Krótki kurs VBA Excel dla inżynierów

Visual Basic for Applications (VBA) to język programowania oparty na Visual Basic (VB) zaimplementowany w aplikacjach MS Office i kilku innych (np. AutoCAD). Ta uproszczona wersja VB służy przede wszystkim do automatyzacji pracy z dokumentami.

Główną różnicą między VBA i VB jest to, że VBA nie pozwala na tworzenie niezależnych aplikacji EXE. Kod programu napisany w VBA jest zawsze zawarty w dokumencie (np. XLS) utworzonym przez program tzw. *hosta* (np. Excel) i wymaga do uruchomienia środowiska wykonawczego, którym jest aplikacja hosta obsługująca dokument. VBA jest dodatkiem do hosta i jest instalowane wraz z nimi. VBA pomaga tworzyć niestandardowe aplikacje i rozwiązania w celu zwiększenia możliwości aplikacji hosta.

Zaletą używania VBA w Excelu jest to, że za jego pomocą można budować bardzo potężne narzędzia przy użyciu programowania liniowego. Mimo że Excel zapewnia szerokie spektrum wbudowanych funkcji, praktycznie większość z nich jest zorientowana na biznes, a wiele funkcji odpowiednich do celów inżynieryjnych jest po prostu niedostępnych. Dlatego muszą być tworzone przez użytkownika w postaci często złożonych formuł Excela, które z reguły są nieczytelne i niewygodne w użyciu. W takim przypadku używanie VBA do tworzenia funkcji zdefiniowanych przez użytkownika (UDF) staje się najbardziej oczywistym rozwiązaniem.

1. Uruchom Excel
2. Otwórz lub utwórz nowy dokument (skoroszyt)
3. Zapisz go jako plik typu XLSM (Skoroszyt Excel z obsługa makr)
4. … i wciśnij

ALT - F11

… aby wywołać VBE

UWAGA.
Każde wciśnięcie **ALT - F11** przełącza między edytorem VBE a skoroszytem.

Utworzenie nowego modułu

(1) Wywołaj menu kursora - kliknij prawym przyciskiem myszy w panelu

Project - VBAProject

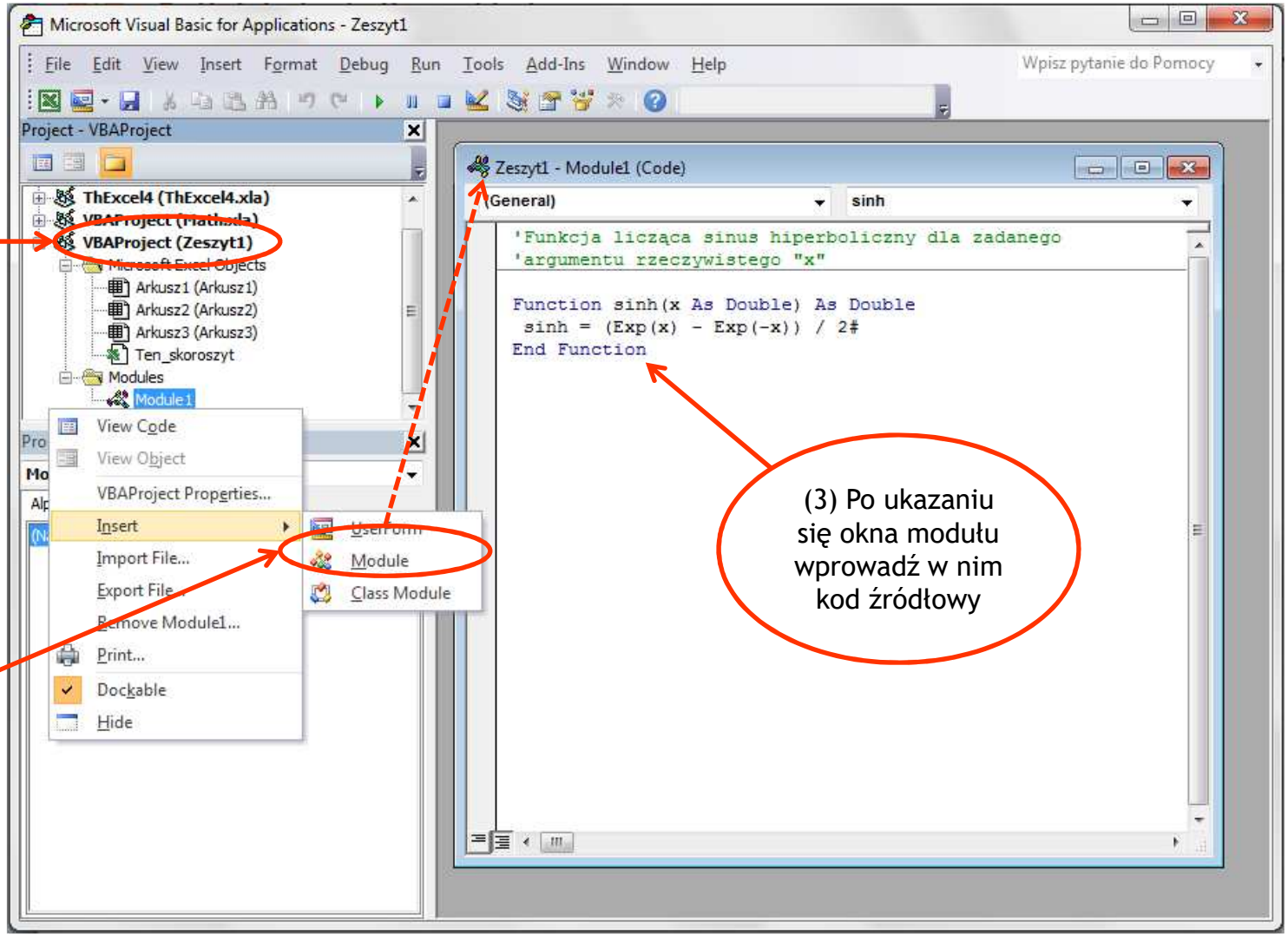
nad sekcją

VBAProject (…)

(2) wybierz pozycję

Insert → Module

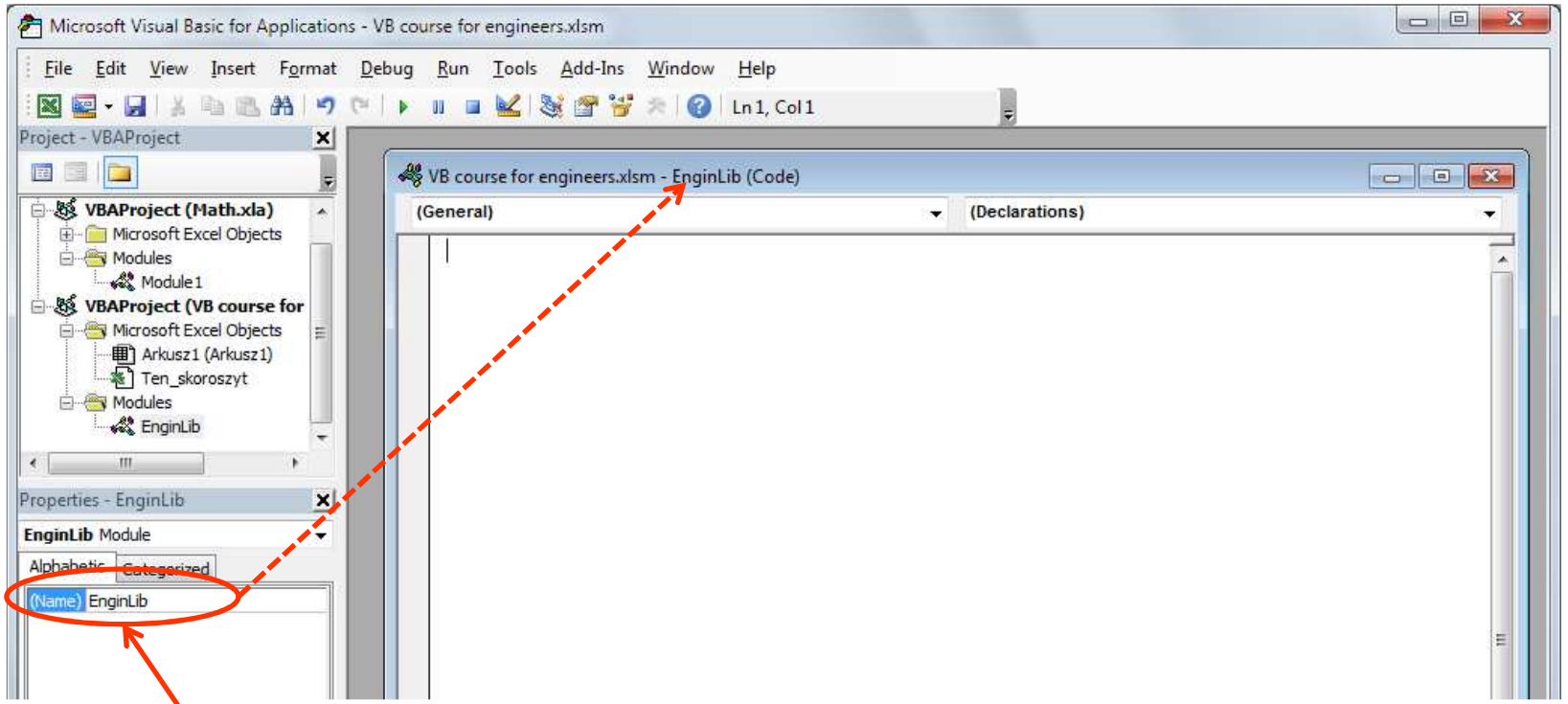
co spowoduje otwarcie okna modułu, w którym wprowadzisz kod.



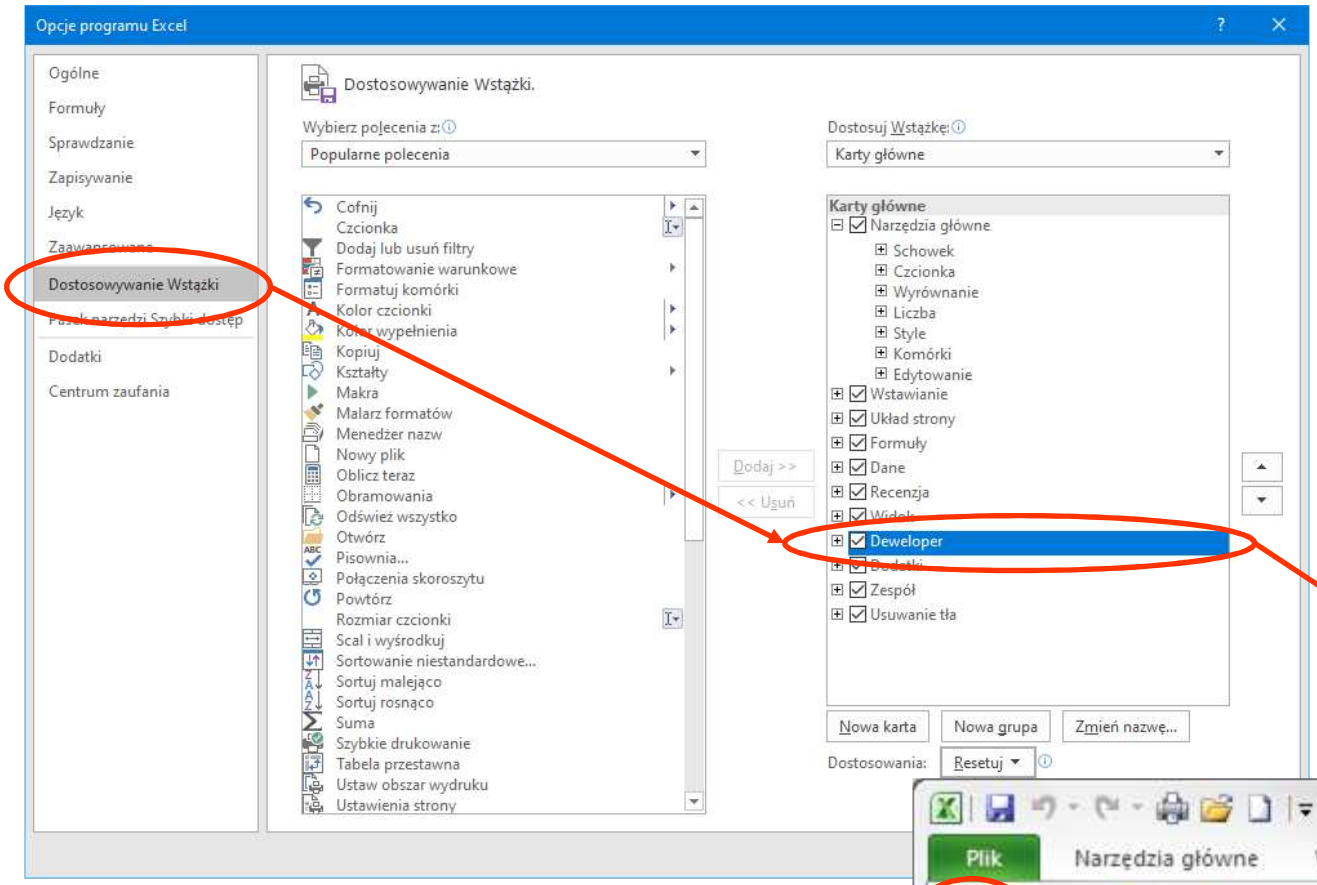
(3) Po ukazaniu się okna modułu wprowadź w nim kod źródłowy

Tools → Options zakładka **Edit**, odznacz opcje
[ ] **Auto syntax check**,
aby uniknąć ciągłego nękania komunikatami o błędzie.

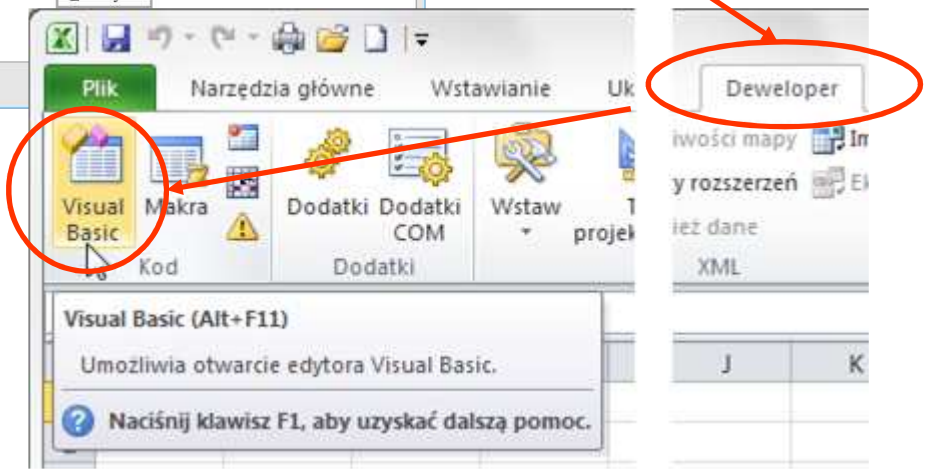
Nadanie nazwy nowemu modułowi



W panelu **Properties** zmień nazwę modułu. Powinna być krótka i samokomentująca



Plik → Opcje → Dostosowanie wstążki (*uaktywnić Deweloper*)



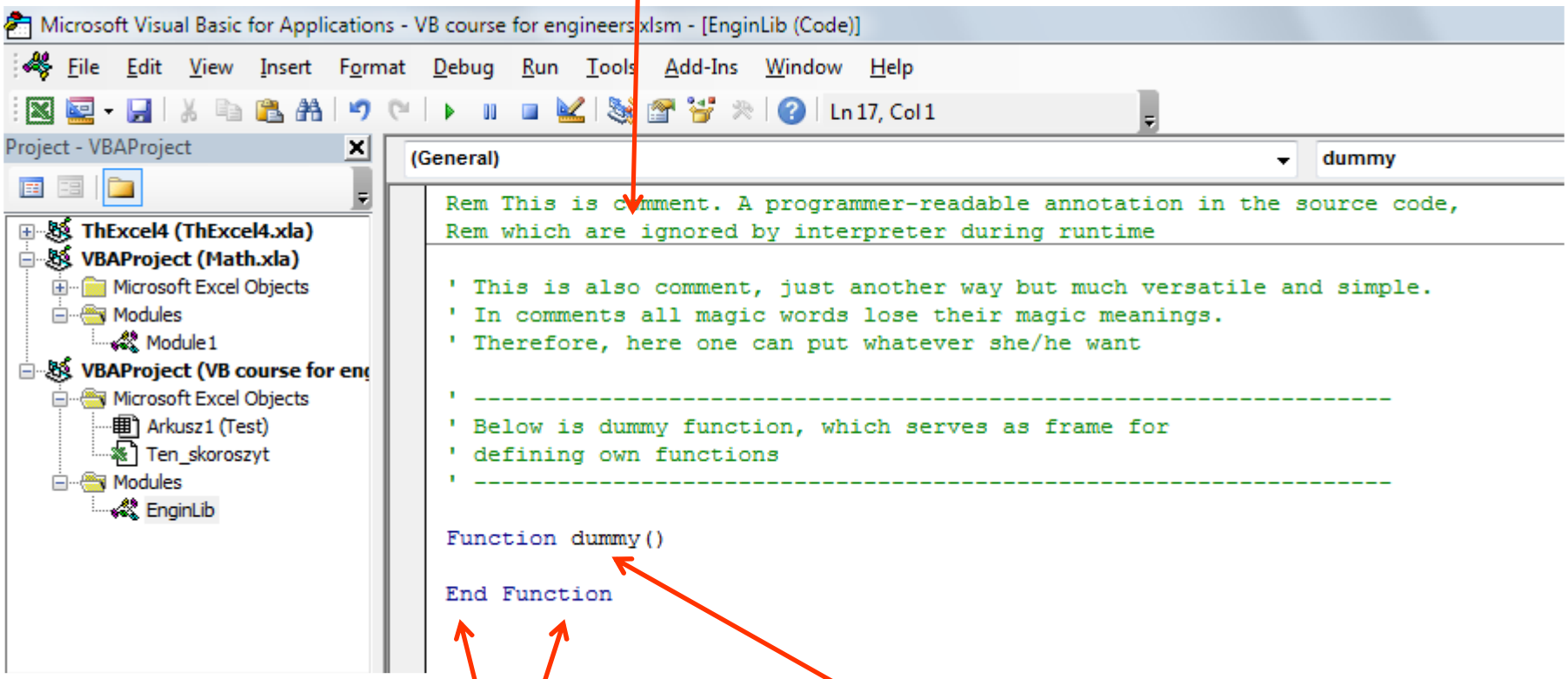
Visual Basic można uruchomić skrótem klawiszowym

ALT - F11

Bądź z odpowiedniej pozycji menu lub wstążki Deweloper

Pierwsza funkcja szkielet

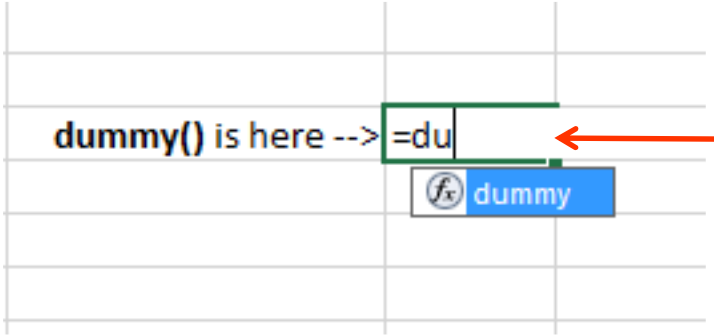
Tu jest komentarz



Identyfikator = Nazwa

Słowa kluczowe

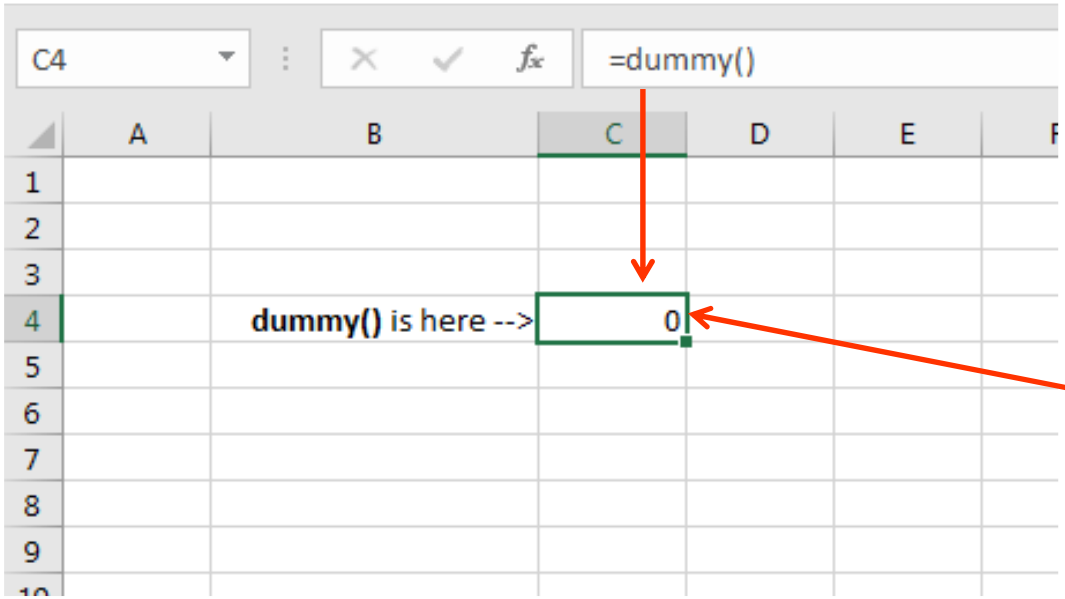
Użycie nowo zdefiniowanej funkcji formule Excela



W komórce zacznij pisać nazwę funkcji

„=dummy”

a Excel wyświetli listę dostępnych funkcji, wśród których można znaleźć tą zdefiniowaną. Kursorem wybierz właściwą i wciśnij **TAB**, aby wprowadzić nazwę do pola formuły



Po wciśnięciu klawisza Enter zobaczysz w komórce wynik zwracany przez nową funkcję.

Zestaw znaków (alfabet) używanych do tworzenia słów w VBA:

```
a b c d e f g h i j k l m n o p q r s t u v w x y z
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
0 1 2 3 4 5 6 7 8 9
_ . , : ' = < > + - * / \ ^ ( ) [ ]% # $ @ ! & " ;
```

Komentarz: adnotacja czytelna dla programisty w kodzie źródłowym, która jest ignorowana przez interpretera podczas działania. W VB komentarze są wprowadzane słowem kluczowym REM lub apostrofem i rozciągają się aż do końca wiersza. Przykłady:

```
REM To jest komentarz. Tu tracą magiczne znaczenie słowa kluczowe
```

lub

```
'Inny sposób wprowadzenia komentarza
```

W komentarzu można użyć dowolnego znaku, który da się wprowadzić z klawiatury, nawet jeśli nie należy on do zestawu znaków VBA.

Stosuj komentatorze bez nich po pewnym czasie Twój własny program stanie się tak samo tajemniczy, jak napisany przez kogoś innego. Komentarze powinno się umieszczać przynajmniej przed każdą funkcją, aby dokładnie opisać jej zastosowanie, użycie, ograniczenia, znaczenie parametrów itp.

Słowo kluczowe: jest to zastrzeżone słowo o szczególnym znaczeniu w określonym kontekście. W VB nie można używać go jako identyfikatora, zmiennej, funkcji lub etykiety. Wielkości liter w słowach kluczowych nie ma znaczenia, dlatego **`REM`**, **`Rem`** i **`rem`** są tym samym słowem kluczowym.

Wybrane słowa kluczowe (zestaw minimalny)

```
REM,
DIM, AS, BOOLEAN, BYTE, INTEGER, LONG, SINGLE, DOUBLE, STRING,
LET, SIN, COS, TAN, ATN, EXP, LOG, SQR, SGN, ABS, RDN, INT, FIX
TRUE, FALSE, IF, THEN, ELSEIF, ELSE
FOR, TO, STEP, NEXT, DO, LOOP, WHILE, UNTIL, WEND
FUNCTION, EXIT, END
```

W VB słowa kluczowe są używane do tworzenia instrukcji. Każda instrukcja zaczyna się od jakiegoś słowa kluczowego.

Identyfikatory - nazwy

Nazwy są nadawane przez programistę różnym elementom, takim jak zmienne, funkcje, typy itp. W VB jest to ciąg (**maks. 255 znaków**) **liter** (w **Excelu VBA dozwolone są znaki diakrytyczne**), **cyfr i znaku podkreślenia „_”**. Nazwy **muszą zaczynać się od litery** i muszą się różnić od słów kluczowych VB. Wielkość liter nie ma znaczenia

Przykłady:

Poprawne identyfikatory VBA:

dT ≡ dt *(!!! uwaga* **dT/dt** *zawsze będzie równe* 1 *jeśli tylko* dt ≠ 0)

A1, p_max, pi, temperatura, długość (←!! *To jest Ok*), ...

Błędne identyfikatory VBA (błędy wyróżnione na czerwono)

1A, a?, nazwisko prac, temp^2

… i sugestia jak to zmienić

nr_1A, a_pytanie, nazwisko_prac *or* **NazwiskoPrac, temp_2**

W VBA do identyfikatora zmiennej lub funkcji można dodać jeden z następujących znaków:

% & # ^ ! $ @

Są to *znaki typu identyfikatora*, które określają tylko typ danych elementu, i nie są częścią identyfikatora.

Instrukcja jest syntaktyczną jednostką języka programowania. W VB instrukcje służą do sterowania kolejnością wykonywania obliczeń oraz do tworzenia definicji i deklaracji.

Program w VB (zwany kodem źródłowym) jest po prostu sekwencją instrukcji.

Instrukcja musi mieścić się w jednym wierszu. Jeśli jest za długa i nieczytelna, to można ją podzielić na kilka linii. Znakiem kontynuacji jest podkreślnik "_" poprzedzony spacją

W wierszu można umieścić wiele instrukcji a do ich oddzielenia stosuje się dwukropek „:”

Przykłady:

```
'Definicja stałej
Const pi = 3.141592
'Deklaracja zmiennej
Dim time, omega As Double
'Instrukcja przypisania
Let time = 1/(2*pi*f)
'Wiele instrukcji oddzielonych dwukropkiem
Dim f As Double: Let f = 100.0
Let omega = 2*pi*f: Let time = 1/omega
'Jedna długa instrukcja . . .
Let a = 1.000023*x + 0.374091*y + 0.096784*z + 0.278868/v
'. . . ta sama, ale podzielona na kilka wierszy
Let a = 1.000023 * x _
      + 0.374091 * y _
      + 0.096784 * z _
      + 0.278868 / v
```

Przekazanie wyniku funkcji do formuły (miejsca wywołania)

```
' Another dummy function, but this time returning
' specific value i.e. pi number

Function dummy_pi()
  dummy_pi = 3.1415192
End Function
```

To jest literał numeryczny (numerał)

Aby funkcja zwróciła wynik, gdzieś w treści funkcji, musi istnieć co najmniej jedna instrukcja przypisania, która przypisuje wynik obliczenia identyfikatorowi funkcji. Ta wartość zostanie zwrócona do miejsca wywołania. Składnia jest następująca:

LET *nazwa_fun* = *wyrażenie*

Bez tego funkcja zawsze zwraca wartość zerową, co jest zachowaniem domyślnym. Słowo kluczowe LET można pominąć.

Literał to ciąg znaków reprezentujący stałą wartość w kodzie źródłowym, taki jak liczba, ciąg tekstowy, data i inne.

Literał całkowity - stosowany do przedstawiania liczb całkowitych. Jest to ciąg cyfr dziesiętnych poprzedzony w razie potrzeby znakiem minus lub opcjonalnie znakiem plus.

`+125 ≡ 125 ; -456 ; 30113 ; 000123 ≡ 123 …`

Znaki: `%` `&` oraz `^` wymuszają odpowiednio typy INTEGER, LONG lub LONGLONG:
`125% 125& 125^` ← ta same wartość 125, ale różnego typu (i rozmiaru w pamięci)

Literał zmiennoprzecinkowy - używany do przedstawiania liczb wymiernych w postaci dziesiętnej. Składa się z sekwencji cyfr i jednej kropki dzielącej ciąg cyfr na część całkowitą i ułamkową. Liczby ujemne poprzedza się znakiem „-". Znak „+" jest opcjonalny. Kropka może być pierwszym znakiem jeśli część całkowita wynosi zero lub ostatnim jeśli część ułamkowa wynosi zero.

Literał ten można zapisać też w postaci mantysa-cecha (notacja naukowa). Tworzy się go z literału całkowitego lub literału w formacie opisanym wyżej przez dodanie na końcu litery „E" (dużej lub małej), a po niej literału całkowitego. Część przed „E" jest mantysą, a część po „E" jest cechą. Zatem zapis *m*E*c* reprezentuje wartość $m \cdot 10^c$. Przykłady:

`+125.0 ≡ 125.0 ≡ 125. ; -124.567 ; 0.001234 ≡ .001234 ≡ 0.00123400`
`0.0 ≡ 0. ≡ .0` → zero zmiennopozycyjne
`1.0E+03 ≡ 1E3 ≡ 10E2 ≡ 100E1 ≡ 1000.0 ≡ 1000.` → jeden tysiąc
`1.23E+03 ≡ 12.3E+2 ≡ 123.0E1 ≡ 123E1 ≡ 1230.0` → $1.23 \cdot 10^3 = 1230.0$

Znaki `!` oraz `#` wymuszają odpowiednio typy SINGLE lub DOUBLE:
`125.0! ≡ 125! 125.0# ≡ 125#` ← ta same wartość 125, ale różnego typu (i rozmiaru w pamięci)

Literały kontynuacja …

Literał łańcuchowy - reprezentuje tekst, który można wyświetlić na ekranie. Jest to ciąg znaków zawarty między parą cudzysłowów ”…”. Pomiędzy nimi można umieścić dowolny znak, który da się wprowadzić z klawiatury, w tym znaki diakrytyczne. **Duże i małe litery są rozróżnialne**. Literał łańcuchowy może być pusty “”.

```
"To jest tekst, w którym można umieścić dowolny znak drukowalny"
```

Literał musi mieścić się w jednej linii, ale jeśli jest za długi, to można podzielić go na wiersze, używając znaków: kontynuacji „_” i operatora konkatenacji „&” w następujący sposób:

```
long_str = "Bla bla 1" _
         & " bla bla 2 " _
         & " BLA BLA 3"
```

Jeśli znak cudzysłowu ma być elementem tekstu, to należy go powtórzyć dwa razy.

```
"Tytuł filmu to ""Bolek i Lolek"""      ← dobrze
""""     ← ten napis przedstawia pojedynczy cudzysłów (!!)

"Tytuł filmu to "Bolek i Lolek""      ← błąd ponieważ
```

pierwsza część jest interpretowana jako łańcuch `"Tytuł filmu to "`,
po której pojawia się błędny składniowo napis: `Bolek i Lolek""`

Literał daty - (reprezentuje datę i godzinę) sekwencja znaków zamknięta między parą znaków skrótu # ... #

```
#1 Luty 1950#
#01.02.1950#
#1 Maj 2014 12:35#
```

Funkcja z jednym parametrem

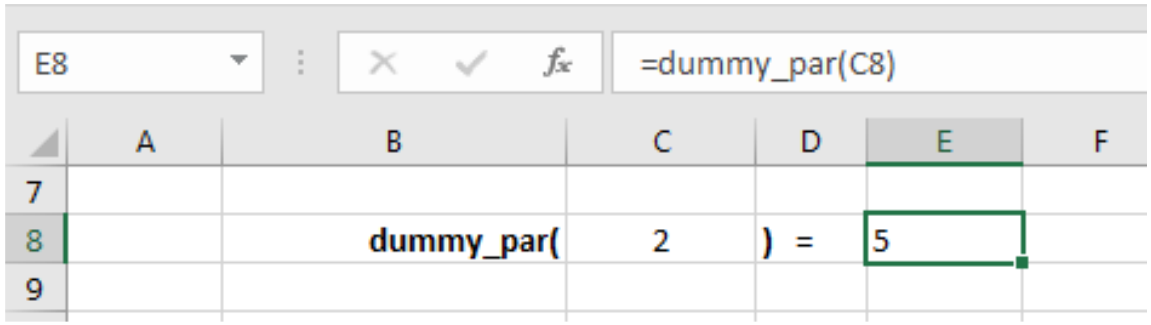
```
' Function with single parameter "x"
' defined as f(x) = 2x + 1

Function dummy_par(x)
  dummy_par = 2 * x + 1
End Function
```

To jest zmienna o nazwie „*x*”

Funkcja jednoargumentowa. Argumenty umieszcza się między nawiasami (tu jest nim „x”). Funkcja wykonuje pewne obliczenia (2x+1), których wynik jest zwracany do formuły. Składnia:

Nazwa_fun (*lista_argumentów*)

dummy_par(x)



Za każdym razem, gdy wprowadzasz poprawki do swojego kodu, powinieneś wcisnąć

Ctrl + Alt + F9

w celu wymuszenia ponownego przeliczenia wszystkich arkuszy.

Zmienne służą do przechowania danych. Zmienna to obszar pamięci operacyjnej komputera o określonym adresie oraz rozmiarze przeznaczony do zapamiętania danej określonego typu oznakowany identyfikatorem. Mówiąc prościej zmienna to nazwany obszar pamięci. Ze zmienną można zrobić dwie rzeczy albo nadać jej wartość, albo odczytać wartość w niej zapisaną. Odczyt ma miejsce zawsze, gdy użyje się jej w wyrażeniu. Przypisanie wartość realizuje się instrukcją przypisania sygnalizowanej słowem kluczowym **LET**. Słowo to w VB można pominąć

LET *nazwa_zmiennej* = *wyrażenie*

lub prościej

nazwa_zmiennej = *wyrażenie*

gdzie:

nazwa_zmiennej - identyfikator zmiennej

wyrażenie – jakieś wyrażenie np. arytmetyczne

Wyrażenie to ciąg literałów, operatorów, zmiennych, funkcji, stałych i nawiasów, który opisuje sposób przetwarzania (obliczania) danych. W przypadku wyrażeń matematycznych obliczenia są wykonywane zgodnie z pierwszeństwem operatorów, które można zmienić za pomocą nawiasów okrągłych (). W tej roli nie można stosować ani nawiasów kwadratowych [ ], ani nawiasów klamrowych { }.

(1) Funkcje	Zapis
Trygonometryczne: *sin(x), cos(x), tg(x), arctg(x)*	`sin(x), cos(x), tan(x), atn(x)`
Potęgowanie e^x=*exp(x)* i logarytm naturalny *ln(x)*	`exp(x), log(x)`
Wartość bezwzględna |x| i znak *sgn(x)*	`abs(x), sgn(x)`
Pierwiastek kwadratowy √x	`sqr(x)` lub `x ^ 0.5`
Część całkowita [x]	`int(x)` *(<x)*, `fix(x)` *(>x dla x<0 i <x dla x>0)*
Liczba losowa	`rnd, randomize` *(inicjacja)*

(2) Operatory arytmetyczne	Zapis
Potęgowanie (^)	`a ^ 2, a ^ 0.5`
Zmiana znaku, minus unarny (-)	`-a`
Mnożenie, dzielenie (*, /)	`a * b, a / b` *(11/2 = 5.5)*
Dzielenie całkowite (\)	`a \ b` *(11\2 = 5)*
Dzielenie modulo (mod)	`a mod b` *(10 mod 3 = 1)*
Dodawanie, odejmowanie (+,-)	`a + b, a - b`
Łączenie napisów (&)	`s & p, "Jeden" & " dwa"`

(3) Operatory porównania (ten sam priorytet):
równy =, , nierówny <>
większy >, mniejszy <,
większy równy >=, mniejszy równy <=

(4) Operatory logiczne	Zapis
Negacja $\neg a$ (not)	`NOT a`
Koniunkcja $a \wedge b$ (and)	`a AND b`
Alternatywa $a \vee b$ (or)	`a OR b`
Alternatywa wykluczająca (xor)	`a XOR b`
Równoważność $a \Leftrightarrow b$ (eqv)	`a EQV b`
Implikacja $a \Rightarrow b$ (ipm)	`a IMP b`

Wyrażenia - kolejność wykonywania obliczeń

Priorytet: Najpierw wyrażenia w nawiasach, potem funkcje, potem operatory arytmetyczne, potem operatory porównania a na końcu operatory logiczne. Operacje są wykonywane wg priorytetu a w przypadku równych priorytetów od **lewej do prawej**. Kolejność wykonywania operacji zmienia się stosując nawiasy (). W przypadku nawiasów zagnieżdżonych pierwszeństwo mają wyrażenia najbardziej zagnieżdżone.

`4*(1+2*(3+4))` obliczany jak `4*(1+2*(3+4))` → `4*(1+2*7)` → `4*(1+14)` → `4*15` → `60`

~~`4*[1+2*(3+4)]`~~ ← źle, dozwolone tylko nawiasy okrągłe

`4/2*12/4` obliczany jak `((4/2)*12)/4` → `(2*12)/4` → `24/4` → `6`

ale nie tak ~~`(4/2)*(12/4)` → `3*4` → `6`~~

Zauważ lewe wiązanie operatora potęgowania i (!) spacje poprzedzające znak ^

`4 ^ 3 ^ 2` obliczany jak `(4 ^ 3) ^ 2` → `64 ^ 2` → `4096`

ale nie tak ~~`4 ^ (3 ^ 2)` → `4 ^ 9` → `262144`~~

Te zapisy mają sens

`x = x + 1` zwiększa x o jeden

`x = x - 1` analogicznie zmniejsza x o jeden

`x = 2 * x` powiększa x dwa razy

Wyrażenia - przykłady transkrypcji wzorów matematycznych

$\frac{a}{b+c}$ → `a/(b+c)`

$\frac{a+b}{c}$ → `(a+b)/c`

$\frac{a+b}{c+d}$ → `(a+b)/(c+d)`

$\frac{a+b}{c\,d} = \frac{a+b}{c}\frac{1}{d}$ →
```
(a+b)/(c*d)
(a+b)/c/d
```

`a+b/c` → $a+\frac{b}{c}$

$a+(-b)$ →
```
a - b
a + -b
a + (-b)
-b + a
```

$\frac{-b+\sqrt{b^2-4ac}}{2a}$ →
```
(-b+sqr(b^2-4*a*c))/(2*a)
(-b+sqr(b*b-4*a*c))/2/a
```

$\frac{ab}{cd}$ →
```
a*b/(c*d)
(a*b)/(c*d)
a/c*b/d
```

$\sqrt{\frac{2\kappa}{\kappa-1}\frac{p}{\varrho}\left[1-\left(\frac{P_0}{p}\right)^{(\kappa-1)/\kappa}\right]}$ →
```
sqr(2*k*p/((k-1)*ro)*(1-(p0/p)^((k-1)/k)))
sqr(2*k/(k-1)*p/ro*(1-(p0/p)^((k-1)/k)))
```

Za długie wyrażenie można podzielić dla wygody na kilka wierszy. Znakiem kontynuacji wiersza jest "_" poprzedzone spacją.

```
a = 1.000023 * x + 0.374091 * y + 0.096784 * z + 0.278868 / v
```

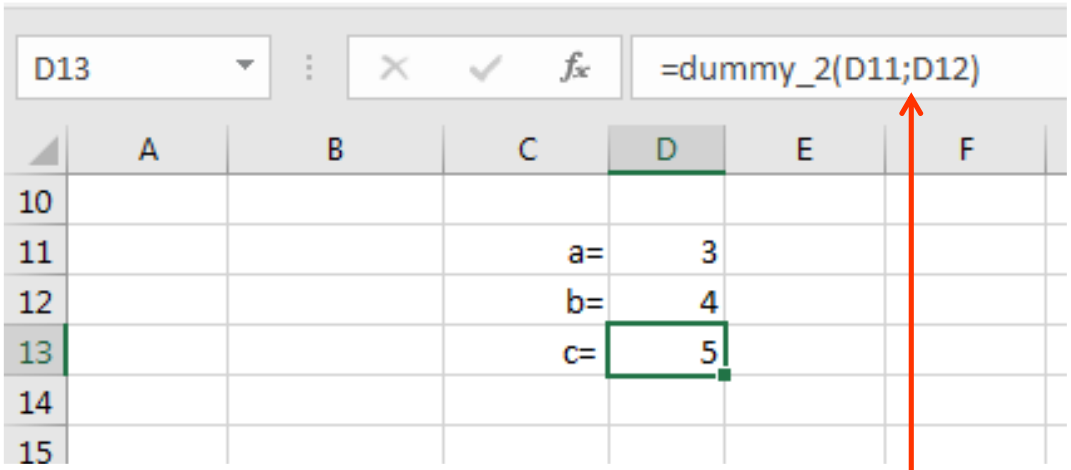
```
a = 1.000023 * x _
  + 0.374091 * y _
  + 0.096784 * z _
  + 0.278868 / v
```

Funkcja z wieloma argumentami

```
' Function wich takes two parameters "a" and "b"
' calculating hypotenuse "c" in right trangle
' of sides a, b and c according to Pythagorean theorem

Function dummy_2(a, b)
  dummy_2 = Sqr(a * a + b * b)
End Function
```

Funkcja przyjmuje dwa parametry wymienione między nawiasami (tutaj są to „a” i „b”) i oblicza przeciwprostokątną „c” w prawym trójkącie boków a, b i c zgodnie z twierdzeniem Pitagorasa.

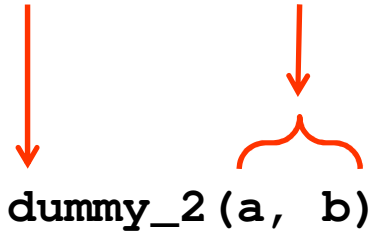
$$c = \sqrt{a^2 + b^2}$$



Uwaga: w polskiej wersji argumenty formalne są oddzielone średnikiem a nie przecinkiem jak w VBA.

Składnia:

Nazwa_fun (lista_argumentów)



Typ danych (po prostu typ) to zbiór wartości, które przyjmować dane wskazanego typu. Typ danych określa, jakie operacja można na nich wykonywać i jak jej wartości są reprezentowane w pamięci. W VB istnieją typy proste i typy złożone. Typy proste reprezentują wartości, których nie można rozłożyć na mniejsze komponenty. Wartości typów złożonych są strukturą, której elementy mogą być zarówno typami pierwotnymi, jak i złożonymi. Poniżej lista podstawowych typów VB:

Typy liczbowe: (zielony - całkowity, czerwony – typ zmiennoprzecinkowy)

Typ	Zakres	Rozmiar [B]
BYTE	0 .. 255 (dokładność 1-3 cyfry)	1
INTEGER	-32768 .. 32767 (dokł. do 4-5 cyfr)	2
LONG	-2147483648 .. 2147483647 (dokł. do 9-10 cyfr)	4
LONGLONG	-9223372036854775808 .. 9223372036854775807 ($\pm 10^{19}$) (dokł. do 18-19 cyfr)	8
SINGLE	$\pm 1{,}5 \cdot 10^{-45}$ .. $\pm 3{,}4 \cdot 10^{38}$ (dokł. do 8 cyfr)	4
DOUBLE	$\pm 5{,}0 \cdot 10^{-324}$ .. $\pm 1{,}7 \cdot 10^{308}$ (dokł. do 16 cyfr)	8

Typ logiczny:

BOOLEAN → posiada jedną z dwóch wartości **TRUE** albo **FALSE**
np. 2 > 3 → FALSE, 2 <> 3 → TRUE, 2 = 3 → FALSE …

Typ łańcuchowy: napisy o różnej długości max. 65535

```
STRING     'Napis o długości do 65535 znaków
STRING*20  'Napis maks. 20 znakowy
```

Deklarowanie zmiennych, stałych i funkcji nie jest obowiązkowe, ale wysoce zalecane. Deklaracje oszczędzają czas wykonania i miejsce w pamięci. Niezadeklarowane zmienne nazywane są *wariantami* i zajmują więcej miejsca w pamięci. Powinny być używane tylko wtedy, gdy zmienna ma przechowywać dane różnego typu. Jeśli typ danych jest dobrze znany, należy zadeklarować każdą zmienną. Składnia deklaracji:

```
DIM lista_zmiennych AS Typ
```

Gdzie: ***lista_zmiennych*** – lista nazw zmiennych oddzielonych przecinkami, ***Typ*** – słowo kluczowe typu (np. **integer, long, single, double, Boolean** itp.) lub identyfikator zadeklarowanego przez programistę własnego typu.

```
' Zadeklarowania nazw a, b jako zmienne podwójnej precyzji
DIM a, b AS Double
' Powyższy zapis jest równoważny następującemu
DIM a AS Double
DIM b AS Double
```

Stosując tzw. *znaki typu* **% & ^! # $** deklarację można uprościć. Wystarczy, przy pierwszym pojawieniu się identyfikatora w kodzie użyć wymienionego sufiksu. Przykłady:

```
Dim i As Integer   ← równoważny →   i%
Dim n As Long      ← równoważny →   n&
Dim x As Single    ← równoważny →   x!
Dim y As Double    ← równoważny →   y#
```

Definicje prostych funkcji przykłady

Przykład definiowania sinusa hiperbolicznego

$$\sinh x = \frac{e^{x} - e^{-x}}{2}$$

Funkcja 1 argumentowa

```
Function sinh(x as double) as double
 sinh = (exp(x) - exp(-x))/2#
End Function
```

- Zalecany sposób definiowania

```
Function sinh#(x#)
 sinh = 0.5*(exp(x) - exp(-x))
End Function
```

- Alternatywny sposób definiowania nagłówka. Typ funkcji i argumentu (double) określony jest znakiem # umieszczonym na końcu identyfikatorów *x* i sinh.

```
Function sinh(x)
  sinh = (exp(x) - exp(-x))/2#
End Function
```

- Akceptowalny, ale nie zalecany sposób definiowania, przynajmniej w przypadku, gdy typy argumentów lub wyniku funkcji są z góry określone.

Przykład definiowania funkcji *d()* liczącej długość przekątnej prostokąta a x b

$$d(a,b) = \sqrt{a^2 + b^2}$$

Funkcja 2 argumentowa

```
Function diag (a as double, b as double) as double
 diag = sqr(a^2 + b^2)
End Function
```

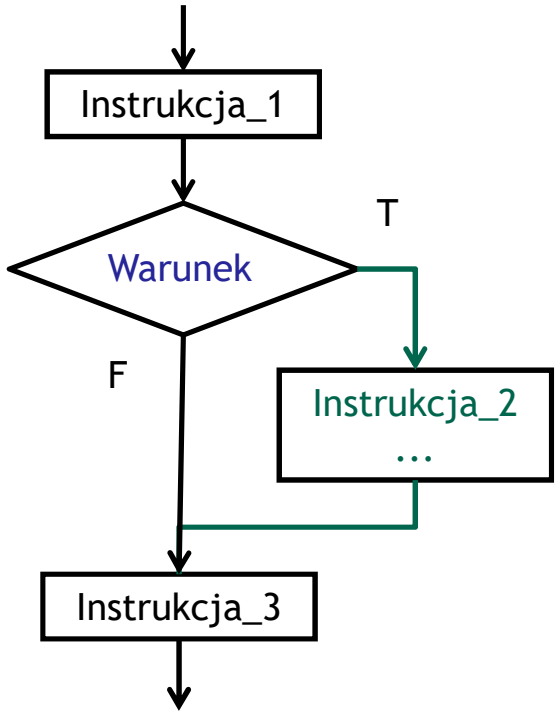
```
Function diag#(a#, b#)
 diag = sqr(a*a + b*b)
End Function
```

Podnoszenie do kwadratu **a^2** ze względu na czas obliczeń lepiej zastąpić mnożeniem **a*a**.

Instrukcja IF dla warunkowego wykonania innej instrukcji

```
Instrukcja_1
IF warunek THEN Instrukcja_2
Instrukcja_3
```

```
Instrukcja_1
IF warunek THEN
  Instrukcja_2
  ...
END IF
Instrukcja_3
```



Przykłady: realizacja |x| `X = ABS(X)`

```
X = -15
...
IF X < 0 THEN X = -X
... ' Tu x=15
```

```
X = -15
...
IF X < 0 THEN
    X = -X
END IF
...'tu już x=15
```

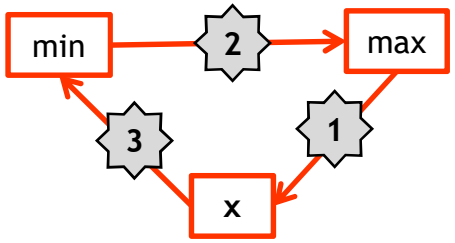
```
Function vb_abs(x As Double) As Double
   If x < 0 Then x = -x
   vb_abs = x
End Function
```

Zamiana miejscami wart. zmiennych *min* i *max*, gdy *min* > *max*.

```
min = 15
max = -1
...
IF min > max THEN x = max: max = min: min = x
... ' Tu min=-1 a max=15
```

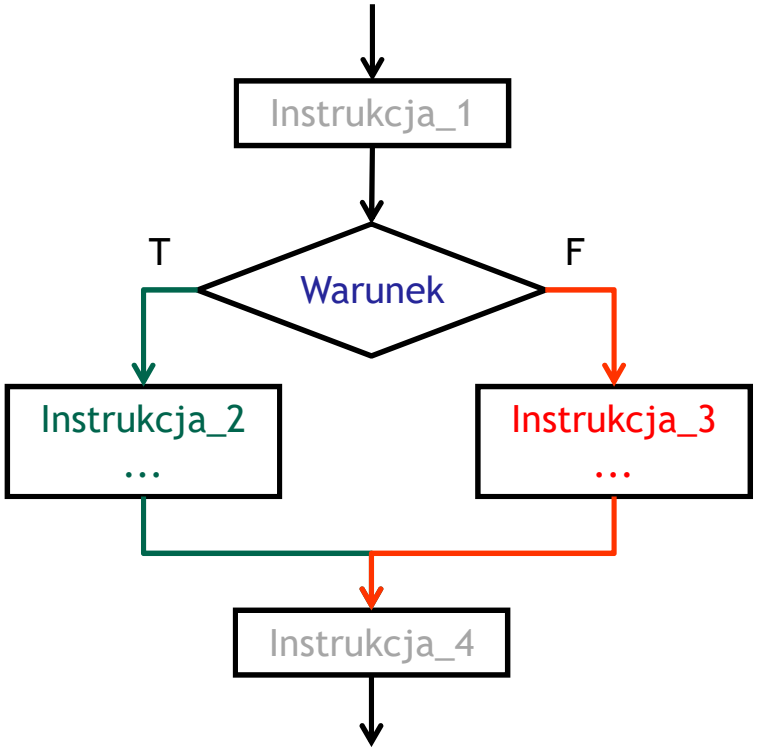
Wiele instrukcji oddziela się dwukropkiem ":"

Algorytm wymiany wartości między zmiennymi *min* i *max* za pośrednictwem zmiennej pomocniczej *x*



Rozgałęzienia - strukturalna instrukcja warunkowa IF THEN ELSE

```
Instrukcja_1
IF warunek THEN
  Instrukcja_2
  ...
ELSE
  Instrukcja_3
  ...
END IF
Instrukcja_4
```



Przykłady:

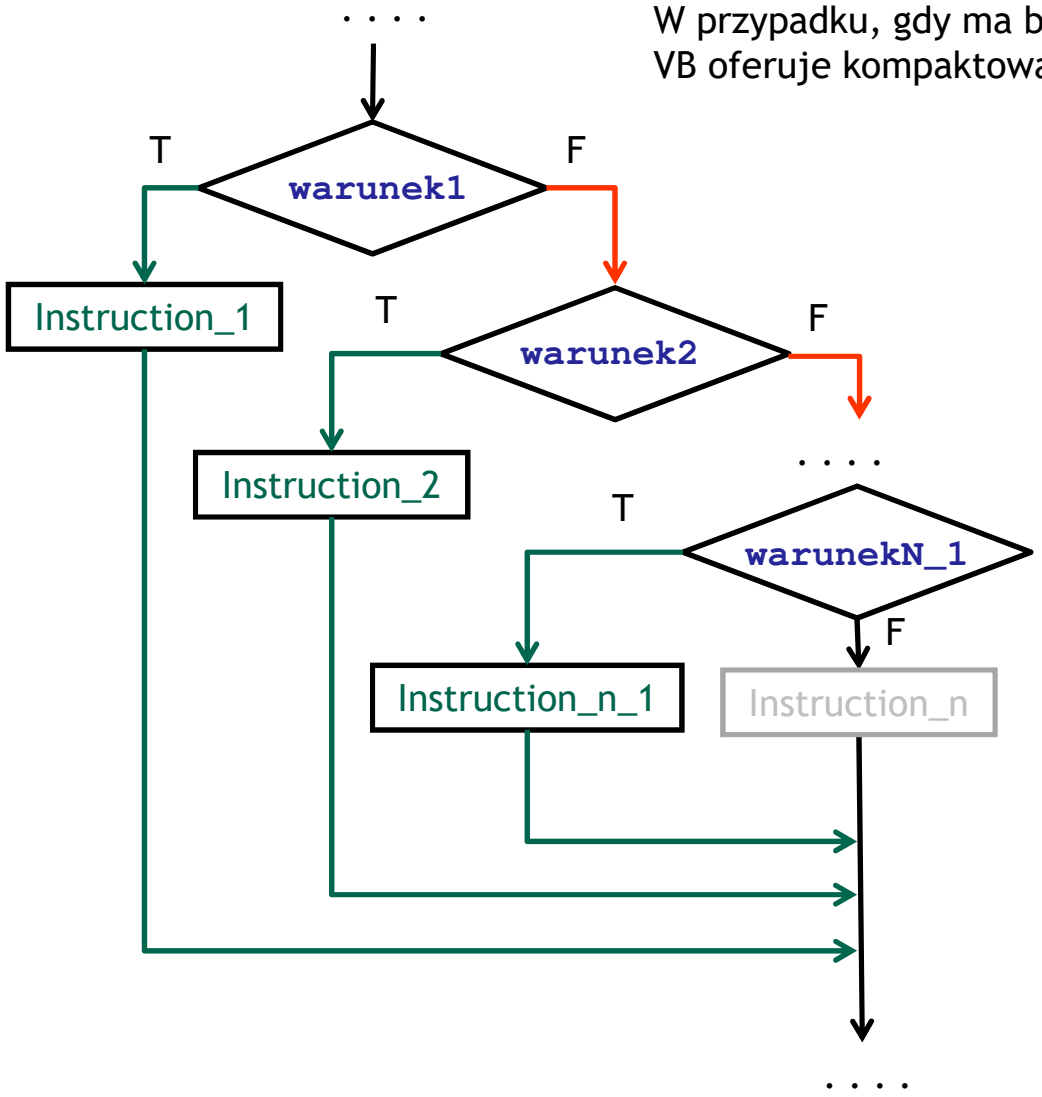
$$\operatorname{sinc}(x) = \begin{cases} 1 & \text{dla } x = 0 \\ \dfrac{\sin(x)}{x} & \text{dla } x \neq 0 \end{cases}$$

```
Function sinc(x As Double) As Double
  If x = 0 Then
    sinc = 1#
  Else
    sinc = Sin(x) / x
  End If
End Function
```

$$\operatorname{sgn}(x) = \begin{cases} -1 & \text{dla } x < 0 \\ 0 & \text{dla } x = 0 \\ 1 & \text{dla } x > 0 \end{cases}$$

```
Function vb_sgn(x As Double) As Double
  If x < 0 Then
    vb_sgn = -1#
  Else
    If x > 0 Then
      vb_sgn = 1#
    Else
      vb_sgn = 0
    End If
  End If
End Function
```

Rozgałęzienia - instrukcja warunkowa IF THEN ELSEIF ELSE



W przypadku, gdy ma być zastosowany łańcuch **if then else if then …**
VB oferuje kompaktową wersję instrukcji IF

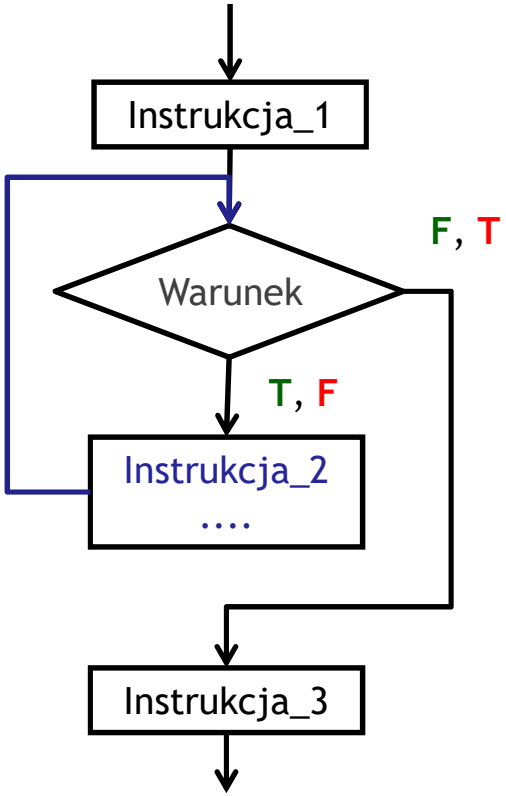
```
IF warunek1 THEN
    instruction_1
ELSEIF warunek2 THEN
    instruction_2
  . . . .
ELSEIF warunekN_1 THEN
     instruction_n_1
ELSE
     instruction_n
END IF
```

Tu ELSE jest opcjonalne

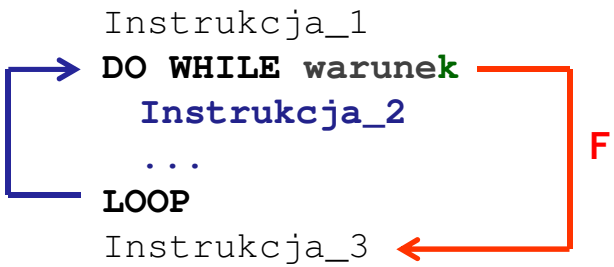
```
Function vb_sgn(x As Double) As Double
  If x < 0 Then
    vb_sgn = -1#
  ElseIf x > 0 Then
    vb_sgn = 1#
  Else
    vb_sgn = 0
  End If
End Function
```

⚠ Uwaga! "ELSEIF" to nie to samo co "ELSE IF"

Strukturalna instrukcja pętli DO LOOP - warunek na początku

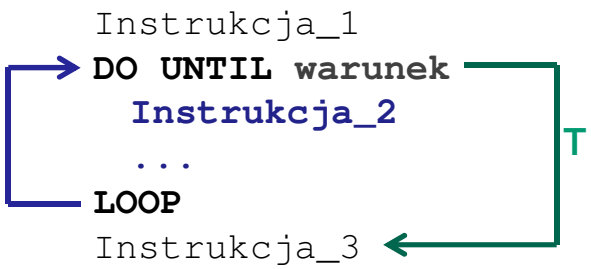


Warunek na utrzymania się w pętli (WHILE - *podczas gdy*)




```
Instrukcja_1
WHILE warunek
  Instrukcja_2
  ...
WEND
Instrukcja_3
```

Warunek na wyjście z pętli (UNTIL - *dopóki nie*)




```
Instrukcja_1
WHILE NOT warunek
  Instrukcja_2
  ...
WEND
Instrukcja_3
```

Przykład, obliczanie silni liczby *n czyli n! = 1·2 ·3 · ... · n*

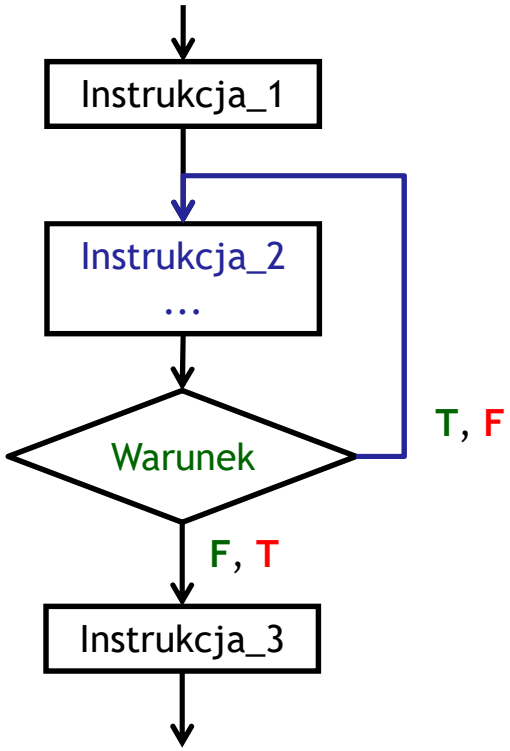
```
silnia = 1
i = 1
DO WHILE i <= n
  silnia = silnia * i
  i = i + 1
LOOP
```

```
silnia = 1
i = 1
DO UNTIL i > n
  silnia = silnia * i
  i = i + 1
LOOP
```

lub `NOT i <= n`

```
Function vb_silnia(n As Byte) As Double
  Dim wynik As Double
  Dim i As Byte
  wynik = 1
  i = 1
  Do Until i > n          (Do While i <= n )
    wynik = i * wynik
    i = i + 1
  Loop
  vb_silnia = wynik
End Function
```

Strukturalna instrukcja pętli DO LOOP - warunek na końcu



Warunek na utrzymanie się w pętli (WHILE)

```
Instrukcja_1
DO
  Instrukcja_2
  ...
LOOP WHILE warunek
Instrukcja_3
```

T

Warunek na wyjście z pętli (UNTIL)

```
Instrukcja_1
DO
  Instrukcja_2
  ...
LOOP UNTIL warunek
Instrukcja_3
```

F

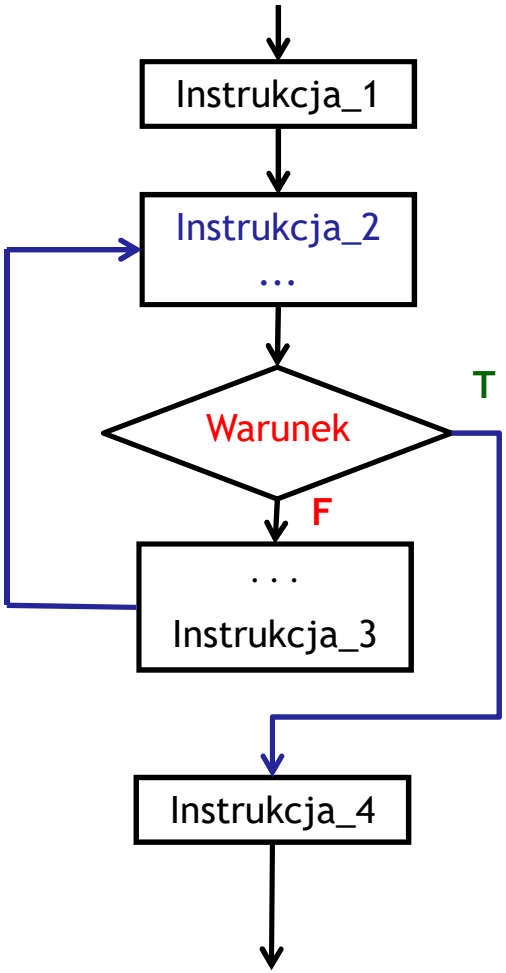
Przykład, wymuszenie wczytanie dodatniej wartości do zmiennej X

```
DO
  INPUT "X = ", X
LOOP WHILE X <= 0
```

```
DO
   INPUT "X = ", X
LOOP UNTIL X > 0
```

```
Function vb_input() As Double
  Dim dana
  Do
     dana = Application.InputBox("Podaj liczbę dodatnią")
  Loop While dana <= 0
  vb_input = dana
End Function
```

Strukturalna instrukcja pętli DO LOOP - warunek wewnątrz



Wyjście z wnętrza pętli **EXIT DO** występuje w jakiejkolwiek instrukcji warunkowej (np. IF, SELECT)

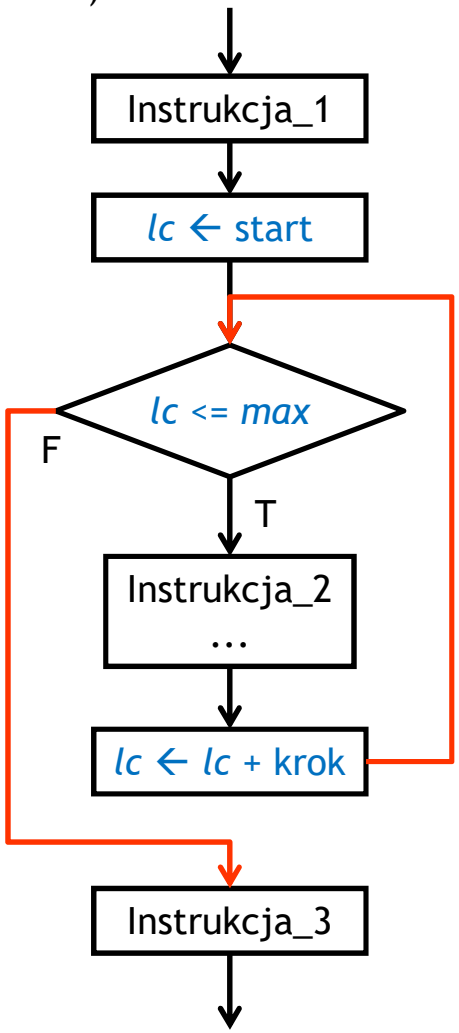
Można łączyć z WHILE i UNTIL

```
Instrukcja_1
DO
  Instrukcja_2
  ...
  IF Warunek THEN EXIT DO
  ...
  Instrukcja_3
LOOP
Instrukcja_4
```

T

Instrukcja pętli iteracyjnej FOR

Pętla jest wykonywana określoną liczbę razy, czym steruje zmienna *lc* zwana licznikiem pętli (jej nazwa może być dowolna).




```
Instrukcja_1
FOR lc = start TO Max STEP krok
  Instrukcja_2
  ...
NEXT lc
Instrukcja_3
```

```
Instrukcja_1
FOR lc = 1 TO Max
  Instrukcja_2
  ...
NEXT
Instrukcja_3
```

Przypadek: start=1, krok=1. Wykonanie Max razy instrukcji *Instrukcja_2*

Przykład, obliczanie silni liczby *n*

```
silnia = 1
FOR i = 2 TO n
  silnia = silnia * i
NEXT
```

Alternatywnie pętla realizowana za pomocą instrukcji DO … LOOP

```
Instrukcja_1
lc = start
DO WHILE lc <= Max
  Instrukcja_2
  lc = lc + krok
LOOP
Instrukcja_3
```

Uwagi:
Część **STEP** ***krok*** można pominąć gdy *krok*=1. Po słowie **NEXT** można pominąć też nazwę licznika *lc*

Definicje stałych

W programie często stosuje się stałe reprezentowane przez literały np. numeryczne. Współczesne jeżyki oferują możliwość oznakowania stałej wygodnym i mówiącym coś identyfikatorem. Taki identyfikator reprezentuje wartość, której program nie może zmienić podczas normalnego wykonywania. Wskazane jest stosowanie stałych zamiast literałów lub wyrażeń stałych (tj. wyrażeń, w których nie stosuje się zmiennych), gdyż dzięki temu kod źródłowy jest bardziej przejrzysty, czytelny, elastyczny i łatwiejszy w utrzymaniu.

Składnia:

CONST *nazwa_stałej* = *stałe_wyrażenie*

lub

CONST *nazwa_stałej* **AS** *type* = *stałe_wyrażenie*

Przykłady:

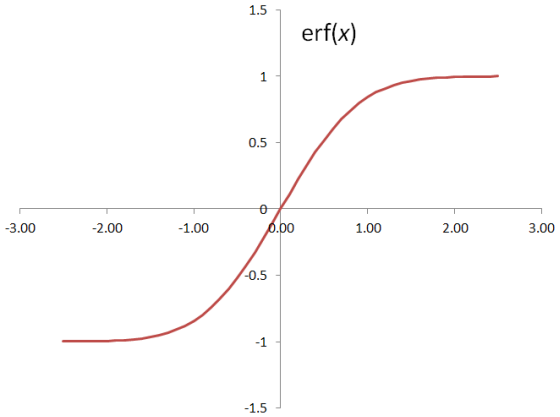
```
CONST pi = 3.141592

CONST deg = pi / 180#

CONST pi_2 AS DOUBLE = pi / 2


' Błąd spowodowany użyciem funkcji standardowej. Można stosować
' jedynie podstawowe operatory

CONST pi = 4 * ATN(1)
```

Przykład definiowania funkcji erf(x)

Funkcja błędu Gaussa **erf**(x) jest nieelementarną funkcją stosowaną w rachunku prawdopodobieństwa, statystyce oraz w teorii równań różniczkowych cząstkowych.

$$\operatorname{erf}(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} dt$$

```
'Funkcja erf(x) - adaptacja errfcc z Numerical Recipes in C
'Sect. 6.2 dokładność 1.2E-7

Function nr_erf(x As Double) As Double
  Dim t, z, ans As Double
  z = Abs(x)
  t = 1.0 / (1.0 + 0.5 * z)
  ans = t * (-0.82215223 + t * 0.17087277)
  ans = t * (1.48851587 + ans)
  ans = t * (-1.13520398 + ans)
  ans = t * (0.27886807 + ans)
  ans = t * (-0.18628806 + ans)
  ans = t * (0.09678418 + ans)
  ans = t * (0.37409196 + ans)
  ans = t * (1.00002368 + ans)
  ans = t * Exp(-z * z - 1.26551223 + ans)
  If x >= 0 Then
     nr_erf = 1.0 - ans
  Else
     nr_erf = ans - 1.0
  End If
End Function
```

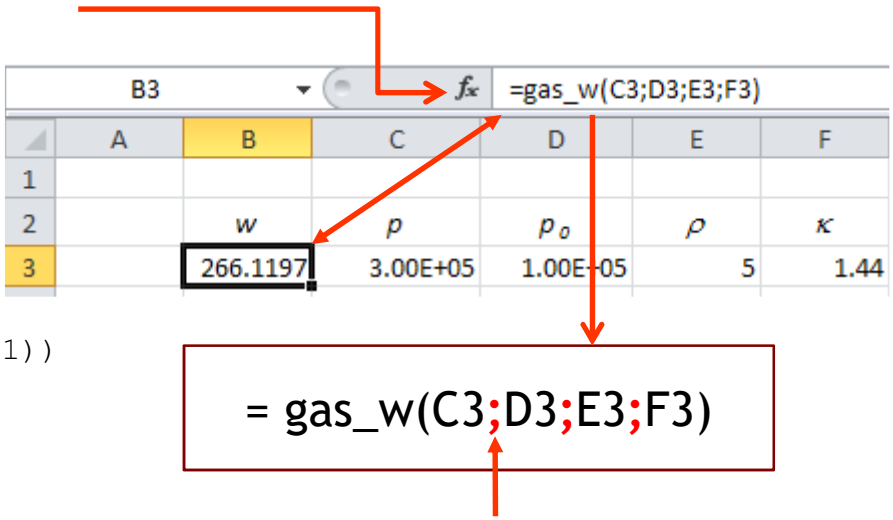


Przykład definiowania i użycia funkcji.

Prędkość wypływu gazu o gęstości ρ, ciśnieniu p i wykładniku adiabaty κ, ze zbiornika do otoczenia, w którym panuje ciśnienie P_0 dana jest wzorem:

$$w = \begin{cases} \sqrt{\frac{2\kappa}{\kappa-1}\frac{p}{\varrho}\left[1-\left(\frac{P_0}{p}\right)^{(\kappa-1)/\kappa}\right]} & \text{dla} \quad \frac{P_0}{p} > \beta_{kr} \\ \sqrt{\frac{2\kappa}{\kappa+1}\frac{p}{\varrho}} & \text{dla} \quad \frac{P_0}{p} < \beta_{kr} \end{cases} \qquad \text{gdzie} \quad \beta_{kr} = \left(\frac{2}{\kappa+1}\right)^{\kappa/(\kappa-1)}$$

```
' Funkcja oblicza prędkość wypływu gazu [m/s] ze
' zbiornika na podstawie
'  p - ciśnienia w zbiorniku [Pa]
'  p0 - ciśnienia otocznia [Pa]
'  ro - gęstości gazu w zbiorniku [kg/m3]
'  kappa - wykładnika adiabaty czynnika
' ------------------------------------------------
Function gas_w(p#, p0#, ro#, kappa#) As Double
  Dim beta, p0_p, k1 As Double
 'zmienne pomocnicze
  p0_p = p0 / p
  k1 = (kappa - 1) / kappa
 'krytyczny stosunek ciśnień
  beta = (2 / (kappa + 1)) ^ (1 / k1)
 'obliczenie prędkości wypływu
  If p0_p > beta Then
    gas_w = Sqr(2 * p / (k1 * ro) * (1 - p0_p ^ k1))
  Else
    gas_w = Sqr(2 * kappa / (kappa + 1) * p / ro)
  End If
End Function
```



Uwaga! separatorem argumentów w formule arkusza jest w PL **średnik** a nie przecinek jak w programie Basic

Linki do stron o tematyce VBA

[1] https://www.exceltrick.com/excel-vba-basics/getting-familiar-with-vbe/

[2] https://msdn.microsoft.com/en-us/library/office/ee814737%28v=office.14%29.aspx#odc_Office14_ta_GettingStartedWithVBAInExcel2010_VBAProgramming101

[2] Excel 2010 Developer Reference
https://msdn.microsoft.com/en-us/library/office/ff846392%28v=office.14%29.aspx

[3] Visual Basic for Applications language reference for Office 2010
https://msdn.microsoft.com/en-us/library/office/gg278919%28v=office.14%29.aspx

[4] Visual Basic for Applications language reference for Office 2013
https://msdn.microsoft.com/en-us/library/office/gg264383%28v=office.15%29.aspx

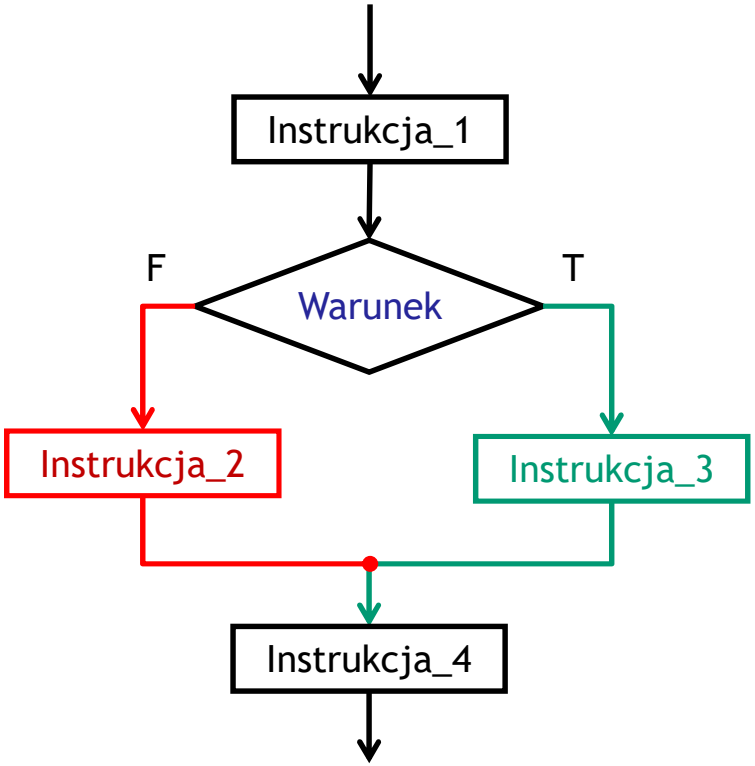
[7] MS Office Dev Center:
https://docs.microsoft.com/en-us/office/vba/excel/concepts/events-worksheetfunctions-shapes/list-of-worksheet-functions-available-to-visual-basic

[8] Excel and VBA Solutions - http://www.globaliconnect.com/excel/

[9] https://docs.microsoft.com/pl-pl/office/vba/language/glossary/vbe-glossary#variable

```
 Instrukcja_1
 IF warunek THEN GOTO Etyk_1:
 instrukcja_2
 GOTO Etyk_2
Etyk_1:
 Instrukcja_3
Etyk_2:
 Instrukcja_4
```

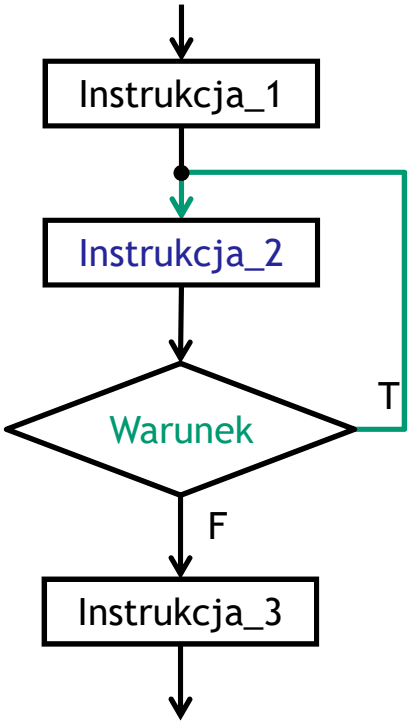
Etyk_x to symboliczna lub numeryczna etykieta. W przykładach obok reprezentują ją identyfikatory: `Dodatnia`, `Omin, Dalej`

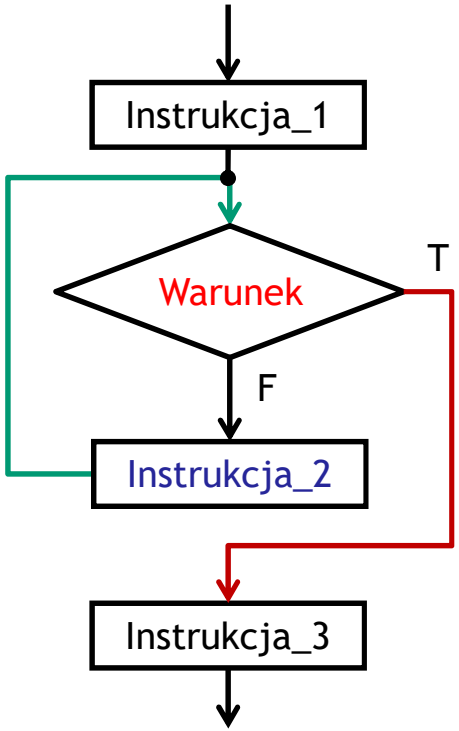


Przykład niepełnej realizacji sgn(*x*)

```
   X = -10
   IF X > 0 THEN GOTO Dodatnia
   SGN = -1
   GOTO Omin
Dodatnia:
   SGN = 1
Omin:
   ... 'tu sgn=-1
```

Niestrukturalna (GOTO) realizacja pętli






```
  Instrukcja_1
1000:
  Instrukcja_2
  IF warunek THEN GOTO 1000
  Instrukcja_3
```

```
  Instrukcja_1
1000:
  IF warunek THEN GOTO 2000
  Instrukcja_2
  GOTO 1000
2000:
  Instrukcja_3
```

W przykładach zastosowano etykiety numeryczne: 1000 i 2000