

Krzysztof Kuciński

Visual Basic dla Excela

w przykładach



Krzysztof Kuciński

Visual Basic dla Excela w przykładach

Wydawnictwo
Witanet

2015

Copyright © by Krzysztof Kuciński; 2015
Copyright © by Wydawnictwo Witanet; 2015

Wszelkie prawa zastrzeżone

Grafiki w tekście: © Krzysztof Kuciński
Skład tekstu oraz projekt okładki: Ryszard Maksym

ISBN: 978-83-64343-49-0

Wydawca:

Wydawnictwo WITANET
ul. Brzoskwiniowa 11/1
62-571 Stare Miasto
tel.: #48 601 35 66 75
<http://www.witanet.net>
wydawnictwo@witanet.net

Niniejszy produkt jest objęty ochroną prawa autorskiego. Książka ani żadna jej część nie mogą być publikowane ani w jakikolwiek inny sposób powielane w formie elektronicznej oraz mechanicznej bez zgody wydawcy.

Spis treści

Historia	6
Przygotowanie środowiska	6
Wykaz plików pomocniczych	7
Makro1	8
Rejestrator makr	9
Edytor Visual Basic	12
Obiekty excelowe	15
Metody wykonywane na obiektach	16
Instrukcja wiążąca With	17
Właściwości obiektów	19
'komentarz	20
Nowe makro	20
Ułatwienia	22
Przeglądarka obiektów	23
Błędy	24
Zapis pliku z makrami	25
Tabliczka mnożenia	25
Pętla For ... Next	26
Zmienne	28
Typy zmiennych	30
Funkcje konwersji typu	32
Usuwanie	33
Pętla Do ... Loop	34
Instrukcja warunkowa If ... Then ... Else	34
Wycinanie	37
Funkcje tekstowe	38
Awaryjne przerywanie działania makra	42
Instrukcja wyboru Select Case	42
Dyski	43
Instrukcja MsgBox	44
Procedura parametryczna	46
Makro Auto_Open	48
Funkcje użytkownika	48
Pierwiastek kwadratowy wg Newtona, czyli iteracja	50
Fibonacci, czyli rekurencja	51
Operatory	52
Funkcje matematyczne	53
Działania	55
Instrukcja InputBox	57
Badania techniczne	58
Elementy sterujące	59
Funkcje sprawdzające	62
Funkcje koloru	65
Funkcje daty i czasu	67
Menu użytkownika	71
Formularz użytkownika	73
Niepisana konwencja nazywania elementów formularza	75
Przycisk sterujący — CommandButton	75
Etykieta — Label	78

Pole tekstowe — TextBox	79
Lista rozwijana — ComboBox	82
Baza do zbierania ankiet	86
Metody .GetOpenFilename i .GetSaveAsFilename	87
Pętla For Each ... Next	90
15	92
Zmienne tablicowe	92
Prywatny moduł arkusza — obsługa zdarzeń	99
Tablice dynamiczne	100
Funkcje tablicowe	101
Gramy dalej, czyli kostka Rubika	109
ParamArray, a funkcje o zmiennej liczbie argumentów	124
Import plików TXT	127
Metody .OpenText i .SaveAs	130
Instrukcja Open	136
Pliki sekwencyjne	136
Pozostałe funkcje przydatne w operowaniu na plikach	137
Przyciski opcji — OptionButtons	141
Funkcje sterujące	143
Pole wyboru — CheckBox	145
Pliki o dostępie bezpośrednim	151
Rekordowy typ danych — instrukcja Type	152
Kalendarz miesięczny	159
Funkcja obliczająca polskie święta	166
Eksport modułu do pliku *.bas	168
Przełączanie obrazków	171
Import modułu z pliku *.bas	176
Harmonogram	176
Formatowanie warunkowe	186
EN/PL	188
Liczby słownie	195
Mówimy po angielsku	207
Planowanie roczne	211
Walidacja komórki	223
Etykieta linii. Instrukcja GoTo etykieta	233
Instrukcje obsługi błędów On Error i Resume	234
5 kości	237
Ocena potencjału	247
Metoda .Sort, a obiekt Sort	266
Prezentacja wyników oceny	269
Własne menu	293
Do Worda	296
Funkcje obiektowe	298
Przez Lotus Notes	300
Odsetki karne	304
Funkcje Finansowe	311
Dialogi excelowe	315
1000 dobierany	317
Wykaz plików z rozwiązaniami	334
Na zakończenie	335

Historia

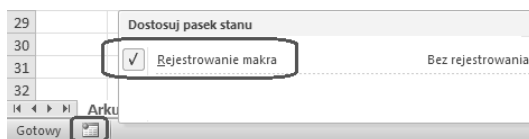
Dawno, dawno temu, kiedy lewica demokratyczna nazywała się jeszcze przewodnią siłą narodu, w czasach gdy głosowania do Sejmu i Rad Narodowych odbywały się bez skreśleń i to obowiązkowo na kandydatów Frontu Jedności Narodu, a frekwencja wyborcza bezdyskusyjnie sięgała 99,99 %. W tamtych to odległych czasach w roku 1963 powstał język programowania BASIC (**B**eginners **A**ll—**P**urpose **S**ymbolic **I**nstruction **C**ode). Język ten z założenia miał umożliwić pisanie programów osobom niezajmującym się zawodowo programowaniem. Parę lat później, w Pewex—ach (to takie najwyższe stadium sklepu w czasach PRL—u), pojawiły się pierwsze 8—bitowe komputery dostępne na kieszeń przeciętnego Polaka — Commodore, Spectrum, Atari. Wszystkie one miały wbudowany interpreter języka BASIC bezpośrednio w pamięci ROM — wystarczyło włączyć komputer i już... Prostowno Basic'a sprawiła, że nie powiodły się próby zastąpienia go bardziej dydaktycznym językiem Logo, czy też bardziej strukturalnym i akademickim językiem Pascal. Pozycja Basic'a była na tyle silna, że kolejne systemy operacyjne pisane dla PC—ta przez Microsoft obowiązkowo posiadały lepszy lub gorszy interpreter tego języka.

Nic dziwnego, że niedługo po pojawieniu się systemu Windows, już w roku 1991 powstała wersja Visual Basic, czyli BASIC połączony ze środowiskiem programowania okienkowego. Dla porządku należy tu podkreślić, że obecny Visual Basic poza podstawami samej składni ma niewiele wspólnego z "pierwotnym" Basic'em. Współczesny Visual Basic, to w pełni profesjonalne narzędzie, pozwalające na pisanie programów z wykorzystaniem wszelkich nowoczesnych reguł ich tworzenia, łącznie z techniką programowania strukturalnego i obiektowego.

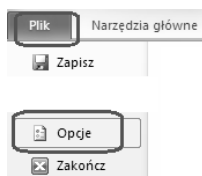
Na bazie Visual Basic'a oparty został język VBA (Visual Basic for Applications), czyli Visual Basic dołączony do aplikacji Microsoft Office (takich jak: Access, Excel, Word, Project...) — miał on ułatwić samodzielne rozszerzanie programu przez użytkowników poprzez tworzenie tzw. makr, czyli listy instrukcji, które aplikacja może automatycznie wykonać.

Przygotowanie środowiska

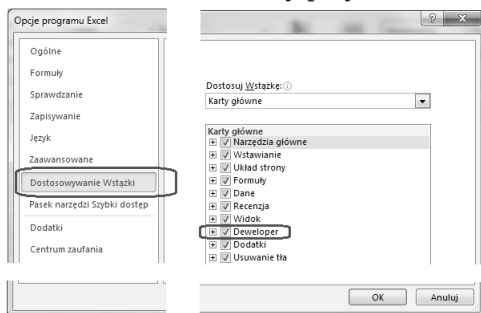
Po pierwsze warto postarać się, aby w pasku stanu (u dołu ekranu) znajdował się przycisk uruchamiania rejestratora makr. W tym celu klikamy na pasku stanu prawym przyciskiem myszy i włączamy pole wyboru — (CheckBox) przy pozycji **Rejestrowanie makra**.



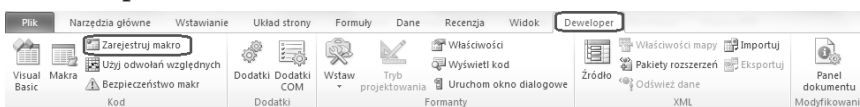
Po drugie — przy pracy z makrami przydatna też jest wstążka **Developer**. Aby ją przywołać na ekran klikamy przycisk **Plik**, a w ukazującym się menu klikamy pozycję **Opcje**:



Teraz w oknie opcji excelowych wybieramy w części nawigacyjnej (po lewej stronie) pozycję **Dostosowywanie Wstążki**, w prawym okienku włączamy pole wyboru — CheckBox **Developer** i na zakończenie klikamy przycisk OK:



W efekcie powinniśmy zaobserwować pojawienie się dodatkowej wstążki menu, o nazwie **Developer**:



Wykaz plików pomocniczych

Wszystkie omawiane dalej przykłady można wykonać samodzielnie od podstaw. Lecz jeśli ktoś nie ma ochoty wklepywać danych, ułatwieniem są pliki pomocnicze dostępne na załączonej tylko do wydania papierowego płycie CD. Znajdują się tam pliki:

- _macro1_raport.xlsx
- _macro2_tabliczka.xlsx
- _macro3_usuwanie.xlsx
- _macro4_wycinanie.xlsx
- _macro6_dzialania.xlsx
- _macro7_przeglady.xlsx
- _macro8_formularz.xlsx
- _macro9_ankieta1.xlsx
- _macro9_ankieta2.xlsx
- _macro9_ankieta3.xlsx
- _macro9_ankieta4.xlsx
- _macro9_ankieta5.xlsx
- _macro9_ankiety_baza.xlsx
- _macro12_funkcje_paramarray.xlsx
- _macro13_import_dane_delimited.txt
- _macro13_import_dane_fixed_width.txt
- _macro14_pliki_sekwencyjne.xlsx
- _macro15_pliki_o_dostepie_bezposrednim.xlsx
- _macro15_plik_o_dostepie_bezposrednim.dat
- _macro16_kalendarz.xlsx
- _macro16_kalendarz_foto01.jpg
- _macro16_kalendarz_foto02.jpg
- _macro16_kalendarz_foto03.jpg

_macro16_kalendarz_foto04.jpg
_macro16_kalendarz_foto05.jpg
_macro16_kalendarz_foto06.jpg
_macro16_kalendarz_foto07.jpg
_macro16_kalendarz_foto08.jpg
_macro16_kalendarz_foto09.jpg
_macro16_kalendarz_foto10.jpg
_macro16_kalendarz_foto11.jpg
_macro16_kalendarz_foto12.jpg
_macro17_modPLHolidays.bas
_macro18_liczby_slovnice.xlsx
_macro21_narzedzie_potencjalu.xlsx
_macro24_odsetki_karne.xlsx
_macro26_dobierany_1000_11.BMP
_macro26_dobierany_1000_12.BMP
_macro26_dobierany_1000_13.BMP
_macro26_dobierany_1000_14.BMP
_macro26_dobierany_1000_15.BMP
_macro26_dobierany_1000_16.BMP
_macro26_dobierany_1000_21.BMP
_macro26_dobierany_1000_22.BMP
_macro26_dobierany_1000_23.BMP
_macro26_dobierany_1000_24.BMP
_macro26_dobierany_1000_25.BMP
_macro26_dobierany_1000_26.BMP
_macro26_dobierany_1000_31.BMP
_macro26_dobierany_1000_32.BMP
_macro26_dobierany_1000_33.BMP
_macro26_dobierany_1000_34.BMP
_macro26_dobierany_1000_35.BMP
_macro26_dobierany_1000_36.BMP
_macro26_dobierany_1000_41.BMP
_macro26_dobierany_1000_42.BMP
_macro26_dobierany_1000_43.BMP
_macro26_dobierany_1000_44.BMP
_macro26_dobierany_1000_45.BMP
_macro26_dobierany_1000_46.BMP

Makro1

A teraz nasze pierwsze zadanie. Powiedzmy, że stale od klienta otrzymujemy następujące raporty excelowe:


	A	B	C	D	E
1	JEDEN	DWA	TRZY	CZTERY	PIĘĆ
2	1	2	3	4	5
3	1	2	3	4	5
4	1	2	3	4	5

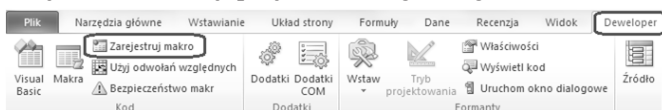
A szef życzy sobie, aby doprowadzić je do nieco innej postaci:

- kolumna z napisem "Pięć" ma trafić na początek tabeli,
- kolumny z napisami "Dwa" i "Trzy" należy usunąć,
- nagłówki należy wyśrodkować,
- nad tabelką trzeba dodać nowy wiersz,
- w komórce **A1** tego dodanego wiersza mamy wpisać nazwę firmy, następnie należy ten napis wyśrodkować nad całą tabelką zlepiając komórki zakresu **A1:C1**, po czym zwiększyć rozmiar czcionki na 16, a samą czcionkę pogubić i pochylić.

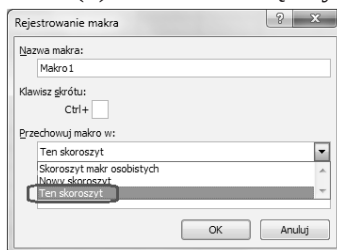
	A	B	C	D
1	ROZBÓJ SA			
2	PIĘĆ	JEDEN	CZTERY	
3	5	1	4	
4	5	1	4	
5	5	1	4	

Rejestrator makr

Ten pierwszy raz skorzystamy z pomocy rejestratora. W tym celu klikamy przycisk **Zarejestruj makro** na pasku stanu u dołu ekranu — , lub korzystając ze wstążki **Developer**, w sekcji **Kod** klikamy przycisk **Zarejestruj makro**.



W ukazującym się okienku decydujemy o nazwie makra (lub pozostawiamy proponowaną nazwę **Makro#**). Nazwa makra może się składać z małych (a—z) i dużych (A—Z) liter, cyfr (0—9) oraz podkreślnika (_). Nie zaleca się używania polskich liter.



Z uruchamianiem makra można powiązać jakiś skrót klawiszowy, (choć nie jest łatwo znaleźć taką kombinację klawiszy, której Microsoft już do czegoś nie wykorzystuje)

Należy też wybrać gdzie ma być zapamiętany kod VBA tworzonoego makra, do wyboru mamy:

Skoroszyt makr osobistych — głęboko zaszyty specjalny plik do gromadzenia kodu tworzonych makr. (Raczej nie zachęcam do wyboru tej opcji — gromadząc w skoroszycie makr osobistych wszystkie nasze makra w końcu przestaniemy się orientować które z nich do czego służy, poz a tym, o tym pliku bardzo łatwo zapomnieć przy przenoszeniu danych na inny komputer...).

Ten skoroszyt — kod będzie zapisany w module podpiętym pod bieżący plik. (To jest ustawienie domyślne — i to jest chyba najlepszy wybór).

Nowy skoroszyt — kod będzie zapisany w module podpiętym pod dodatkowy (nowy) plik.

Kliknijmy wreszcie przycisk OK. — rejestrator został uruchomiony. Odtąd wszystko, co robimy może być użyte przeciwko nam. Stąd zalecany jest przede wszystkim spokój (nadgorliwość jest gorsza od faszyzmu). Pamiętajmy, że rejestracja nie odbywa się na czas — rejestrowane są kolejne zdarzenia związane z czynnościami, które wykonujemy...

Kliknijmy prawym przyciskiem myszy na nagłówku kolumny **A** i w ukazującym się menu wybierzmy **Wstaw** — dla wstawienia nowej kolumny:

	A	C	D	E
1	JEDEN	Y	CZTERY	PIĘĆ
2	1	3	4	5
3	1	3	4	5
4	1	3	4	5
5	1	3	4	5
6	1	3	4	5
7	1	3	4	5

Teraz kliknijmy nagłówek kolumny z nagłówkiem "PIĘĆ" — obecnie jest to już kolumna **F** a nie **E**.

	A	B	C	D	E	F	G
1		JEDEN	DWA	TRZY	CZTERY	PIĘĆ	
2		1	2	3	4	5	
3		1	2	3	4	5	
4		1	2	3	4	5	

Po czym podejżdzamy delikatnie pod brzeg zaznaczenia tak, aby wskaźnik myszy przyjął następującą dziwaczną postać:

Teraz klikamy lewy przycisk myszy i nie zwalniając go przesuwamy na lewo, aż do kolumny **A**:

	A	B	C	D	E
1	PIĘĆ	JEDEN	DWA	TRZY	CZTERY
2	5	1	2	3	4
3	5	1	2	3	4
4	5	1	2	3	4

Następnie zaznaczamy kolumny z napisami "DWA" i "TRZY" — czyli kolumny **C:D**. Klikamy prawy przycisk myszy i z ukazującego się menu wybieramy **Usuń**:

	A	B	C	D	F
1	PIĘĆ	JEDEN	DWA	TRZY	
2	5	1	2	3	
3	5	1	2	3	
4	5	1	2	3	
5	5	1	2	3	
6	5	1	2	3	
7	5	1	2	3	

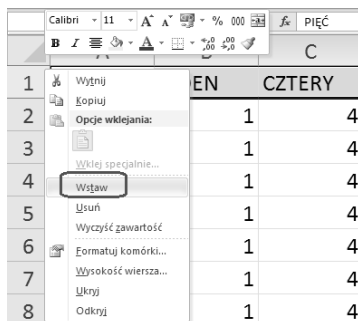
Teraz zaznaczamy wiersz nagłówkowy, czyli pierwszy wiersz:

	A	B	C
1	PIĘĆ	JEDEN	CZTERY
2	5	1	4
3	5	1	4
4	5	1	4

A na wstążce **Narzędzia główne**, w sekcji **Wyrównanie** klikamy przycisk wycentrowania w poziomie:



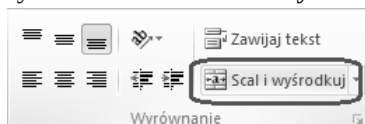
Następnie klikamy prawym przyciskiem na wierszu **1**, a w ukazującym się menu klikamy pozycję **Wstaw**:



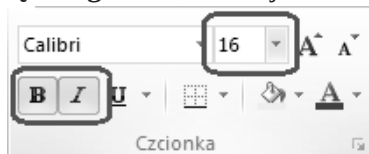
W pojawiającym się nowym wierszu ustawiamy się w komórce **A1** i wpisujemy nazwę firmy (np. "ROZBÓJ SA"). Następnie zaznaczamy zakres komórek nad tabelą, czyli **A1:C1**:

	A	B	C
1	ROZBÓJ SA		
2	PIĘĆ	JEDEN	CZTERY
3	5	1	4
4	5	1	4

A na wstążce **Narzędzia główne**, w sekcji **Wyrównanie** klikamy przycisk wyśrodkowujący w zakresie zaznaczonych kolumn — **Scal i wyśrodkuj**:



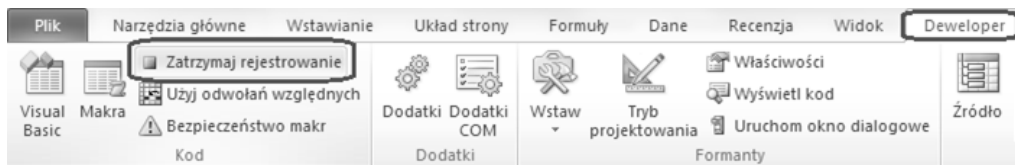
Teraz jeszcze pozostaje kosmetyka związana z samymi czcionkami: Zwiększamy rozmiar czcionki na **16**, włączamy pogrubienie i pochylenie (wszystko znajdziemy na wstążce **Narzędzia główne**, w sekcji **Czcionka**):



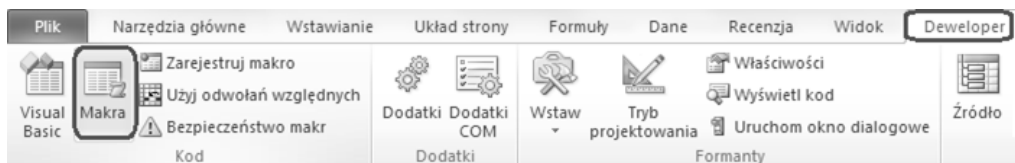
Nareszcie koniec:


	A	B	C	D
1	ROZBÓJ SA			
2	PIĘĆ	JEDEN	CZTERY	
3	5	1	4	
4	5	1	4	

Zatrzymujemy rejestrator używając przycisku w pasku stanu — , lub korzystając ze wstążki **Developer** klikamy przycisk **Zatrzymaj rejestrowanie**:

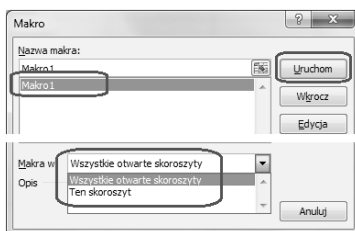


Aby uruchomić nasze nowe makro ustawmy się na surowym raporcie (z kolumnami JEDEN, DWA, TRZY, CZTERY, PIĘĆ) i kliknijmy na wstążce **Developer** przycisk **Makra**:



Lub po prostu — skorzystajmy z kombinacji klawiszy .

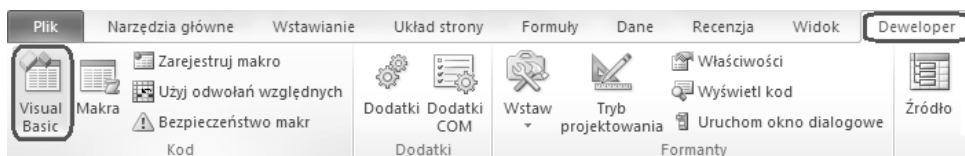
W ukazującym się okienku decydujemy czy chcemy oglądać makra zapisane we wszystkich otwartych plikach czy tylko w konkretnym pliku, wreszcie podświetlamy interesującą nas makro i klikamy przycisk **Uruchom** aby je uruchomić:



Błysk, chwila cicho i zrobiło...

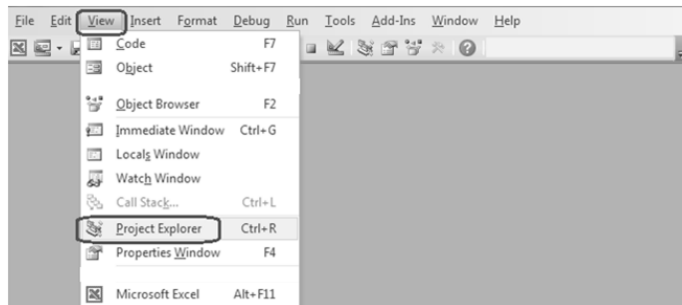
Edytor Visual Basic

To teraz przyjrzyjmy się jak to makro wygląda zapisane w kodzie **Visual Basic For Application**, czyli przejdźmy do edytora kodu. Najprościej przejść do edytora VBA klikając na wstążce **Developer** przycisk **Visual Basic** w sekcji **Kod**:

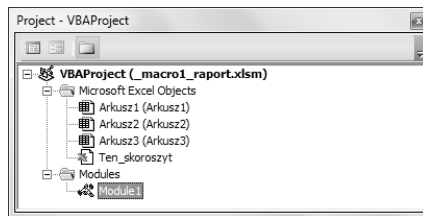


Możemy też użyć kombinacji klawiszy .

Jesteśmy w programie edytora VBA. Teraz postarajmy się, aby wyświetlały się te najbardziej przydatne okna, czyli okno projektu i okno modułu. W tym celu kliknijmy pozycję **View** w menu i tam wybierzmy pozycję **Project Explorer**:




Samo okienko projektu może pojawić się w postaci "fruującej" lub też w postaci "zadokowanej" przy którymś z brzegów ekranu:



(Dobrze jest je "zadokować" przesuwając mocno do lewego brzegu ekranu.)

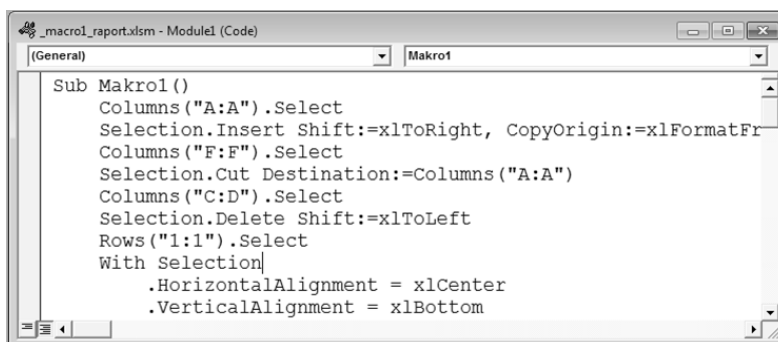
W górnej części okna projektu widzimy 3 przyciski:




Pierwszy od prawej — **Toggle Folders** —  — pozwala włączyć lub wyłączyć kategoryzację obiektów widocznych w okienku projektowym. Osobiście wolę widok z kategoryzacją — wyraźnie widać wtedy obiekty czysto excelowe (**Microsoft Excel Objects**) — arkusze i **Ten_skoroszyt** oraz moduły Visual Basic (**Modules**) czy wreszcie formularze użytkownika (**Forms**) lub moduły klas (**Class Modules**).

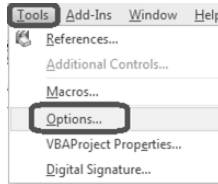
Przycisk środkowy — **View Object** —  — pozwala przejść do wskazanego w oknie projektowym obiektu.

Wreszcie przycisk lewy — **View Code** —  — pozwala przejść do okna modułu wskazanego w oknie projektowym obiektu (z treścią kodu VBA).

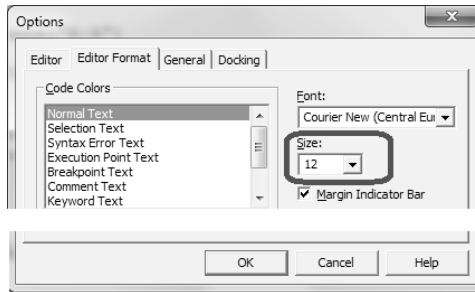
No dobrze. Ustawmy się na obiekcie **Module1** i kliknijmy ten przycisk z lewej — **View Code** — pojawi się okienko modułu **Module1**, a w nim treść naszego makra:





















Samo okienko **Module1** dobrze jest zmaksymalizować używając odpowiedniego przycisku —   , a ponieważ to będzie najważniejsze okno podczas naszej pracy z kodem VBA, to możemy je sobie lepiej przygotować, wybierając menu **Tools**, pozycja **Options**:



Teraz na zakładce **Editor Format** warto dostosować np. rozmiar czcionki tak, aby sobie nie męczył wzroku:



Poruszanie się po oknie modułu przypomina poruszanie się w typowym edytorze tekstu:

-   — **Left, Right** — przesunięcie kursora o znak w lewo, prawo
-   — **Down, Up** — przesunięcie kursora o linię do dołu, do góry
-   — **Home, End** — przesunięcie kursora na początek lub koniec linii
-   — **Page Down, Page Up** — przesunięcie kursora o ekran do dołu, do góry
-  +  — **Ctrl + Home** — przesunięcie kursora na początek dokumentu
-  +  — **Ctrl + End** — przesunięcie kursora na koniec dokumentu
-  — **Tab** — zrobienie wcięcia w tekście
-  +  — **Shift + Tab** — zlikwidowanie wcięcia w tekście
-  — **Enter** — utworzenie nowej linii w miejscu kursora
-  — **Delete** — skasowanie znaku na prawo od kursora
-  — **BackSpace** — skasowanie znaku na lewo od kursora

Wracając do samej treści którą widzimy w **Module1**, to całe nasze makro (jak w okładkach) mieści się między dwoma charakterystycznymi liniami:

```
Sub Makro1()
End Sub
```

Gdzie **Sub** — to skrót od słowa **subroutine** — procedura. W samej treści, między tymi linijkami rozpoznajemy znane angielskie słowa — są to zazwyczaj nazwy obiektów (kolekcji obiektów), metod — czynności wykonywanych na obiektach jak i właściwości tych obiektów. Spotykamy mniej więcej takie konstrukcje:

```
Obiekt.Metoda
Obiekt.Metoda Parametr:=wartość_parametru

Obiekt.Właściwość = wartość
Obiekt.Grupa_właściwości.Właściwość = wartość
```

Obiekty excelowe

Pośród obiektów (czy kolekcji obiektów) zauważymy te typowo excelowe np.:

Workbooks — kolekcja skoroszytów, ale Workbooks("Zeszyt1") — skoroszyt o nazwie "Zeszyt1"

Sheets — kolekcja arkuszy, ale Sheets("Arkusz1") — obiekt — arkusz o nazwie "Arkusz1", czy też Sheets(1) — obiekt — pierwszy arkusz danego skoroszytu

Columns — kolekcja kolumn, ale Columns("A:A") — kolumna "A", czy też Columns(1) — obiekt — pierwsza kolumna danego arkusza

Rows — kolekcja wierszy, ale Rows("1:1") — obiekt — wiersz o nazwie "1", czy też Rows(1) — obiekt — pierwszy wiersz danego arkusza

Cells — kolekcja komórek, ale Cells(3, 5) — obiekt — komórka na przecięciu 3 wiersza i 5 kolumny

Range("A1:C1") — obiekt — zakres komórek "A1:C1"

Obiekty bieżące:

ActiveCell — obiekt — aktywna komórka

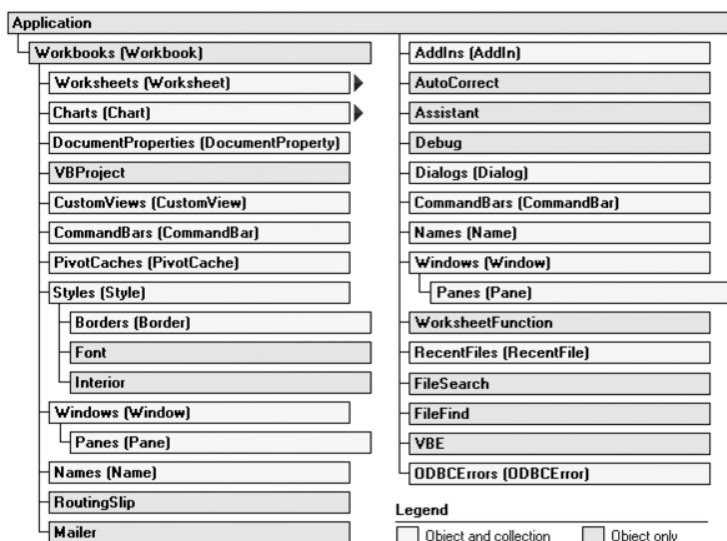
ActiveSheet — obiekt — aktywny arkusz

ActiveWorkbook — obiekt — aktywny skoroszyt

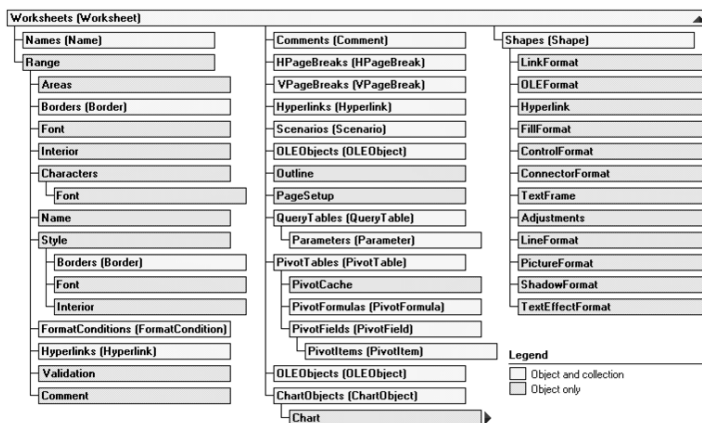
Obiekt zaznaczony:

Selection — obiekt — coś, co zostało zaznaczone

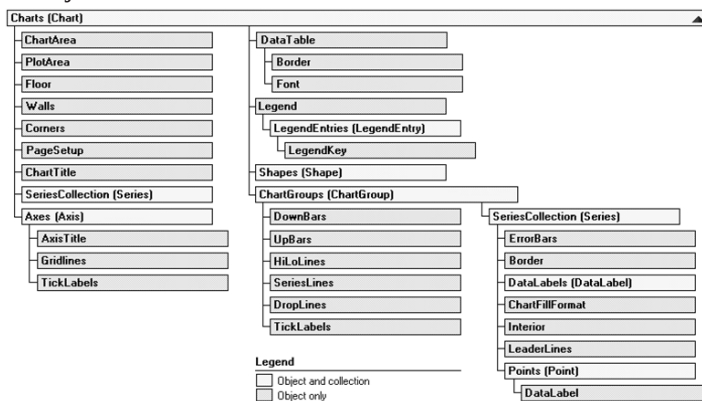
Przy czym obiekty też mają swoją hierarchię poczynając od najważniejszego obiektu (takiego "capo di tutti capi" wszystkich obiektów) — Application. A obiekt Application — to po prostu nasz wspaniały program — MS Excel...



Jeszcze dalej, po rozwinięciu kolekcji obiektów Worksheets:



Albo też kolekcja obiektów Charts:



Metody wykonywane na obiektach

Pośród metod (czynności) wykonywanych na obiektach spotykamy zarówno metody bez parametru, jak i takie, które zawierają pomocnicze parametry. Przy czym nie wszystkie parametry są obligatoryjne, niektóre są jedynie opcjonalne. Naciśnięcie klawisza **F1**, gdy stoimy kursorem na danej metodzie włącza okno pomocy — Help, gdzie zawsze możemy poznać szczegóły składni. Jeżeli używamy parametrów w składni danej metody, pisząc ich nazwy wprost, to między nazwą parametru a jego wartością należy użyć dwuznaku: ":=".

Obiekt `.Select` — metoda bezparametryczna — zaznacz obiekt, np.

`Columns(1).Select` — zaznacz pierwszą kolumnę.

`Columns("F:F").Select` — zaznacz kolumnę "F".

`Columns("C:D").Select` — zaznacz kolumny "C:D".

`Rows("1:1").Select` — zaznacz wiersz "1".

`Range("A1:C1").Select` — zaznacz zakres komórek "A1:C1".

Obiekt `.Merge` — metoda bezparametryczna — złącz komórki obiektu, np.:

`Range("A1:C1").Merge` — złącz zakres komórek "A1:C1".

Obiekt `.Insert` — metoda parameryczna — wstaw, przy czym parametr `Shift` jest tu parametrem opcjonalnym, np.:

`Columns(1).Insert Shift:=xlToRight` — wstaw kolumnę w miejscu kolumny pierwszej, a pozostałe kolumny przesunąć na prawo (`xlToRight` jest stałą excelową zastępującą liczbę, która w excelowym VBA decyduje o przesunięciu na prawo).

`Rows(1).Insert Shift:=xlDown` — wstaw wiersz w miejscu pierwszego wiersza, a pozostałe wiersze przesunąć do dołu (`xlDown` jest stałą excelową zastępującą liczbę, która w excelowym VBA decyduje o przesunięciu do dołu).

Obiekt `.Cut` — metoda parametryczna — wytnij obiekt (z zamiarem późniejszego wstawienia), przy czym parametr `Destination` jest tu parametrem opcjonalnym (jeśli go nie użyjemy wycięty obiekt trafi do schowka pamięci Clipboard), np.:

`Columns("F:F").Cut Destination:=Columns("A:A")` — kolumnę "F" wyciąć i wstawić w miejsce kolumny "A".

To samo ale z zaangażowaniem Clipboard'u wymagałoby dwóch linijek kodu:

`Columns(6).Cut` — wytnij szóstą kolumnę (i schowaj do schowka pamięci)

`Columns(1).Paste` — wklej do pierwszej kolumny zawartość schowka pamięci

Obiekt `.Delete` — metoda parametryczna — usuń, przy czym parametr `Shift` jest tu parametrem opcjonalnym, np.:

`Columns("C:D").Delete Shift:=xlToLeft` — usuń kolumny "C:D", a pozostałe kolumny dosunąć do lewej (aby nie było dziury w arkuszu) (`xlToLeft` jest stałą excelową zastępującą liczbę, która w excelowym VBA decyduje o przesunięciu na lewo)

Instrukcja wiążąca With

Śledząc kod utworzony przez rejestrator zauważymy taką dziwną konstrukcję ze słowami: `With ... End With` — to się nazywa instrukcja wiążąca. Np.:

```
With JakiśTamObiekt
    .HorizontalAlignment = xlCenter
    .VerticalAlignment = xlBottom
    .WrapText = False
    .Orientation = 0
    .AddIndent = False
    .IndentLevel = 0
    .ShrinkToFit = False
    .ReadingOrder = xlContext
    .MergeCells = False
End With
```

I jest równoważna konstrukcji składniowej:

```
JakiśTamObiekt.HorizontalAlignment = xlCenter
JakiśTamObiekt.VerticalAlignment = xlBottom
JakiśTamObiekt.WrapText = False
JakiśTamObiekt.Orientation = 0
JakiśTamObiekt.AddIndent = False
```

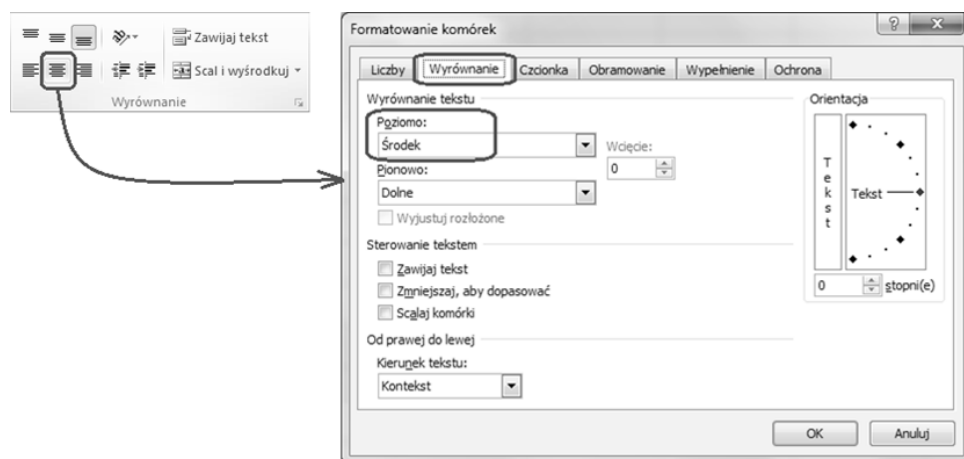
```

JakiśTamObiekt.IndentLevel = 0
JakiśTamObiekt.ShrinkToFit = False
JakiśTamObiekt.ReadingOrder = xlContext
JakiśTamObiekt.MergeCells = False

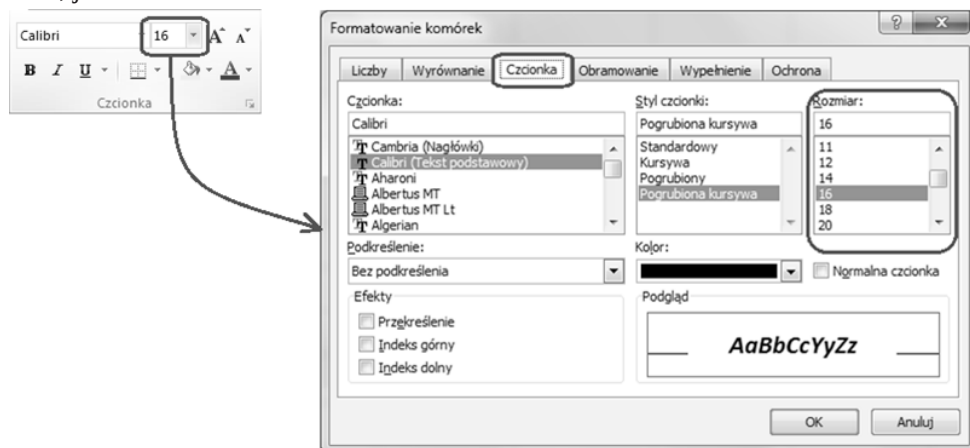
```

Jak widać instrukcja ta jest przydatna, kiedy na konkretnym obiekcie mamy do wykonania wiele różnych metod lub chcemy zmienić wiele różnych właściwości. Po prostu instrukcja wiążąca zaoszczędza nam wielokrotnego przepisywania nazwy obiektu przed każdą metodą i każdą właściwością...

Przy okazji powyższy fragment jest przykładem niedoskonałości excelowego rejestratora makr. Po kliknięciu przez nas przycisku wycentrowania w poziomie, rejestrator uznał za stosowne sumienne przypisanie konkretnych wartości wszystkich właściwości wybranego obiektu, jakie znalazł w okienku **Formatowanie komórek** — zakładka **Wyrównanie**, (choć nikt go o to nie prosił):



Podobne "zagranie" możemy zaobserwować w miejscu gdzie zapragniemy tylko zmienić rozmiar czcionki i nieopatrznie skorzystaliśmy z listy rozwijanej — rejestrator uznał za stosowne "bezmieślnie" przypisać konkretne wartości wszystkich właściwości wybranego obiektu, jakie znalazł w okienku **Formatowanie komórek** — zakładka **Czcionka**:



Co przełożyło się na odpowiedni kod VBA przypisujący właściwości z grupy `Font` konkretnego obiektu:

```
With JakiśTamObiekt.Font
    .Name = "Calibri"
    .Size = 16
    .Strikethrough = False
    .Superscript = False
    .Subscript = False
    .OutlineFont = False
    .Shadow = False
    .Underline = xlUnderlineStyleNone
    .ThemeColor = xlThemeColorLight1
    .TintAndShade = 0
    .ThemeFont = xlThemeFontMinor
End With
```

A gdybyśmy nie chcieli używać instrukcji wiążącej, to wyglądałoby to tutaj tak:

```
JakiśTamObiekt.Font.Name = "Calibri"
JakiśTamObiekt.Font.Size = 16
JakiśTamObiekt.Font.Strikethrough = False
JakiśTamObiekt.Font.Superscript = False
JakiśTamObiekt.Font.Subscript = False
JakiśTamObiekt.Font.OutlineFont = False
JakiśTamObiekt.Font.Shadow = False
JakiśTamObiekt.Font.Underline = xlUnderlineStyleNone
JakiśTamObiekt.Font.ThemeColor = xlThemeColorLight1
JakiśTamObiekt.Font.TintAndShade = 0
JakiśTamObiekt.Font.ThemeFont = xlThemeFontMinor
```

Właściwości obiektów

Przyjrzyjmy się, w jaki sposób zmieniane są różne właściwości obiektów. Naciśnięcie klawisza **F1**, gdy stoimy kursorem na danej właściwości włącza okno pomocy — Help, gdzie zawsze możemy poznać dopuszczalne wartości wskazanej właściwości jak i ewentualne stałe, które Microsoft przygotował do wykorzystania przy nadawaniu jej nowej wartości.

Obiekt `.HorizontalAlignment = wartość` — właściwość decydująca o wyrównaniu w poziomie komórek obiektu, np.:

```
Rows(1).HorizontalAlignment = xlCenter — wpisy w komórkach pierwszego wiersza wyrównaj w poziomie do środka. (xlCenter jest stałą excelową zastępującą jakąś liczbę, która w excelowym VBA decyduje o wyśrodkowaniu, podobnie: xlLeft — dosunięcie do lewej, a xlRight — dosunięcie do prawej).
```

```
Selection.HorizontalAlignment = xlCenter — wycentruj wpisy w zaznaczeniu.
```

Obiekt `.Font.Size` = wartość — właściwość decydująca o rozmiarze czcionki w komórkach obiektu, np.:

`Range("A1:C1").Font.Size = 16` — rozmiar czcionki w komórkach zakresu "A1:C1" ustaw na 16.

Obiekt `.Font.Bold` = wartość — właściwość decydująca o włączeniu/wyłączeniu pogrubienia czcionki w komórkach obiektu.

Obiekt `.Font.Italic` = wartość — właściwość decydująca o włączeniu/wyłączeniu pochylenia czcionki w komórkach obiektu, np.:

`Selection.Font.Italic = True` — włącz pochylenie w komórkach zaznaczonego obiektu.

Obiekt `.FormulaR1C1` = wartość — właściwość decydująca o formule wpisanej do komórek obiektu (formule wpisanej przy sposobie adresowania R1C1, a po polsku: W1K1), np.:

`Range("A1").FormulaR1C1 = "=RC[1]+RC[2]"` — do komórki "A1" wpisano formułę (w formacie R1C1) dodającą komórkę z tego samego wiersza, ale o 1 kolumnę na prawo oraz komórkę z tego samego wiersza, ale o 2 kolumny na prawo, (czyli B1 i C1).

Obiekt `.Formula` = wartość — właściwość decydująca o formule wpisanej do komórek obiektu (formule wpisanej przy standardowym sposobie adresowania gdzie kolumny oznaczamy literowo, a wiersze numerycznie), np.:

`Cells(3, 1).Formula = "=A1+A2"` — do komórki na przecięciu wiersza nr 3 i kolumny nr 1 (czyli do komórki "A3") wpisano formułę dodającą komórkę "A1" oraz komórkę "A2".

Obiekt `.Value` = wartość — właściwość decydująca o wartości wpisanej do komórek obiektu (przy czym teksty wpisujemy w cudzysłowach, liczby bez nawiasów ale jako separatora dziesiętnego zawsze używamy kropki, a daty wpisujemy wewnątrz znaków #), np.:

`ActiveCell.Value = "Jakiś tekst"` — napis wpisany do aktywnej komórki.

`Cells(1, 11).Value = 13.5` — liczba wpisana do komórki na przecięciu wiersza nr 1 i kolumny nr 11 (czyli do komórki "K1").

`Range("J1").Value = #5/22/2014#` — data wpisana do komórki "J1".

komentarz

W wierszach kodu VBA możemy dopisywać komentarze. Znak apostrofa (') jest interpretowany, jako początek komentarza. Wszystko, co się znajduje po znaku ' jest do końca linii ignorowane przez interpreter języka VBA.

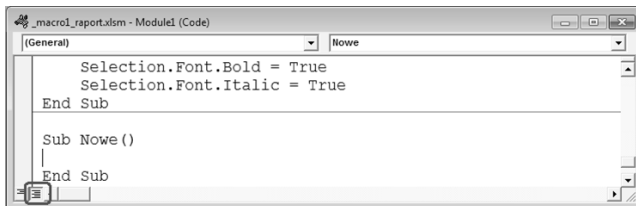
Warto pamiętać, że jeżeli chcemy chwilowo wyłączyć jakąś linię z programu — wystarczy dopisać przed nią znak apostrofa (').

Nowe makro

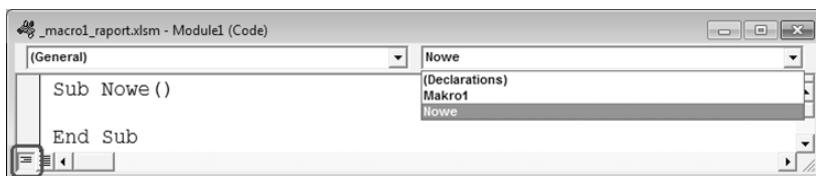
Najwyższy czas, aby pokazać wyższość człowieka nad "bezmyślnym" rejestratorem makr. Napijemy własne makro, które będzie prostsze w konstrukcji, szybsze i ogólnie — lepsze, bo nasze...!

Przejdźmy na koniec kodu w naszym **Module1** — **Ctrl** + **End**. Zróbmy parę linii odstepu — **Enter**. Teraz napiszmy: `Sub Nowe` **Enter**. Edytor sam dopisze nawiasy za nazwą makra, dopisze liniijkę zamykającą (na razie pustą) treść makra — `End Sub` oraz ustawi nasz kursor w środku — tam gdzie spodziewa się treści makra.

Zależnie od ustawienia przełącznika widoków w lewym dolnym rogu edytora: Zobaczymy wszystkie makra, jakie są w module, pooddzielane poziomymi kreskami — jeżeli przełącznik widoków znajduje się w pozycji 2 — **Full module View**:



Albo zobaczymy jedynie bieżące makro — `Nowe()` — jeżeli przełącznik widoków znajduje się w pozycji 1 — **Procedure View**:



(W tym ostatnim widoku, aby przełączyć się do treści innej procedury należy skorzystać z prawej listy rozwijanej u góry ekranu.) No to napiszmy wreszcie treść nowego makra:

```
Sub Nowe ()
    Columns(1).Insert 'wstaw kolumnę
                        'przenieś kolumnę F w miejsce A:
    Columns(6).Cut Destination:=Columns(1)
    Columns("C:D").Delete 'usuń kolumny C:D
    With Rows(1)
        'w stosunku do wiersza 1
        'wyśrodkuj wyrównanie w poziomie:
        .HorizontalAlignment = xlCenter
        .Insert 'wstaw wiersz
    End With
        'wpisz w komórce A1:
    Cells(1, 1).Value = "AFERAX sp zoo"
    With Range("A1:C1") 'w stosunku do zakresu A1:C1:
        .Merge 'zlep
        'wyśrodkuj wyrównanie w poziomie:
        .HorizontalAlignment = xlCenter
        With .Font
            'jeżeli chodzi o czcionki:
            .Size = 16 'ustaw rozmiar czcionki na 16
            .Bold = True 'włącz pogrubienie
            .Italic = True 'włącz pochylenie
        End With
    End With
End Sub
```

Podczas pisania możemy zauważyć, że po przejściu do nowej linii automatycznie sprawdzana jest poprawność składni i dokonywane formatowanie, czyli zmiana wielkości niektórych liter w słowach kluczowych. Dodatkowo opuszczona linijka zyskuje kolory. Przy czym:

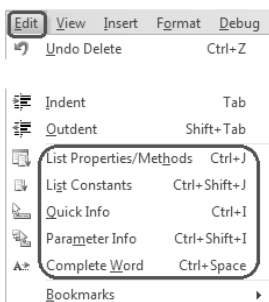
Kolorem **niebieskim** oznaczane są **słowa kluczowe**.

Kolorem **zielonym** oznaczane są **komentarze**.

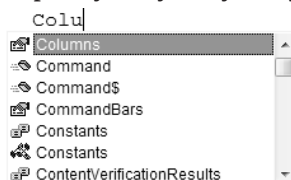
Kolorem **czerwonym** oznaczane są **błędne linie**.

Ułatwienia

Podczas pisania kodu możemy skorzystać z narzędzi, które uczynią naszą pracę przyjemniejszą — wszystkie one są dostępne w menu **Edit** edytora VBA:



Zamiast pracować wpisywać nazwę właściwości, czy procedury (a nawet obiektu), można skorzystać z **List Properties/Methods** — w skrócie **Ctrl+J** — **Ctrl** + **J**. Wystarczy że zaczniemy pisać nazwę obiektu, po czym wykorzystamy kombinację klawiszy **Ctrl** + **J**.



Inny sposób, to po napisaniu nazwy obiektu napisać kropkę:



Teraz posługując się suwakiem po prawej stronie ukazującej się listy (lub korzystając z klawiszy kursora **↓** **↑**) dochodzimy do interesującej nas pozycji, po czym możemy użyć klawisza:

Tab — aby wstawić wybrany element i pozostać w bieżącej linii — **Tab**.

Enter — aby wstawić wybrany element i przejść do nowej linii — **Enter**.

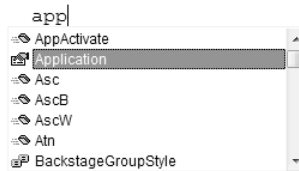
Esc — aby zrezygnować ze wstawiania — **Esc**.

Zamiast zastanawiać się, jaka jest nazwa stałej, którą chcemy przypisać do wybranej właściwości obiektu wystarczy skorzystać z **List Constants** — w skrócie **Shift+Ctrl+J** — **Shift** + **Ctrl** + **J**.

```
ActiveWindow.View=
```

- xlNormalView
- xlPageBreakPreview
- xlPageLayoutView

Aby dokończyć rozpoczęty wyraz też możemy skorzystać z ułatwienia **Complete Word** — w skrócie **Ctrl+Spacja**.



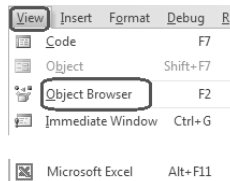
Zamiast zastanawiać się, jakie parametry ma konkretna metoda używana dla danego obiektu, wystarczy skorzystać z **Parameter Info** — w skrócie **Ctrl+Shift+I** — **Ctrl** + **Shift** + **I**. Otrzymamy podgląd wykazu wymaganych parametrów:

```
ActiveCell.Offset (
```

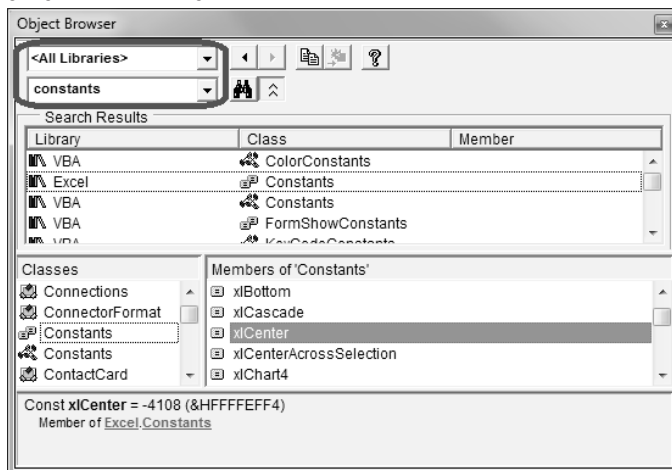
Offset([RowOffset], [ColumnOffset]) As Range

Przeglądarka obiektów

Jeżeli chcemy sprawdzić, jakie są dostępne obiekty, metody, właściwości, stałe, funkcje, itp. To najlepszą metodą będzie skorzystanie z przeglądarki obiektów — w menu **View** pozycja **Object Browser** lub klawisz skrótu — **F2** — **F2**.



Pierwsza lista rozwijana pozwala zdecydować czy wyszukujemy we wszystkich bibliotekach czy np. tylko w Excelu. Druga lista rozwijana pozwala wpisać szukaną frazę, a naciśnięcie **Enter** lub kliknięcie przycisku z lornetką rozpoczyna wyszukiwanie pozycji zawierających wskazany tekst.



Błędy

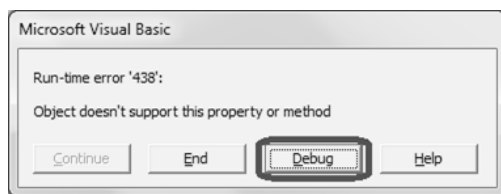
Mimo tych wszystkich wspaniałych narzędzi zapewne zdarzy się nam od czasu do czasu popełnić jakiś błąd. Albowiem mylić się jest rzeczą ludzką, (ale trzeba dopiero komputera, aby się to wszystko pomieszało)...

Powiedzmy że pisząc nasz kod bez korzystania z ułatwień zrobiliśmy błąd w pisowni właściwości `HorizontalAlignment` — np. napisaliśmy o jedno "e" za dużo:

```
With Rows(1)
    .horizontalalignement = xlCenter
```

Co więcej, takich błędów edytor jakby nie do końca zauważał podczas pisania, bo nawet po przejściu do nowego wiersza, ta błędna przecięź linia nie zmienia koloru na czerwony... Dobrą metodą jest pisanie kodu bez korzystania z klawisza `Shift`. Po przejściu do nowej linii we wszystkich obiektach, metodach, właściwościach i stałych pewne litery powinny się zmienić na duże (np.: H i A w `HorizontalAlignment`), jeżeli tego nie zrobiły, to jest to dla nas sygnał, że powinniśmy się dokładniej przyjrzeć temu, co napisaliśmy...

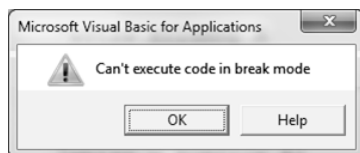
No dobrze, a co jeśli mimo wszystko nie zauważyliśmy błędu, przeszliśmy do Excela — `Alt+F11` i próbujemy uruchomić nasze makro — `Alt+F8` — **Nowe...**? Powinniśmy otrzymać komunikat błędu:



Zawsze gdy spotykamy taki komunikat warto wybrać przycisk **Debug**, bo to spowoduje, że edytor VBA sam pokaże nam linijkę kodu, która mu się nie podoba — zaznaczy ją na żółto:


```
Sub Nowe ()
    Columns(1).Insert
    Columns(6).Cut Destination:=Columns(1)
    Columns("C:D").Delete
    With Rows(1)
        .horizontalalignement = xlCenter
        .Insert
    End With
End Sub
```

Skoro błąd już namierzony, to trzeba go poprawić. Ale to nie wszystko...! Możemy łatwo sprawdzić, że próba uruchomienia makra znajdującego się w trybie przerwania spełni na niczym:



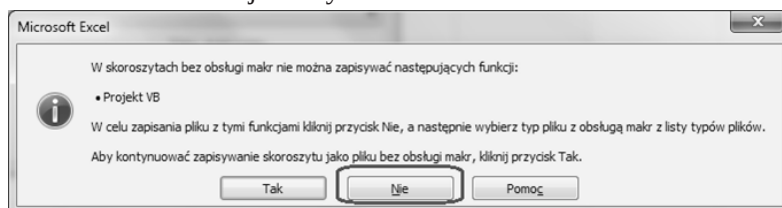
Należy jeszcze coś zrobić, aby pozbyć się tego żółtego koloru w naszym kodzie...! Mamy dwie możliwości do wyboru:

Albo decydujemy się na kontynuację makra po poprawce — **Continue** — 

Albo też decydujemy się na zatrzymanie makra — **Reset** — 

Zapis pliku z makrami

Jeżeli wszystko już działa poprawnie, to najwyższy czas zapisać nasz plik z makrami. Trzeba pamiętać, że od wersji MS-Office 2007 dokumenty zawierające makra mają inne rozszerzenia nazwy niż dokumenty, które makra nie zawierają. W przypadku Excela pliki bez makr mają standardowe rozszerzenie XLSX, a pliki z makrami mają standardowe rozszerzenie XLSM. Domyślnym typem zapisywanego pliku jest XLSX — stąd kliknięcie przycisku **Zapisz** — **Ctrl**+**S** — zaproponuje zapis w pliku wolnym od makr **Skoroszyt programu Excel (*.xlsx)**: Zapisz jako typ: Skoroszyt programu Excel (*.xlsx) .
Próba kontynuowania zapisu naszego pliku spowoduje pojawienie się na ekranie ostrzegawczego komunikatu "ostatniej szansy":



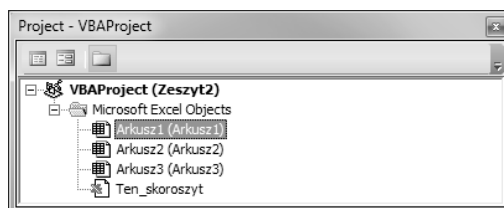
Jeżeli nie chcemy utracić naszego kodu VBA, to koniecznie wybierzmy przycisk **No**. Następnie w powracającym okienku **Zapisz jako** zmienimy typ pliku na **Skoroszyt programu Excel z obsługą makr (*.xlm)** Zapisz jako typ: Skoroszyt programu Excel z obsługą makr (*.xlm) .

Tabliczka mnożenia

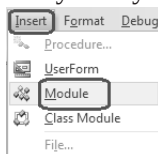
Chcemy napisać makro, które (bez łaski rejestratora) utworzy nam klasyczną tabliczkę mnożenia 10x10:

	A	B	C	D	E	F	G	H	I	J
1	1	2	3	4	5	6	7	8	9	10
2	2	4	6	8	10	12	14	16	18	20
3	3	6	9	12	15	18	21	24	27	30
4	4	8	12	16	20	24	28	32	36	40
5	5	10	15	20	25	30	35	40	45	50
6	6	12	18	24	30	36	42	48	54	60
7	7	14	21	28	35	42	49	56	63	70
8	8	16	24	32	40	48	56	64	72	80
9	9	18	27	36	45	54	63	72	81	90
10	10	20	30	40	50	60	70	80	90	100

W nowym pliku excelowym przejdźmy do edytora Visual Basic — **Alt**+**F11**. Pewien niepokój może u nas budzić fakt, że w oknie projektu nigdzie nie odnajdziemy ogólnego modułu w którym moglibyśmy napisać kod makra:



Poprzednio nie mieliśmy tego problemu, moduł od razu był...! A był dlatego, ponieważ rejestrator makr sam go sobie utworzył. Teraz ambitnie postanowiliśmy utworzyć makro samodzielnie... Skąd wziąć moduł? Nic trudnego — wystarczy w menu **Insert** kliknąć pozycję **Module**:



I już mamy moduł **Module1**, gdzie możemy pisać kod makra.

Oczywiście nie chodzi nam o makro składające się ze 100 linijek kodu, gdzie każda linijka będzie miała za zadanie wpisać jedną liczbę do tabliczki mnożenia. Podejrzmy do tego bardziej ambitnie — wykorzystamy instrukcję pętli `For ... Next`.

Pętla For ... Next

Instrukcja pętli `For ... Next` wykorzystywana jest wówczas gdy jakiś zestaw instrukcji chcemy powtórzyć zadaną ilość razy. Składnia instrukcji jest następująca:

```
For Licznik = wartość_początkowa To wartość_końcowa [Step krok]
...
instrukcje
...
Next [Licznik]
```


Działanie takiej pętli jest następujące: w linijce ze słowem `For Licznik` przyjmuje wartość równą wartości początkowej. Następnie wykonywany jest ciąg instrukcji aż do napotkania linijki ze słowem `Next`, gdzie `Licznik` zostaje zwiększony o wartość kroku (jeżeli nie umieścimy słowa `Step krok`, to domyślnie wartość kroku = 1). Teraz następuje powrót do linijki ze słowem `For`. Sprawdzane jest czy nowa wartość `Licznika` nie przekracza wartości końcowej. Jeśli nie, to znowu wykonywany jest ciąg instrukcji aż do napotkania linijki ze słowem `Next`, gdzie ponownie `licznik` zostaje zwiększony o wartość kroku i znowu powrót do linijki ze słowem `For`. W momencie, gdy nowa wartość `Licznika` jest większa od wartości końcowej następuje przeskok do pierwszej linii programu za linią ze słowem `Next`.

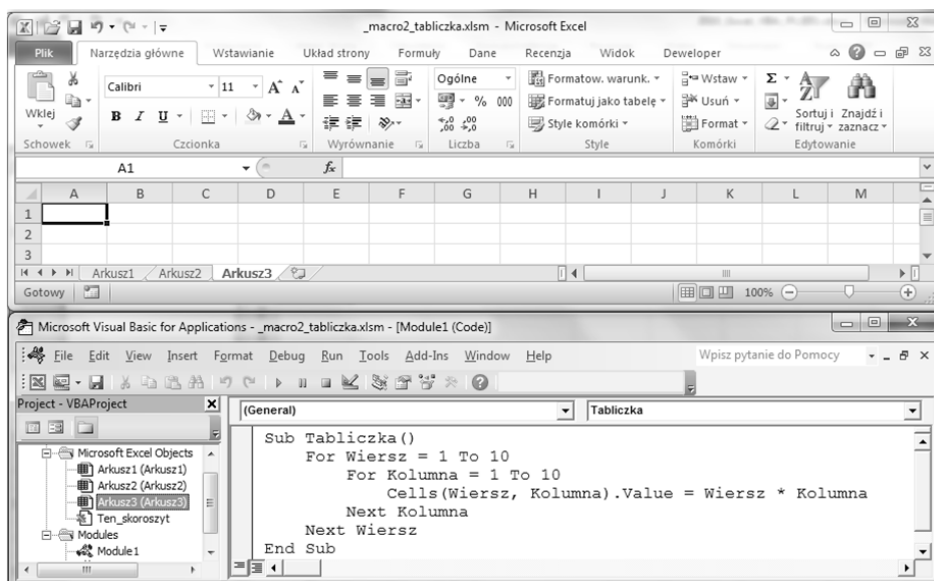
No dobrze — to napiszmy takie makro **Tabliczka**:

```
Sub Tabliczka()
    `dla licznika Wiersz zmieniającego się od 1 do 10:
    For Wiersz = 1 To 10
        `dla licznika Kolumna zmieniającego się od 1 do 10:
        For Kolumna = 1 To 10
            `wpisz do komórki na przecięciu
            `Wiersz i Kolumna,
            `wartość równą Wiersz x Kolumna:
            Cells(Wiersz, Kolumna).Value = Wiersz * Kolumna
        Next Kolumna            `zwiększ Kolumna o 1
    Next Wiersz                `zwiększ Wiersz o 1
End Sub
```


W naszej procedurze wykorzystujemy aż dwie pętle — jedna zagnieżdżona w drugiej. W pierwszej pętli licznik nazwaliśmy Wiersz, a w drugiej pętli licznik nazwaliśmy Kolumna. Analizując krok po kroku komórki (na przecięciu konkretnych numerów wierszy i kolumn) oraz ich wartości (wynikające z pomnożenia numeru wiersza i numeru kolumny), zauważymy:

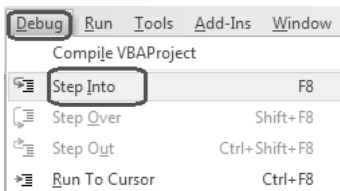
Pozycja	Wiersz	Kolumna	Komórka	Wiersz * Kolumna
1	1	1	A1	1
2	1	2	B1	2
...				
10	1	10	J1	10
11	2	1	A2	2
12	2	2	B2	4
...				
20	2	10	J2	20
...				
91	10	1	A10	10
92	10	2	B10	20
...				
100	10	10	J10	100


Aby lepiej przyjrzeć się jak działają w naszym programie pętle, spróbujemy ustawić na ekranie oba okna (zarówno okno Excela jak i okno edytora VBA) sąsiadująco. Przyda nam się użycie środkowego przycisku z prawego górnego rogu każdego ze wspomnianych okien — . Interesuje nas uzyskanie mniej więcej takiego układu:



Teraz, uruchomimy krokowe wykonanie naszego makra. W tym celu aktywujemy okno edytora VBA, tam ustawmy się kursorem na linii początkowej naszej procedury,

czyli w wierszu zawierającym Sub Tabliczka() i w menu **Debug** kliknijmy pozycję **Step Into** — w skrócie klawisz **F8** — .

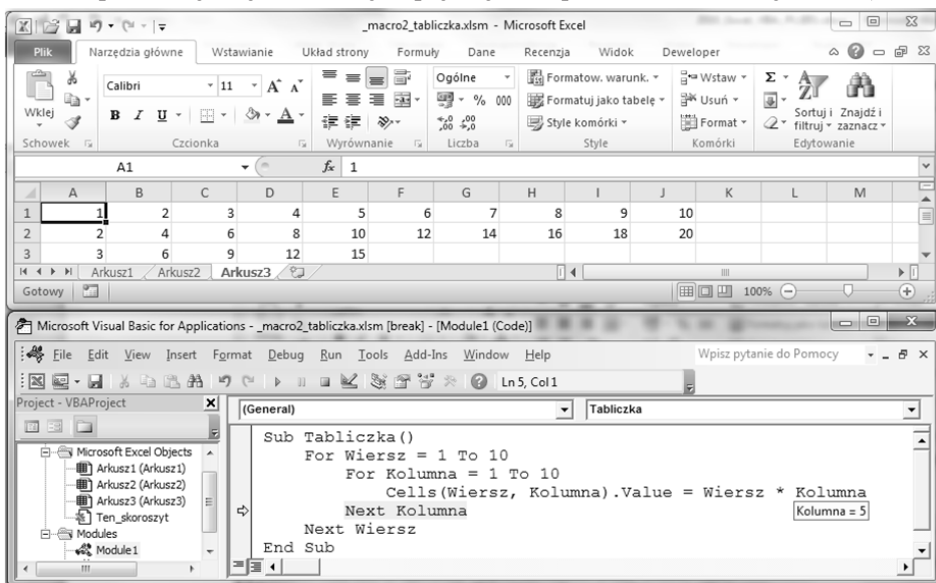


A teraz naciskajmy kolejno klawisz  i za każdym naciśnięciem obserwujmy, co cię dzieje na ekranie Excela oraz w którym miejscu kodu w module VBA znajduje się żółte podświetlenie. Jeśli chcemy w danym momencie znać chwilową wartość licznika **Wiersz** lub licznika **Kolumna**, to wystarczy podjechać kursorem myszy do dowolnego wystąpienia żądanego napisu (**Wiersz** lub **Kolumna**) w kodzie naszej procedury, a po chwili ukáže się tam prostokątna etykieta z informacją. Np.:

Wiersz = 3

Kolumna = 5

Zauważmy, że zarówno **Wiersz** jak i **Kolumna** może w naszym przykładzie przyjmować wartości od 1 do 11 (tak do 11, bo dopiero przekroczenie końcowej wartości — 10 spowoduje przeskok do pierwszej linijki kodu występującej za odpowiednią instrukcją Next.)



Jak już nam się znudzi naciskanie klawisza , to możemy:

Szybko dokończyć makro klikając — **Continue:**   

Albo zatrzymać dalsze wykonanie makra — **Reset:**  


Zmienne

Wszystko pięknie, ale zastanawia nas, jakim cudem VBA rozumie jakieś polskie słowa: **Wiersz**, **Kolumna**? Jak to właściwie jest? Otóż każdy taki napotkany w kodzie "nietypowy" wyraz VBA traktuje, jako nazwę pewnej zmiennej. Program wychodzi z założenia, że użytkow-

nik chce nazwać pewien obszar pamięci, aby gromadzić tam jakieś dane. Jakie...? To tylko użytkownik może wiedzieć — na wszelki wypadek rezerwowany jest największy obszar pamięci, na jaki VBA może sobie w przypadku zmiennych pozwolić. Po prostu VBA nie wie czy chcemy w zmiennej przechowywać tekst, liczbę, datę a może odwołanie do obiektu i na wszelki wypadek przydziela maksimum tego, co może...?

To jest poważna wada. Nieoszczędne gospodarowanie pamięcią. Kolejną wadą takiego niejawnego wprowadzania zmiennych jest to, że w przypadku pomyłki (zwykłej literówki) możemy mieć zaskakujące wyniki działania programu. Powiedzmy, że zrobiliśmy błąd literowy w czwartej linijce naszego makra i zamiast planowanego **Kolumna** napisaliśmy **Kolumn**:

```
Cells(Wiersz, Kolumna).Value = Wiersz * Kolumn
```

Przejdźmy do Excela —  i spróbujmy uruchomić nasze makro —  ...? Na ekranie zobaczymy coś takiego:

	A	B	C	D	E	F	G	H	I	J
1	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	0

Same zera. Żadnego komunikatu o błędzie — czyli naszemu VBA wszystko się podoba tylko, że wynik działania makra trochę odbiega od tego, czego się spodziewaliśmy...

Po prostu VBA uznał, że mamy w naszej procedurze 3 zmienne: **Wiersz**, **Kolumna** i **Kolumn**. O ile dwie pierwsze mają w każdej chwili przypisane w pętlach konkretne wartości, to do trzeciej z nich nie przypisano nic, a to znaczy, że domyślnie jej wartość wynosi 0.

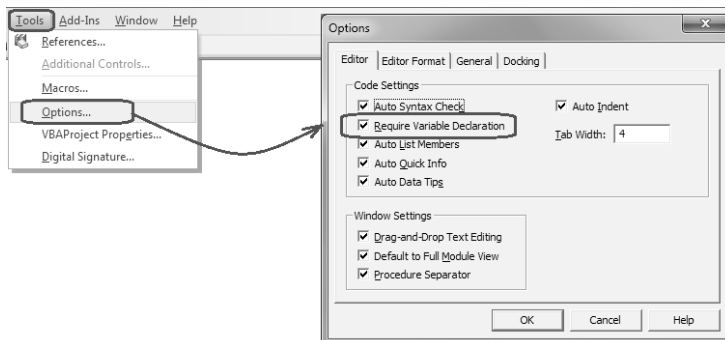
Jedyną radą, aby się uchronić od takich błędów jest wymuszenie stosowania jawnej deklaracji zmiennych (jak na przyzwoity język programowania przystało). W tym celu przejdźmy na początek modułu do części (**Declarations**) — wystarczy skorzystać z prawej listy rozwijanej u góry okna modułu:



I tam wpisujemy klauzulę:

```
Option Explicit 'wymuszenie jawnej deklaracji zmiennych
```

Możemy też spowodować, aby na przyszłość każdy nowy moduł zawierał już powyższą klauzulę jawnej deklaracji zmiennych w sekcji (**Declarations**). Wystarczy w tym celu wybrać w menu **Tools** pozycję **Options**, a tam na zakładce **Editor** zaznaczyć pole wyboru: **Require Variable Declaration** i kliknąć OK. Od tej pory każdy nowy moduł będzie wyposażony w klauzulę jawnej deklaracji zmiennych `Option Explicit`.



No dobrze. Ale sama klauzula to nie wszystko. Trzeba jeszcze wiedzieć jak się zmienne deklaruje. Otóż do deklaracji zmiennych służy polecenie **Dim** (jak dimension — wymiar). W najprostszej postaci będzie to:

```
Dim Zmienna [As Typ_zmiennej]
```

Nazwa zmiennej może się składać z małych (a-z) i dużych (A-Z) liter, cyfr (0-9) oraz podkreślnika (_). Nie zaleca się używania polskich liter. Nazwa zmiennej musi zaczynać się od litery. Ponadto nazwa zmiennej nie może być żadnym słowem kluczowym używanym przez VBA — czyli jesteśmy w lepszej sytuacji niż Anglosasi. Po prostu możemy używać naszych polskich wyrazów — na 100% nie będą to słowa kluczowe VBA.

Typy zmiennych

Typ zmiennej określa jakie dane będzie można pod tą zmienną zapamiętać. Dopuszczalne są w VBA następujące typy zmiennych:

Typ danych		Zakres:	Rozmiar:
Byte		Od 0 do 255	1 bajt
Boolean		True lub False	2 bajty
Integer	%	od -32 768 do 32 767	2 bajty
Long	&	od -2 147 483 648 do 2 147 483 647	4 bajty
Single	!	od $-3,402823 \cdot 10^{38}$ do $-1,401298 \cdot 10^{-45}$ (dla wartości ujemnych); od $1,401298 \cdot 10^{-45}$ do $3,402823 \cdot 10^{38}$ (dla wartości dodatnich)	4 bajty
Double	#	od $-1,797 693 134 862 32 \cdot 10^{308}$ do $-4,94065645841247 \cdot 10^{-324}$ (dla wartości ujemnych); od $4,94065645841247 \cdot 10^{-324}$ do $1,79769313486232 \cdot 10^{308}$ (dla wartości dodatnich)	8 bajtów,
Currency	@	od -922 337 203 685 477,5808 do 922 337 203 685 477,5807	8 bajtów,
Decimal		+/-79 228 162 514 264 337 593 543 950 335 (bez przecinka dziesiętnego); lub: +/-7,9228162514264337593543950 335 (z 28 miejscami po przecinku); najmniejszą liczbą niezerową jest: +/-0,00000000000000000000000000000001	14 bajtów

Date	od 1 stycznia 100r. do 31 grudnia 9999r.		8 bajtów
Object	dowolne odwołanie do wartości typu Object		4 bajty
String	\$	(zmiennej długości)	od 0 do około 2 miliardów znaków 10 bajtów + długość ciągu
		(ustalonej długości)	od 1 do około 65 400 znaków długość ciągu
Variant	(z liczbami)	(Każda wartość numeryczna w zakresie typu Double	16 bajtów
	(ze znakami)	taki sam zakres, co dla zmiennej typu String zmiennej długości	22 bajty + długość ciągu
Definiowany przez użytkownika (przy użyciu deklaracji Type)		zakres każdego elementu jest taki sam jak zakres typu danych tego elementu	wartość wymagana przez definiowane elementy

(Oprócz tego Excel posiada całą masę dodatkowych typów danych, które należą do kategorii typu **Object**, np.: *Chart, Comment, Dialog, Filter, Font, Range, Scenario, Sheet, Sort, Workbook, Worksheet, WorksheetFunction* etc...)

Domyślnym typem danych jest typ **Variant**. Czyli w przypadku niejawnej deklaracji zmiennych (jak to robiliśmy do tej pory) wszelkie pojawiające się w kodzie zmienne były traktowane, jako zmienne typu **Variant** (koszt co najmniej 16 bajtów). Również deklaracja zmiennej bez podania typu jest równoznaczna z podaniem typu **Variant**:

```
Dim Zmienna
Dim Zmienna As Variant
```

Możliwe jest stosowanie skróconej deklaracji typu zmiennej przez użycie znaków specjalnych — zaoszczędza nam to pisania zwrotu `As Typ_zmiennej`:

```
% - Integer    - np.:    Dim Zmienna%
& - Long       - np.:    Dim Zmienna&
! - Single    - np.:    Dim Zmienna!
# - Double    - np.:    Dim Zmienna#
@ - Currency - np.:    Dim Zmienna@
$ - String    - np.:    Dim Zmienna$
```

No dobrze, a gdzie należy deklarować zmienne, w której części kodu mamy to wpisywać...? Wszystko zależy od tego, jaki zasięg ma mieć deklaracja zmiennej:

Jeżeli zmienna ma być zmienną **lokalną** na potrzeby konkretnej procedury — to należy deklarować ją w obrębie kodu tej procedury (oczywiście deklaracja musi poprzedzać pierwsze użycie zmiennej w kodzie).

Jeżeli zmienna ma być zmienną **globalną** na potrzeby całego modułu (i wszystkich zapisanych tam procedur) to wówczas deklaracji zmiennej dokonuje się na początku modułu w sekcji (**Declarations**).

Funkcje konwersji typu

Czasami zachodzi konieczność zmiany typu danych i do tego celu VBA dysponuje specjalną grupą funkcji. Są to funkcje konwersji typu:

Funkcja **CBool(wyrażenie)** — zamienia wyrażenie na typ logiczny Boolean.

Funkcja **CByte(wyrażenie)** — zamienia wyrażenie na typ Byte.

Funkcja **CCurr(wyrażenie)** — zamienia wyrażenie na typ Currency.

Funkcja **CDate(wyrażenie)** — zamienia wyrażenie na typ Date.

Funkcja **Cdbl(wyrażenie)** — zamienia wyrażenie na typ Double.

Funkcja **CDec(wyrażenie)** — zamienia wyrażenie na typ Decimal.

Funkcja **CInt(wyrażenie)** — zamienia wyrażenie na typ Integer.

Funkcja **CLng(wyrażenie)** — zamienia wyrażenie na typ Long.

Funkcja **CSng(wyrażenie)** — zamienia wyrażenie na typ Single.

Funkcja **CStr(wyrażenie)** — zamienia wyrażenie na typ String.

Funkcja **CVar(wyrażenie)** — zamienia wyrażenie na typ Variant, (czyli Double — dla wyrażeń numerycznych, a String — dla wyrażeń tekstowych).

Funkcja **CVErr(numer_błędu)** - zwraca błąd o podanym numerze zdefiniowanym przez użytkownika.

Funkcja **Hex(liczba_całkowita)** — zwraca łańcuch znaków (String) odpowiadający szesnastkowej reprezentacji podanej liczby całkowitej:

```
Hex(245) = "F5"
```

Funkcja **Oct(liczba_całkowita)** — zwraca łańcuch znaków (String) odpowiadający ósemkowej reprezentacji podanej liczby całkowitej:

```
Oct(245) = "365"
```

Funkcja **Str(liczba)** - konwertująca podaną liczbę na łańcuch znaków (tekst):

```
Str(13) = "13"
```

Funkcja **Val("ciąg_znaków")** - konwertująca podany ciąg znaków na liczbę:

```
Val("13") = 13
```

No dobrze, starczy tej teorii - uzupełnijmy teraz kod naszego makra. Całość powinna wyglądać mniej więcej tak (pomijając komentarze):

Option Explicit

Sub Tabliczka()

Dim Wiersz As Integer

Dim Kolumna As Integer

For Wiersz = 1 To 10

For Kolumna = 1 To 10

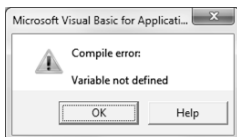
Cells(Wiersz, Kolumna).Value = Wiersz * **Kolumn**

Next Kolumna

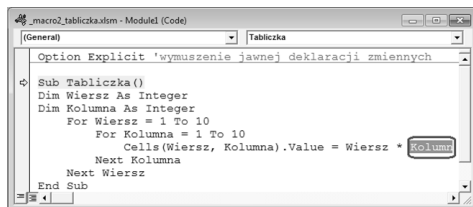
Next Wiersz


End Sub

Przejdźmy do Excela —  i spróbujmy uruchomić nasze makro — . Otrzymamy komunikat błędu:



A kliknięcie OK — wskazuje nam miejsce gdzie w kodzie pojawiła się niezadeklarowana zmienna:



Poprawiamy **Kolumn** na **Kolumna**, resetujemy makro —  i już po powrocie do Excela i ponownym uruchomieniu powinniśmy się cieszyć wynikami.

Oczywiście całość możemy jeszcze zapisać — **Zapisz jako** pamiętając, aby zmienić typ bieżącego pliku na **Skoroszyt programu Excel z obsługą makr (*.xslm)**:

Zapisz jako typ:  .

Usuwanie

Powiedzmy, że mamy arkusz, gdzie występują powtarzające się wpisy w kolumnie A. I powiedzmy, że chcemy napisać makro, które zostawi nam tylko pierwsze (unikalne) wystąpienia wyrazów — te zaznaczone kolorem czerwonym:

	A	B
	Kolumna	wiersz
1	kryterium	
11	Ela	wiersz 11
12	Ela	wiersz 12
13	Ela	wiersz 13
14	Kasia	wiersz 14
15	Zosia	wiersz 15
16	Zosia	wiersz 16
17	Zosia	wiersz 17
18	Zosia	wiersz 18
19	Zosia	wiersz 19
20	Ewa	wiersz 20
21	Ewa	wiersz 21

Oczywiście zamiast pisać takie makro wystarczyłoby na wstępie **Data** skorzystać z przycisku **Remove Duplicates**, ale dla nas napisanie tego makra stanie się pretekstem do poznania pętli `Do...Loop` oraz instrukcji warunkowej `If...Then...Else`.

Pętla Do ... Loop

Konstrukcja pętli `Do...Loop` zawiera warunek poprzedzony słowem **While** lub **Until**. Warunek ten zazwyczaj umieszczany jest zaraz za słowem **Do**, ale możliwe jest też użycie go za słowem **Loop**. Różnica jest taka, że w tym drugim przypadku jest on sprawdzany dopiero po pierwszym przejściu przez instrukcje znajdujące się między liniami kodu **Do** a **Loop**.

Jeżeli mamy wersję z `While` to instrukcje zawarte między **Do** i **Loop** będą powtarzane tak długo jak długo warunek za słowem **While** ma wartość **True**. W momencie, gdy ten warunek otrzyma wartość **False** — nastąpi skok do linii kodu za słowem **Loop**:

```
Do While warunek
    ...
    instrukcje
    ...
Loop
```

Jeżeli mamy wersję z `Until` to instrukcje zawarte między **Do** i **Loop** będą powtarzane tak długo jak długo warunek za słowem **Until** ma wartość **False**. W momencie, gdy ten warunek otrzyma wartość **True** — nastąpi skok do linii kodu za słowem **Loop**:

```
Do Until warunek
    ...
    instrukcje
    ...
Loop
```

Na zakończenie

Nasz "spacer" po makrach excelowych dobiega już końca... Przykłady można mnożyć w nieskończoność, a i tak nie wyczerpią one wszystkich możliwości programu. Zabawa z programowaniem w excelowym VBA wciąga, pozwala poczuć piękno i niesłychaną wszechstronność tego programu... Wiele pomysłów można spotkać w Internecie, a to, czego nie ma ani w książkach ani w sieci można odkryć metodą własnych prób. Zdobyte tą drogą doświadczenie stanowi najcenniejszy podręcznik, a Excel jest dobry prawie na wszystko...!