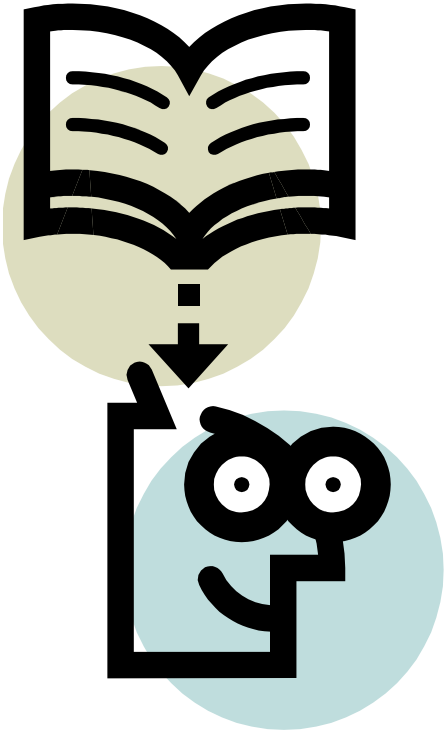
Programowanie w Visual Basic

- Wykład



Opracowanie:

Łukasz Sturgulewski

Literatura:

John Walkenbach „*Programowanie Excel 2000 Visual Basic*”, Wydawnictwo RM, Warszawa 2000

Visual Basic dla Aplikacji (VBA)

- Historia.
  - BASIC: *Begginer's All-purpose Symbolic Instruction Code*.
  - Powstanie: Początek lat 60-tych.
  - Przeznaczenie: Nauka technik programowania.
  - Rozwinięcie: Visual Basic for Windows (1991 rok).
  - Excel 5: Pierwsza aplikacja wyposażona w VBA.
  - Obecnie: Wiele aplikacji różnych producentów jest wyposażonych w VBA.
- VBA – oparty na BASIC'u ale obecnie nie mający z nim wiele wspólnego.
- VBA – wspólny język skryptowy dla aplikacji.

Edytor Visual Basic

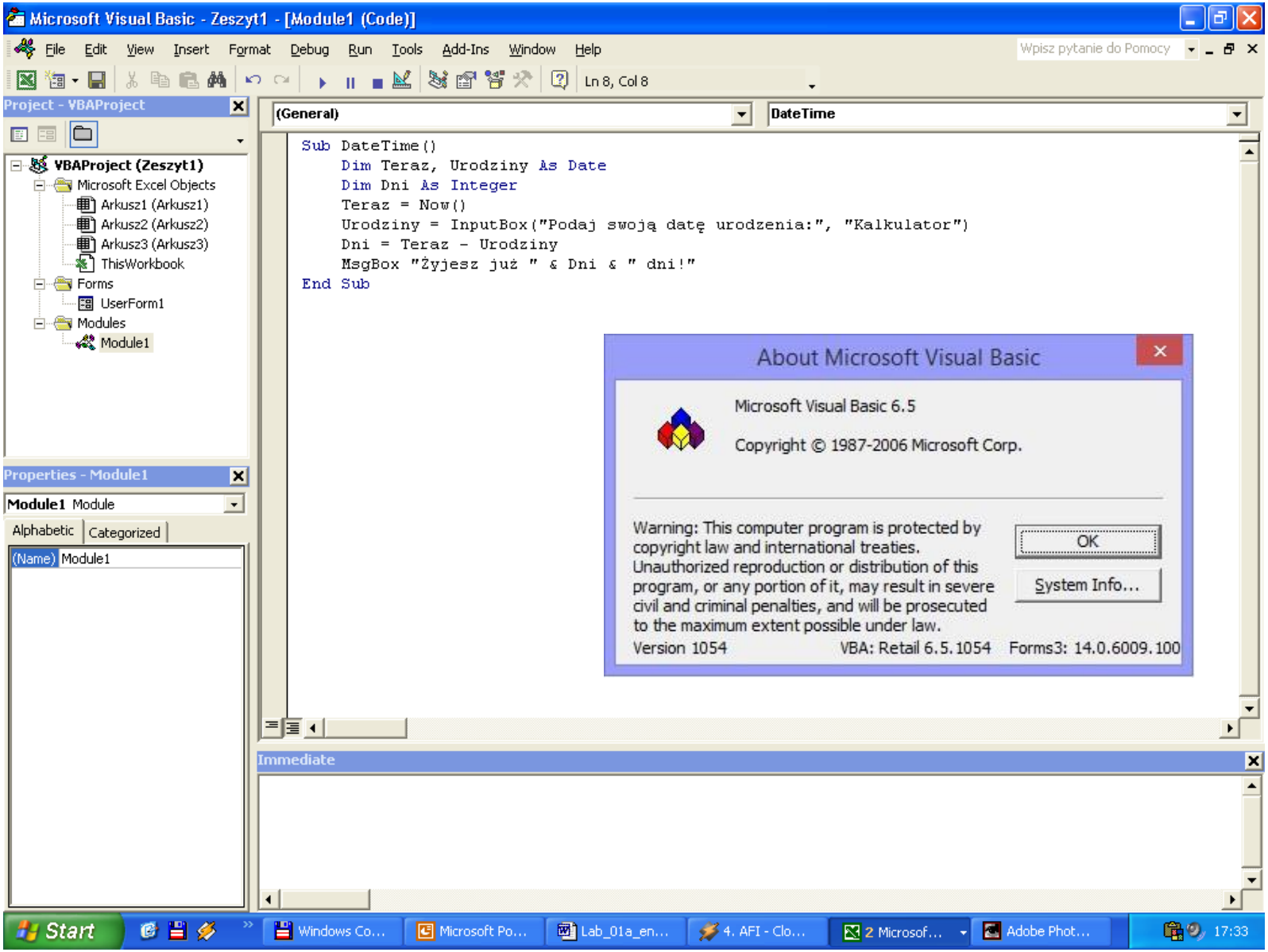
- W Excelu 5 i Excelu 95 moduł Visual Basic pojawiał się jako oddzielny arkusz skoroszytu.
- Począwszy od Excela 97 moduły Visual Basic nie tworzą już oddzielnych arkuszy.
  Zamiast tego są wyświetlane i edytowane za pomocą Edytora Visual Basica (VBE).
  Moduły Visual Basic są nadal przechowywane w plikach skoroszytów. Nie są jedynie widoczne, dopóki nie zostanie uruchomiony VBE.
- **Edytor Visual Basic** jest oddzielną aplikacją ściśle współpracującą z Excelem. Polega to na tym, że Excel sam troszczy się o wszystkie szczegóły dotyczące otwierania edytora w razie potrzeby. Nie można jednak uruchomić Edytora Visual Basic w trybie niezależnym - aby działał musi być uruchomiony Excel**.**

Uruchamianie Edytora Visual Basic

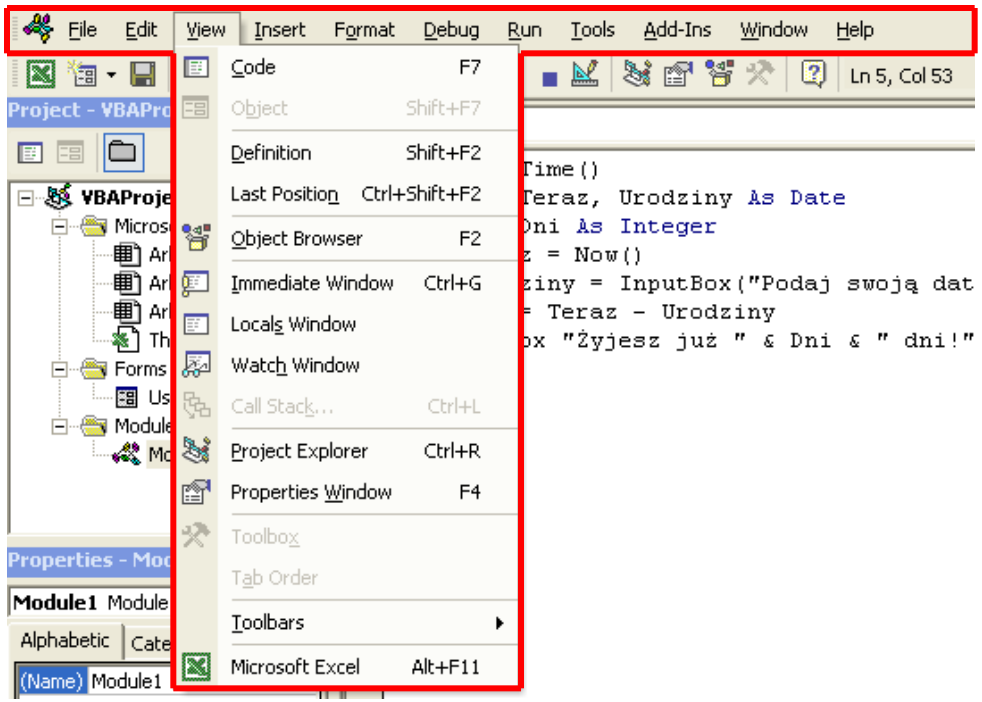
- Okno Edytora Visual Basic można otworzyć na trzy różne sposoby:
  - Wybrać z menu polecenie *Deweloper-> Visual Basic*.
  - Nacisnąć kombinację klawiszy [*Alt+F11*].
- Okno Edytora Visual Basic pokazane jest na następnym slajdzie.

Uwaga: Twoje okno może wyglądać nieco inaczej (!) ze względu na szerokie możliwości dostosowywania. Między innymi można ukrywać poszczególne okna wewnętrzne, zmieniać ich rozmiar, „dokować", przemieszczać itp.

Okno Edytora Visual Basic



Elementy okna: **Pasek menu**

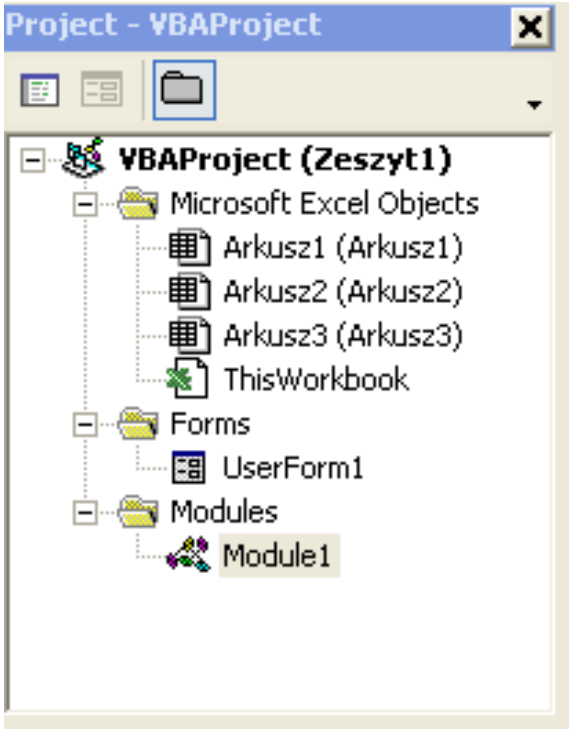


- Pasek menu zwiera polecenia używane do wykonywania operacji w edytorze.
- Wiele poleceń z menu ma przypisane skróty klawiszowe np. polecenie *View -> Project Explorer* ma przypisany klawisz skrótu [*Ctrl+R*].

Elementy okna: **Paski narzędzi**

- Paski narzędzi są zwykle wyświetlane tuż poniżej paska menu.
- Standardowy pasek narzędzi, jest jednym z kilku pasków dostępnych w edytorze.
- Można je dostosowywać do własnych potrzeb tzn. przemieszczać, ukrywać, wyświetlać, zmieniać zawartość itp. Aby zmienić pasek narzędzi, należy kliknąć na dowolnym z nich prawy przycisk myszy i wybrać z menu pozycję *Dostosuj*.

Elementy okna: **Eksplorator projektu**



- W oknie Eksploratora projektu (*Project*) wyświetlane są diagramy zawierające wszystkie skoroszyty otwarte aktualnie w Excelu (łącznie ze skoroszytami ukrytymi).
- Każdy skoroszyt nazywany jest projektem.
- Jeśli okno Eksplorator projektu nie jest widoczne należy wybrać z menu pozycję *View -> Project Explorer* lub nacisnąć kombinację klawiszy [*Ctrl+R*].
- Aby ukryć to okno, należy kliknąć w jego prawym górnym rogu przycisk: **X**

Elementy okna: **Eksplorator projektu**

- W trakcie pracy z Edytorem Visual Basic każdy aktualnie otworzony skoroszyt Excela traktowany jest jako projekt.
- Aby „rozszerzyć” projekt, wystarczy kliknąć znak ‘+’ na lewo od jego nazwy w oknie Eksploratora projektu. W celu „zwinięcia” projektu trzeba z kolei kliknąć znak ‘-‘.
- W przypadku próby rozwinięcia zabezpieczonego projektu, na ekranie pojawi się pytanie o hasło.
- Każdy rozwinięty projekt zawiera „węzeł” o nazwie *Microsoft Excel Objects*. Przechowuje on elementy odpowiadające wszystkim arkuszom i wykresom zawartym w skoroszycie oraz obiekt o nazwie *ThisWorkbook* reprezentujący aktywny skoroszyt.
- Jeśli projekt zawiera jakiekolwiek moduły, na liście występuje ponadto węzeł *Modules*. Po jego rozwinięciu pojawia się wykaz wszystkich istniejących modułów.
- Projekt może także zawierać węzeł o nazwie *Forms* reprezentujący formularze użytkownika.

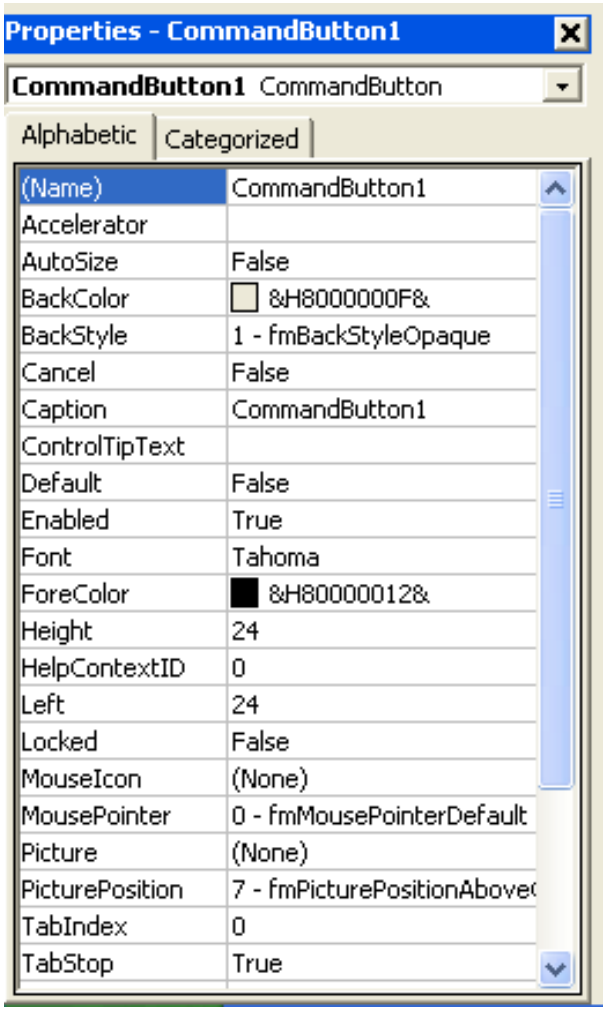
Elementy okna: **Eksplorator projektu**

- Dodawanie nowego modułu Visual Basic:

  Aby dodać do projektu nowy moduł należy zaznaczyć nazwę projektu w oknie Eksploratora projektu i wybrać z menu polecenie *Insert -> Module*.
  Można też kliknąć nazwę projektu prawym przyciskiem myszy i wybrać to samo polecenie z menu podręcznego.
- Usuwanie modułu Visual Basic:

  Aby usunąć moduł z projektu, należy zaznaczyć nazwę tego modułu w oknie Eksploratora projektu i wybrać z menu polecenie *File -> Remove „nazwa modułu”.*
  Można też kliknąć nazwę modułu prawym przyciskiem myszy i wybrać to samo polecenie z menu podręcznego.
  Przed usunięciem modułu pojawi się pytanie, o eksport do pliku.

Elementy okna: **Właściwości obiektu**

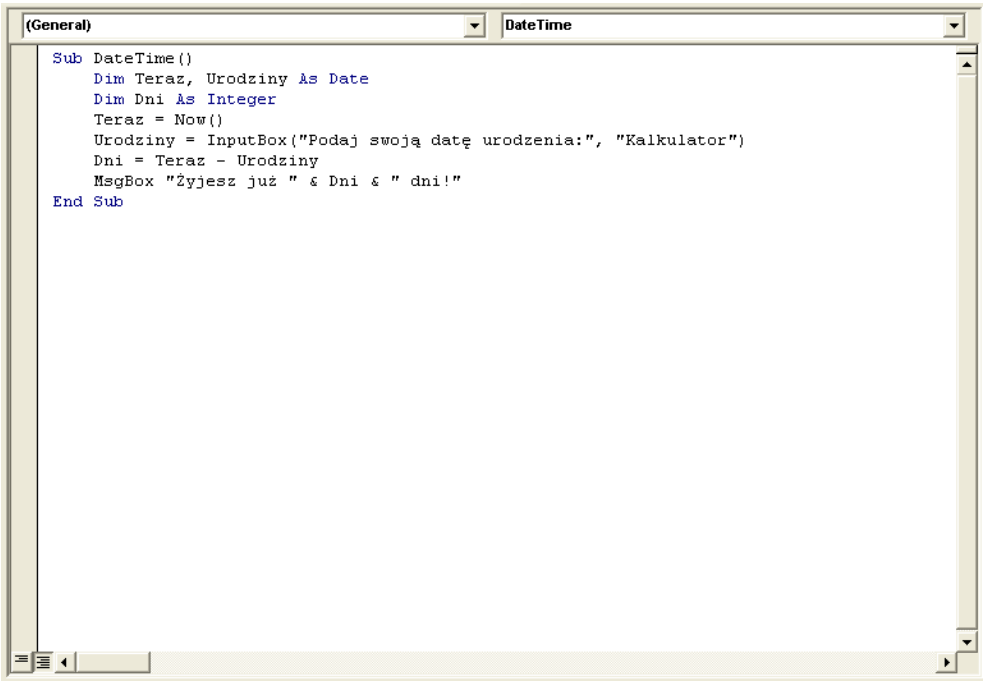


- W oknie Właściwości projektu (*Properties*) wyświetlane są właściwości wybranego obiektu (arkusza, formularza, przycisku itp.).
- Zakładka *Alphabetic*: Wyświetlanie listy właściwości obiektu w porządku alfabetycznym.
- Zakładka *Categorized*: Wyświetlanie właściwości obiektu według kategorii.
- Jeśli okno właściwości nie jest widoczne należy wybrać z menu pozycję *View -> Properties Window* lub nacisnąć klawisz [*F4*].
- Aby ukryć to okno, należy kliknąć w jego prawym górnym rogu przycisk: **X**

Elementy okna:
Instrukcje bezpośrednie

- Okno Instrukcji bezpośrednich (*Immediate*) służy do bezpośredniego wprowadzania i wykonywania instrukcji VB (testowanie, śledzenie kodu).
- Aby wywołać to okno należy wybrać z menu pozycję *View -> Immediate Window* lub nacisnąć kombinację klawiszy [*Ctrl*+*G*].
- Aby ukryć to okno, należy kliknąć w jego prawym górnym rogu przycisk: **X**

Elementy okna: **Kod programu**



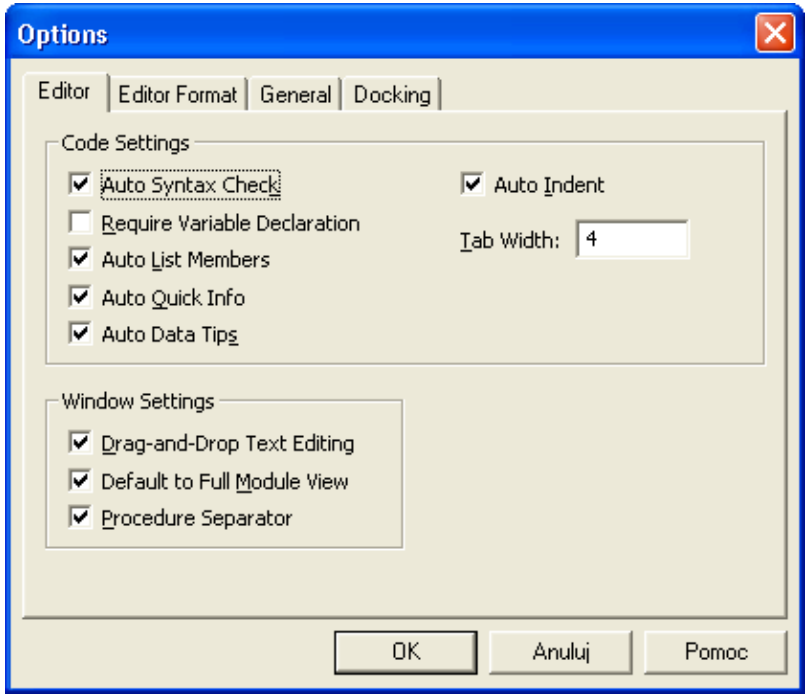
- Każdy element projektu ma związane ze sobą okno kodu np.:
  - Sam skoroszyt (ThisWorkbook);
  - Arkusz;
  - Moduł;
  - Formularz użytkownika;
- Aby wyświetlić okno kodu dla danego obiektu, wystarczy kliknąć dwukrotnie jego nazwę w oknie Eksploratora projektu.

- Kod do modułu można dodać na trzy sposoby:
  - Wpisać przy użyciu klawiatury.
  - Użyć *Rejestratora makr*, rejestrującego wszystkie wykonane operacje i zamieniającego je na odpowiedni kod.
  - Skopiować kod z innego modułu.

Dostosowywanie środowiska Edytora Visual Basic

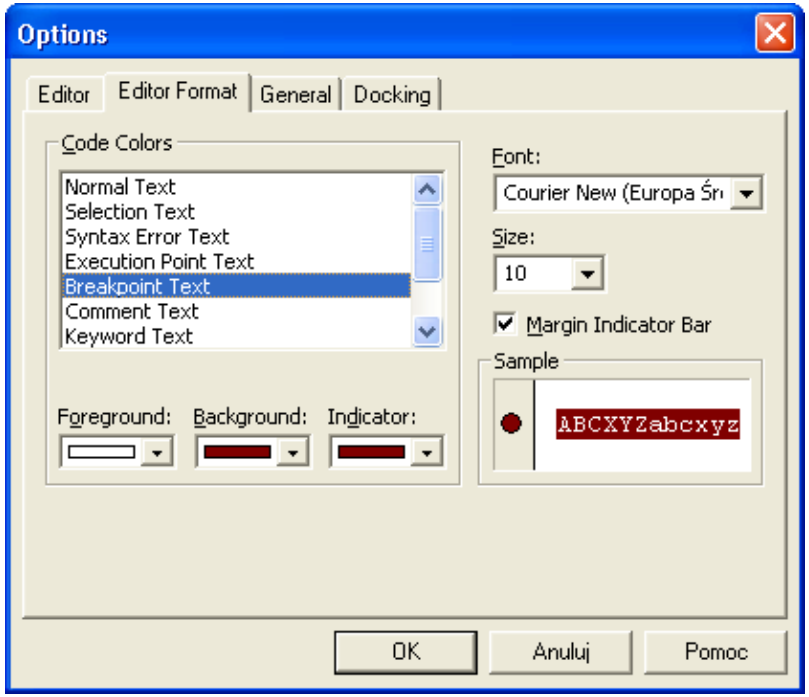
- Aby wywołać okno dialogowe umożliwiające ustawienie parametrów pracy Edytora Visual Basic należy wybrać z menu polecenie *Tools -> Options.*
- Powyższe okno posiada cztery zakładki:
  - **Editor** (edytor)
  - **Editor Format** (format edytora)
  - **General** (ogólne)
  - **Docking** (dokowanie)

Dostosowywanie środowiska Edytora Visual Basic



- ***Default to Full Module View*** *(Domyślnie pełny widok modułu)*
  Ustawienie opcji *Domyślnie pełny widok modułu* decyduje o standardowym stanie nowych modułów (nie ma wpływu na już istniejące). Jeśli opcja ta jest włączona wszystkie procedury w oknie kodu są wyświetlane w jednym przewijanym oknie. Gdy opcja jest wyłączona, w danej chwili możesz oglądać tylko jedną procedurę.
- ***Procedure Separator*** *(Separator procedur)*
  Jeśli pole wyboru *Separator procedur* jest zaznaczone, na końcu każdej procedury w oknie kodu wyświetlany jest oddzielający pasek.
- ***Auto Syntax Check*** *(Automatyczne sprawdzanie składni)*
  Ustawienie opcji *Automatyczne sprawdzanie składni* decyduje o tym, czy Edytor VB wyświetla okna dialogowe po odkryciu błędu syntaktycznego podczas wprowadzania kodu.
- ***Require Variable Declaration*** *(Wymagane deklaracje zmiennych)*
  Jeśli pole wyboru *Wymagane deklaracje zmiennych* jest zaznaczone, Edytor VB wstawia na początku każdego dodawanego przez użytkownika modułu Visual Basic następującą instrukcję: *Option Explicit*
  Obecność tej instrukcji w module zmusza do jawnego definiowania każdej używanej zmiennej.
- ***Auto List Members*** *(Automatyczne wyświetlanie składowych)*
  Jeśli pole wyboru *Automatyczne wyświetlanie składowych* jest zaznaczone Edytor VB oferuje dodatkową pomoc podczas wprowadzania kodu, poprzez wyświetlanie listy elementów obiektu.
- ***Auto Quick Info*** *(Automatyczne szybkie informacje)*
  Jeśli pole wyboru *Automatyczne szybkie informacje* jest zaznaczone, podczas wpisywania funkcji Edytor VB wyświetla informacje o niej i o jej argumentach.
- ***Auto Data Tips*** *(Automatyczne wskazywanie danych)*
  Jeśli pole wyboru *Automatyczne wskazywanie danych* jest zaznaczone, w trybie śledzenia kodu Edytor VB wyświetla wartość zmiennej, nad którą umieszczony jest wskaźnik myszy.
- ***Auto Indent*** *(Automatyczne wcięcia)*
  Ustawienie opcji *Automatyczne wcięcia* decyduje o tym, czy Edytor VB automatycznie zrobi wcięcie każdej linii kodu o tę samą wielkość, co linii poprzedniej.
- ***Drag-and-Drop Text Editing*** *(Edycja tekstu metodą "Przeciągnij i upuść")*
  Jeśli pole wyboru *Edycja tekstu metodą "Przeciągnij i upuść"* jest zaznaczone, można kopiować i przenosić tekst stosując przeciąganie i upuszczanie.

Dostosowywanie środowiska Edytora Visual Basic



- ***Margin Indicator Bar*** *(Wskaźniki marginesowe)*

  Opcja ta decyduje o wyświetlaniu w modułach pionowego paska marginesu. Dzięki temu na marginesie pojawiają się użyteczne wskaźniki graficzne (podczas śledzenia wykonania kodu).

- ***Code Colors*** *(Kolory kodu programu)*

  Opcja *Kolory kodu programu* pozwala wybrać kolor tekstu, tła oraz kolor wyróżniający elementy kodu.

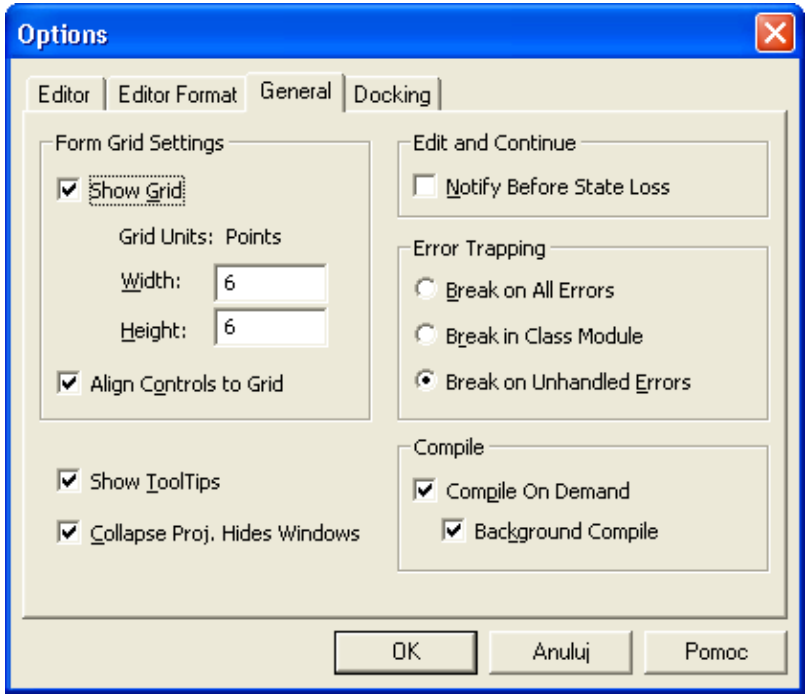
- ***Font*** *(Czcionka)*

  Opcja *Czcionka* pozwala wybrać czcionkę używaną w modułach VBA. Najlepiej jest stosować czcionkę o stałej szerokości, taką jak *Courier New*. W czcionce o stałej szerokości wszystkie znaki zajmują tyle samo miejsca w linii. Dzięki temu kod jest bardziej czytelny.

- ***Size*** *(Rozmiar)*

  Ustawienie opcji *Rozmiar* decyduje o rozmiarze czcionki. Przy wyborze należy uwzględnić między innymi rozdzielczość obrazu.

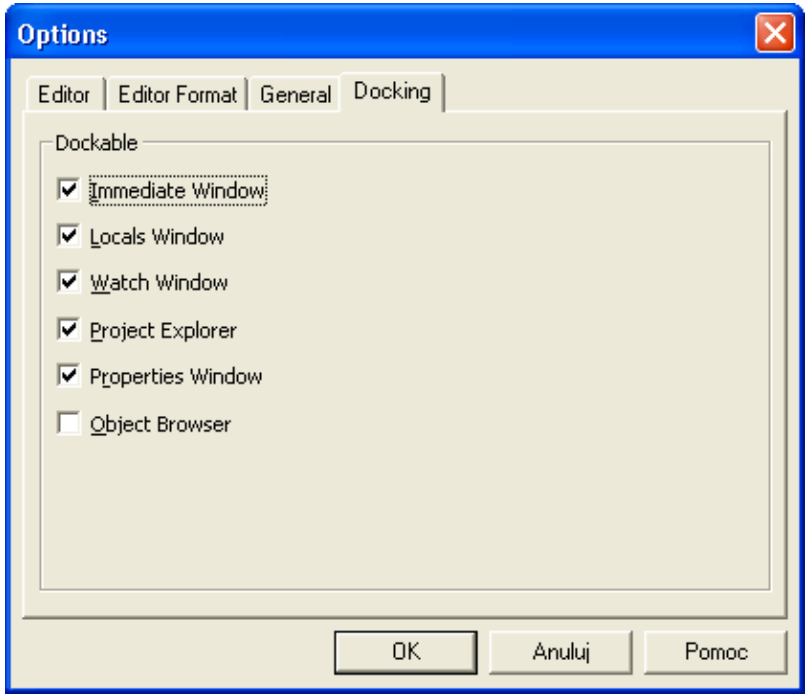
Dostosowywanie środowiska Edytora Visual Basic



- Ogólne ustawienia Edytora Visual Basic.

Generalnie można przyjąć, że standardowa konfiguracja jest optymalna.

Dostosowywanie środowiska Edytora Visual Basic



- Ustalenia sposobu wyświetlania elementów okna Edytora VB.
- Jeżeli element jest zadokowany, oznacza to, że będzie wyświetlany zawsze w tym samym miejscu, wzdłuż jednej z krawędzi okna Edytora VB. Ułatwia to jego lokalizowanie i identyfikację.

Makrodefinicje

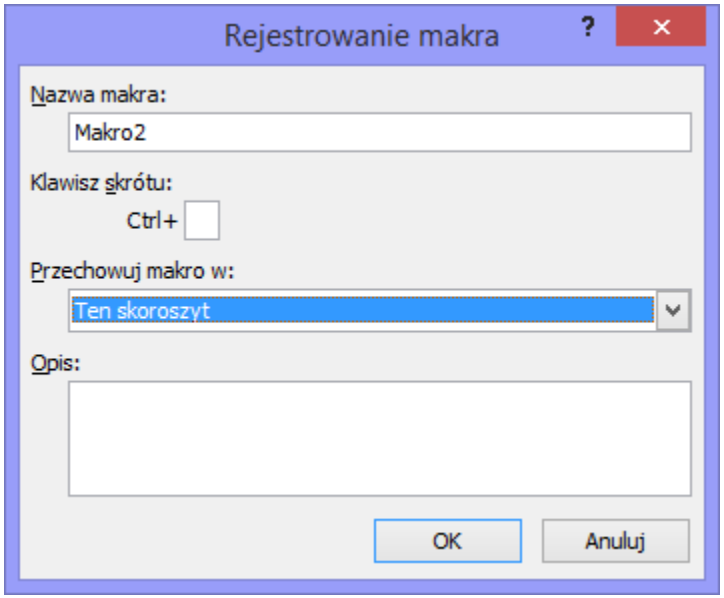
- ***Makrodefinicja*** jest serią poleceń wykonywanych automatycznie w momencie jej uruchomienia.
- Microsoft Excel posiada wbudowany ***Rejestrator Makrodefinicji***, który zamienia akcje wykonywane za pomocą myszki oraz klawiatury na kod Visual Basic. Rejestrowanie własnych makrodefinicji pozwala na dopasowanie Excela do własnych potrzeb i uczynienie pracy bardziej efektywną.
- Zapisaną makrodefinicję można powiązać z elementem menu lub przyciskiem.

Rejestrator makrodefinicji (makr)

- Może być stosowany jedynie do rejestracji prostych makr.
- Automatycznie generowany kod przez rejestrator zawiera dużo zbędnych poleceń.
- Rejestrator nie ma możliwości pełnego wykorzystania wszystkich funkcji Visual Basic (np. brak pętli).
- Uruchamianie:

  Deweloper -> Zarejestruj makro

Rejestrator makrodefinicji



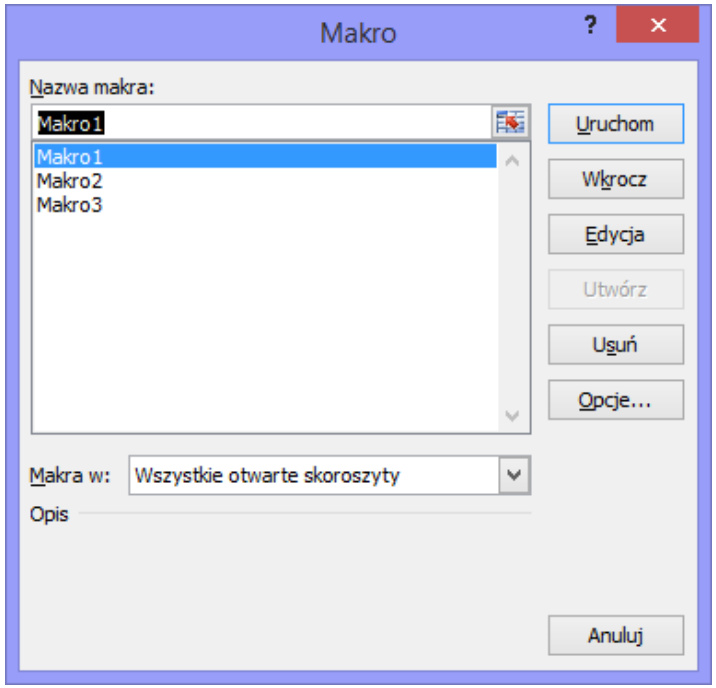
- **Opis**: Dowolny tekst opisujący makro. Zostanie on umieszczony jako komentarz na początku kodu makra.

- **Nazwa makra**: Nazwa może zawierać litery, cyfry i znaki podkreślenia, musi zaczynać się od litery i nie może zawierać spacji ani znaków przystankowych.
- **Klawisz skrótu**: Kombinacja przycisków, za pomocą której można będzie wywołać stworzone makro.
- **Przechowuj makro w**: Miejsce zapisania makra:
  - W aktywnym skoroszycie (*Ten skoroszyt*);
  - W nowym skoroszycie (*Nowy skoroszyt*);
  - W osobistym skoroszycie makr (*Skoroszyt makr osobistych*). Osobisty skoroszyt makr jest ukrytym skoroszytem, który zawsze pozostaje otwarty. Wszystkie zapisane w nim makrodefinicje są zawsze dostępne.

Rejestrator makrodefinicji

- Rozpoczęcie rejestracji makra: Przycisk OK.
- Wszystkie wykonywane od tego momentu operacja w arkuszu Excel będą zapisywane przez rejestrator w postaci kodu Visual Basic.
- Zakończenie rejestracji makra:
  Deweloper -> Zatrzymaj rejestrowanie

Zarządzanie makrami



- Przycisk **Usuń**: Usunięcie wybranego makra.
- Przycisk **Opcje**: Ustawianie opcji dotyczących makra:
  - **Klawisz skrótu**: Kombinacja przycisków za pomocą, której można wywołać stworzone makro.
  - **Opis**: Dowolny tekst opisujący makro umieszczany w jego komentarzu.

- **Uruchamianie**: *Deweloper -> Makra*
- **Nazwa makra**: Nazwa może zawierać litery, cyfry, znaki podkreślenia, musi zaczynać się od litery i nie może zawierać spacji ani znaków przestankowych.

  Jeśli operacje mają dotyczyć już istniejącego makra należy wybrać jego nazwę z listy.
- Przycisk **Uruchom**: Uruchomienie wybranego z listy makra.
- Przycisk **Anuluj**: Zamknij okno (bez żadnej reakcji).
- Przycisk **Wkrocz**: Wykonuj makro polecenie po poleceniu.
- Przycisk **Edycja**: Edytuj kod makra.
- Przycisk **Utwórz**: Tworzenie nowego makra o podanej nazwie.

Adresy bezwzględne i względne

Jedną z opcji, którą można zmieniać podczas rejestracji makrodefinicji jest rodzaj stosowanych podczas rejestracji adresów.
Do realizacji tego zadania służy drugi od lewej przycisk znajdujący się na pasku narzędziowym wyświetlanym podczas rejestracji makra.

- W przypadku **adresów bezwzględnych** Microsoft Excel zapisuje dokładną pozycję każdej wybranej komórki.
- W przypadku **adresów względnych** pozycja każdej nowo wybranej komórki jest liczona względem komórki poprzednio wybranej.

Ważne pytania

- **Jak podejrzeć kod wygenerowany przez rejestrator makr?**

  Wybrać z menu: *Deweloper -> Makra*
  - W oknie zarządcy makr wybrać z listy nazwę makra.
  - Nacisnąć przycisk *Edycja*.
  - Automatycznie zostanie uruchomiony Edytor Visual Basic i wyświetlony kod makra.
- **Czy można wprowadzać zmiany w kodzie makra?**

  Oczywiście TAK. Należy jednak znać składnie, funkcje oraz metody języka Visual Basic.
  Pisząc samodzielnie program (makro) można osiągnąć znacznie więcej niż poprzez rejestrację makrodefinicji.

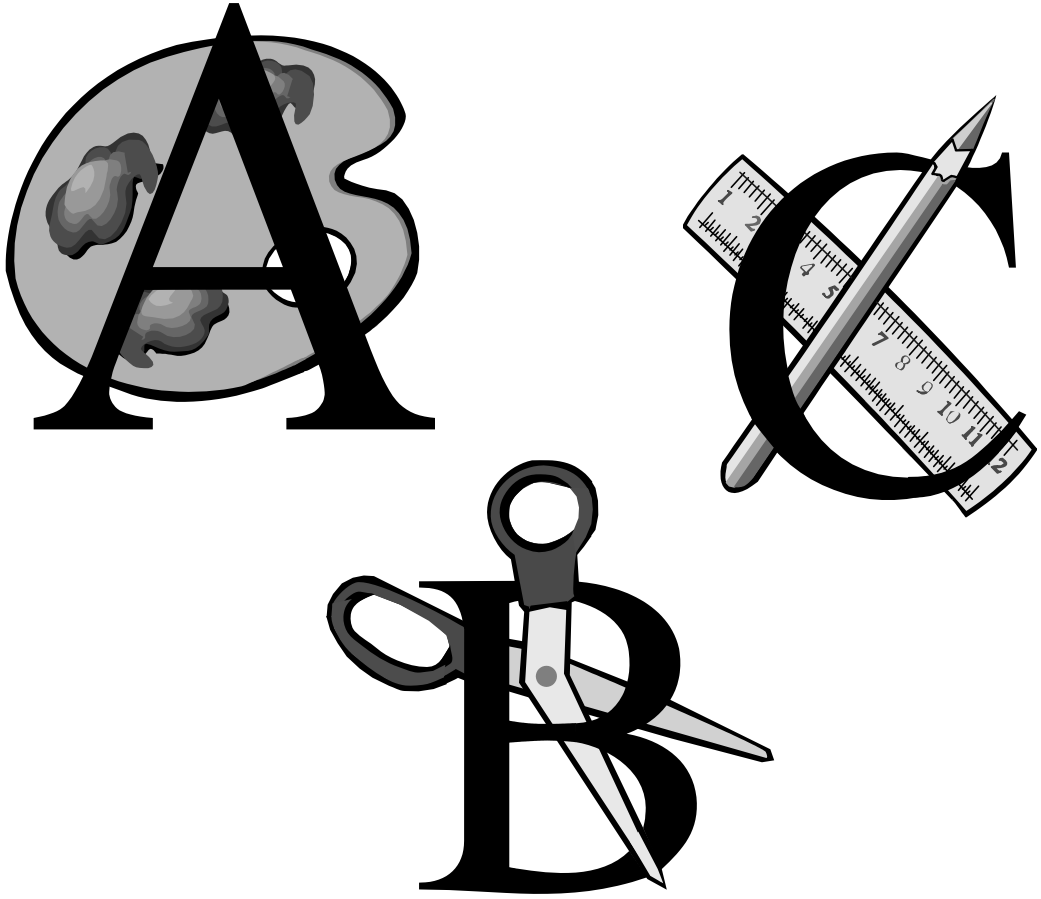
  Programy użytkownika mogą sprawdzać różne warunki i w zależności od nich wykonywać odpowiednie działania, mogą także pytać o dane użytkownika i informować go o postępie w realizacji zadań.
- **Czy kod utworzony przez rejestrator jest optymalny?**

  Niestety NIE. Rejestrator makrodefinicji tworzy kod, który odpowiada większości przeprowadzanych przez użytkownika czynności. Często jednak w module Visual Basic zostaje zapisanych więcej instrukcji niż potrzeba aby operacja przebiegła prawidłowo. Należy usunąć nadmiarowy kod, aby makrodefinicja robiła dokładnie to, czego zażyczył sobie użytkownika w optymalny sposób.

Programowanie w Visual Basic

część 1

- Zmienne
- Typy danych
- Stałe
- Tablice
- Komentarze



Zmienne

VBA wykonuje operacje na **danych**.

Dane są przechowywane w obiektach np. komórki arkusza lub specjalnie utworzonych zmiennych.

Zmienna to obiekt służący do reprezentacji różnych wartości np.:

- Wartości logicznych
- Liczb całkowitych
- Liczb zmiennoprzecinkowych
- Łańcuchów znakowych

Fizyczna reprezentacja zmiennej to określony obszar w pamięci komputera.

Nazwy zmiennych (1)

Nadawanie nazw zmiennym:

- Można używać wszystkich liter i cyfr ale pierwszym znakiem musi być litera.
- Wielkość liter nie ma znaczenia, ale w celu zwiększenia czytelności kodu lepiej jest różnicować rozmiar np.:

```
NumerWiersza = numerwiersza = NUMERWIERSZA
```

- Nie można używać kropek, spacji czy innych znaków specjalnych: #, $, %, &, !
- Nazwa może mieć maksymalnie 254 znaki.
- Nazwa nie może być taka jak słowa zastrzeżone np.:

```
Next, For, If
```

Nazwy zmiennych (2)

Konwencja nadawania nazw zmiennym tak aby po nazwie można było się zorientować jaki typ danych reprezentują.

Podanie typu danych w nazwie zmiennej polega na dodaniu standardowego prefiksu pisanego małymi literami – zgodnie z poniższym zestawieniem:

Boolean	**b**
Integer	**i**
Long	**l**
Single	**s**
Double	**d**
Currency	**c**
Data/Time	**dt**
String	**str**
Object	**obj**
Variant	**v**
Def. Uzytkownika	**u**

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.

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

Deklarowanie typu zmiennej

- Jeśli brak jest w programie deklaracji zmiennej używany jest typ domyślny **`Variant`**.
- Dane przechowywane jako **`Variant`** zmieniają swój typ w zależności od wykonywanych na nich operacjach.
- Instrukcja **`Option Explicit`** powoduje konieczność deklarowania wszystkich zmiennych przed ich użyciem.
- Najbardziej popularny sposób deklarowania zmiennej lokalnej to użycie instrukcji **`Dim`** 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
```

Deklarowanie 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
```

Deklarowanie zmiennych (2)

Zmienne widoczne we wszystkich modułach:

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

```
Public WartoscCalkowita As Integer

Sub Proc1()
   ...
End Sub

Sub Proc2()
   ...
End Sub
```

Deklarowanie zmiennych (3)

Zmienne lokalne:

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

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

- Zmienne lokalne zadeklarowane ze pomocą instrukcji **`Dim`** 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
```

Wartości zmiennych

Przypisanie wartości do zmiennej następuje poprzez użycie **operatora równości** np.:

```
x = 1
NumerWiersza = 100
Oprocentowanie = 0.05
BrakDanych = False
Uzytkownik = ”Zenek”
DataKonferencji = #10/31/2001#
x = x + 1
```

Stosowanie zmiennych (1)

- **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#
```

Stosowanie zmiennych (2)

- Obiektowe:
  - Reprezentują obiekty takie jak zakres czy arkusz.
  - Pozwalają uprościć kod programu.
  - Umożliwiają szybsze wykonywanie programu.
  - Deklaracja:

```
Dim 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”)
```

Deklarowanie stałych

- Wartości stałej nie można zmienić podczas wykonywania programu.

- Deklaracje stałej np.:

```
Const PI As Single = 3.14
Const Oprocentowanie As Single = 0.075
Public Const LiczbaPunktow As Integer = 5
```

Uwaga: Stałe podobnie jak zmienne mają swój zasięg.

Tablice (1)

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

Przykłady:

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

Tablice (2)

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
```

Komentarze

Aby wstawić komentarz w VBA należy:

- Poprzedzić tekst mający być komentarzem znakiem apostrofu: ‘

```
Dim Tab1(1 To 100) As Long ‘dane wejściowe
```

- Poprzedzić cały wiersz słowem: **rem**

```
rem dane wejściowe
Dim Tab1(1 To 100) As Long
```

Programowanie w Visual Basic

część 2

- Typy danych użytkownika
- Konwersje typów danych
- Przypisywanie wyrażeń
  - Operatory matematyczne
  - Operatory logiczne
  - Operatory porównania
- Sterowanie wykonaniem

Typy danych definiowane przez użytkownika

- Tworzone przez użytkownika typy są zawsze publiczne i ich definicje umieszcza się na początku modułu np.:

```
Type  Klient
      Nazwisko As String * 40
      Imie As String * 20
      DataUrodz As Date
End  Type
```

- Przykładowa deklaracja zmiennej tablicowej tego typu:

```
Dim ZbiorKlientow(0 To 999) as Klient
```

- Przykładowe odwołanie się do konkretnego składnika wybranego elementu tablicy:

```
ZbiorKlientow(0).Nazwisko = ”Kowalski”
ZbiorKlientow(0).Imie = ”Jan”
ZbiorKlientow(0).DataUrodz = #01/01/1950#
```

Konwersja typów danych

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

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

- Przykład:

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

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

Przypisywanie wyrażeń

- **Wyrażenie** to kombinacja słów kluczowych, operatorów, zmiennych i stałych, której wartością jest łańcuch, liczba lub obiekt. Wyrażenie może wykonywać obliczenia, przetwarzać tekst lub testować dane.
- **Operator przypisania** to w VBA znak równości: **=**
- Przykłady:

```
x = 1
x = x + 1
y = x*x + 2*x –10
KoniecDanych = True
KoniecDanych = Not KoniecDanych
Cells(2,3).Value = 102
```

Operatory matematyczne

`+`	dodawanie
`-`	odejmowanie
`*`	mnożenie
`/`	dzielenie
`^`	podnoszenie do potęgi
`&`	łączenie łańcuchów
`\`	dzielenie całkowite
`Mod`	dzielenie modulo

Operatory logiczne

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

```
If warunek Then instrukcja

If warunek Then
   instrukcje prawda
End If

If warunek Then
   instrukcje prawda
Else
   instrukcje falsz
End If

If warunek Then
   instrukcje prawda
Else
   If warunek2 Then
       instrukcje prawda2
   Else
       instrukcje falsz2
   End If
End If
```

If – Then (2)

```
If warunek Then
   instrukcje prawda
ElseIf warunek2 Then
   instrukcje prawda2
ElseIf waruenk3 Then
   instrukcje prawda3
Else
   instrukcje domyslna
End If
```

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

Select Case

Warunkowe wykonywane jednej lub wielu instrukcji.

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

```
Select Case Ilosc
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
```

For - Next

Pętla umożliwia wykonywanie tego samego bloku wielokrotnie.

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

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

```
For x = 1 To 100
   For y = 1 To 10
       suma = suma + tabela(x, y)
   Next y
Next x
```

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
Dim suma As Integer
For Each n In tablica
   suma = suma + n
Next n
```

Do While

Pętla wykonuje się dopóki warunek jest spełniony.

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

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

```
Do While X > 1
   If  X mod 2 = 0 Then
       X = X  / 2
   Else
       X = 3*X + 1
Loop
```

Do Until

Pętla wykonuje się do momentu aż warunek będzie spełniony.

```
Do [Until warunek]
   [instrukcje]
   [Exit Do]
   [instrukcje]
Loop
```

```
Do
   [instrukcje]
   [Exit Do]
   [instrukcje]
Loop [Until warunek]
```

```
Do Until X <= 1
   If  X mod 2 = 0 Then
       X = X  / 2
   Else
       X = 3*X + 1
Loop
```

Programowanie w Visual Basic

część 3

- Procedury (*Sub*)
  - Przekazywanie parametrów do procedur
  - Wywoływanie procedur
- Funkcje (*Function*)
  - Przekazywanie parametrów do funkcji
  - Wywoływanie funkcji
- Obsługa błędów
- Funkcje standardowe:
  - *MsgBox*
  - *InputBox*
  - inne funkcje wbudowane

Procedury - **Sub**

- Procedura to grupa instrukcji wykonujących określone zadanie.
- Procedury należy tworzyć i umieszczać w modułach.
- Jeden moduł może zawierać dowolną liczbę procedur.

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

Przekazywanie argumentów do procedur (1)

- **Argumenty** dostarczają procedurze dane wejściowe, które mogą być wykorzystywane podczas jej wykonywania.
- Do procedury można przekazać między innymi:
  - Zmienne
  - Stałe
  - Literały
  - Tablice
  - Obiekty
- Składnia argumentów:

```
[Optional] [ByVal | ByRef] [ParamArray]
nazwa[( )] [As typ] [= wartość_domyślna]
```

Przekazywanie argumentów (2)

- Procedura bez argumentów:
  Sub *proc1*()
- Procedura ze stałą liczbą argumentów:
  Sub *proc2*(*wiersz* **As Integer**, *kol* **As Long**)
- Procedura może mieć część argumentów wymaganych a część opcjonalnych:
  Sub *proc3*(*wiersz* **As Integer**, **Optional** *kol*)
- Procedura może mieć wszystkie argumenty opcjonalne:
  Sub *proc4*(**Optional** *wiersz*, **Optional** *kol=5*)
- Procedura o nieokreślonej liczbie parametrów:
  Sub *proc5*(**ParamArray** *tab*())

Wywoływanie procedur

- Polecenie *Run->Run Sub/User Form* w Edytorze VB.
- Polecenie *Deweloper->Makra* w Excelu.
- Skrót klawiszowy.
- Wybranie pozycji z menu lub paska narzędziowego.
- Z wnętrza innej procedury:

```
proc1 x, y
Call proc1(x, y)
Application.Run("proc1", x, y)
```

- Z wnętrza procedury znajdującej się w innym module:

```
Moduł1.proc1 x, y
Call Moduł1.proc1(x, y)
Application.Run("Moduł1.proc1", x, y)
```

- W momencie wystąpienia zdarzenia.

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
```

Przekazywanie argumentów do funkcji (1)

- **Argumenty** dostarczają funkcji dane wejściowe, które mogą być wykorzystywane podczas jej wykonywania.
- Do funkcji można przekazać między innymi:
  - Zmienne
  - Stałe
  - Literały
  - Tablice
  - Obiekty
- Składnia argumentów:

```
[Optional] [ByVal | ByRef] [ParamArray]
nazwa[( )] [As typ] [= wartość_domyślna]
```

Przekazywanie argumentów (2)

- Funkcja bez argumentów:
  Function *fun1*()
- Funkcja ze stałą liczbą argumentów:
  Function *fun2*(*wie* **As Integer**, *kol* **As Long**)
- Funkcja może mieć część argumentów wymaganych a część opcjonalnych:
  Function *fun3*(*wie* **As Integer**, **Optional** *kol*)
- Funkcja może mieć wszystkie argumenty opcjonalne:
  Function *fun4*(**Optional** *wie*, **Optional** *kol=5*)
- Funkcja o nieokreślonej liczbie parametrów:
  Function *fun5*(**ParamArray** *tab*())

Wywoływanie funkcji

- Z wnętrza procedury:

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

- W formułach arkusza:

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

Zwracanie błędów

- Aby funkcja użytkownika zwróciła wartość określonego błędu formuł Excela należy użyć funkcji **`CVErr(`**`blad`**`).`**
- Stałe odpowiadające wartościom błędów formuł Excela:
  - **`CVErr(`**`xlErrDiv0`**`)`**
  - **`CVErr(`**`xlErrNA`**`)`**
  - **`CVErr(`**`xlErrName`**`)`**
  - **`CVErr(`**`xlErrNull`**`)`**
  - **`CVErr(`**`xlErrNum`**`)`**
  - **`CVErr(`**`xlErrRef`**`)`**
  - **`CVErr(`**`xlErrValue`**`)`**

Obsługa błędów

- Podczas wykonywania programu mogą pojawić się błędy np. dzielenie przez zero, czy brak na dysku potrzebnego pliku.
- Wyrażenie **`On Error GoTo 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.
- Wyłączenie obsługi błędów następuje po **`On Error GoTo 0`**.
- **`Resume`** powraca do instrukcji, w której wystąpił błąd np. w sytuacji, gdy program próbuje coś odczytać z pustej stacji dyskietek wyświetlany jest stosowny komunikat. Po włożeniu dyskietki naturalny jest powrót do linii, w której wystąpił błąd.
- **`Resume Next`** przechodzi do wykonania następnej linii.
- **`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 (opis błędu), zwraca funkcja **`Error(Err)`**.

Funkcja **MsgBox**

- Składnia:

  `wartosc = MsgBox( strWiadomosc [, intRodzaj] [, strTytul] )`

  - **`strWiadomosc`** - obowiązkowy argument zawierający treść komunikatu, do 1024 znaków. Powinien być pisany w jednej linii. Można go rozdzielić na pojedyncze linie za pomocą znaków Chr(13) i Chr(10).
  - **`intRodzaj`** - argument opcjonalny, suma wartości z tabel sterujących.
  - **`strTytul`** - argument opcjonalny, łańcuch wyświetlany na pasku nazwy.
- Kolejność argumentów można zmienić, ale wówczas należy poprzedzić je nazwą, np:

  `MsgBox title:= tyt, prompt:= pytanie, buttons:= przyciski`
- 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**

- Składnia:

```
wartosc = MsgBox( strWiadomosc [, intRodzaj] [, strTytul] )
```

- **intRodzaj -** argument opcjonalny, suma wartości z tabel sterujących.
  - **VbOKOnly = 0** wyświetla przycisk OK.
  - **VbOKCancel = 1** wyświetla przyciski OK. i Anuluj
  - **VbAbortRetryIgnore = 2** wyświetla Przerwij, Ponów, Ignoruj
  - **VbYesNoCancel = 3** wyświetla Tak, Nie, Anuluj
  - **VbYesNo = 4** wyświetla Tak, Nie
  - **VbRetryCancel = 5** wyświetla Ponów, Anuluj
  - **VbCritical = 16** wyświetla ikonę krytyczną
  - **VbQuestion = 32** wyświetla ikonę pytania ostrzegawczego
  - **VbExclamation = 48** wyświetla ikonę wiadomości ostrzegawczej
  - **VbInformation = 64** wyświetla ikonę wiadomości
  - **VbDefaultButton1 = 0** przyciskiem domyślnym jest przycisk pierwszy
  - **VbDefaultButton2 = 256** przyciskiem domyślnym jest przycisk drugi
  - **VbDefaultButton3 = 512** przyciskiem domyślnym jest przycisk trzeci

Funkcja **InputBox**

- Składnia:

```
strZmienna = InputBox( strPytanie [, [strTytul] _
[,strDomyslne] [, intXpoz, intYpoz]] )
```

- **strPytanie** – jedyny argument obowiązkowy, zawierający treść komunikatu (max. 1024 znaków).
- **strTytul** – łańcuch znaków wyświetlany na pasku nazwy.
- **strDomyslne** – domyślna wartość odpowiedzi, którą można zaakceptować lub zmienić.
- **intXpoz, intYpoz** – dokładna pozycja okna na formularzu, w stosunku do lewej i górnej krawędzi ekranu (jeśli pominiemy te parametry, to okno jest wyświetlane na środku ekranu).
- **strZmienna** – wartość zwracana przez funkcję, będąca ciągiem znaków podanych przez użytkownika (jeżeli naciśnięto OK) lub ciągiem pustym (jeżeli wybrano Cancel).

Przykład: **MsgBox, InputBox**

```
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 ("Srednia 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
```

Uwaga: Funkcja ***InputBox*** przekazuje łańcuch znaków!

Przy wpisywaniu liczby należy ją zamienić na wartość liczbową za pomocą funkcji konwersji typów np. ***CInt().***

Funkcje wbudowane

- **`Len` –** zwraca ilość znaków łańcucha lub liczbę bajtów pamięci, jaka jest potrzebna do przechowywania argumentu.
- **Funkcje numeryczne:**

  `Sqr, Atn, Cos, Exp, Sin, Tan`
- **Funkcje łańcuchowe:**
  - **`Chr(int)` –** zwraca znak ASCII o kodzie `int`.
  - **`Ucase(str)` -** zwraca wartość typu `Variant (String)` zawierającą podany ciąg `str` zamieniony na wielkie litery.
  - **`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)` zawierającą kopię podanego ciągu znaków: bez spacji wiodących (`LTrim`), bez spacji końcowych (`RTrim`) lub bez spacji wiodących i końcowych (`Trim`).
  - **`Left(str, int)` - z**wraca 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, od podanej pozycji, z ciągu znaków.
  - **`Right(str, int)` -** zwraca wartość typu `Variant (String)` zawierającą podaną liczbę znaków `int` począwszy od prawej strony ciągu znaków `str`.
  - **`Str(liczba)` -** zwraca wartość typu `Variant (String)` zawierającą ciąg znaków reprezentujący liczbę `liczba` będącą wartością typu `Long`.

Programowanie w Visual Basic

część 4

- Obiekty i kolekcje
  - Hierarchia obiektów
  - Odwołania do obiektów
  - Właściwości obiektu
  - Metody obiektu
  - Przeglądarka obiektów

Hierarchia obiektów

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

Kolekcja

- Kolekcja to grupa obiektów należących do tej samej klasy np:
  - Obiekt **`Workbooks`** to kolekcja wszystkich aktualnie otwartych skoroszytów (**`Workbook`**).
  - Obiekt **`Worksheets`** to kolekcja wszystkich arkuszy (**`Worksheet`**) znajdujących się w danym obiekcie **`Workbook`**.
- Odwołanie do konkretnego obiektu w kolekcji:
  - Arkusz (**`Worksheet`**) o nazwie **`Arkusz1`**:

    ```
    Worksheets("Arkusz1")
    ```

  - Pierwszy arkusz w danym skoroszycie:

    ```
    Worksheets(1)
    ```

Odwołania do obiektów

- Pełne odwołanie do obiektu następuje poprzez podanie nazw kolejnych obiektów nadrzędnych w hierarchii, oddzielonych kropkami np:

```
Application.Workbooks("Zeszyt1").Worksheets("Arkusz1")
```

- Obiekt **`Application`** – można pominąć (jego obecność jest zakładana domyślnie).
- Obiekt **`Workbooks`** można pominąć jeśli skoroszyt **`Zeszyt1`** jest aktualnie aktywny.
- Ostatecznie odwołanie do arkusza **`Arkusz1`** w skoroszycie aktywnym **`Zeszyt1`** może wyglądać tak:

```
Worksheets("Arkusz1")
```

Właściwości obiektu

- Właściwości mogą zwracać odwołania do innych obiektów.
- Każdy obiekt posiada właściwości określające jego stan:
  - Obiekt **Range** posiada właściwość **Value** (wartość).

```
Application.Workbooks(1).Worksheets(1).Range("A1").Value
```

  - Obiekt **Range** posiada właściwość **Font** (czcionka).

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

  - Obiekt **Range** posiada właściwość **Interior** (wnętrze).

```
Range("A1").Interior.ColorIndex
```

  - Obiekt **Range** posiada właściwość **Borders** (obramowanie).

```
Range("A1").Borders.ColorIndex
```

- Wartości można odczytywać lub zapisywać (zmieniać):
  - Odczyt:

```
zmienna = Worksheets(1).Range("A1").Value
```

  - Zapis:

```
Worksheets(1).Range("A1").Value = zmienna
```

Właściwości obiektu **Application**

- Właściwości obiektu **Application** ułatwiające pracę z komórkami i zakresami:
  - **ActiveCell** – Aktywna komórka.
  - **ActiveSheet** – Aktywny arkusz.
  - **ActiveWindow** – Aktywne okno.
  - **ActiveWorkbook** – Aktywny skoroszyt
  - **Selection** – Zaznaczony obiekt.
  - **ThisWorkbook** – Skoroszyt zawierający wykonywaną procedurę.
- Przykłady:
  - **ActiveSheet.Name = "Mój_Arkusz"**
  - **ActiveCell.Value = 123**

Właściwości obiektu **Worksheet**

- Właściwość **Range**:
  - Wstawienie wartości do komórki **A1** w arkuszu **"Arkusz1"**:

    ```
    Worksheets("Arkusz1").Range("A1").Value = 123
    ```

  - Wstawienie wartości do komórki o nazwie **Blok**:

    ```
    Range("Blok").Value = 123
    ```

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

    ```
    Range("A1:B100").Value = 123
    ```

    ```
    Range("A1", "B100").Value = 123
    ```

  - Wstawienie wartości do komórek **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)

  - Wstawienie wartości do komórki **A3** w arkuszu **"Arkusz1"**:

    ```
    Worksheets("Arkusz1").Cells(3, 1).Value = 123
    Worksheets("Arkusz1").Cells(513).Value = 123
    ```

  - 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

  - Wstawienie wartości do komórki bezpośrednio pod komórką **A3** w arkuszu **"Arkusz1"**:

    ```
    Worksheets("Arkusz1").Cells(3, 1).Offset(1,0).Value = 123
    ```

  - Wstawienie wartości do komórki bezpośrednio nad komórką **A3** w arkuszu **"Arkusz1"**:

    ```
    Worksheets("Arkusz1").Cells(3, 1).Offset(-1,0).Value = 123
    ```

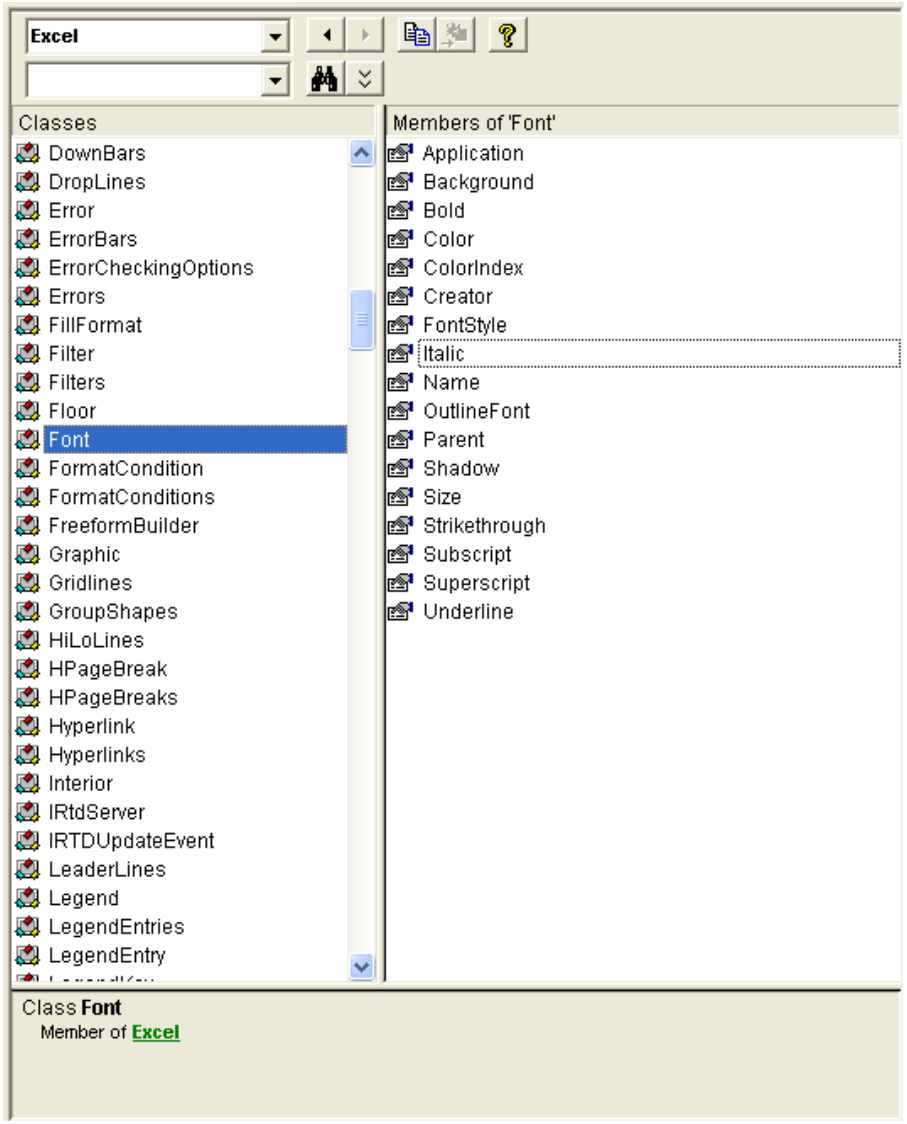
Metody obiektu

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

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

- Jeśli metoda ma parametry należy podać je po jej nazwie oddzielając kolejne przecinkami np.:
  - **`Range(”A1”).Copy Range(”B1”)`**
  - **`Workbooks(1).Protect ”qwer”, True, True`**
  - **`Workbooks(1).Protect , True, True`**
  - **`Workbooks(1).Protect , True, True`**
  - **`Workbooks(1).Protect Structure:=True, _`**
    `Windows:=True`

Przeglądarka obiektów



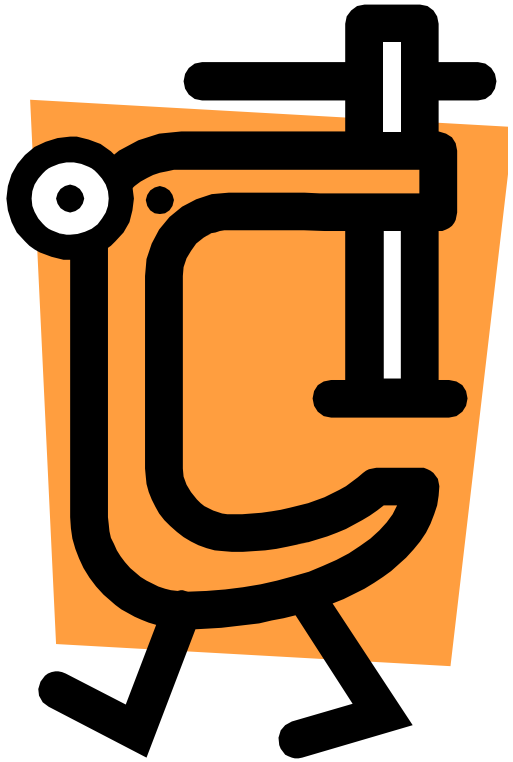
- **Przeglądarka obiektów** (*Object Browser*) umożliwia wyliczenie wszystkich właściwości i metod dowolnego obiektu.

- Uruchamianie przeglądarki w Edytorze VB:
  - Wybrać z menu pozycję *View -> Object Browser*.
  - Kliknąć ikonę *Object Browser* na standardowym pasku narzędzi.
  - Nacisnąć klawisz [*F2*].

Programowanie w Visual Basic

część 5

- Śledzenie kodu



Tryb śledzenia kodu

- Wstawienie punktu przerwania:

  menu: ***Debug -> Toggle Breakpoint***

  skrót: [***F9***]
- Usunięcie punktów przerwania:

  menu: ***Debug -> Clear All Breakpoint***

  skrót: [***Crtl***] + [***Shift***] + [***F9***]
- Wykonywanie programu krok po kroku (instrukcja po instrukcji):

  menu: ***Debug -> Step Into***

  skrót: [***F8***]

Śledzenie wartości wyrażeń

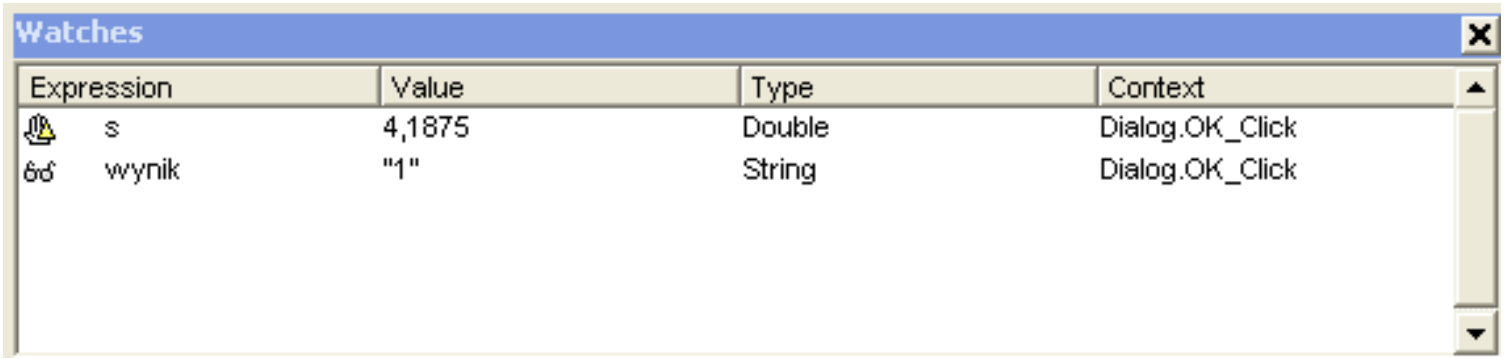
- ***Auto Data Tips*** (*Automatyczne wskazywanie danych*)

  Jeśli powyższa pozycja jest wybrana (opcje Edytora VB) to w trybie śledzenia kodu wyświetlana będzie wartość zmiennej, nad którą umieszczony jest aktualnie wskaźnik myszy.

- ***Watches*** (*Podgląd*)

  Po wybraniu z menu pozycji ***Debug -> Add Watch*** możemy podać wyrażenie, które będzie śledzone w czasie wykonywania programu (wartość, typ, kontekst).

  Pozycja ***Debug -> Edit Watch*** umożliwia edycję wybranego wyrażenia.



Programowanie w Visual Basic

część 6

- Przykłady:
  - Funkcja
  - Procedura 1
  - Procedura 2

Przykład: **Funkcja**

```
Function CenaBrutto(CenaNetto, StawkaVAT)
  CenaBrutto = CenaNetto * (1 + StawkaVAT / 100)
End Function


Function CenaBrutto(CenaNetto As Single, _
  StawkaVAT As Integer) As Single
  CenaBrutto = CenaNetto * (1 + StawkaVAT / 100)
End Function
```

Przykład: **Procedura 1** (1)

```
Sub kody()
   On Error GoTo blad
    Dim kod As String
    Dim test As String
    Dim wynik As String
    Dim x As Integer
    Dim ok As Boolean

    x = 1

    Do While 1
        ok = True
        kod = InputBox("Podaj kod:")

        If kod = "" Then Exit Sub
        If Len(kod) <> 3 Then ok = False

        test = Left(kod, 1)
        If IsNumeric(test) = True And ok = True Then
            If StrComp("0", test, vbTextCompare) = 0 Then
                kod = "O" & Right(kod, 2)
            Else
                ok = False
            End If
        End If
```

Przykład: **Procedura 1** (2)

```
        If IsNumeric(Right(kod, 2)) = False And ok = True Then
            ok = False
        End If

        If ok = True Then
            Cells(x, 1).Value = kod
            x = x + 1
        Else
            wynik = MsgBox("Nieprawidłowy kod!", vbCritical,
   "BŁĄD")
        End If
    Loop

    Exit Sub

blad:
    wynik = MsgBox("Nieznany błąd!", vbCritical, "BŁĄD")
End Sub
```

```
Sub kody()
On Error GoTo blad

    Dim kod As String
    Dim test As String
    Dim wynik As String
    Dim x As Integer
    Dim ok As Boolean

    x = 1

    Do While 1
        ok = True
        kod = InputBox("Podaj kod:")

        If kod = "" Then Exit Sub
        If Len(kod) <> 3 Then ok = False

        test = Left(kod, 1)
        If IsNumeric(test) = True And ok = True Then
            If StrComp("0", test, vbTextCompare) = 0 Then
                kod = "O" & Right(kod, 2)
            Else
                ok = False
            End If
        End If

        If IsNumeric(Right(kod, 2)) = False And ok = True Then
            ok = False
        End If

        If ok = True Then
            Cells(x, 1).Value = kod
            x = x + 1
        Else
            wynik = MsgBox("Nieprawidłowy kod!", vbCritical, "BŁĄD")
        End If
    Loop
    Exit Sub
blad:
    wynik = MsgBox("Nieznany błąd!", vbCritical, "BŁĄD")
End Sub
```

Przykład: **Procedura 2** (1)

```
Sub licz()
    Dim liczWyk As Integer
    Dim x As Integer
    Dim y As Integer
    Dim nowy As Boolean
    Dim wyk(10000, 2)
    Dim wartosc As String

    liczWyk = 0
    x = 1

    Do While 1
        wartosc = Trim(Worksheets("Arkusz1").Cells(x, 1).Value)

        If StrComp(wartosc, "", vbTextCompare) = 0 Then Exit Do

        nowy = True
        For y = 1 To liczWyk
            If StrComp(wartosc, wyk(y, 1), vbTextCompare) = 0 Then
                wyk(y, 2) = wyk(y, 2) + 1
                nowy = False
            End If
        Next
```

Przykład: **Procedura 2** (2)

```
    If nowy = True Then
            wyk(liczWyk + 1, 1) = wartosc
            wyk(liczWyk + 1, 2) = 1
            liczWyk = liczWyk + 1
        End If
        x = x + 1
    Loop

    With Worksheets("Arkusz1").Cells(1, "H")
        .Font.Bold = True
        .Value = liczWyk
    End With

    For y = 1 To liczWyk
        Worksheets("Arkusz1").Cells(y, "E").Value = wyk(y, 1)
        Worksheets("Arkusz1").Cells(y, "F").Value = wyk(y, 2)
    Next y

End Sub
```

```
Sub licz()
      Dim liczWyk As Integer
      Dim x As Integer
      Dim y As Integer
      Dim nowy As Boolean
      Dim wyk(10000, 2)
      Dim wartosc As String

      liczWyk = 0
      x = 1

      Do While 1
          wartosc = Trim(Worksheets("Arkusz1").Cells(x, 1).Value)

          If StrComp(wartosc, "", vbTextCompare) = 0 Then Exit Do

          nowy = True
          For y = 1 To liczWyk
              If StrComp(wartosc, wyk(y, 1), vbTextCompare) = 0 Then
                  wyk(y, 2) = wyk(y, 2) + 1
                  nowy = False
              End If
          Next

          If nowy = True Then
              wyk(liczWyk + 1, 1) = wartosc
              wyk(liczWyk + 1, 2) = 1
              liczWyk = liczWyk + 1
          End If
          x = x + 1
      Loop

      With Worksheets("Arkusz1").Cells(1, "H")
          .Font.Bold = True
          .Value = liczWyk
      End With

      For y = 1 To liczWyk
          Worksheets("Arkusz1").Cells(y, "E").Value = wyk(y, 1)
          Worksheets("Arkusz1").Cells(y, "F").Value = wyk(y, 2)
      Next y
End Sub
```

Programowanie w Visual Basic

część 7

- Formularze:
  - Wstawianie nowego formularza
  - Przybornik
  - Formanty
  - Właściwości obiektów
  - Wyświetlanie oraz zamykanie formularza
  - Obsługa zdarzeń
  - Właściwość i Metody obiektów
  - Przykład

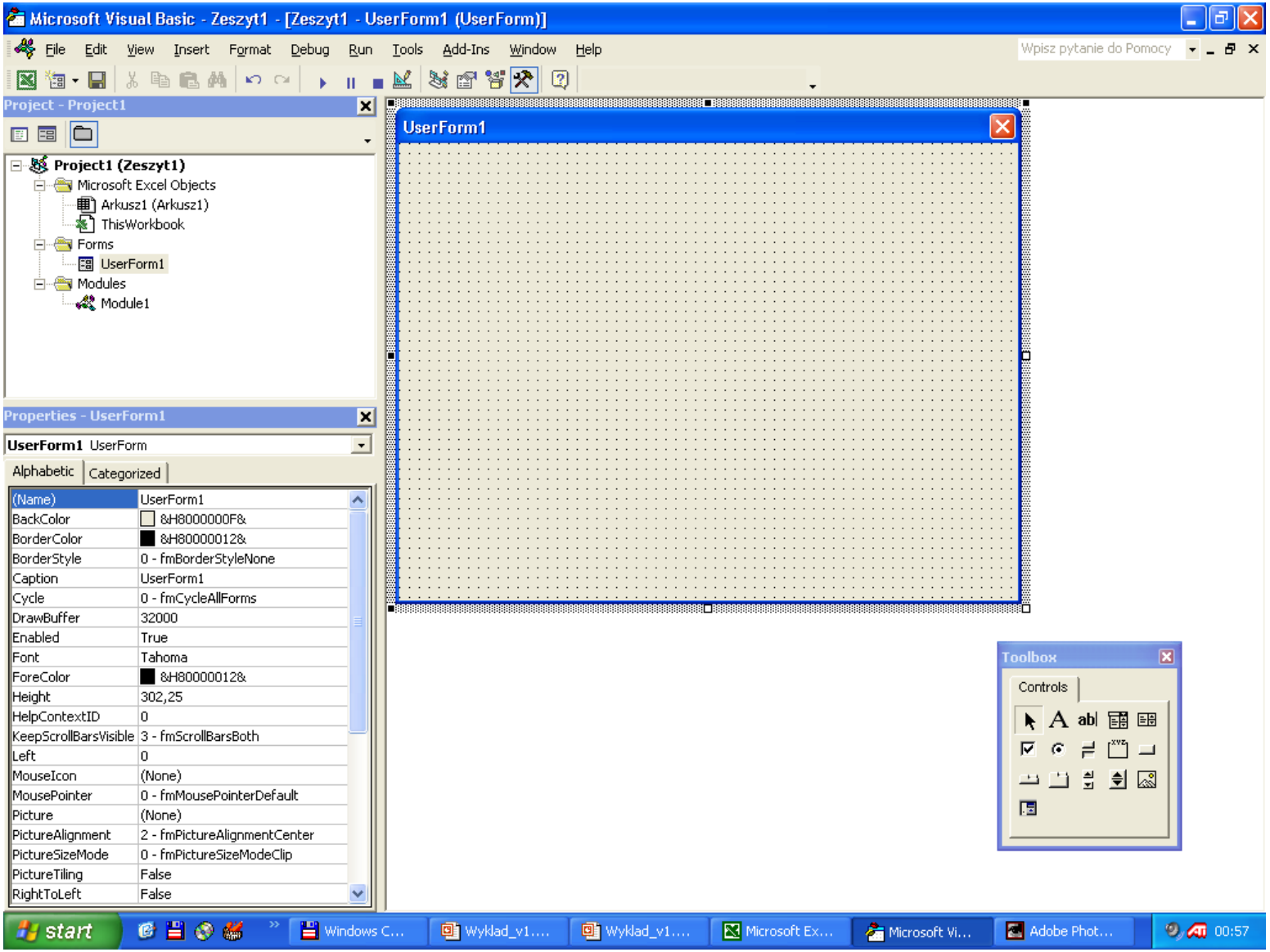
Wstawianie nowego formularza

- Uruchomić Edytor VB.
- Wybrać w Eksploratorze projektu właściwy skoroszyt (projekt).
- Nacisnąć na nim prawy przycisk myszy a następnie wybrać z menu pozycję:

  Insert -> UserForm

- Aby zmienić właściwości formularza należy nacisnąć na nim lewy przycisk myszy i w oknie Właściwości projektu dokonać potrzebnych zmian.

Wstawianie nowego formularza



Przybornik

- Aby wywołać przybornik należy z menu wybrać pozycję:
  View -> Toolbox.
- Przybornik umożliwia umieszczanie elementów kontrolnych w formularzu (potrzebny obiekt przeciąga się z przybornika na formularz).
- Aby zmienić własności dodanego do formularza elementu kontrolnego należy kliknąć na nim lewym przyciskiem myszy i w oknie Właściwości projektu dokonać potrzebnych zmian.
- Aby precyzyjnie wyrównać i rozmieścić elementy na formularzu należy zaznaczyć wybrane elementy a następnie z menu ***Format*** wybrać odpowiednie opcje.

Elementy kontrolne - Formanty (1)

- **`Label (etykieta)`** – Dodanie dowolnego tekstu na formularzu.

- **`Text box (pole tekstowe)`** – Umożliwia użytkownikowi wpisywanie danych np. tekstu, liczb, odniesienia do komórek lub formuł.

- **`Combo box (pole kombi)`** – Pole tekstowe, które po kliknięciu strzałki rozwija się w pole listy, o długości określonej właściwością **`ListRows`**. W polu kombi można wybrać tylko jeden element. Typ pola kombi określa właściwość **`Style`**: **`0`** – wybór elementu z listy lub wpisanie nowego w polu tekstowym; **`2`** – tylko wybór z listy. Wpisanie nowej wartości w polu tekstowym wymaga ustawienia właściwości **`MatchRequired`** na **`False`**.

- **`List box (pole listy)`** – Służy do tworzenia listy wartości. Elementy listy można wpisać w arkuszu i podać bezwzględny adres zawierającego je zakresu komórek we właściwości **`RowSource`**, lub wprowadzić bezpośrednio w procedurze VB przy użyciu metody **`AddItem`** albo przypisując właściwości **`List`** tablicę wartości: **`ListBox.List() = array(”..”).`**

Elementy kontrolne - Formanty (2)

- **Check box (pole wyboru)** – Służy do włączania lub wyłączania danej opcji. Umożliwia wybór jednej lub kilku opcji jednocześnie. Dla opcji wybranej właściwość **Value = True.**
- **Option button (przycisk opcji)** – Umożliwia wybranie jednej spośród kilku wykluczających się opcji. Dla opcji wybranej właściwość **Value = True.**
- **Toggle button (przełącznik)** – Działa podobnie jak przycisk opcji. Wciśnięty przycisk przełącznika ma właściwość **Value = True**
- **Frame (ramka)** – Umożliwia uporządkowanie obiektów formularza, a w przypadku przycisków opcji powoduje, że umieszczone w niej przyciski wzajemnie się wykluczają.
- **CommandButton (przycisk polecenia)** – Prawie każde okno dialogowe zawiera przynajmniej jeden obiekt **CommandButton**, po naciśnięciu którego wykonywane są określone przez użytkownika instrukcje. Najbardziej znane z okien dialogowych są przyciski opisane przeważnie etykietami **OK** i **Cancel** umożliwiające akceptację lub rezygnację z danej operacji.

Elementy kontrolne - Formanty (3)

- **`TabStrip (pasek tabulatorów)`** – Pozwala dodać do formularza nowe strony zawierające te same formanty. Może być wyświetlany w dowolnym położeniu, ustawionym właściwością **`TabOrientation`**. Nowe strony można dodawać po kliknięciu w obrębie paska tabulatorów prawym klawiszem myszy i wybraniu opcji *New page*, a następnie, poprzez użycie opcji, *Rename…* można zmienić ich nazwę i określić klawisz skrótu.
- **`MultiPage (formularz złożony)`** – Pozwala projektować formularze zawierające kilka stron. Nowe strony można dodać z menu skrótów lub metodą **`Add`** z procedury VB.
- **`Scroll bar (pasek przewijania)`** – Służy do wprowadzania wartości liczbowych z określonego przedziału, którego zakres określają właściwości **`Max`** oraz **`Min`**, aktualną wartość właściwość **`Value`**, a krok zmian właściwość **`LargeChange`**. O ile zmieni się wartość po kliknięciu jednej ze strzałek określa właściwość **`SmallChange`**.

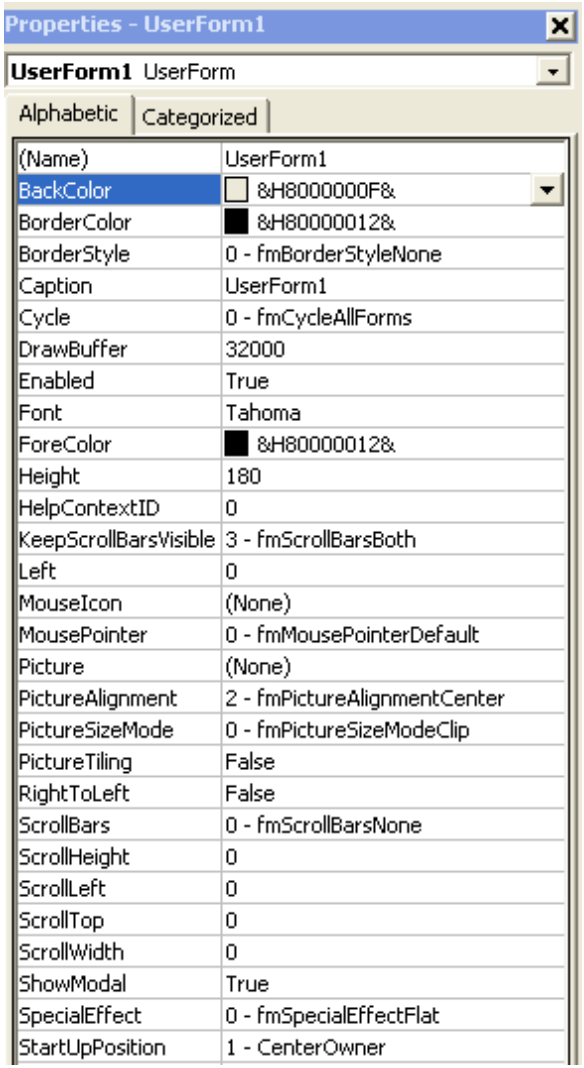
Elementy kontrolne - Formanty (4)

- **`Spin button (pokrętło)`** – Działa analogicznie jak pasek przewijania. Można je połączyć z polem tekstowym i wówczas zamiast wpisywać wartość w tym polu, można ustalać ją pokrętłem.

- **`Image (obrazek)`** – Można wstawić dowolny plik z rozszerzeniem **.BMP, *.CUR, *.GIF, *.ICO, *.JPG, *.WMF*. Rozmiar obrazka można ustalić za pomocą właściwości **`PictureSizeMode`**.

- **`RefEdit`** – umożliwia zaznaczenie zakresu komórek w arkuszu i przekazanie ich adresu do procedury VBA. Występuje tylko w Excelu.

Właściwości obiektów (1)



- Aby zmienić właściwości obiektu (formularza lub formantu) należy:
  - Nacisnąć na nim prawy przycisk myszy i wybrać opcję: **`Properties`**.
  - Nacisnąć na nim lewy przycisk myszy - w oknie Właściwości obiektu ukażą się dostępne opcje.
  - W oknie Właściwości obiektu wybrać z rozwijanej listy odpowiednią pozycję (nazwę obiektu).

Właściwości obiektów (2)

- Okno Właściwości obiektu:
  - Zakładka *Alphabetic*: Wyświetlanie listy właściwości obiektu w porządku alfabetycznym.
  - Zakładka *Categorized*: Wyświetlanie właściwości obiektu według kategorii.
- Aby zmienić wybraną właściwość należy zaznaczyć ją na liście i wprowadzić nową wartość.
  - Własności mogące przyjmować skończoną liczbę wartości, proponują je w formie listy rozwijanej.
  - Inne, w celu ułatwienia wprowadzenia nowej wartości, oferują dodatkowe, kontrolne okno dialogowe.
- Aby uzyskać dodatkowe informacje o konkretnej właściwości należy zaznaczyć ją i nacisnąć przycisk [*F1*] w celu wywołania systemu pomocy.

Właściwości formularza (1)

`Name` - Przypisanie nazwy formularzowi.

`BackColor` - Kolor tła formularza.

`BorderStyle` - Ramka wokół formularza.

`Caption` - Tekst pojawiający się w oknie na pasku tytułu.

`Enabled` - Czy formularz jest dostępny.

`Font` - Parametry czcionki.

`ForeColor` - Ustalenie koloru tekstów na formularzu.

`Height` - Wysokość formularza.

`Left` - Odległość lewego brzegu ekranu od lewego brzegu formularza.

Właściwości formularza (2)

`MousePointer` - Kształt kursora myszy nad formularzem.

`Picture` - Rysunek znajdujący się w tle formularza.

`StartUpPosition` - Położenie formularza na ekranie o po uruchomieniu programu.

`Top` - Odległość od górnego brzegu ekranu do górnego brzegu formularza.

`Width` - Szerokość formularza.

`Zoom` - Powiększenie zawartości formularza.

Właściwości etykiety (1)

`AutoSize` - Jeżeli **`True`** rozmiar etykiety dostosowywany zostanie automatycznie do długości napisu.

`BackColor` - Kolor tła etykiety.

`BackStyle` - Czy tło etykiety jest przezroczyste.

`BorderStyle` - Czy etykieta ma posiadać ramkę.

`Caption` - Tekst pojawiający się na etykiecie.

`Enabled` - Czy etykieta jest dostępna.

`Font` - Parametry czcionki.

`ForeColor` - Kolor tekstu etykiety.

`Height` - Wysokość przycisku.

Właściwości etykiety (2)

`Left` - Odległość lewego brzegu etykiety od lewego brzegu formularza.

`MousePointer` - Kształt kursora myszy nad etykietą.

`TabIndex` - Określa, jako który element etykieta będzie miała fokus.

`TabStop` - Czy może przyjąć fokus.

`TextAlign` - Wyrównanie zawartości pola tekstowego.

`ToolTipText` - Tekst objaśniający.

`Top` - Odległość od górnego brzegu etykiety do góry formularza.

`Visible` - Czy etykieta jest widoczna.

`Width` - Szerokość etykiety.

Właściwości pola tekstowego (1)

`AutoSize` - Jeżeli **`True`** rozmiar pola dostosowywany zostanie automatycznie do długości napisu.
`BackColor` - Kolor tła pola.
`BorderStyle` - Czy pole ma posiadać ramkę.
`Enabled` - Czy pole jest dostępne.
`EnterKeyBehavior` - Jeżeli **`True`** po naciśnięciu klawisza [*Enter*] nastąpi przejście do nowej linii.
`Font` - Parametry czcionki.
`ForeColor` - Kolor tekstu pola.
`Height` - Wysokość pola.
`Left` - Odległość lewego brzegu pola od lewego brzegu formularza.
`Locked` - Czy użytkownik może edytować tekst pojawiający się w polu.

Właściwości pola tekstowego (2)

MaxLenght - Ilość znaków, jaką można wpisać do pola.
MousePointer - Kształt kursora myszy nad polem.
MultiLine - Pozwala na przechowywanie tekstu dłuższego niż jedna linijka.
PasswordChar - Pole do wprowadzania hasła (podajemy znak zastępczy).
ScrollBars - Czy dodać paski przewijania.
TabIndex - Określa, jako który element pole będzie miało fokus.
Text - Zawartość początkowa pola.
ToolTipText - Tekst objaśniający.
Top - Odległość od górnego brzegu pola do góry formularza.
Visible - Czy pole jest widoczne.
Width - Szerokość pola.
WordWrap - Automatyczne zawijanie wierszy.

Właściwości pola listy (1)

`ColumnCount` - Określa ilość kolumn.

`ColumnWidths` - Szerokość pojedynczej kolumny.

`ColumnHeads` - Jeżeli **`True`** podawane są tytuły kolumn.

`BoundColumn` - Numer kolumny z której będzie odczytywana wartość.

`Font` - Parametry czcionki.

`ForeColor` - Kolor tekstu w polu listy.

`Height` - Wysokość elementu.

`Locked` - Blokada pola listy.

Właściwości pola listy (2)

`ListStyle` - Styl listy:

- **`fmListStylePlain`** - tradycyjna
- **`fmListStyleOption`** - z polami wyboru obok elementów

`MultiSelect` - Sposób wyboru elementu z listy:

- **`fmMultiSelectSingle`** - można wybrać tylko jeden element z listy
- **`fmMultiSelectMulti`** - można wybrać kilka elementów myszą lub spacją
- **`fmMultiSelectExtended`** - można wybrać kilka elementów za pomocą klawiszy [*Shift*], [*Ctrl*] i myszy

Właściwości przycisku opcji (1)

`Alignment` - Opis po lewej lub po prawej stronie przycisku.

`BackColor` - Kolor tła przycisku.

`Caption` - Tekst pojawiający się przy przycisku.

`Enabled` - Czy przycisk jest dostępny.

`Font` - Parametry czcionki.

`Height` - Wysokość przycisku.

`Left` - Odległość lewego brzegu przycisku od lewego brzegu formularza.

Właściwości przycisku opcji (2)

`MousePointer` - Kształt kursora myszy nad przyciskiem.

`Picture` - Plik graficzny wyświetlany w tle przycisku.

`PicturePosition` - Sposób umiejscowienia grafiki przy przycisku.

`TabIndex` - Określa, jako który element przycisk będzie miał fokus.

`TabStop` - Czy może przyjąć fokus.

`Top` - Odległość od górnego brzegu przycisku do góry formularza.

`Visible` - Czy jest widoczny.

`Width` - Szerokość przycisku.

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
```

Obsługa zdarzeń (1)

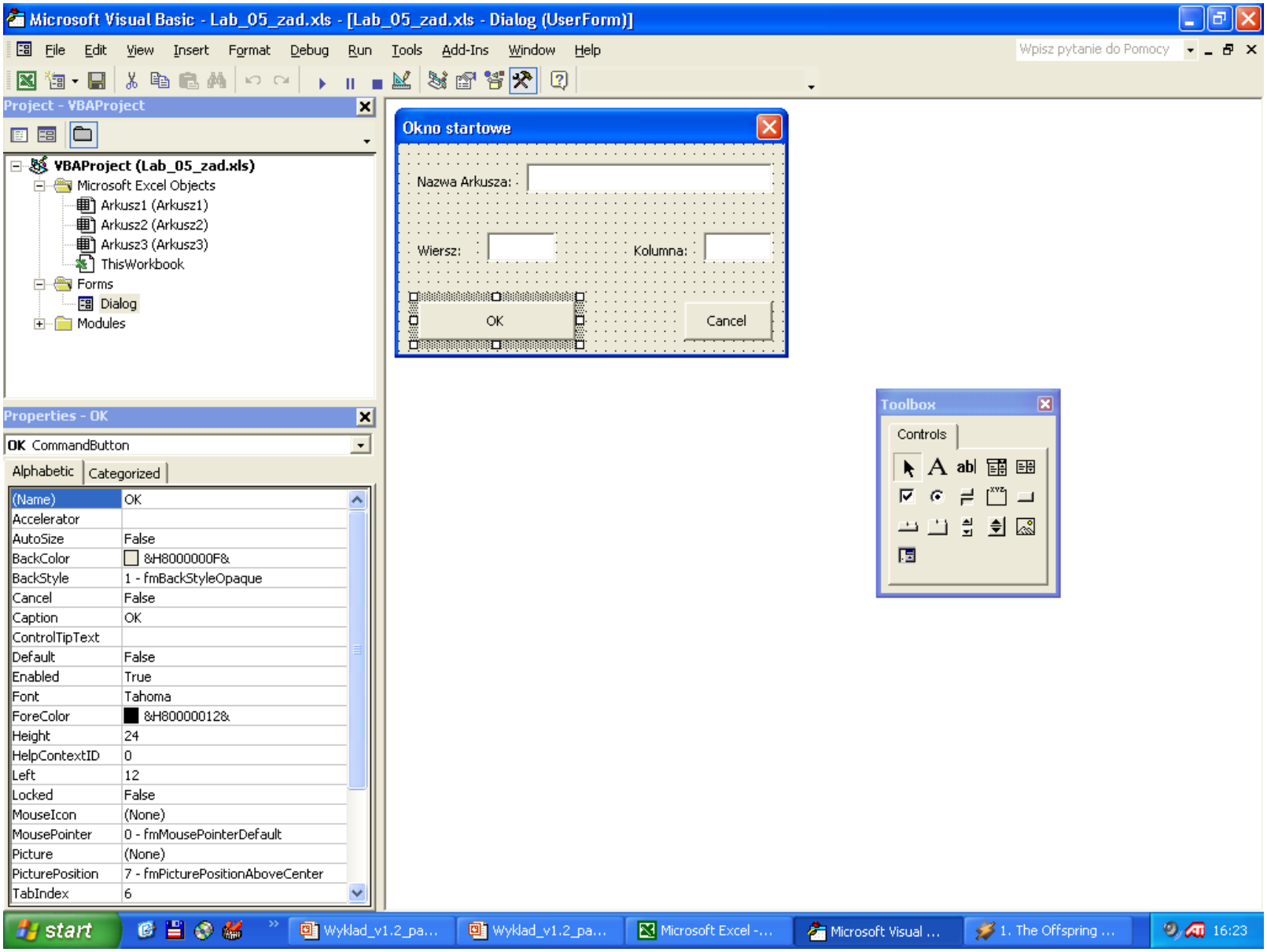
- Działanie użytkownika w oknie dialogowym polega na klikaniu przycisków, wybieraniu elementów z list, wpisywaniu tekstu do pól tekstowych itp.
- Wszystkie te czynności powodują wywołanie **zdarzeń**.
- Kliknięciu przycisku powoduje wystąpienie zdarzenia **`Click`** dla tego formantu.
- Procedura wykonywana w momencie wystąpienia określonego zdarzenia to **procedura obsługi zdarzenia**.
- Procedury obsługi zdarzeń muszą się znajdować w oknie kodu formularza użytkownika.

Obsługa zdarzeń (2)

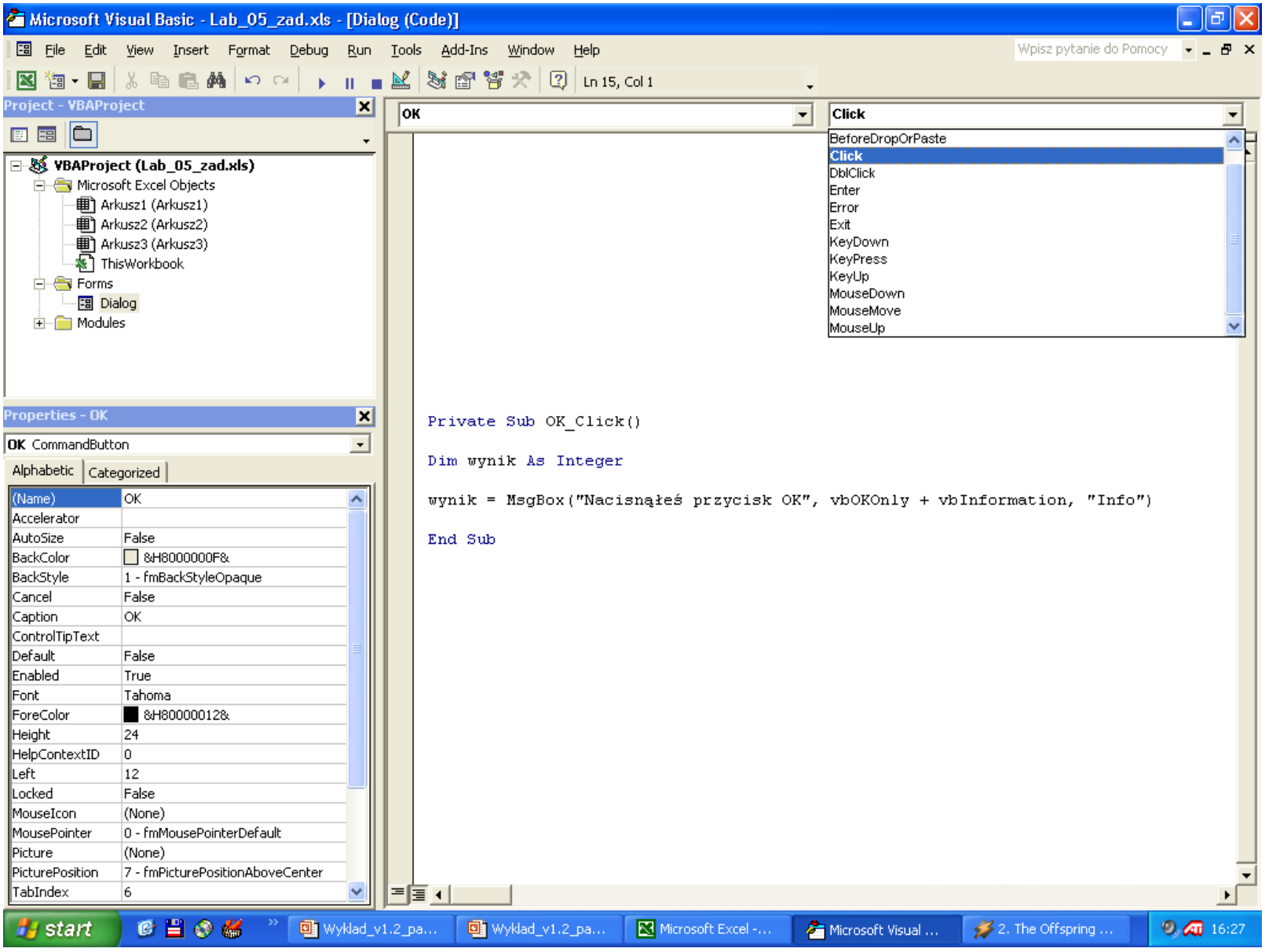
Aby wpisać kod obsługi wybranego zdarzenia należy:

- Kliknąć dwukrotnie na formancie, z którym związane zdarzenie chcemy obsłużyć.
- Rozwinąć listę znajdującą się w prawym górnym rogu modułu. Na tej liście znajdują się wszystkie zdarzenia związane z wybranym formantem.
- Wybrać dowolną pozycję z listy.
- Zostanie automatycznie utworzona pusta procedura obsługi tego zdarzenia.
- Wprowadzić potrzebny kod w języku Visual Basic.

Obsługa zdarzeń (3)



Obsługa zdarzeń (4)



Zdarzenia dotyczące formularza

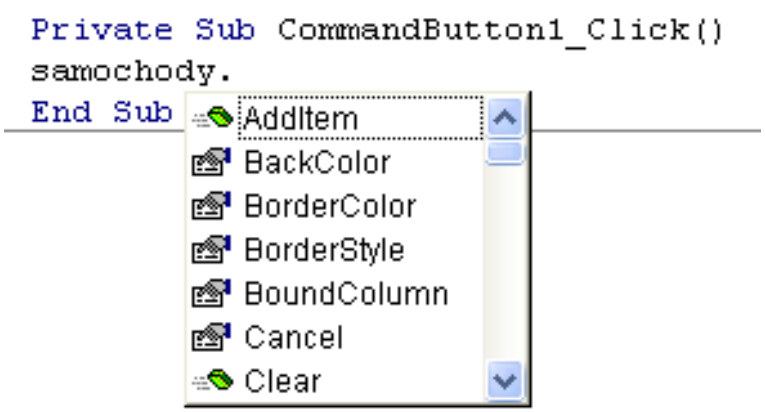
- **`Initialize`** - Wywoływane w momencie uruchomienia formularza użytkownika.
- **`Activate`** - Wywoływane w momencie uaktywnienia formularza użytkownika.
- **`Deactivate`** - Wywoływane w momencie gdy formularza użytkownika przestaje być aktywny.
- **`QueryClose`** - Wywoływane przed usunięciem formularza użytkownika z pamięci.
- **`Terminate`** - Wywoływane po usunięciu formularza użytkownika z pamięci.

Zdarzenia dotyczące formantów

- **`AfterUpdate`** - Po zmianie formantu (przy użyciu interfejsu użytkownika).
- **`BeforUpdate`** - Przed zmianą formantu.
- **`Change`** - Podczas zmiany właściwości **`Value`**.
- **`Click`** - Kilknięcie myszą na obiekcie.
- **`DblClick`** - Podwójne kilknięcie myszą na obiekcie.
- **`KeyDown`** - Naciśnięcie klawisza przez użytkownika (obiekt musi mieć fokus).
- **`KeyUp`** - Zwolnienie klawisza przez użytkownika.
- **`SpinDown`** - Kliknięcie w pokrętle przycisku strzałki w dół.
- **`SpinUp`** - Kliknięcie w pokrętle przycisku strzałki w górę.

Właściwości i Metody

- Właściwości określają stan obiektu.
- Metody to działania, które można wykonać z udziałem wybranego obiektu.



Metody pola listy

- **`AddItem`** - Dodaje element do listy.

  `Variant = object.AddItem( [ item [,  varIndex]])`

  - **`object`** - Nazwa obiektu typu **`ListBox`** lub **`ComboBox`** (wymagany).
  - **`item`** - Podaje nazwę składnika listy (opcjonalny).
  - **`varIndex`** - Podaje pozycję w której zostanie umieszczony nowy składnik (opcjonalny).
- **`Clear`** - Usuwa wszystkie elementy z listy.
- **`Move`** - Zmiana położenia wybranego obiektu na formularzu.
- **`RemoveItem`** - Usuwa pojedynczy element z listy.

Metody formularza

- **`Load`** - Załadowanie formularza do pamięci.
- **`Show`** - Pokazanie, wyświetlenie formularza na ekranie.
- **`UnLoad`** - Usunięcie formularza z pamięci.
- **`Hide`** - Ukrycie formularza.
- **`PrintForm`** - Wydruk formularza.
- **`Repaint`** - Odświeżenie zawartości formularza.

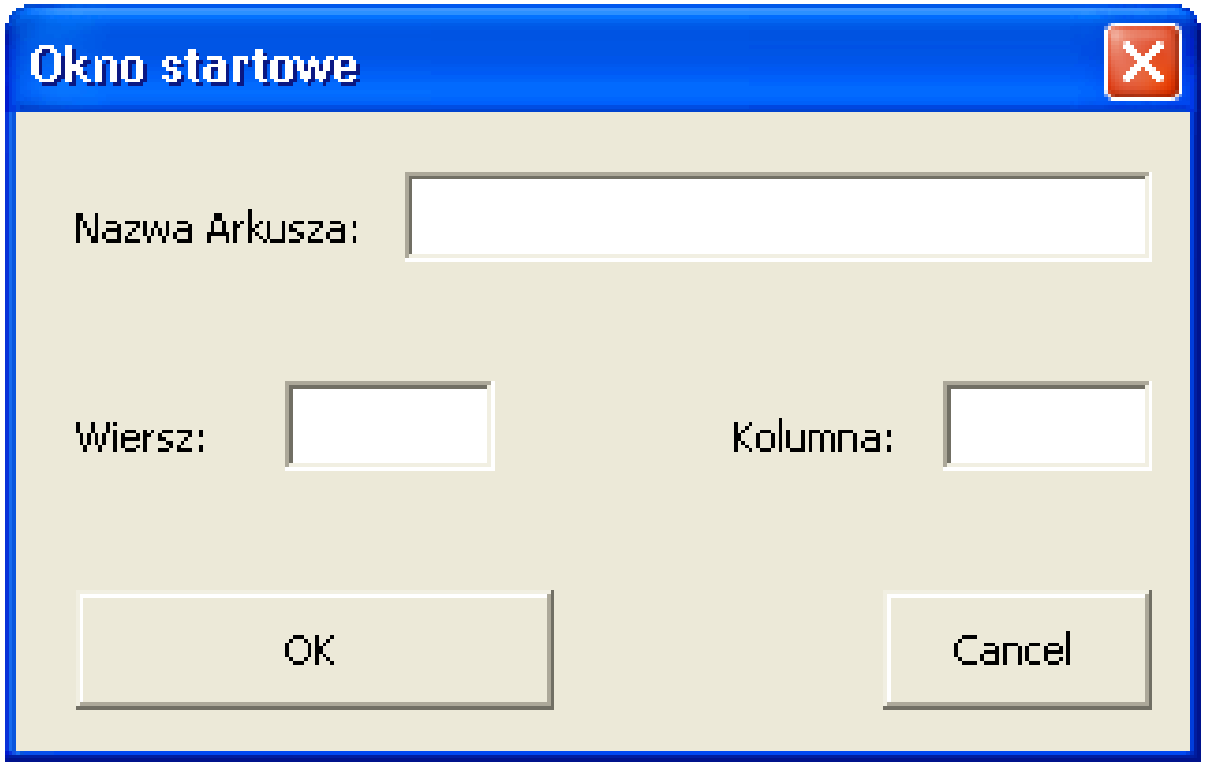
Właściwości, metody, zdarzenia

- Aby uzyskać informację o:
  - **właściwościach**
  - **metodach**
  - **zdarzeniach**

  związanych z danym formantem należy wyszukać w systemie pomocy informacji dotyczących wybranego formantu (pozycja: ***nazwa_formantu control***).

  Następnie wybrać, znajdujący się w górnej części okna, odnośnik do odpowiednio:
  - **Właściwości (*Properties*)**
  - **Metod (*Methods*)**
  - **Zdarzeń (*Events*)**

Przykład



Przykład

```
Sub form()
  Load Dialog
  Dialog.Show
End Sub
```

```
Private Sub OK_Click()
  On Error GoTo blad
  Dim s As Double
  Dim wynik As Integer
  Dialog.Hide
  s = wartosc(Arkusz.Value, Wiersz.Value, _
      Kolumna.Value)
  If s = 0 Then
      wynik = MsgBox("BLAD!", vbCritical + vbOKOnly, _
            "Wynik")
  Else
      wynik = MsgBox(„Wartość: " & Format(s, "0.00"), _
            vbOKOnly + vbInformation, "Wynik")
  End If
  Dialog.Show
  Exit Sub
blad:
  Unload Dialog
End Sub
```

```
Function wartosc(ark As String, w As Integer, k As _
   Integer) As Single
       On Error GoTo blad
       wartosc = Worksheets(ark).Cells(w, k).Value
       Exit Function
blad:
       wartosc = 0
End Function
```