Wykład 2 Wprowadzenie do IDE w Microsoft Office i składni języka VBA

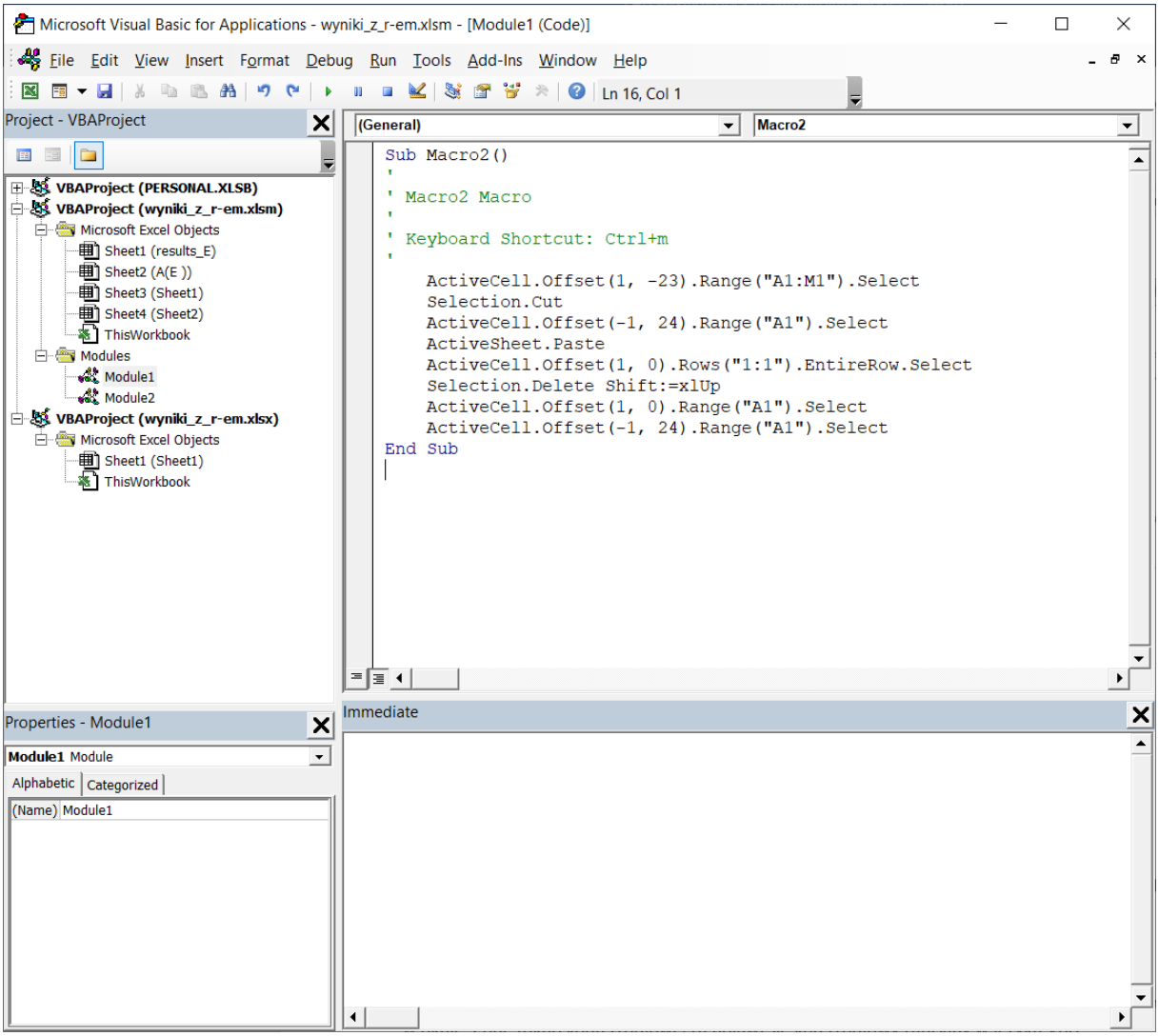
Zintegrowane środowisko programistyczne / edytor VBA (ang. IDE MS Office) – cz. 1

(ang. IDE - integrated development environment) – program lub zespół programów, czyli środowisko, przeznaczone do tworzenia (modyfikowania, testowania i konserwacji) oprogramowania

Dostęp do IDE:

1. Deweloper -> Visual Basic (w sekcji Kod)
2. Klawisz skrótu Alt+F11 (ponowne wciśnięcie Alt+F11 przenosi z powrotem do dokumentu głównego)

Przykładowy widok IDE w Excelu.



Inne (poza prezentowanymi na rysunku) okna – komponenty składowe IDE – dostępne są poprzez opcję **View** na pasku menu.

Okno Project Explorer

Okno Project Explorer zawiera strukturę bieżącego projektu przedstawioną w postaci drzewa.

Kod VBA zapisywany jest w modułach (module). Jeśli wcześniej nie stworzono żadnego skryptu VBA, to w oknie Project nie będzie elementu (folderu) o nazwie Modules. Jeśli w oknie Project są foldery o nazwie Modules, to wewnątrz nich znajdują się elementy domyślnie nazywane Module1, Module2, itp.

Okno Code

Aby zobaczyć, co zawierają poszczególne elementy z folderu Modules, należy klikać je dwukrotnie, a w oknie Code (okno kodu źródłowego) pojawi się kod źródłowy zapisany w języku VBA.

Edytor VBA poprawia czytelność kodu poprzez zastosowanie różnych kolorów czcionki.

- **kolor niebieski** – zarezerwowany dla słów kluczowych,

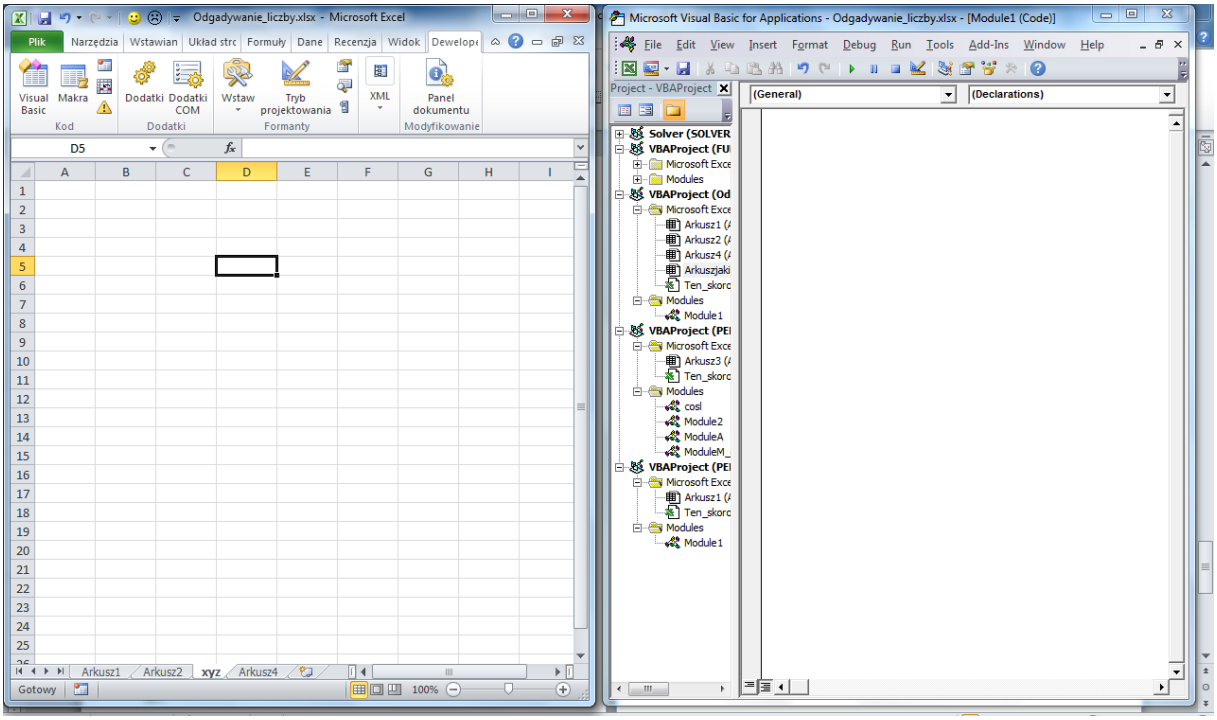
- **kolor zielony** – zarezerwowany dla komentarzy, rozpoczętych znakiem apostrofu,

- **kolor czerwony** – zarezerwowany dla błędów,

- **kolor czarny** – zarezerwowany dla reszty kodu.

Inne typy okien IDE

Pozostałe typy okien oferowane przez IDE VBA zostaną omówione później.

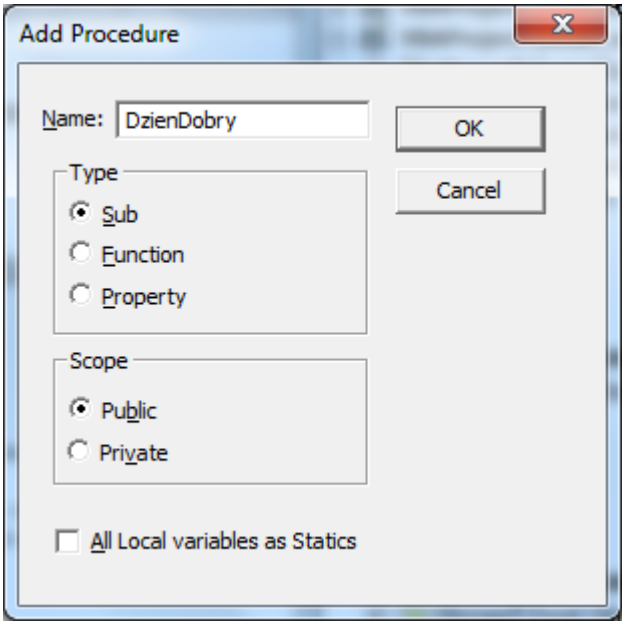


Nazewnictwo:

- **Makro** = makro poleceń = **skrypt** języka VBA = program = procedura;
- 2 typy makr:
  - Procedury (Sub) – programy (podprogramy) stanowiące ciąg instrukcji, nieposiadające wyniku zwracanego;
  - Funkcje użytkownika (Function) – programy (podprogramy) otrzymujące na wejściu zestaw wartości, które poprzez ciąg instrukcji są przemieniane w wartość wyjściową (wynik zwracany).

Przykład procedury – wypisywanie komunikatu typu „Hello world!”

1. Otwórz nowy skoroszyt Excela i edytor VBA. W oknie Projekt Explorer wskaż projekt, w którym zapiszesz procedurę. Na pasku menu wybierz Insert -> Module. Alternatywnie rozwiń menu podręczne klikając prawym przyciskiem myszy na wybranym projekcie. Kliknij 2 razy dodany moduł, aby wyświetlić okno kodu źródłowego.
2. Ustaw się kursorem w oknie kodu i wybierz z paska menu Insert -> Procedure..
3. W oknie Add Procedure wstaw nazwę procedury wg reguł obowiązujących dla nazw makr. (Patrz Wykład 1 -> Nagrywanie makr w Excelu)
4. Upewnij się, że typ tworzonego makra to procedura – w grupie Type zaznacz Sub. Kliknij OK.
5. W oknie kodu zobacz, jak wygląda wiersz nagłówka i zakończenia procedury.
6. Jako ciało procedury wstaw instrukcję: MsgBox „Dzień dobry Drogi Użytkowniku Excela :-)”.
7. Uruchom makro w edytorze VBA wybierając na pasku menu Run -> Run Sub/UserForm
8. Uruchom makro w skoroszycie Deweloper -> Makra -> Uruchom.



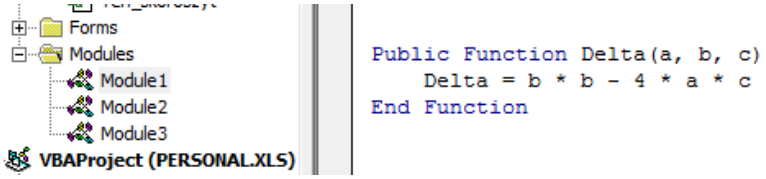
Przykład funkcji użytkownika – obliczanie wyróżnika (tzw. delty) trójmianu kwadratowego

Uwaga! Nagrywanie makr odnosi się tylko do procedur. Funkcję trzeba napisać!

1. Otwórz nowy skoroszyt Excela i edytor VBA. W oknie Projekt Explorer wskaż projekt, w którym zapiszesz funkcję. Na pasku menu wybierz Insert -> Module. Alternatywnie rozwiń menu

podręczne klikając prawym przyciskiem myszy na wybranym projekcie. Kliknij 2 razy dodany moduł, aby wyświetlić okno kodu źródłowego.

2. Ustaw się kursorem w oknie kodu i wybierz z paska menu Insert -> Procedure..
3. W oknie Add Procedure wstaw nazwę funkcji wg reguł obowiązujących dla nazw makr. (Patrz rozdział „Nagrywanie makr w Excelu”)
4. W grupie Type wskaż Function jako typ tworzonego makra i kliknij OK.
5. W oknie kodu źródłowego:
    a. Uzupełnij nagłówek funkcji o listę jej argumentów
    b. Wstaw instrukcję zwracania wyniku w postaci NazwaFunkcji = wartość.
    c. W efekcie powinieneś otrzymać:



6. Przejdź do otwartego skoroszytu. W wybranej komórce zacznij wpisywać wyrażenie „=Del…” i zobacz, że program „widzi” już funkcję Delta. Alternatywnie możesz wybrać zdefiniowaną funkcję stosując kreator wstawiania funkcji i wyszukując ją w kategorii „Zdefiniowane przez użytkownika”.
7. Wprowadź do arkusza potrzebne dane i użyj ich jako argumentów funkcji Delta.
8. Sprawdź poprawność działania funkcji.

Operatory i typy zmienych

https://docs.microsoft.com/en-us/office/vba/language/reference/user-interface-help/operator-summary

OPERATORY

1. Operatory arytmetyczne, konkatenacji, porównania

Operatory	Działanie	Priorytet w VBA	Priorytet w Excelu
^	potęgowanie	1	2
-	wzięcie liczby przeciwnej (operator jednoargumentowy)	2	1
*	mnożenie	2	3
/	dzielenie	2	3
\	dzielenie całkowite, część całkowita z dzielenia liczb całkowitych	2	3
Mod	dzielenie modulo, reszta z dzielenia	2	3
+	dodawanie liczb konkatenacja stringów	3	4
-	odejmowanie (operator dwuargumentowy)	3	4
&	konkatenacja	4	
=, <, >, <=, >=, <>	Porównanie liczb, stringów	5	6
Like	Porównanie stringów		
Is	Porównanie obiektów		

Uwaga 1. W arkuszu operator wzięcia liczby przeciwnej ma najwyższy priorytet. Jak widać powyżej w VBA jego priorytet jest niższy. Dlatego
W arkuszu formuła =-3^2 zwróci wartość 9, bo jest liczona jako (-3)^2
W VBA formuła =-3^2 zwróci wartość -9, bo jest liczona jako –(3^2).

Uwaga 2. Argumenty operatorów dzielenia całkowitego „\" i reszty z dzielenia całkowitego „Mod" mogą być liczbami niecałkowitymi lecz w takim przypadku liczby te najpierw są zaokrąglane do całkowitych a dopiero w drugim kroku następuje wykonanie odpowiednio dzielenia całkowitego lub reszty z dzielenia całkowitego. Zasady zaokrąglania liczb do liczb całkowitych dla tych dwóch operatorów są identyczne (i trochę niekonsekwentne):

dzielna – jest zaokrąglana w dół, gdy część dziesiętna jest z przedziału [0; 0.5] a w górę, gdy część dziesiętna jest z przedziału (0.5; 1)

dzielnik – jest zaokrąglany w dół gdy część dziesiętna jest z przedziału [0; 0.5) a w górę, gdy część dziesiętna jest z przedziału [0.5; 1)

Np. 4.5 \ 1.5 = 4 \ 2 = 2;
4.51 \ 1.49 = 5 \ 1 = 5;
4.5 Mod 1.5 = 4 Mod 2 = 0
4.51 Mod 1.5 = 5 Mod 2 = 1.

Uwaga 3. Operator porównania stringów Like dopuszcza użycie symboli wieloznacznych – patrz: https://docs.microsoft.com/en-us/office/vba/language/reference/user-interface-help/like-operator

2. Operatory logiczne

Operatory	Działanie
Not	negacja logiczna
And	koniunkcja logiczna
~~AndAlso~~	~~Koniunkcja warunkowa~~ (jest tylko w VB)
Or	alternatywa logiczna
~~OrElse~~	~~Alternatywa warunkowa~~ (jest tylko w VB)
Xor	dysjunkcja logiczna
Eqv	równoważność logiczna
Imp	implikacja logiczna

TYPY ZMIENNYCH

typ danych	**liczba użytych bajtów**	**zakres wartości**
Byte	1B	od 0 do 255
Boolean	2B	TRUE, FALSE
Integer	2B	od -32 768 do 32 767
Long	4B	od -2 147 483 648 do 2 147 483 647
Single	4B	od -3,402823E38 do -1,401298E-45 od 1,401298E-45 do 3,402823E38
Double	8B	od -1,7977E308 do -4,9407E-324 od 4,9407E-324 do 1,7977E308
Currency	8B	od -922 337 203 685 477,5808 do 922 337 203 685 477,5807
Decimal	12B	bez cz. dziesiętnej: od ok. -79 288 162 E21 do ok. 79 288 162 E21 z 28 miejscami po przecinku: od ok. -7,9288162 do ok. 7,9288162
Date	8B	od 1 sty 100 r. do 31 gru 9999 r.
Object	4B	dowolne odwołanie do obiektu
String (zmienna długość)	10B + dł. stringa	od 0 do ok. 2 mld znaków
String (stała długość)	długość stringa	od 1 do ok. 65 400 znaków
Variant (z liczbami)	16B	dowolna wartość numeryczna aż do maksymalnej wartości typu Double; dodatkowo może przechowywać wartości typu Empty, Error, Nothing, Null
Variant (ze znakami)	22B + dł. stringa	od 0 od ok. 2 mld znaków
Zdefiniowany przez użytkownika	zmienna	zmienia się, zależnie od elementu

Uwaga! Typu Decimal nie można zadeklarować. Jest to podtyp typu Variant. Można zmienić typ Variant na typ Decimal funkcją *CDec*.

Najnowsza wersja języka VBA zawiera dodatkowe typy takie jak: LongLong (z zakresem od -9 223 372 036 854 775 808 do 9 223 372 036 854 775 807), LongPtr (LongPtr = Long na komputerach z 32b architekturą oraz LongPtr = LongLong na komputerach z 64b architekturą).

Rodzaje zmiennych, deklarowanie zmiennych i definiowanie typów danych

W VBA wyróżniamy zmienne **globalne** i **lokalne**. Rozróżnienie na zmienne globalne i lokalne jest związane z ich zasięgiem (widocznością, dostępnością). Jest ono następujące: zmienne globalne są dostępne (widoczne) we wszystkich modułach; zmienne lokalne to te, które są dostępne w ramach pojedynczych makr lub pojedynczych modułów.

Uwaga! Język VBA nie rozróżnia wielkości liter. Oczywiście jako nazw zmiennych nie można używać słów kluczowych (kolor niebieski w edytorze VBA). Reguły tworzenia poprawnych nazw zmiennych są takie same jak dla nazw makr (porównaj wykład 1).

Słowa kluczowe podczas **deklaracji zmiennych lokalnych** to „Dim" oraz „As". Deklarowanie takich zmiennych ma postać:

Dim *zmienna* As *Typ_zmiennej*

```
Np.:  Dim dzisiaj As Date          'deklaracja zmiennej typu data
      Dim MojString As String *50  'deklaracja zmiennej łańcuchowej o stałej długości max 50
                                   'znaków
```

Słowa kluczowe podczas **deklaracji zmiennych globalnych** to „Public" oraz „As". Deklarowanie takich zmiennych ma postać:

Public *zmienna* As *Typ_zmiennej* .

Sposoby deklarowania zmiennych różnych typów oraz ich zasięg zebrano w poniższej tabeli.

Rodzaj zmiennej	zasięg	sposób deklaracji zmiennej
globalna	wszystkie moduły projektu	przed pierwszym makrem modułu z użyciem słowa Public
lokalna	pojedynczy moduł	przed pierwszym makrem modułu z użyciem słowa Dim lub Private
lokalna	pojedyncze makro	wewnątrz makra z użyciem słowa Dim lub Static

Należy tu wspomnieć jeszcze o możliwości **deklarowania zmiennych obiektowych** jednak więcej o tych zmiennych zostanie przedstawione w jednym z późniejszych wykładów.

Język VBA nie wymaga jawnego definiowania typów danych, ale z uwagi na efektywność docelowego makra jest to zalecane. Niezadeklarowanie typu danych dla zmiennej powoduje, że VBA użyje domyślnego typu: Variant. Dane tego typu zachowują się trochę jak kameleon – zmieniają swój typ w zależności od tego, w jaki sposób są przetwarzane. Oto demonstracja takiej sytuacji:

```
Sub VariantDemo()
    MojaZmienna = ”123”
    MojaZmienna = MojaZmienna / 2
    MojaZmienna = ”Wynik = ” & MojaZmienna
    MsgBox MojaZmienna
End Sub
```

Możliwe jest deklarowanie kilku zmiennych w jednej linii. Służy do tego separator przecinka. Np.:

```
Dim x As Long, y As Long
Dim A As Double, B As Double
Dim d As Date, e As Date
```

Uwaga! Deklaracja postaci

```
Dim x, y As Long
```

jest poprawna, ale oznacza, że tylko zmienna y pozostała zadeklarowana jako Long. Zmienna x jest typu Variant.

Inny sposób określania typów zmiennych – pozostałość po języku Basic

typ danych	znak deklarujący
Integer	%
Long	&
Single	!
Double	#
Currency	@
String	$
LongLong	^

Przykładowe użycie:

```
Dim MojaZmienna%
```

Jeśli chcemy wymusić na sobie jawne definiowanie typów wtedy przed (nad) makrem należy wpisać polecenie: **Option Explicit**. Interpreter podczas wykonywania makra z takim poleceniem, po napotkaniu zmiennej bez jawnie zdefiniowanego typu wyświetli okno z informacją o błędzie: „Compile error: Variable not defined”. Wymuszanie jawnego definiowania typów pozwala uniknąć trudnych do zidentyfikowania błędów. Np. jeśli spodziewaliśmy się, że po wykonaniu kodu

```
Zmienna1 = 3.14
Zmienna2 = 2.77
Znienna1 = Zmienna1 + Zmienna2
```

końcowa wartość zmiennej Zmienna1 jest sumą jej wartości początkowej i wartości zmiennej Zmienna2, to przeżyjemy rozczarowanie, bo jej wartość będzie taka jak jej wartość początkowa. A wszystko dlatego, że w instrukcji przypisania Znienna1 = Zmienna1 + Zmienna2 został popełniony błąd literowy i suma zmiennych Zmienna1 oraz Zmienna2 została przypisana trzeciej zmiennej Znienna1.

Inicjalizacja zmiennych:

```
Dim wiek as Byte
Dim flaga As Boolean
Dim x As Single, y As Single
Dim dzień_urodzin As Date
Dim zachod_slonca As Date
Dim tytul As String
```

```
wiek = 0
flaga = False
x = 3.14
y = 3.0E-3
dzień_urodzin = #3/7/1981#      ‘data ma format: miesiąc / dzień / rok
zachod_slonca = #17:31:05#      ‘sekundy są opcjonalne
tytul = ”Programowanie w VBA”
```

Uwaga! Przedstawiony powyżej zapis daty i godziny jest jedynym zalecanym w VBA (inne są możliwe lecz niezalecane). Nie jest to jednak tożsame ze sposobem wypisywania tych wartości. Np. jeśli do wyświetlenia daty zostanie użyte okno komunikatu, to data zostanie pokazana zgodnie z systemowym formatem krótkiej daty. Podobnie czas jest wyświetlany z uwzględnieniem systemowego formatu czasu (12- godzinny lub 24-godzinny). W celu zmiany sposobu wyświetlania daty i czasu należy dokonać zmian systemowych w panelu sterowania / opcje regionalne i językowe (lub podobnie, w zależności od wersji systemu operacyjnego).

Stałe w VBA

Stała to szczególny rodzaj zmiennej, która w trakcie swojego życia nie zmienia raz przypisanej wartości. Deklaracja stałych ma postać:

Const *stała* As *Typ_stałej*

Inicjalizacja stałej wartością może wystąpić w tej samej linii co deklaracja stałej, np.:

Const drugi_kwartal as Integer = 2

Const oprocentowanie = 0.005

Podobnie jak dla zmiennych nie ma potrzeby jawnego definiowania typu. Również podobnie jak dla zmiennych, stałe posiadają swój zasięg.

rodzaj stałej	zasięg	sposób deklaracji stałej
globalna	wszystkie moduły projektu	przed pierwszym makrem modułu z użyciem słów Public Const np. Public Const oprocentowanie As Double = 0.015
lokalna	pojedynczy moduł	przed pierwszym makrem modułu z użyciem słowa Const
lokalna	pojedyncze makro	wewnątrz makra z użyciem słowa Const

Excel i VBA oferują wiele stałych predefiniowanych. Można ich używać bez żadnej deklaracji. O ich istnieniu łatwo się przekonać w trakcie pisania kodu. Domyślne lub dopuszczalne wartości właściwości obiektów są zwykle zdefiniowane jako stałe i ich nazwy są wyświetlane przez inteligentne podpowiedzi.

Zmienne tablicowe

Tablica to grupa elementów tego samego typu, posiadająca wspólną nazwę.

1. Deklarowanie tablic statycznych

```
Dim miesiace(1 To 12) As String *20
Dim cyfry(0 To 9) As Byte
Dim CyfryKrocej(9) As Byte      'domyślna wartość pierwszego indeksu to 0, dlatego podczas
                                'deklaracji można go pominąć
Dim TabliczkaMnozenia(1 To 10, 1 To 10) As Byte
Dim MojaTablica(9, 19, 99) As Integer   'deklaracja tablicy o rozmiarze 10 x 20 x 100
```

Uwaga! Można zmienić domyślne ustawienie numerowania elementów tablic nie od 0 lecz od 1. W tym celu przed pierwszym makrem modułu należy wpisać:

Option Base *liczba*

Np. Option Base 1 spowoduje, że numeracja elementów tablic będzie się zaczynać od 1 i deklaracja Dim miesiace(1 To 12) As String będzie równoznaczna z Dim Miesiace (12) As String.

Funkcje **LBound**(nazwa_tablicy, **UBound**(nazwa_tablicy) zwracają indeks pierwszego i ostatniego elementu tablicy.

Dostęp do poszczególnych pól tablic uzyskujemy poprzez podanie nazwy tablicy i współrzędnych pola, np.:

```
Miesiące(1) = „Styczeń”
TabliczkaMnozenia (2,3) = 6
```

2. Tworzenie tablic

Do utworzenia tablicy użyteczne jest polecenie **Array**.

```
Dim imiona(1 To 10) As String
Imiona = Array(“Ala”, “Ela”, “Ula”, “Iza”, “Ala”, “Ewa”, “Anna”)
```

3. Tablice dynamiczne

Tablica dynamiczna nie posiada z góry określonej liczby elementów. Jest deklarowana przy użyciu pary pustych nawiasów, np.:

```
Dim MojaTablica() As Boolean
```

Jednak przed pierwszym użyciem tablicy dynamicznej w programie, konieczne jest użycie instrukcji ReDim, która informuje o liczbie elementów tablicy. Można do tego celu użyć zmiennej, od wartości której uzależniony jest rozmiar tablicy. Np.:

```
ReDim MojaTablica(1 To n).
```

Polecenia ReDim można użyć dowolną liczbę razy, każdorazowo zmieniając rozmiar tablicy. Jednak przy każdej tego typu operacji zawartość tablicy jest tracona. Jeśli chcemy zachować istniejące w tablicy wartości (a tylko ją powiększyć lub pomniejszyć), wtedy należy posłużyć się wyrażeniem

ReDim Preserve MojaTablica(1 To k)

Typy danych zdefiniowane przez użytkownika (materiał nieobligatoryjny)

W języku VBA istnieje możliwość zdefiniowania przez użytkownika własnego (niestandardowego) typu danych. **Typ danych zdefiniowany przez użytkownika** może znacząco ułatwić mu pracę z niektórymi rodzajami danych. Dla przykładu przyjmijmy, że sprzedawca przechowuje następując informacje o klientach: nazwa firmy / imię i nazwisko, kod pocztowy, data transakcji, kwota transakcji, łączna kwota transakcji. Zdefiniowanie typu danych obejmującego 5 ww. zmiennych będzie wyglądać następująco:

```
Type InfoOKlientach
    Nazwa As String
    Kod As String*6
    DataPierwszejTransakcji As Date
    DataOstatniejTransakcji As Date
    Razem As Currency
End Type
```

Uwaga! Niestandardowe typy danych muszą być definiowane na początku modułu, przed kodem pierwszego makra.

Utworzenie zmiennej niestandardowego typu jest klasyczne. Dla powyższego przykładu może wyglądać następująco:

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

Zmienna Klienci to tablica o 100 polach, z których każde składa się z pięciu elementów: Nazwa, Kod, DataTransakcji, Kwota, Razem. Inicjalizacja i odwoływanie się do poszczególnych pól i ich elementów może wyglądać następująco:

```
Klienci(1).Nazwa = ”Andrzej Abacki”
Klienci(1).Kod = „35-310”
Klienci.(1).DataPierwszejTransakcji = #11/4/2020#
Klienci.(1).DataOstatniejTransakcji = #11/4/2020#
Klienci(1).Razem = 568.12
```

Każdy element tablicy Klienci może być przetwarzany jako całość, np. Klienci(2) = Klienci(1).

1. Instrukcja warunkowa If Then

W języku VBA istnieje kilka odmian tej instrukcji. Oto one:

++++++++++++++++++++

If *warunek* **Then** *instrukcja*

++++++++++++++++++++

```
If warunek Then
    Instrukcja1
    Instrukcja2
End If
```

++++++++++++++++++++

```
If warunek Then
    Instrukcja1
    Instrukcja2
Else
    Instrukcja3
    Instrukcja4
End If
```

++++++++++++++++++++++

```
If warunek1 Then
    Instrukcja1
    Instrukcja2
ElseIf warunek2 Then    'można też pisać klasycznie: Else If
    Instrukcja3
    Instrukcja4
Else
    Instrukcja5
End If
```

++++++++++++++++++++++

Uwaga! W języku VBA istnieje funkcja **IIf**, działająca jak arkuszowa funkcja JEŻELI, czyli mająca składnię IIf(warunek, co_jeśli_prawda, co_jeśli_fałsz). Funkcję tę można stosować wymiennie z instrukcją warunkową If Then Else jeśli po słowach Then oraz Else występują pojedyncze instrukcje. Przebieg działania funkcji IIf jest nieco inny niż instrukcji If Then Else. Dla funkcji IIf kompilator wykonuje zarówno polecenie co_jeśli_prawda jak i polecenie co_jeśli_fałsz, lecz zwraca wartość tylko jednego z

nich, zależnie od wartości logicznej warunku. W przypadku instrukcji warunkowej instrukcje następujące po słowie Else nie są wykonywane gdy warunek jest prawdziwy.

2. **Instrukcja warunkowa Select Case**

Instrukcja ta zastępuje wielokrotne zagnieżdżenie instrukcji If Then Else. Jej składnia jest następująca:

Select Case zmienna lub wyrażenie
    Case zakres 1 wartości zmiennej lub wyrażenia
        Instrukcja1
        Instrukcja2
    Case zakres 2 wartości zmiennej lub wyrażenia
        Instrukcja3
    Case Else
        Instrukcja4
End Select

Jeżeli w jakichś przypadkach wykonywana jest tylko jedna instrukcja, to można ją pisać w tej samej linii co Case zakres wartości zmiennej lub wyrażenia, lecz poprzedzoną dwukropkiem, np.

Select Case zmienna lub wyrażenie
    Case zakres 1 wartości zmiennej lub wyrażenia: Instrukcja1
    Case zakres 2 wartości zmiennej lub wyrażenia: Instrukcja2
    Case Else Instrukcja3
End Select

Instrukcje Select Case mogą być zagnieżdżane dowolną liczbę razy. Należy przy tym pamiętać, by każdy Select Case kończył się End Select.

Podczas używania instrukcji Select Case często przydatne są operatory **To** oraz **Is**.

Przykład 1. nauczyciel ustanowił następujący sposób przeliczenia zdobytych punktów na ocenę:

[0%; 50%) – ndst; [50%; 60%) – dost; [60%; 70%) – plus dost; [70%; 80%) – db
[80%; 90%) – plus db; [90%; 100%] – bdb.
Dla zaimplementowania tej sytuacji za pomocą instrukcji Select Case należy użyć operatora **Is** w następujący sposób:

Select Case punkty
    Case Is < 50: MsgBox „ndst”
    Case Is < 60: MsgBox „dost”
    Case Is < 70: MsgBox “plus dost”
    Case Is < 80: MsgBox “db”
    Case Is < 90: MsgBox “plus db”
    Case Else: MsgBox “bdb”
End Select

Przykład 2. Podziału studentów na grupy dokonano ze względu na ich kolejność alfabetyczną. Dla zaimplementowania tej sytuacji za pomocą instrukcji Select Case użyto operatora **To** w następujący sposób:

```
Select Case poczatek_nazwiska
    Case "A" To "D": MsgBox "GR 1"
    Case "E" To "J": MsgBox "GR 2"
    Case "K" To "P": MsgBox "GR 3"
    Case Else: MsgBox "GR 4"
  End Select
```

Przykład 3. Użyj instrukcji Select Case do prawidłowego przypisania wybranych czworokątów do odpowiedniej rodziny figur.

```
Function TypCzworokata (ile_par_bokow_rownoleglych As Byte, ile_katow_prostych As Byte)
      Select Case TRUE
        Case ile_par_bokow_rownoleglych = 2 AND ile_katow_prostych = 4
              TypCzworokata = „prostokat”
        Case ile_par_bokow_rownoleglych = 2 AND ile_katow_prostych = 0
              TypCzworokata = „rownoleglobok”
        Case ile_par_bokow_rownoleglych = 1 AND ile_katow_prostych = 2
              TypCzworokata = „trapez prostokatny”
        Case ile_par_bokow_rownoleglych = 1 AND ile_katow_prostych = 0
              TypCzworokata = „trapez”
        Case Else
              TypCzworokata = „czworokat nieregularny”
     End Select
End Function
```

3. Instrukcja skoku GoTo

Instrukcja GoTo *etykieta* przenosi wykonywanie programu do linii kodu, która została oznaczona etykietą. Etykieta to łańcuch tekstowy zakończony dwukropkiem lub liczba bez dwukropka. Makro może zawierać dowolną liczbę unikalnych etykiet. Instrukcja GoTo działa tylko w obrębie danego makra (tj. nie ma możliwości przenieść wykonywania programu do innego makra).

4. Polecenie Exit

Polecenie Exit *instrukcja* przerywa wykonywanie pętli (Exit For, Exit Do – patrz niżej), w której zostało umieszczone i przenosi wykonywanie programu do wiersza kodu pod instrukcją Next dotyczącą aktualnej pętli. Umieszczone w pętli wewnętrznej przerywa działanie tylko tej pętli. Polecenie Exit może być stosowane także do przerwania działania makra (Exit Function, Exit Sub).

5. Pętla For Next

Składnia tej pętli jest następująca:

For *licznik* = wartość_początkowa **To** wartość_końcowa [**Step** wartość_kroku]
Instrukcja 1
Instrukcja2
[Exit For]
Instrukcja3
Next [licznik]

W pętli For Next dopuszczalna jest modyfikacja wartości zmiennej *licznik* wewnątrz pętli. Liczba wartość_kroku może być ujemna.

6. Pętle Do Loop

Pętle Do-Loop w VBA występują w czterech wariantach:
- z warunkiem określającym powtarzanie / przerwanie powtarzania pętli przed blokiem instrukcji (czyli na początku pętli) lub na końcu,
- z warunkiem while bądź z warunkiem until.

a) While

Jest wykonywana tak długo, jak długo **warunek jest prawdziwy**. Przerwanie działania pętli można osiągnąć przez zastosowanie polecenia Exit Do.

++++++++++++++++++++++++

Do [While warunek]
Instrukcje
Loop
++++++++++++++++++++++++

Do
Instrukcje
Loop [**While** warunek]
++++++++++++++++++++++++

b) Until

Ta pętla jest bardzo podobna w swoim działaniu i składni do poprzedniej. Również może występować w dwóch wariantach. Jednak jest ona wykonywana tak długo, jak długo **warunek jest fałszywy**. Przerwanie działania pętli można osiągnąć przez zastosowanie polecenia Exit Do.

++++++++++++++++++++++++

Do [Until warunek]
Instrukcje
Loop
++++++++++++++++++++++++

Do

Instrukcje

Loop [**Until** warunek]

++++++++++++++++++++++++

Uwaga1! Z uwagi na chęć zachowania kompatybilności VBA ze starszymi wersjami możliwe jest stosowanie jeszcze pętli While Wend. Jej działanie pokrywa się z działaniem pętli Do While w wersji pierwszej:

While warunek

Instrukcje

Wend

Uwaga2! Oprócz powyższych pętli w języku VBA istnieje jeszcze pętla For Each przeznaczona do wykonywania operacji na kolekcjach obiektów. Zostanie ona omówiona na wykładzie w kontekście obiektów.