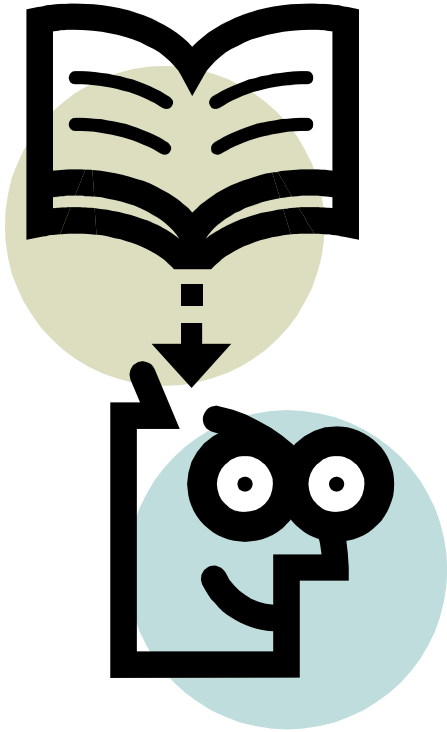
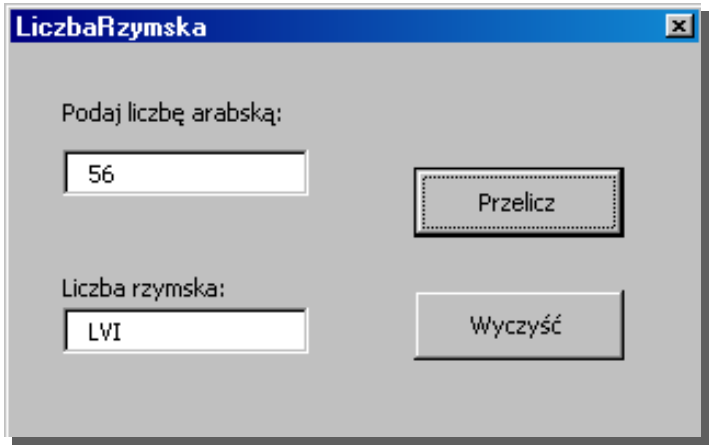
Programowanie w Visual Basic



Visual Basic - dziś

- Programy tworzone w języku **Visual Basic** są programami zdarzeniowymi, co oznacza, że zdarzenia (np. działania użytkownika, kliknięcie myszką, przesunięcie wskaźnika, minięcie określonego czasu i inne) decydują, kiedy i która procedura zostanie wywołana.

Działanie programu zależy od tego, co robi użytkownik i kiedy.



Po co Visual Basic for Applications?

- MS Excel rozbudowany o VBE staje się wszechstronną aplikacją, wykraczającą znacznie poza to, co większość ludzi kojarzy się ze światem arkuszy kalkulacyjnych.

- Spora część własnych aplikacji obliczeniowych może być wykonana w VBA, dzięki czemu Excel staje się produktem programowalnym.

Excel dla programistów

- ***Łatwy dostęp do formantów UserForm***: Excel pozwala dodawać do arkusza różne formanty, takie jak:
  - *Przyciski poleceń,*
  - *Pola list,*
  - *Przyciski opcji.*
- Implementacja tych formantów nie wymaga z reguły programowania żadnego makra.

- ***Własne okna dialogowe*** (*UserForm*).
- ***Dostosowywane menu***: Excel pozwala zmieniać zawartość poszczególnych menu, jak i dodawać całkiem nowe.
- ***Dostosowywane menu podręczne***: Excel daje możliwość definiowania własnego menu podręcznego, uruchamianego przez kliknięcie prawym przyciskiem myszy.

Excel dla programistów

- ***Dostosowywanie pasków narzędzi***, jako nowych elementów interfejsu użytkownika.

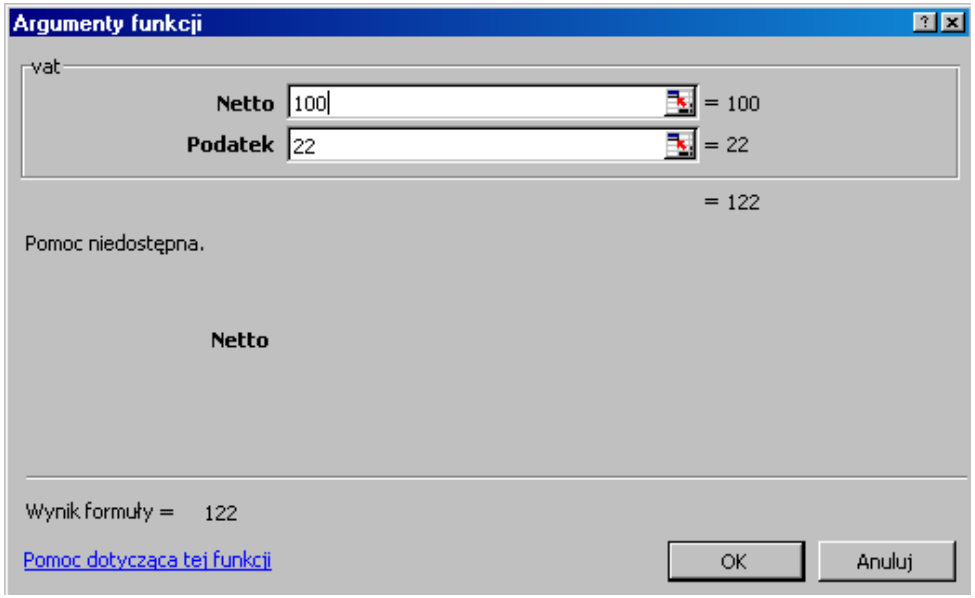
- ***Kwerendy***: ze środowiska skoroszytu można dostawać się bezpośrednio do innych ważnych źródeł danych. Mogą to być pliki w formatach standardowych baz danych, pliki tekstowe, strony www.

- ***Data Access Object*** i ***ActiveX Data Objects***: elementy te umożliwiają łatwą współpracę z zewnętrznymi bazami danych z poziomu VBA.

Excel dla programistów

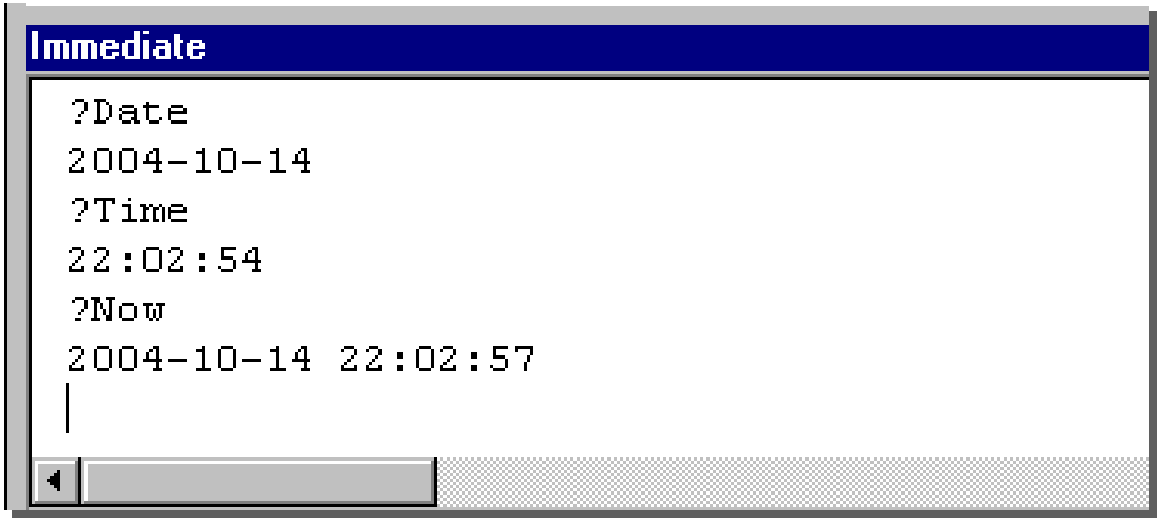
- ***Rozbudowana możliwość zabezpieczania***: utworzona aplikacja może zostać utajniona i zabezpieczona przed zmianami.

- ***Własne funkcje arkusza***: upraszczanie stosowania formuł i obliczeń



Instrukcje bezpośrednie

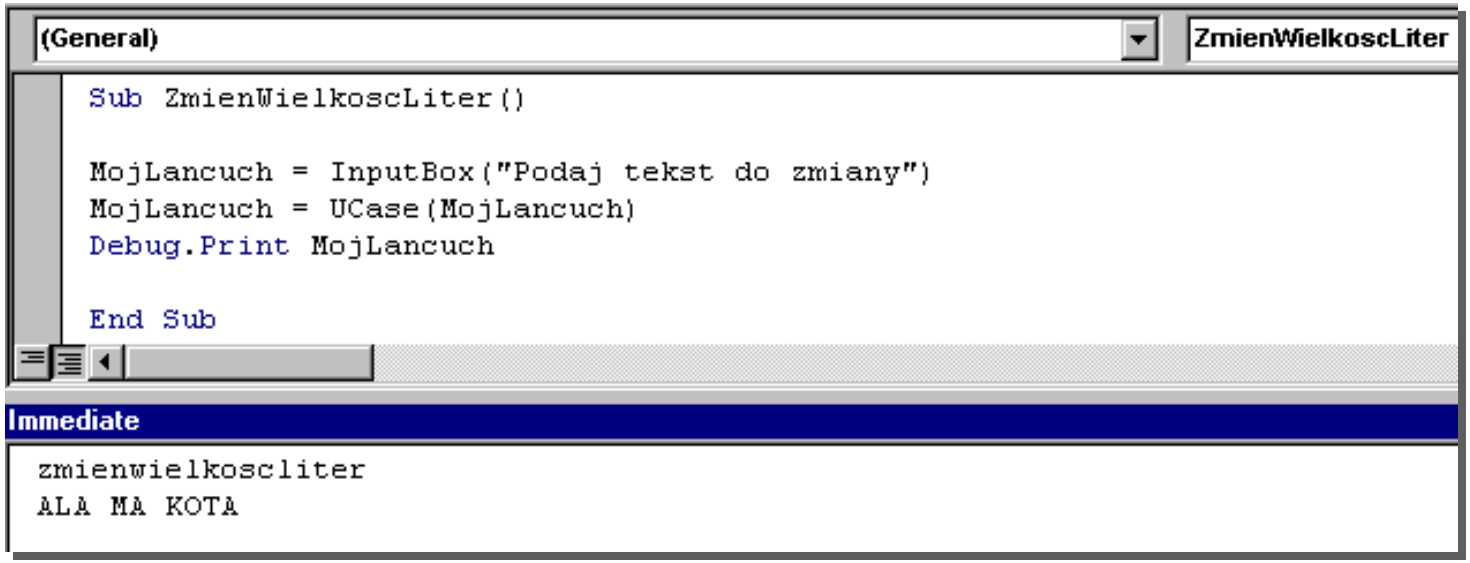
- Instrukcje bezpośrednie funkcji wpisuje się w okno *Immediate* poprzedzając je znakiem zapytania, a w celu wyświetlenia wyniku należy wcisnąć ENTER. Wynik wyświetlony zostanie w oknie *Immediate* poniżej.



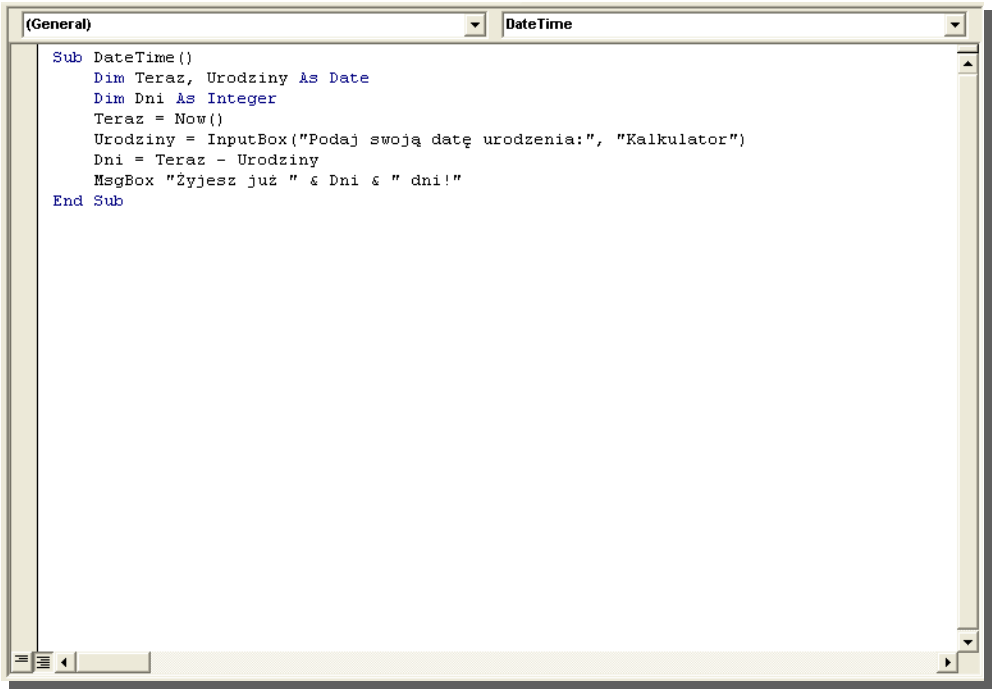
Instrukcje bezpośrednie

- Aby wykonać procedurę w oknie *Immediate* należy wpisać jej nazwę w tym oknie i nacisnąć klawisz ENTER.

- Można też wstawiać do procedur polecenia wyświetlające wyniki bezpośrednio w oknie *Immediate*. Służy do tego celu polecenie `Debug.Print`



Elementy okna: **Kod programu**



- Każdy element projektu ma związane ze sobą okno kodu np.:
  - Sam skoroszyt (*ThisWorkbook*);
  - Arkusz;
  - Moduł;
  - Formularz użytkownika;
  - Formanty
- Aby wyświetlić okno kodu dla danego obiektu, wystarczy kliknąć dwukrotnie jego nazwę w oknie **Eksploratora projektu**.

Typy danych (1)

Poprzez **typ danych** definiujemy jaki rodzaj danych ma przechowywać zmienna oraz jaki obszar pamięci zostanie jej przydzielony.

VBA potrafi ustalić automatycznie typ danych.
Wady takiego rozwiązania:

- Wolniejsze wykonywanie programu.
- Nieefektywne wykorzystanie pamięci.

Deklarowanie typu zmiennej

- Jeśli brak jest w programie deklaracji zmiennej używany jest typ domyślny **`Variant`**.
- Dane przechowywane jako dane typu **`Variant`** zmieniają swój typ w zależności od wykonywanych na nich operacjach.

Poniższa procedura demonstruje przyjmowanie różnych typów danych przez taką zmienną:

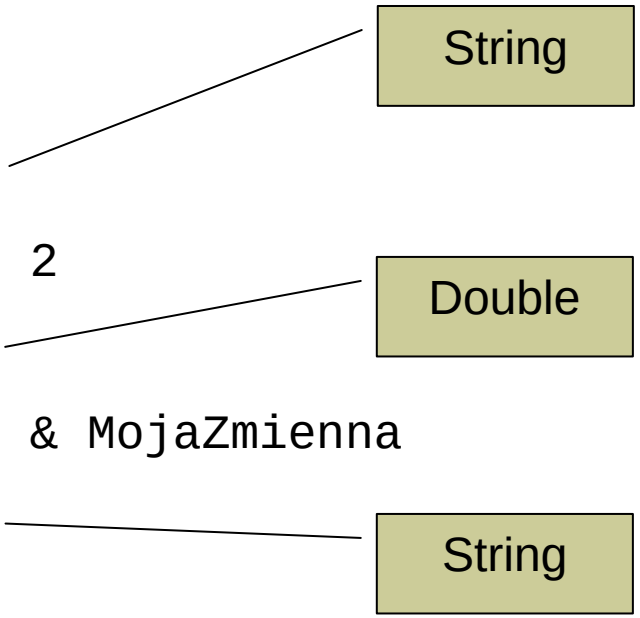
```
Sub VariantDemo()
        MojaZmienna = „123”
        MojaZmienna = MojaZmienna / 2
        MojaZmienna = „Odpowiedź: ” & MojaZmienna
        MsgBox MojaZmienna
End Sub
```

Ustalanie typu danych

- Aby ustalić typ danych zmiennej można skorzystać z funkcji o nazwie **`TypeName`**

Oto modyfikacja poprzedniej procedury. Po każdym kroku wyświetlany będzie typ danych zmiennej `MojaZmienna`:

```
Sub VariantDemo()
        MojaZmienna = „123”
        MsgBox TypeName (MojaZmienna)
        MojaZmienna = MojaZmienna / 2
        MsgBox TypeName (MojaZmienna)
        MojaZmienna = „Odpowiedź: ” & MojaZmienna
        MsgBox TypeName (MojaZmienna)
        MsgBox MojaZmienna
End Sub
```



Typy danych (2)

Typ danych	Zakres wartość	Zajmowany obszar pamięci (bajty)
Boolean	**True** lub **False**	1
Byte	od **0** do **255**	1
Integer	od **-32768** do **+32767**	2
Long	od **-2147483648** do **+2147483647**	4
Single	od **–3,402823E38** do **–1,401298E-45** od **1,401298E-45** do **3,402823E38**	4
Double	od **–1,79769313486232E308** do **-14,94065645841247E-324** od **4,94065645841247E-324** do **1,79769313486232E308**	8
Currency	od **-922337685477,5808** do **+922337203477,5807**	8
Date	od **1 stycznia 100** do **31 grudnia 9999**	8
Object	referencja do dowolnego obiektu	4
String (stała długość)	od **1** do **65535** znaków	długość łańcucha
String (zmienna długość)	od **0** do **2 miliardów** znaków	22 + długość łańcucha
Variant (tylko z liczbami)	**dowolna wartość** (z zakresu typu Double)	16
Variant	od **0** do około **2 miliardów** znaków	22 + długość łańcucha

Wymuszanie deklarowania wszystkich zmiennych

- Aby wymusić deklarowanie wszystkich używanych zmiennych wystarczy umieścić na początku modułu VBA następującą instrukcję:

```
Option Explicit
```

- powoduje to zatrzymanie wykonania programu po każdym natrafieniu na nazwę zmiennej, która nie została wcześniej zadeklarowana.

Deklarowanie typu zmiennej

- Najbardziej popularny sposób deklarowania zmiennej lokalnej to użycie instrukcji **Dim** (skrót od *Dimension*) np.:

```
Dim NumerWiersza As Integer
Dim x As Long
Dim DataAktualna As Date
Dim Oprocentowanie As Single
Dim ZawartoscKomorki As String
Dim Imie As String * 20
Dim Temp
Dim i, j, x As Integer
Dim i As Integer, j As Integer, x As Integer
Dim i As Integer, j As Double
```

Zasięg zmiennych (1)

Zmienne o zasięgu całego modułu:

Instrukcja **Dim** przed pierwszą procedurą w module:

```
Dim WartoscCalkowita As Integer

Sub Proc1()
…
End Sub

Sub Proc2()
…
End Sub
```

Zasięg zmiennych (2)

Zmienne widoczne we wszystkich modułach:

Instrukcja **`Public`** przed pierwszą procedurą w module:

(tylko w modułach standardowych VBA, nie można używać zmiennych typu Public w modułach arkusza czy formularza)

```
Public WartoscCalkowita As Integer

Sub Proc1()
…
End Sub

Sub Proc2()
…
End Sub
```

Zasieg zmiennych (3)

Zmienne lokalne:

- Instrukcje **Dim, Static, Private** wewnątrz procedury:

```
Sub NowaProc()
    Dim x As Integer
    Static Licznik As Integer
    Private NumerWiersza As Integer
End Sub
```

- Zmienne lokalne zadeklarowane ze pomocą instrukcji **Dim** oraz **Private** są usuwane z pamięci po zakończeniu procedury.
- Zmienne lokalne zadeklarowane ze pomocą instrukcji **Static** zachowują swoją wartość nawet po zakończeniu procedury np.:

```
Sub Proc()
    Static Licznik As Integer
    Licznik = Licznik + 1
    MsgBox(”Procedura została wykonana już: ” _
          & Licznik & ”razy”)
End Sub
```

Stosowanie zmiennych

- **Zmienne boolowskie:**

```
Dim x As Boolean
x = True
```

- **Zmienne całkowite:**

```
Dim x As Integer
x = 3
```

- **Zmiennoprzecinkowe:**

```
Dim x As Double
x = 3.14
```

- **Łańcuchowe:**

```
Dim x As String
x = ”Ala ma kota”
```

- **Data i czas:**

```
Dim d As Date
d = #1/1/2000#
d = #12:00:00#
```

Zmienne obiektowe (1)

- **Zmienne obiektowe**:
  - Reprezentują całe obiekty takie jak zakres czy arkusz.
  - Pozwalają uprościć kod programu.
  - Umożliwiają szybsze wykonywanie programu.
  - Deklaracja:

```
Dim o As Object        Public o As Object
```

  - Zmiennej obiektowej przypisujemy wartość przy pomocy słowa kluczowego **Set**:

```
Set o = Worksheets("Ark1").Range("A1")
```

  - Przykłady:

```
Dim zakres As Range
Set zakres = Worksheets("Ark1").Range("A1")
Dim arkusz As Worksheet
Set arkusz = Worksheets("Ark1")
```

Zmienne obiektowe (2)

- Upraszczanie kodu. Drugi wariant prezentuje procedurę z zadeklarowaną zmienną obiektową:

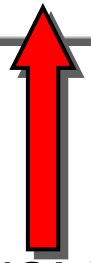
```
Sub BezZmObiekt()
   Worksheets(„Arkusz1”).Range(„A1”).Value = 124
   Worksheets(„Arkusz1”).Range(„A1”).Font.Bold = True
   Worksheets(„Arkusz1”).Range(„A1”).Font.Italic = True
End Sub
```

```
Sub ZmObiekt()
   Dim MojaKomórka As Range
   Set MojaKomórka = Worksheets(„Arkusz1”).Range(„A1”)
   MojaKomórka.Value = 124
   MojaKomórka.Font.Bold = True
   MojaKomórka.Font.Italic = True
End Sub
```

Deklarowanie stałych

- Wartość stałej nie zmienia się podczas wykonywania programu.
- Deklaracje stałej np.:

```
Const PI As Single = 3.14
Const Oprocentowanie As Single = 0.075
Const Stopa = 0.44, Okres = 12
Public Const LiczbaPunktow As Integer = 5
```



Zwykle podaje się w deklaracji typ danych, gdyż stała nigdy nie zmienia swej wartości.

Uwaga: Stałe podobnie jak zmienne mają swój zasięg.

Tablice (1)

- **Tablica:** grupa elementów tego samego typu występujących pod wspólną nazwą.
- **Deklarowanie tablic** wygląda podobnie jak deklaracje zmiennych przy czym należy dodatkowo podać wartość [początkową indeksu oraz] wartość końcową indeksu.
- Odwołanie do elementu tablicy następuje poprzez podanie jej nazwy oraz indeksu.

Przykłady:

```
Dim Tablica1(1 To 100) As String * 2
Dim Tablica2(0 To 100) As String * 2
Public Tablica3(100) As String * 20
```

VBA przyjmuje domyślnie dolną wartość indeksu 0

Tablice (2)

Aby pominąć dolną wartość indeksu i spowodować, że w każdej tablicy tą wartością jest domyślnie np. 1 należy umieścić przed dowolną procedurą w module instrukcję:

```
Option Base 1
```

Deklarowanie tablic wielowymiarowych

Tablice w VBA mogą mieć do 60 wymiarów.

```
Dim Tablica4(1 To 10, 1 To 10) As Integer

Dim Tablica5(0 To 10, 0 To 20, 0 To 100) _
As Long
```

Tablice (3)

Odwoływanie się do elementów tablicy:

- Jednowymiarowej:

```
Tablica1(4) = ”Ala ma kota”
```

- Dwuwymiarowej:

```
Tablica4(4,5) = 25
```

- Trójwymiarowej:

```
Tablica5(1,2,10) = 15
```

Typy danych definiowane przez użytkownika

- VBA pozwala utworzyć własne typy danych.
- Definiowanie typu danych musi przebiegać poza procedurami, na początku modułu. Typy takie są zawsze publiczne.

```
Type KlientInfo
        Firma As String * 20
        Kontakty As String * 15
        Kierunkowy As Integer
        DziałSprzedaży As Long
End Type
```

W module należy zadeklarować zmienną tego typu – zwykle jest to deklaracja tablicy:

```
Dim Klienci(1 To 100) As KlientInfo
```

Odwołanie do składników tej zmiennej

- **Odwołanie do poszczególnych składników tablicy:**

```
Klienci(1).Firma = „InterFlora”
Klienci(1).Kontakt = „Anna Madej”
Klienci(1).Kierunkowy = 3
Klienci(1).DziałSprzedaży = 159806
```

- **Wykonywanie operacji na całych elementach takiej tablicy:**

```
Klienci(2) = Klienci(1)
```

Zamiast

```
Klienci(2).Firma = Klienci(1).Firma
Klienci(2).Kontakt = Klienci(1).Kontakt
```

itd.

Przypisywanie wyrażeń

- **Operator przypisania** wartości wyrażenia do zmiennej to w VBA znak równości: **=**
- Przykłady:

```
x = 1
x = x + 1
y = x*x + 2*x –10
ActiveWindow.DisplayGridlines = True
Cells(2,3).Value = 102
```

Operatory matematyczne

+	dodawanie
-	odejmowanie
*****	mnożenie
/	dzielenie
^	podnoszenie do potęgi
&	łączenie łańcuchów
****	dzielenie całkowite
Mod	dzielenie modulo (z resztą)

Operatory logiczne

Not	negacja logiczna wyrażenia
And	iloczyn logiczny dwóch wyrażeń
Or	suma logiczna dwóch wyrażeń
XoR	suma rozłączna dwóch wyrażeń
Eqv	spr. równoważności logicznej dwóch wyrażeń
Imp	implikacja dwóch wyrażeń

Operatory porównania

=	równe
>	większe niż
<	mniejsze niż
>=	większe lub równe
<=	mniejsze lub równe
<>	różne od

Sterowanie wykonaniem

- **Sterowanie wykonaniem** polega na pomijaniu lub wielokrotnym wykonywaniu pewnych fragmentów kodu oraz sprawdzaniu warunków w celu ustalenia dalszego przebiegu wykonania procedury.
- Najważniejsze sposoby sterowania wykonaniem:
  - Konstrukcja **`If – Then`**
  - Konstrukcja **`Select Case`**
  - Pętla **`For – Next`**
  - Pętla **`Do While`**
  - Pętla **`Do Until`**
  - Instrukcja **`GoTo`**

If – Then (1)

Warunkowe wykonywane jednej lub wielu instrukcji.

1

```
If warunek Then instrukcja
```

2

```
If warunek Then
   instrukcja prawda
End If
```

3

```
If warunek Then
   instrukcja prawda
Else
   instrukcja fałsz
End If
```

4 Zagnieżdżenie instrukcji

```
If warunek Then
instrukcja prawda
Else
        If warunek2 Then
        instrukcja prawda2
        Else
        instrukcja fałsz2
        End If
End If
```

If - Then (2)

5

```
If warunek Then
   instrukcje prawda
ElseIf warunek2 Then
   instrukcje prawda2
ElseIf warunek3 Then
   instrukcje prawda3
Else
   instrukcje domyślne
End If
```

Testowanie wielu warunków

6

```
If Ilość >= 0 And Ilość < 25 Then
   Rabat = 0.03
ElseIf Ilość >= 25 And Ilość < 50 Then
   Rabat = 0.05
ElseIf Ilość >= 50 And Ilość < 75 Then
   Rabat = 0.08
ElseIf Ilość >= 75
   Rabat = 0.11
End If
```

Select Case (1)

Warunkowe wykonywane jednej lub wielu instrukcji.
Alternatywa dla `If – Then - Else`

```
Select Case wyrażenieTestujące
   Case listaWyrażeń1
        instrukcje1
   Case listaWyrażeń2
        instrukcje2
   Case Else
         instrukcje domyślne
End Select
```

```
Select Case Ilość
Case 0 To 24
        Rabat = 0.03
Case 25 To 49
        Rabat = 0.05
Case 50 To 74
        Rabat = 0.08
Case Is >= 75
        Rabat = 0.11
End Select
```

Wyjście z konstrukcji `Select case` następuje po znalezieniu i wykonaniu pierwszego pasującego przypadku – aby uzyskać maksymalnie efektywny kod, można sprawdzać najczęściej zachodzący przypadek na samym początku.

Select Case (2)

- Zagnieżdżanie instrukcji

```
Select Case Application.WindowState
   Case xlMaximized: MsgBox „Aplikacja maksymalna”
   Case xlMinimized: MsgBox „Aplikacja minimalna”
   Case xlNormal: MsgBox „Aplikacja normalna”
       Select Case ActiveWindow.WindowState
          Case xlMaximized: MsgBox „Zeszyt maksymalny”
          Case xlMinimized: MsgBox „Zeszyt minimalny”
          Case xlNormal: MsgBox „Zeszyt normalny”
       End Select
End Select
```

For - Next

Pętla umożliwia wykonywanie tego samego bloku wielokrotnie.
Liczba wykonań pętli jest znana, z góry ustalona.

```
For licznik = start To koniec [Step przyrost]
   [Instrukcje]
   [Exit For]
   [Instrukcje]
Next [licznik]
```

Licznik – zmienna sterująca

```
For licz = 1 To 10
   suma = suma + licz
Next licz
```

```
For licz = 1 To 10 Step 2
   suma = suma + licz
Next licz
```

```
For x = 0 To 10
  For y = 0 To 100
      suma = suma + tabela(x, y)
  Next y
Next x
```

Zagnieżdżenie instrukcji

For Each - Next

Pętla umożliwia wykonywanie operacji dla wszystkich elementów kolekcji

```
For Each element In grupa
   [Instrukcje]
   [Exit For]
   [Instrukcje]
Next [element]
```

```
Dim tablica(5) As Integer

For i = 0 To 5
   Tablica(i) = Rnd
Next i

For Each n In tablica
   Debug.Print n
Next n
```

Do While

Pętla wykonuje się dopóki podany warunek jest spełniony.

1

```
Do While [warunek]
   [Instrukcje]
   [Exit Do]
   [Instrukcje]
Loop
```

2

```
Do
   [Instrukcje]
   [Exit Do]
   [Instrukcje]
Loop While [warunek]
```

```
Dim licz As Integer
licz = 10
Do While licz >= 1
   suma = suma + licz
   licz = licz - 1
Loop
```

```
Do
   ActiveCell.Value = 0
   ActiveCell.Offset(1, 0).Select
Loop While Not IsEmpty(ActiveCell)
```

Do Until

Pętla wykonuje się do momentu aż warunek zostanie spełniony.

1
```
Do [Until warunek]
   [Instrukcje]
   [Exit Do]
   [Instrukcje]
Loop
```

2
```
Do
   [Instrukcje]
   [Exit Do]
   [Instrukcje]
Loop [Until warunek]
```

```
Dim licz As Integer
licz = 10

Do Until licz < 1
   suma = suma + licz
   licz = licz - 1
Loop
```

Procedury - **Sub**

- Procedura to grupa instrukcji umieszczonych w module, wykonujących określone zadanie.

```
[Private | Public] [Static] Sub nazwa (lista_arg)
   [instrukcje]
   [Exit Sub]
   [instrukcje]
End Sub
```

Private – procedura jest dostępna tylko dla pozostałych procedur z tego samego modułu.

Public – procedura jest dostępna dla wszystkich procedur z wszystkich modułów. Standardowo wszystkie procedury są publiczne.

Static – zmienne procedury są zachowywane po jej zakończeniu.

Przekazywanie argumentów (3)

- Argument może być przekazywany do procedury na dwa sposoby:

- **`ByRef`** – przez referencję (standardowo) – polega na przekazaniu adresu pamięci zmiennej;

- **`ByVal`** – przez wartość – polega na przekazaniu „*kopii*” oryginalnej zmiennej. W konsekwencji zmiana argumentu w procedurze nie ma wpływu na oryginalną zmienną.

Wywoływanie procedur

- Z wnętrza innej procedury:

```
NazwaProcedury                TwojaSub
Call NazwaProcedury           Call TwojaSub
```

- Aby precyzyjnie wskazać procedurę w innym skoroszycie, trzeba podać nazwę projektu, nazwę modułu i na końcu nazwę procedury:

```
MójProjekt.MójModuł.MojaSub
Call MójProjekt.MójModuł.MojaSub
```

- Innym sposobem jest użycie metody **Run** obiektu **Application**:

```
Application.Run „’makra_budżetowe.xls’!Konsoliduj”
```

  - Wywołanie procedury `Konsoliduj` znajdującej się w skoroszycie `makra_budżetowe.xls`

Wywołanie procedury w momencie wystąpienia zdarzenia

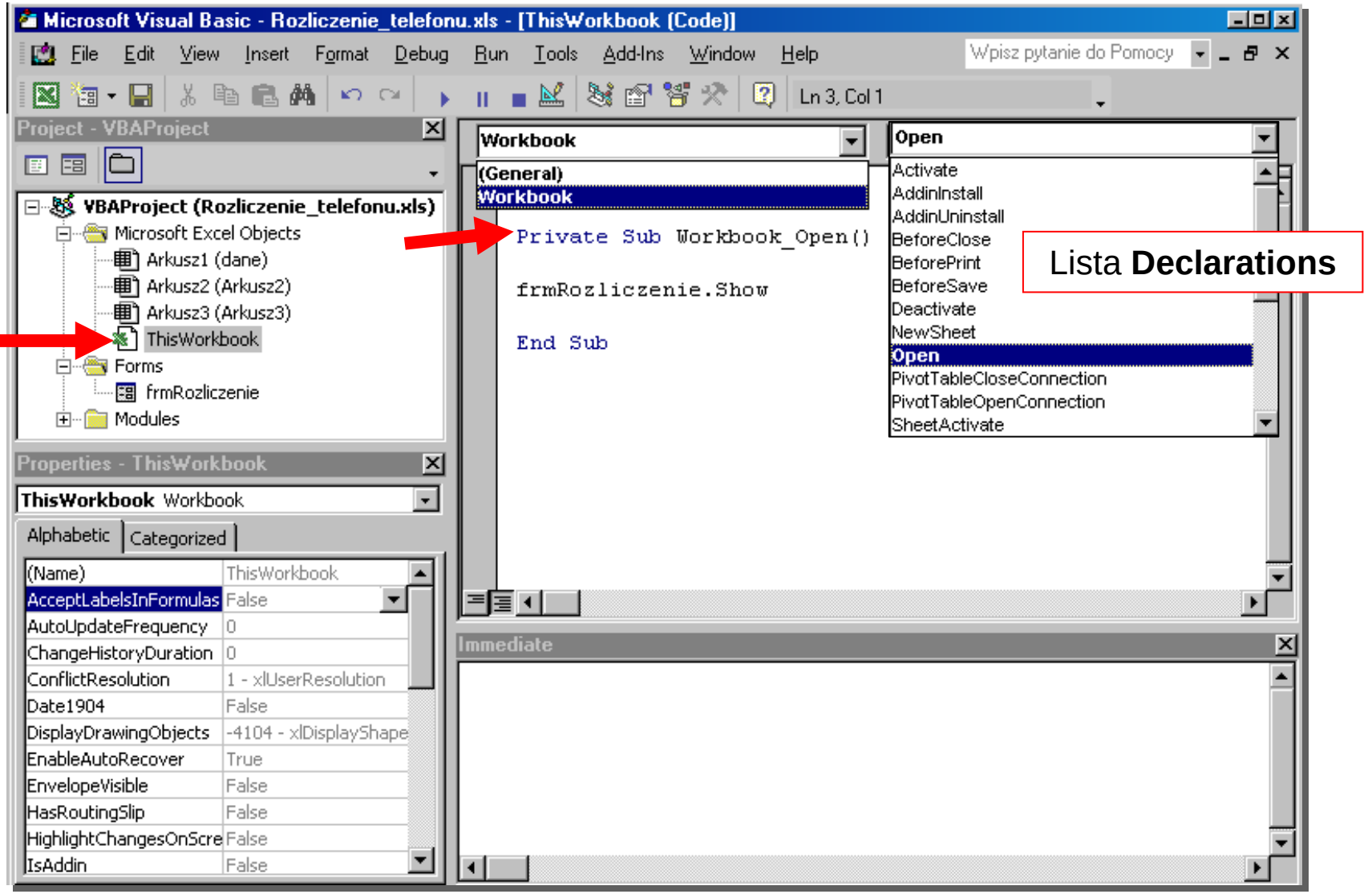
- Zdarzeniem jest np. otwarcie skoroszytu, zaznaczenie komórki, kliknięcie obiektu, zapisanie skoroszytu itd. Procedura taka nazywa się **Procedurą obsługi zdarzenia**.
- Procedury takie mają charakterystyczne nazwy składające się z nazwy obiektu, znaku podkreślenia i nazwy zdarzenia.

  Np. procedura wykonywana w momencie otwierania skoroszytu ma nazwę:

  `Workbook_Open`
- Procedury te są przechowywane w oknie kodu konkretnego obiektu.

Przykład:

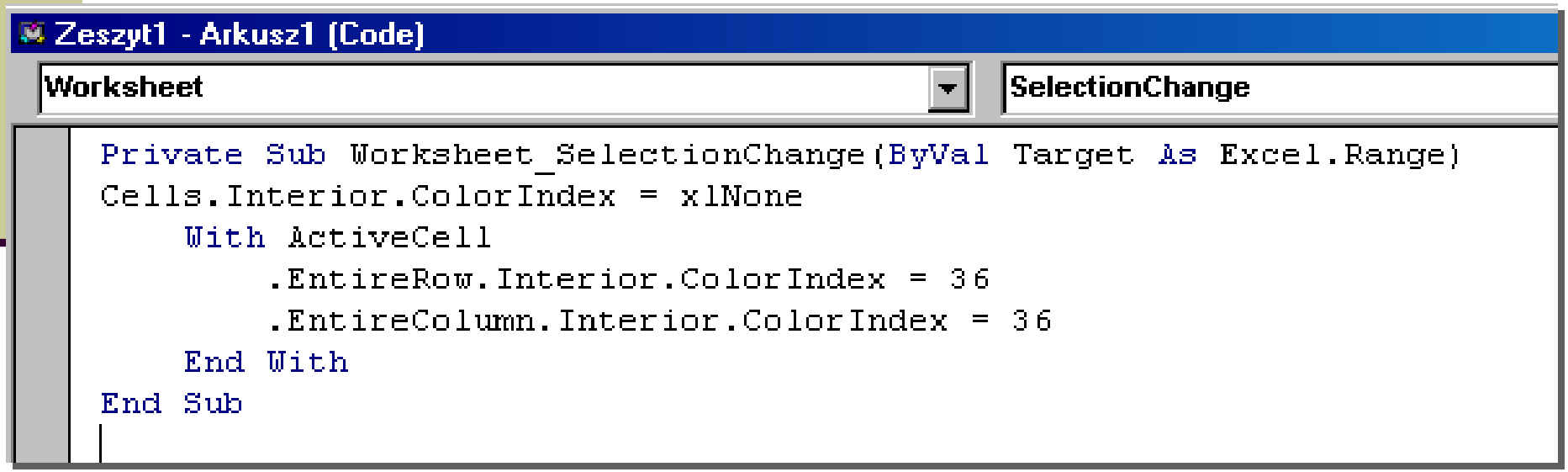
Procedura **Workbook_Open**



Przykład procedury obsługującej zdarzenie

Worksheet_SelectionChange

- Jest wykonywana za każdym razem, gdy użytkownik zmieni zaznaczenie na arkuszu.
- Usuwa cieniowanie aktywnej komórki.
- Kolumna i wiersz aktywnej komórki zmieniają kolor na jasnożółty.



Zdarzenia nie związane z obiektami Excela

- Zdarzenia występujące na poziomie aplikacji. Powinny być umieszczane w ogólnych modułach VBA.
  - **OnTime** – występuje w określonym czasie
  - **OnKey** – występuje po kliknięciu przycisku.

(General) SetAlarm

```
Sub SetAlarm()
    Application.OnTime TimeValue("3:00:00 pm"), "DisplayAlarm"
    End Sub

Sub DisplayAlarm()
    Beep
    MsgBox "Czas na przerwę obiadową!"
End Sub
```

Funkcje - **Function**

- Są to procedury, których zadaniem jest wykonanie określonych operacji (obliczeń) i zwrócenie ich wyniku.
- Funkcje należy tworzyć i umieszczać w modułach.
- Jeden moduł może zawierać dowolną liczbę funkcji.

```
[Private | Public] [Static] Function nazwa [(lista_arg)] [As typ]
   [instrukcje]
   [Exit Function]
   [instrukcje]
End Function
```

Wywoływanie funkcji

- Z wnętrza procedury:

```
Razem = SumaTab(tab)
```

- W formułach arkusza:

```
=SumaTab(A1:A100)
=Zeszyt2.xls!SumaTab(A1:A100)
```

Funkcje wbudowane

- VBA udostępnia całą gamę funkcji wbudowanych, które pozwalają uprościć obliczenia i operacje (ponad 130 funkcji).
- Często pozwalają one wykonać działanie, które inaczej byłoby bardzo trudne lub wręcz niemożliwe do zrealizowania.
- Wiele funkcji VBA przypomina funkcje Excela (wiele jest nawet identycznych). Na przykład funkcja **`UCase`**, która zmienia litery łańcucha na wielkie, jest równoważna excelowej funkcji arkusza **LITERY.WIELKIE**.

Funkcje wbudowane – przegląd (1)

- **`Len`** – zwraca ilość znaków łańcucha lub liczbę bajtów pamięci, jaka jest potrzebna do przechowywania argumentu.
- **Funkcje numeryczne**:
  - **`Sqr`** – oblicza pierwiastek kwadratowy,
  - **`Atn`** – zwraca wartość arcustangens liczby,
  - **`Cos`** – zwraca wartość cosinus liczby,
  - **`Exp`** – zwraca podstawę logarytmu naturalnego liczby podniesionej do odpowiedniej potęgi,
  - **`Sin`** – zwraca wartość sinus liczby,
  - **`Tan`** - zwraca wartość tangens liczby.
  - **`Log`** - zwraca wartość logarytmu naturalnego liczby.

Funkcje wbudowane – przegląd (2)

- **Funkcje łańcuchowe**:
  - **`Chr(int)`** – zwraca znak ASCII o kodzie `int`.

    `Zmienna = Chr(84)` wynik: `T`
  - **`UCase(str)`** – zwraca wartość typu `Variant(String)` zawierającą podany ciąg `str` zamieniony na litery wielkie.

    `Zmienna = UCase(„Ala ma kota”)` wynik: `ALA MA KOTA`
  - **`LCase(str)`** – zamienia wszystkie litery w ciągu `str` na małe.
  - **`LTrim(str)`**, **`RTrim(str)`**, **`Trim(str)`** – każda z tych funkcji zwraca wartość typu `Variant(string)` zawierająca kopię podanego ciągu znaków: bez spacji wiodących (`LTrim`), bez spacji końcowych (`RTrim`) lub bez spacji wiodących i końcowych (`Trim`).

    `MsgBox („B” & LTrim(„ Ala ma kota ”) & „B”`

    wynik: `BAla ma kota B`

Funkcje wbudowane – przegląd (3)

- **Funkcje łańcuchowe**:
  - **`Left(str, int)`** – zwraca wartość typu `Variant(String)` zawierającą podaną liczbę znaków `int` począwszy od lewej strony ciągu znaków `str`.
  - **`Mid(str, intStart [, intLen])`** - zwraca wartość typu `Variant(String)` zawierającą podaną liczbę znaków z ciągu znaków, od podanej pozycji.
  - **`Right(str, int)`** – zwraca wartość typu `Variant(String)` zawierającą podaną liczbę znaków `int` począwszy od prawej strony ciągu znaków `str`.
  - **`Str(liczba)`** – zwraca znakową reprezentację liczby.

Funkcje łańcuchowe - przykład

- `zmienna = **Left**("Ala ma kota", 3)` *wynik:* `Ala`
- `zmienna = **Right**("Ala ma kota", 4)` *wynik:* `kota`
- `zmienna = **Mid**("Ala ma kota", 5, 2)` *wynik:* `ma`

Funkcje wbudowane – przegląd (4)

- **Funkcje daty i czasu**:
  - **`Date`** – zwraca bieżącą datę systemową.
  - **`DateAdd`** – dodaje przedział czasowy do danej daty.
  - **`DateDiff`** – zwraca przedział czasowy między dwoma datami.
  - **`Day`** – zwraca dzień miesiąca podanej daty.
  - **`Month`** – zwraca miesiąc podanej daty.
  - **`Minute`** – zwraca liczbę określającą minutę.
  - **`Now`** – zwraca bieżącą datę i czas systemowy.
  - **`Time`** – zwraca bieżący czas systemowy.
  - **`Weekday`** – zwraca numer oznaczający dzień tygodnia.

Funkcje daty i czasu - przykład

- `zmienna = **Now()**` *wynik:* `2006-10-29 09:40:00`
- `zmienna = **Year**(Now)` *wynik:* `2006`
- `zmienna = **Month**(Now)` *wynik:* `10`
- `zmienna = **Day**(Now)` *wynik:* `29`
- `zmienna = **WeekDay**(Now)` *wynik:* `7`

Funkcje wbudowane – przegląd (5)

- **Funkcje logiczne**:
  - **`IsArray`** – zwraca `True`, jeśli zmienna jest tablicą.
  - **`IsDate`** – zwraca `True`, jeśli zmienna jest datą.
  - **`IsNull`** – zwraca `True`, jeśli wyrażenie nie zawiera danych.
  - **`IsNumeric`** – zwraca `True`, jeśli wyrażenie można potraktować jako wartość numeryczną.

```
varSprawdzajaca = IsNumeric(varSprawdzana)
```

Wywoływanie funkcji Excela

- Jeżeli VBA nie ma odpowiednika funkcji używanej w Excelu, można wywołać funkcję arkuszową Excela bezpośrednio z poziomu programu VBA. Wystarczy, że poprzedzi się jej nazwę odwołaniem do obiektu `WorksheetFunction`.
- Na przykład VBA nie ma funkcji zamieniającej radiany na stopnie. Ponieważ Excel ma funkcję arkuszową służącą właśnie do tego, można wywołać następującą instrukcję VBA:

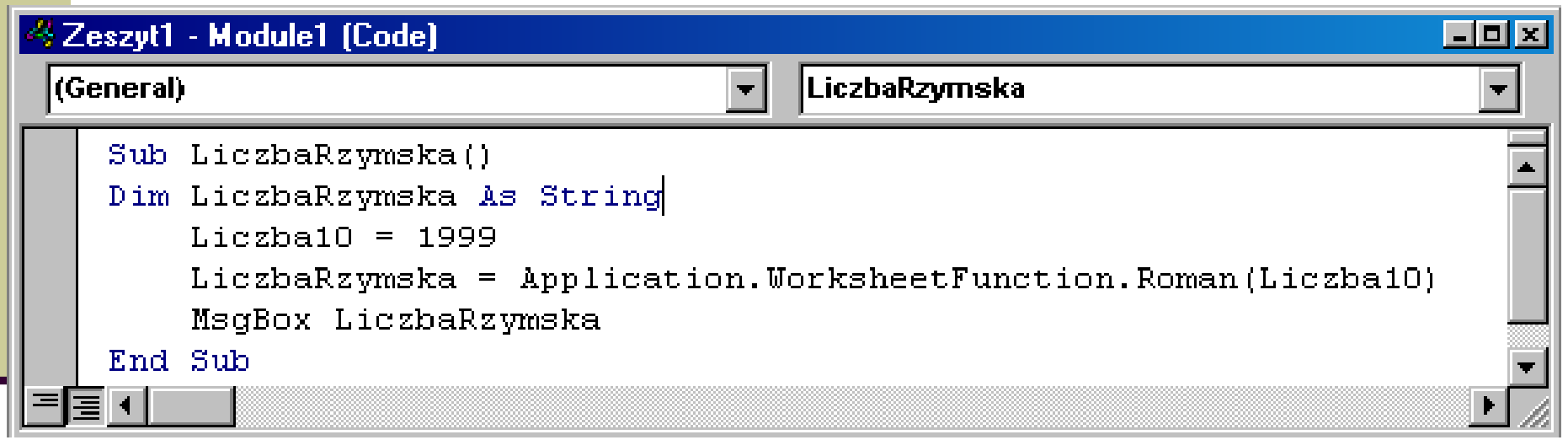
```
Deg =
Application.WorksheetFunction.Degrees(3.14)
```

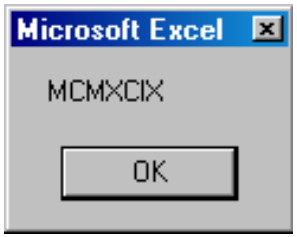
- Obiekt `WorksheetFunction` został wprowadzony w Excelu 97. Aby zachować zgodność z poprzednimi wersjami, można pominąć odwołanie do obiektu `WorksheetFunction` następująco:

```
Deg = Application.Degrees(3.14)
```

Funkcje Excela - przykład

- Poniższy przykład demonstruje sposób użycia w procedurze VBA excelowej funkcji arkusza ROMAN. Ta rzadko wykorzystywana funkcja dokonuje konwersji liczby dziesiętnej na rzymską.





Funkcja **MsgBox**

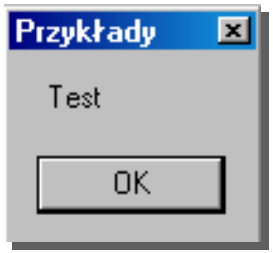
- Funkcja **MsgBox** jest jedną z najbardziej użytecznych funkcji VBA. Funkcja ta zwraca pojedynczą wartość w oknie dialogowym w formie komunikatu, ale także może udostępniać użytkownikowi różne przyciski.
- Podstawowa składnia wywołania funkcji **MsgBox** obejmuje pięć argumentów (opcjonalne są umieszczone w nawiasach kwadratowych):

  `MsgBox(`*`prompt`*`[, `*`buttons`*`] [, `*`title`*`] [, `*`helpfile`*`] [, `*`context`*`])`

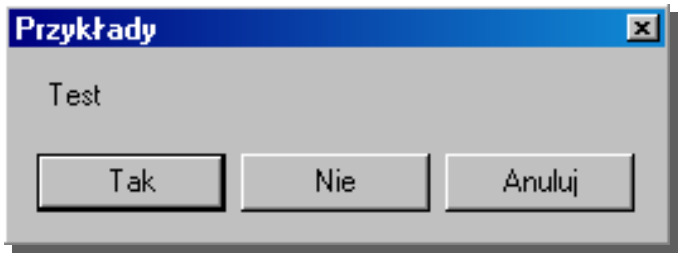
  - *prompt* – wyświetlana wiadomość
  - *buttons* – to przyciski, np. `vbYesNo` (przyciski Tak i Nie)
  - *title* – tytuł okna na pasku komunikatu (standardowo jest to *Microsoft Excel*)
  - *helpfile* – plik pomocy
  - *context* – ID tematu pomocy, wskazuje konkretny temat pomocy, który ma być wyświetlany.

Funkcja **MsgBox**

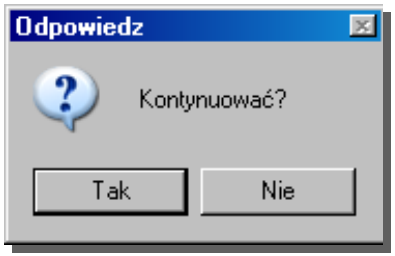
`vbOKOnly`



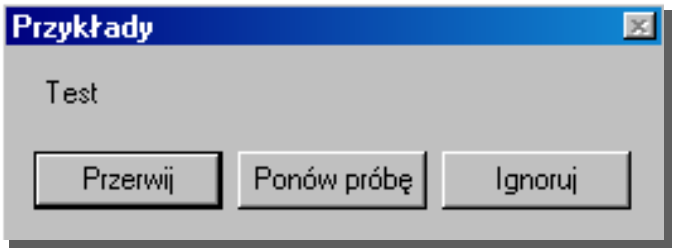
`vbYesNoCancel`



`vbYesNo`

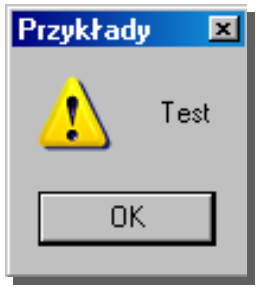


`vbAbortRetryIgnore`

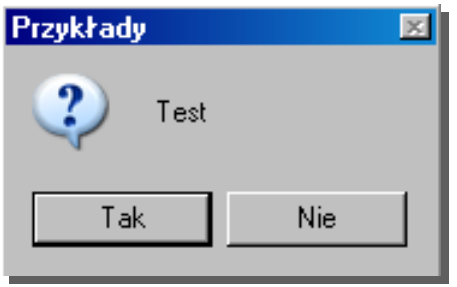


Funkcja **MsgBox**

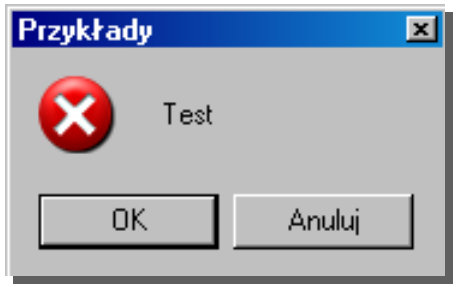
vbOKOnly +vbExclamation



vbYesNo +vbQuestion



vbOKCancel +vbCritical



vbOKCancel +vbInformation

Funkcja **MsgBox**

- Wartość zwracana przez funkcję **`MsgBox`** jest liczbą całkowitą:
  - 1 - **`vbOK`** (Akceptuj).
  - 2 – **`vbCancel`** (Anuluj).
  - 3 – **`vbAbort`** (Przerwij).
  - 4 – **`vbRetry`** (Ponów próbę).
  - 5 – **`vbIgnore`** (Ignoruj).
  - 6 – **`vbYes`** (Tak).
  - 7 – **`vbNo`** (Nie).

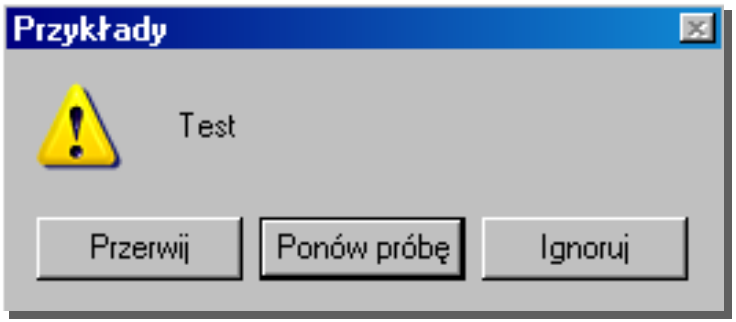
Funkcja **MsgBox**

- Argument **buttons** jest opcjonalny. Zamiast nazw można użyć wartości liczbowych z tabel sterujących, które są odpowiednikami argumentu:
  - **vbOKOnly - 0**
  - **vbOKCancel - 1**
  - **vbAbortRetryIgnore - 2**
  - **vbYesNoCancel - 3**
  - **vbYesNo - 4**
  - **vbRetryCancel - 5**
  - **vbCritical – 16**
  - **vbQuestion – 32**
  - **vbExclamation – 48**
  - **vbInformation - 64**
  - **vbDefaultButton1 – 0** – przyciskiem domyślnym jest przycisk 1
  - **vbDefaultButton2 – 256** – przyciskiem domyślnym jest przycisk 2
  - **vbDefaultButton3 – 512** – przyciskiem domyślnym jest przycisk 3

Funkcja **MsgBox**

- Liczby, które są odpowiednikami poszczególnych opcji argumentu **`Buttons`** można sumować:

```
Sub Przykład()
    zmienna = "Test"
    MsgBox zmienna, 306, "Przykłady"
End Sub
```

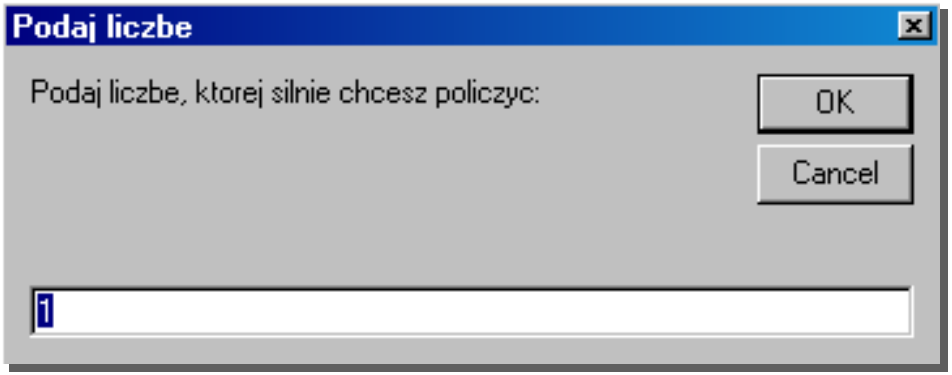


306 = 2 + 48 + 256

2 – `vbAbortRetryIgnore`
48 – `vbExclamation`
256 – `domyślny przycisk 2`

Funkcja **InputBox**

- Funkcja **`InputBox`** wyświetla zapytanie o jedną informację.
- Zawsze zwraca łańcuch, więc w przypadku wprowadzania wartości liczbowych konieczne jest dokonanie konwersji jej wyniku.
- Zapytanie może liczyć około 1024 znaków.
- Oprócz niego można określić tytuł okna dialogowego, wartość domyślną oraz jego położenie na ekranie.
- Można też dodać własny temat pomocy. W takim przypadku pole wprowadzania danych zawiera przycisk *Pomoc*.



Składnia funkcji **InputBox**

```
InputBox(prompt[, title][, default]
[, xpos][, ypos][, helpfile, context])
```

- Znaczenie argumentów:
  - ***prompt*** – zapytanie (wymagany) – tekst wyświetlany w polu wprowadzania danych.
  - ***title*** – tytuł okna (opcjonalny).
  - ***default*** – wartość domyślna (opcjonalna).
  - ***xpos, ypos*** – współrzędne ekranowe lewego górnego rogu okna (opcjonalne).
  - ***helpfile, context*** – plik pomocy i ID tematu pomocy (opcjonalne).

Przykład z funkcją **InputBox** (2)

```
Sub ObliczSr()
  Dim Licz1 As Integer
  Dim Licz2 As Integer
  Dim Licz3 As Integer
    Licz1 = CInt (InputBox („Wpisz pierwszą liczbę:„)
    Licz2 = CInt (InputBox („Wpisz drugą liczbę:„)
    Licz3 = CInt (InputBox („Wpisz trzecią liczbę:„)
  MsgBox („Średnia wynosi: „ & Chr(13) & Sr(Licz1,
  Licz2, Licz3)
End Sub
```

```
Function Sr(Licz1 As Integer, Licz2 As Integer, Licz3
  As Integer) As Single
  Const ileLiczb As Integer = 3
  Sr = (Licz1 + Licz2 + Licz3) / ileLiczb
End Function
```

Konwersja typów danych

- Zamiana dowolnego typu zmiennej na określony przez zastosowaną funkcję typ.
- Najważniejsze funkcje konwersji:

  CBool, CByte, CInt, CLng, CSng, CDbl, CStr, CVar, CDate, CCur.

- Przykład:

```
Ilosc = CInt(”123”)
Wyplata  =  CCur (Godziny * StawkaGodz)
```

Uwaga: Jeżeli wyrażenie przekazywane do funkcji wykracza poza przedział dopuszczalnych wartości dla typu danych, do którego próbowano dokonać konwersji, wystąpi błąd.

Obsługa błędów

- Podczas wykonywania programu mogą pojawić się błędy np. dzielenie przez zero lub brak dyskietki w stacji dyskietek.
- Wyrażenie **`On Error Go To etykieta`** sprawia, że w wypadku błędu nastąpi przeniesienie do bloku zaczynającego się od linii **`etykieta:`**.
- Wyrażenie **`On Error Resume Next`** sprawia, że w wypadku błędu przechodzimy do wykonania kolejnej linii programu.
- Samodzielna instrukcja **`Resume`** powoduje, że program powraca do miejsca, w którym wystąpił błąd, np. w sytuacji, gdy program próbuje coś odczytać z dysku A, a w stacji dysków brakuje dyskietki może ukazać się stosowny komunikat. Po włożeniu dyskietki naturalny jest powrót do linii, w której wystąpił błąd.
- Wyłączenie obsługi błędu następuje po **`On Error Go To 0`**.

Obsługa błędów

- **`Resume linia`** skacze do linii o numerze **`linia`** (lub etykiety) w obrębie tej samej procedury. **`Linia`** nie może być zerem.
- Kod błędu zwraca funkcja **`Err`**, zaś komunikat wyjaśniający, co się stało, zwraca funkcja **`Error(Err)`**.

Hierarchia obiektów

- Na szczycie hierarchii obiektów znajduje się obiekt **`Application`** reprezentujący program Excel.
- Obiekt **`Application`** zawiera 47 innych obiektów np.
  - `Workbooks`
  - `Window`
- Obiekt **`Workbooks`** zawiera wszystkie otwarte obiekty **`Workbook`**, a te z kolei zawierają następne np.:
  - `Worksheets`
  - `Charts`
  - `Names`

Odwołania do obiektów

- Odwołanie do arkusza:

```
Application.Workbooks(„Zeszyt1.xls”).
Worksheets(„Arkusz1”)
```

- Odwołanie do komórki w arkuszu:

```
Application.Workbooks(„Zeszyt1.xls”).
Worksheets(„Arkusz1”).Range(„A1”)
```

- Jeśli wiadomo, że Arkusz1 jest aktywnym arkuszem, można uprościć odwołanie stosując zapis:

```
Range(„A1”)
```

Właściwości obiektów

- Obiekty mają **właściwości**.
- Na przykład obiekt **Range** ma właściwość **Value** i **Name** (wartość i nazwa), a obiekt **Chart** ma właściwość **HasTitle** (tytuł) i **Type** (typ).
- Używając **VBA** można ustalić właściwości obiektu, a także zmienić je. W zapisie należy podać odwołanie do obiektu i jego właściwości oddzielając dane kropką, np.

```
Worksheets(„Arkusz1”).Range(„A1”).Value
```

- Zmiennym VBA można przypisywać **wartości**. Zapis poniżej podaje jak przypisać wartości komórki A1 do zmiennej **Stopa**.

```
Stopa = Worksheets(„Arkusz1”).Range(„A1”).Value
Worksheets(„Arkusz1”).Range(„A1”).Value = Stopa
```

Metody obiektów

- Obiekty mają **metody**.
- **Metoda to działanie wykonywane przez obiekt**.
- Na przykład jedną z metod obiektu **Range** jest `ClearContents` (usuwanie zawartość komórek zakresu).

  Oto przykład zapisu:

```
Range(„A1”).ClearContents
```

Właściwości obiektu **Range**

- Każdy obiekt posiada właściwości określające jego stan:
  - Obiekt **`Range`** posiada właściwość **`Value`** (wartość)

```
Application.Workbooks(1).Worksheets(1).Range(„A1”).Value
Range(„A1”).Value = 14
```

  - właściwość **`Font`** (czcionka)

```
Range(„A1”).Font.Bold
```

  - właściwość **`Interior`** (wnętrze)

```
Range(„A1”).Interior.ColorIndex = 15
```

  - właściwość **`Borders`** (obramowanie)

```
Range(„A1”).Borders.ColorIndex = 2
```

- Wartości można odczytywać lub zapisywać (zmieniać):

```
Zmienna = Worksheets(1).Range(„A1”).Value
```
– odczyt

```
Worksheets(1).Range(„A1”).Value = Zmienna
```
– zapis

Właściwości obiektu **Application**

- Właściwości obiekt **`Application`** ułatwiają pracę z komórkami i zakresami:
  - **`ActiveCell`** – aktywna komórka

```
ActiveCell.Value = 14
```

  - **`ActiveSheet`** – aktywny arkusz

```
ActiveSheet.Name = „Mój arkusz”
```

  - **`ActiveWindow`** – aktywne okno
  - **`ActiveWorkbook`** – aktywny skoroszyt

```
ActiveWorkbook.Name = „Dane personalne”
```

  - **`Selection`** – zaznaczony obiekt

```
Selection.Value = 12
```

Uwaga, w tym wypadku próba przypisania wygeneruje błąd, gdy zaznaczony będzie obiekt inny niż zakres (np. wykres)

  - **`ThisWorkbook`** – skoroszyt zawierający wykonywaną procedurę

Właściwości obiektu **Worksheet**

- Właściwość **Range**:
  - Wstawianie wartości do komórki **A1** w arkuszu **„Arkusz1”** :

```
Worksheets(„Arkusz1”).Range(„A1”).Value = 14
```

  - Wstawianie wartości do komórki o nazwie **„Blok”** :

```
Range(„Blok”).Value = 2
```

  - Wstawianie wartości do komórek od **A1** do **B100** :

```
Range(„A1:B100”).Value = 123
Range(„A1”, „B100”).Value = 123
```

  - Wstawianie wartości do komórek od **A1**, **A10**, **A15** :

```
Range(„A1, A10, A15”).Value = 123
```

Właściwości obiektu **Worksheet**

- Właściwość **`Cells`**:

`Cells(x, y)`

x to numer wiersza (1 – 65536)

y to numer kolumny (1 – 256)

- Wstawianie wartości do komórki **`A3`** w arkuszu **`„Arkusz1”`** :

```
Worksheets(„Arkusz1”).Cells(3, 1).Value = 14
Worksheets(„Arkusz1”).Cells(513).Value = 14
```

- Wykasowanie zawartości wszystkich komórek:

```
Worksheets(„Arkusz1”).Cells.ClearContents
```

Właściwości obiektu **Worksheet**

- Właściwość **`Offset`**:

`Offset(x, y)`

x to przesunięcie o x wierszy

y to przesunięcie o y kolumn

- Wstawianie wartości do komórki bezpośrednio pod komórką **`A3`** w arkuszu **`„Arkusz1”`** :

```
Worksheets(„Arkusz1”).Cells(3, 1).Offset(1,0).Value = 14
```

- Wstawianie wartości do komórki bezpośrednio nad komórką **`A3`** w arkuszu **`„Arkusz1”`** :

```
Worksheets(„Arkusz1”).Cells(3, 1).Offset(-1,0).Value =
14
```

Metody obiektu

- Obiekty posiadają także metody czyli działania, które można wykonać z udziałem obiektu np:
  - Metoda **Clear** usuwa zawartość wybranej komórki:

    ```
    Worksheets(1).Range(”A1”).Clear
    ```

    ```
    Range(”A1”).Copy Range(„B1”)
    ```

  - Jeżeli metoda ma argumenty należy je podać po jej nazwie oddzielając kolejne przecinkami:

    Metoda **Protect** obiektu **Workbook** używa trzech argumentów – hasło, struktura i okno (odpowiadają opcjom w oknie dialogowym **Chroń skoroszyt**)

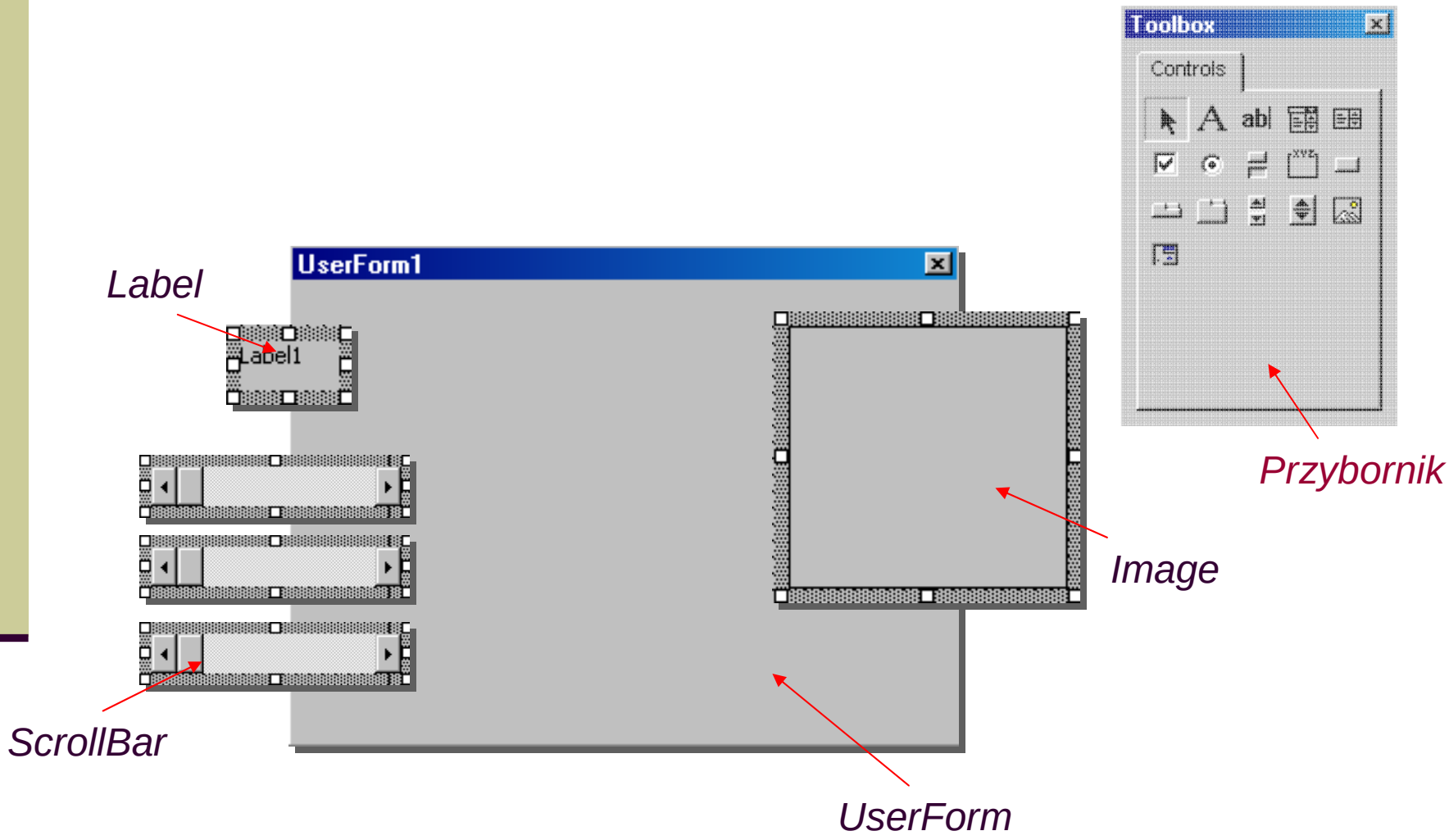
    ```
    Workbooks(„Dane poufne”). Protect „hasło”, True, True
    ```

    ```
    Workbooks(„Dane poufne”). Protect , True, True
    ```

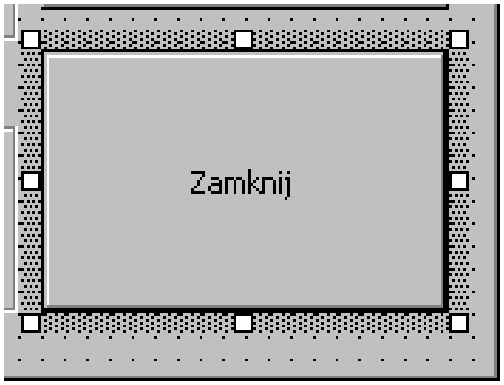
    ```
    Workbooks(„Dane poufne”). Protect Structure:=True,
      Windows:=True
    ```

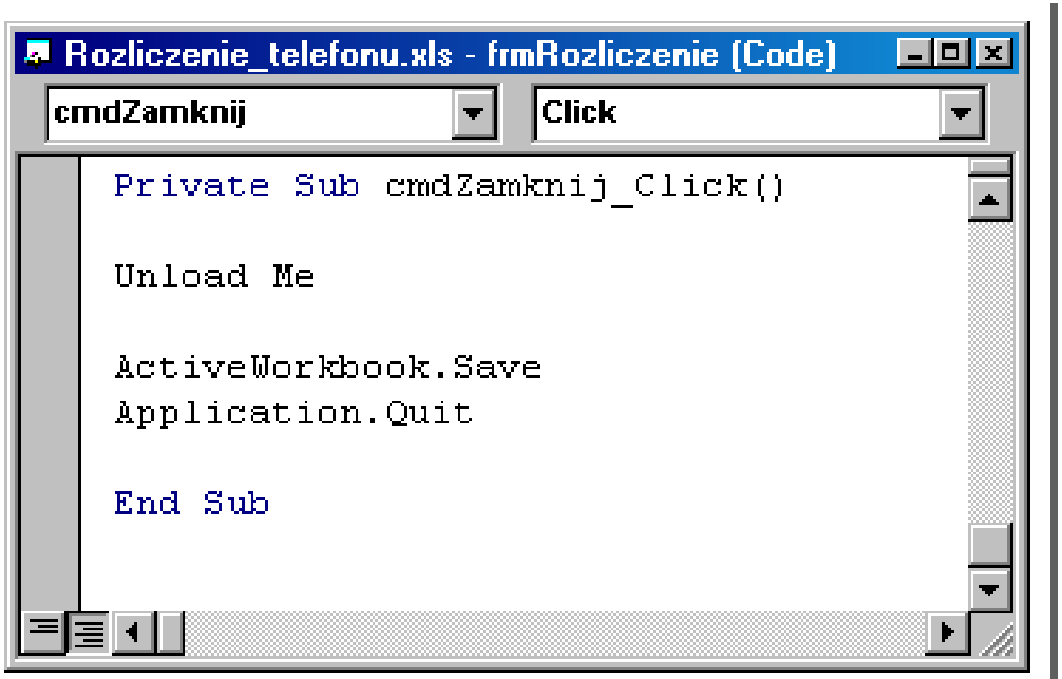
    - Nazwanie metod w celu ich rozróżnienia

Formanty graficzne



Programowanie formantów





Wyświetlanie formularza

- Dla celów testowych:
  - Wybranie pozycji z menu pozycji *Run -> Run Sub/UserForm*
  - Naciśnięcie klawisza [*F5*].
  - Kliknięcie przycisku *Run Sub/UserForm* na standardowym pasku narzędziowym.

- Z poziomu kodu VBA:

  Stworzenie procedury wyświetlającej formularz np:

```
Sub Dialog
  Load UserForm1
  UserForm1.Show
End Sub
```

Zamykanie formularza

- Standardowo formularz zamykany jest po naciśnięciu przycisku **X** w prawym górnym rogu okna.

- Z poziomu kodu VBA.

  Ukrycie formularza:

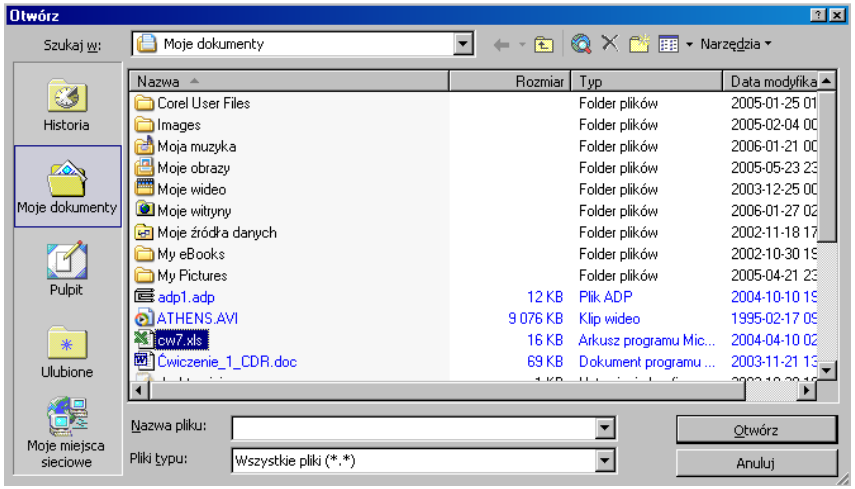
```
UserForm1.Hide
```

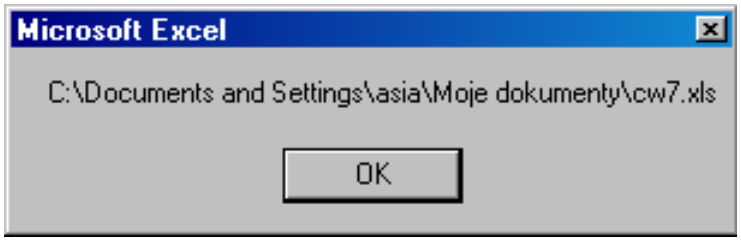
  Usunięcie formularza z pamięci komputera:

```
UnLoad UserForm1
```

Metoda GetOpenFilename

- Metoda obiektu `Application`.
- Gwarantuje uzyskanie poprawnej nazwy pliku i ścieżki dostępu.
- Wyświetla standardowe okno dialogowe **Otwórz**.
- Nie otwiera pliku, tylko zwraca łańcuch zawierający ścieżkę i nazwę pliku zgodnie ze wskazaniem użytkownika.





Metoda GetOpenFilename

- Składnia metody (wszystkie argumenty są opcjonalne):

```
obiekt.GetOpenFilename (FileFilter,
  FilterIndex, Title, ButtonText, Multiselect)
```

- **`FileFilter`** – decyduje o pozycjach dostępnych w oknie dialogowym w polu listy **Pliki typu**. Argument ma postać par składających się z łańcucha opisującego filtr oraz definiujących go symboli wieloznacznych. Poszczególne pary muszą być oddzielone przecinkami.
- Wartość domyślna tego argumentu to:

`„Wszystkie pliki (*.*),*.*”`

Metoda GetOpenFilename

- Inne wartości np.:
- **„Pliki tekstowe (*.txt),*.txt,”**
- **„Pliki ASCII (*.asc),*.asc,”**

- **FilterIndex** – wskazuje domyślny filtr plików.

- **Title** – zawiera tekst wyświetlany na pasku tytułu okna dialogowego.

- **ButtonText** – nie jest używany w Excelu.

- **MultiSelect** – wielokrotne zaznaczenie – ma wartość True; użytkownik może zaznaczyć w oknie dialogowym wiele plików jednocześnie, ich nazwy zostaną wówczas zwrócone w tablicy.

- Dziękuję za uwagę

- Test
- 6.06.2007, godzina 8.15
- 15.06.2007, godzina 8.15