Programowanie
przed 12 wykładem

Andrzej Giniewicz

14.06.2024

Dziś zajmiemy się makrami w języku Visual Basic for Applications.

1 Przygotowanie Excela

Zanim przystąpimy do pracy, trzeba odpowiednio przygotować Excela. Po pierwsze, musimy się upewnić, że nie korzystamy z wersji 365 pakietu Office, ponieważ wtedy nie będziemy mieli możliwości tworzenia makr. Możemy to zrobić wchodząc w menu „Plik" i klikając „Konto". Tam możemy sprawdzić, jaka wersja pakietu jest zainstalowana. Jeśli korzystamy z Excela w wersji pełnej, czyli 2021 lub starszy, ale nie 365, możemy wejść w menu „Plik", następnie kliknąć „Opcje". W zakładce „Dostosowywanie Wstążki" po prawej stronie zaznaczamy pole wyboru obok nazwy „Deweloper". Spowoduje to, że w Excelu pojawi się nowa wstążka, w której znajdował się będzie edytor makr, narzędzia do ich nagrywania oraz uruchamiania.

2 Dialekty języka BASIC

BASIC jest skrótem od angielskiego *Beginners' All-purpose Symbolic Instruction Code*, języka zaprojektowanego dla początkujących użytkowników komputerów, który powstał w 1963 roku w Dartmouth College, stąd pierwszy dialekt języka BASIC nazywany jest Dartmouth BASIC. Od tego czasu powstały dziesiątki wariantów tego języka, nieznacznie różniące się pod kątem składni. Niekiedy ze względu na inne ograniczenia lub wymagania sprzętu, niekiedy ze względu na chęć wprowadzenia nowych funkcjonalności. Niekiedy BASIC pełnił wręcz rolę systemu operacyjnego, ponieważ komputer uruchamiał się do interpretera języka BASIC i z jego poziomu ładowało się inne programy. Z tego powodu był mocno stowarzyszony z rodzajem sprzętu, na którym działał. Język Microsoft BASIC był dialektem języka BASIC, który powstał w 1975 roku dla komputerów Altair 8800, pierwotnie nazywany Altair BASIC. Co ciekawe, był to pierwszy produkt firmy Microsoft, praktycznie ten, dla którego powstała, wtedy jeszcze z nazwą Micro-Soft.

Komputer Altair 8800, fotografia autorstwa Timothy'ego Colegrove'a, źródło:
https://en.wikipedia.org/wiki/Altair_8800#/media/File:
Altair_8800_and_Model_33_ASR_Teletype_.jpg.
Wprawne oko zauważy przełącznik pomiędzy dwoma stanami, „magic" oraz „more magic".
Jest to referencja do historii przełącznika z takimi podpisami w laboratorium sztucznej
inteligencji na uczelni MIT — http://www.catb.org/jargon/html/magic-story.html.

Od tamtego czasu sam Microsoft wydał warianty języka BASIC na wiele komputerów. W różnych wariantach język ten funkcjonował do końca lat 80-tych XX wieku. W 1991 roku pojawił się język Microsoft Visual Basic, na bazie którego powstały Microsoft Visual Basic Scripting Edition (1996 rok) oraz Microsoft Visual Basic for Applications (1993 rok). Ten pierwszy był językiem skryptowym dla systemu operacyjnego oraz przeglądarki Internet Explorer, który ma zostać usunięty z systemu Windows w 2027 roku[1]. Język Visual Basic for Applications funkcjonuje do dzisiaj jako część pakietu Microsoft Office. Warto zwrócić uwagę, że sam Visual Basic, na którym bazują dwa wspomniane języki do tworzenia skryptów, nie rozwija się od 1998 roku i został zastąpiony przez nowy język — Microsoft Visual Basic .NET. Język Visual Basic .NET rozwija się do dzisiaj, jednakże ma nieco inną składnię, niż język Visual Basic i bazujący na nim Visual Basic for Applications. Szukając przykładów w Internecie, trzeba zwrócić szczególną uwagę na to, dla jakiego dialektu języka powstały.

Oprócz tego, do współcześnie rozwijanych wersji, należy na przykład FreeBASIC[2], Gambas[3] oraz Xojo[4], które są językami ogólnego przeznaczenia, oraz Libre Office BASIC[5], który stara się być kompatybilny z Visual Basic for Applications. Co ciekawe LibreOffice posiada więcej języków programowania makr, na przykład Pythona[6].

[1] Patrz https://techcommunity.microsoft.com/t5/windows-it-pro-blog/vbscript-deprecation-timelines-and-next-steps/ba-p/4148301.

[2] Patrz https://www.freebasic.net/.

[3] Patrz https://gambas.sourceforge.net/en/main.html.

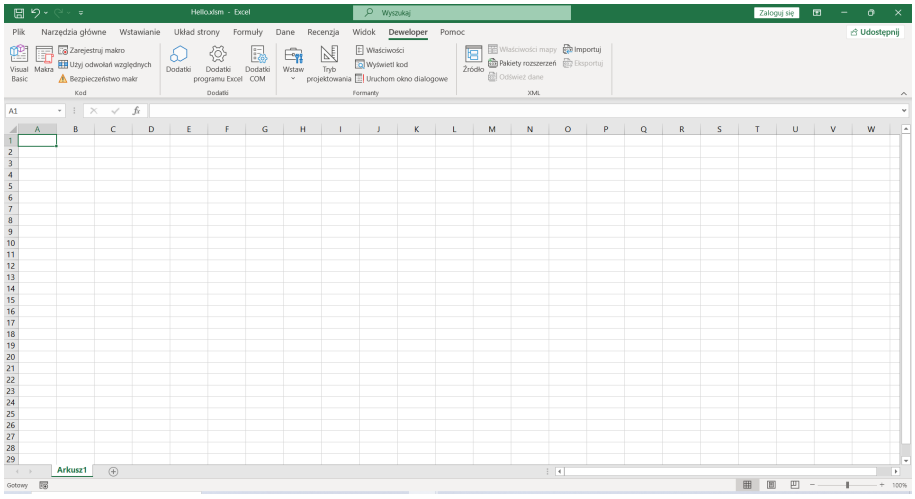
[4] Patrz https://www.xojo.com/.

[5] Patrz https://help.libreoffice.org/latest/pl/text/sbasic/shared/main0601.html.

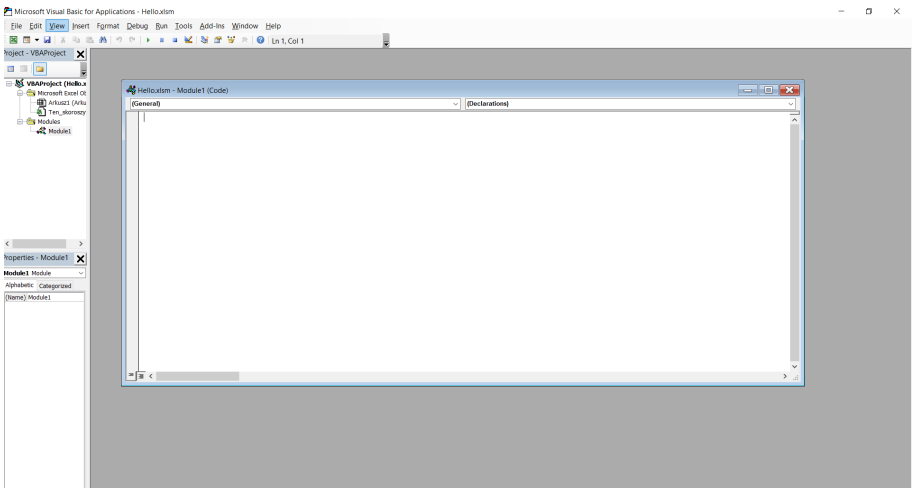
[6] Patrz https://help.libreoffice.org/latest/pl/text/sbasic/python/main0000.html.

3 Droga do pierwszego makra

Zaczynamy od włączenie Excela, utworzenia pustego dokumentu i zapisania go jako plik z rozszerzeniem `xlsm`, co oznacza „Skoroszyt programu Excel z obsługą makr”. Podczas otwierania plików `xlsm` makra są domyślnie wyłączone z powodów bezpieczeństwa, na żółtym pasku nad arkuszem pojawi się informacja o wyłączeniu makr oraz przycisk służący do włączenia obsługi makr. Utwórz plik `Hello.xlsm` i wejdź do zakładki „Developer”.



Następnie uruchamiamy edytor Visual Basic. Klikamy prawym na liście projektów po lewej na nazwie odpowiadającej plikowi `Hello.xlsm` i wybieramy „Insert” oraz „Module”. Pojawi się nowy moduł, w którym możemy pisać kod. Jeśli listy projektów nie widać, można ją włączyć lub wyłączyć w menu „View” wybierając „Project Explorer”.



Wpisujemy następujący fragment kodu, chwilowo nie zastanawiając się, co oznaczają poszczególne komendy.

```
Sub HelloWorld()
  Range(”A1”).Value = ”Hello World”
End Sub
```

Następnie zapisujemy plik, w Arkuszu wchodzimy w menu „Makra” i powinniśmy widzieć makro o nazwie „HelloWorld”. Po kliknięciu w przycisk „Uruchom”, w aktywnym arkuszu w lewym górnym rogu pojawi się napis „Hello World”. W menu do uruchamiania możemy też dodać skróty klawiszowe do poszczególnych makr, co możemy zrobić w menu „Opcje...”. Dzięki temu można przygotować skróty do poszczególnych funkcji.

Inną opcją jest wejście w menu „Wstaw” i wybranie przycisku podpisanego „Przycisk formularza”. W pojawiającym się edytorze możemy wybrać dowolne z napisanych makr, w naszym przypadku „HelloWorld”.

Jeśli nie udało się napisać makra, pobierz plik „Hello.xlsm”[7] i sprawdź, czy możesz uruchomić makro. Jeśli nie, upewnij się, że podczas otwierania pliku wyraziłeś zgodę na uruchamianie makr oraz nie korzystasz z Excela 365. Makro z przykładu powinno działać również w LibreOffice Calc, ale konieczne jest włączenie obsługi makr w opcjach bezpieczeństwa. Podczas otwierania pliku użytkownik zostanie pokierowany w odpowiednie miejsce.

4 Podstawy języka VBA

Przejrzyjmy składnię języka VBA. Znając już inne języki, możemy sobie pozwolić na szybkie przejście przez podstawowe elementy, które mają odpowiedniki w innych językach.

4.1 Komentarze

Komentarze możemy wpisywać na dwa sposoby, od apostrofu lub komendą `REM`.

```
' To komentarz
REM to też komentarz
```

Komentarz z apostrofem może pojawić się również na końcu innej linii z kodem, `REM` powinien być używany tylko na początku linii.

4.2 Długie linie

Jeśli linia jest długa, można ją podzielić symbolem _ i wykonując wcięcie w kolejnej linii.

```
Bardzo długa _
   linia
```

4.3 Wymuszenie sprawdzania definicji zmiennych

Na początku kodu, jeśli umieścimy linię

```
Option Explicit
```

Visual Basic for Applications będzie sprawdzał, czy nie użyliśmy niezdefiniowanych zmiennych. Dzięki temu unikniemy błędów, jest to więc zalecana opcja.

[7] Patrz `https://im.pwr.edu.pl/~giniew/Hello.xlsm`.

4.4 Obsługa błędów

Domyślnie po pojawieniu się błędu, program zakończy działanie zadanie. Istnieje możliwość, żeby powiedzieć VBA, co ma robić, gdy nastąpi błąd.

```
On Error GoTo n
```

gdzie *n* to numer linii większy od 0. Jeśli wywołamy tę komendę, po pojawieniu się błędu, program przejdzie do wykonywania kodu znajdującego się w linii *n* kodu programu. O wiele częściej używaną opcją, jest

```
On Error Resume Next
```

powodujące zignorowanie błędu i przejście do następnej linii. Wykorzystujemy ten format obsługi błędów, na przykład, gdy chcemy skasować arkusz o zadanej nazwie, a nie jesteśmy pewni, czy taki arkusz istnieje. Skasowanie nieistniejącego arkusza powoduje błąd. Jeśli włączymy ignorowanie błędów na czas kasowania, będziemy mogli się upewnić, że arkusza o zadanej nazwie na pewno nie ma, bez instrukcji warunkowej.

Specjalną instrukcją

```
On Error GoTo 0
```

wyłączamy zdefiniowaną wcześniej obsługę błędów, przywracając zachowanie programu do domyślnej.

4.5 Operatory

operacja	**operator**
dodawanie	`+`
odejmowanie i negacja	`-`
mnożenie	`*`
dzielenie	`/`
dzielenie całkowite	`\`
reszta z dzielenia	`Mod`
potęga	`^`
porównania	`=, <>, <, >, <=, >=`
scalanie napisów	`&`
iloczyn logiczny	`And`
suma logiczna	`Or`
logiczna różnica symetryczna	`Xor`
negacja logiczna	`Not`
równoważność logiczna	`Eqv`
implikacja logiczna	`Imp`

Szczególną uwagę należy zwrócić na dodawanie, gdy choć jeden z elementów nie jest liczbą. Dokładny opis zachowania operatora znajduje się w dokumentacji[8].

[8] Patrz `https://learn.microsoft.com/en-us/office/vba/Language/Reference/User-Interface-Help/plus-operator#remarks`.

4.6 Zmienne

VBA posiada typy proste, takie jak `Boolean`, `Integer`, `Long`, `Double`, `Date` lub `String`. Istnieje typ `Variant`, który może przechować dowolny z powyższych typów. Pozostałe dane, są typu `Object`.

Zmienne należy zadeklarować instrukcją `Dim`, piszemy je w osobnych liniach lub oddzielając przecinkami

```
Dim nazwa As typ
Dim nazwa1 As typ1, nazwa2 As typ2
```

Podstawienie wartości następuje za pomocą równości dla typów prostych

```
nazwa = wartość
```

natomiast za pomocą komendy `Set` dla obiektów. Specjalna wartość `Nothing` oznacza brak wartości dla obiektu.

```
Set referencja = obiekt
Set referencja = Nothing
```

Tablice możemy definiować, podając ich rozmiar w nawiasie. Indeksowanie następuje za pomocą nawiasu okrągłego. Tablice domyślnie indeksujemy od jedynki.

```
Dim tablica(n) As typ
tablica(1) = wartość1
tablica(2) = wartość2

tablica(n) = wartośćn
```

Istnieje też możliwość ustalania dynamicznego rozmiaru, jeśli tablica zostanie zwrócona przez inną funkcję, na przykład

```
Dim tablica() As String
tablica = Split("1,2,3", ",")
```

Wymiar tablicy możemy zmienić komendą `ReDim` lub `ReDim Preserve`. Pierwsza z opcji podczas zmiany rozmiaru tablicy czyści ją, druga zachowuje wartości.

```
ReDim Preserve tablica(m)
```

4.7 Instrukcje warunkowe

Mamy dostępną instrukcję `If`. Obsługuje ona blok `Else` oraz `ElseIf`, choć są one opcjonalne.

```
If warunek1 Then
  ' instrukcje, gdy warunek1 spełniony
ElseIf warunek2 Then
```

```
  ' instrukcje, gdy warunek1 nie jest spełniony ale warunek2 jest spełniony
Else
  ' instrukcje, gdy warunek1 ani warunek2 nie są spełnione
End If
```

Wróćmy uwagę, że `Then` po warunku oraz `End If` na koniec instrukcji warunkowej są konieczne.

Istnieje również możliwość użycia instrukcji `Switch Case`. Więcej na jej temat można sprawdzić w dokumentacji[9].

4.8 Pętle

Podstawowa forma pętli to

```
For licznik = początek To koniec
   ' instrukcje
Next licznik
```

Istnieje możliwość użycia instrukcji `Continue For` oraz `Exit For`, jeśli chcemy wcześniej przerwać wykonywanie kodu. Istnieje też możliwość wykonywania pętli co pewien krok według zasięgu.

```
For licznik = początek To koniec Step krok
   ' instrukcje
Next licznik
```

VBA posiada też pętle `While`

```
Do While warunek
   ' instrukcje
Loop
```

działające tak długo, jak warunek jest spełniony, przy czym najpierw sprawdzany jest warunek, a potem wykonywany jest kod. Istnieje też wariant, który najpierw wykonuje kod, a potem sprawdza warunek, gwarantując tym samym, że kod wykona się minimum jeden raz.

```
Do
   ' instrukcje
Loop While warunek
```

Jeśli chcemy wcześniej przerwać pętle, używamy `Continue Do` lub `End Do`.

Kolejny wariant pętli to pętle warunkowe `Until`, działające do momentu aż po raz pierwszy będzie spełniony warunek — można o nich myśleć, jak o `While` z zaprzeczeniem.

[9] Patrz https://learn.microsoft.com/en-us/office/vba/Language/Reference/User-Interface-Help/select-case-statement.

```
Do Until warunek
   ' instrukcje
Loop

Do
   ' instrukcje
Loop Until warunek
```

Ostatnią formą pętli jest pętla `For Each` działająca dla obiektów mających więcej niż jedną wartość, na przykład dla tablic.

```
For Each element In grupa
   ' instrukcje
Next element
```

Zwróćmy uwagę, że w pętle zaczynające się od `For` kończą się instrukcją `Next`, w której musimy powtórzyć nazwę indeksu wykorzystanego w pętli.

4.9 Funkcje i procedury

Procedury to fragmenty kodu, które możemy uruchomić, ale nie zwracają one wartości. Do ich zdefiniowania używamy instrukcji `Sub`, podajemy nazwę oraz parametry w nawiasie.

```
Sub nazwa(arg1 As typ1, Optional nazwa As typ = wartość)
   ' instrukcje
End Sub
```

Przykład powyżej pokazuje procedurę z jednym parametrem wymaganym oraz jednym opcjonalnym o domyślnej wartości. Uruchomienie procedury odbywa się za pomocą podania nazwy z nawiasami i odpowiednimi wartościami argumentów

```
nazwa(wartość1)
```

Jeśli chcemy, aby fragment kodu zwracał wartość, zamiast `Sub` używamy `Function`. W Visual Basic for Applications nie ma instrukcji `Return`, aby zwrócić wartość, podstawiamy ją pod nazwę funkcji.

```
Function nazwa(arg1 As typ1) As typ_wartości_zwracanej
   ' instrukcje
   nazwa = wartość_zwracana
End Function
```

Domyślnie argumenty przekazywane są jako referencja, czyli jeśli zmienimy wartość argumentu wewnątrz funkcji lub procedury, zmieni się również poza nią. Jeśli chcemy, aby argumenty były kopiowane podczas wywołania funkcji i aby zmiana ich wartości wewnątrz nie wpływała na funkcję wywołującą, dodajemy instrukcję `ByVal` przed nazwą zmiennej.

```
Sub nazwa(ByVal arg1 As typ1, Optional ByVal nazwa As typ = wartość)
   ' instrukcje
End Sub
```

Funkcje zdefiniowane za pomocą słowa kluczowego `Function`, możemy wywoływać z innych funkcji, procedur oraz formuł.

4.10 Funkcje wbudowane

VBA posiada wiele zdefiniowanych funkcji. Ich listę można sprawdzić w dokumentacji[10]. Sprawdź definicje funkcji `InStr`, `Mid`, `Left`, `StrConv`, `Trim` oraz `DateSerial`. Ich znajomość przyda się w dalszej części wykładu.

4.11 Obiekty, atrybuty i metody

Visual Basic for Applications obsługuje obiekty. W obiektach możemy dostać się do ich argumentów oraz metod. Używamy do tego kropki.

```
x = obiekt.pole
obiekt.pole = x
obiekt.metoda_bezargumentowa()
obiekt.metoda(...)
```

Jeśli nie chcemy powtarzać obiektu w wielu liniach, możemy użyć instrukcji `With`, która automatycznie wstawia obiekt przed referencję zaczynającą się od kropki.

```
With obiekt
  x = .pole
  .pole = x
  .metoda_bezargumentowa()
  .metoda(...)
End With
```

Instrukcji `With` używamy niekiedy, gdy obiekt, na rzecz którego wywołujemy kilka metod, ma długą nazwę i nie chcemy tworzyć zmiennej tymczasowej i jej powtarzać.

4.12 Obiekty Excela

Excel udostępnia do VBA zestaw obiektów, za pomocą których możemy wchodzić w interakcje z aplikacją. Najważniejszym obiektem jest `Application`[11]. Mamy w nim trzy atrybuty, z których będziemy korzystać:

Application.DisplayAlerts domyślnie `True`, odpowiada za to, czy pojawiać się będą komunikaty ostrzegawcze,

[10] Patrz `https://learn.microsoft.com/en-us/office/vba/language/reference/functions-visual-basic-for-applications`.

[11] Patrz `https://learn.microsoft.com/en-us/office/vba/api/excel.application(object)`.

Application.ScreenUpdating domyślnie `True`, odpowiada za odświeżanie interfejsu graficznego, niekiedy możemy chcieć wyłączyć aktualizacje na czas działania makra, aby nie wyświetlać niekompletnych danych,

Application.ActiveWorkbook zawiera aktywny obecnie skoroszyt, który również jest obiektem.

Innym obiektem pozwalającym na zarządzanie skoroszytami jest `Workbooks`[12]. Posiada on między innymi dwie przydatne metody:

Workbooks.Add tworzącą nowy skoroszyt,

Workbooks.Open otwierającą skoroszyt z pliku.

W obiekcie reprezentującym obecny skoroszyt znajduje się obiekt `Worksheets`, pozwalający na manipulowanie arkuszami skoroszytu:

.Worksheets("nazwa") wybiera arkusz o zadanej nazwie,

.Worksheets(1) wybiera arkusz o zadanym numerze,

.Worksheets.Add() tworzy nowy arkusz.

Często od razu tworząc arkusz, nadajemy mu nazwę. Możemy to zrobić w jednej linii, ponieważ `Add` zwraca arkusz. Wobec tego często zobaczymy fragmenty kodu typu

```
Application.ActiveWorkbook.Worksheets.Add().Name = "Hello"
```

Kod ten spowoduje na dodanie arkusza o nazwie „Hello" w obecnym skoroszycie.

Na wybranym arkuszu, na przykład poprzez `Application.ActiveWorkbook.Worksheets(1)` możemy wykonać między innymi następujące operacje:

.Activate aktywuje arkusz, jak kliknięcie,

.Delete kasuje arkusz, jeśli nie istnieje, zwraca wyjątek,

.Name zawiera nazwę arkusza.

Na arkuszu możemy również wykonywać zaznaczenia:

.Range("...") wybiera zakres, np.: "`A1:C3`",

.Cells wybiera wszystkie komórki,

.Cells(i, j) wybiera komórkę o współrzędnych (i, j),

.Rows(i) wybiera i-ty wiersz,

.Columns(j) wybiera j-tą kolumnę,

[12] Patrz `https://learn.microsoft.com/en-us/office/vba/api/excel.workbooks`.

.Rows("1:5") wybiera zakres wierszy,

.Columns("A:D") wybiera zakres kolumn,

.Range(Cels(i,j), Cels(k,l)) wybiera zakres z wykorzystaniem współrzędnych.

Dysponując jakimś zakresem, możemy na przykład:

.Clear wyczyścić zawartość i formatowanie,

.CurrentRegion poszerzyć zaznaczenie do największego obszaru prostokątnego bez wolnych kolumn i wierszy,

.Offset(i,j) przesunąć wybrany obszar o wektor,

.Value dostać się do wartości w komórce, również ją zmienić,

.Formula dostać się do formuły w komórce, również ją zmienić,

.Rows dostać listę wierszy,

.Columns dostać listę kolumn.

Wyciąganie listy wierszy i kolumn często idzie w parze z użyciem atrybutu `.Count` zwracającego liczbę elementów tablicy, zatem odpowiednio

.Rows.Count oznacza liczbę wierszy zaznaczenia, natomiast

.Columns.Count oznacza liczbę kolumn zaznaczenia.

5 Analiza makra

Pobierz plik `Dane_vba.xlsm`[13]. Uruchom makro `clear_data` i przeanalizuj jego kod. Wszystkie potrzebne elementy powinny już być Ci znane lub możesz je łatwo uzupełnić z dokumentacji.

[13] Patrz `https://im.pwr.edu.pl/~giniew/Dane_vba.xlsm`.