze VBE i wykonaj instrukcję analogiczną do tej:

```
? xlCalculationAutomatic
```

Jeżeli okno *Immediate* nie jest widoczne, naciśnij kombinację klawiszy *Ctrl+G*. Znak zapytania jest odpowiednikiem polecenia `Print`.

Łańcuchy znaków

Excel może wykonywać operacje nie tylko na liczbach, lecz również na danych tekstowych. Nie zdziwi Cię więc fakt, że VBA dysponuje podobnymi umiejętnościami. Fragmenty tekstu określane są często jako *łańcuchy znaków* (ang. *string*). W VBA rozróżnia się dwa rodzaje łańcuchów.

- **Łańcuchy znaków o stałej długości** deklaruje się z podaniem określonej liczby znaków. Maksymalna długość to 65 526 znaków. To całkiem spora liczba! Dla porównania, liczba znaków w tym rozdziale to mniej więcej połowa tego limitu.
- **Łańcuchy znaków o zmiennej długości** teoretycznie mogą przechowywać aż dwa miliardy znaków. Jeżeli potrafisz pisać z prędkością pięciu znaków na sekundę, wystukanie dwóch miliardów znaków zajęłoby Ci prawie 4700 dni (przy założeniu, że zrezygnowałbyś z przerw na sen i posiłki).

Deklarując zmienną łańcuchową za pomocą słowa kluczowego `Dim`, możesz określić maksymalną długość łańcucha, o ile ją znasz (jest to wówczas łańcuch o stałej długości), lub też pozwolić, by długość była określana dynamicznie przez VBA (łańcuch o zmiennej długości). W poniższym przykładzie zadeklarowano zmienną `MyString` jako łańcuch o maksymalnej długości wynoszącej 50 znaków. (Aby określić liczbę znaków, maksymalnie 65 526, używa się znaku `*` (*asterisk*) , czyli naszej popularnej „gwiazdki"). Zmienna `YourString` również została zadeklarowana jako łańcuch znaków, ale jego długość pozostała nieokreślona.

```
Dim MyString As String * 50
Dim YourString As String
```

Deklarując zmienne łańcuchowe o liczbie znaków przekraczającej 999, w zapisie tej liczby nie używaj spacji jako separatora tysięcy. Nawiasem mówiąc, kiedy będziesz wprowadzał wartości liczbowe w VBA, nie używaj żadnego separatora tysięcy. Pamiętaj również, by w roli separatora dziesiętnego używać kropki zamiast przecinka, charakterystycznego dla polskiego formatowania wartości liczbowych.

Daty i godziny

Kolejnym użytecznym typem danych jest `Date`. Do przechowywania dat można wprawdzie stosować zmienne łańcuchowe, ale niemożliwe jest wówczas wykonywanie na nich operacji. Używając dedykowanego datom typu `Date`, zapewniasz sobie więcej swobody w programowaniu. Załóżmy przykładowo, że chcesz obliczyć liczbę dni, jakie dzielą dwie daty. Byłoby to karkołomne (jeśli nie niemożliwe) wyzwanie, jeżeli daty byłyby zapisane w postaci normalnych ciągów znaków.

Zmienna o typie zadeklarowanym jako `Date` może przechowywać daty z zakresu od 1 stycznia 100 roku do 31 grudnia 9999 roku. Jest to przedział niemalże 10 000 lat i z pewnością wystarczy do opracowania nawet najbardziej śmiałej pro-

gnozy finansowej. Typ `Date` może być również wykorzystywany do wykonywania operacji na czasie zegarowym (VBA nie posiada typu danych dedykowanego określeniom czasu zegarowego).

W poniższym przykładzie zadeklarowane zostały dwie zmienne i dwie stałe typu `Date`.

```
Dim Today As Date
Dim StartTime As Date
Const FirstDay As Date = #1/1/2019#
Const Noon = #12:00:00#
```

Jak widać, w języku VBA określenia daty i godziny umieszczane są pomiędzy dwoma krzyżykami (*hash*).


ZAPAMIĘTAJ

Sposób wyświetlania zmiennych typu `Date` zależy od formatu daty, jaki ustawiony został w systemie operacyjnym Twojego komputera. Daty wyświetlane są zgodnie z wybranym krótkim formatem daty, a czas według wybranego formatu czasu systemowego (format 12- lub 24-godzinny). Ustawienia te przechowywane są w rejestrze systemu Windows i można je modyfikować po wybraniu karty *Zegar, język i region* w *Panelu sterowania*. Jak widać, format wyświetlania daty i czasu z poziomu języka VBA może się różnić w zależności od ustawień systemu, w którym uruchomiono aplikację.

Pisząc programy w języku VBA, musisz posługiwać się jednym z amerykańskich formatów dat (na przykład *mm/dd/rrrr*). W poniższej instrukcji zmiennej `MyDate` przypisana zostaje zatem data *11 października 2019* (a nie *10 listopada 2019*), nawet jeśli w ustawieniach systemu wybrano format *dd/mm/rrrr*.

```
MyDate = #10/11/2019#
```

Gdy wyświetlisz powyższą zmienną (na przykład przy użyciu funkcji `MsgBox`), VBA pokaże datę w formacie zgodnym z ustawieniami systemu. Jeżeli Twój system operacyjny używa formatu *rrrr-mm-dd* (domyślnego dla polskiej wersji językowej), wartość zmiennej `MyDate` będzie wyświetlana jako *2019-10-11*.

Instrukcje przypisania

Instrukcja przypisania to instrukcja języka VBA, która przypisuje wynik pewnego wyrażenia do zmiennej lub obiektu. System pomocy Excela definiuje termin *wyrażenie* jako:

> *...kombinację słów kluczowych, operatorów, zmiennych i stałych, której wynikiem jest łańcuch znaków, liczba lub obiekt. Wyrażenie może być użyte do wykonania obliczeń, operacji na znakach lub weryfikacji danych.*

Duża część pracy w języku VBA polega na tworzeniu (i debugowaniu) wyrażeń. Jeśli wiesz, jak opracowywać formuły w Excelu, tworzenie wyrażeń również nie przysporzy Ci trudności. Wynik formuły arkusza kalkulacyjnego wyświetlany jest w komórce, natomiast wynik wyrażenia VBA może zostać przypisany do zmiennej.

Przykłady instrukcji przypisania

W poniższych przykładach instrukcji przypisania wyrażenia znajdują się po prawej stronie znaku równości.

```
x = 1
x = x + 1
x = (y * 2) / (z * 2)
HouseCost = 375000
FileOpen = True
Range("Rok").Value = 2019
```

Wyrażenia mogą być na tyle złożone, na ile jest to potrzebne. Aby poprawić czytelność dłuższych wyrażeń, używaj znaku kontynuacji wiersza kodu (jest to znak podkreślenia poprzedzony spacją).

Wyrażenia często zawierają funkcje, takie jak predefiniowane funkcje języka VBA, funkcje skoroszytów Excela lub funkcje napisane przez Ciebie w języku VBA. Funkcje zostaną bardziej szczegółowo omówione w rozdziale 9.

O znaku równości

Jak widziałeś w poprzednim przykładzie, znak równości funkcjonuje w VBA jako operator przypisania. Prawdopodobnie zwykłeś traktować ten znak jako matematyczny symbol równości. Dlatego też instrukcje przypisania podobne do tej mogą wprawić w niezłe osłupienie:

```
z = z + 1
```

W jakim zwariowanym świecie `z` równe jest samemu sobie plus `1`? Odpowiedź brzmi: w żadnym z poznanych dotąd. W tym przypadku w wyniku wykonania operacji przypisania wartość zmiennej `z` zostaje zwiększona o `1`. Jeżeli więc `z` równe jest `12`, wykonanie tej instrukcji spowoduje, że będzie wynosić `13`. Po prostu zapamiętaj, że w instrukcjach przypisania znak równości używany jest nie jako znak równości, lecz jako operator.

Proste operatory

Operatory odgrywają ważną rolę w języku VBA. Poza znakiem równości, opisanym w poprzednim punkcie, VBA udostępnia kilka innych operatorów, które znajdziesz w tabeli 7.3. Powinieneś je znać, ponieważ tych samych operatorów (z wyjątkiem `Mod`) używa się w formułach arkusza.

TABELA 7.3. Operatory języka VBA

Funkcja	Symbol operatora
Dodawanie	+
Mnożenie	*
Dzielenie	/
Odejmowanie	-
Potęgowanie	^
Konkatenacja ciągów znaków	&
Dzielenie całkowite (wynikiem jest zawsze liczba całkowita)	\
Dzielenie modulo (wyznaczanie reszty z dzielenia)	Mod

W formułach Excela dzielenie modulo (wyznaczanie reszty z dzielenia) wykonuje się przy użyciu funkcji `MOD`. Dla przykładu poniższa formuła zwróci jako wynik `2` (jest to reszta z dzielenia `12` przez `5`).

```
=MOD(12;5)
```

W języku VBA operator `Mod` używany jest w sposób zaprezentowany poniżej. Po wykonaniu tej instrukcji wartością zmiennej `z` będzie `5`.

```
z = 12 Mod 5
```

Konkatenacja jest terminem należącym do żargonu programistów i oznacza mniej więcej tyle, co po prostu *łączenie*; inaczej mówiąc, kiedy dokonujesz *konkatenacji ciągów znaków*, oznacza to mniej więcej tyle, że łączysz ze sobą ciągi znaków w celu utworzenia jednego, nowego łańcucha tekstu.

Jak pokazano w tabeli 7.4, język VBA dysponuje pełnym wachlarzem operatorów logicznych.

TABELA 7.4. Operatory logiczne języka VBA

Operator	Funkcja
`Not`	Negacja logiczna wyrażenia
`And`	Koniunkcja logiczna (iloczyn logiczny) dwóch wyrażeń
`Or`	Alternatywa logiczna (suma logiczna) dwóch wyrażeń
`XoR`	Alternatywa wykluczająca (różnica symetryczna) dwóch wyrażeń
`Eqv`	Równoważność logiczna dwóch wyrażeń
`Imp`	Implikacja logiczna dwóch wyrażeń

Priorytety operatorów w VBA są dokładnie takie same jak w formułach Excela. Najwyższy priorytet ma potęgowanie. Następne są mnożenie i dzielenie, a za nimi plasują się dodawanie i odejmowanie. Aby zmienić naturalną kolejność wykonywania działań, możesz użyć nawiasów. Cokolwiek znajdzie się wówczas między nawiasami, będzie miało pierwszeństwo przed innymi operatorami. Spójrz na poniższy kod.

```
x = 3
y = 2
z = x + 5 * y
```

Jaka będzie wartość `z` po wykonaniu tych instrukcji? Jeżeli odpowiedziałeś, że `13`, dostajesz złoty medal za znajomość kolejności wykonywania działań. Jeżeli odpowiedziałeś, że `16`, przeczytaj to: operacja mnożenia (`5 * y`) wykonywana jest w pierwszej kolejności, a jej wynik jest dodawany do `x`.

Jeżeli masz problemy z zapamiętaniem prawidłowej kolejności pierwszeństwa operatorów, dodaj nawias wokół części, które mają zostać obliczone jako pierwsze. Na przykład poprzednie wyrażenie przypisania z użyciem nawiasów wygląda następująco:

```
z = x + (5 * y)
```

Nie obawiaj się używać nawiasów również tam, gdzie nie są wymagane, szczególnie wtedy, kiedy dzięki nim kod będzie łatwiejszy do zrozumienia. W VBA nadmiarowe nawiasy nie mają żadnego znaczenia.

Praca z tablicami

Tablice obsługiwane są w większości języków programowania, a język VBA nie jest tutaj wyjątkiem. *Tablica* to grupa zmiennych dzielących tę samą nazwę. Aby odwołać się do konkretnej zmiennej z tablicy, należy podać nazwę tablicy oraz numer indeksu tej zmiennej w nawiasach. Aby na przykład przechować nazwy wszystkich miesięcy, można zdefiniować tablicę dwunastu zmiennych łańcuchowych. Jeżeli nazwę tablicy określisz jako `MonthNames`, będziesz mógł się odwołać do jej pierwszego elementu za pomocą wyrażenia `MonthNames(1)`, do elementu drugiego poprzez `MonthNames(2)` i tak dalej.

Deklarowanie tablic

Aby można było korzystać z tablicy, trzeba ją najpierw zadeklarować. Wyjątków od tej reguły nie przewidziano. W odróżnieniu od zwykłych zmiennych język VBA jest tutaj bardzo restrykcyjny. Tablicę deklarujemy za pomocą wyrażenia `Dim` lub `Public`, tak samo jak zwykłą zmienną. Należy tutaj jednak dodatkowo określić liczbę elementów tablicy. Robi się to, podając pierwszy numer indeksu, słowo kluczowe `To`

oraz ostatni numer indeksu, całość umieszcza się jednocześnie w nawiasie. W poniższym przykładzie pokazano, jak zadeklarować tablicę składającą się ze 100 liczb całkowitych.

```
Dim MyArray(1 To 100) As Integer
```

Deklarując tablicę, możesz podać tylko górną granicę zakresu indeksów. Jeżeli pominiesz numer indeksu początkowego, VBA przyjmie 0 jako wartość domyślną. Obie poniższe instrukcje deklarują zatem tę samą tablicę, składającą się ze 101 elementów.

```
Dim MyArray(0 To 100) As Integer
Dim MyArray(100) As Integer
```

Jeżeli chcesz, aby domyślną wartością początkową indeksu tablic deklarowanych w VBA było 1 (zamiast 0), umieść poniższą instrukcję w sekcji *Declarations* na początku odpowiedniego modułu.

```
Option Base 1
```

Instrukcja ta wymusza na VBA przyjmowanie 1 jako pierwszego numeru indeksu, jeżeli w deklaracji tablicy określono jedynie wartość indeksu końcowego. Przy założeniu, że użyto powyższej instrukcji, efekt wywołania następujących dwóch instrukcji jest identyczny — deklarowana jest tablica zawierająca 100 elementów.

```
Dim MyArray(1 To 100) As Integer
Dim MyArray(100) As Integer
```

Tablice wielowymiarowe

Wszystkie tablice, jakie utworzyliśmy w poprzednich przykładach, były tablicami jednowymiarowymi. Każdą z nich możesz sobie wyobrazić jako pojedynczą oś wartości. Niemniej jednak tablice tworzone w języku VBA mogą mieć aż 60 wymiarów (w praktyce rzadko będziesz potrzebować tablicy o więcej niż dwóch lub trzech wymiarach). W poniższym przykładzie deklarowana jest dwuwymiarowa tablica 81 liczb całkowitych.

```
Dim MyArray(1 To 9, 1 To 9) As Integer
```

Tablicę tę możesz sobie wyobrazić jako tabelę o wymiarach 9×9 — w sam raz na planszę do sudoku.

Aby odwołać się do wybranego elementu takiej tablicy, należy określić dwa numery indeksów (odpowiadające numerowi wiersza i kolumny w tabeli). W poniższym przykładzie pokazano, jak przypisać wartość wybranemu elementowi tej tablicy.

```
MyArray(3, 4)= 125
```

Instrukcja ta przypisuje wartość pojedynczemu elementowi tablicy. Przyrównując tablicę do tabeli o wymiarach 9 × 9, można powiedzieć, że wartość 125 została przypisana elementowi znajdującemu się w trzecim wierszu i czwartej kolumnie tabeli.

Oto jak można zadeklarować tablicę trójwymiarową, zawierającą 1000 elementów.

```
Dim MyArray(1 To 10, 1 To 10, 1 To 10) As Integer
```

Tablicę trójwymiarową przyrównać można do sześcianu. Wizualizacja tablicy o więcej niż trzech wymiarach jest już nieco trudniejsza.

Tablice dynamiczne

Możesz również tworzyć **tablice dynamiczne**. Tablica dynamiczna nie posiada odgórnie określonej liczby elementów. Tablice tego typu deklaruje się, używając pary pustych nawiasów.

```
Dim MyArray() As Integer
```

Zanim można będzie użyć takiej tablicy, trzeba posłużyć się instrukcją `ReDim`, aby przekazać do VBA informację, ile elementów posiada tablica. Zazwyczaj liczba elementów tablicy określana jest w trakcie wykonywania programu. Instrukcji `ReDim` można używać wielokrotnie, zmieniając w ten sposób rozmiar tablicy tak często, jak potrzeba. W poniższym przykładzie pokazano, jak zmienić liczbę elementów w tablicy dynamicznej. Zakładamy przy tym, że zmienna `NumElements` zawiera wartość, która została obliczona w kodzie programu.

```
ReDim MyArray(1 To NumElements)
```


OSTRZEŻENIE

Zmiana rozmiarów tablicy za pomocą instrukcji `ReDim` powoduje usunięcie wszystkich wartości, jakie były do tej pory przechowywane w jej elementach. Aby uniknąć usunięcia starych wartości, należy użyć słowa kluczowego `Preserve`. W poniższym przykładzie demonstruję, jak zachować wartości przechowywane w tablicy przy zmianie jej wymiarów.

```
ReDim Preserve MyArray(1 To NumElements)
```

Jeżeli tablica `MyArray` posiada aktualnie dziesięć elementów, a wartością zmiennej `NumElements` jest `12`, to po wywołaniu powyższej instrukcji pierwsze dziesięć elementów tablicy pozostanie bez zmian, a dodatkowo w tablicy pojawią się dwa nowe miejsca na dodatkowe elementy (w wyniku dopełnienia do wartości przechowywanej w zmiennej `NumElements`). Jeżeli jednak wartością zmiennej `NumElements` będzie `7`, to pierwsze siedem elementów zostanie zachowanych, ale pozostałe trzy zostaną unicestwione.

Temat tablic pojawi się ponownie w rozdziale 10., gdzie będziemy omawiać działanie pętli.

Stosowanie etykiet

We wczesnych wersjach języka BASIC każdy wiersz kodu musiał rozpoczynać się od numeru wiersza. Jeżeli przykładowo w latach 70. ubiegłego wieku (ubrany — oczywiście — w dzwony) napisałeś w tym języku jakiś program, mógł on wyglądać mniej więcej tak:

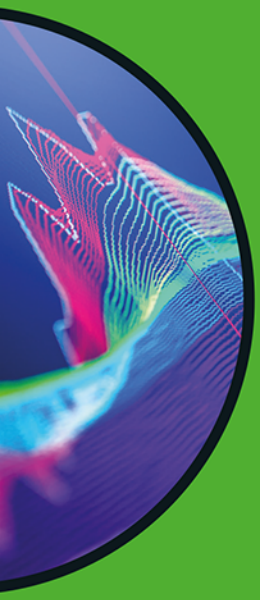
```
010: LET X=5
020: LET Y=3
030: LET Z=Z*Y
040: PRINT Z
050: END
```

VBA dopuszcza użycie takiego numerowania wierszy, a nawet dozwolone są etykiety tekstowe. Zazwyczaj nie używa się etykiet dla każdego wiersza kodu, jednak czasami etykieta taka może być przydatna. Etykietę wstawiamy na przykład wtedy, gdy chcemy użyć instrukcji `GoTo` (opiszemy ją w rozdziale 10.). Etykieta musi być umieszczona na początku wiersza, zaczynać się od znaku alfanumerycznego (bez spacji) i musi kończyć się średnikiem.


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

Pracuj efektywniej dzięki językowi VBA

Niezależnie od tego, czy chcesz być bardziej produktywny, zautomatyzować nużące zadania, czy stworzyć własną, zabójczą aplikację, książka *Excel. Programowanie w VBA dla bystrzaków. Wydanie V* zapewni Ci podstawową wiedzę na temat elementów i koncepcji programowania w Excelu. Skorzystaj z przyjaznych porad na temat najprostszych sposobów tworzenia niestandardowych okien dialogowych, pasków narzędzi i menu i twórz aplikacje Excel dostosowane do Twoich unikalnych potrzeb!



W książce:

- Automatyzacja monotonnych zadań z rutynowym przetwarzaniem danych
- Tworzenie makr, które przetwarzają i formatują dane
- Tworzenie dynamicznie generowanych tabel przestawnych i wykresów
- Niestandardowe funkcje arkuszowe

Michael Alexander

jest certyfikowanym developerem aplikacji Microsoft (MCAD — Microsoft Certified Application Developer) i autorem kilku książek na temat zaawansowanych analiz biznesowych w programach Microsoft Access i Microsoft Excel. Otrzymał tytuł MVP (Most Valuable Professional) za ciągły wkład w funkcjonowanie społeczności użytkowników Excela.

dla bystrzaków

Zamówienia telefoniczne:
0 801 339900 0 601 339900

septem
septem.pl

Sprawdź najnowsze promocje:
http://dlabystrzakow.pl/promocje
Książki najchętniej czytane:
http://dlabystrzakow.pl/bestsellery
Zamów informacje o nowościach:
http://dlabystrzakow.pl/nowosci

Helion SA
ul. Kościuszki 1c, 44-100 Gliwice
tel.: 32 230 98 63
e-mail: rady@dlabystrzakow.pl
http://dlabystrzakow.pl

Helion

Cena 59,00 zł

ISBN 978-83-283-6511-7

9 788328 365117