

## 29

Wykorzystanie  
Visual Basic

## ZAGADNIENIA

- Użycie struktur VBA
- Różnice pomiędzy funkcjami i podprogramami
- Manipulowanie danymi przy użyciu VBA
- Tworzenie interakcji pomiędzy użytkownikami a bazą danych
- Obsługa wyjątków – tworzenie własnej obsługi błędów
- Wstępne informacje na temat VBA: edytor VBA, pojęcie zmiennej i stałej, funkcje i podprogramy, pojęcie procedury publicznej i prywatnej, konstrukcja tablic, zestawy rekordów, kontrola obiektów, okna dialogowe i informacyjne, odpluskwanie, Error Messages
- Sposoby ochrony VBA
- Używanie makr
- Procedura dodawania przycisków komend pochodzących od kreatorów
- Użycie opcji DoCmd

Osoby zajmujące się bazami danych oraz pisaniem interfejsów dla baz danych to zazwyczaj informatycy mający wykształcenie i spore doświadczenie w dziedzinie programowania. Omawianie podstaw VBA można zatem teoretycznie uznać za niekonieczne. Założenie takie w niektórych wypadkach może okazać się błędne, ponieważ są osoby, które pomimo wykształcenia informatycznego mają niewiele wspólnego nawet z elementarnymi zasadami pisania programów komputerowych. W rozdziale tym zostaną jedynie przybliżone metody programowania w VBA. Ponieważ nie jest to podręcznik do VBA, czytelnik powinien uznać prezentowane informacje jedynie za wprowadzenie do pisania makr i posługiwania się VBA.

VBA to skrót od Visual Basic for Applications. Jest to język komputerowy, który pozwala na pisanie programów będących rozwinięciem aplikacji. W naszym wypadku aplikacją będzie Microsoft Access, natomiast programy pisane w VBA będą służyły dostosowaniu Accessa oraz zaimplementowaniu w nim nowych funkcji. Częściową funkcjonalność, którą uzyskuje się, tworząc kod VBA, możemy osiągnąć, posługując się kreatorem i edytorem makra. Zasadnicza różnica pomiędzy używaniem makr i VBA polega na tym, że makro daje możliwości wykorzystania jedynie funkcji oferowanych w rozwijanych menu, podczas gdy VBA pozwala na osiągnięcie znaczenie większej funkcjonalności i elastyczności, ponieważ kod funkcji piszemy sami i dzięki temu możemy tworzyć procedury i funkcje, których program nie udostępnia.

Praca w środowisku VBA przebiega z użyciem dwóch rodzajów procedur.

- Do pierwszego rodzaju należą: **procedura publiczna** i **procedura prywatna** (sub). Ich oddzielny zasięg działania określany jest za pomocą słowa **public** lub **private**.



- Drugim rodzajem procedur są **funkcje (function)**, które podobnie do procedur możemy podzielić na publiczne i prywatne. Tworzenie miniprogramów w obrębie Bazy Access ma celu wzbogacenie tego programu o zautomatyzowanie i ułatwienie pracy oraz większą elastyczność w dostosowaniu Accessa do indywidualnych potrzeb.

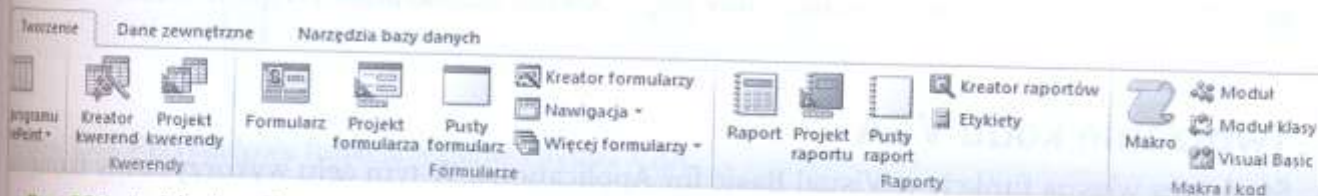
Stosowane w VBA procedury oraz podprocedury nie zwracają wartości tak jak funkcje. Procedura **sub** to seria instrukcji Visual Basic, która występuje w klamrze zaczynającej się od **sub** i zakończonej **end sub**. Jej zadaniem jest wykonanie operacji w momencie, gdy zostanie wywołany identyfikator znajdujący się bezpośrednio po **sub**. Ponieważ jest to procedura, ogranicza się jedynie do wykonania instrukcji, które są w niej zawarte.

Procedura **sub** ma zasięg określany przez słowo kluczowe stojące bezpośrednio przed **sub**: **private** albo **public**<sup>\*</sup>.

Procedura **function**, podobnie do **sub**, może być poprzedzona słowem kluczowym **private** lub **public**. Funkcje w VBA mogą być zbudowane z użyciem parametrów przekazywanych do funkcji. Podprogram to zestaw instrukcji uruchamianych z poziomu aplikacji (w naszym wypadku programu Access). Podobnie do **sub**, po słowie kluczowym **function** występuje identyfikator funkcji, a cała funkcja zakończona jest przez **end function**. Na tym kończą się podobieństwa. Funkcja, w odróżnieniu od procedury **sub**, może pobierać wartości, które występują jako parametry, i po wykonaniu operacji na danych pobranych jako parametry zwraca wyniki. Podczas programowania VBA stosuje się konwencję zapisu identyfikatorów, np. Function LiczbaCałkowita. Ponieważ VBA nie jest wrażliwy na wielkość liter, to praktyka taka poprawia czytelność kodu.

W obiektach typu moduł mamy dwa rodzaje modułów:

- moduł (moduł standardowy),
- moduł klasy.

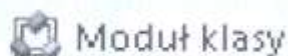
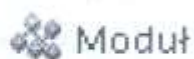


Rys. 29.1. Zakładka Tworzenie programu Access

Warto zapamiętać, że kod umieszczony w **module standardowym** dostępny jest z każdego poziomu bazy danych Access i nie jest związany z żadnym określonym obiektem. W przeciwieństwie do modułu standardowego, moduł klasy zawiera kod związany z określonym obiektem, np. formularzem, raportem, tabelą. Funkcjonalność **modułów klasy** opiera się na wykonaniu kodu w odpowiedzi na określone zdarzenie. Moduł klasy określonego formularza może zostać uruchomiony np. w chwili użycia zaprojektowanego w tym formularzu przycisku lub w momencie umieszczenia w formularzu danych, które mają zostać zmodyfikowane od razu po wprowadzeniu. Moduły klasy nie są zatem dostępne z poziomu całej bazy Access, tak jak standardowe moduły, tylko z poziomu wybranego obiektu, którego dotyczą. Można pokusić się o twierdzenie, że rozpatrywane pod tym kątem moduły przypominają swoim działaniem wyzwalacze (*trigger*). Na etapie tworzenia kodu VBA należy zwrócić uwagę na różnicę pomiędzy ikoną dającą dostęp do modułów standardowych a ikoną

<sup>\*</sup> Posługiwanie się słowami kluczowymi **public** i **private** ma szczególne zastosowanie w przypadku modułów klas, ponieważ określa dostęp do funkcji lub procedur w obrębie określonej klasy lub poza nią. W przypadku modułów standardowych (globalnych) dostęp do funkcji i procedur uzyskiwany jest z całej bazy danych, więc funkcje i procedury w nich zawarte są globalne przez samo umieszczenie ich w module standardowym.

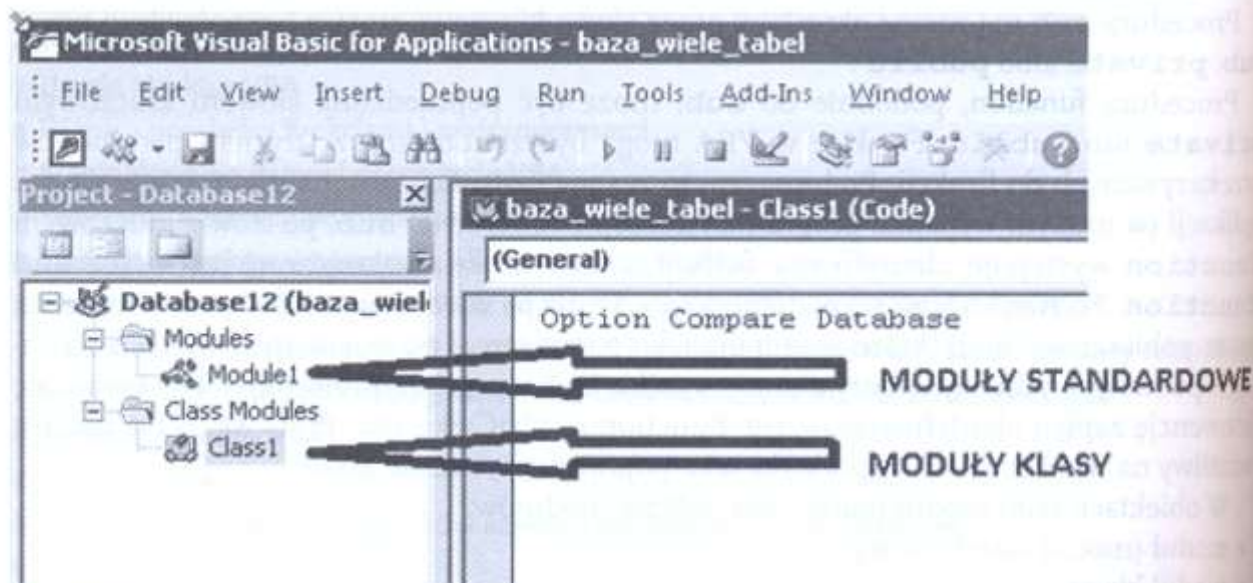




Rys. 29.2. Ikona modułu klasy i modułu standardowego (globalnego)

dającą dostęp do modułów klasy. Dzięki temu unikniemy pomyłki i kod umieścimy zawsze we właściwym miejscu.

Gdy użyjemy przycisku **Moduł klasy** w lewej części interfejsu edytora modułów, zostaną wyświetlone dotychczasowe moduły klas i moduły standardowe. To menu umożliwia uzyskanie łatwego dostępu do każdego z modułów przez wybranie go za pomocą kursora myszy i umieszczenie odpowiednich fragmentów kodu we właściwych miejscach.

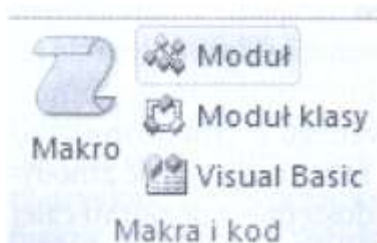


Rys. 29.3. Moduły klas i moduły standardowe pogrupowane osobno przez program Access

## Tworzenie kodu VBA

Stwórzmy własną funkcję w Visual Basic for Applications. W tym celu wykorzystamy ikonę **Moduł** w zakładce **Tworzenie**.

Po uruchomieniu tego przycisku powinno ukazać się okno Microsoft Visual Basic for Applications (rys. 29.5).



Rys. 29.4. Tworzenie modułu standardowego

Dość istotne jest, aby widoczna była przestrzeń **Immediate** umożliwiająca natychmiastowe sprawdzanie działania funkcji.

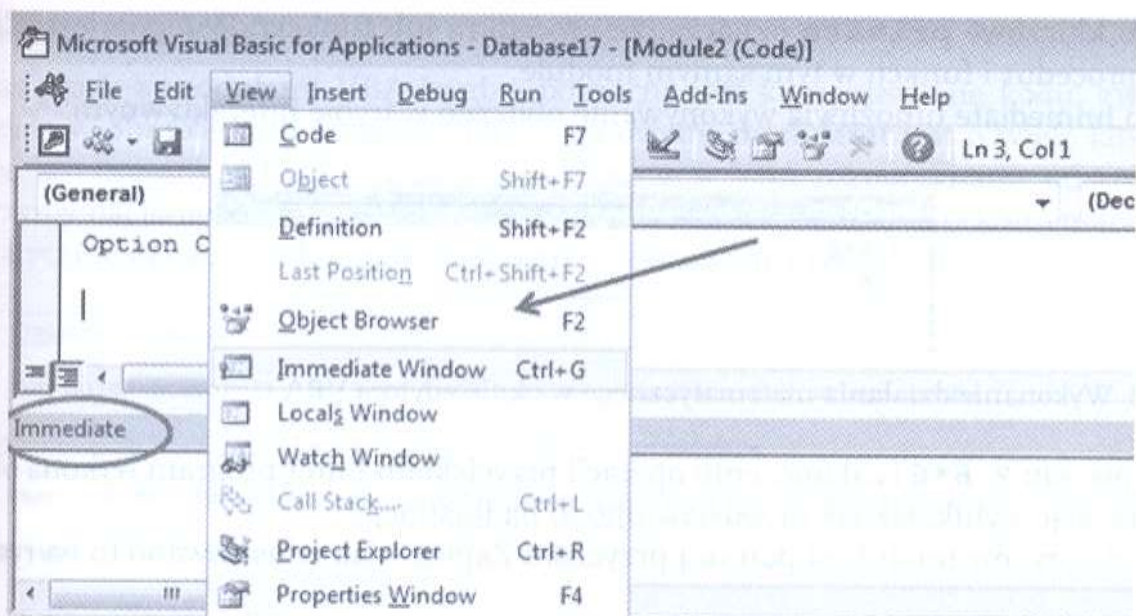
Pierwsza z funkcji będzie miała na celu dodanie dwóch wartości przekazanych jako parametry. Funkcja zwraca wartość przez swoją nazwę – identyfikator z rysunku 29.6.

Jeśli funkcja została poprawnie napisana, przechodzimy do pola **Immediate** i natychmiast przetestujemy nasz kod źródłowy. Aby to zrobić, musimy wywołać nazwę funkcji, podając przy tym parametry, które chcemy dodać. Nazwę funkcji, która ma zostać wykonana, poprzedzamy znakiem zapytania (?), a całość operacji kończymy, naciskając Enter.

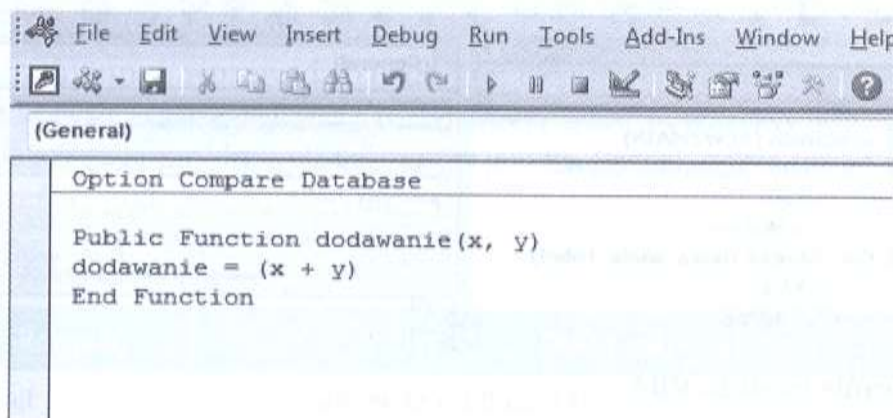
Warto zaznaczyć, że podczas tworzenia identyfikatorów w VBA nie ma konieczności zwracania uwagi na wielkość liter. Powyższą funkcję z powodzeniem można wywołać w następujący sposób: **?DoDawaNie (5, 8)**.

Podczas stawiania pierwszych kroków w VBA w programie Access każdy nowo tworzony moduł ma automatycznie wpisaną linię **Option Compare Database**.

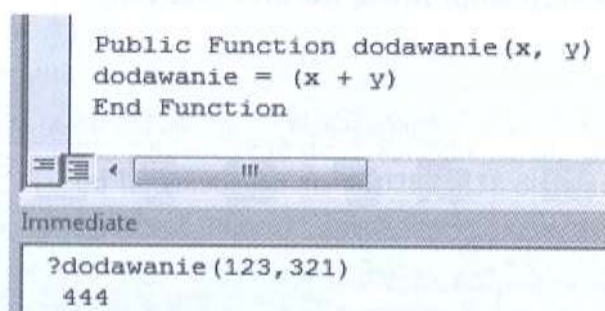




Rys. 29.5. Okno edytora VBA



Rys. 29.6. Przykładowa funkcja w oknie edytora VBA



Rys. 29.7. Przykład działania funkcji VBA

Instrukcja `Option Compare` określa, jaka metoda porównywania łańcuchów zostanie użyta w obrębie modułu. Dlatego musi być zastosowana na samym początku przed wszystkimi procedurami i funkcjami. Dostępne są trzy opcje: Binary, Text, Database.

Opcja Database może być użyta tylko w programie Access. Porównanie łańcuchów uzależnione jest od porządku narzucanego przez lokalne ID bazy.

Opcja Binary – porównywanie łańcuchów – opiera się na binarnej reprezentacji znaków. Natomiast opcja Text oparta jest na lokalnym ustaleniu systemu.

Słowo kluczowe **public** oznacza, że procedura jest dostępna dla całej bazy danych.

Słowo kluczowe **private** oznacza, że procedura lub funkcja dostępna jest tylko dla innych procedur i funkcji w tym samym module.

Okno **Immediate** umożliwia wykonywanie obliczeń w trybie interaktywnym:



Rys. 29.8. Wykonanie działania matematycznego w oknie edytora VBA

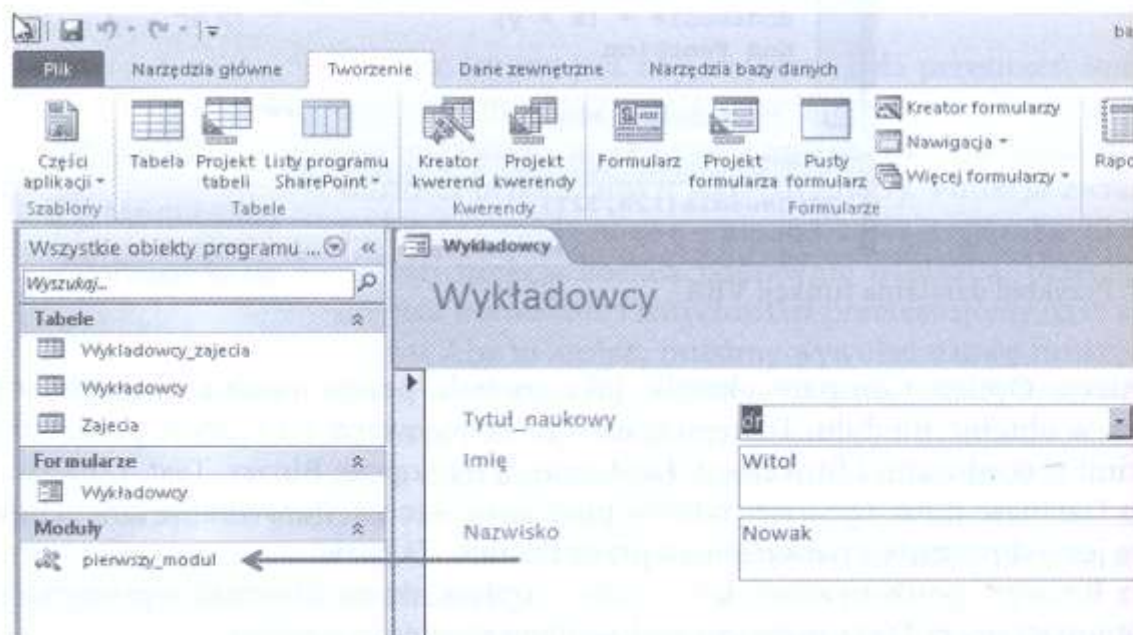
Po wpisaniu `? 6*6` i zakończeniu operacji przyciskiem Enter program wykona obliczenie, zwracając wynik, tak jak przedstawiono to na ilustracji.

Gdy utworzymy moduł, za pomocą przycisku Zapisz – jak zilustrowano to na rysunku:



Rys. 29.9. Zapisywanie modułu VBA

możemy wskazać nazwę dla tworzonego modułu (np. **pierwszy\_modul**) i w każdej chwili uzyskać do niego dostęp za pomocą menu obiektów.



Rys. 29.10. Uzyskanie dostępu do modułu za pomocą okna obiektów programów Access



## Komentarze VBA

Podczas pracy z modułami VBA bardzo pomocne jest komentowanie kodu, zwłaszcza gdy działa w obrębie bazy danych. Praca z gotowymi modułami jest o wiele łatwiejsza, gdy kod jest opisany w komentarzach. Komentarze w VBA rozpoczynamy od apostrofu, a kończymy naciśnięciem przycisku Enter. Prawidłowo skonstruowany komentarz będzie automatycznie oznaczony kolorem zielonym i ignorowany przez kompilator.

```
Option Compare Database

Public Sub MojaProcedura() 'to jest komentarz zaczynający się od apostrofu
End Sub                    'w tej linii kończy się procedura

Public Function MojaFunkcja() 'tu zadeklarowana jest funkcja o identyfikatorze: MojaFunkcja
End Function
```

Rys. 29.11. Komentarze VBA

## Typy danych w VBA

VBA daje możliwości deklarowania zmiennych w locie. Oznacza to, że w momencie, gdy użyjemy jakiejś zmiennej, wprowadzając do niej określoną wartość, zostanie automatycznie zadeklarowana, np.  $x = a + b$  będzie deklaracją trzech zmiennych  $a$ ,  $b$ ,  $x$ . Nie ma konieczności deklarowania zmiennych, jak np. w języku Pascal, gdzie przed użyciem zmiennej należało określić, jakiego będzie typu.

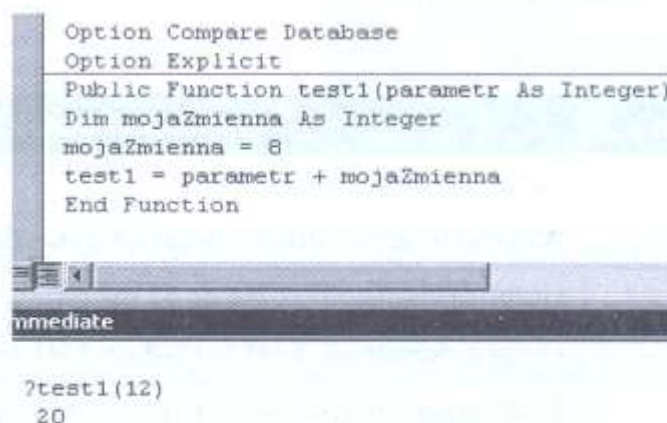
Tabela 29.1. Typy danych

Typ danych	Rozmiar	Opis i zakres wartości
Byte	1 bajt	Liczby całkowite od 0 do 255
Boolean	2 bajty	Wartości logiczne: True (prawda) lub False (fałsz)
Integer	2 bajty	Liczby całkowite od -32 768 do 32 767
Long	4 bajty	Liczby całkowite od -2 147 483 648 do 2 147 483 647
Single	4 bajty	Liczby zmiennoprzecinkowe pojedynczej precyzji: od -3,402823E38 do -1,401298E-45 dla wartości ujemnych od 1,401298E-45 do 3,402823E38 dla wartości dodatnich
Double	8 bajtów	Liczby zmiennoprzecinkowe podwójnej precyzji: od -1,79769313486231E308 do -4,94065645841247E-324 dla wartości ujemnych od 4,94065645841247E-324 do 1,79769313486232E308 dla wartości dodatnich
Currency	8 bajtów	Duża precyzyjna liczba; może zawierać 19 cyfr, w tym cztery na prawo od przecinka. Wartości z przedziału: od -922 337 203 685 477,5808 do 922 337 203 685 477,5807
Decimal	14 bajtów	Bardzo duża, bardzo precyzyjna liczba; może zawierać 29 cyfr oraz do 28 miejsc na prawo od przecinka. Wartości z przedziału: +/-79 228 162 514 264 337 593 543 950 335 bez przecinka dziesiętnego +/-7,9228162514264337593543950335 z 28 miejscami po przecinku Najmniejsza wartość niezerowa to: +/-0,00000000000000000000000000000001



Typ danych	Rozmiar	Opis i zakres wartości
Date	8 bajtów	Daty i godziny: od 1 stycznia 100 do 31 grudnia 9999
Object	4 bajty	Dowolne odesłanie do wartości typu Object
String	10 bajtów + liczba znaków	Tekst o zmiennej długości od 0 do 2 miliardów znaków
String	Liczba znaków	Tekst o stałej długości od 1 do 65 400 znaków
Variant	16 bajtów	Dowolna wartość liczbowa w zakresie określonym dla typu Double
Variant	22 bajty + dłu- gość ciągu	Taki sam zakres jak dla zmiennej długości typu String
Definiowa- ny przez użytkownika za pomocą struktury Type	Liczba wyma- gana przez elementy	Zakres każdego elementu jest taki sam jak zakres typu danych tego elementu

Jeśli chcemy, aby zmienna została zadeklarowana zgodnie z ustalonym dla niej typem, wówczas używamy słowa kluczowego **dim** (*dimension* – wymiar).



**Rys. 29.12.** Deklaracja danych w VBA

Powyższa ilustracja prezentuje zadeklarowanie typu integer dla zmiennej **mojaZmienna** wewnątrz funkcji **test1**.

VBA daje też możliwości definiowania własnych typów przed zadeklarowaniem zmiennej. Deklarowania zmiennych dokonujemy, podając identyfikator zmiennej (pierwszy znak identyfikatora nie może być cyfrą) oraz słowo kluczowe **As**, po którym występuje nazwa wybranego przez nas typu, który będzie odpowiedni dla wartości przechowywanych w zmiennej.

Gdy piszemy programy w środowisku VBA, istnieje możliwość, że nad przygotowaną przez nas bazą danych pracować będą inni programiści, których zadaniem będzie modyfikacja kodu VBA. Dlatego warto, tworząc identyfikatory zmiennych procedur i funkcji, stosować pewną konwencję. Wprowadzenie reguł w wyborze nazw – identyfikatorów dla zmiennych, uzasadnia sytuacja, w której jesteśmy zmuszeni edytować i zmieniać własny kod po kilku latach. Zasada, którą proponujemy, nie jest nowością i na pewno ułatwi



pisanie, zwłaszcza bardzo skomplikowanego kodu – nie tylko w VBA. Jeśli tworzymy identyfikatory dla zmiennych, którym nadajemy typ, warto użyć trzyznakowego skrótu nazwy typu zmiennej, po którym występuje dowolnie wybrana przez nas nazwa, np. **zmienna1**. Przykładem stosowania takiej zasady dla zmiennej (**zmienna1**) typu: Byte, String, Date, są następujące identyfikatory:

**bytZmienna1, strZmienna1, datZmienna1.**

Łatwo zauważyć, że gdy zachodzi potrzeba użycia zmiennej, to sama nazwa podpowiada nam, jakiego typu jest zmienna.

Oprócz zmiennych możemy również deklarować stałe z użyciem słowa kluczowego **const**.

Używamy w tym celu następującej składni: **const identyfikator\_stalej As typ**.

Przykład deklaracja stałej **liczba\_zawodnikow**:

```
const liczba_zawodnikow As Integer = 12
```

Deklaracja stałej w przeciwieństwie do zmiennej powoduje, że za pomocą identyfikatora odwołujemy się do stałej, której wartość nie może zostać zmodyfikowana (tak jak w przypadku zmiennej) i próba przypisania innej wartości skończy się niepowodzeniem.

## Tablice w VBA

VBA pozwala również na użycie tablic, które deklarujemy podobnie jak zmienne za pomocą słowa kluczowego **dim**. Różnica w deklaracji tablicy polega na umieszczeniu po identyfikatorze tablicy nawiasu, w którym za pomocą liczby całkowitej określamy, z ilu elementów będzie się składać tablica. Jeśli tworzymy tablicę do przechowywania wartości typu String, które będą dniami tygodnia, zadeklarujemy tablicę za pomocą konstrukcji:

```
Dim moja_tablica(7) As String.
```

Przykład użycia tablicy prezentuje poniższa ilustracja.

General)

```
Option Compare Database
Public Function MojaFunkcja(parametr2 As Integer)
Dim tablica(7) As String
tablica(1) = "Poniedzialek"
tablica(2) = "Wtorek"
tablica(3) = "Środa"
tablica(4) = "Czwartek"
tablica(5) = "Piątek"
tablica(6) = "Sobota"
tablica(7) = "Niedziela"
MojaFunkcja = tablica(parametr2)
End Function
```

Immediate

```
?MojaFunkcja(2)
Wtorek
```

Rys. 29.13. Przykład zastosowania tablic w VBA

Tablice wieloelementowe tworzymy, oddzielając przecinkiem podczas deklaracji definiowane wartości elementów dla każdego z wymiarów tablicy:

```
Dim tablica_dwuwymiarowa(2,2) As Integer
```

Istotnym zagadnieniem podczas pisania kodu jest czas życia zmiennej, który zależy od miejsca jej zadeklarowania.



Jeśli zadeklarujemy zmienne wewnątrz procedury lub funkcji, wówczas dostęp do zmiennej będzie można uzyskać tylko z procedury lub funkcji, wewnątrz której została zadeklarowana (np. zmienne *x* i *y* na ilustracji). Jeśli zmienna zostanie zadeklarowana globalnie – zmienne **wiek** i **imie**<sup>\*</sup>, wówczas dostęp do niej można uzyskać z każdego miejsca wpisywanego kodu.

(General)	
Option Compare Database	
Option Explicit	
Dim wiek As Integer	
Public imie As String	
<hr/>	
Public Sub procedura(parametr As Integer)	
Dim x As Integer	
'treść procedury	
'instrukcje	
End Sub	
<hr/>	
Public Function nowa_funkcja(parametr As Integer)	
y As Integer	
'treść funkcji	
'operacje na danych	
End Function	

Rys. 29.14. Zmienne lokalne i globalne – sposoby definiowania

Na tym etapie przystąpimy do tworzenia procedur i funkcji.

Zdarzają się sytuacje, gdy podczas tworzenia kodu pisana przez nas pojedyncza linia jest dość długa. Na tyle długa, że nie mieści się na ekranie. Oczywiście interfejs VBA umożliwia pisanie nawet bardzo długich linii, jednak aby je przeczytać, musimy posługiwać się przyciskiem przewijania u dołu okna. Aby pojedyncza linia była kontynuowana od nowego wiersza tak, aby w całości była widoczna na ekranie, a kompilator został poinformowany, że kolejna linia kodu to kontynuacja poprzedniej (a nie nowa linia), stosuje się znak podkreślenia  (underscore). Zakończenie linii w VBA następuje z chwilą przejścia do kolejnej linii przez użycie klawisza ENTER. Wówczas kompilator uznaje linię za zakończoną.

Prześledźmy to na przykładzie:

```
Option Compare Database
Public Function pubDodawanie(intZmienna1 As Integer, intZmienna2 As Integer)
pubDodawanie = intZmienna1 + intZmienna2
End Function
```

Rys. 29.15. Przykład kodu VBA

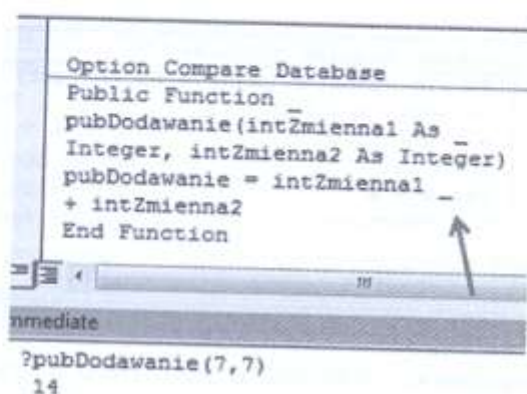
Linia rozpoczynająca się od **Public Function** jest dość długa. Spróbujmy rozdzielić ją na dwa wiersze kodu, stosując znak podkreślnika.

Jak ilustruje przykład na rys. 29.16, technika kontynuacji linii przez stosowanie podkreślnika działa bez zarzutu.

Pozostawienie na końcu linii podkreślnika informuje kompilator, że linia nie jest zakończona i ma ją kontynuować od kolejnego wiersza kodu.

\* Gdy deklarujemy zmienną wraz z określeniem jej zasięgu – słowo kluczowe **public**, wówczas nie używamy słowa kluczowego **dim**. Zmienne zadeklarowane globalnie i poprzedzone słowem kluczowym **private** dostępne są tylko i wyłącznie wewnątrz modułu, w którym zostały zadeklarowane.





Rys. 29.16. Zastosowanie podkreślnika w celu przejścia do następnego wiersza edytora z zachowaniem ciągłości kodu

## Słowo kluczowe ME

**Me** jest odwołaniem do obiektu w klasie, w której w danym momencie wykonuje się kod. Gdy zastosujemy słowo **me** z kropką, np.: **me.**, pojawiają się podpowiedzi dostępnych właściwości, metod (w ramach tego obiektu) oraz nazwy obiektów podrzędnych. Stosowanie słowa **me** jest zalecane, gdyż przyspiesza pracę i redukuje liczbę błędów.

## Pętle w VBA

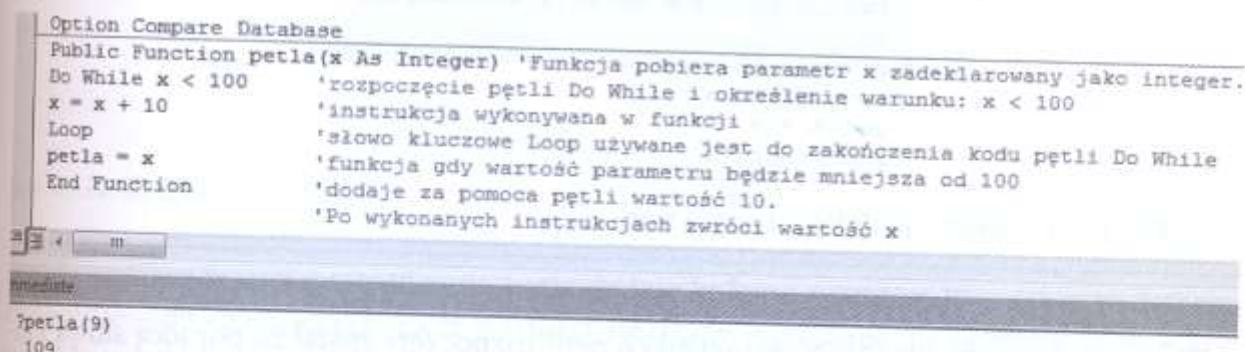
VBA pozwala na stosowanie pętli **Do While**.

Struktura **Do While** stosowana w VBA różni się od tych znanych z języków: Pascal i C++.

Pierwszy przypadek, który rozpatrzmy, to konstrukcja:

Do While [warunek]	Do Until [warunek]
Instrukcja pierwsza	Instrukcja pierwsza
Instrukcja druga	Instrukcja druga
....	....
Loop	Loop

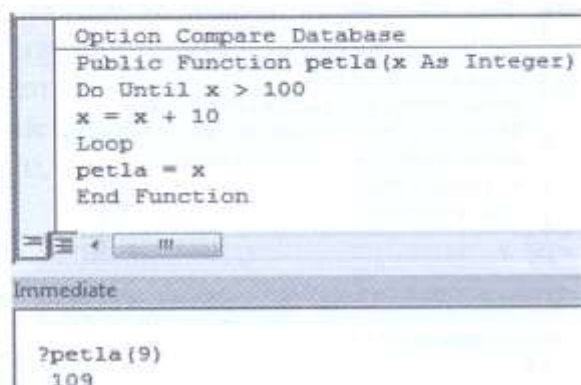
Na przykładzie kodu VBA powyższy schemat przedstawia się następująco:



Rys. 29.17. Przykład pętli Do While w VBA

Innym wariantem tej pętli jest zastosowanie **Until**.





Rys. 29.18. Przykład pętli Do Until

## SPRAWDŹ SWOJE UMIEJĘTNOŚCI

1. Dlaczego zmieniliśmy **warunek**  $x < 100$  na  $x > 100$ , aby przy parametrze wejściowym 9 funkcja zwróciła wartość 109, podobnie jak w powyższym przykładzie?
2. Jaka jest różnica między znaczeniem słów **While** i **Until**?

Warunek możemy również umieścić bezpośrednio po **Loop**, gdy kod bloku instrukcji rozpoczniemy za pomocą **Do**.

<b>Do</b> Instrukcja pierwsza Instrukcja druga .... <b>Loop Until</b> [warunek]	<b>Do</b> Instrukcja pierwsza Instrukcja druga .... <b>Loop While</b> [warunek]
---	---

### PRZYKŁAD 29.1

```

Option Compare Database
Public Function petla(x As Integer)
Do
x = x + 10
Loop While x < 100
petla = x
End Function

```

Rys. 29.19. Przykład pętli Do... Loop While...

W powyższym przykładzie blok instrukcji pętli rozpoczęty został za pomocą słowa kluczowego **Do**, po którym mogą występować instrukcje wykonywane za każdym przejściem pętli. Analogicznie do powyższych przykładów kod przedstawiony na ilustracji poniżej zawiera zmieniony warunek  $x > 100$ .



```
Option Compare Database
Public Function petla(x As Integer)
Do
x = x + 10
Loop Until x > 100
petla = x
End Function
```

Rys. 29.20. Przykład pętli Do Loop Until

Dość często pętle zawierają instrukcje o charakterze zabezpieczenia, które dbają, aby pętla nie przechodziła w nieskończoność (zapętlenie programu). W takich przypadkach należy umieścić kod, który zakończy działanie pętli (wyjdzie z pętli).

Rozważmy następujący przykład:


```
Option Compare Database
Public Function petla(x As Integer)
Do
x = x + 10
If (x = 59) Then Exit Do
Loop Until x > 100
petla = x
End Function
```

Rys. 29.21. Przykład kodu VBA z zastosowaniem warunku logicznego If

Jeśli parametrem wejściowym funkcji będzie 2, funkcja zwróci wartość 102, natomiast gdy parametrem wejściowym będzie 9, wówczas instrukcja IF (x=59) – jeżeli x jest równe 59 – otrzyma wartość **true**, tzn. zostanie spełniona i wykona kod **Exit Do**, co spowoduje wyjście z pętli i zwrócenie wyniku 59.

Wyobraźmy sobie sytuację, gdy kod wygląda następująco:

```
Option Compare Database
Public Function petla(x As Integer)
Do
x = x + 10
Loop Until x = 100
petla = x
End Function
```



The screenshot shows the VBA Immediate window with the text `?petla(0)` entered and the result `100` displayed below it.

Rys. 29.22. Kod funkcji VBA

Gdy wprowadzamy wartość zero, pętla dodaje kolejno wartości 10 – czyni to dziesięciokrotnie – do czasu osiągnięcia wartości 100. Gdy ją osiągnie, wykonany zostaje kod **Until x = 100** oraz **petla = x**.

Co się stanie, gdy np. wprowadzimy wartość jeden? Na kolejnych przejściach pętli zmienna x będzie osiągać wartości 11,21,31,...,91,101,111.... Ponieważ nigdy nie osiągnie wartości równej 100, dojdzie do zapętlenia.