Programowanie arkuszy kalkulacyjnych (VBA)

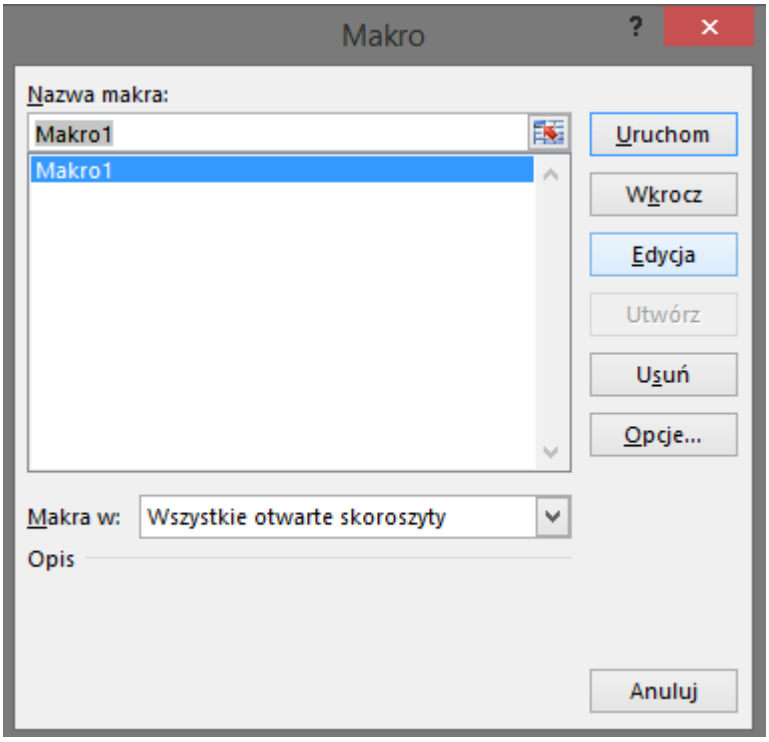
Rafał Zduńczyk

Lato 2018

Treść

Visual Basic Plik → Opcje → Dostosowywanie wstążki. ☑ Deweloper.

Rejestrowanie makr Po kliknięciu na Zarejestruj makro w menu Developer, wszystkie akcje w Excelu zostają nagrane i przetłumaczone na kod w języku Visual Basic. Nie są rejestrowane ruchy myszki, otwieranie menu, czy obsługa innych niż Excel aplikacji. Kiedy mamy nagrane wszystko, czego potrzebujemy, klikamy Zatrzymaj rejestrowanie. Kod nagranej pracy możemy obejrzeć wybierając Makra z menu Deweloper...



...i wybierając opcję Edycja. Przykładowe Makro powstałe w ten sposób:

```
Sub Makro2()
'
' Makro2 Makro
'

'
    ActiveCell.FormulaR1C1 = "=2"
    Range("I6").Select
End Sub
```

Kluczowe słowo `Sub` to skrót od Subroutine lub Subprocedure, a Makro to skrót od Macroinstruction. Znaczenie tego kodu jest następujące: w aktywnej komórce została wpisana komenda o zakresie 1 wiersz × 1 kolumna, a ta formuła to `=2`. Następnie został wybrany pewien zakres (tu: jedna komórka: `I6`). Ten fragment kodu odpowiada wciśnięciu `Enter` po wpisaniu formuły.

Gdyby przed rejestracją makra była zaznaczona opcja Użuj odwołań względnych (pod Zarejestruj), to ta linijka wyglądałaby tak

```
ActiveCell.Offset(1, 0).Range("A1").Select
```

Tutaj `Offset` odpowiada excelowej funkcji `przesunięcie`. Cały tekst oznacza zaznaczenie (`Select`) jednej komórki (`"A1"`) bezpośrednio pod (`Offset(1, 0)`) komórką dotąd aktywną (`ActiveCell`). W ten sam sposób — `Range`, nawias, cudzysłów — odwołuje się do większego zakresu, np. `A1:B5` lub do komórki, która ma nazwę.

Ikona Po zarejestrowaniu makra. Plik → Opcje → Dostosowywanie wstążki. Wybierz polecenia z Makra. Nowa Karta (Niestandardowa). Dodaj >>

Komentarze Komentarz rozpoczyna apostrof — dwa klawisze w prawo od L, bez Shift.

Komentarze blokowe Są dwie możliwości. Pisząc komentarz przed złamaniem linii wciskając [↵], wciśnij [Spacja] i podkreślnik, czyli [Shift] + [Lewy minus]. Drugi sposób. W oknie edytora z menu View wybierz Toolbars i zaznacz [✓] Edit. Pojawią się wtedy przyciski [Comment]/[Uncomment]

Duże/małe litery VBA **nie** jest „case sensitive", nie odróżnia dużych liter od małych. Choć, jak widać, edytor trzyma się zasady zaczynania słów dużą literą, zaleca się nie stosowanie tej praktyki samemu. Łatwiej w ten sposób wykryć własne błędy: skoro edytor przerobił pierwsze litery na duże, to znaczy, że rozpoznał komendy i nazwy funkcji. Jeśli sami zaczniemy je dużą literą, nie zobaczymy tego efektu.

1 Zmienne

Deklaracja zmiennej bez określenia typu: `Dim x`
Deklaracja zmiennej z okresleniem typu: `Dim x As String`
Deklaracje można łączyć w bloki i oddzielać przecinkami: `Dim x As String, y As String, a As Long`
Deklaracja stałej: `Const pi as Single = 3.1415927`

Typ	**Rozmiar (bajty)**	**Wartości**
Boolean	2	`True` (1)/`False` (0)
Byte	1	0–255
Date	8	od 1.01.100 do 31.12.9999
Currency	8	od ok $-9 \cdot 10^{14}$ do $9 \cdot 10^{14}$ (+4 miejsca po kropce)
Integer	2	od $-32\,768$ do $32\,767$
Long	4	od ok. $-2 \cdot 10^{10}$ do $2 \cdot 10^{10}$
Single	4	wymierne ok. $\pm 3 \cdot 10^{38}$, dokładność do ok. 10^{-45}
Double	8	wymierne ok. $\pm 3 \cdot 10^{308}$, dokładność do ok. 10^{-324}
String * n	n	ciągi znaków o długości $\leqslant n$ (do 65 400 znaków)
String	$10 + n$	j.w. do 2 milionów znaków
Object	4	odwołanie do dowolnego obiektu
Variant	16	jakiekolwiek numeryczne
Variant	$22 + n$	jakiekolwiek tekstowe

Wartości zwracane przez funkcję `VarType` oraz odpowiadające im funkcje konwersji:

0. `vbEmpty`
1. `vbNull`
2. `vbInteger` `CInt`
3. `vbLong` `CLng`
4. `vbSingle` `CSng`
5. `vbDouble` `CDbl`
6. `vbCurrency` `CCur`
7. `vbDate` `CDate`
8. `vbString` `CStr`
9. `vbObject`
10. `vbError`
11. `vbBoolean` `CBool`
12. `vbVariant` `CVar`
13. `vbDataObject` (np. zawartość schowka)
14. `vbDecimal` `CDec`
17. `vbByte` `CByte`

Przykład:

```
Sub one()
Dim a, b As String
MsgBox "a: " & VarType(a) & Chr(10) & "b: " & VarType(b)
a = 1
b = 1
MsgBox "a: " & VarType(a) & Chr(10) & "b: " & VarType(b)
End Sub
```

Najpierw zmienna `a` nie ma typu (`vbEmpty`), a potem dostaje przydzielony pierwszy z brzegu typ liczbowy (`vbInteger`). Zmienna `b` jest cały czas typu tekstowego (`vbString`).

2 Instrukcje warunkowe

2.1 If

2.1.1 Jednowierszowa

W tym przypadku **nie pize się** `End If`

```
If Sheets("Arkusz2").Range("A1") = 1 Then MsgBox "Jeden"
```

Kolejne komendy można oddzielać dwukropkami:

```
a = Sheets("Arkusz2").Range("A1")
If a = True Then MsgBox "Prawda": Sheets("Arkusz2").Range("A2") = 7
```

2.1.2 Blokowa

W postaci blokowej trzeba wszystko zamknąć za pomocą `End If`

```
If a = True Then
    MsgBox "Prawda"
    Sheets("Arkusz2").Range("A2") = 7
End If
```

Obsługa większej liczby opcji:

```
If a = True Then
    MsgBox "Prawda"
ElseIf a = False Then
    MsgBox "Fałsz"
Else
    MsgBox "Nie wiem"
End If
```

```
If a = True Then
    MsgBox "Prawda"
ElseIf a = False
    MsgBox "Fałsz"
End If
```

2.2 Select Case

```
Select Case a
Case Is < 0
    MsgBox "Liczba ujemna"
Case 0
    MsgBox a
Case 1.5 To 3
    MsgBox "Między 1,5 oraz 3"
Case 4, 6, 8, 10
    MsgBox "Parzyste"
Case "Nic"
    MsgBox "Tekst"
Case Else
    MsgBox "Coś innego"
End Select
```

3 Łamanie linii wewnątrz poleceń

W długich instrukcjach można łamać linie za pomocą podkreślnika po spacji:

```
If (s >0 And s < 1) Or (s > 2 And s < 3) Or (s > 4 And s < 5) Or _
(s >7 And s < 8) Then
```

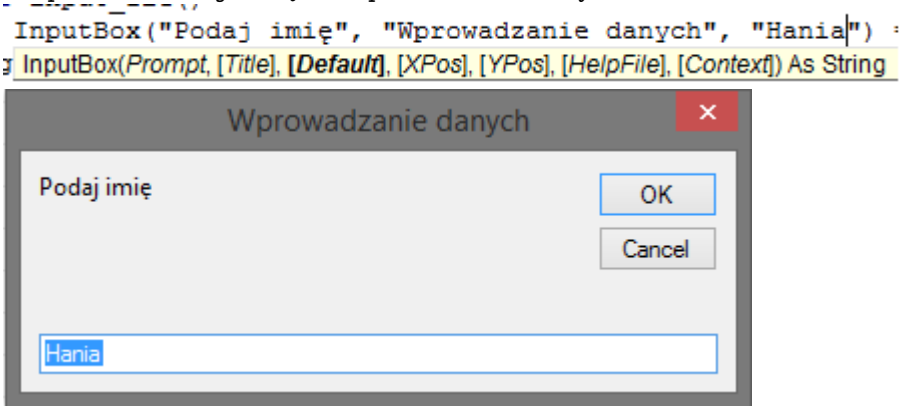
4 InputBox

4.1 Składnia

`IputBox` to funkcja, która ma wymagany argument: podpowiedź oraz kilka argumentów opcjonalnych:

2. tytuł — pojawi się na pasku u góry
3. wartość domyślna — pojawi się w okienku w miejscu wprowadzania danych
4. położenie w poziomie — domyślnie: wyśrodkowane
5. położenie w pionie — domyślnie: wyśrodkowane

```
a = InputBox("Podaj imię", "Wprowadzanie danych", "Hania", 5000, 500)
```



4.2 Zwracane wartości

- tekst wpisany przez użytkownika
- domyślny ciąg znaków w przypadku, kiedy się nic nie wpisze
- pusty ciąg, kiedy się wciśnie Cancel (lub gdy nie ma domyślnego tekstu i nic się nie wpisze)

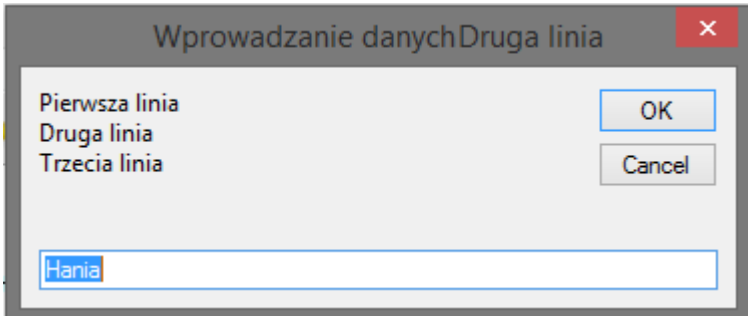
4.3 Łamanie linii

W podpowiedzi można łamać linię za pomocą komend

Nazwa	Kod	Znaczenie
`vbLf`	`Chr(10)`	Line Feed — wypełnienie linii
`vbCr`	`Chr(13)`	Carriage Return — powrót karetki
`vbCrLf`	`Chr(13)+Chr(10)`	
`vbNewLine`		to samo
`vbVerticalTab`	`Chr(11)`	Ręczne łamanie linii, jak Shift + Enter

Przykład:

```
a = InputBox("Pierwsza linia" & Chr(10) & "Druga" & Chr(13) & "Trzecia"
```



W nagłówku nie daje się łamać linii.

5 MsgBox

Ta funkcja, choć na pozór „tylko” wyświetla komunikaty, daje całkiem spore możliwości. Można kontrolować następujące opcje

- Tekst komunikatu
- Tytuł
- Przyciski
- Domyślnie wybrany przycisk
- Ikony
- Zachowanie wobec innych aplikacji (modalność)
- Wyrównywanie tekstu

5.1 Składnia

`MsgBox(Komunikat,opcje,Tytuł)`
Lub, jeśli sam komunikat, bez dalszej interaktywności:
`MsgBox komunikat`

5.2 Opcje

https://analysistabs.com/vba/msgbox/ (na dole strony)
Można

- samemu obliczyć odpowiednią sumę:
  `b = MsgBox("",290)` — przyciski: przerwij, ponów, ignoruj (2), pytanie (32), domyślny drugi przycisk (256),
- zostawić jako sumę: `b = MsgBox("",2 + 32 + 256)`
- użyć nazw: `b = MsgBox("", vbAbortRetryIgnore + vbDefaultButton2 + vbQuestion)`

Modalność (4096): blokuje działanie wszystkich aplikacji do czasu wyboru opcji z tego `MsgBoksa`. W przeciwnym wypadku — tylko Excel zablokowany.

5.3 Wartości MsgBox

1. `vbOK` OK
2. `vbCancel` Cancel lub wciśnięcie Escape, o ile Cancel wyświetlony
3. `vbAbort` Przerwij
4. `vbRetry` Ponów
5. `vbIgnore` Ignoruj
6. `vbYes` Tak
7. `vbNo` Nie

5.4 Własności MsgBox i InputBox z przypisaniem „:=”

```
Sub msg()
MsgBox Title:="tytuł", Buttons:=vbExclamation, Prompt:="Nic"
End Sub

Sub msg()
a = MsgBox(Title:="tytuł", Buttons:=vbExclamation + vbYesNoCancel, Prompt:="Nic")
MsgBox Prompt:=a
End Sub
```

6 Pętle

6.1 Określona liczba iteracji

6.1.1 For Each

For each działa m.in. na

1. tablicach
2. obszarach `Range` i `Selection`

Ad 1. Deklaracja tablicy: `Dim tablica() As String`

Wypełnienie tablicy danymi: `tablica = Array("Jeden, ", "Dwa, ", "Trzy")`

Przykładowe użycie pętli `For Each` do połączenia tekstów z tablicy w jeden tekst:

```
For Each element in tablica
    lista = lista + element
Next element
```

Ad 2. Sprawdzenie parzystości liczb w zaznaczonych komórkach:

```
Dim j As Byte, kom As Range
j = 1
For Each kom In Selection
    MsgBox j & ": " & WorksheetFunction.IsEven(kom.Value)
    j = j + 1
Next kom
```

6.1.2 For

Wypełnienie tablicy przy użyciu pętli `For`:

```
Dim tablica(10) As String, j As Integer
For j = 1 To 10
    tablica(j) = InputBox("Element nr " & CStr(j) &": ")
Next j
```

6.1.3 Wyjście z pętli

Instrukcja Exit For pozwala wyjść z pętli w określonym momencie:

```
j = 1
For Each kom In Selection
    If MsgBox(j & "/" & Selection.Count & ": " & _
        WorksheetFunction.IsEven(kom.Value), _
        vbOKCancel, "True - parzyste, False - nieparzyste") = vbCancel Then Exit For
    j = j + 1
Next kom
```

Uwaga! W przypadku zagnieżdżonej pętli `Exit For` wyprowadza na zewnątrz tylko jednej. Jeśli trzeba opuścić obie pętle jednocześnie, można użyć flagi. Alterntywne rozwiązanie jest w rozdziale 6.2.3. Dla przykładu sprawdźmy, czy w zaznaczonym zakresie liczby ułożone są w kolumnach niemalejąco. Jeśli nie, to nie szukajmy dalej, tylko pokażmy winnego i koniec.

```
Dim a() As Variant, r As Byte, c As Byte, i As Byte, j As Byte, flag As Boolean
a = Selection
r = Selection.Rows.Count
c = Selection.Columns.Count
flag = False
For j = 1 To c
    For i = 1 To r - 1
        If a(i, j) > a(i + 1, j) Then flag = True: Exit For
    Next i
    If flag Then Exit For
Next j
Selection(i + 1, j).Select
MsgBox i + 1 & ":" & j, , "Wiersz:kolumna"
End Sub
```

6.1.4 Step

Sprawdzenie, czy liczba jest pierwsza: sprawdzamy tylko nieparzyste liczby, a więc co drugą począwszy od 3.

```
j = InputBox("")
comp = False
If j > 2 And j Mod 2 = 0 Then
    comp = True
Else
    For k = 3 To Sqr(j) Step 2
        If j Mod k = 0 Then comp = True: Exit For
    Next k
End If
MsgBox Not (comp)
```

6.2 Nieokreślona liczba iteracji: Do While/Until

Znajdowanie pierwszych 50 liczb pierwszych.

```
Dim primes(50) As Integer
i = 0
j = 1
Do While i < 50
    j = j + 1
    comp = False
    If j > 2 And j Mod 2 = 0 Then
        comp = True
    Else
        For k = 3 To Sqr(j) Step 2
            If j Mod k = 0 Then comp = True: Exit For
        Next k
    End If
    If comp = False Then i = i + 1: primes(i) = j
Loop
For i = 1 To 50
    Cells(i, 15).Value = primes(i)
Next i
```

6.2.1 Test pozytywny/negatwny na początku/na końcu

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

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

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

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

Pierwsza opcja jest równoważna z

```
While [warunek]
    [instrukcje]
Wend
```

Ostatnia konstrukcja (z `Wend`) nie daje możliwości testowania na końcu.

6.2.2 Test na początku/na końcu

Dodawajmy liczby z listy, aż suma przekroczy określoną wartość.
Lista: $\{23, 29, 31, 37, 41\}$
Ustalona wartość: 21

```
Dim List() As Integer, i As Integer, _
    sum1 As Integer
List = Array(23, 29, 31, 37, 41)
sum1 = 22
i = -1
Do
    i = i + 1
    sum1 = sum1 + List(i)
Loop While sum1 <= 21
MsgBox i + 1 & ": " & sum1
```

Pętla wykonana zawsze przynajmniej raz
(w tym przypadku: raz)

```
Dim List() As Integer, i As Integer, _
    sum1 As Integer
List = Array(23, 29, 31, 37, 41)
sum1 = 22
i = -1
Do While sum1 <= 21
    i = i + 1
    sum1 = sum1 + List(i)
Loop
MsgBox i + 1 & ": " & sum1
```

Pętla być może nigdy nie wykonana
(w tym przypadku: nigdy)

6.2.3 Wyjście z zagnieżdżonej pętli II — „dummy loop"

```
Dim a() As Variant, r As Byte, c As Byte, i As Byte, j As Byte
a = Selection
r = Selection.Rows.Count
c = Selection.Columns.Count
Do
    For j = 1 To c
        For i = 1 To r - 1
            If a(i, j) > a(i + 1, j) Then Exit Do
        Next i
    Next j
Loop While False
Selection(i + 1, j).Select
MsgBox i + 1 & ":" & j, , "Wiersz:kolumna"
```

6.2.4 Awaryjne przerwanie działania programu

Jeśli przypadkiem powstała i została uruchomiona pętla, z której nie ma możliwości wyjścia, uratować się można kombinacją Ctrl + Pause

7 Funkcje

Stwórzmy funkcję obliczającą pole dowolnego trójkąta z wykorzystaniem wzoru Herona:

$$P = \sqrt{p(p-a)(p-b)(p-c)}, \qquad \text{gdzie } p = \frac{a+b+c}{2}.$$

```
Function area(side1 As Double, side2 As Double, side3 As Double) As Double
Dim p As Double
p = (side1 + side2 + side3) / 2
area = Sqr(p * (p - side1) * (p - side2) * (p - side3))
End Function
```

Funkcja powinna być zapisana w `Module1`, można jej użyć bezpośrednio w komórkach Excela, albo w jakimś programie, ale trzeba w nim zadeklarować argumenty tych samych typów, co występują w funkcji:

```
Sub area1()
Dim a As Double, b As Double, c As Double
a = InputBox("Side 1")
b = InputBox("Side 2")
c = InputBox("Side 3")
MsgBox area(a, b, c)
End Sub
```

8 Funkcje tablicowe i tablice dynamiczne

8.1 Tablice statyczne i dynamiczne

Tablica statyczna nie zmienia rozmiaru:

```
Sub tablica1()
Dim ta(1 To 2, 2 To 3) As Integer
For i = 1 To 2
    For j = 2 To 3
        ta(i, j) = i ^ j
    Next j
Next i
ActiveCell.Range("A1:B2") = ta()
End Sub
```

i musi mieć rozmiar podany jako liczbę:

```
Sub tablica1()
Dim m As Integer, n As Integer
m = InputBox("Wiersze:")
n = InputBox("Kolumny:")
Dim ta(1 To m, 1 To n) As Integer
...
End Sub
```

Tylko z tablicami **dynamicznymi** tak można:

```
Sub tablica1()
Dim ta() As Integer
Dim m As Integer, n As Integer
m = InputBox("Wiersze:")
n = InputBox("Kolumny:")
ReDim ta(1 To m, 1 To n) As Integer
For i = 1 To m
    For j = 1 To n
        ta(i, j) = i ^ j
    Next j
Next i
ActiveCell.Range(Cells(1, 1), Cells(m, n)) _
= ta()
End Sub
```

Uwaga! Komenda `ReDim` użyta do tablicy, która zawierała jakieś dane, powoduje, że dane te zostają utracone. Chcesz zmienić rozmiar niepustej tablicy — przenieś dane do tablicy pomocniczej i dopiero zmieniaj rozmiar.

Uwaga! Można zmieniać tylko ostatni wymiar tablicy. Żeby to obejść można albo tablicę stransponować — tak, żeby żądany wymiar stał się ostatnim, albo przenieść dane do innej tablicy — o odpowiednim rozmiarze.

8.2 Funkcje czytające `Range` jako tablice

Żeby funkcja zrozumiała zakres (Range) jako tablicę, trzeba to do funkcji wczytać jako Range i dopiero wewnątrz funkcji przerobić na tablicę. Poniżej przykład funkcji usuwającej kolumnę o podanym numerze.

```
Function delone(a As Range, n As Long) As Variant
Dim a1 As Variant, c() As Variant
a1 = a
u1 = UBound(a1, 1)
u2 = UBound(a1, 2)
ReDim c(1 To u1, 1 To u2 - 1)
For i = 1 To u1
    For j = 1 To u2
        If j < n Then
            c(i, j) = a1(i, j)
        ElseIf j > n Then
            c(i, j - 1) = a(i, j)
        End If
    Next
Next
delone = c()
End Function
```

Uwaga! Żeby użyć funkcji tablicowej, zaznacza się obszar, wpisuje funkcję z argumentami i zatwierdza Ctrl + Shift + Enter.

8.3 Funkcje zwracające tablice

Stwórzmy funkcję rozwiązującą równanie kwadratowe

$$ax^2 + bx + c = 0,$$

wykorzystującą wzory

$$\Delta = b^2 - 4ac,$$
$$x_1 = \frac{-b - \sqrt{\Delta}}{2a}, \qquad x_2 = \frac{-b + \sqrt{\Delta}}{2a}.$$

Wykorzystajmy zewnętrzną funkcję do policzenia delty:

```
Function delta(a As Double, b As Double, c As Double) As Double
delta = b ^ 2 - 4 * a * c
End Function
```

Funkcja właściwa powinna w zależności od wartości delty zwrócić

- albo macierz 1×2 ,
- albo 1×1 $(\Delta = 0)$,
- albo błąd, jeśli $\Delta < 0$.

Zdefiniujmy pomocniczą dynamiczną tablicę `sols`, która przechowa rozwiązania niezależnie od ich liczby. Zrobimy to tak, że w zależności od delty, `sols` przyjmie odpowiedni rozmiar.

```
Function quadratic(a As Double, b As Double, c As Double) As Double
Dim de As Double, sols() As Double
de = delta(a, b, c)
MsgBox de
If a <> 0 Then
    Select Case de
        Case Is > 0
            ReDim sols(1 To 2)
            sols(1) = (-b - Sqr(de)) / (2 * a)
            sols(2) = (-b + Sqr(de)) / (2 * a)
        Case 0
            ReDim sols(1 To 1)
            sols(1) = -b / (2 * a)
        Case Else
    End Select
    Else
        If b <> 0 Then
            ReDim sols(1 To 1)
            sols(1) = -c / b
        End If
    End If
quadratic = sols
End Function
```

9 Opcjonalne argumenty funkcji

Powyższa funkcja nie obsługuje ujemnej delty. Mamy przecież

```
        Case Else
    End Select
```

Wtedy jednak pierwiastki są, tylko że zespolone i wzory dalej obowiązują, przy czym

$$\operatorname{Re} x_1 = -\frac{b}{2a}, \qquad \operatorname{Im} x_1 = -\frac{\sqrt{-\Delta}}{2a},$$

$$\operatorname{Re} x_2 = -\frac{b}{2a}, \qquad \operatorname{Im} x_2 = \frac{\sqrt{-\Delta}}{2a}.$$

Pozwólmy użytkownikowi dodać czwarty — **opcjonalny** argument (Prawda/Fałsz) — jeśli chce uwzględnić ten przypadek:

```
Function quadratic2(a As Double, b As Double, c As Double, _
Optional complex As Boolean) As Variant
```

Uwaga! Wszystkie argumenty po słowie `Optional` są uważane za opcjonalne.

Zastanówmy się jednak, jak chcemy żeby funkcja się zachowała, kiedy użytkownik nie poda opcjonalnego argumentu. Można to rozwiązać na 2 sposoby:

1. Przypisać w nagłówku funkcji pewną domyślną wartość,
2. Użyć funkcji `IsMissing`, która do dobrego działania wymaga argumentu typu `Variant`.

Ad 1. Modyfikacja nagłówka funkcji:

```
Function quadratic2(a As Double, b As Double, c As Double, _
Optional complex As Boolean = False) As Variant
```

Obsługa opcjonalnego argumentu:

```
    Case Else
        If complex = True Then
            ReDim sols(1 To 2, 1 To 2)
            sols(1, 1) = -b / (2 * a)
            sols(1, 2) = -Sqr(-de) / (2 * a)
            sols(2, 1) = -b / (2 * a)
            sols(2, 2) = Sqr(-de) / (2 * a)
        End If
    End Select
```

Ad 2. Nagłówek funkcji:

```
Function quadratic3(a As Double, b As Double, c As Double, _
Optional complex As Variant) As Variant
```

Obsługa opcjonalnego argumentu:

```
    Case Else
        If IsMissing(complex) = False Then
            If complex = True Then
                ReDim sols(1 To 2, 1 To 2)
                sols(1, 1) = -b / (2 * a)
                sols(1, 2) = -Sqr(-de) / (2 * a)
                sols(2, 1) = -b / (2 * a)
                sols(2, 2) = Sqr(-de) / (2 * a)
            End If
        End If
    End Select
```

Uwaga! `IsMissing` wymaga argumentu typu `Variant`

Stwórzmy funkcję obliczającą sumę neiskończonego szeregu geometrycznego, a jeśli poda się trzeci argument — liczbę wyrazów — obliczającą n-tą sumę częściową.

```
Function geom(first As Double, ratio As Double, Optional count As Double) _
As Double
If IsMissing(count) Then
    If Abs(ratio) < 1 Then geom = first / (1 - ratio)
Else
    geom = first * (1 - ratio ^ count) / (1 - ratio)
End If
End Function
```

Przetestujmy z argumentami `first = 1` oraz `ratio = 0,5`. Powinno wyjść 2, a wychodzi 0.

Wystarczy zmienić deklarację opcjonalnego argumentu

```
Function geom(first As Double, ratio As Double, Optional count As Variant) _
As Double
```

i już działa dobrze. Przynajmniej pod tym względem. Dla argumentów `first = 1` oraz `ratio = 2` daje bezsensowną wartość 0.

10 Obsługa błędów

10.1 Generowanie błędów Excela

Choć zwykle zmierzamy raczej do ukrywania systemowych komunikatów o błędach, w przypadku ostatniej funkcji aż prosi się o komunikat `#ARG!` Zatem dodajmy ten komunikat w przypadku szeregu rozbieżnego

```
If Abs(ratio) < 1 Then
    geom = first / (1 - ratio)
Else
    geom = CVErr(xlErrValue)
End If
```

Trzeba też zmienić deklarację typu zwracanych przez funkcję wartości na `Variant`

Oto kompletna lista błędów Excela z numerami i ich kody

1. `#ZERO!` `xlErrNull`
2. `#DZIEL/0!` `xlErrDiv0`
3. `#ARG!` `xlErrValue`
4. `#ADR!` `xlErrRef`
5. `#NAZWA?` `xlErrName`
6. `#LICZBA!` `xlErrNum`
7. `#N/D!` `xlErrNA`

10.2 Wyłapywanie i rozróżnianie błędów

Do wyłapania błędu służy instrukcja `On Error`, którą należy umieścić przed instrukcją, która może powodować błąd. Własności `Number` oraz `Description` pozwalają lepiej zidentyfikować i opisać błąd:

```
Sub error2()
Dim x, y, z
x = InputBox("x:")
y = InputBox("y:")
On Error GoTo handler1
z = x / y
MsgBox z
Exit Sub
handler1:
Select Case Err.Number
    Case 11
        MsgBox "Division by 0", , Err.Number & " " & Err.Description
    Case 13
        MsgBox "Can't be a string!", , Err.Number & " " & Err.Description
End Select
End Sub
```

`On Error` działa aż do kolejnego `On Error` lub wyłączenia błędów komendą `On Error GoTo 0`

```
Sub error3()
Dim x, y, z, a, b, c
x = InputBox("x:")
y = InputBox("y:")
On Error GoTo handler1
```

```
z = x / y
MsgBox z
a = InputBox("a:")
b = InputBox("b:")
c = a / b
Exit Sub
handler1:
Select Case Err.Number
    Case 11
        MsgBox "y can't be 0", , Err.Number
    Case 13
        MsgBox "y can't be a string!", , Err.Number
End Select
End Sub
```

Błąd jest źle wyłapany (np. $x = y = a = 1$, $b = 0$). Program „nie wie" o co chodzi. Nad podejrzaną komendą można roboczo wyłączyć obsługę błędu `On Error GoTo 0`, żeby sprawdzić, co się dzieje.

```
Sub error3()
Dim x, y, z, a, b, c
x = InputBox("x:")
y = InputBox("y:")
On Error GoTo handler1
z = x / y
MsgBox z
a = InputBox("a:")
b = InputBox("b:")
On Error GoTo 0
c = a / b
Exit Sub
handler1:
Select Case Err.Number
    Case 11
        MsgBox "y can't be 0", , Err.Number
    Case 13
        MsgBox "y can't be a string!", , Err.Number
End Select
End Sub
```

Taraz można dodać obsługę drugiego błędu:

```
Sub error3()
Dim x, y, z, a, b, c
[...]
On Error GoTo handler2
c = a / b
Exit Sub
handler1:
Select Case Err.Number
    Case 11
        MsgBox "y can't be 0", , Err.Number
    Case 13
        MsgBox "y can't be a string!", , Err.Number
End Select
handler2:
Select Case Err.Number
    Case 11
        MsgBox "b can't be 0", , Err.Number
    Case 13
        MsgBox "b can't be a string!", , Err.Number
End Select
End Sub
```

10.3 Naprawa błędu: `Resume` oraz `Resume Next`

`Resume Next` powoduje powrót do linii **następnej** po generującej błąd:

```
Sub error2()
Dim x, y, z
x = InputBox("x:")
y = InputBox("y:")
On Error GoTo handler1
z = x / y
MsgBox "x/y = " & x & "/" & y & " = " & z
Exit Sub
handler1:
Select Case Err.Number
    Case 11
        MsgBox "Division by 0", , Err.Number & " " & Err.Description
    Case 13
        MsgBox "Can't be a string!", , Err.Number & " " & Err.Description
End Select
y = 1: z = x
Resume Next
End Sub
```

`Resume` powoduje powrót do linii generującej błąd:

```
Sub error2()
Dim x, y, z
x = InputBox("x:")
y = InputBox("y:")
On Error GoTo handler1
z = x / y
MsgBox "x/y = " & x & "/" & y & " = " & z
Exit Sub
handler1:
Select Case Err.Number
    Case 11
        MsgBox "Division by 0", , Err.Number & " " & Err.Description
    Case 13
        MsgBox "Can't be a string!", , Err.Number & " " & Err.Description
End Select
y = InputBox("Podaj y nie równe zero i nie tekst")
Resume
End Sub
```

Uwaga! Widać, że wpisanie złego `x` (nie-liczby) powoduje wpadnięcie w pętlę bez wyjścia. Instrukcja `Resume` odesłała sterowanie do linii

```
z = x / y
```

bo to tylko ona powoduje błąd.

10.4 Ignorowanie błędów

Można użyć `On Error Resume Next`, by zignorować błąd:

```
Sub error4()
Dim x, y, z, a, b, c
x = InputBox("x:")
y = InputBox("y:")
On Error Resume Next
z = x / y
MsgBox "x/y = " & x & "/" & y & " = " & z
End Sub
```

10.5 Przeskakiwanie bloku instrukcji

Jeśli po obsłudze błędu chcemy wrócić do jakiejś dalszej linii, można użyć `Resume` następująco

```
On Error Goto handler1
[komenda genrująca błąd]
[inne komendy]
...
[inne komendy]
comeback1: ' Stąd chcemy kontynuować
[komendy]
...
[komendy]
Exit Sub
handler1:
[naprawa błędu]
Resume comeback1
End Sub
```

Uwaga! Etykieta `comeback1` jest **poniżej** `On Error Goto handler1`

10.6 Błąd w błędzie

```
Sub error5()
Dim x
On Error GoTo handler1
x = 1 / 0
Exit Sub
handler1:
MsgBox "Handler1 entered"
On Error GoTo -1
On Error GoTo handler2
x = 1 / 0
Exit Sub
handler2:
MsgBox "Handler2 entered"
End Sub
```

10.7 Celowe wywoływanie błędów

Komenda `Error 13` generuje błąd o podanym numerze (tu: 13). Odradza się jej stosowanie. Jest lepsza alternatywa:

`Err.Raise` numer, źródło, opis

Wolne numery są od 513 do 65535. Jako źródło można podać nazwę makra lub dokładniejsze „współrzędne”. Będzie to przypisane do własności `Source`, zatem można się do niej odwołać przez `Err.Source`, zaś do opisu — przez `Err.Description`

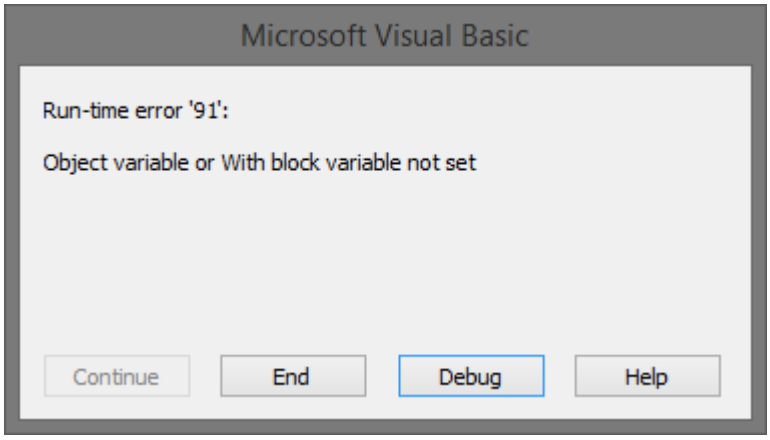
11 Deklaracje

11.1 Różnice między `Set` i zwykłym przypisaniem

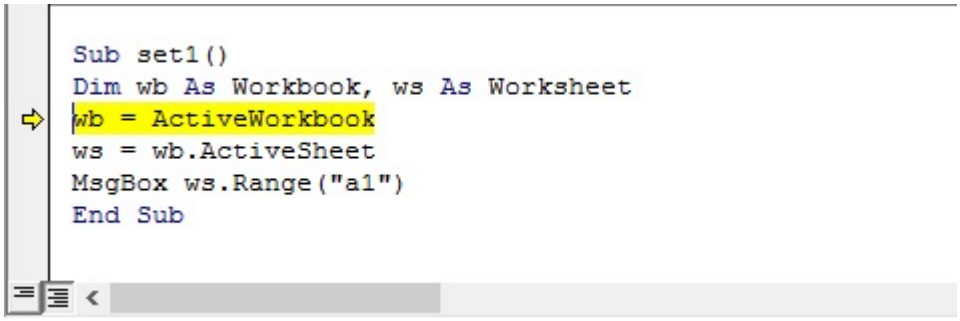
Odwołanie do konkretnej komórki w konkretnym skoroszycie i arkuszu. Zaczynamy od deklaracji

```
Sub set1()
Dim wb As Workbook, ws As Worksheet
wb = ActiveWorkbook
ws = wb.ActiveSheet
MsgBox ws.Range("a1")
End Sub
```

Przy próbie uruchomienia dostajemy błąd



i wskazana zostaje linia



W tej właśnie linii brakuje `Set` na początku. Brakuje go także w następnej, gdyż to słowo musi być zawsze, kiedy chcemy przypisać zmiennej obiekt, który może mieć jakieś właściwości (lub do którego można stosować metody), czyli coś po kropce. Teraz będzie działać

```
Sub set1()
Dim wb As Workbook, ws As Worksheet
Set wb = ActiveWorkbook
Set ws = wb.ActiveSheet
MsgBox ws.Range("a1")
End Sub
```

Podobnie dla pojedynczej komórki:

- jeśli chce się przypisać zmiennej domyślną własność komórki, czyli jej wartość — stosuje się zwykłe przypisanie
- jeśli chce się przypisać zmiennej komórkę jako obiekt, by móc potem odwoływać się do własności komórki — stosuje się `Set`

To daje błąd:

```
Sub set2()
Dim mycell
mycell = Range("C3")
MsgBox mycell.Interior.Color
End Sub
```

To też daje błąd:

```
Sub set2()
Dim mycell As Range
mycell = Range("C3")
MsgBox mycell.Interior.Color
End Sub
```

A to już działa:

```
Sub set2()
Dim mycell As Range
Set mycell = Range("C3")
MsgBox mycell.Interior.Color
End Sub
```

11.2 Różnice między `Dim` i `Public`

Zmienne można deklarować na początku modułu poza makrami. Zmienna zadeklarowana komendą `Dim` jest widziana jedynie z danego modułu — inne moduły jej nie widzą. Żeby ją widziały — trzeba ją zadeklarować komendą `Public` i musi być w module — jeśli będzie w arkuszu, albo w formularzu, deklaracja `Public` nie będzie skuteczna.

Umieśćmy w jednym module kod

```
Public a1a1 As Integer
Dim b1b1 As Integer

Sub start_first()
a1a1 = 1
b1b1 = -1
Call show_values
End Sub
```

A w drugim

```
Sub show_values()
MsgBox VarType(a1a1)
MsgBox VarType(b1b1)
'Call show_message
End Sub
```

Uruchomienie makra z pierwszego modułu pokazuje, że drugi moduł nie widzi zmiennej `b1b1` — myśli, że jest pusta.

11.3 Różnice między `Sub` i `Private Sub`

Dodajmy do pierwszego modułu

```
Public a1a1 As Integer
Dim b1b1 As Integer

Sub start_first()
a1a1 = 1
b1b1 = -1
Call show_values
End Sub

Private Sub show_message()
MsgBox "Can you see me?"
End Sub
```

A w drugim odkomentujmy `Call`

```
Sub show_values()
MsgBox VarType(a1a1)
MsgBox VarType(b1b1)
Call show_message
End Sub
```

Teraz drugi moduł nie widzi makra `show_message`. Ale może się tam włamać zamiast `Call` używając `Application.Run`:

```
Sub burgle()
Application.Run ("show_message")
End Sub
```

11.4 Funkcje i ByVal/ByRef

`By Value` (*przez wartość*) oraz `By Reference` (*przez odwołanie*) określają sposób, w jaki funkcje i makra mają dostęp do zmiennych, które przetwarzają. Domyślnie jest to `ByRef`. Dostęp `By Value` powoduje, że procedura nie może zmienić zmiennej, operuje tylko na kopii zmiennej, podczas, gdy `ByRef` wpływa na wartość oryginału (odwołanie do faktycznej zmiennej, a nie tylko jej wartości).

```
Sub boxes2()
Dim x As Long
x = InputBox("x=?")
MsgBox "squarebyval(x): " & squarebyval(x) & Chr(10) & "             x : " & x
MsgBox "squarebyref(x): " & squarebyref(x) & Chr(10) & "             x : " & x
End Sub

Function squarebyval(ByVal x As Long)
x = x * x
squarebyval = x
End Function
```

```
Function squarebyref(ByRef x As Long)
x = x * x
squarebyref = x
End Function
```

11.5 Makra z parametrami

```
Sub imieplusnazwisko(imie As String, nazwisko As String)
Dim result As String
result = imie + " " + nazwisko
MsgBox result
End Sub

Sub pokaz()
imieplusnazwisko nazwisko:="Wielki", imie:="Aleksander"
End Sub
```

11.6 Makra i ByVal/ByRef

```
Sub outer()
Dim A As Long, B As Long
A = 2
B = 3
inner X:=A, Y:=B
MsgBox "A=" & A & Chr(10) & "B=" & B & Chr(10) & "X=" & X & Chr(10) & "Y=" & Y
End Sub

Sub inner(ByRef X As Long, ByVal Y As Long)
X = X + 1
Y = Y + 1
End Sub
```

Widać: `inner` zmieniło A, ale nie zmieniło B. X i Y w ogóle nie są widziane przez `outer`. Można wymusić `ByVal`:

```
Sub outer2()
Dim A As Long, B As Long
A = 2
B = 3
inner (A), (B)
MsgBox "A=" & A & Chr(10) & "B=" & B & Chr(10) & "X=" & X & Chr(10) & "Y=" & Y
End Sub
```

12 Zewnętrzne pliki

12.1 Otwieranie pliku daną aplikacją

Otwieranie pliku tekstowego notatnikiem:

```
Sub OpenInNotepad()
Dim MyTxtFile
    MyTxtFile = Shell("C:\WINDOWS\notepad.exe C:\...\file.txt", 1)
End Sub
```

Dostępne opcje:

0. `vbHide` (jako proces w tle)
1. `vbNormalFocus` (otwiera się domyślnie)
2. `vbMinimizedFocus` (zminimalizowana, ale aktywna)

3. `vbMaximizedFocus` (zmaksymalizowan, aktywna)
4. `vbNormalNoFocus` (otwarta normalnie, nieaktywna)
6. `vbMinimizedNoFocus` (sensowniejsze niż 2)

12.2 Otwieranie samej aplikacji

```
Sub LaunchNotepad()
Call Shell("Explorer.exe C:\Windows\system32\notepad.exe", vbNormalFocus)
End Sub
```

12.3 Pliki CSV (Comma Separated Value)

Następująca komenda służy do otwierania plików
`Open` [co] `For` [po co] `As` [Nr pliku]
przy czym:

[co] musi zawierać pełną ścieżkę z nazwą pliku i rozszerzeniem (.txt lub .csv)

[po co] może przyjąć wartości:

- `Append` — dopisywanie na końcu pliku
- `Ouptut` — nadpisywanie
- `Input` — odczyt
- `Binary`, `Random` — nie będą nam potrzebne na razie.

[Nr pliku] musi być w formacie `#5` (5 to oczywiście tylko przykład)

Uwaga! Po zakończeniu pracy z plikiem koniecznie pamiętaj o zamknięciu go:
`Close #5`

12.3.1 Operacje na otwartym pliku

Jeśli plik otwarty `For Input` dostępne są

- funkcja `EOF` *End of File*, zmienną jest numer pliku bez `#`, wartości to `True` oraz `False`
- komenda `Line Input` z dwoma parametrami: nr pliku poprzedzony `#` oraz zmienna, pod którą będzie podstawiony odczytany tekst

Najprostszy schemat (przepisanie pliku do arkusza)

```
Open ... For Input As #1
row = 0
Do Untill EOF(1)
    Line Input #1, line
    ActiveCell.Offset(row,0).value = line
    row = row +1
Loop
Close #1
```

Jeśli plik otwary jest `For Output` lub `For Append`, to dostępna jest komenda
`Write` też z dwoma parametrami: nr pliku (poprzedzony `#`) oraz co zapisać, czyli odwołanie do zmiennej lub tekst wprost (w cudzysłowie):

```
Write #3, zmienna
Write #3, "Tekst do wpisania"
```

Uwaga! Zapis do pliku otwartego `For Output` **wykasuje bez ostrzeżenia** dane z tego pliku!

12.3.2 Ślad macierzy, wykorzystanie funkcji Split

Split to funkcja zwracająca macierz. Argumentem jest tekst i znak, który ma posłużyć do rozdzielenia tekstu na części — najczęściej spacja lub przecinek. Np.

```
Split("Ala ma kota", " ")
```

da w wyniku tablicę z trzema wyrazami: `Ala`, `ma` oraz `kota`. Zaś

```
text1 = "migdały, orzechy laskowe, nerkowce, orzechy włoskie''
Split(text1, ",")
```

rozdzieli tekst na cztery części, przy czym pierwszym znakiem w drugiej, trzeciej i czwartej będzie spacja.

Umieśćmy w dowolnym folderze

- plik z rozszerzeniem `.csv` zawierający kwadratową macierz z liczbami w wierszach oddzielonymi przecinkami
- arkusz excela z obsługą makr

Napiszmy w tym arkuszu program obliczający ślad macierzy z utworzonego pliku. Oto kod

```
Dim path1 As String, file1 As String, count1 As Integer, _
    split1() As String, split2 As String, trace1 As Single
path1 = ThisWorkbook.Path & "\"
file1 = InputBox("Podaj nazwę pliku bez rozszerzenia") & ".csv"
Open path1 & file1 For Input As #1
    count1 = 0
    Do Until EOF(1)
        Line Input #1, line1
        ReDim split1(count1)
        split1 = Split(line1, ",")
        split2 = split2 + split1(count1) + "+"
        trace1 = trace1 + CSng(split1(count1))
        count1 = count1 + 1
    Loop
Close #1
MsgBox Left(split2, Len(split2) - 1) & "=" & trace1
```

W powyższym kodzie

- ścieżkę i nazwę pliku trzeba było rozdzielić backslashem „\”, gdyż inaczej nazwa folderu połączyłaby się w całość z nazwą pliku
- zmienna `split1` w każdej iteracji (n) przechowuje pierwsze n wyrazów z n-tego wiersza
- `split2` przechowuje jako tekst wyrazy macierzy połączone znakami plus „+”
- `trace` faktycznie kumuluje sumę wyrazów z przekątnej, czyli ślad macierzy
- `count` liczy wiersze
- `MsgBox` wyświetla `split2` z ostatnim bezsensownym plusem zamienionym na znak równości i wartość śladu macierzy.

13 UserFormy i zdarzenia

13.1 Kontrolki

`UserForm` wstawia się jak moduł. Jeśli chodzi o widzialność obiektów wewnątrz, obowiązują zasady jak dla modułów. Zmienne, które mają być widziane bez względu na to, czy `UserForm` jest otwarty, czy zamknięty, trzeba deklarować jako `Public` w **module**. Oto przykładowe obiekty (formanty, kontrolki), które może zawierać `UserForm` i ich przykładowe własności:

- `CommandButton`, czyli przycisk polecenia. Właściwości:

- ▷ `(Name)`, czyli nazwa
- ▷ `Caption`, czyli co jest na przycisku napisane
- ▷ `ControlTipText`, czyli podpowiedź, która jak komenarz jest widoczna, kiedy najedzie się myszką na przycisk
- ▷ `Font`, czyli czcionka, można ustawić nie tylko krój, ale też rozmiar, kolor itd.
- ▷ `Enabled`, czyli, czy obiekt jest aktywny, czy użytkownik może go zmieniać
- ▷ `Visible`, czy kontrolka jest widoczna
- ▷ `Left` oraz `Top`, odpowidają za współrzętne położenia lewego górnego rogu obiektu
- ▷ `Height` oraz `Width` — wysokość i szerokość

- `TextBox`, pole tekstowe, nie ma `Caption`, tylko `Text`, czyli domyślny tekst, który widać po otwarciu. Użytkownik może edytować tekst w czasie działania aplikacji.
- `ToggleButton`, przycisk przełącznika, podobny do przycisku polecenia, ma dwa możliwe stany dla właściwości `Value`: `True/False`, przy czym `True` to wciśnięty, a `False` — zwolniony.
- `Label`, czyli etykieta. Tekstu etykiety (`Caption`) użytkownik nie może edytować bezpośrednio, lecz można to robić przez makra powiązane z kontrolkami.
- `CheckBox` — funkcjonalnie podobne do `ToggleButton`, różni się wizualnie: wyświetla się kwadracik, który użytkownik może „zaptaszkować”, napis jest domyślnie po prawej stronie.
- `OptionButton`, wizualnie podobne do `CheckBox`, ale dodatkowo jest możliwość zgrupowania kilku `OptionButton`ów za pomocą ramki (`Frame`). W obrębie jednej ramki tylko jeden `OptionButton` może mieć wartość `True`, kliknięcie automatycznie odznacza pozostałe `OptionButton`y.
- `Pages` (`Multipage`) oraz `TabStrip`. Żeby dodać nową zakładkę/stronę, trzeba kliknąć prawym przyskiem myszy na nagłówek. Za aktualnie otwartą zakładkę odpowiada własność `TabStrip1.Value`, przy czym wartości zaczynają się od 0 (pierwsza od lewej zakładka). Nagłówki kontrolować można powiązując nadanie im nazw ze zdarzeniem (patrz dalej) uruchomienia `UserForm`a:

  ```
  Private Sub UserForm1_Initialize()
  MultiPage1.Pages(0).Caption = "Pierwsza"
  MultiPage1.Pages(1).Caption = "Druga"
  TabStrip1.Tabs(0).Caption = "Takie"
  TabStrip1.Tabss(0).Caption = "Inne"
  End Sub
  ```

  Umieszczenie obiektu na `TabStrip` powoduje, że jest on widoczny na wszystkich zakładkach. Tym się `TabStrip` różni od `MultiPage`, gdzie każdą stronę można zorganizować indywidualnie.
- `ScrollBar` oraz `SpinButton`, czyli pasek przewijania i podobna kontrolka bez paska (same końcówki). Można edytować dla nich własności:
  - ▷ `Min`, czyli minimum
  - ▷ `Max`, czyli maksimum
  - ▷ `LargeChange` oraz `SmallChange`, czyli skok — zmiana wartości przy kliknięciu
  - ▷ `Orientation`, które może przyjąć wartości: -1 (automatyczna, 0 (pionowa), 1 (pozioma)

13.2 Właściwości `Enabled` oraz `Locked`

Zilustrujemy różnice własności `Enabled` oraz `Locked` na przykładzie

- pola tekstowego
- pola wyboru
- przycisku polecenia

umieszczonych w formularzu (te w arkuszu mają problem z `Locked`).

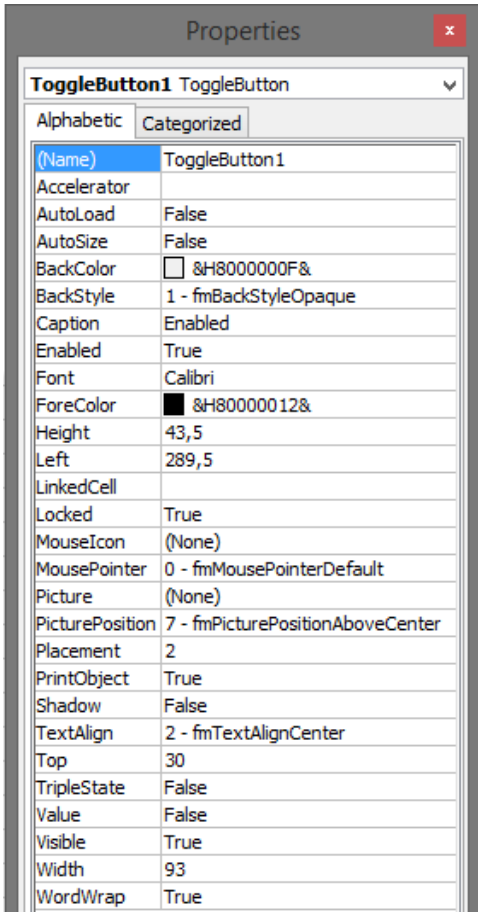
Stwórzmy fomularz z wyżej wymienionymi obiektami oraz dwoma przyciskami opcji z następującymi kodami związanymi ze zdarzeniem `_Click`:

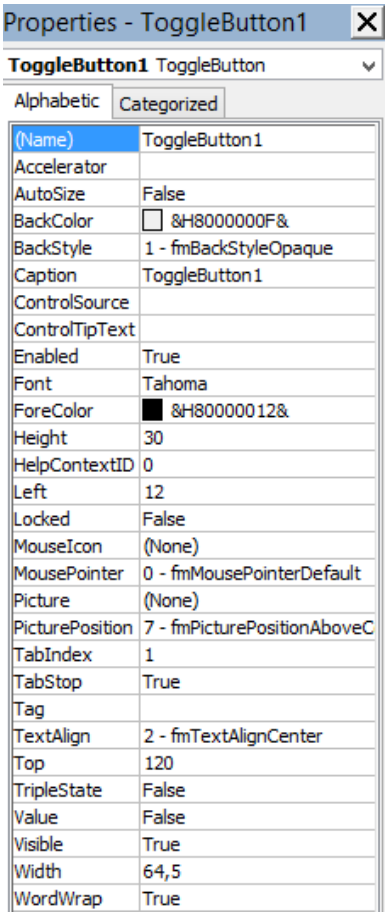
```
If ToggleButton1.Value = True Then
        TextBox1.Locked = True
        CheckBox1.Locked = True
        CommandButton1.Locked = True
        ToggleButton1.Caption = "Locked"
    Else
        TextBox1.Locked = False
        CheckBox1.Locked = False
        CommandButton1.Locked = False
        ToggleButton1.Caption = "Unlocked"
End If
```

i analogicznie z drugim przyciskiem i opcją `Enabled`. Sprawdźmy edycję i kopiowanie.
Uwaga! `Locked` i `Enabled` nie mają wpływu na możliwość zmian własności kontrolek kodem. Możemy się o tym przekonać dodając na formularzu przycisk polecenia i z jego `Click`iem wiążąc kod

```
If CheckBox1.Value = True Then
CheckBox1.Value = False
Else
CheckBox1.Value = True
End If
```

13.3 Porównanie właściwości w arkuszu i w formularzu





Powyżej widać właściwości `ToggleButton`a — po lewej w arkuszu, po prawej w formularzu. W arkuszu widać m.in. własności `LinkedCell`, czyli adres komórki, króra może przechowywać aktualną wartość kontrolki oraz `Placement`, które odpowiada za ewentualne zmiany położenia kontrolki przy manipulowaniu szerokością i wysokością komórek w arkuszu. Kontrolka może być statyczna względem lewego górnego roku arkusza, może się tylko przesuwać wraz z lewym górnym rogiem komórki, w której jest lewy górny róg kontrolki, lub ostatecznie — może ona także zmieniać swój rozmiar wraz z komórkami, na tle których oryginalnie była umieszczona.

13.4 Zdarzenia

Podwójne kliknięcie na obiekt w trybie projektowania automatycznie włącza edycję kodu powiązanego z domyślnym zdarzeniem obiektu, np. `CommandButton1_Click()` Pozostale dostępne zdarzenia widać w prawym górnym rogu.

13.4.1 Uruchomienie `UserForma`

Najwygodniej zrobić to umieszczając przycik w arkuszu Excela i do zdarzenia `Click` przywiązując kod

```
UserForm1.Show
```

Jeśli chce się mieć dostęp do innych obiektów Excela w tym czasie (np. komórek), trzeba zmienić „modalność" `UserForma`:

```
UserForm1.Show vbModeless
```

Można także powiązać to otwarcie z innymi zdarzeniami:

- `WorkSheet_Activate` — otwarcie arkusza (przy przełączeniu z innego arkusza)
- `WorkBook_Open` — otwarcie pliku (skoroszytu).

Uwaga! Nie można wywołać `UserForma` z opcją `vbModeless` z innego `UserForma` bez tej opcji.
Uwaga! Opcja `vbModeless` powoduje, że dalsza część kodu zostanie wykonana natychmiat po wyświetleniu `UserForma`, bez czekania na reakcję użytkownika.

Komenda

```
Load NazwaUserForma
```

wywołuje dla tego `UserForma` zdarzenie `Initialize`, ale tylko w następujących sytuacjach:

- pierwsze użycie `UserForma` po otwarciu pliku
- pierwsze użycie `UserForma` po odwieszeniu Visual Basika
- pierwsze użycie po edycji właściwości `UserForma` z poziomu VBA

`Initialize` zostaje wywołane także przez:

- zdarzenie `QueryClose`, czyli np. komendę `Unload Me` i to bez względu na to, czy `UserForm` był wcześniej zamknięty, czy ukryty, czy nie.

Komenda

```
NazwaUserForma.Show
```

1. powoduje ustawienie właściwości `Visible = True`, zaś `.Hide` — odpowiednio — zmienia ją na `False`. **Nie ma** możliwości ustawienia tego przez

   ```
   NazwaUserForma.Visible = True
   ```

   Powyższa składnia działa tylko na kontrolki formularza, czyli np.

   ```
   Private Sub UserForm_Click()
   Me.CommandButton1.Visible = False
   End Sub
   ```

   spowoduje, że kliknięcie na formularz (nie na którąś z jego kontrolek, ani `x`) spowoduje, że przycisk `CommandButton1` przestanie być widoczny. Niemniej

   ```
   NazwaUserForma.Visible
   ```

   przechowuje informację o widzialności formularza.

2. wywołuje zdarzenie `Activate` dla formularza, jeśli

- nie użyto opcji `vbModeless` lub
- użyto jej, ale dalsza część kodu nie powoduje wyskoczenia kolejnych `UserForm`ów, `InputBox`ów, ani `MsgBox`ów.

Podobnie, jak `Visible`, własności `ShowModal` dla formularza nie można ustawić na `False`/`True` w kodzie, a jedynie we właściowościach. W kodzie ustawia się to wyłącznie przez

```
UserForm1.Show vbModeless
```

zaś `NazwaUserForma.ShowModal` przechowuje jako `True`/`False` informację o modalności formularza.

13.4.2 Zamknięcie `UserForma`

Czym innym jest ukrycie:

```
Me.Hide
```

kiedy wszystkie wartości kontrolek są zachowane (co widać po komendzie `UserForm1.Show`) i czym innym zamknięcie:

```
Unload Me
```

kiedy wszystkie nie zapisane dane zostają utracone.

Z zamknięciem `UserForma` powiązane są 2 zdarzenia następujące po kolei:

1. `QueryClose` oraz
2. `Terminate`

Ad 1. `QueryClose` pozwala powstrzymać zamkniecie przez zmianę parametru `Cancel` na jakąkolwiek wartość różną od zera, a także pozwala wykryć sposób, w jaki doszło do zamknięcia. Odpowiada za to `CloseMode`, które może przyjąć wartości

0 — zamknięcie komendą `Unload`
1 — zamknięcie przez kliknięcie „x” (ikona w prawym górnym rogu)
2 — zakończenie sesji Windowsa oraz
3 — zamknięcie przez menedżera zadań

Ad 2. `Terminate` to moment, kiedy zamknięcia formularza nie da się już powstrzymać i wszystkie jego neizapisane właściwości powracają do swoich domyślnych wartości.

Uwaga! `QueryClose` paradoksalnie wyołuje zdarzenie `Initialize`.
Uwaga! Przycisk umeiszczony na `UserForm`ie ma dostępną właściwość `Cancel`, która domyślnie jest ustawiona na `False`. Przełączenie jej na `True` **nie powoduje**, że przycisk ukryje czy zamknie formularz, a jedynie, że wciśnięcie Escape na klawiaturze zadziała jak kliknięcie tego przycisku. Nie może więcej niż jeden przycisk formularza mieć jednocześnie ustawionego `Cancel = True`.

Podobnie do `Unload NazwaForm` działa `Set NazwaForm = Nothing`, z tym, że nie wywołuje to zdarzeń `QueryClose` ani `Terminate`.

13.4.3 Przekazywanie danych między formularzami

Żeby dane (właściwości kontrolek) zostaly przekazane z jednego formularza można użyć jednej z następujących metod

Zmienne deklarowane w module
Żeby to zadziałało trzeba się upewnić, że wartości kontrolek zostaną przypisane do zmiennych zadeklarowanych w module (nie w formularzu!) wcześniej niż wywołane będzie zamknięcie pierwszego formularza.

Bezpośrednie przekazanie własności do kontrolek
Przeniesienie wartości może być pozwiązane ze zdarzeniem `Activate` nowego formularza, który musi być otwarty zanim stary zostanie zamknięty.

Użycie właściwości formularza: Tag
Przeniesienie wartości `Tag`→`Tag` przed otwarciem nowego formularza i zamknięciem starego, np.

```
UserFormNew.Tag = Me.Tag
UserFormNew.Show
Unload Me
```

lub

```
UserFormNew.Tag = Me.Tag
Unload Me
UserFormNew.Show
```

13.4.4 Zdarzenia myszki na przykładzie `CommandButton`

Zdarzenie `_Click` można „rozdzielić” na dwa zdarzenia:

1. `_MouseDown`, czyli wciśnięcie któregoś przycisku myszy
2. `_MouseUp`, czyli zwolnienie któregoś przycisku myszy

Zdarzenia te mogą rozróżniać

- który przycisk (`Button`) je wywołał
  - 1 lewy
  - 2 prawy
  - 4 środkowy
- niektóre klawisze klawiatury wciśnięte w czasie klikania (`Shift`)
  1. Shift
  2. Ctrl
  3. Shift i Ctrl
  4. Alt (lewy)
  5. lewy Alt i Shift
  6. lewy Alt i Ctrl lub prawy Alt, Alt gr (*graphical*)
  7. lewy Alt, Ctrl i Shift

13.4.5 `KeyDown`, `KeyPress` oraz `KeyUp`

Zdarzenia te następują kolejno po sobie, z pierwszym i ostatnim związany jest parametr `KeyCode` a z `KeyPress` — parametr `KeyAscii`. Lista wartości `KeyCode` dostępna jest tu http://www.vbforums.com/showthread.php?551538-keycode-list. Można te kody wykorzystać, by ograniczyć możliwość wpisywania pewnych znaków do `TextBox`a. W tym przykładzie została zablokowana możliwość wpisania litery A:

```
Private Sub TextBox2_KeyDown(ByVal KeyCode As MSForms.ReturnInteger, ByVal Shift As Integer)
If KeyCode = vbKeyA Then KeyCode = 0
End Sub
```

Zdarzenie `KeyDown` to początek wciskania przycisku. Zdarzenie `KeyPress` trwa do zwolnienia przycisku, czyli zdarzenia `KeyUp`. Zwróćmy uwagę, że `KeyCode` obejmują i rozróżniają praktycznie wszystkie przyciski z klawiatury, m.in. odróżniają cyfry z klawiatury numerycznej i tekstowej. Wartości parametru `KeyAscii` dostępne są tu: http://thecodeforyou.blogspot.com/2013/01/vb-keyascii-values.html

Ponadto, `Keycode` nie ma możliwości rozróżnienia wprost między małymi i dużymi literami — wciśnięcie Shift lub CapsLock to są oddzielne zdarzenia, podczas gdy `KeyAscii` rozróżnia te znaki bez problemu.

14 Czas

14.1 Formaty daty i czasu

Symbol	przykład	opis/uwagi
yy	18	ostatnie 2 cyfry roku
yyyy	2018	rok pełną liczbą
m	3	miesiąc bez zer
mm	03	miesiąc z zerami
mmm	Mar	miesiąc, 3 litery
mmmm	Marzec	miesiąc, pełnym słowem
d	9	dzień miesiąca bez zera
dd	09	dzień miesiąca z zerem
ddd	Wt	skrót dnia tygodnia
dddd	Wtorek	dzień tygodnia słownie
h	5	godzina bez zera do 24 lub 12
hh	05	godzina z zerem
m lub n	3	minuta bez zera
mm lub nn	13	minuta z zerem
s	4	sekunda bez zera
ss	04	sekunda z zerem
AM/PM	AM	przełącza w tryb 12-godzinny

Symboli m/mm w znaczeniu sekund można używać **tylko** po godzinie, inaczej, oznaczają miesiąc.

Przykład użycia:

```
Format(date_test, "m.d.yy h:nn AM/PM")
```

Inne symbole niż ., -, :, AM/PM trzeba poprzedzać backslashem.

14.2 Funkcje

Funkcja `Today()` zwraca aktualną datę, a `Now()` — datę i godzinę. W obu przypadkach jednostką jest dzień. Godzina odpowiada ułamkowi 1/24, minuta — $1/(24 \cdot 60)$ i sekunda — $1/(24 \cdot 3600)$. Dostępny jest też `Timer`, który zwraca aktualną liczbę sekund, które upłynęły od północy w typie `Single`, zatem z uwzględnieniem ułamków sekund.

14.3 PopUp

Wyskakujące okienko o funkcjonalności `MsgBox` to `PopUp`. Dodatkowo ustawia się czas przez jaki okienko ma być widoczne. Używa się go tak

```
Set costam = CreateObject("WScript.Shell")
wynik = costam.Popup("Your message text.", 5, "Your message box title")
Set x = Nothing
```

14.4 Ontime

Metoda `OnTime` ze składnią `Application.OnTime kiedy, "nazwa_makra"` powoduje, że program `nazwa_makra` uruchomi się o czasie wskazanym zmienną `kiedy`. Nazwa ta musi być w cudzysłowie. Nie powoduje to zawieszenia ani Excela, ani kodu pod `OnTime`, czyli np. kod

```
Application.OnTime Now + TimeValue("00:00:10"), "nazwa_makra"
MsgBox "Koniec!"
End Sub
```

powoduje, że po kolei:

1. zaczyna odliczać się czas
2. wyskakuje `MsgBox`
3. za (teraz już niecałe) 10 sekund uruchamia się `nazwa_makra`. Jeśli `MsgBox` został zamknięty później niż po dziesięciu sekundach, to `nazwa_makra` zaczeka w kolejce.

Można ustalić najpóźniejszy czas, po upływie którego program się nie wykona. Tzn. np. jeśli `MsgBox` nie zostanie zamknięty w ciągu minuty, to, żeby `nazwa_makra` się nie uruchamiała:

```
Application.OnTime Now + TimeValue("00:00:10"), "nazwa_makra", _
Now + TimeValue("00:01:10")
```

Jest jeszcze czwarty opcjonalny argument, który można ustawić na `False`, by wyzerować inne wcześniej zaplanowane przez `OnTime` akcje.

14.5 Wstrzymanie działania programu na jakiś czas

14.5.1 Wait

Instrukcja `Application.Wait Now + TimeSerial(0, 0, 1)` powoduje powieszenie `Excela` na sekundę. Zamiast `TimeSerial` można — jak wyżej — użyć `TimeValue`. Składnia w obu wariantach nie przewiduje ułamków sekund: będą one zaokrąglane w dół, do całej sekundy.

14.5.2 Timer

Blok instrukcji

```
Start = Timer
Do While Timer < Start + 0.5
    countsheep = countshhep + 1
    'DoEvents
Loop
```

powoduje półsekundową przerwę na liczenie owiec. Odkomentowanie `DoEvents` spowoduje, że procesor i Excel nie będą w tym czasie blokowane, zatem Excela można używać w tym czasie.

14.5.3 Sleep

Podobną funkcjonalność jak `Timer` w pętli, daje `Sleep`. Jest to makro, które jest w Windowsie przechowywane w bibilotece `kernel32`. Makra te, w odróżnieniu od np. `InputBox`, czy `MsgBox` nie są ładowane przy uruchamianiu Excela. Chęć ich użycia każdorazowo trzeba zadeklarować:

```
Declare Sub Nazwa Lib "kernel32" (opcje)
```

Jeśli chcemy dla tego makra używać innej nazwy, deklarujemy to tak

```
Declare Sub NowaNazwa Lib "kernel32" Alias "StaraNazwa" (opcje)
```

W naszym przypadku, dodatkowym problemem jest zegar: makro z 32-bitowego jądra próbujemy użyć w 64-bitowym systemie. Żeby to uzgodnić, trzeba zapewnić bezpieczne przejście, czyli dodać opcję `PtrSafe` (`Ptr` to skrót od *pointer*, czyli wskaźnik). Ponadto, deklaracja ta, musi znaleźć się w module, a możemy chcieć użyć `Sleep` w arkuszu lub w formularzu, zatem pełna deklaracja w module wyglądać będzie tak:

```
Public Declare PtrSafe Sub Sleep Lib "kernel32" (ByVal dwMilliseconds As LongPtr)
```

Jak widać z tej deklaracji, jednostką jest tu milisekunda. Przykład użycia w kodzie:

```
Sleep 333
```

15 Losowość

15.1 Rnd

Żeby otrzymać liczby pseudolosowe można odwołać się do odpowiednich funkcji Excela lub użyć wewnętrznej funkcji VBA, czyli `Rnd`. Komenda ta zwraca pseudolosową (czyli pochodzącą z deterministycznego algorytmu) liczbę z przedziału $[0, 1)$. Jeśli potrzebujemy liczby całkowitej z konkretnego przedziału, np. $\{1, 2, \ldots, 6\}$ wystarczy wartość `Rnd` pomnożyć, przesunąć i wziąć część całkowitą: `Int(6*Rnd+1)`.

Przy ponownym uruchomieniu Excela trafimy na ten sam ciag liczb pseudolosowych. Można temu zapobiec przenosząc się w bardziej losowe miejsce tego ciągu (wskazane aktualną wartością `Timera`) za pomocą komendy `Randomize` poprzedzającej `Rnd`.

15.2 Losowa permutacja

Tablicę zadeklarowaną jako

```
Dim arr(1 To 10)
```

można poddać losowej permutacji np. w pętli

```
For j = 1 To 10
    r = Int(10 * Rnd + 1)
    x = arr(j)
    arr(j)=arr(r)
    arr(r)=x
Next
```

w której każdy wyraz tablicy jest zamieniany z losowym.