MS Excel: podstawy programowania w języku VBA

Materiały szkoleniowe

KONTAKT

Adres

Expose sp. z o. o.
ul. Skierniewicka 10a
01-230 Warszawa

Telefon

+ 48 22 465 88 88
+ 48 22 240 19 99

Online

biuro@expose.pl
www.expose.pl
www.chcesieuczyc.pl

Spis treści

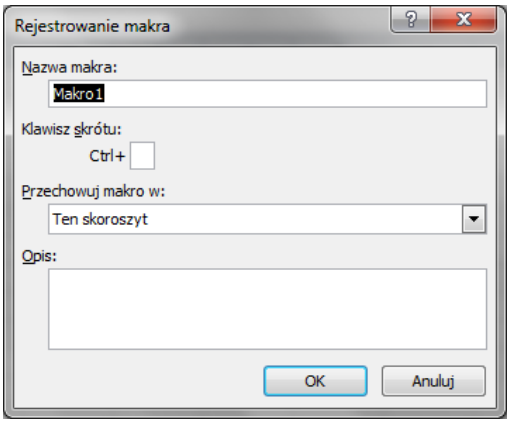
1 Rejestrowanie makr

Makra są zbiorem instrukcji, które automatyzują te same powtarzające się zadania. Po uruchomieniu makra Excel wykonuje poszczególne czynności w kolejności, w jakiej zostały one zawarte w tym makrze. Pozwala to na znaczne uproszczenie wielu prac

Rejestracja makr przy użyciu interfejsu programu Microsoft Office Excel dostępna jest na karcie **Widok** w grupie **Makra** lub na karcie **Deweloper** w grupie **Kod**.

- **Edycja i uruchamianie makr**

Aby zarejestrować makro należy użyć **Zarejestruj makro** na karcie **Deweloper**. W otwartym oknie wprowadzamy **Nazwę makra**, **Klawisz skrótu** oraz **Przechowuj makro w**.



Nazwa makra

Istnieje możliwość podania nazwy dla rejestrowanego makra. Domyślnie Excel dla kolejnych zarejestrowanych makr używa nazw Makro1, Makro2 itd.

Klawisz skrótu

Umożliwia przypisanie do makra kombinacji klawiszy, których wciśnięcie spowoduje uruchomienie makra. Należy pamiętać, że zdefiniowany w ten sposób skrót klawiszowy nadpisuje wbudowane skróty (jeżeli takie istnieją).

Przechowuj makro w

Decydujemy gdzie ma zostać zapisane rejestrowane makro. Domyślnie Excel umieszcza makra w module aktywnego skoroszytu. Tworzone makra można również zapisać w nowym

skoroszycie (Excel utworzy w tym celu nowy, pusty skoroszyt) lub w skoroszycie makr osobistych.

Excel zapamiętuje wybór lokalizacji, dzięki czemu przy następnej rejestracji użyje tych samych ustawień.

Modyfikowanie zarejestrowanych makr

Wynikiem zarejestrowania nawet jednego, prostego polecenia może być całkiem spora ilość kodu języka VBA. W wielu przypadkach po zarejestrowaniu makra wiele zbędnych poleceń kodu źródłowego można po prostu usunąć. Excel zazwyczaj rejestruje to, co jest zaznaczone (czyli obiekt), a następnie w kolejnych poleceniach posługuje się obiektem Selection. Poniżej przedstawiono przykładowy kod źródłowy, który zostanie wygenerowany przez rejestrator makr po zaznaczeniu zakresu komórek, a następnie użyciu kilku przycisków z karty Narzędzia główne do zmiany formatowania liczb oraz zastosowania pogrubienia i kursywy.

```
Sub Makro1()
      Range("A7:A12").Select
      Selection.Font.Bold = True
      Selection.Font.Italic = True
      Selection.NumberFormat = "0.00"
End Sub
```

Wygenerowany kod VBA oczywiście działa, ale to tylko jeden ze sposobów wykonania takich operacji. Zamiast tego można użyć bardziej efektywnej konstrukcji z poleceniami With i End With:

```
Sub Makro1()
      Range("A7:A12").Select
      With Selection
      .Font.Bold = True
      .Font.Italic = True
      .NumberFormat = "0.00"
      End With
End Sub
```

Pomijając metodę Select, można stworzyć jeszcze bardziej wydajny kod:

```
Sub Makro1()
      With Range("A7:A12")
      .Font.Bold = True
      .Font.Italic = True
      .NumberFormat = "0.00"
      End With
End Sub
```

- **Bezpieczeństwo makr**

Podczas pracy z makrami w programie Excel należy ustawić odpowiedni poziom zabezpieczeń. Aby to zrobić w zakładce **Deweloper**, w karcie **Kod** należy wybrać **Bezpieczeństwo makr**.

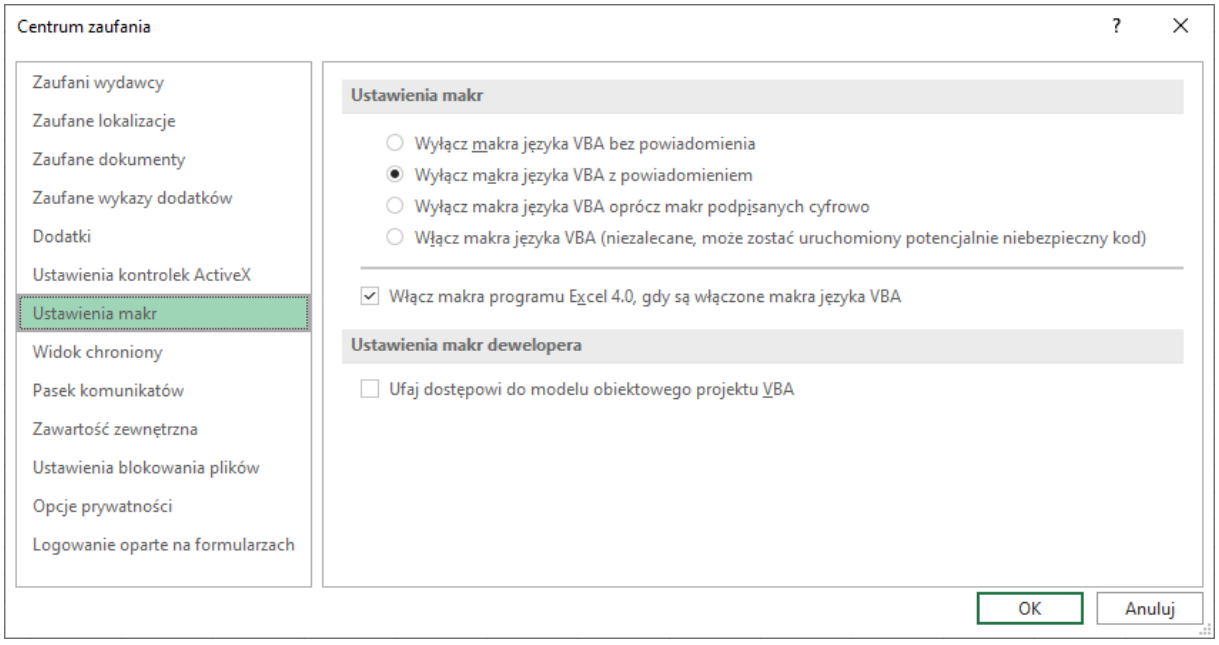
Mamy do wyboru cztery opcje:

- Włącz wszystkie makra
- Wyłącz wszystkie makra i wyświetl powiadomienie
- Wyłącz wszystkie marka oprócz makr podpisano cyfrowo
- Włącz wszystkie makra (niezalecane, może zostać uruchomiony niebezpieczny kod)

Włącz wszystkie makra, opcja niezalecana przez Microsoft ale z powodzeniem stosowana przez wielu użytkowników.

Wszystkie makra są uruchamiane, jest to bardzo wygodne i śmiało może być używane o ile użytkownicy uruchamiają wyłącznie pliki z makrami, które sami utworzyli lub zostały utworzone przez ich współpracowników, nie ponoszą więc ryzyka uruchomienia niebezpiecznego kodu.

Wyłącz wszystkie makra i wyświetl powiadomienie – Excel każdorazowo wyświetli informacje o tym że makra zostały wyłączone i pozwoli je nam włączyć, opcja niezalecana przeze mnie, od setnego zapytania wzwyż korzystanie z niej może mieć niekorzystny wpływ na system nerwowy użytkownika.



Jeżeli zdecydujemy się na **Wyłącz wszystkie makra i wyświetl powiadomienie**, po uruchomieniu pliku zawierającego makra zostanie wyświetlony pasek z ostrzeżeniem o zabezpieczeniach.

Budowa makr

```
Sub nazwa()
      'komentarze pomijane przez VBA
      'dokumentują sens fragmentów programu
      instrukcje – zestaw poleceń
End Sub
```

Moduł może zawierać wiele programów i funkcji

Elementy programu VBA

- Słowa kluczowe – polecenia sterujące wykonywaniem programu – słowa (if), skróty (mid) i zbitki skrótów (rmid) z języka angielskiego.
- Identyfikatory – nazwy zmiennych, stałych, obiektów, programów/podprogramów/funkcji
- Komentarze

Nazwy makr

- ciągi liter i cyfr
- nie mogą zawierać znaków specjalnych: <spacja>, (), :, ;, itp. (ale mogą znak podkreślenia _)
- mogą mieć dowolną długość
- nazwa MUSI rozpoczynać się literą

Nazwy makr można zmieniać w dowolnym momencie, gdy nie są uruchomione, należy jednak pamiętać o tym, że nazwa może być użyta gdzieś w naszych innych programach korzystających z tego makra jako podprogramu;

Konstrukcja With … End With

Makra rejestrowane w Excelu warto skracać, gdyż przy każdej operacji, Excel zapisuje w makrze pełen stan danego obiektu po operacji, także właściwości niezmieniane.

Przy skracaniu warto upewnić się, że:

- Nie skracamy początków lub końców bloków słów kluczowych (np. With);
- Nie usuwamy przesunięć kursora w makrze względnym (Offset);
- Nie usuwamy właściwości, które zostały zmienione;

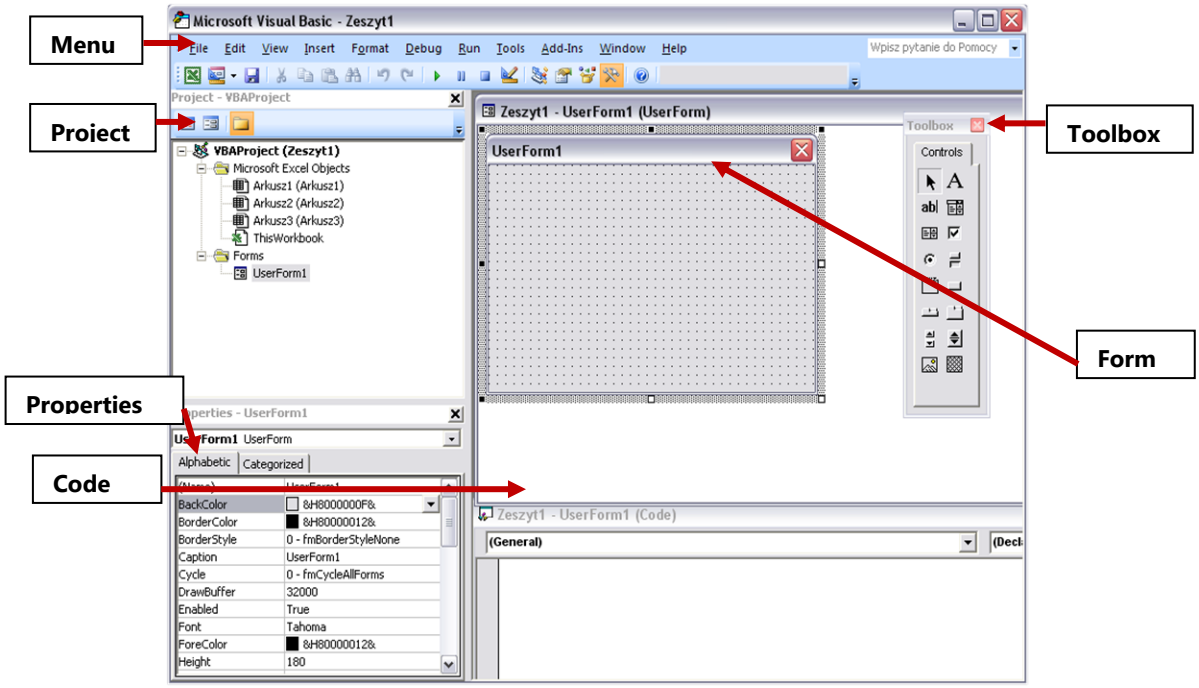
Notatki własne:

2 Edytor VBA

Podstawy edytora Visual Basic

Dzisiejsze języki programowania znacznie się różnią od języków programowania sprzed kilku lat. Dzięki systemowi operacyjnemu Microsoft Windows, który używa graficznego interfejsu, powstało wiele narzędzi, bardziej zaawansowanych, które wykorzystują interfejs graficzny. Programy te, choć często spełniają taką samą rolę jak ich poprzednicy, są ze względu na swój wygląd, bardziej przyjazne dla użytkownika. Tworzenie programów z graficznym interfejsem i wykorzystujących zdarzenia generowane przez użytkownika stało się czasochłonne w językach strukturalnych. Wymusiło to rozwój języków programowania. Wprowadzenie wizualizacji do języków programowania znacznie uprościło proces tworzenia graficznego interfejsu aplikacji.

Visual Basic jest aplikacją Windows i sam wykorzystuje interfejs graficzny. Platforma projektowa i środowisko projektowe Visual Basic nosi nazwę Developer Studio. Składa się ono z wielu okien i narzędzi, które pozwalają (i ułatwiają) tworzyć rozbudowane aplikacje działające w środowisku Microsoft Windows. W fazie projektowania programu, główne okno Visual Basic zawiera kilka różnych okien. Każde z nich może być ukryte w miarę potrzeby. Każde z nich ma inne zadanie i jest wykorzystywane do innych czynności niezbędnych przy tworzeniu aplikacji.



Budowa edytora Visual Basic

Pasek menu jest paskiem rozwijanych, wielopoziomowych menu, z których można wybrać polecenia konieczne do ustawienia odpowiednich opcji zarówno tworzonej aplikacji, jak i samego Visual Basic.

Pasek narzędziowy Toolbar, jest szeregiem ikon, które tworzą skrót do odpowiednich pozycji menu. Zamiast rozwijać wielopoziomowe menu i wybierać odpowiednie elementy, można kliknąć ikonę, która spowoduje wykonanie tych samych czynności.

Okno Form (Object) jest głównym oknem aplikacji. W tym oknie tworzony jest cały wygląd aplikacji, rozmiar, kolor, rozmieszczenie elementów.

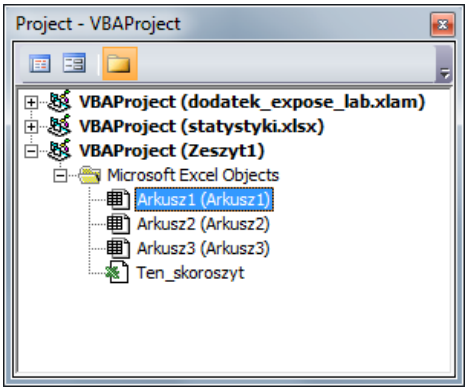
Okno Project jest oknem zawierającym informacje o wszystkich częściach i plikach aktualnie tworzonej aplikacji.

Okno Toolbox jest zbiorem kontrolek, które można umieszczać na formatce tworzonej aplikacji, aby wykonywały polecone im zadania.

Okno Properties panel, w którym można ustawić właściwości wykorzystywanych elementów, form, kontrolek, obiektów. Można tam ustawić cechy odpowiedzialne za wygląd, położenie, zachowanie poszczególnych elementów. Aby wybrać odpowiednią wartość właściwości, należy wpisać ją w odpowiednie okno lub wybrać jedną z pozycji rozwijanej listy.

Okno Code miejsce na kod programu. Kod ten to zbiór dostępnych rozkazów, które zostaną wykonane, gdy nastąpi określone zdarzenie, które spowoduje wykonanie odpowiedniej procedury i zawartej w niej kodu.

Okno Project – okno eksploratora projektu.



Zawartość projektu:

- Obiekty Excela
  - This Workbook – skoroszyt projektu
  - Arkusze
  - Wykresy
- Formularze – własne okna dialogowe
- Moduły – moduły kodu (standardowe)
- Moduły klas – moduły klas wykorzystywane przy tworzeniu własnych obiektów
- Odwołania do innych projektów

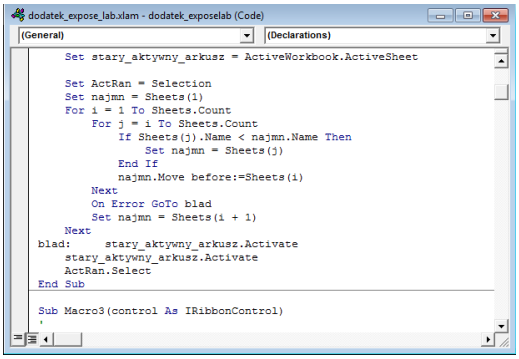
Okno Properties

Obiekty – poszczególne arkusze, wykresy, moduły, moduły klas oraz *This Workbook*

- Zbiory właściwości różnych obiektów są różne
- Nie wszystkie właściwości zawsze można zmieniać
- Nie wszystkie właściwości są zawsze widoczne

Okno Code

Możliwości wyświetlania:



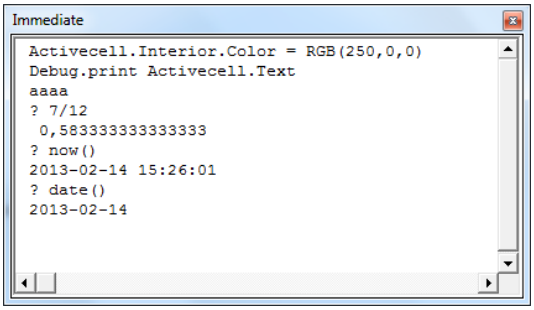
- Wszystkie procedury z modułu oddzielone poziomymi liniami
- Tylko jedna aktualnie wybrana procedura
- Okno podzielone na dwie niezależnie przewijane części
- Zawartości listy obiektów i listy procedur zależą od obiektu aktualnie wybranego do edycji

Edycja kodu

- Po przejściu do nowej linii automatycznie jest sprawdzana poprawność i dokonywane formatowanie (zamiana wielkości liter, stosowanie wcięć i kolorowanie składni)
- Znak podkreślenia '_' umożliwia kontynuowanie instrukcji w nowej linii, ale nie można dzielić w ten sposób ciągu znaków w cudzysłowach

Okno Immediate

Okno instrukcji bezpośrednich służy do wykonywania pojedynczych poleceń VBA. Wypisywanie tekstu następuje instrukcją Debug.Print. Umożliwia sprawdzanie wyniku wykonania instrukcji

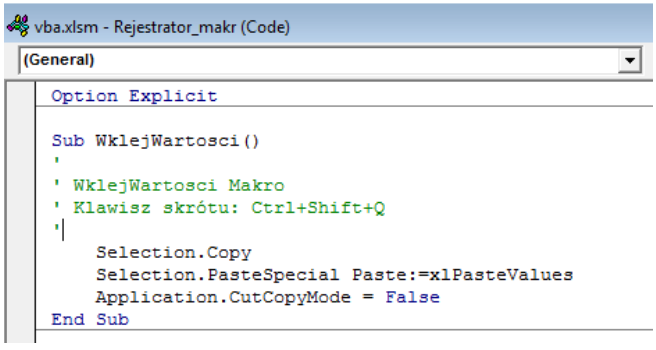


Okno Immediate jest bardzo przydatne do testowania poleceń i wyrażeń języka VBA. Na przykład po wprowadzeniu wyrażenia:

```
? Range("A1").Value
```

i naciśnięciu Enter w następnym wierszu okna Immediate zostanie wyświetlony wynik.

- **Znaczenie kolorów w edytorze**



kolor niebieski – słowa kluczowe VBA

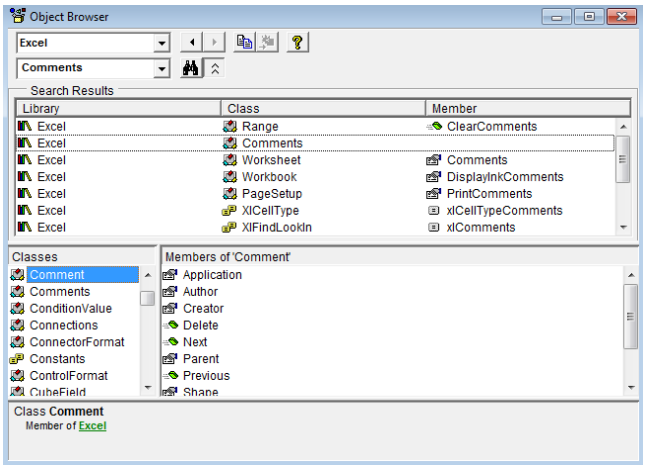
kolor zielony – wszystkie dodatkowe informacje, opisy – jednym słowem tekst własny programisty, w którym możemy wpisywać wszystkie uwagi. Zaczynają się one od górnego apostrofu ', kończą naciśnięciem Enter, czyli przejściem do następnej linii.

- **Przeglądarka obiektów**

Przeglądarka Obiektów, czyli **Object Browser** wyświetla wszystkie właściwości i metody dostępne dla danego obiektu. Po uaktywnieniu edytora Visual Basic przeglądarkę można uruchomić na jeden z trzech sposobów:

- Przyciskiem F2
- Z menu View wybrać polecenie Object Browser
- Klikając przycisk Object Browser znajdujący się na pasku narzędzi Standard

Okno przeglądarki Object Browser zostało przedstawione poniżej.



Lista rozwijalna w lewym górnym narożniku Object Browser zawiera wszystkie dostępne biblioteki obiektów:

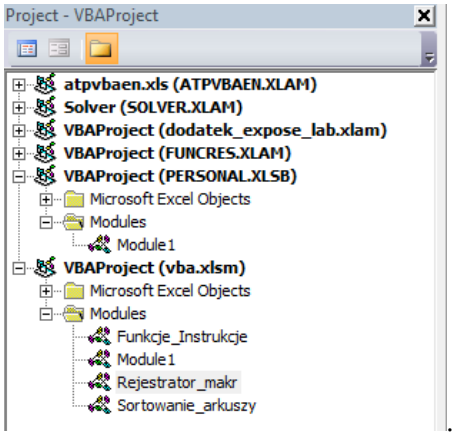
- Program Excel
- MSForms (używana do tworzenia niestandardowych okien dialogowych)
- Office (obiekty wspólne dla wszystkich aplikacji pakietu Microsoft Office)
- Stdole (obiekty automatyzacji OLE)
- VBA
- Bieżący projekt i wszystkie skoroszyty, do których się odwołuje.

Zawartość okna Classes zależy od elementu wybranego z listy. Z kolej zawartość okna Members Of zależy od elementu, który zostanie zaznaczony w oknie Clases.

Jeśli po wybraniu biblioteki chce się uzyskać listę właściwości i metod zawierających określony tekst, można go wprowadzić w polu szukania Search Text.

- **Moduły**

Ważnym elementem edytora VBA jest część Project, oznaczona na rysunku poniżej



Jeżeli nie widzisz w swoim edytorze ramki Project, kliknij kombinację klawiszy Ctrl+R.

W tej sekcji edytora przedstawione są wszystkie komponenty otwartego projektu VBA.

Każdy projekt VBA może składać się z następujących części:

- Microsoft Excel Object (obiekty Excela). Są to wszystkie elementy związane z normalną obsługą Excela, czyli arkusze i wykresy.
    - Czasem może się zdarzyć, że w edytorze VBA widocznych będzie więcej arkuszy niż w normalnym widoku Excela – wynika to z faktu, że niektóre arkusze mogą być ukryte i nie są widoczne w normalnym widoku, natomiast w edytorze VBA pokazane są wszystkie arkusze bez względu na to, czy są ukryte czy nie.
- Forms (formularze). Graficzne interfejsy użytkownika.
- Modules (moduły). Moduły to podstawowe i najbardziej uniwersalne obiekty do przechowywania kodu VBA.
- Class Modules (moduły klas). Najbardziej skomplikowany obiekt edytora VBA.

- **Komentowanie kodu**

Komentarz jest tekstem spełniającym funkcję opisu kodu źródłowego. VBA całkowicie ignoruje zawartość komentarza. Warto stosować komentarze do opisu przeznaczenia instrukcji, ponieważ nie zawsze jest to oczywiste.

Komentarz może zostać umieszczony w oddzielnym wierszu lub za instrukcją w tym samym wierszu. Komentarz jest rozpoczynany znakiem apostrofu. VBA ignoruje tekst napisany za apostrofem do końca wiersza.

Notatki własne:

3 Zmienne i stałe

Podstawowym zdaniem języka VBA jest przetwarzanie danych, Niektóre dane znajdują się w obiektach, takich jak zakresy arkusza. Inne są przechowywane w zadeklarowanych zmiennych.

Zmienna jest po prostu nazwanym obszarem przechowywania danych zawartym w pamięci komputera. Dla zmiennych może być definiowanych wiele typów danych, począwszy od prostego typu Boolean, a skończywszy na dużych wartościach o podwójnej precyzji typu Double. Wartość jest przypisywana zmiennej przy użyciu operatora przypisania mającego postać znaku równości.

Deklarowanie zmiennych:

Deklarowanie zmiennej jest to operacja polegająca na nadaniu jej nazwy oraz określeniu typu i dostępności. Jeżeli zadeklarujemy zmienną to jednocześnie przydzielamy jej pamięć. Zmienną możemy zadeklarować wewnątrz konkretnej procedury lub w sekcji deklaracji modułu kodu. Miejsce deklaracji ma wpływ na dostępność danej zmiennej.

Do deklarowania zmiennej zazwyczaj stosowane jest słowo kluczowe **Dim**. Instrukcja deklaracji, w której użyliśmy słowa kluczowego **Dim** może być umieszczona wewnątrz procedury, wówczas zostanie utworzona zmienna na poziomie procedury. Jeżeli natomiast deklaracja zostanie umieszczona na początku modułu w sekcji deklaracji, utworzona będzie zmienna na poziomie modułu. Poniżej przykład deklaracji, w którym deklarujemy zmienną o nazwie **MojaLiczba**.

```
Dim MojaLiczba
```

Oprócz deklarowania zmiennych za pomocą słowa kluczowego **Dim**, w deklarowaniu zmiennych możemy użyć słów kluczowych **Private**, **Public**, oraz **Static**. Słowa te służą nie tylko do deklarowania zmiennych, ale i do określania ich zakresu.

Zasady tworzenia nazw zmiennych:

- Maksymalnie 255 znaków;
- Pierwszy znak musi być literą;
- Można stosować znaki alfanumeryczne, liczby i niektóre znaki interpunkcyjne;
- Nie jest rozróżniana wielkość znaków;
- Nie można stosować spacji, przecinka i znaku kropki;
- Nie można umieszczać znaku deklarującego typ (%, &, ^, !, #, @, $);
- Nie można używać słów zastrzeżonych języka VBA (słowa kluczowe, instrukcje, funkcje, operatory).

Słowa kluczowe

As	Binary	ByRef	ByVal	Date	Else
Empty	Error	False	For	Friend	Get
Input	Is	Len	Let	Lock	Me
Mid	New	Next	Nothing	Null	On
Option	Optional	ParamArray	Print	Private	Property
Public	PtrSafe	Resume	Seek	Set	Static
Step	String	Then	Time	To	True
WithEvents					

Typy danych:

Deklarując zmienne możemy określić, jakiego typu dane zmienna będzie przechowywać. Deklaracja zmiennej to jednocześnie rezerwacja w pamięci komputera miejsca potrzebnego do przechowania wartości, która zostanie przypisana do zmiennej. Jeżeli podczas deklarowania zmiennej podamy jej typ określimy tym samym bardziej precyzyjnie ile miejsca VBA ma zarezerwować dla tej zmiennej. W poniższym przykładzie deklarujemy zmienną typu Integer która zajmuje 2 bajty pamięci. Dla porównania, ta sama zmienna zadeklarowana bez podania typu danych, będzie zajmować najmniej 16 bajtów.

```
Dim MojaLiczba As Integer
```

Typy danych przedstawiano w tabeli poniżej:

Typ danych	Liczba używanych bajtów	Zakres wartości
Boolean	2	wartość logiczna prawda lub fałsz wartości: Ture, False wartość domyślna False
Integer	2	liczba całkowita od -32 768 do 32 767 wartość domyślna 0
Long	4	liczba całkowita od -2 147 483 648 do 2 147 483 647 wartość domyślna 0
Single	4	liczba rzeczywista od -3,402823E38 do -1,401298E-45 i od 1,401298E-45 do 3,402823E38 wartość domyślna 0
Double	8	liczba rzeczywista dwukrotnie większa od Single od -1,79769313486232e+308 do -4,9406564581247e-324 i od 4,9406564581247e-324 do 1,79769313486232e+308 wartość domyślna 0

Typ danych	Liczba używanych bajtów	Zakres wartości
String	1 + 1 na każdy znak	tekst maksymalnie do 32 767 znaków wartość domyślna: "" – pusty
Date	8	data i czas w przedziale od 1 stycznia 100 r. do 31 grudnia 9999 roku wartość domyślna 0
Currency	8	typ walutowy, liczby rzeczywiste od -922 337 203 658 477,5808 do 922 337 203 685 477,5807 wartość domyślna 0
Variant	1 do 8	dowolny typ identyfikowany przez Visual Basic w zależności od pierwszego użycia tej zmiennej wartość domyślna: EMPTY (puste) domyślny typ dla nie zadeklarowanych zmiennych

Znaki deklarujące typ danych

Typ danych	Znak	Typ danych	Znak
Integer	%	Double	#
Long	&	LongLong	^
Currency	@	Single	!
String	$		

Przykład:

```
Dim KomputerNazwa$     'zmienna KomputerNazwa typu String
```

Zakres zmiennej:

Zakres zmiennej czyli to, w jakich częściach programu jest ona dostępna, określamy podczas jej deklarowania. Zakres ten zależy od:

- miejsca, w którym zmienna jest zadeklarowana, w sekcji deklaracji modułu czy wewnątrz konkretnej procedury.
- za pomocą jakiego słowa kluczowego tj. **Dim**, **Public**, **Private**, lub **Static**, zmienna została zadeklarowana.

Przykłady:

Dim MojaLiczba – instrukcja ta może być umieszczona wewnątrz procedury, wówczas zostanie utworzona zmienna na poziomie procedury. Jeżeli natomiast deklaracja zostanie umieszczona na początku modułu, w sekcji deklaracji, utworzona będzie zmienna na poziomie modułu.

Private MojaZmienna – stosowana na poziomie modułu do deklaracji zmiennych prywatnych oraz do przydziału pamięci. Zmienne te są dostępne tylko w tym module, w którym zostały zadeklarowane. Słowa kluczowego Private nie można użyć wewnątrz procedury.

Public WynikRazem – stosowana do deklarowania zmiennych publicznych na poziomie modułu. Zmienne zadeklarowane za pomocą instrukcji Public są dostępne dla wszystkich procedur we wszystkich modułach wszystkich projektów. Słowo kluczowe Public należy stosować wyłącznie w sekcji deklaracji modułu.

Static Licznik – wykorzystywana na poziomie procedury do deklaracji zmiennych i przydziału pamięci. Zadeklarowana w ten sposób zmienna zachowuje swoją wartość między wywołaniami procedury. Zmienne statyczne można deklarować tylko wewnątrz procedur.

Przypisanie wartości do zmiennej:

Aby zmienna mogła przechowywać pewne określone wartości musimy to wartość przypisać do zmiennej. Operacje przypisania wartości do zmiennej nazywamy instrukcją przypisania. Instrukcja przypisania składa się z nazwy zmiennej, znaku równości oraz wartości (lub wyrażenia określającego wartość), która ma być przypisana do zmiennej.

Przykłady:

MojaWartosc = 3 – Zmiennej o nazwie MojaWartosc przypisujemy wartość 3.

Przywitanie = "Pozdrawiam wszystkich" – przypisanie tekstu umieszczonego z prawej strony znaku równości do zmiennej Przywitanie. Łańcuchy znakowe przypisywane do zmiennych należy ujmować w cudzysłów.

Nazwisko = InputBox("Jak się nazywasz?") – w przykładzie przy użyciu funkcji InputBox przypisujemy wartość do, zmiennej Nazwisko.

Jawne deklarowanie:

Deklarowanie zmiennych za pomocą słów kluczowych **Dim**, **Private**, **Public**, oraz **Static** nazywamy jawnym deklarowaniem. Zmienna w języku Visual Basic może być też niejawnie zadeklarowana po prostu przez użycie jej w instrukcji przypisania. Wszystkie zmienne zadeklarowane niejawnie są typu **Variant**. Zmienne typu Variant wymagają więcej zasobów pamięci niż większość innych zmiennych. Jawne deklarowanie wszystkich zmiennych redukuje niebezpieczeństwo wystąpienia błędów wynikających z konfliktów nazw i pomyłek w pisowni. Aby uniknąć przykrych niespodzianek dobrze wyrobić sobie nawyk jawnego deklarowania wszystkich zmiennych. Bardzo pomocna w tym może okazać się instrukcja Option Explicit. Jeżeli w sekcji deklaracji modułu kodu wpiszesz **Option Explicit** – wyświetlony zostanie komunikat o błędzie, ilekroć wykryta zostanie niezadeklarowana zmienna. W takim przypadku możemy dodać brakującą deklarację. Instrukcja **Option Explicit** wykorzystywana jest na poziomie modułu w celu wymuszenia jawnego deklarowania wszystkich zmiennych w danym module.

Deklaracje i używanie stałych

- Stała to pewna wartość (niekoniecznie liczbowa) przypisana określonej nazwie.

- Deklaruje się ją podobnie do zmiennej z tą różnicą, że oprócz deklaracji typu nadaje się jej od razu wartość:

```
Const nazwa As typ = wartość
Const Rconst As Single = 8.31451
Const Firma As String = „Expose”
```

Po co definiować stałe?

- Porządek dla programisty – łatwiejsze zmiany i poprawki
- Skrócenie kodu
- Mniejsza szansa na błędy

Notatki własne:

4 Operatory

Operatory wykonują operacje logiczne, matematyczne, łańcuchowe, podstawienia lub porównania. VBA zapewnia pełen zestaw operatorów, co przedstawiono w poniższej tabeli:

Operatory matematyczne	
^	Operator potęgowania
-	Operator negacji
*****	Operator mnożenia
/	Operator dzielenia
****	Operator dzielenia całkowitego
Mod	Operator modulo
+	Operator dodawania
-	Operator odejmowania

Operatory porównania	
<	Operator mniejszości
<=	Operator mniejszości lub równości
>	Operator większości
>=	Operator większości lub równości
=	Operator równości
<>	Operator różności
Is	Operator porównania referencji
Like	Operator porównania łańcuchów

Operatory łańcuchowe	
+	Operator dodawania
&	Operator konkatenacji

Operatory logiczne	
Not	Operator negacji
And	Operator iloczynu logicznego, koniunkcji
Or	Operator sumy logicznej, alternatywy
Xor	Operator wyłączenia
Eqv	Operator równości
Imp	Operator implikacji, jeśli..to..

Operatory – kolejność

- Kolejność wykonywania operatorów: zmiana znaku, tożsamość, potęgowanie, mnożenie (*,/,\,mod), dodawanie (+/-), przypisanie (=);
- Nawiasy – kolejność wykonywania od najgłębiej zagnieżdżonego – nie ma ograniczeń w zagnieżdżeniu! Warto używać nawiasów dla pewności, że operacje będą wykonywane w żądanej kolejności;

Operacje na tekstach

- Konkatenacja – łączenie tekstów: + lub &

  np.: "Jan" + "Kowalski" => "JanKowalski„

  "Jan" & Chr(32) & "Kowalski" => „Jan Kowalski”

  "A" & Chr(98) & "c" => „Abc”

  1 + 2 = "3"

  1 & 2 = "12"

  (gdzie liczby mogą być też w postaci zmiennych typu string/variant)

Uwaga, operator + może być używany tylko do zmiennych tego samego typu (liczbowy/tekstowy), na potrzeby konkatenacji Variant liczy się jak string, o ile dodawany jest do stringa.

Operator logiczne – tablica prawdy:

x	y	Not x	Not y	x And y	x Or y	x Xor y	x Eqv y	x Imp y
0	0	1	1	0	0	0	1	1
0	1	1	0	0	1	1	0	1
1	0	0	1	0	1	1	0	0
1	1	0	0	1	1	0	1	1

Przykłady:

```
Dim Wynik
    Wynik = 10 * 3 ' Wynikiem jest 30
    MsgBox Wynik
```

```
Dim Wynik
    Wynik = 10 Mod 3 ' Wynikiem jest 1
    MsgBox Wynik
```

```
Dim Wynik
    Wynik = 10 – 3 ' Wynikiem jest 7
    MsgBox Wynik
    Wynik = -Wynik ' Wynikiem jest -7
    MsgBox Wynik
```

W poniższym przykładzie wyświetlane jest okno dialogowe, w którym użytkownik powinien wpisać swoje imię. Następnie wyświetlone jest okno komunikatu z tekstem powitania.

```
Dim imię, powitanie
    imię = InputBox("Podaj swoje imię")
    powitanie = "Witaj " & imię & " miłej zabawy"
    MsgBox powitanie
```

W poniższym przykładzie liczba z aktywnej komórki jest dzielona przez liczbę z komórki znajdującej si z prawej strony; wynik w kolejnej komórce.

```
Sub operatory1()

‘poniżej deklaracja zmienych:
Dim Warunek As Boolean
Dim Dzielna, Dzielnik, Iloraz

On Error GoTo problem

‘Przypisanie wartości do zmiennych:
Dzielna = ActiveCell.
Dzielnik = ActiveCell.Offset(0, 1).Value
Warunek = Dzielnik <> 0

If Warunek = True Then
    Iloraz = Dzielna/Dzielnik
    ActiveCell.Offset(0, 2) = Iloraz
Else
```

```
    MsgBox "Dzielenie przez zero, wprowadź poprawną
    wartość"
End If
Exit Sub
problem:
    MsgBox "Wystąpił błąd w programie. " & Err.Description
    & ", wprowadź poprawne wartości"
End Sub
```

W przykładzie obliczamy pole powierzchni. Na początku deklarujemy zmienne Długość, Szerokość i Pole, oraz zmienną Warunek typu Boolean. Wartość zmiennej Długość to zawartość komórki B2 arkusza Excela, natomiast wartość zmiennej Szerokość jest zawartością komórki D2 arkusza. Za pomocą operatora And sprawdzamy czy wartości zmiennej Długość i Szerokość są większe od zera.Za pomocą operatora Or określamy górne granice zmiennych Długość i Szerokość czyli górne granice długości boków. Jeżeli warunki są spełnione wykonywane jest mnożenie i obliczane pole powierzchni. Wynik wyświetlany jest w komórce F2.

```
Sub CommandButton1_Click()
Dim Warunek As Boolean
Dim Długość, Szerokość, Pole

On Error GoTo problem

Długość = Range("B2").Value
Szerokość = Range("D2").Value
Warunek = Długość > 0 And Szerokość > 0 'Operator And użyty
    w instrukcji przypisania.

If Warunek = True Then
    If Długość > 100 Or Szerokość > 100 Then 'Operatora Or
    użyty bezpośrednią w instrukcji warunkowej.
        MsgBox "Wprowadź poprawne wartości"
    Else
        Pole = Długość * Szerokość
        Range("F2").Value = Pole
    End If
Else
    MsgBox "Wprowadź wartości większe od zera"
End If

Exit Sub
problem:
    MsgBox "Wystąpił błąd w programie. " & Err.Description
    & "Wprowadź poprawne wartości"
End Sub
```

Notatki własne:

5 Instrukcje VBA

5.1 Instrukcje bezwarunkowe

Instrukcja	Składnia, przykład	Opis
GoTo	Składnia: GoTo linia Przykład: GoTo linia1 … Linia1:	Przekazanie wykonania dalszych instrukcji jako bezwarunkowy skok do określonego miejsca w obrębie procedury wyznaczonego przez argument linia, tj. etykiety lub numeru wiersza.
GoSub … Return	Składnia: GoSub linia … linia … Return	Przekazanie wykonania dalszych instrukcji jako bezwarunkowy skok do podprocedury w obrębie procedury określonej przez argument linia, tj. etykiety lub numeru wiersza, a następnie powrót do instrukcji Return.

5.2 Instrukcje warunkowe

Konstrukcja	Składnia, przykład	Opis
On … GoSub	Składnia: On wyrażenie GoSub docelowe_wiersze Przykład: Wiersz = 2 On wiersz GoSub Linia1, Linia2 *'skok do etykiety linia2* … Linia2: Return	Przekazuje sterowanie do jednego z kilku określonych miejsc w procedurze zależnie od wartości wyrażenia (po instrukcji **Return** powrót z wykonania). Wartość **wyrażenie** (liczba całkowita od 0 do 255) określa, do którego wiersza z listy zostanie przekazane sterowanie.

Konstrukcja	Składnia, przykład	Opis
On … GoTo	Składnia: On wyrażenie GoTo docelowe_wiersze Przykład: Wiersz = 1 On wiersz GoTo Linia1, Linia2 *'skok do etykiety Linia1* … Linia1: …	Przekazuje sterowanie do jednego z kilku określonych miejsc w procedurze zależnie od wartości wyrażenia (bez powrotu po wykonaniu przekazania sterowania). Wartość **wyrażenie** (liczba całkowita od 0 do 255) określa, do którego wiersza z listy zostanie przekazane sterowanie.
If … Then … Else …	Składnia: If warunek Then [instrukcje1] [Else instrukcje2] 'jeśli warunek jest spełniony to wykonywane są instrukcje1, jeśli nie to instrukcje2	Warunkowo wykonuje grupę instrukcji zależnie od wartości wyrażenia. Każdy **warunek** reprezentowany jest przez wyrażenie warunkowe, które może przyjmować dwie wartości: prawda (**True**) lub fałsz (**False**). Zależnie od wartości wykonywany jest określony blok instrukcji. Klauzule **Else** i **ElseIf** są opcjonalne. W wyrażeniu warunkowym można stosować operatory porównania i operatory logiczne. W przypadku zastosowania składni pojedynczej linii, możliwe jest umieszczenie wielu instrukcji do wykonania wyniku **If … Then** pod warunkiem, że wszystkie instrukcje będą umieszczone w tej samej linii oraz muszą być oddzielone znakiem :.
If … Then … **[ElseIf … Then …]** **[Else …] End If**	Składnia: If warunek Then [instrukcje] 'wykonywane, jeśli warunek jest spełniony, kolejna instrukcja to [ElseIf *warunek-n*] Then [Instrukcje] … 'wykonywane, jeśli warunek-n jest spełniony, kolejna instrukcja to End If [Else [instrukcje]] 'wykonywane, jeśli żaden warunek nie jest spełniony End If	

Konstrukcja	Składnia, przykład	Opis
Select Case	Składnia: Select Case wyrażenie_testowe [Case lista_wyrażeń-n [instrukcje-n]] ... [Case Else [instrukcje-domyślne]] ... End Select	Wykonanie jednego lub kilku bloków instrukcji zależnie od wartości podanego wyrażenia. Część **lista_wyrażeń-n** to ograniczona lista jednej lub kilku następujących form: **wyrażenie(Case 1, 4, 6)**; **wyrażenie To wyrażenie** (do określania zakresu wartości, **Case 10 To 100**); **Is wyrażenie porównawcze** (stosowane z operatorami: **<, >, >=, <=, =, <>, Case Is < 15**). Można stosować różne formy wyrażeń lub zakresów każdym warunku: **Case (Case 1 to 4, 7 to 10, 14, 17, Is > maxliczba**). Struktury **Select Case** mogą być zagnieżdżane Każda zagnieżdżona konstrukcja **Select Case** musi być zakończona instrukcją **End Select**.

5.3 Funkcje warunkowe

Funkcje	Składnia, przykład	Opis
IIf	Składnia: IIf(wyrażenie, część_prawda, część_fałsz)	Zwraca jedną z dwóch możliwych części (wartość lub wyrażenie) w zależności od sprawdzanego wyrażenia.
Choose	Składnia: Choose(indeks, wybór_1 [,wybór_2, ... [,wybór_n]])	Wybiera i zwraca wartość z listy argumentów. **Indeks** to wyrażenie numeryczne z zakresu od 1 do liczby możliwych wyborów.
Switch	Składnia: Switch(wyrażenie_1, wartość_1 [, wyrażenie_2, wartość_2 ... [,wyrażenie_n, wartość_n]])	Wyznacza listę wyrażeń i zwraca wartość typu **Variant** lub wyrażenie skojarzone z pierwszym wyrażeniem z listy, które jest prawdziwe.

5.4 Pętle

<table>
<tr><th>Konstrukcja</th><th>Składnia, przykład</th><th>Opis</th></tr>
<tr><td>For … Next</td><td>Składnia:
For licznik = start To koniec [Step krok]
[instrukcje]
[Exit For]
[instrukcje]
Next [licznik]</td><td>Powtarza instrukcję lub blok instrukcji określoną liczbę razy, gdzie: licznik – zmienna używana jako licznik pętli, start i koniec – wartość początkowa i końcowa licznika, opcjonalnie argument słowa kluczowego Step określa wartość zmian licznika dla każdego wykonania bloku instrukcji (wartość domyślna wynosi 1). Pętla może zawierać instrukcje Exit For umieszczone w dowolnym miejscu pomiędzy For … Next jako alternatywne wyjście z bloku pętli. Exit For przekazuje sterowanie do instrukcji bezpośrednio następujące po Next.</td></tr>
<tr><td rowspan="3">Do … Loop</td><td>Wariant I
Do [{While | Until} warunek]
[instrukcje]
[Exit Do]
[instrukcje]
Loop</td><td>Powtarza instrukcję lub blok instrukcji, dopóki warunek While ma wartość True (warunek jest spełniony) lub warunek Until nie jest spełniony (ma wartość False).</td></tr>
<tr><td>Wariant II
Do
[instrukcje]
[Exit Do]
[instrukcje]
Loop [{While | Until} warunek]</td><td>Warunek sprawdzany jest na końcu pętli (pętla zostanie wykonana przynajmniej jeden raz).</td></tr>
<tr><td colspan="2">Pętla może zawierać instrukcję Exit Do umieszczone w dowolnym miejscu pomiędzy Do … Loop jako alternatywne wyjście z bloku pętli. Exit Do przekazuje sterowanie do instrukcji bezpośrednio następującej po Loop.
W przypadku zagnieżdżonych pętli Do … Loop instrukcja Exit Do przekazuje sterowanie do pętli o jeden poziom powyżej pętli, w której nastąpiło przerwanie.</td></tr>
<tr><td>While … Wend</td><td>While warunek
[instrukcje]
Wend</td><td>Wykonuje instrukcję lub blok instrukcji, dopóki warunek będzie miał wartość True. Pętla może być zagnieżdżona.</td></tr>
</table>

Można umieszczać pętlę w obrębie drugiej pętli. Wewnętrzna pętla zostanie wykonana w całości (wszystkie instrukcje zawarte w pętli) w każdym cyklu pętli zewnętrznej. Można dowolnie zagnieżdżać pętle bez względu na ich rodzaj.

Nieskończone pętle można przerwać kombinacją klawiszy – ctrl+break.

Przykłady:

Instrukcja warunkowa IF i jej warianty

```
Sub GreetMe1()
    If Time > 0.5 Then MsgBox "Witam przed południem!"
End Sub
```

```
Sub GreetMe2()
    If Time < 0.5 Then MsgBox "Witam przed południem!"
    If Time >= 0.5 Then MsgBox "Witam przed południem!"
End Sub
```

```
Sub GreetMe3()
    If Time < 0.5 Then
        MsgBox "Witam przed południem!"
        ' pozostałe polecenia
    Else
        MsgBox "Witam! Już po południu!"
        ' pozostałe polcenia
    End If
End Sub
```

```
Sub GreetMe4()
    If Time < 0.5 Then MsgBox "Witam przed południem"
    If Time >= 0.5 And Time < 0.75 Then MsgBox "Witaj! Już
    po południu!"
    If Time >= 0.75 Then MsgBox "Dobry wieczór!"
End Sub
```

```
Sub GreetMe5()
    If Time < 0.5 Then
        MsgBox "Witam przed południem!"
    ElseIf Time >= 0.5 And Time < 0.75 Then
        MsgBox "Witaj! Już po południu!"
    Else
        MsgBox "Dobry wieczór!"
    End If
End Sub
```

```
Sub Discount1()

Dim Quantity As Variant
Dim Discount As Double

Quantity = InputBox("Wprowadź liczbę kupowanych książek: ")
If Quantity = "" Then Exit Sub

    If Quantity >= 0 Then Discount = 0.1
    If quantoty >= 25 Then Discount = 0.15
    If Quantity >= 50 Then Discount = 0.2
    If Quantity >= 75 Then Discount = 0.25

MsgBox "Rabat: " & Discount
End Sub
```

Pętle FOR

Suma pierwiastków kwadratowych pierwszych 100 liczb całkowitych

```
Sub SumPierw()

Dim Sum As Double
Dim Count As Integer

Sum = 0
    For Count = 1 To 100
        Sum = Sum + Sqr(Count)
    Next Count
    MsgBox "Suma kwadratów 100 liczb całkowitych wynosi: " &
    Sum
End Sub
```

Usuwa z aktywnego skoroszytu 2, 4, 6, 8, oraz 10 wiersz

```
Sub DelRows()

Dim RowNum As Long
    For RowNum = 10 To 2 Step -2
        Rows(RowNum).Delete
    Next RowNum
End Sub
```

Wypełnianie od 1 do 100 od aktywnej komórki

```
Sub GoodLoop()

Dim Start As Integer
Dim NumToFill As Integer
Dim cnt As Integer

Start = 1
NumToFill = 100
    For cnt = 0 To NumToFill - 1
        ActiveCell.Offset(cnt, 0).Value = Start + cnt
    Next cnt
End Sub
```

Pętla WHILE (Do While)

Wypełnia komórki datami z obecnego miesiąca

```
Sub EnterDate1()
Dim TheDate As Date

TheDate = DateSerial(Year(Date), Month(Date), 1)
    Do While Month(TheDate) = Month(Date)
        ActiveCell = TheDate
        TheDate = TheDate + 1
        ActiveCell.Offset(1, 0).Activate
    Loop
End Sub
```

Otwiera plik tekstowy, odczytuje kolejne wiersze, zamienia małe litery na wielkie i zapisuje je w aktywnym arkuszy, począwszy od komórki A1 w dół kolumny

```
Sub DoWhileDemo1()
Dim LineCt As Long
Dim LineOfText As String

Open "c:\data\plik_tekstowy.txt" For Input As #1
LineCt = 0
    Do While Not EOF(1)
        Line Input #1, LineOfText
        Range("A1").Offset(LineCt, 0) = UCase(LineOfText)
        LineCt = LineCt + 1
    Loop
Close #1
End Sub
```

Instrukcja SELECT CASE

```
Sub GreetMe6()
Dim Msg As String

Select Case Time
    Case Is < 0.5
        Msg = "Witam przed południem!"
    Case 0.5 To 0.75
        Msg = "Witaj! Już po południu!"
    Case Else
        Msg = "Dobry wieczór!"
End Select
MsgBox Msg
End Sub
```

```
Sub Discount2()

Dim Quantity As Variant
Dim Discount As Double

Quantity = InputBox("Wprowadź liczbę kupowanych książek: ")

Select Case Quantity
    Case ""
        Exit Sub
    Case 0 To 24
        Discount = 0.1
    Case 24 To 49
        Discount = 0.15
    Case 50 To 74
        Discount = 0.2
    Case Is >= 75
        Discount = 0.25
End Select

MsgBox "Rabat: " & Discount
End Sub
```

Czy dzisiejszy dzień to sobota lub niedziela?

```
Sub Weekend()

Select Case Weekday(Now)
    Case 1, 7
```

```
        MsgBox "Mamy już weekend!"
    Case Else
    MsgBox "To jeszcze nie weekend!"
End Select

End Sub
```

Notatki własne:

6 Funkcje VBA

Funkcja to specyficzna procedura języka VBA wykonująca określone obliczenia i zwracająca wartość będącą ich rezultatem. Podstawową, a tak naprawdę jedyną, różnicą pomiędzy procedurą a funkcją jest fakt, że funkcja potrafi i powinna zwrócić jakąś wartość, zmienną lub obiekt.

Funkcje matematyczne

Funkcja	Opis
Rnd	liczba losowa
Abs (X)	wartość bezwzględna
Sgn (X)	znak liczby
Fix (X)	część całkowita
Int (X)	część całkowita
frac(X) = X – Fix(X)	część ułamkowa
Log (X)	logarytm naturalny
LogN(X) = Log(X) / Log (N)	logarytm o podstawie N
Exp (X)	e do potęgi
Sqr (X)	pierwiastek kwadratowy
Hex (X)	wartość Hex
Oct (X)	wartość Oct
Sin (X)	sinus
Cos (X)	cosinus
Tan (X)	tangens

Funkcje daty i czasu

Funkcja	Opis
Date	zwraca aktualną datę
DateAdd	dodaje dwie daty
DateDiff	liczy czas między dwoma datami
DatePart	dzieli datę na części
DateSerial	składa datę z części
DateValue	zwraca datę z wyrażenia
Day	zwraca dzień z daty

Funkcja	Opis
Format	formatowanie daty i czasu
Hour	zwraca godzinę z czasu
Minute	zwraca minutę z czasu
Month	zwraca miesiąc z daty
Now	zwraca bieżąca data i czas
Second	zwraca sekundy z czasu
Time	zwraca bieżący czas
Timer	zwraca czas od północy
TimeSerial	składa czas z części
TimeValue	zwraca czas z wyrażenia
Weekday	zwraca dzień tygodnia z daty
Year	zwraca rok z daty

Funkcje tekstowe

Funkcja	Opis
Asc	znak na kod ASCII
AscB	znak na kod ASCII
AscW	znak na kod Unicode
Chr	kod ASCII na znak
ChrB	kod ASCII na znak
ChrW	kod Unicode na znak
Format	formatowanie łańcuchów
Val	łańcuch na wartość numeryczną
Str	wartość numeryczna na łańcuch
LCase	łańcuch na małe litery
UCase	łańcuch na duże litery
StrConv	konwertuje łańcuch wg specyfikacji
Len	zwraca długość łańcucha
LTrim	usuwa początkowe spacje
RTrim	usuwa końcowe spacje
Trim	usuwa początkowe i końcowe spacje
Left	zwraca część łańcucha od lewej strony

Funkcja	Opis
Right	zwraca część łańcucha od prawej strony
Mid	zwraca część łańcucha od wybranego miejsca
MidB	zwraca część łańcucha od wybranego miejsca
Space	zwraca łańcuch wypełniony spacjami
String	zwraca łańcuch wypełniony znakami
StrComp	porównuje dwa łańcuchy
InStr	szuka jednego łańcucha w drugim

Funkcje formatujące

Funkcja	Opis
Format	formatowanie wyrażeń
LCase	łańcuch na małe litery
UCase	łańcuch na duże litery
StrConv	formatowanie tekstu
LTrim	usuwa początkowe spacje
RTrim	usuwa końcowe spacje
Trim	usuwa początkowe i końcowe spacje
Left	zwraca część łańcucha od lewej strony
LeftB	zwraca część łańcucha od lewej strony
Right	zwraca część łańcucha od prawej strony
RightB	zwraca część łańcucha od prawej strony
Mid	zwraca część łańcucha od wybranego miejsca
MidB	zwraca część łańcucha od wybranego miejsca
Space	zwraca łańcuch wypełniony spacjami
String	zwraca łańcuch wypełniony znakami

Funkcje wejścia-wyjścia

Funkcja	Opis
MsgBox	wyświetla okno komunikatu
InputBox	wyświetla okno wprowadzania danych
Command	zwraca argumenty wywołania z linii komend
Shell	uruchamia zewnętrzny program

Environ	zwraca wartość zmiennej systemowej
CurDir	zwraca bieżący katalog lub folder
Dir	zwraca nazwę szukanego pliku lub folderu
FreeFile	zwraca wolny numer pliku
FileLen	zwraca rozmiar pliku
LOF	zwraca rozmiar otwartego pliku
Loc	określa pozycję odczytu/zapisu
Seek	zwraca pozycję odczytu/zapisu
EOF	sprawdza czy osiągnięto koniec pliku
FileAttr	zwraca tryb otwarcia pliku
Input	odczytuje dane z pliku
SetAttr	ustawia atrybuty pliku
GetAttr	zwraca atrybuty pliku
FileDateTime	zwraca datę i czas ostatniej modyfikacji pliku
LoadPicture	ładuje obraz graficzny do obiektów

Funkcje konwersji danych

Funkcja	Opis
Array	Variant na tablicę
Asc	znak na kod ASCII
AscB	znak na kod ASCII jednobajtowy
AscW	znak na kod Unicode
Chr	kod ASCII na znak
ChrB	kod ASCII jednobajtowy na znak
ChrW	kod Unicode na znak
Str	liczba na tekst
StrConv	formatowanie tekstu
Val	tekst na liczbę
Oct	liczba na oct
Hex	liczba na hex
CBool	wartość na Boolean
CByte	wartość na Byte
CCur	wartość na Currency

CDate (CVDate)	wartość na Date
CDbl	wartość na Double
CDec	wartość na Decimal
CInt	wartość na Integer
CLng	wartość na Long
CSng	wartość na Single
CStr	wartość na String
CVar	wartość na Variant

Funkcje testujące dane

Funkcja	Opis
IIf	sprawdza wartość wyrażenie i zwraca jedną z dwóch podanych wartości
IsArray	sprawdza czy zmienna jest tablicą
LBound	sprawdza dolny zakres tablicy
UBound	srawdza górny zakres tablicy
IsDate	sprawdza czy wartość jest datą
IsNumeric	sprawdza czy wartość numeryczna
IsEmpty	sprawdza czy zmienna ma wartość Empty
IsNull	sprawdza czy wyrażenie ma wartość Null
IsMissing	sprawdza czy przekazano opcjonalny argument procedury
VarType	zwraca typ zmiennej
TypeName	zwraca typ zmiennej

Inne funkcje

Funkcja	Opis
Error	zwraca komunikat błędu o podanym numerze
DoEvents	przekazuje sterowanie do systemu operacyjnego
QBColor	zwraca wartość RGB podanego koloru
RGB	zwraca wartość koloru RGB

- **Korzystanie z gotowych funkcji VBA**

Wiele popularnych operacji w VBA nie wymaga tworzenia własnych funkcji, ponieważ w języku tym wbudowanych jest wiele gotowych funkcji, które z powodzeniem mogą być wykorzystane w kodzie. Poniżej przedstawiono tabelę z gotowymi funkcjami VBA

6.1 Funkcje dotyczące typów danych

Funkcje określające typ danych

Funkcja	Opis
TypeName	Zwraca łańcuch znaków typu String określający typ danej zmiennej.
VarType	Zwraca wartość typu Integer reprezentującą podtyp zmiennej.

Funkcje sprawdzające typ danych

Funkcja	Opis
IsArray	Zwraca True, jeśli zmienna jest tablicą.
IsDate	Zwraca True, jeśli wyrażenie może zostać przekonwertowane do typu Date.
IsEmpty	Zwraca True, jeśli zmienna nie została zainicjowana lub została ustalona, jako Empty.
IsError	Zwraca True, jeśli wyrażenie ma wartość Error.
IsMissing	Zwraca True, jeśli do procedury nie przekazano opcjonalnego argumentu.
IsNull	Zwraca True, jeśli wyrażenie zawiera wartość Null.
IsNumeric	Zwraca True, jeśli wyrażeniu można nadać wartość liczby.
IsObject	Zwraca True, jeśli w wyrażeniu występuje odwołanie do obiektu.

Funkcje zwracają typ danych Boolean, określając, czy określane wyrażenie, występujące jako argument funkcji, jest danego typu i może zostać przekonwertowane na określony typ.

Funkcje konwersji typów danych

Funkcja	Zakres dla wyrażenia (argumentu)
CBool	Przekształca wyrażenie na typ danych Boolean.
CByte	Przekształca wyrażenie na typ danych Byte.
CCur	Przekształca wyrażenie na typ danych Currency.
CDate	Przekształca wyrażenie na typ danych Date.
CDbl	Przekształca wyrażenie na typ danych Double.
CDec	Przekształca wyrażenie na typ danych Decimal.

Funkcja	Zakres dla wyrażenia (argumentu)
CInt	Przekształca wyrażenie na typ danych Integer.
CLng	Przekształca wyrażenie na typ danych Long.
CLngLng	Przekształca wyrażenie na typ danych LongLong.
CLngPtr	Przekształca wyrażenie na typ LongPtr.
CSng	Przekształca wyrażenie na typ danych Single.
CStr	Przekształca wyrażenie na typ danych String.
CVar	Przekształca wyrażenie na typ danych Variant.
CVErr	Zwraca typ danych Variant (podtyp Error) zawierający numer błędu zdefiniowany przez użytkownika.

Nazwa funkcji określa zwracany typ danych. Wyrażenie jako argument funkcji jest ciągiem znaków lub wyrażeniem numerycznym. Jeżeli wyrażenie przekazywane do funkcji wykracza poza zakres dla danego typu danych, wówczas podczas konwersji generowany jest błąd.

Funkcje konwersji pomiędzy różnymi typami danych

Funkcja	Opis	
Int, Fix	Zwraca całkowitą część liczby (typu Integer). Różnica pomiędzy funkcją Int a Fix polega na różnicy konwersji liczb ujemnych.	
Hex	Zwraca ciąg znaków (typ String) reprezentujący liczbę w postaci szesnastkowej.	
Oct	Zwraca daną typu Variant (String) reprezentującą liczbę w postaci ósemkowej	
Funkcja	Przykład składni	Opis
Format	Składnia: Format(wyrażenie[,format[, pierwszy_dzień_tygodnia[, pierwszy_tydzień_roku]]])	Zwraca daną typu Variant (String) zawierającą wyrażenie sformatowane zgodnie z instrukcjami zawartymi w wyrażeniu format. Argumenty pierwszy_dzień_tygodnia oraz pierwszy_tydzień_roku określone są przez wartości i stałe.

6.2 Data i czas

Do przechowywania dat i danych związanych z czasem przeznaczona jest zmienna typu Date. Dostępny jest zakres dla zmiennej typu Date od 1 stycznia 100 roku do 31 grudnia 9999 roku. Daty i czas należy umieszczać pomiędzy znakami #. Zmienne typu Date zawsze są definiowane przy użyciu formatu **miesiąc/dzień/rok.**

```
Dim Data_Czas_Poczatek As Date
    Data_Czas_Poczatek = #12/11/2011 2:00:00 PW
```

```
    Data_Czas_Poczatek = #09/30/2011#
    Data_Czas_Poczatek = #12:00:00#
```

Format wyświetlanej daty i czasu zależy od formatu obowiązującego w systemie operacyjnym (ustawienia systemowe)

Instrukcje Date oraz Time służą do ustawienia bieżącej daty i czasu systemowego.

```
Date = #11/2/2011#     'zmiana czasu systemowego na 2
    listopada 2011 roku
Time = #2:10:15 PM" 'zmiana czasu systemowego na 14:10:15
```

Funkcje dotyczące daty i czasu

Funkcja	Składnia, przykład	Opis
DateAdd	Składnia: DateAdd(interwał, liczba, data) Przykład: DateAdd("yyyy", 89, #1/1/2011#) 'zwraca 2100-01-01	Zwraca daną typu Variant(Date) zawierającą datę, do której został dodany określony przedział czasu. Argument interwał określa przedział czasu, jaki będzie użyty do obliczeń.
DateDiff	Składnia: DateDiff(interwał, data1, data2[, pierwszy_dzień_tygodnia[, pierwszy_tydzień_roku]]) Przykład: DateDiff("yyyy", #1/1/2011", #1/1/2100#)	Zwraca daną typu Variant(Long) określającą liczbę określonego przedziału czasu pomiędzy dwoma określonymi datami. Argument interwał określa przedział czasu, jaki będzie użyty do obliczeń.
DatePart	Składnia: DatePart(interwał, data[, pierwszy_dzień_tygodnia[, pierwszy_tydzień_roku]]) Przykład: DatePart("yyyy", Date) 'zwraca "2012"	Zwraca daną typu Variant(Integer) określającą część danej daty. Argument interwał określa przedział czasu jaki będzie zwracany.
DateSerial	Składnia: DateSerial(rok, miesiąc, dzień) Przykład: DateSerial(2003, 4, 27)	Zwraca daną typu Variant(Date) dla określonego roku, miesiąca i dnia.
TimeSerial	Składnia: TimeSerial(godzina, minuta, sekunda) Przykład: TimeSerial(7,45,10)	Zwraca daną typu Variant(Date) zawierającą czas dla określonej godziny, minuty i sekundy.

Funkcja	Składnia, przykład	Opis
FormatDateTime	Składnia: FormatDateTime(data, [,Nazwany_Format]) Przykład: FormatDateTime(Time, vbShortTime)	Zwraca wyrażenie sformatowane jako data lub godzina. Argument Nazwany_Format określa, jaki format wyświetlania daty/czasu będzie użyty.
MonthName	Składnia: MonthName(miesiąc [,skrót]) Przykład: MonthName(1, True)	Zwraca miesiąc w postaci ciągów znaków. Argument skrót typu Boolean określa, czy nazwa miesiąca będzie zwrócona w formie pełnej, czy skróconej.
Weekday	Składnia: Weekday(data [,pierwszy_dzień_tygodnia]) Przykład: Weekday (Date, 1) 'zwraca "5"	Zwraca daną typu Variant(Integer) zawierającą liczbę całkowitą reprezentującą dzień tygodnia.
WeekdayName	Składnia: WeekdayName(dzień_tygodnia [,skrót[,pierwszy_dzień_tygodnia]]) Przykład: WeekdayName(3, False, vbSunday)	Zwraca łańcuch znaków reprezentujący dzień tygodnia. Argument skrót typu Boolean określa, czy nazwa dnia tygodnia będzie zwrócona w formie pełnej, czy skróconej.
Funkcja	Opis	
Date	Zwraca daną typu Variant(Date) zawierającą bieżącą datę systemową.	
DateValue	Zwraca daną typu Variant(Date). Przekształca ciąg znaków na datę.	
TimeValue	Zwraca daną typu Variant(Date) zawierającą czas. Przekształca łańcuch znaków na liczbę reprezentującą czas.	
Day	Zwraca daną typu Variant(Integer) określającą liczbę całkowitą z zakresu 1-31, reprezentującą dzień miesiąca.	
Hour	Zwraca daną typu Variant(Integer) określającą liczbę całkowitą z zakresu 0-23, reprezentującą godzinę dnia.	
Minute	Zwraca daną typu Variant(Integer) określającą liczbę całkowitą z zakresu 0-59, reprezentującą minutę godziny.	
Month	Zwraca daną typu Variant(Integer) określającą liczbę całkowitą z zakresu 1-12, reprezentującą miesiąc roku.	
Now	Zwraca daną typu Variant(Date) określającą bieżącą datę i godzinę zgodnie z datą i godziną systemową komputera.	
Second	Zwraca daną typu Variant(Integer) określającą liczbę całkowitą z zakresu 0-59, reprezentującą sekundę minuty.	

Funkcja	Składnia, przykład	Opis
Time	Zwraca daną typu Variant(Date) zawierającą bieżący czas systemowy.	
Timer	Zwraca daną typu Single reprezentującą liczbę sekund, jaka upłynęła od północy.	
Year	Zwraca daną typu Variant(Integer) zawierającą liczbę całkowitą reprezentującą rok.	

Tworzenie własnych funkcji

Słowem kluczowym otwierającym funkcję jest Function. Po tym słowie następuje nazwa funkcji. Następnie należy w nawiasie wymienić argumenty niezbędne do obliczenia tej funkcji wraz z podaniem ich typu. Oczywiście czasem zdarzają się funkcje, które nie posiadają żadnych argumentów wejściowych – wówczas nawias pozostaje pusty.

Ostatnim elementem wiersza otwarcia funkcji jest podanie słowa kluczowego As oraz typu wartości jaki jest zwracany przez tę funkcję.

Ogólna postać wiersza otwarcia funkcji wygląda więc następująco:

```
Function nazwaFunkcji([arg1 As typ, ..., argN As typ]) [As
                        typFunkcji]
```

Przykłady:

- **Funkcje**

```
Public Function Prowizja(Wart)

'Przypisanie funkcji do odpowiedniej kategorii w oknie
‘wstawianie funkcji: '0-wszystkie, 1-finansowe, 2-daty
‘i czasu, 3-matematyczne, 4-statystyczne, 5-wyszukiwania
‘i adresu, 6-bazy danych,7-tekstowe, 8-logiczne, 9-
‘informacyjne, 10-16 – niestandardowe (polecenia,
‘dostosowywania, sterowania makrami, dde, u zytkownika,
‘inżynierskie, modułowe)

Application.MacroOptions Macro:="Prowizja",
    Description:="Oblicza prowizję od sprzedaży",
    Category:= 10

‘definiowanie stałych
Const Poz1 = 0.08
Const Poz2 = 0.105
Const poz3 = 0.12
Const poz4 = 0.14
```

```
‘Obliczanie prowizji od sprzedaży
Select Case Wart
    Case 0 To 9999.99: Prowizja = Wart * Poz1
    Case 10000 To 19999.99: Prowizja = Wart * Poz2
    Case 20000 To 29999.99: Prowizja = Wart * poz3
    Case Is > 30000: Prowizja = Wart * poz4
End Select

End Function
```

```
Sub LiczProwizje()
‘Procedura korzystająca z funkcji Prowizja
‘Wartość sprzedaży podawana przez użytkownika

Dim Wart As Long

Wart = InputBox("wprowadź wartość sprzedaży:")
MsgBox "Prowizja wynosi " & Prowizja(Wart)
End Sub
```

```
Sub LiczProwizje2()

Dim Wart As Long
Dim Msg As String, Ans As String

'Prośba o podanie wartości sprzedaży
Wart = Val(InputBox("Wprowadź wartość sprzedaży:",
    "Kalkulator prowizji"))

'Tworzenie komunikatu
Msg = "Wartość sprzedaży:" & vbTab & Format(Wart, "#.##0.00
    zł")
Msg = Msg & vbCrLf & "Prowizja:" & vbTab
Msg = Msg & Format(Prowizja(Wart), "#.##.0.00 zł")
Msg = Msg & vbCrLf & vbCrLf & "Jeszcze raz?"

'Wyświetla wynik i prosi o podanie kolejnej wartości
Ans = MsgBox(Msg, vbYesNo, "Kalkulator prowizji")

If Ans = vbYes Then LiczProwizje2
End Sub
```

```
Sub WprWart()
'Użytkownik wprowadza wartość liczbową z zakresu 1 – 12
'Błędna wartość jest ignorowana a okno wyświetlane ponownie

Dim TekstUzyt As Variant
Dim Msg As String

Const MinWart = 1
Const MaxWart = 12

Msg = "Podaj liczbę z zakresu od " & MinWart & " do " &
    MaxWart

Do
    TekstUzyt = InputBox(Msg)
        If TekstUzyt = "" Then Exit Sub
           If IsNumeric(TekstUzyt) Then
                If TekstUzyt > MinWart And TekstUzyt <
    MaxWart Then Exit Do
           End If
    Msg = "Wprowadzona wartość jest nieprawidłowa"
    Msg = Msg & vbNewLine
    Msg = Msg & "Wprowadź wartość z zakresu od " & MinWart
    & " do " & MaxWart
Loop

ActiveSheet.Range("A1").Value = TekstUzyt
End Sub
```

Funkcje osłonowe

```
Function ExcelDir() As String
'zwraca nazwę katalogu, w którym jest zainstalowany excel

ExcelDir = Application.Path
End Function
```

```
Function SheetCount() As Integer
'zwraca liczbę arkuszy

SheetCount = ActiveWorkbook.Sheets.Count
End Function
```

```
Function SheetName() As String
'zwraca nazwę skoroszytu

SheetName = ActiveWorkbook.Name
End Function
```

```
Function LastPrinted()

Application.Volatile
LastPrinted = ThisWorkbook.BuiltinDocumentProperties("Last
    Print Name")
End Function
```

```
Function LastSaved()

Application.Volatile
LastSaved = ThisWorkbook.BuiltinDocumentProperties("Last
    Save Time")
End Function
```

Notatki własne:

7 Procedury

Grupa instrukcji realizująca określone zadania. Zasady nadawania nazw procedurom są takie same jak w przypadku nazw zmiennych.

7.1 Procedura typu Sub

Procedura typu **Sub** nie zwraca żadnej wartości. Procedury przechowywane są w modułach VBA. Składnia deklaracji:

```
[Private |Public |Friend] [Static] Sub NazwaProcedury
     ([lista_argumentów])
     [instrukcje procedury]
     [Exit Sub]
     [instrukcje procedury]
End Sub 'koniec procedury
```

7.2 Zasięg procedury typu Sub

Domyślnie wszystkie procedury są publiczne, tzn. dostępne dla innych procedur z innych modułów. Dostępność procedury określona jest przez słowa kluczowe: [Private | Public | Friend] – Private oznacza, że procedura dostępna jest tylko dla procedur z tego samego modułu; Public (wartość domyślna) – dostępna dla procedur z innych modułów, Friend (tylko w modułach klasy) – wskazuje, że procedura jest widoczna w projekcie, a nie jest widoczna dla kontrolera instancji obiektu.

7.3 Zmienne procedury typu Sub

Zmienne deklarowane w obrębie procedury tracą swoją wartość po zakończeniu działania procedury (End Sub) bądź w przypadku wymuszonego jej zakończenia (Exit Sub). Użycie słowa kluczowego Static wskazuje, że zmienne lokalne procedury są zapamiętywane pomiędzy kolejnymi wywołaniami procedury.

7.4 Wywołanie procedury typu Sub

Procedury typu Sub mogą być wywołane na wiele sposobów (m. in. z poziomu okna dialogowego Makro, za pomocą niestandardowego menu, jako zdarzenie obiektu, z poziomu okna Immediate i przede wszystkim z innej procedury).

7.4.1 **Wywołanie procedury w tym samym module**

```
Nazwa_procedury argument1, argument2, …argument-n

Call nazwa_procedury(argument1, argument2, … argument-n)

Application.Run "nazwa_procedury", arg1, arg2, … arg-n
```

Przykład:

```
Rysowanie ”test.xls”

Call Rysowanie (”test.xls”)

Application.Run ”Rysowanie”, ”test.xls”
```

7.4.2 Wywołanie procedury umieszczonej w innym module

```
Nazwa_modulu.nazwa_procedury arg1, arg2, … arg-n

Call nazwa_modulu.nazwa_procedury (arg1, arg2, … arg-n)

Application.Run ”nazwa_modulu.nazwa_procedury”, arg1, arg2,
… arg-n
```

Przykład:

```
Module1.Rysowanie ”test.xls”

Call Module1.Rysowanie (”test.xls”)

Application.Run ”Module1.Rysowanie”, ”test.xls”
```

7.4.3 Wywołanie procedury umieszczonej w innym skoroszycie

```
nazwa_projektu.nazwa_modulu.nazwa_procedury argument1,
argument2, … argument-n

Call nazwa_projektu.nazwa_modulu.nazwa_procedury (argument1,
argument2, … argument-n)

Application.Run nazwa_projektu.nazwa_modulu.nazwa_procedury”,
argument1, argument2, … argument-n
```

Przykład:

```
Projekt1.Module1.Rysowanie ”test.xls” ‘konieczność
utworzenia odwołania do Projektu1 (edytor VBE – menu Tools –
polecenie References…)

Call Projekt1.Module1.Rysowanie (”test.xls”) ‘konieczność
utworzenia odwołania do Projektu1 (edytor VBE – menu Tools –
polecenie References…)
```

```
Application.Run "D:\plik001.xls'!Module1.Rysowanie",
"test.xls" 'brak konieczności tworzenia odwołania do
Projektu1 (menu Tools – polecenie References…)
```

Wymuszenie dostępności wszystkich procedur w danym module jako Private następuje poprzez Option Private Module (nawet dla procedur zadeklarowanych za pomocą słowa kluczowego Public)

7.5 Procedura typu Function

Procedura typu Function, zwana również funkcją, wykonuje określone obliczenia oraz zwraca pojedynczą wartość lub tablicę. Mogą być stosowane w modułach i formułach arkuszy.

Składnia deklaracji:

```
[Private |Public| Friend] [Static] Function NazwaFunkcji
([lista_argumentów]) [As typ]
      [instrukcje procedury]
      [NazwaFunkcji = wyrażenie]
      [Exit Function] 'natychmiastowe wyjście z funkcji
      [instrukcje procedury]
      [NazwaFunkcji = wyrażenie]
End Function 'koniec procedury
```

7.6 Zasięg procedury typu Function

Domyślnie wszystkie procedury typu Function są publiczne, tzn. są dostępne dla innych procedur z innych modułów. Dostępność procedury określana jest przez słowa kluczowe: [Private | Public | Friend] – Private oznacza, że procedura dostępna jest tylko dla procedur z tego samego modułu, a ponadto nie jest widoczna w oknie dialogowym Wstawianie funkcji, Public (wartość domyślna) – dostępna dla procedur z innych modułów, Friend (tylko w modułach klasy) – wskazuje, że procedura jest widoczna w projekcie, a nie jest widoczna dla kontrolera instancji obiektu.

7.7 Wartości zwracane

Wartość zwracaną przez funkcję należy przypisać nazwie funkcji (w trakcie procedury można wielokrotnie dokonywać takiej operacji). Domyślnym typem zwracanym przez funkcję jest typ Variant.

7.8 Przekazywanie argumentów

Argumentami mogą być zmienne, stałe literały lub wyrażenia. Argumenty przekazywane do procedur zawarte są w części [*lista_argumentów*]. Poszczególne zmienne oddzielają od siebie przecinki.

Składnia argumentu lista_argumentów **oraz jej części:**

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

Gdzie:

[Optional] – wskazuje, że argument nie jest wymagany;

[ByVal] – wskazuje, że argument jest przekazywany przez wartości;

[ByRef] – wskazuje, że argument jest przekazywany przez referencję (domyślny sposób);

[ParamArray] – używany jako ostatni argument w lista_argumentów, który wskazuje, że jest to opcjonalna tablica elementów typu Variant. Pozwala na przekazywanie dowolnej liczby argumentów. Nie może być używany z ByVal, ByRef lub Optional.

Przykłady:

```
Function funkcja1() 'bez argumentów
…
End Function

Function funkcja2(liczba1, liczba2) As Long 'stała określona
    liczba argumentów (maksymalnie do 60)
…
End Function

Function funkcja3(tablica) As Double
…
End Function

Function funkcja4(ParamArray lista() As Variant) As Double
    'nieokreślona liczba argumentów
…
End Function

Function funkcja5(zakres As Variant, Optional komorka As
    Variant = False) As Double 'argumenty wymagane i
    opcjonalne
…End Function
```

Domyślny typ argumentu to Variant. Tylko dla argumentów Optional można określić stałą jako wartość domyślną. Do sprawdzenia, czy argument Optional został przekazany do procedury można wykorzystać funkcję IsMissing. Jeżeli argument jest typu Object, to wartość domyślna nie może mieć wartości Nothing.

7.9 Wywołanie procedury typu Function

Procedury typu Function można wywoływać w formule arkuszy z poziomu okna Immediate oraz z innych procedur.

7.9.1 Wywołanie procedury w tym samym module

```
wynik = nazwa_procedury(argument1, argument2, … argument-n)

wynik = Application.Run (”nazwa_procedury”, argument1,
argument2, … argument-n)
```

Przykład:

```
wynik = ObliczV(1)

wynik = Application.Run (”ObliczV”, 2)
```

7.9.2 Wywołanie procedury umieszczonej w innym module

```
wynik = nazwa_modulu.nazwa_procedury(arg1, arg2, … arg-n)

wynik = Application.Run (”nazwa_modulu.nazwa_procedury”,
argument1, argument2, … argument-n)
```

Przykład:

```
wynik = Module1.ObliczV(10)

wynik = Application.Run (”Module1.ObliczV”, 20)
```

7.9.3 Wywołanie procedury umieszczonej w innym skoroszycie

```
wynik = nazwa_projektu. nazwa_modulu.nazwa_procedury(arg1,
arg2, … argt-n)

wynik = Application.Run (”nazwa_projektu.
nazwa_modulu.nazwa_procedury”, arg1, arg2, … arg-n)
```

Przykład:

```
wynik = Projekt1.Module1.ObliczV(100) ‘konieczność
utworzenia odwołania do Projektu1 (edytor VBE – menu Tools –
polecenie References…)

wynik = Application.Run (” ‘D:\test.xls’!Module1.ObliczV”,
200) ‘brak konieczności tworzenia odwołania do Projektu1
(menu Tools – polecenie References…)
```

7.10 Procedury (funkcje) bibliotek DLL (dynamic-link library) Windows API (Application Programming Interface)

Korzystanie z procedur I funkcji zadeklarowanych w zewnętrznych bibliotekach możliwe jest po uprzedniej deklaracji za pomocą instrukcji Declare. Instrukcja musi być umieszczona w części deklaracyjnej każdego typu modułu.

Składnia 1:

```
[Private |Public] Declare Sub NazwaProcedury Lib
"NazwaBiblioteki" [Alias "NazwaAlias"] [([lista
argumentów])]
```

Składnia 2:

```
[Private |Public] Declare Function NazwaProcedury Lib
"NazwaBiblioteki" [Alias "NazwaAlias"] [([lista
argumentów])] [As typ]
```

Składnia argumentu lista argumentów oraz jej części:

```
[Optional] [ByVal | ByRef] [ParamArray] nazwa_zmiennej [()]
[As typ]
```

Przykład:

```
Private Declare Sub MessageBeep Lib "User32" (ByVal N As
Integer)
Call MessageBeep(0) 'wygenerowanie dźwięku
```

Notatki własne:

.....

.....

.....

.....

.....

.....

.....

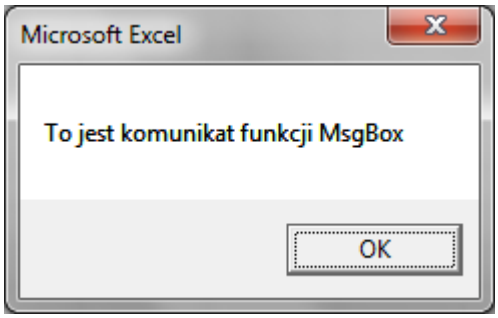
.....

8 Komunikacja z programem

8.1 Obiekt MsgBox

Funkcja MsgBox służy do tworzenia prostych okien komunikatów, które ułatwiają komunikacje. Funkcja **MsgBox** – wyświetla okno dialogowe z jednym lub więcej przyciskami i czeka na reakcję, po czym zwraca wartość typu Integer określającą który przycisk został naciśniety. Składnia funkcji:

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



Argmenty funkcji MsgBox (argumenty zawarte w nawiasach prostokątnych są nieobowiązkowe):

- ***prompt*** – argument obowiązkowy, wyrażenie znakowe wyświetlane jako komunikat w oknie dialogowym. Maksymalna długość prompt wynosi około 1024 znaki, zależnie od szerokości znaków w zastosowanej czcionce.
- ***buttons*** – argument nieobowiązkowy, wyrażenie numeryczne określające liczbę i typ wyświetlanych przycisków, rodzaj i styl używanych ikon, identyfikator domyślnego przycisku oraz modalność okna komunikatu. Brak parametru buttons spowoduje przyjęcie wartości domyślnej równej **0** (zero). Dostępne parametry:

Stała	Wartość	Opis
vbOK	1	Naciśnięto przycisk OK.
vbCancel	2	Naciśnięto przycisk Anuluj.
vbAbort	3	Naciśnięto przycisk Przerwij.
vbRetry	4	Naciśnięto przycisk Ponów.
vbIgnore	5	Naciśnięto przycisk Ignoruj.
vbYes	6	Naciśnięto przycisk Tak.
vbNo	7	Naciśnięto przycisk Nie.

- ***title*** – argument nieobowiązkowy, wyrażenie znakowe wyświetlane na pasku tytułu okna dialogowego. Brak argumentu title spowoduje, że na pasku tytułu zostanie umieszczona nazwa aplikacji.

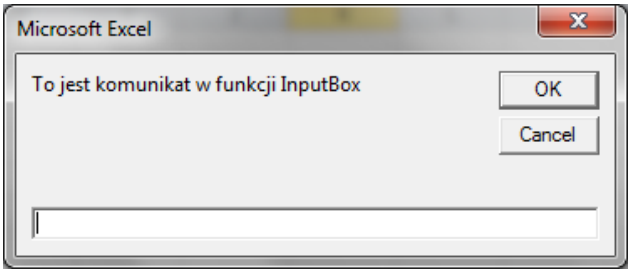
- ***helpfile*** – argument nieobowiązkowy, wyrażenie znakowe określające plik Pomocy zawierający pomoc kontekstową. Argument helpfile musi być zawsze podany z argumentem ***context***.
- ***context*** – argument nieobowiązkowy, wyrażenie numeryczne określające identyfikator tematu w pliku Pomocy. Jeśli podany jest argument context, to musi być również podany argument ***helpfile***.

8.2 Obiekt InputBox

Funkcja InputBox pozwala szybko pobrać dane od użytkownika. Możemy w ten sposób przekazać programowi dowolną wartość wprowadzoną z klawiatury.

Funkcja InputBox wyświetla okno dialogowe z polem tekstowym i dwoma przyciskami, po czym zwraca typ danych **String** będący zawartością pola tekstowego.

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



Argumenty funkcji InputBox:

- ***prompt*** – argument obowiązkowy, wyrażenie znakowe wyświetlane jako komunikat w oknie dialogowym. Maksymalna długość prompt wynosi około 1024 znaki, zależnie od szerokości znaków w zastosowanej czcionce.
- ***title*** – argument nieobowiązkowy, wyrażenie znakowe wyświetlane na pasku tytułu okna dialogowego. Brak argumentu title spowoduje, że na pasku tytułu zostanie umieszczona nazwa aplikacji.
- ***default*** – argument nieobowiązkowy, wyrażenie znakowe wyświetlane w polu tekstowym (jeśli nie zostanie podany inny tekst). Brak parametru default spowoduje, że pole tekstowe będzie puste.
- ***xpos*** – argument nieobowiązkowy, wyrażenie numeryczne określające w jednostkach zwanych "twips", odległość lewej krawędzi okna dialogowego od lewej krawędzi ekranu. Brak argumentu xpos spowoduje, że okno dialogowe zostanie umieszczone w równej odległości od lewej i prawej krawędzi ekranu.
- ***ypos*** – argument nieobowiązkowy, wyrażenie numeryczne określające w jednostkach zwanych "twips", odległość górnej krawędzi okna dialogowego od górnej krawędzi

ekranu. Brak argumentu ypos spowoduje, że okno dialogowe zostanie umieszczone na poziomie dwóch trzecich wysokości ekranu.

- ***helpfile*** – argument nieobowiązkowy, wyrażenie znakowe określające plik Pomocy zawierający pomoc kontekstową. Argument helpfile musi być zawsze podany z argumentem context.
- ***context*** – argument nieobowiązkowy, wyrażenie numeryczne określające identyfikator tematu w pliku Pomocy. Jeśli podany jest argument context, to musi być również podany argument helpfile.

8.3 Komunikacja z arkuszem:

- ActiveCell – oznacza aktualnie wybraną komórkę, jeśli wybrany jest zakres, to jest to komórka od której zaczęto zaznaczać;
- Selection – wybrany zakres komórek;
- Cells – bezpośrednie odwołanie do komórek adresując bezwzględnie, (czyli niezależnie od zaznaczenia);
- Adresowanie jak w macierzach: A(1,2) – 1.wiersz, 2. Kolumna

Aktualnie zaznaczona komórka:

- ActiveCell.Value – zwraca wartość tej komórki lub umożliwia przypisanie jej wartości:

```
Dim dana As Variant
dana = ActiveCell.Value
ActiveCell.Value = dana * 2
```

- Można też używać znanych z rejestracji makr metod związanych z formatowaniem: ActiveCell.Font, ActiveCell.FontSize, itd.
- ActiveCell.Row – zwraca numer wiersza zaznaczonej komórki;
- ActiveCell.Column – zwraca numer (liczbowy) kolumny zaznaczonej komórki;

Aktualnie zaznaczony zakres:

- Selection.Rows.Count – zwraca ilość wierszy zaznaczenia;
- Selection.Columns.Count – zwraca ilość kolumn zaznaczenia;

Zmienna – zakres:

```
Dim zmienna as Range
Set zmienna = Selection
```

- zmienna.Rows.Count – zwraca ilość wierszy zakresu przechowywanego w zmiennej;
- zmienna.Columns.Count – zwraca ilość kolumn zakresu przechowywanego w zmiennej;
- zmienna.Row – zwraca numer pierwszego wiersza zakresu ze zmiennej;
- zmienna.Column – zwraca numer pierwszej kolumny zmiennej;

- Ostatni wiersz zakresu:
- zmienna.Row + zmienna.Rows.Count – 1

Odwołania bezwzględne

- Cells(<wiersz>,<kolumna>) – odwołanie do komórki w arkuszu (domyślnie jest to wartość);
- Cells(w,k).Value – zarówno można przypisać wartość, jak i użyć do przypisania (w procedurach);
- Inne metody działają tak jak w poprzednich przypadkach (np. formatowanie jakiejś komórki);

```
dana = Cells(1,1).Value
Cells(1,2).Value = dana
```

(wynik: przepisanie zawartości komórki A1 do B1, bez znaczenia, jaka komórka jest zaznaczona)

Notatki własne:

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

9 Model obiektowy MS Excel

9.1 Czym są obiekty?

Kiedy pracujemy z językiem VBA musimy znać koncepcję obiektów i poznać model obiektowy Excela. Bardzo pomocne będzie tutaj myślenie o obiektach w kategorii hierarchii. Na szczycie modelu znajduję się obiekt Application, którym w tym przypadku jest sam Excel.

Obiekt Application zawiera inne obiekty. Na przykład w obiekcie Application zawierają się:

- Workbooks (kolekcja wszystkich obiektów Workbook – skoroszyt)
- Windows (kolekcja wszystkich obiektów Window – okna)
- AddIns (kolekcja wszystkich obiektów AddIn – dodatek)

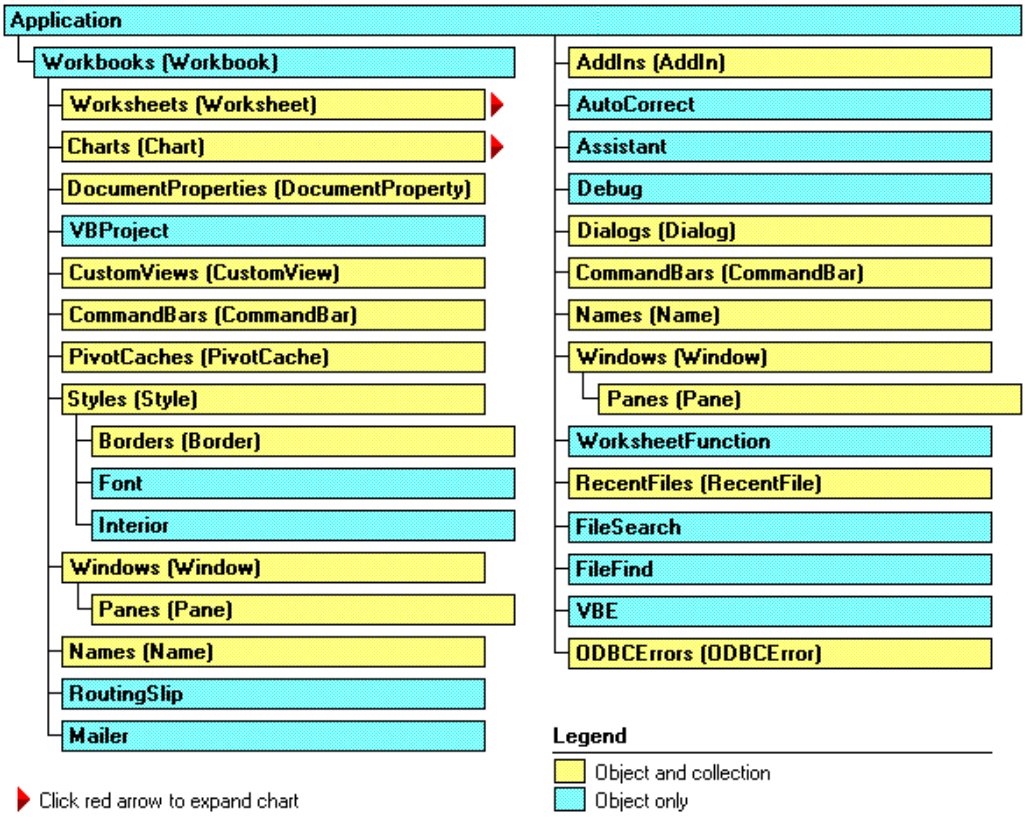
Niektóre obiekty są kontenerami dla innych obiektów. Przykładowo zbiór Workbooks składa się z wszystkich otwartych obiektów Workbook. Z kolei obiekt Workbook zawiera inne obiekty. Na przykład:

- Worksheets (kolekcja obiektów Worksheet – arkusz)
- Charts (kolekcja obiektów Chart – wykres)
- Names (Kolekcja obiektów Name – nazwa)

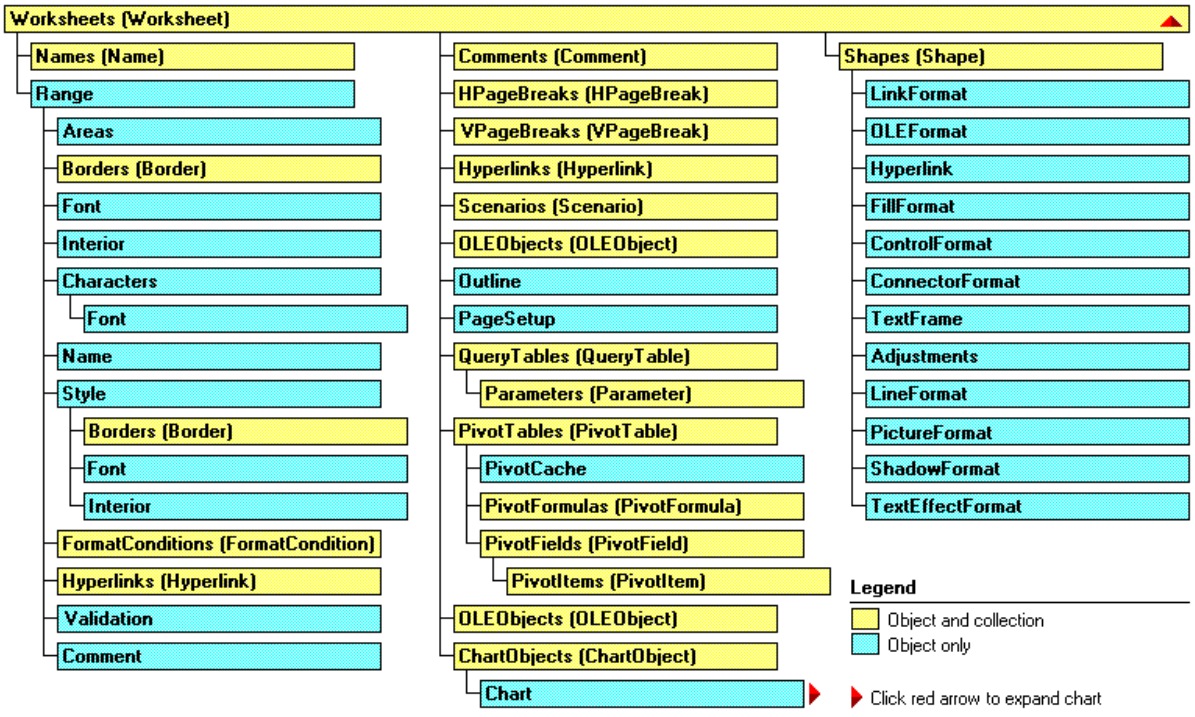
Z kolei każdy z powyższych obiektów przechowuje inne obiekty. Obiekt Worksheet zawiera wiele innych obiektów. Na przykład:

- ChartObjects
- Range
- PageSetup
- PivotTables

Hierarchia obiektów: (szczegółowa hierarchia dostępna jest w dokumentacji VBA):



Hierarchia obiektów WorkSheets



Odwołanie do właściwości obiektu:

```
obiekt.właściwość
Range("A1").Value
ActiveSheet.Range("A1").Value
```

Zmiana właściwości obiektu:

```
obiekt.właściwość = wartość
Range("A1").Value = 25
ActiveSheet.Range("A1").Value = "nowy tekst"
```

Wywołanie metody bezargumentowej:

```
obiekt.metoda
Range("A1").Clear
ActiveSheet.Range("A1").ClearFormats
```

Wywołanie metody z argumentami:

```
obiekt.metoda(arg1, arg2, ..., argN)
ActiveCell.Offset(3,2)
ActiveCell.Offset(rowOffset:=3,columnOffset:=2)
```

Zmienne obiektowe:

```
Dim zmienna As typ
Set zmienna = obiekty
```

np.

```
Dim rng As Range
Set rng = ActiveSheet.Rows(1)
```

Zwalnianie zmiennych obiektowych:

```
Set obiekt = Nothing
```

np.

```
Set rng = Nothing
```

9.2 Zmienna obiektowa

9.2.1 Deklaracja zmiennej obiektowej

Składnia:

```
[Public] Dim nazwa_zmiennej [As [New] typ_zmiennej]
```

Przykład:

```
Dim obj1 As Workbook
Dim obj2 As New Word.Application
```

9.2.2 Przypisanie zmiennej obiektowej

Składnia:

```
Set nazwa_zmiennej = {[New] wyrażenie_obiektowe | Nothing}
```

nazwa_zmiennej – nazwa zmiennej lub właściwości.

wyrażenie_obiektowe – wyrażenie zawierające nazwę obiektu, inną zadeklarowaną zmienną obiektową tego samego typu lub funkcję (metodę) zwracającą obiekt tego samego typu.

New – włączenie niejawnego (ukrytego) utworzenia obiektu. Tworzona jest nowa instancja klasy. Jeżeli zmienna obiektowa zawierała odwołanie do obiektu, to odwołanie jest zwalniane.

Nothing – przypisanie do obiektu zwalnia wszystkie zasoby systemu i pamięć związane z poprzednim odwołaniem obiektu.

Przykład:

```
Set obj = New Word.Application
Set obj = Nothing
```

9.2.3 Porównanie zmiennych obiektowych

Is	Składnia: Wynik = obiekt1 Is obiekt2 Przykład: Set A = C Set B = C A Is B 'wynik True	Operator używany do porównania dwóch zmiennych obiektowych (referencji do obiektu). Wynik porównania jest True tylko wtedy, gdy oba obiekty obiekt1 i obiekt2 wskazują na ten sam obiekt.

9.3 Klasy

Klasy to definicja obiektu, na podstawie której tworzone są obiekty, czyli instancje klasy. Do tworzenia klas wykorzystuje się moduły klas.

9.3.1 Tworzenie nowej klasy

- Dodanie do projektu nowego modułu klas oraz zmiana nazwy modułu (nazwa klasy to nazwa modułu);
- Dodanie właściwości klasy (deklaracja zmiennych);
- Dodanie metod klasy (deklaracja procedur);

- Deklaracja procedur właściwości na podstawie instrukcji (właściwość tylko do odczytu – Property Get, właściwość tylko do zapisu – Property Let, właściwość do odczytu i zapisu – Property Get i Let);

Property Let – przypisuje wartość do właściwości.
Składnia deklaracji: [Private|Public|Friend] [Static] Property Let *Nazwa* ([*lista_argumentów*,] *wartość*) [*instrukcje procedury*] [Exit Property] *'natychmiastowe wyjście z procedury* [*instrukcje procedury*] End Property *'koniec procedury*
Property Get – zwraca wartość właściwości.
Składnia deklaracji: [Private|Public|Friend] [Static] Property Get *Nazwa* ([*lista_argumentów*,] [As *typ*] [*instrukcje procedury*] [*Nazwa = wyrażenie*] [Exit Property] *'natychmiastowe wyjście z procedury* [*instrukcje procedury*] [*Nazwa = wyrażenie*] End Property *'koniec procedury*
Property Set – ustawia referencję do obiektu.
Składnia deklaracji: [Private|Public|Friend] [Static] Property Set *Nazwa* ([*lista_argumentów*,] *wartość*) [*instrukcje procedury*] [Exit Property] *'natychmiastowe wyjście z procedury* [*instrukcje procedury*] End Property *'koniec procedury*

- Definiowanie automatycznie tworzonych procedur zdarzeń (Class_Initialize – podczas inicjalizacji nowego obiektu na podstawie klasy oraz zdarzenia Class_Terminate – podczas niszczenia obiektu, zwalniania zasobów pamięci).

Wszystkie publiczne procedury w module klasy to metody obiektu, natomiast wszystkie publiczne zmienne modułu lub procedur właściwości są właściwościami obiektu.

9.4 Kolekcje

To grupa obiektów powstałych na podstawie tej samej klasy.

9.4.1 Tworzenie kolekcji

VBA dostarcza obiekt `Collection`, za pomocą którego można tworzyć kolekcje obiektów.

- Obiekt Collection zawiera metody umożliwiające sprawdzenie liczby elementów (Count), dodanie elementu do kolekcji (Add), usunięcie elementu z kolekcji (Remove), odwołanie się do elementu (Item)

Przykład:

```
Dim kolekcja As New Collection
kolekcja.Add (”Excel”)
kolekcja.Add (”Word”)
Debug.Print kolekcja.Count 'liczba elementów kolekcji – 2
```

- Odwołanie się do elementu kolekcji następuje za pomocą indeksu lub nazwy.

Przykład:

```
Workbooks(1).Worksheets(”Arkusz1”)
```

9.5 Operacje na obiektach i kolekcjach

Konstrukcja	Składnia, przykład	Opis
With … End With	With obiekt [instrukcje] End With	Wykonanie wielu operacji na pojedynczym obiekcie lub danej zdefiniowanej przez użytkownika.
For Each … Next	For Each element In grupa [instrukcje] [Exit For] [instrukcje] Next [element]	Wykonanie wielu operacji dla każdego elementu należącego do grupy (kolekcja lub tablica). Instrukcja Exit For pozwala na przerwanie pętli w dowolnym miejscu wewnątrz konstrukcji For Each … Next.

9.6 Obsługa błędów

Błędy składni (syntaktyczne) oraz błędy wykonania (w trakcie działania programu), które mogą być obsłużone lub nieobsłużone przez odpowiednie mechanizmy.

9.7 Instrukcja On Error

Włącza procedurę obsługi błędów oraz określa miejsce procedury. Może być również użyta do wyłączenia procedury obsługi błędów.

Składnia	Opis
On Error GoTo etykieta	Włącza procedurę obsługi błędów przez określenie miejsca w procedurze do którego przekazane zostanie dalsze wykonywanie instrukcji w przypadku wystąpienia błędu. Argument etykieta określa miejsce procedury obsługi błędów jako etykieta wiersza lub numer wiersza i musi być umieszczony w obrębie tej samej procedury co instrukcja On Error.
On Error Resume Next	Określa, że w przypadku wystąpienia błędu wykonania programu następuje natychmiastowe przekazanie sterowania do następnej instrukcji

Składnia	Opis
	(zignorowanie błędu i zezwolenie VBA na wykonywanie dalszych instrukcji procedury).
On Error GoTo 0	Wyłączenie wcześniej włączonej procedury obsługi błędów w bieżącej procedurze (przywrócenie standardowej obsługi błędów)

9.8 Instrukcja Resume

Składnia	Opis
Resume [0]	Jeżeli błąd wystąpił w tej samej procedurze co procedura obsługi błędów, wykonanie programu przywracane jest do instrukcji, która spowodowała błąd. Jeżeli błąd wystąpił w wywołanej procedurze, wykonanie powraca do instrukcji, która ostatnia wywołała procedurę zawierającą obsługę błędów.
Resume Next	Działania analogiczne jak w przypadku instrukcji Resume [0] z tą różnicą, że wykonanie powraca do następnej instrukcji po instrukcji, która ostatnia wywołała procedurę zawierającą obsługę błędów (lub po instrukcji On Error Resume Next).
Resume etykieta	Powoduje dalsze wykonywanie kodu od linii określonej przez argument etykieta. Argument ten określony przez etykietę lub numer wiersza musi znajdować się w tej samej procedurze co obsługa błędu.

Przykład procedury z obsługą błędów:

```
Sub NazwaProcedura1 (argument1, argument2, …)
    … 'instrukcje procedury
On Error GoTo etykieta1 'miejsce obsługi błędu
… 'kolejne instrukcje procedury
On Error Resume Next 'zignorowanie błędu
    …
On Error GoTo 0 'wyłączenie obsługi błędów
    …
Exit Sub
etykieta1: 'instrukcje obsługi błędu
    …
Resume Next 'instrukcje powrotu po zakończeniu obsługi
    błędów
End Sub
```

Do identyfikacji błędu wykonania wykorzystywany jest obiekt Err.

Notatki własne:

10 Przykładowe makra

Przykład 1

Makro pobierające kursy walut (rejestracja) i obramowujące komórki:

```
Sub KursyWalutNBP()
'Procedura pobierająca ze strony internetowej kursy walut

'wyłączenie trybu kopiowania
Application.CutCopyMode = False

'QueryTables – kolekcja w VBA, która reprezentuje zewnętrzne
    źródła danych
'dodanie do aktywnego arkusza danych zewnętrznych z strony
    internetowej, której adres podajemy; miejscem
    wstawienia danych jest aktywna komórka; przy
    generowaniu makra z rejestratora dostajemy również
    właściwość .CommandType = 0, którą musimy usunąć; jest
    ona wykorzystywana przy pobieranu z baz danych i jeśli
    jej nie usuniemy, makro nie będzie działać

With ActiveSheet.QueryTables.Add(Connection:= _
    "URL;http://www.nbp.pl/kursy/KursyA.html", _
    Destination:=ActiveCell)
    .Name = "KursyA"
    .FieldNames = True
    .RowNumbers = False
    .FillAdjacentFormulas = False
    .PreserveFormatting = True
    .RefreshOnFileOpen = False
    .BackgroundQuery = True
    .RefreshStyle = xlInsertDeleteCells
    .SavePassword = False
    .SaveData = True
    .AdjustColumnWidth = True
    .RefreshPeriod = 0
    .WebSelectionType = xlSpecifiedTables
    .WebFormatting = xlWebFormattingNone
    .WebTables = "4"
    .WebPreFormattedTextToColumns = True
    .WebConsecutiveDelimitersAsOne = True
    .WebSingleBlockTextImport = False
    .WebDisableDateRecognition = False
    .WebDisableRedirections = False
    .Refresh BackgroundQuery:=False
End With
```

```
'obramowanie tabelki
'Zaznaczanie aktywnej komórki wraz z całym obszarem aktywnym
ActiveCell.CurrentRegion.Select

'Call – odwołanie do innej procedury z podaniem jej nazwy
Call obramowanie

'wyróżnienie komórek nagłówka dla danego zakresu
With Range(ActiveCell, ActiveCell.End(xlToRight))
     .Font.Bold = True 'właczenie pogrubienia czcionki
     .Font.Color = RGB(15, 44, 222) 'kolor czcionki w RGB
     .HorizontalAlignment = xlCenter 'wyśrodkowanie
     .Interior.Color = RGB(180, 215, 30) 'kolor wypełnienia
End With

End Sub
```

Procedura obramowująca zaznaczony zakres (makro zarejestrowane):

```
Sub Obramowanie()

'Przekątna od lewego górnego narożnika wyłączona
Selection.Borders(xlDiagonalDown).LineStyle = xlNone

'Przekątna od lewego dlonego rogu wyłączona
Selection.Borders(xlDiagonalUp).LineStyle = xlNone

'Lewa zewnętrzna krawędź zaznaczenia
With Selection.Borders(xlEdgeLeft)
     .LineStyle = xlContinuous 'Styl linii – ciągły
     .ColorIndex = xlAutomatic 'Kolor linii – automatyczny
     .TintAndShade = 0 'Odcień linii neutralny
     .Weight = xlThick 'Grubość linii – pogrubiona
End With

'Górna zewnętrzna krawędź zaznaczenia
With Selection.Borders(xlEdgeTop)
     .LineStyle = xlContinuous
     .ColorIndex = xlAutomatic
     .TintAndShade = 0
     .Weight = xlThick
End With

'Dolna zewnętrzna krawędź zaznaczenia
With Selection.Borders(xlEdgeBottom)
     .LineStyle = xlContinuous
     .ColorIndex = xlAutomatic
```

```
     .TintAndShade = 0
     .Weight = xlThick
End With

'Prawa zewnętrzna krawędź zaznaczenia
With Selection.Borders(xlEdgeRight)
     .LineStyle = xlContinuous
     .ColorIndex = xlAutomatic
     .TintAndShade = 0
     .Weight = xlThick
End With

'Pionowa wewnętrzna krawędź zaznaczenia
With Selection.Borders(xlInsideVertical)
     .LineStyle = xlContinuous
     .ColorIndex = xlAutomatic
     .TintAndShade = 0
     .Weight = xlThin 'Grubość linii - cienka
End With

'Pozioma wewnętrzna krawędź zaznaczenia
With Selection.Borders(xlInsideHorizontal)
     .LineStyle = xlContinuous
     .ColorIndex = xlAutomatic
     .TintAndShade = 0
     .Weight = xlThin
End With
End Sub
```

Notatki własne:

Notatki własne:

Przykład 2

Makro Komunikacji z użytkownikiem

```
Sub Przykal1()
'Procedura wyświetlająca prośbę o podanie wieku użytkownika,
    a następnie na jego podstawie określająca, czy
    użytkownik jest pełnoletni

Dim wiek 'Deklarowanie zmiennej bez podawania jej typu
Dim komunikat As String 'Deklarowanie zmiennej typu tekst

'Przypisywanie do zmienniej wiek treści podanej przez
    użytkownika w oknie InputBox
wiek = InputBox("Ile masz lat?", "Pytanie")

'Druga możliwość podawania argumentów funkcji:poprzez
    określenie nazwy argumentu 'nazwa argumentu':='treść':
'wiek = InputBox(Prompt:="Ile masz lat?", Title:="Pytanie")
'Funkcja warunkowa If sprawdzająca wartości zmiennej wiek

If wiek = "" Then
    Exit Sub 'Opuść procedurę
ElseIf wiek >= 18 Then 'Jeśli poprzedni warunek nie jest
    prawdziwy, sprawdź czy wiek jest większy równy 18
    komunikat = "Witaj," & vbCrLf & "jesteś pełnoletni"
    'do zmiennej komunikat jest przypisywana wartość
    'znak & -łączenie tekstów
    'vbCrLf - przejście do nowej linijki w wyświetleniu
    ' vbCrLf można zastąpić poleceniem vbNewLine
Else 'Jeśli powyższe warunki nie są spełnione to
    komunikat = "Witaj," & vbCrLf & "jesteś niepełnoletni"
End If 'Zamknięcie instrukcji warunkowej If

'Polecenie wypisujące komunikat dla użytkownika o treści
    pobranej ze zmiennej komunikat z jednym przyciskiem
    (OK)i z ikoną informacyjną;w tytule wiek:
MsgBox Prompt:=komunikat, Buttons:=vbOKOnly + vbInformation,
    Title:="Twój wiek"
End Sub
```

Notatki własne:

.....

.....

.....

Notatki własne:

Przykład 3

Makro Pobierające dane

Kolejność makr w module może być dowolna

```
'Deklarowanie zmiennych modułowych dostępnych we wszystkich
    procedurach odbywa się przed wszystkimi procedurami

Dim sciezka_pliku As String 'ścieżka wybranego pliku
Dim LW_Plik As Long 'liczba wierszy w pobieranym pliku
Dim LW_Dane As Long 'liczba wierszy ogólnie pobranych
```

```
Sub Importuj_Dane()
'Procedura wywołująca kolejno wszystkie procedury

On Error GoTo blad 'włączenie obsługi błędów – w razie
    wystąpienia błędu następuje skok do miejsca oznaczonego
    etykietą blad

Application.ScreenUpdating = False 'wyłączenie odświeżania
    ekranu, aby nie migał podczas wykonywania makra

'Odwołania do poszczególnych procedur
Call Import_CSV
Call Wypelnij_Puste
Call Przenies_Dane
Call Podsumowanie

Application.ScreenUpdating = True 'Przywrócenie odświeżania
Exit Sub 'zakończenie procedury przed rozpoczęciem poleceń
    odpowiadających za obsługę błędu
blad: 'etykieta informująca o początku poleceń
    odowiadających za obsługę błędu
MsgBox "Coś poszło nie tak, spróbuj jeszcze raz…" 'funkcja
    wyświetlająca komunikat

End Sub
```

```
Sub Import_CSV()
'Procedura pobiera dane z pliku CSV

'ChDir – funkcja ustawiająca domyślny folder
ChDir "C:\Expose\Excel_vba_pliki\importer\"

'GetOpenFilename – metoda otwierająca okienko wyboru pliku
```

```
'Przypisanie do zmiennej sciezka nazwy pliku
Sciezka_pliku = Application.GetOpenFilename

'Przy naciśnięciu Anuluj w metodzie GetOpenFilename do
    zmiennej sciezka zostanie przypisana wartość False
If sciezka = "False" Then
    End 'kończy działanie wszystkich procedur
End If

Workbooks.OpenText _
    Filename:=sciezka, _
    Origin:=xlWindows, _
    StartRow:=3, _
    DataType:=xlDelimited, _
    TextQualifier:=xlDoubleQuote, _
    ConsecutiveDelimiter:=False, _
    Tab:=True, _
    Semicolon:=False, _
    Comma:=False, _
    Space:=False, _
    Other:=False, _
    FieldInfo:=Array(Array(1, 1), Array(2, 1), Array(3, 5),
    Array(4, 1), Array(5, 1), Array(6, 1), Array(7, 2)), _
    DecimalSeparator:=".", _
    TrailingMinusNumbers:=True
'Z kolekcji skoroszytów używamy metody otwarcia pliku
    tekstowego
'Filename – parametr nazwy i lokalizacji pliku określony za
    pomocą zmiennej globalnej sciezka_pliku
'Origin –pochodzenie pliku, w tym przypadku plik Windowsa
'StartRow – parametr określający od którego wiersza należy
    zacząć importowanie tekstu
'DataType – parametr określający jak są dzielone kolumny –
    typ rozdzielany czy stała szerokość
'TextQualifier – określa separator używany do dzielenia
    tekstu użyty w pliku
'ConsecutiveDelimiter – parametr określający, czy kolejne
    ograniczniki są traktowane jako jeden, jeśli występują
    po sobie
'Tab, Semicolon, Comma, Space, Other – parametry określające
    rozdział kolumn (wartości PRAWDA lub FAŁSZ)
'FieldInfo –określenie typów danych w kolumnach – tablice
'Array(Array(1[numer kolumny], numer odpowiadający rodzajowi
    danych[od 1 do 10]), ... , Array(n-ty, m-ty)) – każda
    tablica to jedna kolumna, jeśli jest więcej niż jedna
    kolumna w tekscie, to tablica jest zbiorem tablic –
    kolumn
```

```
'DecimalSeparator –określenie separatora dziesiętnego
'TrailingMinusNumbers – parametr określający, czy liczby
     z minusem na końcu mają być jako liczby ujemne, czy
     jako tekst
End Sub
```

```
Sub Wypelnij_Puste()
‘makro wypełnia wszystkie puste komórki w zakresie

Selection.CurrentRegion.SpecialCells(xlCellTypeBlanks).Formu
     laR1C1 = "=R[-1]C"
'Zaznaczenie całego obszaru danych, wybranie spośród nich
     komórek pustych i wpisanie do nich formuły pobierającej
     danej z komórki znajdującej się bezpośrednio powyżej

Selection.CurrentRegion.Copy ' Skopiowanie całego obszaru

Selection.PasteSpecial xlPasteValues ‘wklejenie wartości

Application.CutCopyMode = False 'Wyłączenie trybu kopiowania

LWPlik = Range("A1").End(xlDown).Row – 1
'Przypisanie do zmiennej globalnej LW_Plik liczby wierszy
     w importowanym pliku; liczba wierszy minus 1, by odjąć
     wiersz nagłówka

End Sub
```

```
Sub PrzeniesDane()

'zmienne z nazwami plików
Dim skoroszyt_csv As Workbook
Dim skoroszyt_dane As Workbook

'zmienna z przenoszonym zakresem
Dim zakres_kopiowany As Range

'zmienna z nazwą arkusza na dane (zbiorcze)
Dim arkusz_dane As Worksheet

'zmienna z adresem komórki od której wklejamy dane
Dim komorka_wklejenia As Range

'zmienna przchowująca inforacje o nowym/tworzonym arkuszu
Dim nowy_arkusz As Worksheet
```

```
'zmienne obiektowe wykorzystują SET
Set skoroszyt_csv = ActiveWorkbook
Set skoroszyt_dane = Workbooks("importer.xlsm")

Set zakres_kopiowany =
    skoroszyt_csv.Worksheets(1).Range("a1").CurrentRegion

Set arkusz_dane = skoroszyt_dane.Worksheets("dane")

'skopiowanie danych do arkusza docelowego
If arkusz_dane.Range("a1") = Empty Then
    zakres_kopiowany.Copy arkusz_dane.Range("a1")
Else
    Set komorka_wklejenia =
        arkusz_dane.Range("a1").End(xlDown).Offset(1, 0)
    zakres_kopiowany.Copy komorka_wklejenia
    komorka_wklejenia.EntireRow.Delete ' usunięcie wiersza
    z komórką
End If

'zliczanie wierszy w arkuszu dane i przypisanie do zmiennej
LW_dane = arkusz_dane.Range("a1").End(xlDown).Row - 1

'wstawienie kopii danych do nowego arkusza (utworzenie
    arkusza)
'wstawienie nowego arkusza na końcu wszystkich arkuszy
Set nowy_arkusz = skoroszyt_dane.Worksheets.Add( _
    after:=skoroszyt_dane.Worksheets( _
    skoroszyt_dane.Worksheets.Count))

'zmiana nazwy arkusza
nowy_arkusz.Name = "dane_" & Format(Now, "dd-mm-yyyy
    hh_mm_ss")

'moment skopiowania danych
zakres_kopiowany.Copy nowy_arkusz.Range("a1")

'automatyczna szerokość kolumn
nowy_arkusz.Range("a1").CurrentRegion.EntireColumn.AutoFit

'zamknięcie pliku bez zapisywania zmian
skoroszyt_csv.Close False

End Sub
```

```
Sub Podsumowanie()
'Procedura uzupełniająca arkusz "podsumowanie"

'Cells – odwołanie do komórki, w postaci (numer wiersza,
    numer kolumny)
'Cells.Rows.Coutnt – z kolekcji komórek wybieranie
    właściwości wierszy i podanie ich liczby dla całego
    arkusza
'End(xlUp) – przejście w górę do komórki wypełnionej, a
    następnie Offset(1, 0) o jeden wiersz w dół
'wartość komórki ma się równać dzisiejszej dacie dzięki
    funkcji Now()
Cells(Cells.Rows.Count, 1).End(xlUp).Offset(1, 0).Value =
    Now()
'wartość komórki ma się równać zmiennej globalnej sciezka
Cells(Cells.Rows.Count, 2).End(xlUp).Offset(1, 0).Value =
    sciezka
'Environ – funkcja, która odwołuje sie do systemu
    operacyjnego; wartość komórki ma się równać nazwie
    użytkownika
Cells(Cells.Rows.Count, 3).End(xlUp).Offset(1, 0).Value =
    Environ("username")
'wartość komórki ma się równać zmiennej globalnej LWPlik
Cells(Cells.Rows.Count, 4).End(xlUp).Offset(1, 0).Value =
    LWPlik
'wartość komórki ma się równać zmiennej globalnej LWDane
Cells(Cells.Rows.Count, 5).End(xlUp).Offset(1, 0).Value =
    LWDane

End Sub
```

Notatki własne:

.....

.....

.....

.....

.....

.....

.....

Notatki własne:

Przykład 4

Makro Przykład pętli For…Next

```
Sub PetlaForNext()

'Deklaracja zmiennej użytej w pętli
Dim i As Integer

‘Rozpoczęcie pętli słowem kluczowym For, następnie
    początkowa wartość liczniki i wartość mksymalna dla
    zmiennej; na końcu skok licznika
For i=1 To 10 Step 2

'Wywołanie okna MsgBox z komunikatem „Pętla numer: „ i”,
    gdzie i przy każdym kroku wzrasta
MsgBox "Pętla numer: " & i
‘kończenie danego wykonania pętli i ropoczęcie kolejnego
Next i
‘i (zmienna pętli) jest opcjonalne, ale poprawia czytelność

End Sub
```

Notatki własne:

Przykład 5

Makro Przykład pętli For Each...Next

```
Sub PetlaForEachNext()
'Deklaracja zmiennej obiektowej na zakres
Dim komorka As Range

'Rozpoczęcie pętli słowami kluczowymi For Each, następnie:
    nazwa zmiennej,
    słowo kluczowe In
    określenie kolekcji (można wpisać również stały zakres)
For Each komorka In Selection
    MsgBox komorka.Address
Next komorka

End Sub
```

Notatki własne:

Przykład 6

Makro Przykład pętli Do While/Until...Loop

```
Sub PetlaDoWhileUntilLoop()

‘Deklaracja zmiennej bez podania typu, by zapobiec błędom,
    jeśli użytkownik poda inny typ niż zadeklarowany
Dim wynik

'Rozpoczęcie pętli słowem kluczowym Do
'następnie polecenie While – jeśli warunek jest spełniony
    pętla się wykonuje
Do While wynik <> 4 'lub równoważne: ' Do Until wynik = 4'

‘Przypisanie do zmiennej wynik wartości podanej przez
    użytkownika w okienku InputBopx
wynik = InputBox("Podaj wynik działania 2 + 2", "Zagadnka")
Loop ‘zamknięcie pętli
'polecenia While lub Until można wpisać po słowie Loop,
    wtedy pętla będzie sprawdzać warunek dopiero po
    wykonaniu poleceń – pętla wykona się miniumum raz,
    jeśli warunki są na górze, to pętla może się wcale nie
    wykonać

'Po zakończeniu działania pętli wyświetla się komunikat
MsgBox "Brawo! Wynik poprawny."

End Sub
```

Notatki własne:

.....

.....

.....

.....

.....

.....

.....

.....

Przykład 7

Makro Usuwające wiersze

```
Sub UsunWiersze()
'Procedura usuwa co 2 wiersz

Dim LiczbaWierszy As Integer
Dim i As Integer

'Przypisywanie do zmiennej wartości liczby wierszy z zakresu
    od A1 do ostatniej na dole komórki z danymi
LiczbaWierszy = Range("A1").End(xlDown).Row

'Wywołanie pętli For...Next dla zmiennej i, której wartość
    będzie się zmieniać aż do uzyskania LiczbyWierszy
    z krokiem -2
For i = LiczbaWierszy To 2 Step -2
    Rows(i).Delete 'Dla każdego kroku pętli będzie usuwany
    wiersz o numerze równemu zmiennej i
Next i

End Sub
```

Notatki własne:

Przykład 8

Makro Tworzące Arkusze

```
Sub TworzenieArkuszy()
'Procedura dodaje nowe arkusze o nazwach, które są
    wartościami zaznaczonych komórek, bez powtórzeń

Dim komorka As Range 'zmienna obiektowa jako zakres
Dim arkusz As Worksheet ’zmienna obiektowa jako arkusz
    'Worksheet – kolekcja arkuszy roboczych 'standardowy'
    ‘Sheet – kolekcja wszystkich arkuszy

'Przypisywanie do zmiennej obietkowej (koniecznie z użyciem
    słowa Set) aktywnego arkusza
Set arkusz = ActiveSheet

‘Wywołanie pętli For Each
For Each komorka In Selection
'instrukcja If sprawdza czy wartość zmiennej komorka jest
    różna od pustej i czy wynik funkcji CzyArkuszIstnieje
    jest równy FAŁSZ
    If komorka.Value <> "" And _
        CzyArkuszIstnieje(komorka.Value) = False Then
        ‘dodanie arkusza na końcu
        Worksheets.Add after:=Worksheets(Worksheets.Count)
        ‘zmiana nazwy aktywnego arkusza
        ActiveSheet.Name = komorka.Value
    End If
Next komorka

‘powrót do pierwszego arkusza
arkusz.Select

End Sub
```

```
Function CzyArkuszIstnieje(NazwaArkusza As String) As
    Boolean
'Funkcja sprawdzająca czy podany tekst jest nazwą któregoś
    z arkuszy – zwraca w wyniku PRAWDA lub FAŁSZ

CzyArkuszIstnieje = False 'Przypisanie wartości FAŁSZ

Dim arkusz As Worksheet 'Deklaracja zmiennej obiektowej

‘Wywołanie pętli For Each...Next
For Each arkusz In Worksheets
```

```
    'instrukcja If sprawdza czy nazwa zmiennej arkusz równa
    się zmiennej, z którą wywołujemy funkcję
    If arkusz.Name = NazwaArkusza Then
        'Przypisanie do funkcji wartości PRAWDA
        CzyArkuszIstnieje = True
        'Instrukcja wyjścia z pętli For bez kontynuowania
    kolejnych kroków
        Exit For
    End If
Next arkusz

End Function
```

Notatki własne:

Przykład 9

Makro Odkrywające ukryte arkusze

```
Sub OdkryjArkusze()

'Deklarowanie zmiennej obiektowej
Dim arkusz As Worksheet

'Wywołanie pętli For Each...Next
For Each arkusz In Worksheets
     'Ustawienie widoczności zmiennej arkusz na odkryty
     arkusz.Visible = xlSheetVisible
Next arkusz

End Sub
```

Notatki własne:

Przykład 10

Makro Powielające wiersze

```
Sub PowielWiersze()
'Procedura sprawdza wartości w kolejnych komórkach, dodaje
    tyle wierszy, by w sumie było ich tyle, ile wynosi
    wartość w testowanej komórce

Dim komorka As Range 'Deklarowanie zmiennej obiektowej

Set komorka = Range("I2") 'Przypisanie wartości do zmiennej

'Pętla Do While wykonywana dopóki wartość zmiennej komorka
    jest niepusta
Do While komorka.Value <> Empty
    If komorka.Value > 1 Then

    'Zaznaczenie obszaru (od komórki o wiersz niżej niż
    zmienna komorka, liczba komórek - o 1 mniej niż wartość
    w komórce)
    'EntireRow - właściwość zaznaczenia całych wierszy
    'Insert - metoda wstawienia (wiersze wstawiają się nad
    zaznaczonymi w ilości zaznaczonych)
        Range(komorka.Offset(1, 0), komorka.Offset
            (komorka.Value -1, 0)).EntireRow.Insert

    'FillDown - metoda wypełnienia w dół
        Range(komorka, komorka.End(xlToLeft).Offset
            (komorka.Value - 1, 0)).FillDown
    End If

    'Przypisanie zmiennej komorka nowej wartości komórki,
    która jest niżej o tyle wierszy ile znajduje się w
    komórce
    Set komorka = komorka.Offset(komorka.Value, 0)

Loop

End Sub
```

Notatki własne:

.....

.....

.....

Notatki własne:

Przykład 11

Makro Funkcja Podająca nazwę pliku

```
Function NazwaPliku(Sciezka As String, Optional Rozszerzenie
    As Boolean = True) As String

'Funkcja zwracająca nazwę pliku z podanej ścieżki domyślnie
    z rozszerzeniem

'Definiowanie zmiennych
Dim PolozenieBS As Integer
Dim PolozenieKropki As Integer
Dim DlugoscSciezki As Integer

'Przypisanie do PolozenieBS za pomocą funkcji InStrRev
    pozycji znaku \ w zmiennej Sciezka
'Funkcja InStrRev – zwracapozycję pierwszego wystąpienia
    zadanego ciągu, zaczynając od prawej, lecz podając
    pozycję od lewej
PolozenieBS = InStrRev(Sciezka, "\")

'Przypisanie wartości do zmiennej
PolozenieKropki = InStrRev(Sciezka, ".")

'Przypisywanie wartości do zmiennej DlugoscSciezki za pomocą
    funkcji Len
'Funkcja Len – zwraca liczbę znaków w ciągu znaków
DlugoscSciezki = Len(Sciezka)

'Instrukcja If sprawdza wartość logiczną argumentu
    Rozszezenie
If Rozszerzenie Then

    'Dla wartości True: za pomocą funkcji Mid przypisuje do
    wyniku funkcji NazwaPliku wartość tekstową(początek –
    PolozenieBS + 1; długość – różnica wartości zmiennych)
    NazwaPliku = Mid(Sciezka, PolozenieBS + 1, _
    DlugoscSciezki – PolozenieBS)
Else

    'Dla wartości False zmiennej Rozszerzenie – długość
    równa różnicy zmiennych PolozenieKropki i PolozenieBS -
    1
    NazwaPliku = Mid(Sciezka, PolozenieBS + 1, _
    PolozenieKropki – PolozenieBS – 1)
End If
```

```
End Function
```

Notatki własne:

Przykład 12

Makro Funkcja Licząca prowizję

```
Function Prowizja(Wartosc As Currency) As Currency

'Funkcja zwracająca wartość prowizji dla danej wartości
    liczbowej

'Deklarowanie stałych i przypisanie do nich wartości
Const Prow1 As Single = 0.08
Const Prow2 As Single = 0.105
Const Prow3 As Single = 0.12
Const Prow4 As Single = 0.14

'Użycie instrukcji warunkowej Select Case dla zmiennej
    Wartosc
Select Case Wartosc
'W przypadku, gdy zmienna przyjmie wartości od 0 do 9999.99
    (separator dziesiątny VBA to kropka)
    Case 0 To 9999.99
        'Funkcji przypisujemy wartość iloczynu argumentu
           Wartosc oraz stałej Prow1
        Prowizja = Wartosc * Prow1
    Case 10000 To 19999.99
        Prowizja = Wartosc * Prow2
    Case 20000 To 29999.99
        Prowizja = Wartosc * Prow3
    Case Is > 30000
        Prowizja = Wartosc * Prow4
    Case Else 'W każdym innym przypadku
        Prowizja = 0
End Select

End Function
```

Notatki własne:

Przykład 13

Makro tworzące przykładową tabelę przestawną

```
Sub tab_przest()

'Deklaracja zmiennych
Dim PTCache As PivotCache
Dim PT As PivotTable

'Tworzenie bufora
Set PTCache = ActiveWorkbook.PivotCaches.Create(SourceType:=
    xlDatabase, _ SourceData:=Range("A1").CurrentRegion)

'Dodawanie nowego arkusza dla tabeli przestawnej
Worksheets.Add

'Tworzenie tabeli przestawnej
Set PT = ActiveSheet.PivotTables.Add(PivotCache:=PTCache, _
    TableDestination:=Range("A3"))

'Dodawanie poszczególnych pól tabeli przestawnej
With PT
    .PivotFields("FL_Korp").Orientation = xlRowField
    .PivotFields("FL_Korp").Position = 1
    .PivotFields("NZW_KL").Orientation = xlRowField
    .PivotFields("NZW_KL").Position = 2
    .PivotFields("KWOTA W KPLN").Orientation = xlDataField
    .ShowDrillIndicators = False'Usunięcie przycisków +/-
    .DataBodyRange.NumberFormat = "0.00" 'format liczbowy
    .SubtotalHiddenPageItems = False 'Usunięcie Sum
    częściowych
End With

End Sub
```

Notatki własne:

Notatki własne:

EXPOSÉ

ul. Skierniewicka 10a

01-230 Warszawa

Te.: 22 465 88 88

biuro@expose.pl

www.expose.pl