Microsoft®

Excel® 2016 PL

Programowanie w VBA

Michael Alexander
Dick Kusleika

Helion

Tytuł oryginału: Excel 2016 Power Programming with VBA

Tłumaczenie: Grzegorz Kowalczyk

ISBN: 978-83-283-2858-7

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

Pliki z przykładami omawianymi w książce można znaleźć pod adresem:
ftp://ftp.helion.pl/przyklady/e26pvw.zip

Drogi Czytelniku!
Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres
http://helion.pl/user/opinie/e26pvw
Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Printed in Poland.

Spis treści

Część II. Zaawansowane techniki programowania 359

Rozdział 2

Wprowadzenie do języka VBA

W TYM ROZDZIALE:

- Jak korzystać z rejestratora makr programu Excel
- Jak korzystać z edytora VBE?
- Model obiektowy Excela
- Przykłady zastosowań obiektów `Range`
- Gdzie szukać dodatkowych informacji

Rejestrator makr Excela

Makro to w zasadzie nic innego jak kod programu napisany w języku Visual Basic for Applications (VBA), który możesz uruchamiać w celu wykonania wielu różnych operacji. Makra w Excelu mogą być pisane bezpośrednio w edytorze lub rejestrowane za pomocą rejestratora makr.

Terminologia używana w programowaniu Excela może być nieco myląca. Zarejestrowane makro z technicznego punktu widzenia nie różni się niczym od procedury VBA, którą możesz napisać ręcznie. Określenia „makro" i „procedura VBA" są bardzo często używane wymiennie. Wielu użytkowników Excela wszystkie procedury języka VBA nazywa makrami. Z drugiej jednak strony, bardzo często użytkownicy Excela, mówiąc o makrach, mają na myśli procedury VBA zarejestrowane przy użyciu rejestratora makr.

Rejestrowanie makra przypomina nieco zapisywanie nowego numeru w pamięci telefonu komórkowego. Aby to zrobić, najpierw musisz ręcznie wybrać i zapisać numer w telefonie, dzięki czemu możesz później szybko się z nim połączyć za pomocą naciśnięcia jednego klawisza. Podobnie ma się sprawa z Excelem, gdzie możesz rejestrować różne operacje podczas ich wykonywania. Kiedy Ty jesteś zajęty działaniem, Excel przetwarza wszystkie wykonywane przez Ciebie operacje, naciśnięcia klawiszy i kliknięcia myszy na odpowiadające im polecenia VBA. Po zarejestrowaniu makra możesz w dowolnym momencie uruchomić i odtworzyć wykonywane wcześniej działania.

Zdecydowanie najlepszym sposobem na szybkie zapoznanie się ze specyfiką i składnią poleceń języka VBA jest uruchomienie rejestratora makr, wykonanie kilku operacji w Excelu, a następnie uważne przeanalizowanie kodu utworzonego przez rejestrator.

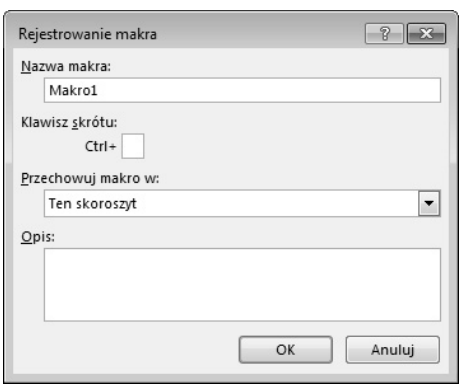
W tym podrozdziale będziesz pracował z różnymi makrami i dowiesz się, jak używać rejestratora makr do poznawania tajników programowania w języku VBA.

Tworzenie pierwszego makra

Aby móc rozpocząć rejestrowanie swojego pierwszego makra, musisz najpierw odszukać przycisk uruchamiający rejestrator makr, który znajduje się na karcie *Deweloper*. Niestety domyślnie po zainstalowaniu Excela karta *Deweloper* może nie być widoczna. Jeżeli chcesz pracować z makrami VBA, będziesz musiał ją samodzielnie włączyć. Aby to zrobić, powinieneś wykonać następujące operacje:

1. Przejdź na kartę *Plik* i naciśnij przycisk *Opcje*.
2. Na ekranie pojawi się okno dialogowe opcji Excela. Wybierz opcję *Dostosowywanie Wstążki*.
3. Na liście dostępnych kart, znajdującej się w prawej części okna, odszukaj i zaznacz kartę *Deweloper*.
4. Naciśnij przycisk *OK*, aby powrócić do głównego okna Excela.

Od tej chwili na Wstążce Excela karta *Deweloper* powinna być widoczna. Aby teraz uruchomić rejestrator makr, przejdź na kartę *Deweloper* i naciśnij przycisk *Zarejestruj makro*, znajdujący się w grupie opcji *Kod*. Na ekranie pojawi się okno dialogowe *Rejestrowanie makra*, pokazane na rysunku 2.1.



Rysunek 2.1. Okno dialogowe Rejestrowanie makra

W oknie dialogowym *Rejestrowanie makra* możemy wyróżnić cztery pola:

- ***Nazwa makra***. Nazwa tego pola nie wymaga chyba specjalnych objaśnień. Excel przypisuje do nowego makra domyślną nazwę, składającą się ze słowa *Makro* i kolejnego numeru, ale oczywiście możesz (i w większości przypadków powinieneś) zmienić tę nazwę na coś bardziej opisowego. Na przykład makro, które będzie zmieniało wygląd tabeli, możesz nazwać `FormatowanieTabeli`.
- ***Klawisz skrótu***. Aby dane makro zostało uruchomione, musi zajść określone zdarzenie. Takim zdarzeniem może być naciśnięcie danej kombinacji klawiszy, naciśnięcie przycisku polecenia czy otwarcie lub zamknięcie skoroszytu. Jeżeli przypiszesz do makra określoną kombinację klawiszy, to makro zostanie uruchomione za każdym razem, kiedy użytkownik naciśnie taką kombinację. Pole *Klawisz skrótu* jest opcjonalne.
- ***Przechowuj makro w***. Domyślną wartością tego pola jest *Ten skoroszyt*. Wybranie tej opcji oznacza po prostu, że nasze makro zostanie zapisane w aktualnie otwartym skoroszycie. Dzięki temu następnym razem, kiedy otworzysz ten skoroszyt, będziesz mógł uruchomić zapisane w nim makro. Co więcej, jeżeli udostępnisz ten skoroszyt innemu użytkownikowi, to on również będzie mógł takie makro uruchomić (oczywiście przy założeniu, że będzie miał odpowiednio ustawione opcje bezpieczeństwa makr — więcej informacji na ten temat znajdziesz w dalszej części tego rozdziału).
- ***Opis***. Jest to pole opcjonalne, w którym możesz zamieścić opis przeznaczenia i sposobu działania makra. Takie pole jest bardzo przydatne zwłaszcza w sytuacji, kiedy w skoroszycie zapisanych jest bardzo wiele makr lub chcesz przekazać innym użytkownikom dodatkowe informacje na temat danego makra.

Aby utworzyć proste makro, które będzie wpisywało Twoje imię do wybranej komórki arkusza, otwórz okno rejestratora makr i wykonaj polecenia opisane poniżej:

1. W polu *Nazwa makra* wpisz nową nazwę makra, zastępując domyślną nazwę *Makro1*. W naszym przykładzie wybraliśmy dla tego makra nazwę `MojeDane`.
2. Przypisz do makra kombinację klawiszy *Ctrl+Shift+N*. Aby to zrobić, w polu *Klawisz skrótu* wpisz wielką literę *N*.
3. Naciśnij przycisk *OK*, aby zamknąć okno dialogowe *Rejestrowanie makra* i rozpocząć rejestrowanie makra.
4. Kliknij dowolnie wybraną komórkę arkusza, wpisz w niej swoje imię i nazwisko, a następnie naciśnij klawisz *Enter*.
5. Przejdź na kartę *Deweloper* i naciśnij przycisk *Zatrzymaj rejestrowanie*, znajdujący się w grupie poleceń *Kod* (zamiast tego możesz po prostu nacisnąć przycisk *Zatrzymaj rejestrowanie*, znajdujący się na pasku stanu w dolnej części okna programu Excel).

Przeglądanie i sprawdzanie utworzonego makra

Nasze makro zostało zarejestrowane w nowym module kodu o nazwie `Module1`. Aby wyświetlić kod VBA znajdujący się w tym module, musisz uruchomić edytor VBE. Możesz to zrobić na jeden z dwóch sposobów:

- Naciśnij kombinację klawiszy *Alt+F11*.
- Przejdź na kartę *Deweloper* i naciśnij przycisk *Visual Basic* znajdujący się w grupie opcji *Kod*.

W oknie edytora VBE znajdziesz panel o nazwie *Project* (projekt), w którym wyświetlana jest lista wszystkich aktualnie otwartych skoroszytów i dodatków. Każdy skoroszyt i dodatek stanowi osobny *projekt*, którego węzły składowe możesz dowolnie zwijać i rozwijać. Kod naszego zarejestrowanego przed chwilą makra znajduje się w module kodu o nazwie `Module1` aktywnego skoroszytu. Kiedy dwukrotnie klikniesz lewym przyciskiem myszy ikonę tego modułu, kod makra pojawi się w oknie kodu w prawej części ekranu.

Uwaga

Jeżeli okno eksploratora projektów nie jest widoczne, możesz je otworzyć, wybierając z menu głównego edytora VBE polecenie *View/Project Explorer* (widok/eksplorator projektów). Ten sam efekt możesz uzyskać, naciskając po prostu kombinację klawiszy *Ctrl+R*.

Kod naszego zarejestrowanego makra powinien wyglądać mniej więcej tak:

```
Sub MojeDane()
'
' MojeDane Makro
'
' Klawisz skrótu: Ctrl+Shift+N
'
  ActiveCell.FormulaR1C1 = "Maksymilian Kowalski"
End Sub
```

Nasze makro zostało zarejestrowane w postaci procedury `Sub` o nazwie `MojeDane`. Polecenia kodu VBA w ciele procedury informują Excela o tym, co powinien zrobić po uruchomieniu makra.

Zauważ, że na początku procedury Excel wstawił kilka wierszy komentarza, zawierających informacje, które wprowadziłeś w oknie *Rejestrowanie makra*. Wiersze komentarza (które rozpoczynają się od znaku apostrofu) nie są niezbędne do poprawnego działania makra i usunięcie ich nie będzie miało żadnego wpływu na jego działanie. Zatem jeżeli zignorujemy wszystkie komentarze, okaże się, że nasza procedura tak naprawdę składa się tylko z jednego wiersza kodu:

```
ActiveCell.FormulaR1C1 = "Maksymilian Kowalski"
```

Wykonanie powyższego polecenia powoduje wpisanie ciągu znaków *Maksymilian Kowalski* do aktywnej komórki arkusza.

Testowanie działania makra

Przed rozpoczęciem rejestrowania przypisałeś do makra kombinację klawiszy *Ctrl+Shift+N*. Aby przetestować działanie makra, powinieneś najpierw powrócić do głównego okna Excela. Możesz to zrobić na jeden z dwóch sposobów:

- Naciśnij kombinację klawiszy *Alt+F11*.
- Naciśnij przycisk *View Microsoft Excel* (wyświetl okno programu Excel), znajdujący się na standardowym pasku narzędzi edytora VBE.

Po przejściu do okna Excela aktywuj skoroszyt (może to być skoroszyt zawierający nasz moduł kodu VBA, ale równie dobrze może to być dowolny inny skoroszyt). Kliknij wybraną komórkę arkusza i naciśnij kombinację klawiszy *Ctrl+Shift+N*. Excel uruchomi makro i wpisze w wybranej komórce ciąg znaków *Maksymilian Kowalski*.

Uwaga

Jak pamiętasz, przed rozpoczęciem rejestrowania makra zaznaczyłeś wybraną komórkę arkusza. Jest to bardzo ważny krok. Jeżeli zaznaczysz żądaną komórkę dopiero po uruchomieniu rejestratora, to komórka, którą wybierzesz, zostanie zapisana w kodzie generowanego makra. W takiej sytuacji makro zawsze będzie formatowało tylko tę jedną, zaznaczoną podczas rejestrowania komórkę i dlatego nie będzie makrem uniwersalnym.

Edytowanie makra

Po zarejestrowaniu makra możesz dowolnie modyfikować jego kod. Na przykład załóżmy, że chcesz, aby imię i nazwisko wpisywane do komórki były wyróżnione pogrubioną czcionką. W zasadzie mógłbyś usunąć stare makro i zarejestrować je na nowo, ale żądana zmiana sposobu działania makra jest bardzo prosta, dlatego znacznie lepszym rozwiązaniem będzie po prostu bezpośrednie dokonanie odpowiedniej modyfikacji kodu makra. Naciśnij kombinację klawiszy *Alt+F11*, aby aktywować edytor VBE. Następnie przejdź do modułu kodu `Module1` i wstaw polecenie **`ActiveCell.Font.Bold = True`**, tak jak to zostało pokazane w przykładzie poniżej:

```
Sub MojeDane()
'
' MojeDane Makro
'
' Klawisz skrótu: Ctrl+Shift+N
'
  ActiveCell.Font.Bold = True
  ActiveCell.FormulaR1C1 = "Maksymilian Kowalski"
End Sub
```

Przetestuj nowe makro i sprawdź, czy działa zgodnie z Twoimi oczekiwaniami.

Porównanie rejestrowania makr z odwołaniami względnymi i bezwzględnymi

Poznałeś już podstawowe zasady posługiwania się interfejsem rejestratora makr, zatem nadszedł czas, aby zacząć rejestrowanie znacznie bardziej złożonych makr. Zanim jednak zaczniemy, musisz wiedzieć, że Excel posiada dwa tryby rejestrowania makr — rejestrowanie z odwołaniami bezwzględnymi i rejestrowanie z odwołaniami względnymi.

Rejestrowanie makr w trybie odwołań bezwzględnych

Domyślnie Excel rejestruje makra z odwołaniami bezwzględnymi. Jak zapewne wiesz, odwołania bezwzględne są bardzo często używane w formułach arkuszowych. Jeżeli w formule umieścimy bezwzględne odwołanie do danej komórki, to takie odwołanie nie zostanie automatycznie dostosowane po skopiowaniu formuły do nowej lokalizacji.

Najlepszym sposobem, aby zrozumieć, jak działają odwołania bezwzględne, jest ich wypróbowanie w praktyce. Otwórz skoroszyt z przykładowym zestawem danych, a następnie zarejestruj makro, które będzie zliczało liczbę wierszy (patrz rysunek 2.2).

	A	B	C	D	E	F	G	H	I	J
1		**Region**	**Sklep**	**Kod oddziału**			**Region**	**Sklep**	**Kod oddziału**	
2		Północ	Gdańsk	601419			Południe	Chorzów	173901	
3		Północ	Gdańsk	701407			Południe	Chorzów	301301	
4		Północ	Gdańsk	802202			Południe	Chorzów	302301	
5		Północ	Kołobrzeg	910181			Południe	Chorzów	601306	
6		Północ	Kołobrzeg	920681			Południe	Gliwice	202600	
7		Północ	Ustka	101419			Południe	Gliwice	490260	
8		Północ	Ustka	501405			Południe	Gliwice	490360	
9		Północ	Ustka	503405			Południe	Gliwice	490460	
10		Północ	Ustka	590140			Południe	Katowice	301316	
11		Północ	Szczecin	801211			Południe	Katowice	701309	
12		Północ	Szczecin	802211			Południe	Katowice	702309	
13		Północ	Szczecin	804211			Południe	Wrocław	601310	
14		Północ	Szczecin	805211			Południe	Wrocław	602310	
15		Północ	Szczecin	806211			Południe	Wrocław	801607	
16										

Rysunek 2.2. Przykładowy zestaw danych składa się z dwóch tabel

FTP Na FTP

Skoroszyt z tym przykładem (*Przykładowe dane.xlsm*) znajdziesz na serwerze FTP wydawnictwa Helion (*ftp://ftp.helion.pl/przyklady/e26pvw.zip*).

Aby zarejestrować makro, wykonaj polecenia opisane poniżej:

1. Przed rozpoczęciem rejestrowania makra upewnij się, że zaznaczyłeś komórkę `A1`.
2. Przejdź na kartę *Deweloper* i wybierz polecenie *Zarejestruj makro*.

3. W polu *Nazwa makra* wpisz **`DodajPodsumowanie`**.
4. W polu *Przechowuj makro w* wybierz opcję *Ten skoroszyt*.
5. Naciśnij przycisk *OK*, aby rozpocząć rejestrowanie makra.

   W tym momencie Excel rozpocznie rejestrowanie wykonywanych przez Ciebie operacji.
6. Zaznacz komórkę o adresie `A16` i wpisz w niej etykietę **`RAZEM`**.
7. Zaznacz pierwszą wolną komórkę w kolumnie `D` (powinna to być komórka o adresie `D16`) i wpisz w niej następującą formułę: `=ILE.NIEPUSTYCH(D2:D15)`, która policzy liczbę oddziałów i umieści ją na dole kolumny `D`. Funkcja `ILE.NIEPUSTYCH` została tutaj użyta, ponieważ pozwala na uwzględnienie wszystkich kodów oddziałów (liczby), które ewentualnie zostały zapisane jako tekst.
8. Przejdź na kartę *Deweloper* i zatrzymaj rejestrowanie makra.

Gotowy arkusz powinien wyglądać tak, jak to zostało pokazane na rysunku 2.3.

	A	B	C	D	E	F	G	H	I	J
1		**Region**	**Sklep**	**Kod oddziału**			**Region**	**Sklep**	**Kod oddziału**	
2		Północ	Gdańsk	601419			Południe	Chorzów	173901	
3		Północ	Gdańsk	701407			Południe	Chorzów	301301	
4		Północ	Gdańsk	802202			Południe	Chorzów	302301	
5		Północ	Kołobrzeg	910181			Południe	Chorzów	601306	
6		Północ	Kołobrzeg	920681			Południe	Gliwice	202600	
7		Północ	Ustka	101419			Południe	Gliwice	490260	
8		Północ	Ustka	501405			Południe	Gliwice	490360	
9		Północ	Ustka	503405			Południe	Gliwice	490460	
10		Północ	Ustka	590140			Południe	Katowice	301316	
11		Północ	Szczecin	801211			Południe	Katowice	701309	
12		Północ	Szczecin	802211			Południe	Katowice	702309	
13		Północ	Szczecin	804211			Południe	Wrocław	601310	
14		Północ	Szczecin	805211			Południe	Wrocław	602310	
15		Północ	Szczecin	806211			Południe	Wrocław	801607	
16	RAZEM			14						
17										

Rysunek 2.3. Wygląd arkusza po zakończeniu rejestrowania makra

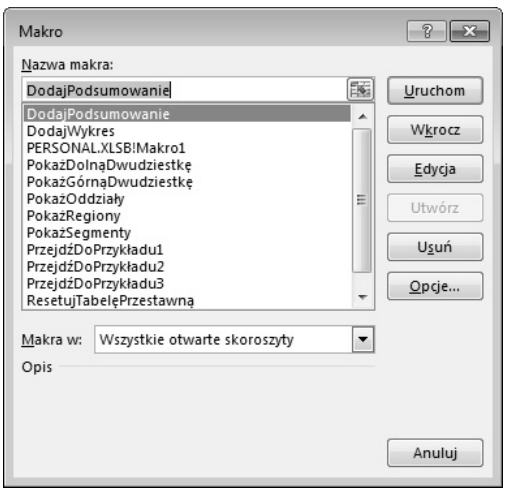
Aby teraz zobaczyć zarejestrowane makro w działaniu, usuń wiersz podsumowania, który przed chwilą dodałeś, i wykonaj makro. Aby uruchomić makro, wykonaj następujące polecenia:

1. Przejdź na kartę *Deweloper* i wybierz polecenie *Makra*.
2. Na liście dostępnych makr odszukaj i zaznacz zarejestrowane przed chwilą makro `DodajPodsumowanie`.
3. Naciśnij przycisk *Uruchom*.

Jeżeli wszystko poszło zgodnie z oczekiwaniami, makro zostanie wykonane bez przeszkód i na dole tabeli pojawi się odpowiednie podsumowanie. A teraz twardy orzech do zgryzienia: niezależnie od tego, jak wiele razy będziesz próbował, nie będziesz w stanie zmusić tego makra

do utworzenia podsumowania dla drugiej tabeli. Dlaczego? Ponieważ zostało zarejestrowane w trybie odwołań bezwzględnych.

Aby zrozumieć, co to znaczy, musisz przeanalizować kod VBA zarejestrowanego makra. Aby to zrobić, przejdź na kartę *Deweloper* i wybierz polecenie *Makra*. Na ekranie pojawi się okno dialogowe *Makra*, przedstawione na rysunku 2.4, w którym domyślnie wyświetlona jest lista makr dostępnych we wszystkich aktualnie otwartych skoroszytach Excela (wraz ze wszystkimi zainstalowanymi dodatkami). Aby ograniczyć liczbę wyświetlanych makr, możesz zmienić ustawienie listy rozwijanej *Makra w* na opcję *Ten skoroszyt*.



Rysunek 2.4. Okno dialogowe Makra

Zaznacz makro `DodajPodsumowanie` i naciśnij przycisk *Edycja*, co spowoduje uruchomienie edytora VBE i wyświetlenie kodu zarejestrowanego makra.

```
Sub DodajPodsumowanie()
  Range("A16").Select
  ActiveCell.FormulaR1C1 = "RAZEM"
  Range("D16").Select
  ActiveCell.FormulaR1C1 = "=COUNTA(R[-14]C:R[-1]C)"
End Sub
```

Zwróć szczególną uwagę na drugi i czwarty wiersz kodu makra. Kiedy zaznaczałeś komórki o adresach `A16` i `D16`, Excel zarejestrował dokładnie takie adresy komórek. Ponieważ makro było rejestrowane w trybie odwołań bezwzględnych, Excel zinterpretował odwołania do komórek jako bezwzględne. Innymi słowy, jeżeli w takim trybie zaznaczyłeś komórkę `A16`, to po uruchomieniu makra Excel zaznaczy komórkę `A16`. W kolejnym podrozdziale przekonasz się, jak wygląda takie makro zarejestrowane w trybie odwołań względnych.

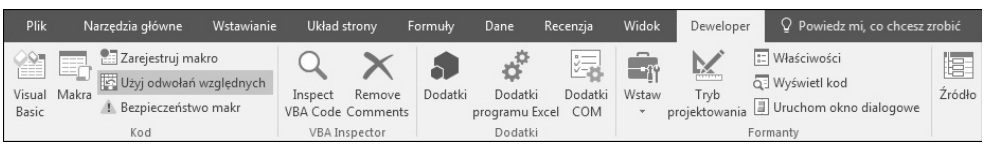
Rejestrowanie makr w trybie odwołań względnych

W terminologii makr Excela odwołanie względne oznacza odwołanie względem aktualnie aktywnej komórki arkusza. Z tego powodu powinieneś zawsze zachować pewną ostrożność, wybierając komórkę aktywną, zarówno podczas rejestrowania makra, jak i później, podczas jego uruchamiania.

Otwórz skoroszyt z przykładowym zestawem danych, z którego korzystaliśmy w poprzednim podrozdziale (plik skoroszytu znajdziesz na serwerze FTP wydawnictwa Helion — *ftp://ftp.helion.pl/przyklady/e26pvw.zip*), a następnie zarejestruj makro w trybie odwołań względnych. Aby to zrobić, wykonaj polecenia opisane poniżej:

1. Przejdź na kartę *Deweloper* i włącz opcję *Użyj odwołań względnych*, tak jak zostało to pokazane na rysunku 2.5.
2. Przed rozpoczęciem rejestrowania makra upewnij się, że zaznaczyłeś komórkę `A1`.
3. Przejdź na kartę *Deweloper* i wybierz polecenie *Zarejestruj makro*.
4. W polu *Nazwa makra* wpisz **`DodajPodsumowanieWzględne`**.
5. W polu *Przechowuj makro w* wybierz opcję *Ten skoroszyt*.
6. Naciśnij przycisk *OK*, aby zacząć rejestrowanie makra.

   W tym momencie Excel rozpocznie rejestrowanie wykonywanych przez Ciebie operacji.
7. Zaznacz komórkę o adresie `A16` i wpisz w niej etykietę **`RAZEM`**.
8. Zaznacz pierwszą wolną komórkę w kolumnie `D` (powinna to być komórka o adresie `D16`) i wpisz w niej następującą formułę: **`=ILE.NIEPUSTYCH(D2:D15)`**.
9. Przejdź na kartę *Deweloper* i zatrzymaj rejestrowanie makra.



Rysunek 2.5. Rejestrowanie makra w trybie odwołań względnych

Po zakończeniu będziemy mieć zarejestrowane dwa makra. Teraz uważnie przeanalizuj kod nowo utworzonego makra.

Aby to zrobić, przejdź na kartę *Deweloper* i wybierz polecenie *Makra*. Na ekranie pojawi się okno dialogowe *Makra*. Odszukaj i zaznacz makro `DodajPodsumowanieWzględne`, a następnie naciśnij przycisk *Edycja*.

Naciśnięcie tego przycisku spowoduje uruchomienie edytora VBE i wyświetlenie kodu zarejestrowanego makra. Tym razem kod naszego makra wygląda następująco:

```
Sub DodajPodsumowanieWzględne()
  ActiveCell.Offset(15, 0).Range("A1").Select
  ActiveCell.FormulaR1C1 = "RAZEM"
  ActiveCell.Offset(0, 3).Range("A1").Select
  ActiveCell.FormulaR1C1 = "=COUNTA(R[-14]C:R[-1]C)"
End Sub
```

Zauważ, że w kodzie procedury nie ma bezpośrednich odwołań do komórek, z wyjątkiem początkowej komórki `A1`. Przyjrzymy się teraz samej procedurze i wyjaśnimy, co znaczą poszczególne fragmenty jej kodu.

Zwróć uwagę, że w wierszu 2. Excel używa właściwości `Offset` aktywnej komórki, która wskazuje konieczność przeniesienia kursora o określoną liczbę komórek w górę lub w dół oraz o określoną liczbę komórek w lewo lub w prawo.

W naszym przypadku właściwość `Offset` nakazuje Excelowi przenieść kursor 15 wierszy w dół i 0 kolumn w prawo od aktywnej komórki (którą jest w tym przypadku komórka `A1`). Excel nie musi bezpośrednio zaznaczać komórki docelowej, tak jak to miało miejsce w przypadku makra z odwołaniami bezwzględnymi.

Aby zobaczyć nasze nowe makro w działaniu, usuń wiersz podsumowania i wykonaj następujące polecenia:

1. Zaznacz komórkę `A1`.
2. Przejdź na kartę *Deweloper* i wybierz polecenie *Makra*.
3. Na liście wyszukaj makro `DodajPodsumowanieWzględne`.
4. Naciśnij przycisk *Uruchom*.
5. Zaznacz komórkę `F1`.
6. Ponownie przejdź na kartę *Deweloper* i wybierz polecenie *Makra*.
7. Na liście wyszukaj makro `DodajPodsumowanieWzględne`.
8. Naciśnij przycisk *Uruchom*.

Zauważ, że tym razem nasze nowe makro, w przeciwieństwie do makra z poprzedniego przykładu, działa poprawnie na obydwu zestawach danych. Ponieważ makro wykonuje operacje na komórkach wyznaczanych w odniesieniu do aktualnie aktywnej komórki, wszystkie etykiety i podsumowania są tworzone poprawnie.

Aby nasze makro działało poprawnie, musisz się po prostu upewnić, że:

- Przed uruchomieniem makra zaznaczyłeś odpowiednią komórkę bazową.
- Zestawy danych mają taką samą liczbę wierszy i kolumn jak zestaw danych użyty podczas rejestrowania makra.

Mamy nadzieję, że po uważnym przeanalizowaniu obu przytoczonych przykładów będziesz już wiedział, na czym polega różnica między rejestrowaniem makra w trybie odwołań bezwzględnych a rejestrowaniem makra w trybie odwołań względnych.

Inne zagadnienia związane z makrami

Na tym etapie rejestrowanie makr w Excelu nie powinno już Ci sprawiać najmniejszych trudności. W tym podrozdziale omówimy kilka dodatkowych zagadnień, o których powinieneś pamiętać podczas samodzielnego tworzenia lub rejestrowania makr.

Rozszerzenia skoroszytów z obsługą makr

Począwszy od wersji 2007, skoroszyty Excela są domyślnie zapisywane z rozszerzeniem *.xlsx*. Niestety pliki w takim formacie nie mogą przechowywać żadnych makr. Jeżeli Twój skoroszyt zawiera jakieś makra i zapiszesz go w formacie *.xlsx*, to automatycznie cały kod VBA zostanie z takiego arkusza usunięty. Na szczęście wcześniej Excel wyświetli na ekranie odpowiednie ostrzeżenie i poprosi o potwierdzenie zamiaru wykonania takiej operacji.

Jeżeli chcesz zachować makra utworzone w skoroszycie, musisz zapisać go jako *Skoroszyt programu Excel z obsługą makr*. Takie pliki posiadają rozszerzenie *.xlsm*. Cała idea takiego rozgraniczenia formatów plików polega na tym, że wszystkie skoroszyty z rozszerzeniem *.xlsx* są automatycznie uznawane za bezpieczne, natomiast skoroszyty *.xlsm* mogą potencjalnie stanowić pewne zagrożenie.

Bezpieczeństwo makr w Excelu

Począwszy od wersji 2010, firma Microsoft wprowadziła znaczące zmiany w modelu bezpieczeństwa pakietu Office. Jedną z najbardziej istotnych zmian było wdrożenie koncepcji dokumentów zaufanych. W skrócie i bez zbytniego zagłębiania się w szczegóły techniczne możemy powiedzieć, że dokument zaufany (ang. *trusted document*) to skoroszyt, który uznałeś za bezpieczny poprzez włączenie w nim obsługi makr.

Jeżeli otworzysz w Excelu skoroszyt zawierający makra, to na ekranie pojawi się okno ostrzeżenia z informacją, że makra (aktywna zawartość skoroszytu) zostały zablokowane.

Jeżeli naciśniesz przycisk *Włącz makra*, to automatycznie taki skoroszyt zostanie uznany za zaufany. Oznacza to, że Excel nie będzie już więcej pytał o włączenie makr dla tego pliku. Podstawowe założenie tego rozwiązania jest takie, że jeżeli raz pokazałeś Excelowi, że „ufasz" danemu skoroszytowi, włączając w nim możliwość wykonywania makr, to istnieje bardzo duże prawdopodobieństwo, że będziesz takie makra włączał za każdym razem, kiedy będziesz otwierał ten plik. Aby ułatwić Ci życie, Excel zapamiętuje, że już kiedyś włączyłeś makra w danym dokumencie, i zawiesza ponowne wyświetlanie komunikatów o włączeniu makr w tym dokumencie.

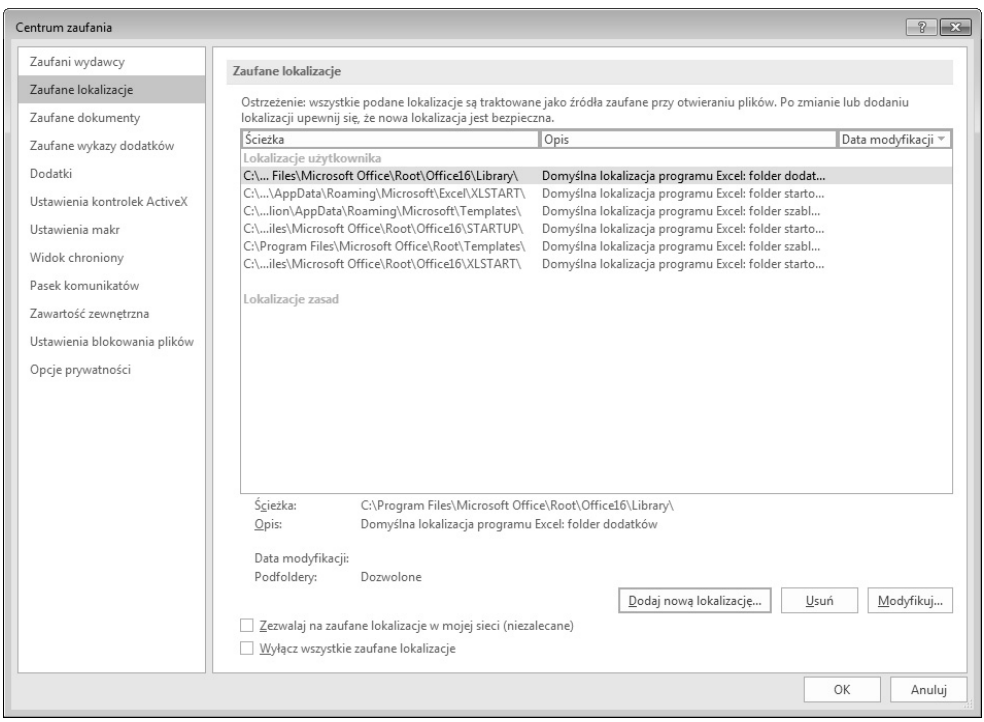
W sumie jest to całkiem niezła wiadomość zarówno dla Ciebie, jak i dla Twoich klientów, którzy po jednorazowym włączeniu makr nie będą już więcej monitowani komunikatami o włączaniu makr, a Ty nie będziesz musiał się martwić, że Twoja oparta na makrach aplikacja Excela przestanie działać z powodu zablokowanych makr.

ZAUFANE LOKALIZACJE

Jeżeli irytuje Cię konieczność nawet jednorazowego włączania obsługi makr, to możesz dla swoich skoroszytów zdefiniować tak zwaną zaufaną lokalizację, czyli utworzyć dedykowany, bezpieczny katalog, w którym będziesz przechowywać wyłącznie zaufane, sprawdzone i bezpieczne skoroszyty. Utworzenie współdzielonej zaufanej lokalizacji pozwoli zarówno Tobie, jak i Twoim klientom na uruchamianie przechowywanych tam skoroszytów z obsługą makr bez żadnych ograniczeń.

Aby utworzyć taką zaufaną lokalizację, powinieneś wykonać następujące polecenia:

1. Przejdź na kartę *Deweloper* i wybierz polecenie *Bezpieczeństwo makr*. Na ekranie pojawi się okno dialogowe *Centrum zaufania*.
2. Wybierz opcję *Zaufane lokalizacje*. W prawej części okna dialogowego pojawi się panel *Zaufane lokalizacje* (patrz rysunek 2.6), w którym znajdziesz listę wszystkich katalogów uważanych za zaufane.



Rysunek 2.6. Panel Zaufane lokalizacje pozwala na dodawanie nowych katalogów uważanych za bezpieczne

3. Naciśnij przycisk *Dodaj nową lokalizację*.
4. Na ekranie pojawi się okno dialogowe *Zaufana lokalizacja pakietu Microsoft Office*. Naciśnij przycisk *Przeglądaj* i wybierz katalog, który chcesz dodać do listy zaufanych lokalizacji.

Po dodaniu zaufanej lokalizacji wszystkie przechowywane w niej skoroszyty po otwarciu w Excelu będą miały automatycznie włączoną obsługę makr.

Zapisywanie makr w skoroszycie makr osobistych

Większość makr tworzonych przez użytkowników jest przeznaczona do działania w określonym skoroszycie, ale od czasu do czasu możesz napisać makro, które powinno być dostępne we wszystkich skoroszytach. Takie makra ogólnego przeznaczenia możesz zapisywać w tzw. skoroszycie makr osobistych (ang. *Personal Macro Workbook*), dzięki czemu będziesz je miał zawsze pod ręką. Skoroszyt makr osobistych jest automatycznie ładowany za każdym razem, kiedy uruchamiasz Excela. Plik tego skoroszytu nosi nazwę *personal.xlsb* i jest automatycznie tworzony dopiero w momencie, kiedy po raz pierwszy zarejestrujesz makro, którego docelową lokalizacją będzie ten skoroszyt.

Aby zapisać rejestrowane makro w skoroszycie makr osobistych, powinieneś w oknie *Rejestrowanie makra* wybrać z listy *Przechowuj makro w* opcję *Skoroszyt makr osobistych* (patrz rysunek 2.1 we wcześniejszej części tego rozdziału).

Jeżeli przechowujesz swoje makra w skoroszycie makr osobistych, to nie musisz już pamiętać o jego otwieraniu, ponieważ Excel będzie to robił automatycznie. Jeżeli dokonasz w skoroszycie makr osobistych jakichś zmian, to przed zakończeniem działania Excel zapyta Cię, czy chcesz zachować wprowadzone zmiany.

Uwaga

Skoroszyt makr osobistych jest domyślnie ukryty, tak aby nie przeszkadzał Ci w codziennej pracy z Excelem.

Przypisywanie makr do przycisków i innych formantów

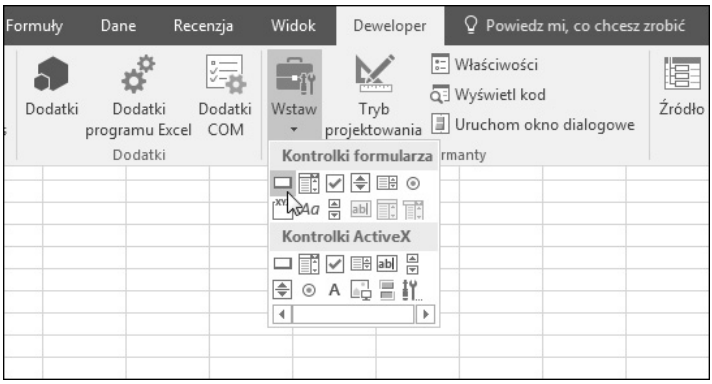
Pracując z makrami, bardzo często będziesz chciał, aby można je było uruchamiać w prosty i wygodny sposób. Jednym z najlepszych rozwiązań jest zazwyczaj utworzenie w skoroszycie przycisku, którego naciśnięcie będzie uruchamiało wybrane makro.

Na szczęście Excel oferuje cały szereg różnego rodzaju formantów pozwalających na tworzenie interfejsów użytkownika bezpośrednio na arkuszu. Istnieje wiele różnych formantów, począwszy od prostych przycisków (to chyba najczęściej wykorzystywany formant), aż do pasków przewijania.

Koncepcja używania formantów jest bardzo prosta. Umieszczasz wybrany formant bezpośrednio na arkuszu, a następnie przypisujesz do niego odpowiednie makro, które zostanie automatycznie uruchomione po kliknięciu lub naciśnięciu takiego formantu.

Aby to wypróbować, utworzymy teraz na arkuszu przycisk i następnie przypiszemy do niego utworzone wcześniej makro `DodajPodsumowanieWzględne`. Wykonaj następujące polecenia:

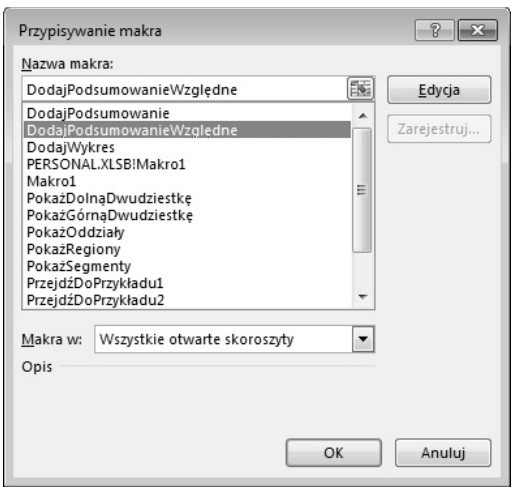
1. Przejdź na kartę *Deweloper* i naciśnij przycisk *Wstaw* (patrz rysunek 2.7).



Rysunek 2.7. Listę dostępnych formantów znajdziesz na karcie Deweloper

2. Na liście dostępnych kontrolek odszukaj i naciśnij ikonę formantu *Przycisk (formant formularza)*.
3. Kliknij na arkuszu miejsce, w którym chcesz umieścić przycisk.

   Kiedy umieścisz przycisk na arkuszu, na ekranie pojawi się okno dialogowe *Przypisywanie makra*, za pomocą którego możesz przypisać do przycisku wybrane makro (patrz rysunek 2.8).



Rysunek 2.8. Przypisywanie wybranego makra do przycisku

4. Wybierz z listy makro, które chcesz przypisać do przycisku, i naciśnij przycisk *OK*.

Od tej chwili masz już gotowy przycisk, za pomocą którego możesz uruchamiać zarejestrowane wcześniej makro. Pamiętaj, że wszystkie formanty dostępne w grupie *Kontrolki formularza* działają podobnie — możesz do nich przypisywać dowolnie wybrane makra, które zostaną uruchomione po kliknięciu formantu lewym przyciskiem myszy.

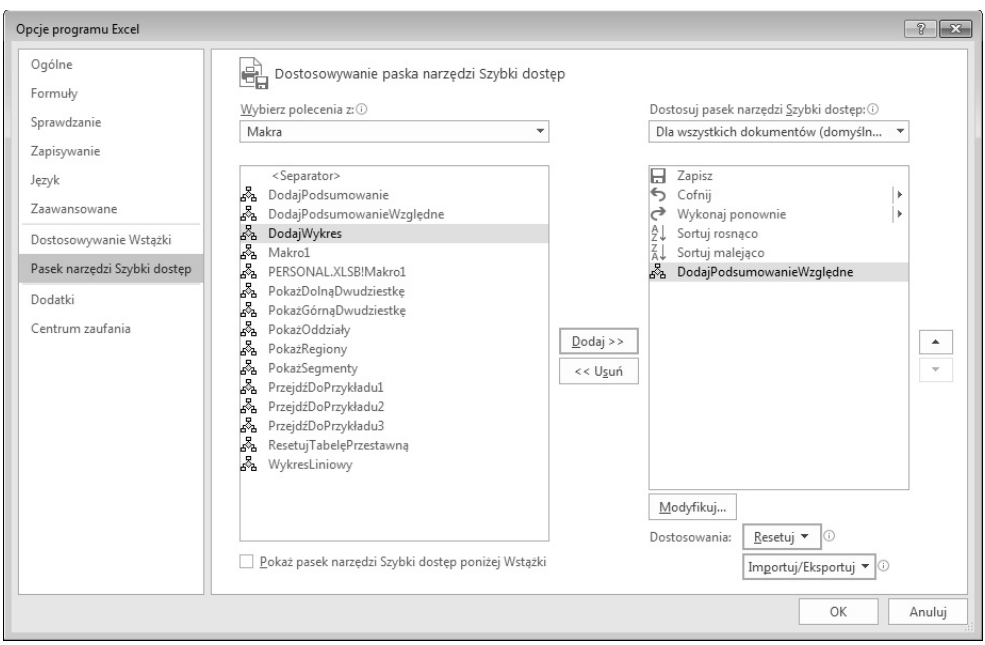
Uwaga

Zwróć uwagę, że formanty widoczne na rysunku 2.7 podzielone są na dwie grupy: *Kontrolki formularza* i *Kontrolki ActiveX*. Kontrolki formularza zostały zaprojektowane specjalnie do użycia bezpośrednio na arkuszach, podczas gdy kontrolki ActiveX są zazwyczaj używane na formularzach *UserForm*. Pracując na arkuszach, w zdecydowanej większości przypadków powinieneś używać kontrolek formularza. Dlaczego? Kontrolki formularza działają bardziej efektywnie, a ich konfiguracja jest zdecydowanie łatwiejsza niż ich odpowiedników — kontrolek ActiveX.

Umieszczanie makr na pasku narzędzi Szybki dostęp

Makra możesz również przypisywać do przycisków umieszczanych na pasku narzędzi *Szybki dostęp*, który w zależności od ustawień użytkownika znajduje się powyżej lub poniżej Wstążki. Aby umieścić na tym pasku nowy przycisk i przypisać do niego własne makro, powinieneś wykonać polecenia pokazane poniżej:

1. Kliknij pasek narzędzi *Szybki dostęp* prawym przyciskiem myszy i z menu podręcznego, które pojawi się na ekranie, wybierz polecenie *Dostosuj pasek narzędzi Szybki dostęp*. Na ekranie pojawi się okno opcji programu Excel, przedstawione na rysunku 2.9.



Rysunek 2.9. Dodawanie makra do paska narzędzi Szybki dostęp

2. W panelu po lewej stronie okna wybierz opcję *Pasek narzędzi Szybki dostęp*.
3. Rozwiń listę *Wybierz polecenia z:* i wybierz z niej opcję *Makra*.
4. Odszukaj i zaznacz wybrane makro, a następnie naciśnij przycisk *Dodaj*.
5. Naciśnij przycisk *Modyfikuj* i wybierz dla makra odpowiednią ikonę przycisku.
6. Naciśnij przycisk *OK*.

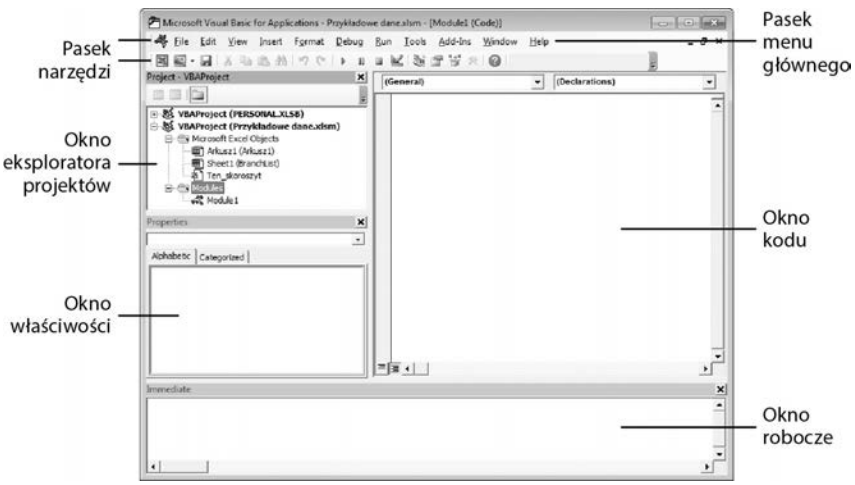
Praca z edytorem Visual Basic Editor (VBE)

Tworzenie i modyfikacja kodu języka VBA odbywa się przy użyciu edytora VBE (ang. *Visual Basic Editor*). Edytor Visual Basic jest co prawda aplikacją oddzielną, ale bardzo ściśle zintegrowaną z Excelem. Nie możesz uruchomić edytora Visual Basic bez uprzedniego uruchomienia programu Excel. Aby uruchomić edytor VBE, naciśnij kombinację klawiszy *Alt+F11*. Aby powrócić do Excela, ponownie naciśnij kombinację klawiszy *Alt+F11*.

Edytor VBE możesz również uruchomić, przechodząc na kartę *Deweloper* i naciskając przycisk *Visual Basic* znajdujący się w grupie opcji *Kod*.

Podstawowe elementy edytora VBE

Na rysunku 2.10 przedstawiono wygląd głównego okna edytora VBE. Oczywiście w Twoim przypadku okno edytora VBE może wyglądać nieco inaczej. Użytkownik ma duże możliwości dopasowywania wyglądu tego okna do własnych potrzeb — możesz ukrywać poszczególne okna składowe, zmieniać ich rozmiary, zmieniać układ okien na ekranie itd.



Rysunek 2.10. Okno edytora VBE

Pasek menu głównego edytora VBE

Pasek menu edytora VBE działa jak każdy inny tradycyjny pasek menu, z którym się do tej pory spotkałeś. Znajdziesz tam szereg poleceń wykorzystywanych w pracy z różnymi elementami składowymi edytora. Wiele poleceń posiada powiązane z nimi skróty klawiaturowe.

Edytor VBE oferuje również menu podręczne, w którym znajdziesz często używane polecenia. Aby przywołać menu na ekran, kliknij prawym przyciskiem myszy w dowolnym miejscu okna edytora VBE.

Paski narzędzi edytora VBE

Standardowy pasek narzędzi, domyślnie znajdujący się bezpośrednio pod paskiem menu, jest jednym z kilku dostępnych pasków narzędzi edytora VBE (pasek menu jest traktowany jako pasek narzędzi). Paski narzędzi edytora VBE możesz dostosowywać do własnych wymagań, przemieszczać w inne miejsca, wyświetlać i ukrywać w zależności od potrzeb. Aby wyświetlić lub ukryć wybrany pasek narzędzi, z menu głównego wybierz polecenie *View/Toolbars*.

Okno eksploratora projektów

W oknie *Project* znajdziesz diagram zawierający wszystkie skoroszyty aktualnie otwarte w Excelu (w tym także dodatki i skoroszyty ukryte), które możesz dowolnie zwijać i rozwijać według potrzeb. Każdy skoroszyt jest określany mianem *projektu*. Okno eksploratora projektów bardziej szczegółowo zostanie omówione w podrozdziale zatytułowanym „Tajemnice okna Project" w dalszej części tego rozdziału.

Jeżeli okno *Project* nie jest widoczne, naciśnij kombinację klawiszy *Ctrl+R* lub z menu głównego edytora VBE wybierz polecenie *View/Project Explorer* (widok/eksplorator projektów). Aby ukryć okno *Project*, naciśnij przycisk *Close* (zamknij), znajdujący się w prawej części paska tytułowego tego okna. Zamiast tego możesz również kliknąć prawym przyciskiem myszy dowolny obszar okna *Project* i z menu podręcznego, które pojawi się na ekranie, wybrać polecenie *Hide* (ukryj).

Okno kodu

Okno *Code* (czasem nazywane oknem *Module*) przechowuje kod źródłowy języka VBA. Aby otworzyć okno kodu dla wybranego obiektu, należy dwukrotnie kliknąć taki obiekt w oknie *Project*. Na przykład: aby otworzyć okno *Code* dla obiektu `Arkusz1`, powinieneś po prostu dwukrotnie kliknąć ten arkusz w oknie *Project*. Jeżeli dany obiekt nie posiada przypisanego kodu VBA, okno *Code* będzie puste.

Okno *Code* zostanie bardziej szczegółowo omówione w podrozdziale „Tajemnice okna Code" w dalszej części tego rozdziału.

Okno robocze (okno Immediate)

Okno *Immediate* jest bardzo użyteczne w przypadku bezpośredniego wykonywania i testowania poleceń języka VBA oraz usuwania błędów z kodu źródłowego. Okno może być widoczne lub ukryte. Jeżeli okno *Immediate* jest niewidoczne, powinieneś nacisnąć kombinację

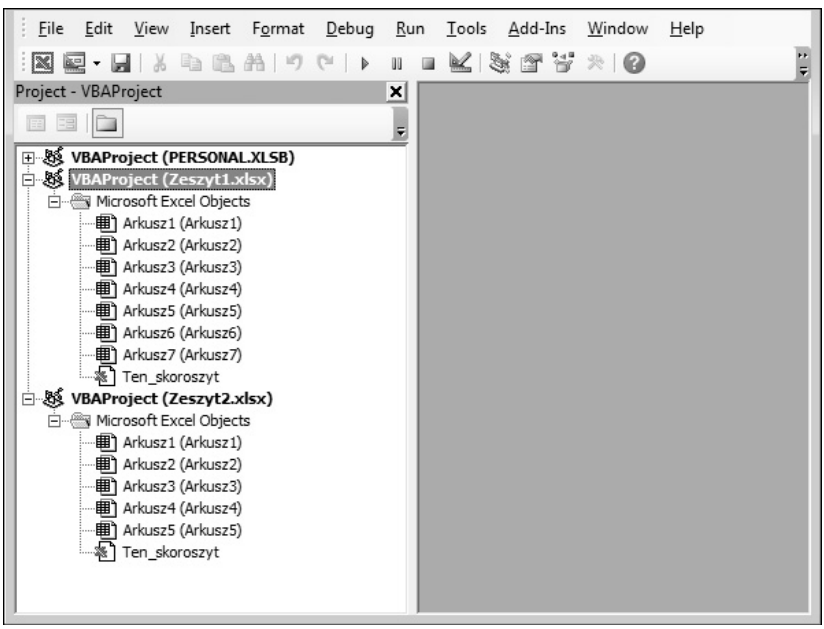
klawiszy *Ctrl+G*. Aby je zamknąć, kliknij przycisk *Zamknij* na pasku tytułowym. Zamiast tego możesz również prawym przyciskiem myszy kliknąć w dowolnym miejscu okna *Immediate* i z menu podręcznego wybrać polecenie *Hide*.

Okno *Immediate* jest bardzo użyteczne — nadaje się zwłaszcza do bezpośredniego wykonywania poleceń VBA oraz testowania i wyszukiwania błędów w kodzie. Jeżeli dopiero zaczynasz swoją przygodę z językiem VBA, okno *Immediate* nie będzie Ci zbyt potrzebne, więc możesz je po prostu ukryć.

Tajemnice okna Project

Kiedy pracujesz z edytorem VBE, każdy otwarty skoroszyt i dodatek Excela jest traktowany jako projekt. *Projekt* można potraktować jako zbiór obiektów uporządkowanych na wzór konspektu. Aby rozwinąć zawartość projektu, należy kliknąć ikonę ze znakiem plus, znajdującą się z lewej strony nazwy projektu w oknie *Project*. W celu zwinięcia zawartości projektu należy kliknąć ikonę ze znakiem minus, widoczną z lewej strony nazwy projektu. Jeżeli spróbujesz rozwinąć zawartość projektu chronionego hasłem, zostaniesz poproszony o wpisanie hasła.

Na rysunku 2.11 pokazano okno *Project* zawierające trzy projekty (skoroszyt makr osobistych, skoroszyt o nazwie *Zeszyt1* oraz skoroszyt o nazwie *Zeszyt2*).



Rysunek 2.11. W oknie Project widoczne są trzy projekty; dwa z nich mają rozwiniętą listę obiektów

W każdym projekcie rozwiniętym w oknie *Project* wyświetlany jest przynajmniej jeden węzeł o nazwie *Microsoft Excel Objects*. Po jego rozwinięciu będą widoczne poszczególne arkusze i arkusze wykresu zawarte w danym skoroszycie (każdy arkusz jest traktowany jak osobny obiekt). Dodatkowo wyświetlany jest również obiekt o nazwie `ThisWorkbook`, który reprezentuje obiekt `Workbook`. Jeżeli projekt zawiera moduły VBA, na liście pojawi się też węzeł *Modules* reprezentujący moduły kodu VBA.

Dodawanie nowego modułu kodu VBA

Kiedy rejestrujesz makro, Excel automatycznie wstawia odpowiedni moduł VBA zawierający wygenerowany kod źródłowy. Skoroszyt, w którym przechowywany jest rejestrowany kod, zależy od docelowej lokalizacji makra, którą wybrałeś w oknie dialogowym przed rozpoczęciem rejestrowania.

Ogólnie rzecz biorąc, w oknie kodu możesz przechowywać trzy rodzaje kodu języka VBA:

- **Deklaracje**. *Deklaracja* to informacja na temat zmiennej zastosowanej w kodzie źródłowym języka VBA. Możesz na przykład zadeklarować typ danych dla zmiennych, których planujesz użyć w procedurze czy funkcji.
- **Procedury `Sub`**. Jak pamiętasz, *procedura* to inaczej zbiór poleceń, które wykonują określone operacje. Wszystkie makra są rejestrowane w postaci procedur `Sub`.
- **Procedury `Function`** (funkcje). *Funkcja* jest zbiorem instrukcji zwracających pojedynczą wartość lub tablicę (pod względem działania przypominają funkcje arkusza, na przykład funkcję `SUMA`).

W pojedynczym module VBA możesz przechowywać dowolną liczbę procedur, funkcji oraz deklaracji. Sposób zorganizowania modułu VBA jest całkowicie zależny od Ciebie. Niektórzy programiści umieszczają cały kod VBA aplikacji w jednym module, z kolei inni preferują podział i przechowanie poszczególnych części kodu w kilku różnych modułach.

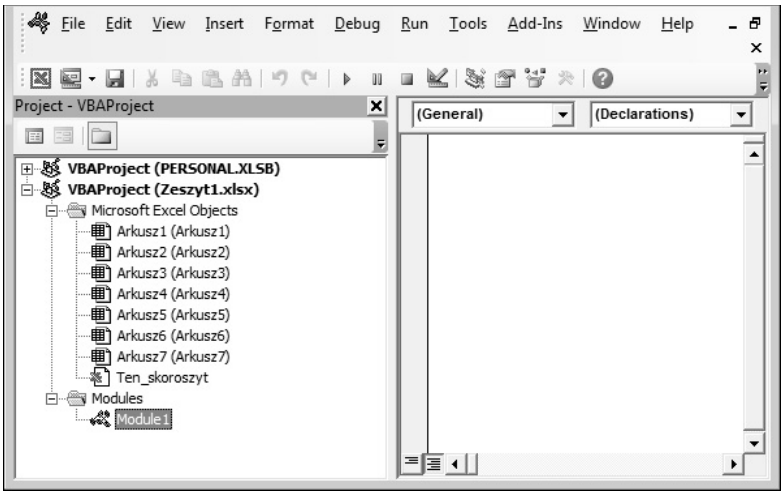
Aby dodać nowy moduł kodu VBA do projektu, powinieneś wykonać następujące polecenia:

1. Zaznacz w oknie *Project* nazwę wybranego projektu.
2. Z menu głównego edytora VBE wybierz polecenie *Insert/Module* (wstaw/moduł).

Inny sposób:

1. Kliknij prawym przyciskiem myszy nazwę wybranego projektu.
2. Z menu podręcznego wybierz polecenie *Insert/Module*.

   Nowy moduł jest dodawany w folderze *Modules* projektu (patrz rysunek 2.12). Wszystkie inne moduły kodu VBA będą również umieszczane w tym folderze.



Rysunek 2.12. Moduły kodu umieszczane są w folderze Modules

USUWANIE MODUŁU KODU VBA

Jeżeli chcesz usunąć z danego projektu moduł kodu VBA, który nie jest już potrzebny, powinieneś wykonać polecenia opisane poniżej:

1. Zaznacz w oknie *Project* nazwę wybranego projektu.
2. Z menu głównego edytora VBE wybierz polecenie *File/Remove **xxx*** (plik/usuń *xxx*), gdzie *xxx* to nazwa usuwanego modułu.

Inny sposób:

1. Kliknij prawym przyciskiem myszy nazwę modułu, który chcesz usunąć.
2. Z menu podręcznego wybierz polecenie *Remove **xxx***.

UWAGA

Możesz usuwać moduły kodu VBA, ale nie masz możliwości usunięcia modułów kodu źródłowego, które są powiązane ze skoroszytem (np. moduł kodu `ThisWorkbook`) lub arkuszem (np. moduł kodu dla obiektu `Arkusz1`).

Tajemnice okna Code

Gdy już nabierzesz wprawy w korzystaniu z języka VBA, będziesz spędzał *mnóstwo* czasu na pracy z oknami *Code* (okno kodu źródłowego). W modułach kodu zapisywane są rejestrowane makra, ale możesz również ręcznie wpisywać w nich kod procedur.

Minimalizacja i maksymalizacja okien kodu

Okna kodu zachowują się bardzo podobnie jak okna arkuszy Excela. Możesz je minimalizować, maksymalizować, ukrywać, zmieniać ich położenie itd. Wiele osób twierdzi, że najefektywniej pracuje się z oknami kodu rozciągniętymi do maksymalnych rozmiarów. Dzięki takiemu rozwiązaniu możesz zobaczyć większą ilość kodu źródłowego i skupić się na wykonywanym zadaniu.

Aby zmaksymalizować okno *Code*, naciśnij przycisk *Maksymalizuj* znajdujący się na pasku tytułowym okna lub dwukrotnie kliknij pasek tytułu. Aby przywrócić poprzednią wielkość okna *Code*, naciśnij przycisk *Przywróć okno* znajdujący się poniżej paska tytułowego okna aplikacji.

Czasem jednak bardzo korzystne może być jednoczesne wyświetlenie dwóch lub większej liczby okien *Code*, na przykład kiedy chcesz porównać kod źródłowy zawarty w dwóch modułach lub przekopiować kod z jednego modułu do drugiego. Aby to zrobić, możesz zmienić wielkość i położenie okien ręcznie albo skorzystać z automatycznej zmiany układu, wybierając z menu głównego edytora VBE polecenie *Window/Tile Horizontaly* lub *Window/Tile Vertically*.

Aby szybko przechodzić między kolejnymi oknami kodu, powinieneś naciskać kombinację klawiszy *Ctrl+Tab*. Aby przechodzić między oknami kodu w odwrotnej kolejności, naciskaj kombinację klawiszy *Ctrl+Shift+Tab*.

Minimalizacja okna *Code* spowoduje jego ukrycie. Aby całkowicie zamknąć okno *Code*, naciśnij przycisk *Zamknij* znajdujący się na pasku tytułowym okna. Zamknięcie okna powoduje wyłącznie jego ukrycie — zamykając okno, nie utracisz w żaden sposób przechowywanego w nim kodu. Aby ponownie otworzyć okno kodu, wystarczy dwukrotnie kliknąć odpowiedni obiekt w oknie *Project*.

Wprowadzanie kodu języka VBA do modułu

Zanim będziesz mógł czegokolwiek dokonać za pomocą języka VBA, musisz w oknie *Code* umieścić odpowiedni kod programu, mający postać procedury. Kod programu możesz umieścić w module VBA na trzy sposoby:

- Użyj rejestratora makr programu Excel, który umożliwia zarejestrowanie wykonywanych operacji i automatyczną zamianę na odpowiedni kod źródłowy języka VBA.
- Wprowadź kod źródłowy bezpośrednio przy użyciu klawiatury.
- Skopiuj kod z innego modułu i wklej go do modułu, nad którym pracujesz.

Zapoznałeś się już z rewelacyjnymi możliwościami, jakie daje tworzenie kodu VBA za pomocą rejestratora makr. Niestety istnieją operacje, których nie można przekształcić na kod VBA za pomocą rejestratora makr. W takiej sytuacji jedynym rozwiązaniem pozostaje ręczne utworzenie odpowiedniego kodu. Innymi słowy, musisz po prostu ręcznie „wklepać" odpowiedni kod programu lub po prostu skopiować i wkleić kod pochodzący z innego źródła (na przykład ze strony internetowej).

Wprowadzanie i edytowanie kodu źródłowego modułu VBA wygląda dokładnie tak, jak mogłeś się tego spodziewać — kod programu możesz zaznaczać, kopiować lub wycinać, a następnie wklejać w inne miejsce.

Pojedynczy wiersz kodu VBA może być tak długi, jak tylko będzie to potrzebne, aczkolwiek dla zachowania czytelności kodu długą instrukcję warto podzielić na dwie lub większą liczbę wierszy. Aby to zrobić, na końcu wiersza po spacji powinieneś wstawić znak podkreślenia, nacisnąć klawisz *Enter* i ciąg dalszy instrukcji umieścić w następnym wierszu. Oto przykład pojedynczego polecenia umieszczonego w trzech wierszach:

```
Selection.Sort Key1:=Range("A1"), _
  Order1:=xlAscending, Header:=xlGuess, _
  Orientation:=xlTopToBottom
```

Tak podzielone polecenie będzie działać w dokładnie taki sam sposób jak identyczne polecenie umieszczone w jednym, długim wierszu (oczywiście w takim wypadku z pominięciem znaków kontynuacji wiersza). Zwróć uwagę, że drugi i trzeci wiersz polecenia zostały wcięte. Wcięcia są opcjonalne, ale pozwalają szybko zorientować się, że te wszystkie wiersze rzeczywiście stanowią jedno wyrażenie.

Edytor VBE posiada efektywny mechanizm wycofywania wprowadzonych zmian, pozwalający na wycofanie lub ponowne wykonanie wielu kolejnych poleceń. Jeżeli przypadkowo usuniesz jakiś wiersz polecenia, który jest Ci potrzebny, możesz nacisnąć przycisk *Undo* (wycofaj), znajdujący się na pasku narzędzi edytora VBE, lub po prostu nacisnąć kombinację klawiszy *Ctrl+Z*, co spowoduje przywrócenie skasowanego wiersza. Po wycofaniu dowolnego polecenia lub serii poleceń aktywuje się przycisk *Redo* (wykonaj ponownie), który pozwala na ponowne wykonanie wycofanych poleceń.

Aby przekonać się, jak w praktyce wygląda tworzenie kodu procedury VBA, wykonaj polecenia opisane poniżej:

1. Utwórz w Excelu nowy skoroszyt.
2. Przejdź do edytora VBE, naciskając kombinację klawiszy *Alt+F11*.
3. W oknie *Project* kliknij nazwę nowo utworzonego skoroszytu.
4. Wstaw do projektu nowy moduł VBA, wybierając z menu głównego edytora polecenie *Insert/Module* (wstaw/moduł).

5. W oknie kodu wpisz procedurę VBA przedstawioną poniżej:

```
Sub GuessName()
    Dim Msg as String
    Dim Ans As Long
    Msg = "Czy nazywasz się " & Application.UserName & "?"
    Ans = MsgBox(Msg, vbYesNo)
    If Ans = vbNo Then MsgBox "No cóż, każdy może się pomylić."
    If Ans = vbYes Then MsgBox "Jestem jasnowidzem!"
End Sub
```

6. Upewnij się, że kursor znajduje się w dowolnym miejscu kodu, i uruchom procedurę, naciskając klawisz *F5*.

WSKAZÓWKA

Klawisz *F5* spełnia rolę skrótu klawiszowego dla polecenia *Run/Run Sub/UserForm* (uruchom/uruchom procedurę lub formularz *UserForm*) z menu głównego edytora VBE.

Wprowadzając kod procedury, z pewnością zauważyłeś, że edytor VBE automatycznie dokonuje drobnych poprawek i korekt wpisywanego kodu. Kiedy na przykład wpiszesz polecenie `Sub` rozpoczynające procedurę, edytor VBE automatycznie je dopełni, wstawiając polecenie `End Sub` kończące procedurę. Podobnie jeżeli pominiesz spację przed lub po znaku równości, edytor VBE automatycznie wstawi pominiętą spację. Oprócz tego edytor VBE zmienia kolor czcionki słów kluczowych języka VBA oraz zmienia wielkość liter wybranych fragmentów tekstu. Jest to całkowicie normalne i pożądane zachowanie, dzięki któremu VBE ułatwia Ci utrzymywanie porządku i przejrzystości wprowadzanego kodu.

Jeżeli poprawnie wykonałeś wszystkie opisane wyżej polecenia, utworzyłeś właśnie swoją pierwszą procedurę `Sub` języka VBA (nazywaną też *makrem*). Po wykonaniu polecenia uaktywniającego makro (czyli naciśnięciu klawisza *F5*), edytor Visual Basic szybko skompilował kod źródłowy i uruchomił procedurę. Innymi słowy, poszczególne polecenia zostały przetworzone i Excel po prostu wykonał to, co mu nakazano. Nasze przykładowe makro możesz uruchomić dowolną liczbę razy, chociaż po jakimś czasie cała zabawa z pewnością nie będzie już tak atrakcyjna…

Nasza prosta procedura składa się z następujących elementów:

- Deklaracja procedury `Sub` (pierwszy wiersz).
- Deklaracje zmiennych (polecenia `Dim`).
- Przypisywanie wartości do zmiennych (zmienne `Msg` i `Ans`).
- Wykonanie operacji łączenia łańcuchów tekstu (przy użyciu operatora `&`).
- Korzystanie z wbudowanych funkcji języka VBA (`MsgBox`).
- Używanie wbudowanych stałych języka VBA (`vbYesNo`, `vbNo` oraz `vbYes`).
- Używanie instrukcji warunkowej `If-Then-Else` (dwukrotnie).
- Zakończenie procedury `Sub` (ostatni wiersz).

Jak już wspominaliśmy wcześniej, inną metodą umieszczania kodu w module VBA jest skopiowanie kodu z zewnętrznego źródła i wklejenie go do modułu. Wyobraź sobie, że w ramach jednego projektu utworzyłeś procedurę, która może być przydatna również w innym projekcie. Zamiast tracić czas na ponowne, mozolne wprowadzanie takiego samego kodu źródłowego, możesz po prostu otworzyć odpowiedni skoroszyt, uaktywnić wybrany moduł, skopiować kod wybranej procedury (naciskając kombinację klawiszy *Ctrl+C*) i następnie wkleić go do aktualnie tworzonego modułu VBA (za pomocą kombinacji klawiszy *Ctrl+V*). W razie konieczności po wklejeniu kodu można go zmodyfikować i dopasować do wymagań bieżącego projektu.

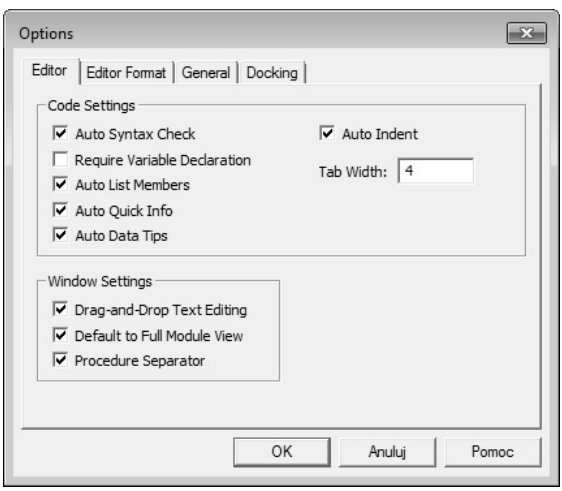
Dostosowywanie środowiska edytora Visual Basic

Jeżeli naprawdę chcesz zostać programistą tworzącym aplikacje Excela, musisz przyzwyczaić się do myśli, że będziesz spędzał mnóstwo czasu pracując z edytorem Visual Basic. Aby ułatwić życie programisty, edytor VBE posiada całkiem sporo opcji pozwalających na dopasowanie go do indywidualnych wymagań użytkownika.

Po uruchomieniu edytora Visual Basic z menu *Tools* (narzędzia) wybierz polecenie *Options* (opcje). Na ekranie pojawi się okno dialogowe zawierające cztery karty — *Editor* (ustawienia edytora), *Editor Format* (ustawienia formatowania), *General* (opcje ogólne) oraz *Docking* (ustawienia dokowania okien). W kolejnych podrozdziałach omówimy kilka najważniejszych opcji znajdujących się na tych kartach.

Karta Editor

Na rysunku 2.13 przedstawiono opcje znajdujące się na karcie *Editor* okna dialogowego *Options*, za pomocą których możesz zmieniać sposób działania wybranych mechanizmów edytora VBE.



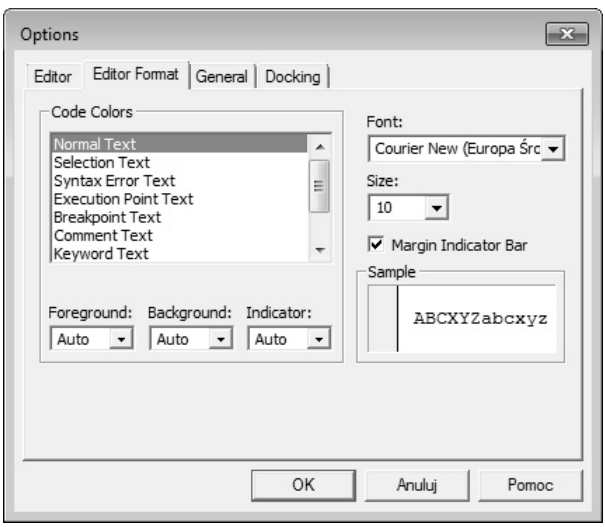
Rysunek 2.13. Karta Editor okna dialogowego Options

- **Opcja *Auto Syntax Check* —** opcja *Auto Syntax Check* określa, czy edytor Visual Basic wyświetli okno dialogowe po wykryciu błędu składni w trakcie wprowadzania kodu źródłowego języka VBA. Okno dialogowe zawiera przybliżoną informację o zaistniałym problemie. Jeżeli opcja *Auto Syntax Check* zostanie wyłączona, edytor Visual Basic oznaczy niepoprawny kod innym kolorem niż pozostałą część kodu, a okno dialogowe z komunikatem o błędzie nie będzie pojawiało się na ekranie. Jeżeli dopiero zaczynasz uczyć się języka VBA, opcja *Auto Syntax Check* może okazać się bardzo pomocna.
- **Opcja *Require Variable Declaration* —** jeżeli opcja *Require Variable Declaration* zostanie włączona, edytor Visual Basic na początku każdego tworzonego modułu VBA wstawi następujące polecenie: `Option Explicit`. Włączenie tej opcji nie ma wpływu na istniejące moduły kodu, ale polecenie będzie wstawiane we wszystkich nowych modułach, które utworzysz od momentu włączenia tej opcji. Jeżeli takie polecenie pojawi się w module, będziesz musiał jawnie zadeklarować wszystkie zmienne używane w tym module. Jednym ze sposobów deklarowania zmiennych jest używanie polecenia `Dim`.
- **Opcja *Auto List Members* —** po włączeniu opcji *Auto List Members* edytor Visual Basic będzie oferował pomoc w trakcie wprowadzania kodu źródłowego języka VBA. Pomoc edytora polega na wyświetlaniu listy elementów powiązanych z obiektem, takich jak metody i właściwości użytego obiektu. Jest to jeden z najbardziej przydatnych mechanizmów edytora VBE.
- **Opcja *Auto Quick Info* —** po włączeniu opcji *Auto Quick Info* edytor Visual Basic wyświetla informacje o argumentach wprowadzanych funkcji, właściwości i metod. Sposób działania tego mechanizmu jest bardzo podobny do sposobu, w jaki Excel podpowiada składnię argumentów funkcji podczas wpisywania formuł w komórkach arkusza.
- **Opcja *Auto Data Tips* —** po włączeniu opcji *Auto Data Tips* możesz ustawić wskaźnik myszy nad wybraną zmienną, a edytor VBE wyświetli na ekranie jej wartość. Pamiętaj jednak, że mechanizm ten działa tylko wtedy, kiedy wystąpi błąd, działanie procedury zostanie wstrzymane i wejdziesz w tryb wyszukiwania i usuwania błędów (ang. *debugging*). Jest to bardzo użyteczna opcja i naprawdę nie ma żadnego powodu, aby ją kiedykolwiek wyłączać.
- **Opcja *Auto Indent* —** opcja *Auto Indent* określa, czy edytor Visual Basic automatycznie wstawi na początku każdego nowego wiersza identyczne wcięcie jak w przypadku wcześniej wprowadzonego. Zdecydowana większość programistów VBA jest wielkimi zwolennikami stosowania wcięć w kodzie źródłowym, stąd opcja ta zazwyczaj bywa włączona.
- **Opcja *Drag-and-Drop Text Editing* —** włączenie opcji *Drag-and-Drop Text Editing* umożliwia kopiowanie i przenoszenie tekstu myszą za pomocą metody przeciągnij i upuść.

- Opcja ***Default to Full Module View*** — opcja *Default to Full Module View* określa sposób przeglądania procedur. Po jej włączeniu procedury zawarte w oknie kodu będą wyświetlane w pojedynczym, przewijanym oknie. Po wyłączeniu tej opcji w danej chwili widoczna będzie tylko jedna procedura.
- Opcja ***Procedure Separator*** — kiedy opcja *Procedure Separator* jest aktywna, na końcu każdej procedury zawartej w oknie *Code* wyświetlany będzie pasek separatora (przy założeniu oczywiście, że włączona jest również opcja *Default to Full Module View*). Zwykle dobrze jest widzieć, gdzie kończy się kod danej procedury, dlatego zawsze warto pozostawić tę opcję włączoną.

Karta Editor Format

Na rysunku 2.14 przedstawiono kartę *Editor Format* okna dialogowego *Options*. Opcje zamieszczone na tej karcie odpowiadają za wygląd edytora VBE.



Rysunek 2.14. Karta Editor Format okna dialogowego Options

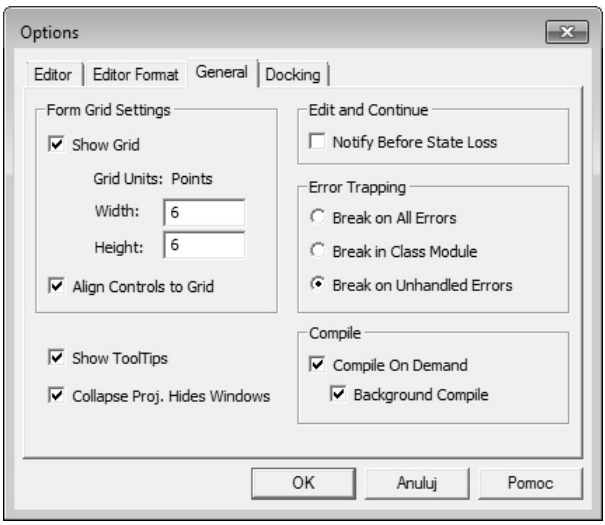
- Opcja ***Code Colors*** — opcja *Code Colors* umożliwia wybór koloru tekstu (tła i pierwszego planu) oraz kolorów powiązanych z różnymi elementami kodu źródłowego języka VBA. W tym przypadku główną rolę odgrywają indywidualne upodobania każdego użytkownika. Ze swojej strony uważam, że kolory domyślne są całkiem odpowiednie, ale pomimo to od czasu do czasu, chcąc zmienić scenerię, możesz zmienić ustawienia tej opcji.
- Opcja ***Font*** — opcja *Font* umożliwia wybranie czcionki, która zostanie użyta w modułach VBA. Aby uzyskać jak najlepsze wyniki, powinieneś raczej pozostać przy *czcionce o stałej szerokości*, takiej jak na przykład Courier New.

W takich czcionkach wszystkie znaki mają dokładnie taką samą szerokość. Takie rozwiązanie powoduje, że kod źródłowy jest bardziej czytelny, ponieważ znaki są ładnie wyrównane w pionie i bardzo łatwo można odnaleźć powielane spacje (co w pewnych sytuacjach może być bardzo użyteczne).

- **Pole *Size*** — pole *Size* pozwala określić rozmiar czcionki stosowanej w modułach VBA. Wielkość czcionki zależy zwykle od indywidualnych upodobań użytkownika oraz używanej rozdzielczości obrazu.
- **Opcja *Margin Indicator Bar*** — opcja *Margin Indicator Bar* decyduje o tym, czy będzie wyświetlany pasek wskaźnika pionowego marginesu. Opcja powinna być włączona. W przeciwnym razie podczas pracy w trybie wykrywania i usuwania błędów w kodzie źródłowym nie zobaczysz przydatnych graficznych wskaźników szerokości marginesu.

Karta General

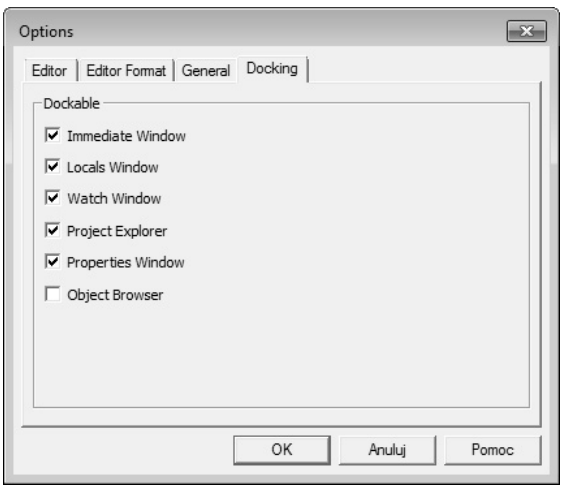
Na rysunku 2.15 przedstawiono opcje znajdujące się na karcie *General* okna dialogowego *Options*. W zdecydowanej większości przypadków ustawienia domyśle będą w zupełności wystarczające. Najważniejszymi opcjami na tej karcie są opcje grupy *Error Trapping* (przechwytywanie i obsługa błędów). Jeżeli dopiero rozpoczynasz swoją karierę jako programista makr Excela, to najlepszym rozwiązaniem będzie wybranie opcji *Break on Unhandled Errors*, która powoduje, że Excel będzie przerywał działanie procedury i wyświetlał odpowiedni komunikat po napotkaniu błędu nieposiadającego w kodzie swojej procedury obsługi.



Rysunek 2.15. Zakładka General okna dialogowego Options

Karta Docking

Na rysunku 2.16 przedstawiono kartę *Docking* okna dialogowego *Options*. Opcje znajdujące się na tej karcie pozwalają określić zachowanie różnych okien edytora Visual Basic. Po zadokowaniu okno znajduje się przy jednej z krawędzi głównego okna edytora Visual Basic, dzięki czemu znacznie łatwiej można je zidentyfikować i zlokalizować. Gdy wszystkie opcje znajdujące się na karcie *Docking* zostaną wyłączone, okna będą nieuporządkowane i na ekranie powstanie niezłe zamieszanie. Zazwyczaj domyślne ustawienia są uznawane za odpowiednie.



Rysunek 2.16. Karta Docking okna dialogowego Options

Podstawowe informacje o języku VBA

Visual Basic for Applications (VBA) jest językiem programowania zorientowanym obiektowo. Podstawowa koncepcja programowania zorientowanego obiektowo polega na tym, że każda aplikacja (w naszym przypadku Excel) składa się z szeregu różnych obiektów, z których każdy posiada swój własny zestaw cech i zastosowań. Excel składa się ze skoroszytów, arkuszy, komórek, wykresów, tabel przestawnych, kształtów i wielu innych obiektów. Każdy z obiektów posiada swój zestaw cech nazywanych **właściwościami** oraz zestaw zastosowań nazywanych **metodami**.

Taką koncepcję łatwo sobie wyobrazić, ponieważ podobnymi cechami wykazują się wszystkie obiekty, z którymi spotykasz się w codziennym życiu, takie jak Twój komputer, samochód czy lodówka. Każdy z tych obiektów posiada swoje cechy pozwalające na jego identyfikację, takie jak wysokość, waga czy kolor. Każdy z obiektów ma również swoje zastosowanie:

komputera używasz do pracy z Excelem, samochodu używasz do przenoszenia się w inne miejsca, a lodówki do przechowywania żywności w warunkach chłodniczych.

Obiekty VBA również posiadają swoje charakterystyczne właściwości i metody. Komórka arkusza jest obiektem i posiada takie właściwości jak adres, wysokość, kolor wypełnienia i tak dalej. Skoroszyt również jest obiektem VBA, a wśród metod można wymienić otwieranie, zamykanie czy możliwość dodania do skoroszytu wykresu lub tabeli przestawnej.

Pracując w Excelu, codziennie będziesz się stykał ze skoroszytami, arkuszami czy zakresami komórek. Najprawdopodobniej traktujesz te wszystkie wymienione „obiekty" po prostu jako część Excela, tak naprawdę bez separowania ich od siebie. Excel traktuje wszystkie obiekty jako część hierarchicznego modelu, nazywanego **modelem obiektowym Excela**. Model obiektowy Excela jest zdefiniowany jako zestaw obiektów, które są zorganizowane według relacji między nimi.

Obiekty

W rzeczywistym świecie wszystko, co widzisz, możesz opisać jako obiekt. Kiedy patrzysz na swój dom, widzisz obiekt. Twój dom posiada pokoje, a każdy z nich to osobny obiekt. W pokojach stoją szafy, każda z nich to osobny obiekt. Jeżeli teraz pomyślisz o domu, jego pokojach i stojących w nich szafach, zobaczysz gotowy, hierarchiczny model relacji między tymi obiektami. Excel działa dokładnie w taki sam sposób.

W Excelu obiektem nadrzędnym dla wszystkich innych jest obiekt `Application`, będący w Excelu odpowiednikiem Twojego domu. Wewnątrz obiektu `Application` Excel posiada skoroszyty. Wewnątrz każdego skoroszytu znajdują się arkusze, a w każdym arkuszu — zakresy i komórki. Wszystkie wymienione obiekty znajdują się w strukturze hierarchicznej.

Aby w języku VBA wskazać wybrany obiekt Excela, musisz wymienić kolejno wszystkie poziomy hierarchii modelu obiektowego. Aby na przykład zaznaczyć komórkę `A1` na arkuszu `Arkusz1`, możesz użyć następującego wiersza kodu:

```
Application.ThisWorkbook.Sheets("Sheet1").Range("A1").Select
```

W większości przypadków hierarchia modelu obiektowego jest na tyle jednoznaczna, że nie musisz podawać nazw wszystkich kolejnych poziomów modelu. Przykładowo wykonanie polecenia przedstawionego poniżej również spowoduje zaznaczenie komórki o adresie `A1`, ponieważ Excel domyślnie założył, że miałeś na myśli aktywny arkusz bieżącego skoroszytu.

```
Range("A1").Select
```

Jeśli jednak kursor znajdował się już wcześniej w komórce `A1`, to mogłeś po prostu użyć obiektu `ActiveCell`, całkowicie eliminując konieczność definiowania adresu komórki.

```
Activecell.Select
```

Kolekcje

Wiele obiektów Excela należy do kolekcji. Twój dom stoi zapewne w otoczeniu innych domów, które stanowią kolekcję domów o nazwie osiedle. Każde osiedle jest częścią kolekcji osiedli tworzących kolejną kolekcję o nazwie miasto. Excel traktuje kolekcje jako osobne obiekty.

W każdym skoroszycie (obiekt `Workbook`) znajduje się kolekcja arkuszy (kolekcja o nazwie `Worksheets`). Kolekcja `Worksheets` również jest obiektem, do którego możesz się odwoływać za pomocą kodu VBA. Każdy arkusz w skoroszycie jest częścią kolekcji `Worksheets`.

Jeżeli chcesz odwołać się do danego arkusza w kolekcji `Worksheets`, możesz to zrobić za pomocą pozycji tego arkusza w kolekcji, podając jego numer indeksu (począwszy od 1), lub za pomocą nazwy arkusza ujętej w znaki cudzysłowu. Jeżeli w danym skoroszycie znajduje się tylko jeden arkusz i nosi on nazwę `MójArkusz`, to oba polecenia VBA przedstawione poniżej będą miały taki sam efekt:

```
Worksheets(1).Select
Worksheets("MójArkusz").Select
```

Jeżeli w danym skoroszycie znajdują się dwa arkusze, jeden o nazwie `MójArkusz`, a drugi o nazwie `TwójArkusz`, i są umieszczone w tej właśnie kolejności, to do drugiego z tych arkuszy możesz odwołać się za pomocą dowolnego z poleceń przedstawionych poniżej:

```
Worksheets(2).Select
Worksheets("YourSheet").Select
```

Jeżeli chciałbyś się odwołać do arkusza o nazwie `MójArkusz`, znajdującego się w nieaktywnym skoroszycie o nazwie *MojeDane.xlsx*, to będziesz musiał użyć pełnego, kwalifikowanego odwołania do skoroszytu i arkusza, tak jak to zostało pokazane poniżej:

```
Workbooks("MojeDane.xlsx").Worksheets("MójArkusz").Select
```

Właściwości

Każdy obiekt ma swoje właściwości, które określają jego charakterystykę. Twój dom ma określony kolor, powierzchnię, wiek i tak dalej. Niektóre właściwości można zmieniać, tak jak na przykład kolor domu. Inne właściwości nie mogą być zmieniane, na przykład rok, w którym Twój dom został wybudowany.

Podobnie sprawy mają się w Excelu. Obiekt Excela, taki jak `Worksheet` (arkusz), posiada właściwość `Name` (nazwa), którą możesz dowolnie zmieniać, ale już właściwość `Rows.Count` (liczba wierszy) nie może być zmieniana.

Do wybranej właściwości obiektu odwołujemy się, podając najpierw nazwę obiektu, a następnie nazwę właściwości. Na przykład nazwę wybranego arkusza możesz zmienić poprzez zmianę wartości jego właściwości `Name`.

W przykładzie przedstawionym poniżej zmienimy nazwę arkusza z `Arkusz1` na `MójArkusz`:

```
Sheets("Arkusz1").Name = "MójArkusz"
```

Niektóre właściwości są przeznaczone tylko do odczytu, co oznacza, że nie możesz bezpośrednio przypisywać do nich żadnych wartości. Przykładem takiej sytuacji może być właściwość `Text` komórki (czyli obiektu `Cell`). Właściwość `Text` udostępnia Ci sformatowaną wartość danej komórki, której jednak nie możesz w żaden sposób zmieniać ani nadpisywać.

Niektóre właściwości posiadają argumenty, które jeszcze dokładniej określają charakter wartości tej właściwości. Na przykład w wierszu kodu zamieszczonym poniżej używamy argumentów `RowAbsolute` i `ColumnAbsolute` do pobrania wartości adresu komórki `A1` w postaci adresu bezwzględnego (`$A$1`).

```
MsgBox Range("A1").Address(RowAbsolute:=True, ColumnAbsolute:=True)
```

Określanie właściwości aktywnego obiektu

Podczas pracy z Excelem za każdym razem może być aktywny tylko jeden skoroszyt. W takim skoroszycie w dowolnej chwili aktywny może być tylko jeden arkusz. Jeżeli arkusz aktywny jest arkuszem komórek, to w danej chwili aktywna może być tylko jedna komórka (nawet jeżeli zaznaczysz obszar składający się z wielu komórek). VBA zna hierarchiczną strukturę skoroszytów, arkuszy i komórek i dzięki temu pozwala na używanie uproszczonego formatu odwołań do aktywnych obiektów.

Takie rozwiązanie często się przydaje, ponieważ nie zawsze będziesz dokładnie wiedział, który konkretnie skoroszyt, arkusz lub zakres zostanie użyty. Język VBA znakomicie ułatwia takie odwołania dzięki wykorzystaniu właściwości obiektu `Application`. Przykładowo obiekt `Application` posiada właściwość `ActiveCell` zwracającą odwołanie do aktywnej komórki. Poniższa instrukcja przypisuje aktywnej komórce wartość `1`:

```
ActiveCell.Value = 1
```

Zwróć uwagę, że w powyższym przykładzie pominęliśmy odwołanie do obiektu `Application` oraz aktywnego arkusza. Pamiętaj jednak, że próba wykonania takiego polecenia zakończy się niepowodzeniem, jeżeli arkusz aktywny nie będzie arkuszem komórek. Jeżeli takie polecenie zostanie wykonane w chwili, kiedy aktywny będzie arkusz wykresu, to działanie procedury zostanie zatrzymane i zakończy się wyświetleniem odpowiedniego komunikatu o wystąpieniu błędu.

Jeżeli w danym arkuszu zaznaczymy zakres komórek, komórką aktywną będzie zawsze komórka należąca do zaznaczonego zakresu. Inaczej mówiąc, komórką aktywną jest zawsze pojedyncza komórka arkusza (a nigdy wielokomórkowy zakres).

Obiekt `Application` posiada również właściwość o nazwie `Selection`, która zwraca odwołanie do aktualnie zaznaczonego elementu, którym może być pojedyncza komórka (aktywna komórka), zakres komórek czy obiekt taki jak `ChartObject`, `TextBox` lub `Shape`.

W tabeli 2.1 zamieszczamy listę innych właściwości obiektu `Application`, które mogą być użyteczne podczas pracy z komórkami i zakresami komórek.

TABELA 2.1. NIEKTÓRE PRZYDATNE WŁAŚCIWOŚCI OBIEKTU APPLICATION

Właściwość	Zwracany obiekt
`ActiveCell`	Aktywna komórka
`ActiveChart`	Aktywny arkusz wykresu lub wykres powiązany z obiektem `ChartObject` znajdującym się w arkuszu. Jeżeli wykres nie jest aktywny, właściwość będzie miała wartość `Nothing`
`ActiveSheet`	Aktywny arkusz (zwykły lub wykresu)
`ActiveWindow`	Aktywne okno
`ActiveWorkbook`	Aktywny skoroszyt
`Selection`	Zaznaczony obiekt (mogą to być takie obiekty jak `Range`, `Shape`, `ChartObject` itp.).
`ThisWorkbook`	Skoroszyt zawierający wykonywaną procedurę VBA. Ten obiekt może (ale nie musi) być taki sam jak obiekt `ActiveWorkbook`.

Zaletą stosowania wymienionych właściwości zwracających obiekty jest to, że nie musisz wiedzieć, która komórka, arkusz lub skoroszyt są aktualnie aktywne, ani też nie musisz tworzyć do nich odwołania. Dzięki temu możesz tworzyć kod źródłowy języka VBA, który nie jest ściśle powiązany z określonym skoroszytem, arkuszem lub zakresem. Na przykład poniższa instrukcja czyści zawartość aktywnej komórki, pomimo że nie jest znany jej adres:

```
ActiveCell.ClearContents
```

Kolejna instrukcja wyświetla komunikat zawierający nazwę aktywnego arkusza:

```
MsgBox ActiveSheet.Name
```

Aby poznać nazwę i ścieżkę aktywnego skoroszytu, należy użyć następującej instrukcji:

```
MsgBox ActiveWorkbook.FullName
```

Gdy zaznaczysz w arkuszu zakres komórek, za pomocą jednego polecenia możesz go w całości wypełnić wybraną wartością. W poniższym przykładzie właściwość `Selection` obiektu `Application` zwraca obiekt `Range` odpowiadający zaznaczonym komórkom. Polecenie po prostu modyfikuje właściwość `Value` obiektu `Range`, a wynikiem takiej operacji jest zakres komórek wypełnionych daną wartością:

```
Selection.Value = 12
```

Jeżeli zostanie zaznaczone coś innego niż zakres (np. obiekt `ChartObject` lub `Shape`), próba wykonania takiego polecenia spowoduje wygenerowanie błędu, ponieważ obiekty `ChartObject` i `Shape` nie posiadają właściwości `Value`.

Zwróć jednak uwagę na fakt, że kolejne polecenie wpisuje wartość `12` do komórek reprezentowanych przez obiekt `Range`, który został zaznaczony jeszcze przed zaznaczeniem innego obiektu, niebędącego obiektem typu `Range`. Jeżeli przyjrzysz się opisowi właściwości `RangeSelection` w pomocy systemowej, przekonasz się, że właściwość ta ma zastosowanie wyłącznie do obiektu `Window`.

```
ActiveWindow.RangeSelection.Value = 12
```

Aby dowiedzieć się, ile komórek zostało zaznaczonych w aktywnym oknie, należy użyć właściwości `Count`. Oto przykład:

```
MsgBox ActiveWindow.RangeSelection.Count
```

METODY

Oprócz wspomnianych wcześniej właściwości obiekty posiadają również swoje metody. Metoda to inaczej operacja wykonywana na obiekcie. Wracając do naszej analogii „domowej", w każdej chwili możesz pomalować swój dom na inny kolor, co w języku VBA przekładałoby się na coś w stylu metody `dom.pomaluj`.

Prostym przykładem jest użycie metody `Select` obiektu `Range`:

```
Range("A1").Select
```

Innym przykładem może być użycie metody `Copy` obiektu `Range`:

```
Range("A1").Copy
```

Niektóre metody posiadają argumenty pozwalające na doprecyzowanie sposobu działania metody. Na przykład, metoda `Paste` może działać w sposób bardziej efektywny po zdefiniowaniu argumentu `Destination`:

```
ActiveSheet.Paste Destination:=Range("B1")
```

Kilka słów o argumentach

Jednym z zagadnień, które często wywołują zamieszanie wśród niedoświadczonych programistów używających języka VBA, są argumenty metod i właściwości. Niektóre metody korzystają z argumentów w celu dokładniejszego zdefiniowania operacji. Z kolei niektóre właściwości korzystają z argumentów w celu uzyskania możliwości dodatkowego określenia wartości. W niektórych przypadkach argumenty są opcjonalne.

Weźmy dla przykładu metodę `Protect` obiektu `Workbook`. Jeżeli sprawdzisz w systemie pomocy opis tej metody, to zobaczysz, że metoda `Protect` pobiera trzy argumenty, reprezentujące odpowiednio hasło, status ochrony struktury skoroszytu i status ochrony okna. Argumenty odpowiadają opcjom zawartym w oknie dialogowym *Chronienie struktury i systemu Windows*.

Jeżeli na przykład zależy Ci na ochronie skoroszytu o nazwie *MójSkoroszyt.xlsx*, możesz użyć następującego polecenia:

```
Workbooks("MójSkoroszyt.xlsx").Protect "xyzzy", True, False
```

W tym przypadku skoroszyt jest chroniony hasłem (argument 1). Dodatkowo jest chroniona jego struktura (argument 2), ale okna skoroszytu już nie (argument 3).

Jeżeli nie chcesz przypisywać hasła, możesz użyć następującej instrukcji:

```
Workbooks("MójSkoroszyt.xlsx").Protect , True, False
```

Pierwszy argument został pominięty, a zamiast niego pozostawiliśmy puste miejsce identyfikowane przy użyciu przecinka.

Kolejna metoda zwiększająca czytelność kodu źródłowego polega na zastosowaniu argumentów, którym nadano nazwy. Oto przykład demonstrujący, w jaki sposób w poprzedniej instrukcji zastosować nazwane argumenty:

```
Workbooks("MójSkoroszyt.xlsx").Protect Structure:=True, Windows:=False
```

Zastosowanie nazwanych argumentów jest dobrym pomysłem, zwłaszcza w przypadku metod mających wiele argumentów opcjonalnych, a także w sytuacji, gdy konieczne jest użycie jedynie kilku z nich. Przy korzystaniu z nazwanych argumentów nie ma potrzeby pozostawiania pustych miejsc identyfikujących pominięte argumenty.

W przypadku właściwości i metod zwracających wartość konieczne jest umieszczenie argumentów w nawiasach okrągłych. Przykładowo właściwość `Address` obiektu `Range` pobiera pięć argumentów, z których wszystkie są opcjonalne. Ze względu na to, że właściwość `Address` zwraca wartość, poniższa instrukcja nie jest poprawna, ponieważ pominięto nawiasy okrągłe:

```
MsgBox Range("A1").Address False ' nieprawidłowa instrukcja
```

Aby składnia takiej instrukcji była poprawna, konieczne jest zastosowanie nawiasów okrągłych:

```
MsgBox Range("A1").Address(False)
```

Instrukcja może też zostać zapisana przy użyciu nazwanego argumentu w następujący sposób:

```
MsgBox Range("A1").Address(rowAbsolute:=False)
```

Tego typu niuanse stają się bardziej zrozumiałe w miarę zdobywania doświadczenia z zakresu języka VBA.

Tajemnice obiektów Range

Wiele zadań realizowanych przy użyciu instrukcji języka VBA jest powiązanych z komórkami i zakresami arkuszy — w końcu w tym właśnie celu stworzono arkusze kalkulacyjne. Z tego względu poświęcimy teraz trochę czasu na bardziej szczegółowe zapoznanie się z obiektem `Range`.

Wyszukiwanie właściwości obiektów Range

Uruchom edytor VBE i następnie wybierz z jego menu głównego polecenie *Help/Microsoft Visual Basic for Applications Help* (pomoc/pomoc dla języka Microsoft VBA). Zostaniesz przeniesiony na stronę internetową Microsoft Developer Network (MSDN), gdzie w wyszukiwarce możesz wpisać słowo kluczowe `Range` i zapoznać się z tematami pomocy dostępnymi dla obiektu `Range`. Dowiesz się tam, że obiekt `Range` posiada trzy właściwości, których możesz używać podczas pracy z arkuszami. Są to:

- właściwość `Range` obiektu klasy `Worksheet` lub `Range`,
- właściwość `Cells` obiektu `Worksheet`,
- właściwość `Offset` obiektu `Range`.

Właściwość Range

Właściwość `Range` zwraca obiekt `Range`. Jeżeli poszukasz w pomocy systemowej, to dowiesz się, że ta właściwość posiada dwie wersje składni:

```
obiekt.Range(komórka1)
obiekt.Range(komórka1, komórka2)
```

Właściwość Range jest powiązana z dwoma typami obiektów — Worksheet lub Range. Słowa komórka1 i komórka2 są odpowiednikami terminów, które Excel traktuje jako identyfikator zakresu (w pierwszym przypadku) lub jego granice (drugi przypadek). Poniżej zamieszczono kilka przykładów użycia właściwości Range.

Do tej pory poznałeś przykłady podobne do zamieszczonego poniżej. Polecenie takie po prostu wpisuje wybraną wartość do określonej komórki arkusza. W tym konkretnym przypadku w komórce A1 arkusza Arkusz1 jest umieszczana wartość 12.3.

```
Worksheets("Arkusz1").Range("A1").Value = 12.3
```

Właściwość Range rozpoznaje też nazwy zdefiniowane w skoroszytach. A zatem jeżeli komórce nadano nazwę Input, można wpisać do niej wartość przy użyciu następującego polecenia:

```
Worksheets("Arkusz1").Range("Input").Value = 100
```

W kolejnym przykładzie do zakresu złożonego z 20 komórek aktywnego arkusza jest wstawiana taka sama wartość (jeżeli aktywna karta Excela nie jest zwykłym arkuszem, zostanie wygenerowany komunikat błędu):

```
ActiveSheet.Range("A1:B10").Value = 2
```

Kolejne polecenie daje takie same wyniki, jak poprzednio:

```
Range("A1", "B10") = 2
```

Co prawda w powyższym poleceniu pominięto odwołanie do aktywnego arkusza, ale jest ono używane domyślnie. Poza tym pominięto właściwość Value obiektu Range, ponieważ jest to właściwość domyślna. W powyższym przykładzie użyto drugiej składni właściwości Range. W jej przypadku pierwszym argumentem jest górna lewa komórka, natomiast drugim — dolna prawa komórka zakresu.

W następnym przykładzie zastosowano operator przecięcia zakresów Excela (znak spacji), który zwraca przecięcie dwóch zakresów. W tym przypadku w miejscu przecięcia zakresów znajduje się jedna komórka, C6, stąd następująca instrukcja wpisuje do tej komórki wartość 3:

```
Range("C1:C10 A6:E6") = 3
```

Na koniec kolejne polecenie wpisuje wartość 4 do pięciu komórek tworzących nieciągły zakres. Przecinek spełnia funkcję operatora złączenia (zwróć uwagę, że ciąg odwołań do poszczególnych komórek, oddzielonych od siebie przecinkami, został ujęty w znaki cudzysłowu):

```
Range("A1, A3, A5, A7, A9") = 4
```

Dotychczas we wszystkich przykładach używaliśmy właściwości Range obiektu Worksheet. Jak już wspominaliśmy wcześniej, możesz też używać właściwości Range obiektu Range.

Poniżej zamieszczono przykład takiego użycia właściwości Range obiektu Range. W tym przypadku obiektem Range jest aktywna komórka. Poniższe polecenie traktuje obiekt Range tak, jakby komórka znajdowała się w górnym lewym narożniku arkusza, i wstawia wartość 5 do komórki, którą *będzie* B2. Innymi słowy, zwrócone odwołanie jest względne w stosunku do górnego lewego narożnika obiektu Range. A zatem instrukcja wpisze wartość 5 bezpośrednio do komórki znajdującej się na prawo od aktywnej komórki i jeden wiersz poniżej:

```
ActiveCell.Range("B2") = 5
```

Jest to trochę zagmatwane, ale na szczęście istnieje o wiele bardziej zrozumiała metoda uzyskania dostępu do komórki względnej w stosunku do zakresu. Jest to właściwość Offset, którą omówimy w jednym z kolejnych podrozdziałów.

Właściwość Cells

Kolejna metoda odwoływania się do zakresu polega na użyciu właściwości Cells. Podobnie jak właściwość Range, tak i właściwość Cells jest używana w stosunku do obiektów Worksheet i Range. Kiedy zajrzysz do pomocy systemowej programu Excel, przekonasz się, że właściwość Cells ma trzy warianty składni:

```
obiekt.Cells(rowIndex, columnIndex)
obiekt.Cells(rowIndex)
obiekt.Cells
```

Poniżej podano kilka przykładów zastosowania właściwości Cells. W pierwszym z nich do komórki A1 arkusza Arkusz1 zostanie wpisana wartość 9. W tym przypadku posłużę się pierwszą składnią akceptującą numer indeksu wiersza (wartości z przedziału od 1 do 1048576) i numer indeksu kolumny (wartości z przedziału od 1 do 16384):

```
Worksheets("Arkusz1").Cells(1, 1) = 9
```

Kolejny przykład to instrukcja wpisująca do komórki D3 (czyli komórki znajdującej się na przecięciu wiersza numer 3 i kolumny numer 4 aktywnego arkusza) wartość 7:

```
ActiveSheet.Cells(3, 4) = 7
```

Można też użyć właściwości Cells obiektu Range. W tym przypadku obiekt Range zwracany przez właściwość Cells jest względny w stosunku do górnej lewej komórki obiektu Range, do którego jest wykonywane odwołanie. Zagmatwane? Raczej tak. Kolejny przykład może sprawić, że stanie się to bardziej przejrzyste. Instrukcja wpisuje do aktywnej komórki wartość 5. Pamiętaj, że aktywna komórka jest traktowana tak, jakby była komórką A1 arkusza.

```
ActiveCell.Cells(1, 1) = 5
```

UWAGA

Prawdziwa korzyść wynikająca z zastosowania tego typu odwołania do komórek stanie się widoczna przy omawianiu zmiennych i pętli (patrz rozdział 3.). W większości przypadków zamiast faktycznych wartości argumentów będziesz raczej używał zmiennych.

Aby do komórki znajdującej się bezpośrednio poniżej aktywnej komórki wpisać wartość `5`, należy użyć następującej instrukcji:

```
ActiveCell.Cells(2, 1) = 5
```

Instrukcję tę można opisać za pomocą następującego zdania: „Rozpocznij od aktywnej komórki i traktuj ją tak, jakby to była komórka `A1`. Następnie wpisz wartość `5` w komórce znajdującej się w drugim wierszu i pierwszej kolumnie".

W drugim wariancie składni właściwości `Cells` jest używany jeden argument, który może przyjmować wartości z przedziału od 1 do 17 179 869 184. Wartość ta jest równa liczbie komórek arkusza programu Excel 2013. Komórki są numerowane, począwszy od `A1`, kolejno w prawą stronę, a następnie od początku następnego wiersza. Na przykład komórka numer 16 384 ma adres `XFD1`, natomiast komórka o numerze 16 385 ma już adres `A2`.

W następnym przykładzie do komórki `SZ1` aktywnego arkusza (która jest 520. komórką tego arkusza) wprowadzono wartość `2`.

```
ActiveSheet.Cells(520) = 2
```

Aby wyświetlić wartość ostatniej komórki arkusza (o adresie `XFD1048576`), należy użyć następującej instrukcji:

```
MsgBox ActiveSheet.Cells(17179869184)
```

Taka składnia może też zostać użyta w przypadku obiektu `Range`. W tym przypadku zwrócona komórka jest względna w stosunku do obiektu `Range`, do którego się odwołano. Jeżeli na przykład obiekt `Range` reprezentuje zakres `A1:D10` (40 komórek), właściwość `Cells` może pobrać argument o wartości z przedziału od 1 do 40 i zwrócić jedną z komórek obiektu `Range`. W poniższym przykładzie wartość `2000` jest wpisywana do komórki `A2`, ponieważ jest to piąta komórka zakresu (licząc od góry i w prawą stronę, a następnie od początku następnego wiersza), do którego się odwołano:

```
Range("A1:D10").Cells(5) = 2000
```

UWAGA

W poprzednim przykładzie argument właściwości `Cells` nie jest ograniczony do wartości z przedziału od 1 do 40. Jeżeli wartość argumentu przekracza liczbę komórek zakresu, zliczanie jest kontynuowane, tak jakby zakres był większy niż w rzeczywistości. A zatem polecenie podobne do powyższego może zmienić wartość komórki spoza zakresu `A1:D10`. Na przykład polecenie przedstawione poniżej zmienia wartość komórki `A11`:

```
Range("A1:D10").Cells(41) = 2000
```

Trzeci wariant składni właściwości `Cells` po prostu zwraca wszystkie komórki arkusza, do którego jest wykonywane odwołanie. W przeciwieństwie do dwóch pozostałych składni w tym przypadku nie jest zwracana pojedyncza komórka. Poniższa instrukcja używa metody `ClearContents` w stosunku do zakresu zwróconego przez właściwość `Cells` aktywnego arkusza. W wyniku wykonania instrukcji czyszczona jest zawartość wszystkich komórek arkusza.

```
ActiveSheet.Cells.ClearContents
```

Pobieranie informacji z komórki

Jeżeli chcesz odczytać zawartość wybranej komórki, to VBA udostępnia Ci kilka właściwości. Poniżej zamieszczamy krótkie zestawienie najczęściej używanych właściwości:

- Właściwość `Formula` zwraca formułę zapisaną w pojedynczej komórce (pod warunkiem oczywiście, że taka formuła istnieje). Jeżeli dana komórka nie posiada formuły, zwracana jest wartość komórki. Właściwość `Formula` jest właściwością przeznaczoną do odczytu i zapisu i posiada kilka podobnych odmian, takich jak `FormulaR1C1`, `FormulaLocal` czy `FormulaArray` (więcej szczegółowych informacji na ich temat znajdziesz w systemie pomocy).
- Właściwość `Value` zwraca nieformatowaną wartość komórki. Jest to właściwość przeznaczona do odczytu i zapisu.
- Właściwość `Text` zwraca tekst, który jest wyświetlany w komórce. Jeżeli w danej komórce znajduje się wartość numeryczna, właściwość zwraca wartość razem z formatowaniem, takim jak przecinki czy symbole walut. Właściwość `Text` jest przeznaczona tylko do odczytu.
- Właściwość `Value2` jest nieco podobna do właściwości `Value`, z wyjątkiem tego, że nie korzysta z danych typu `Date` oraz `Currency`. W przypadku napotkania takich typów danych, właściwość dokonuje ich konwersji na zmienne typu `Variant` zawierające wartości typu `Double`. Na przykład, jeżeli w komórce została zapisana data `2016-03-16`, to właściwość `Value` zwróci ją jako wartość typu `Date`, podczas gdy właściwość `Value2` zwróci wartość typu `Double` (w tym wypadku `42445`).

Właściwość Offset

Właściwość `Offset` (podobnie jak właściwości `Range` i `Cells`) również zwraca obiekt typu `Range`. Jednak w przeciwieństwie do dwóch poprzednio omawianych metod, właściwość `Offset` jest powiązana tylko i wyłącznie z obiektem `Range`. Oto jej składnia:

```
obiekt.Offset(rowOffset, columnOffset)
```

Właściwość `Offset` pobiera dwa argumenty odpowiadające względnemu położeniu odniesionemu do górnej lewej komórki określonego obiektu `Range`. Argumentami mogą być wartości dodatnie (przesunięcie w dół lub w prawo), ujemne (przesunięcie w górę lub w lewo) lub zero. Poniższa instrukcja wpisuje wartość `12` do komórki znajdującej się bezpośrednio poniżej komórki aktywnej:

```
ActiveCell.Offset(1, 0).Value = 12
```

Kolejna instrukcja wstawia wartość `15` do komórki znajdującej się bezpośrednio powyżej aktywnej komórki:

```
ActiveCell.Offset(-1, 0).Value = 15
```

Nawiasem mówiąc, jeżeli aktywna komórka znajduje się w wierszu 1, właściwość `Offset` z powyższej instrukcji spowoduje wygenerowanie błędu, ponieważ nie będzie mogła zwrócić obiektu `Range`, który nie istnieje.

Właściwość `Offset` jest dość przydatna, zwłaszcza w przypadku stosowania zmiennych w procedurach wykonujących pętlę. Więcej szczegółowych informacji na ten temat znajdziesz w kolejnym rozdziale.

Po zarejestrowaniu makra w trybie odwołań względnych, Excel użyje właściwości `Offset` w celu odwołania się do komórek względnych w stosunku do pozycji początkowej (komórki, która jest aktywna w momencie rozpoczęcia rejestrowania makra). Na przykład do wygenerowania poniższego kodu użyłem rejestratora makr. Najpierw umieściłem kursor w komórce `B1`, a następnie wprowadziłem wartości do komórek zakresu `B1:B3` i powróciłem do komórki `B1`.

```
Sub Makro1()
   ActiveCell.FormulaR1C1 = "1"
   ActiveCell.Offset(1, 0).Range("A1").Select
   ActiveCell.FormulaR1C1 = "2"
   ActiveCell.Offset(1, 0).Range("A1").Select
   ActiveCell.FormulaR1C1 = "3"
   ActiveCell.Offset(-2, 0).Range("A1").Select
End Sub
```

Zwróć uwagę, że rejestrator makr użył właściwości `FormulaR1C1`. Zazwyczaj w celu wprowadzenia wartości do komórki będziemy używać właściwości `Value`, co nie zmienia jednak faktu, że po zastosowaniu właściwości `FormulaR1C1` lub nawet `Formula` uzyskasz takie same wyniki.

Warto również zauważyć, że wygenerowany kod odwołuje się do komórki `A1`, co może wydawać się trochę dziwne, ponieważ komórka ta nie była nawet użyta w trakcie rejestrowania makra. Jest to nietypowy element procedury rejestrowania, który sprawia, że kod źródłowy jest niepotrzebnie aż tak złożony. Możesz śmiało usunąć wszystkie odwołania do `Range("A1")`, a makro nadal będzie znakomicie działać:

```
Sub ZmodyfikowaneMakro1()
   ActiveCell.FormulaR1C1 = "1"
   ActiveCell.Offset(1, 0).Select
   ActiveCell.FormulaR1C1 = "2"
   ActiveCell.Offset(1, 0).Select
   ActiveCell.FormulaR1C1 = "3"
   ActiveCell.Offset(-2, 0).Select
End Sub
```

Poniżej zamieszczamy jeszcze bardziej efektywną wersję makra, która nie używa metody `Select`:

```
Sub Makro1()
   ActiveCell = 1
   ActiveCell.Offset(1, 0) = 2
   ActiveCell.Offset(2, 0) = 3
End Sub
```

Podstawowe zagadnienia, które należy zapamiętać

W tym podrozdziale przedstawimy kilka dodatkowych zagadnień, których opanowanie jest niezbędne dla każdego, kto chce na poważnie zajmować się programowaniem w języku VBA. Poszczególne zagadnienia staną się bardziej zrozumiałe w miarę zdobywania doświadczenia i czytania kolejnych rozdziałów.

- **Obiekty posiadają unikatowe właściwości i metody** — Każdy obiekt dysponuje własnym zestawem właściwości i metod. Jednak niektóre obiekty mają wspólne niektóre właściwości (np. `Name`) i metody (np. `Delete`).
- **Możesz manipulować obiektami bez ich zaznaczania** — Może to być niezgodne ze standardowym wyobrażeniem przetwarzania obiektów w Excelu. W praktyce jednak wykonywanie operacji na obiektach bez ich uprzedniego zaznaczenia jest zazwyczaj efektywniejsze. Kiedy rejestrujesz nowe makro, Excel najpierw zaznacza obiekt, a dopiero potem wykonuje na nim operacje. Nie jest to konieczne i właściwie spowalnia działanie makra.

- **Powinieneś dokładnie zrozumieć koncepcję i zasady pracy z kolekcjami** — W większości przypadków odwołujesz się do obiektu w sposób pośredni za pośrednictwem kolekcji, w której dany obiekt się znajduje. Aby odwołać się na przykład do obiektu klasy `Workbook` o nazwie `Mój_plik`, powinieneś użyć następującego odwołania do kolekcji `Workbooks`:

  ```
  Workbooks("Mój_plik.xlsx")
  ```

  Powyższe odwołanie zwraca obiekt, który jest żądanym przez Ciebie skoroszytem.

- **Właściwości mogą zwracać odwołanie do innego obiektu** — Przykładowo w poniższej instrukcji właściwość `Font` zwraca obiekt `Font` zawarty w obiekcie `Range`. `Bold` to właściwość obiektu `Font`, a nie obiektu `Range`.

  ```
  Range("A1").Font.Bold = True
  ```

- **Istnieje wiele różnych metod odwoływania się do tego samego obiektu** — Załóżmy, że dysponujesz skoroszytem o nazwie `Sprzedaż` i jest to jedyny otwarty w danej chwili skoroszyt. Przyjmijmy też, że skoroszyt zawiera jeden arkusz o nazwie `Zestawienie`. Do arkusza można odwołać się przy użyciu następujących sposobów:

  ```
  Workbooks("Sprzedaż.xlsx").Worksheets("Zestawienie")
  Workbooks(1).Worksheets(1)
  Workbooks(1).Sheets(1)
  Application.ActiveWorkbook.ActiveSheet
  ActiveWorkbook.ActiveSheet
  ActiveSheet
  ```

To, która metoda zostanie wybrana, przeważnie zależy od wiedzy użytkownika na temat obszaru roboczego. Jeżeli na przykład otwartych jest więcej skoroszytów niż jeden, nieodpowiednia będzie druga lub trzecia metoda. Jeżeli używasz aktywnego arkusza (dowolnego typu), każda z trzech ostatnich metod będzie skuteczna. Aby mieć całkowitą pewność, że odwołujesz się do właściwego arkusza określonego skoroszytu, zastosuj pierwszą metodę.

Kilka słów o przykładach kodu

W książce znajdziesz wiele fragmentów kodu źródłowego języka VBA, które mają na celu zilustrowanie danego zagadnienia czy problemu. Bardzo często takie przykłady kodu mogą składać się tylko z jednego polecenia. W niektórych przypadkach cytowany przykład składa się tylko z pojedynczego *wyrażenia*, które samo w sobie nie jest jeszcze poprawnym poleceniem języka VBA.

Na przykład kod przedstawiony poniżej jest wyrażeniem:

```
Range("A1").Value
```

Aby przetestować wyrażenie, musisz je obliczyć (wykonać). W tym przypadku dobrym rozwiązaniem będzie użycie funkcji `MsgBox`:

```
MsgBox Range("A1").Value
```

Jeżeli chcesz samodzielnie wypróbować podane przykłady, powinieneś umieścić polecenie w procedurze utworzonej w module kodu VBA, na przykład:

```
Sub Test()
'    tutaj wstaw testowane polecenie
End Sub
```

Następnie umieść kursor w dowolnym miejscu kodu procedury i naciśnij klawisz *F5* aby ją wykonać. Dodatkowo powinieneś się upewnić, czy kod procedury jest wykonywany we właściwym kontekście. Na przykład: jeżeli testowane polecenie odnosi się do arkusza `Arkusz1`, powinieneś upewnić się, czy aktywny skoroszyt faktycznie zawiera arkusz o takiej nazwie.

Jeżeli kod źródłowy składa się tylko z jednej instrukcji, możesz użyć okna *Immediate* edytora Visual Basic. Okno to jest bardzo przydatne w przypadku natychmiastowego wykonywania instrukcji bez konieczności tworzenia procedury. Jeżeli okno *Immediate* nie jest widoczne, w edytorze Visual Basic powinieneś nacisnąć kombinację klawiszy *Ctrl+G*.

Po wprowadzeniu instrukcji w oknie *Immediate* wystarczy nacisnąć klawisz *Enter*. Aby obliczyć (wykonać) dane wyrażenie, powinieneś na jego początku wstawić znak zapytania (`?`), który pełni funkcję odpowiednika metody `Print`. Na przykład w oknie *Immediate* możesz wprowadzić następujące wyrażenie:

```
? Range("A1").Value
```

Wynik wyrażenia zostanie wyświetlony w następnym wierszu okna *Immediate*.

Nie panikuj — nie jesteś sam

Jeżeli po raz pierwszy masz styczność z językiem VBA, prawdopodobnie będziesz trochę przytłoczony mnogością obiektów, właściwości i metod. To zupełnie normalne. Nikt nie staje się ekspertem programowania w języku VBA w ciągu jednego dnia. Nabieranie doświadczenia w programowaniu VBA to kwestia czasu i praktyki. Dobra wiadomość jest taka, że nie jesteś pierwszą osobą, która przez to przechodzi. Do dyspozycji masz ogromną ilość różnych źródeł, w których z pewnością znajdziesz odpowiedzi na nurtujące Cię pytania. W tym podrozdziale przedstawimy kilka dobrych sposobów zdobycia informacji, które pomogą Ci skierować się we właściwą stronę.

Przeczytaj resztę książki

Nie zapomnij o tym, że tytuł tego rozdziału brzmi „Wprowadzenie do języka VBA". W pozostałej części książki znajdziesz wiele dodatkowych szczegółów oraz przydatnych i pouczających przykładów.

Pozwól Excelowi napisać makro za Ciebie

Bez wątpienia najlepszą metodą zaznajomienia się z językiem VBA jest po prostu uaktywnienie rejestratora makr i zapisanie kilku operacji wykonywanych w Excelu. Po zakończeniu rejestrowania makra możesz przeanalizować wygenerowany kod i spróbować jeszcze lepiej dostosować go do swoich potrzeb.

Na przykład załóżmy, że potrzebujesz makra, które odświeża wszystkie tabele przestawne w skoroszycie i resetuje ustawienia wszystkich filtrów tabel przestawnych. Napisanie takiego makra od zera byłoby dosyć trudnym zadaniem. Zamiast tego możesz po prostu uruchomić rejestrator makr i za jego pomocą zarejestrować, jak ręcznie odświeżasz wszystkie tabele przestawne i resetujesz ich filtry. Po zatrzymaniu rejestratora makr możesz przeanalizować otrzymany kod VBA i ewentualnie dokonać niezbędnych modyfikacji.

Korzystaj z systemu pomocy

Początkującemu użytkownikowi system pomocy Excela może się wydawać nieprzyjaznym mechanizmem, który w odpowiedzi na pytania użytkownika zwraca oszałamiającą liczbę tematów wydających się nie mieć nic wspólnego z oryginalnym zapytaniem. Prawda jest jednak taka, że kiedy nauczysz się korzystać z tego systemu pomocy, to stanie się on najlepszym i najszybszym źródłem szczegółowych informacji o niuansach programowania w VBA.

Pamiętaj o dwóch podstawowych założeniach systemu pomocy Excela: podczas zadawania pytań lokalizacja kursora w kodzie ma ogromne znaczenie oraz aby skorzystać z systemu pomocy, musisz mieć aktywne połączenie z internetem.

Podczas zadawania pytań lokalizacja ma znaczenie

Excel tak naprawdę posiada dwa systemy pomocy: pierwszy, który dostarcza informacje na temat funkcji i sposobu działania samego Excela, i drugi, który oferuje pomoc na temat programowania w języku VBA. Zamiast wykonywać globalne wyszukiwanie tematów pasujących do zadanego pytania, Excel stara się ograniczyć kryteria wyszukiwania tylko do takich, które mają sens w kontekście aktualnego położenia kursora. W praktyce oznacza to, że pomoc wyświetlana przez Excela jest kontekstowa i zależy od tego, co w danej chwili robisz. Wynika stąd po prostu, że jeżeli poszukujesz pomocy na temat jakiegoś zagadnienia związanego z makrami i programowaniem w języku VBA, musisz pracować w edytorze VBE, dzięki czemu wyszukiwanie tematów pasujących do zapytania będzie się odbywało we właściwym systemie pomocy.

Musisz mieć połączenie z internetem

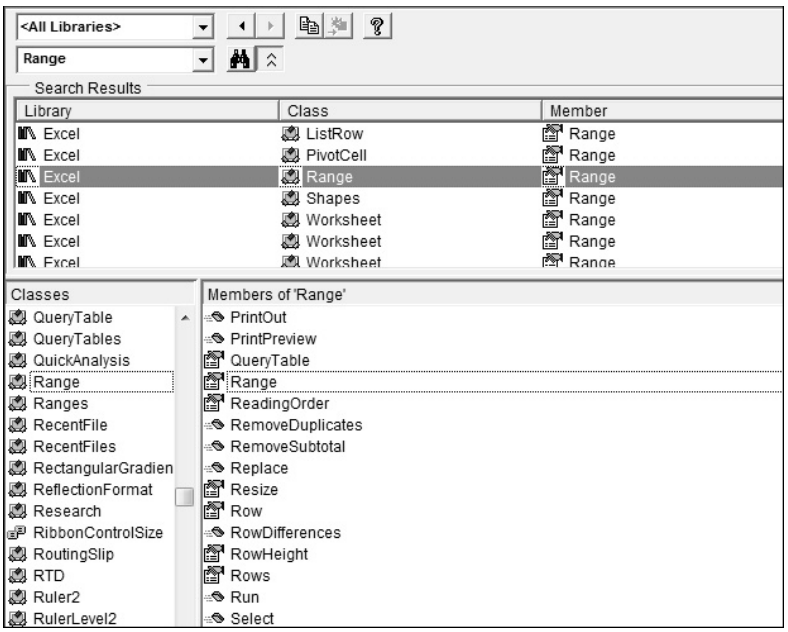
Kiedy poszukujesz pomocy na określony temat, Excel sprawdza, czy jesteś podłączony do internetu. Jeżeli tak, Excel zabiera Cię na stronę sieciową MSDN (Microsoft Developer Network), gdzie możesz przeszukiwać system pomocy online. Jeżeli nie jesteś połączony z internetem, Excel wyświetli na ekranie okno dialogowe z informacją, że aby uzyskać pomoc, musisz się połączyć.

Używaj przeglądarki obiektów

Object Browser, czyli przeglądarka obiektów, jest bardzo przydatnym narzędziem wyświetlającym wszystkie właściwości i metody dostępne dla danego obiektu. Po uaktywnieniu edytora Visual Basic przeglądarkę *Object Browser* można uruchomić na jeden z trzech sposobów:

- Naciśnij klawisz *F2*.
- Wybierz z menu *View* (widok) polecenie *Object Browser* (przeglądarka obiektów).
- Kliknij przycisk *Object Browser* znajdujący się na pasku narzędzi *Standard*.

Okno przeglądarki *Object Browser* zostało przedstawione na rysunku 2.17.



Rysunek 2.17. Przeglądarka Object Browser jest znakomitym źródłem informacji o obiektach

Lista rozwijana w górnym lewym narożniku przeglądarki *Object Browser* zawiera wszystkie dostępne biblioteki obiektów:

- Program Excel.
- *MSForms* (używana do tworzenia niestandardowych okien dialogowych).
- *Office* (obiekty wspólne dla wszystkich aplikacji pakietu Microsoft Office).
- *Stdole* (obiekty automatyzacji OLE).
- VBA.
- Bieżący projekt (wybrany w oknie *Project*) i wszystkie skoroszyty, do których się odwołuje.

Zawartość okna *Classes* zależy od elementu wybranego z listy rozwijanej. Z kolei zawartość okna *Members of* zależy od elementu, który zostanie zaznaczony w oknie *Classes*.

Jeśli po wybraniu biblioteki chcesz uzyskać listę właściwości i metod zawierających określony tekst, możesz go wprowadzić w polu szukania *Search Text* drugiej listy rozwijanej, a następnie kliknąć ikonę *Search* (lornetka).

1. Zaznacz interesującą Cię bibliotekę.

   Jeżeli nie jesteś pewien, która biblioteka obiektów jest właściwa, wybierz pozycję *All Libraries*.

2. W polu listy rozwijanej znajdującej się poniżej listy bibliotek wpisz nazwę poszukiwanego obiektu.
3. Aby rozpocząć wyszukiwanie, kliknij ikonę lornetki.

Wyniki wyszukiwania zostaną wyświetlone w oknie *Search Results*. Aby w oknie *Classes* wyświetlić klasy obiektu, zaznacz go na liście. Aby uzyskać listę składników klasy (właściwości, metody i stałe), kliknij nazwę klasy. Zwróć uwagę na dolny panel wyświetlający dodatkowe informacje o obiekcie. Aby bezpośrednio przejść do właściwego tematu pomocy, naciśnij klawisz *F1*.

Na pierwszy rzut oka korzystanie z przeglądarki *Object Browser* może się wydawać trudne, ale z czasem się do niej przyzwyczaisz i docenisz jej przydatność.

Szukaj kodu w internecie

Składnia wszystkich poleceń VBA, których kiedykolwiek będziesz potrzebował, z pewnością już dawno została opublikowana gdzieś w sieci. Pod wieloma względami współczesne programowanie nie polega już tylko na tworzeniu potrzebnego kodu od zera, ale coraz częściej sprowadza się do wyszukiwania potrzebnego kodu w sieci i dostosowywania go do potrzeb określonego zadania.

Jeżeli utknąłeś podczas pisania makra wykonującego takie czy inne zadanie, po prostu uruchom swoją ulubioną przeglądarkę sieciową i w polu wyszukiwania krótko opisz zadanie, które sprawia Ci problem. Aby osiągnąć lepsze rezultaty, możesz przed opisem problemu umieścić frazę *Excel VBA*.

Jeżeli próbujesz napisać na przykład makro, które będzie usuwało wszystkie puste wiersze w arkuszu, możesz wpisać w wyszukiwarce następujące zapytanie: **`excel vba usuwanie pustych wierszy`**. Możesz śmiało założyć się o swoją dwumiesięczną pensję, że ktoś w internecie borykał się już kiedyś z takim samym problemem. Co więcej, w dziewięciu przypadkach na dziesięć znajdziesz również gotowy kod rozwiązujący takie zadanie; może on być dla Ciebie znakomitym punktem wyjścia do zastosowania w swoim programie.

Wykorzystuj fora dyskusyjne użytkowników Excela

Jeżeli znajdziesz się w naprawdę poważnych tarapatach, możesz zamieścić swoje pytanie na wybranym forum dyskusyjnym użytkowników Excela, i to nie bez nadziei na szybką, profesjonalną odpowiedź.

Fora dyskusyjne to internetowe grupy użytkowników zajmujących się określonymi zagadnieniami. W takich miejscach możesz zamieścić swoje pytania i oczekiwać, że znajdzie się ekspert lub chociaż bardziej doświadczony użytkownik, który przedstawi propozycję rozwiązania opisanego problemu. Użytkownicy odpowiadający na takich forach to zazwyczaj pasjonaci, którzy na ochotnika podejmują się pomagania innym.

Istnieje wiele forów internetowych zajmujących się tematyką Excela i programowania w VBA. Aby znaleźć takie miejsce, po prostu wpisz w wyszukiwarce sieciowej hasło **`Excel forum`**.

Poniżej zamieszczamy kilka najważniejszych zasad korzystania z większości forów internetowych:

- Przed rozpoczęciem korzystania z danego forum zawsze przeczytaj i przestrzegaj jego regulaminu. W takich regulaminach często zamieszczane są odpowiedzi na najczęściej zadawane pytania i reguły etykiety danego forum.
- Zamieszczając posty staraj się nadawać im krótkie tematy związane z problemem. Nigdy nie umieszczaj na forach postów zatytułowanych *Potrzebna pomoc* czy *Mam pytanie*.
- Staraj się zadawać precyzyjnie sformułowane pytania, dotyczące konkretnego problemu. Nie zadawaj pytań w stylu *Jak mam napisać makro podliczające faktury?*
- Bądź cierpliwy. Pamiętaj, że użytkownicy udzielający odpowiedzi na forach to pasjonaci, którzy oprócz działania na formach mają swoje normalne życie, pracę, rodzinę i dzieci. Daj im trochę czasu na odpowiedź.
- Często zaglądaj na forum. Po zadaniu pytania możesz otrzymać wiele dodatkowych pytań z prośbą o doprecyzowanie problemu bądź dodatkowe informacje. Staraj się ułatwiać innym życie i szybko odpowiadaj na zadawane pytania.
- Kiedy otrzymasz odpowiedź, podziękuj osobie, która Ci ją przesłała oraz wszystkim innym osobom, które były zaangażowane w rozwiązywanie Twojego problemu.

Odwiedzaj blogi ekspertów

Jest wielu ekspertów w dziedzinie Excela, którzy dzielą się swoją wiedzą z innymi za pośrednictwem blogów internetowych. Takie miejsca to prawdziwe kopalnie wiedzy, wskazówek i trików, które może eksplorować każdy, kto chce poszerzyć swoją wiedzę. A co najlepsze, wszystko za darmo!

Choć na takich bloga nie zawsze będzie rozwiązanie Twojego konkretnego problemu, to jednak z całą pewnością znajdziesz tam wiele interesujących artykułów i porad, które pozwolą Ci nabierać nowych doświadczeń i używać Excela w nowych zastosowaniach.

Poniżej zamieszczamy listę kilku najlepszych blogów internetowych, zajmujących się zagadnieniami Excela:

http://chandoo.org

http://www.contextures.com

http://www.datapigtechnologies.com/blog

http://www.dailydoseofexcel.com

http://www.excelguru.ca/blog

http://www.mrexcel.com

Poszukaj szkolenia wideo na YouTube

Niektórzy z nas uczą się lepiej i szybciej, kiedy oglądają film ilustrujący sposób rozwiązania danego problemu. Jeżeli należysz do takich osób, powinieneś poszukać odpowiedniego szkolenia wideo na popularnym serwisie YouTube. Znajdziesz tam dziesiątki kanałów prowadzonych przez pasjonatów chcących podzielić się swoją wiedzą i doświadczeniem. Z pewnością będziesz zaskoczony ilością i jakością zamieszczanych tam materiałów.

Wejdź na stronę *http://youtube.com* i w pasku wyszukiwania wpisz **`Excel VBA`**.

Ucz się z Microsoft Office Dev Center

Microsoft Office Dev Center to serwis internetowy pomagającym nowym deweloperom rozpocząć karierę w programowaniu produktów pakietu Office. Część tego serwisu, poświęconą Excelowi znajdziesz pod adresem *https://msdn.microsoft.com/en-us/library/office/fp179694.aspx*.

Choć poruszanie się po stronach Microsoft Office Dev Center może na pierwszy rzut oka nie być takie proste, to jednak warto poświęcić chwilę na zapoznanie się z nim. Znajdziesz tam bardzo wiele różnych zasobów, włącznie z gotowymi fragmentami kodu, narzędziami, instrukcjami krok po kroku i innymi.

Analizuj inne aplikacje Excela, które są używane w Twojej organizacji

Inne aplikacje Excela używane w Twojej organizacji mogą być dla Ciebie prawdziwą kopalnią wiedzy, pomysłów i rozwiązań. Jeżeli znajdziesz taki plik, postaraj się zajrzeć mu „pod maskę" i przeanalizować zawarte w nim makra. Przekonaj się, jak inni użytkownicy piszą swoje makra. Spróbuj przeanalizować ich kod wiersz po wierszu i postaraj się wyłapać różne interesujące sztuczki i techniki programowania. Być może natkniesz się nawet na całe fragmenty kodu, które będziesz mógł wykorzystać w swoich zastosowaniach.

Zapytaj lokalnego guru

Czy masz w swojej organizacji jakiegoś lokalnego guru zajmującego się Excelem? Postaraj się z nim poznać i zaprzyjaźnić. Większość ekspertów uwielbia dzielić się swoją wiedzą z innymi. Nie bój się podejść lub zadzwonić do takiej osoby i poprosić o pomoc w rozwiązaniu trapiącego Cię problemu.

Skorowidz

A

B

C

D

E

F

G

H

I

J

K

L

Ł

M

N

O

P

Q

R

S

Ś

T

U

V

W

X

Y

Z

Połącz VBA z Excelem i napisz najlepszą aplikację!

Wielu z nas nie wyobraża sobie pracy bez Excela. Studenci, naukowcy, biznesmeni, analitycy, księgowi, a nawet gospodynie domowe znajdują dla tego potężnego arkusza kalkulacyjnego najróżniejsze zastosowania. Mimo że możliwości Excela są imponujące, można je jeszcze rozszerzać i budować aplikacje szczególnego przeznaczenia za pomocą dostarczonego przez Microsoft narzędzia — języka Visual Basic for Applications (VBA). Wystarczy tylko poznać składnię i zasady programowania w VBA!

Niniejsza książka jest zaktualizowanym wydaniem unikalnego przewodnika po języku VBA. W przystępny sposób wyjaśniono tu wiele złożonych zagadnień, dzięki czemu szybko zaczniesz pisać programy, które automatyzują wykonywanie różnych zadań w Excelu. Będą to zarówno proste makra, jak i wyrafinowane aplikacje i narzędzia, pozwalające na zaawansowaną interakcję z użytkownikiem.

Układ treści w książce pozwala zarówno na systematyczną naukę VBA, jak i na szybkie wyszukanie porad i wskazówek umożliwiających rozwiązanie konkretnego zadania. Nie zabrakło również wielu praktycznych przykładów gotowego do użycia kodu.

Michael Alexander to Microsoft Certified Application Developer (MCAD) i laureat prestiżowego tytułu Microsoft MVP, który otrzymał za bezustanne wspieranie użytkowników Excela. Jest również autorem kilku książek o zaawansowanych technikach analitycznych dla biznesu w MS Excel i Access.

Dick Kusleika od ponad dwudziestu lat pracuje z pakietem Office, a od dwunastu lat zdobywa tytuł Microsoft MVP. Ma ogromne doświadczenia w opracowywaniu rozwiązań bazujących na MS Excel i Access. Jest cenionym szkoleniowcem w dziedzinie obsługi pakietu Office w Australii i w USA.

Dzięki tej książce poznasz:

- **podstawy programowania w VBA, w tym tworzenie funkcji i procedur**
- **projektowanie aplikacji arkusza kalkulacyjnego**
- **automatyzację operacji na tabelach przestawnych i wykresach**
- **integrację aplikacji Excela z innymi aplikacjami, takimi jak Word czy Outlook**
- **pracę z zewnętrznymi źródłami danych**

Helion
księgarnia internetowa
http://helion.pl
zamówienia telefoniczne
0 801 339900
0 601 339900
Informatyka w najlepszym wydaniu

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

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

WILEY

ISBN 978-83-283-2858-7

cena: 99,00 zł