Martin Král

Pro verze
2007
2010

# Excel VBA

## Výukový kurz

Od základů po programování procedur a vlastních tříd

Ovládací prvky, práce s daty na listu, formuláře

Shrnutí výkladu, otázky a odpovědi

**CD obsahuje**

Kódy všech příkladů z knihy

Užitečné soubory od předních vývojářů

CPRESS

Martin Král

# Excel VBA
## Výukový kurz

Computer Press
Brno
2012

# Excel VBA
## Výukový kurz

**Martin Král**

**Obálka:** Martin Sodomka
**Odpovědný redaktor:** Libor Pácl
**Technický redaktor:** Jiří Matoušek

Objednávky knih:
http://knihy.cpress.cz
www.albatrosmedia.cz
eshop@albatrosmedia.cz
bezplatná linka 800 555 513
ISBN 978-80-251-2358-4

Vydalo nakladatelství Computer Press v Brně roku 2012 ve společnosti Albatros Media a. s. se sídlem Na Pankráci 30, Praha 4. Číslo publikace 16 061.
© Albatros Media a. s. Všechna práva vyhrazena. Žádná část této publikace nesmí být kopírována a rozmnožována za účelem rozšiřování v jakékoli formě či jakýmkoli způsobem bez písemného souhlasu vydavatele.

Dotisk 1. vydání.

ALBATROS MEDIA a.s.

# Obsah

# ÚVOD

## Komu je kniha určena

Tato kniha je především o *času*, kterého nikdo z nás nemá nazbyt. Možná jste manažer, který si chce firemní data pravidelně analyzovat sám, aby si byl jistý, že mu údaje předkládané oddělením controllingu nezkreslují skutečnost. Možná jste pracovník oddělení controllingu a každý měsíc do pátého pracovního dne musíte zkonsolidovat tuny dat a zaplnit standardní tabulku definovanou kdesi v mateřské firmě kýmsi, kdo nemá ponětí o databázovém uspořádání dat. Možná jste pracovník IT oddělení, expert na databáze a všechny možné dialekty SQL, ale váš šéf jednoduše vyžaduje reporty v Excelu. Jako databázový expert a zkušený programátor v SQL a ASP.NET považujete něco takového pod vaši úroveň, nicméně šéf je šéf a chce mít svou zprávu co nejdřív.

Pro vás všechny je tato kniha. Naučí vás, jak vám VBA může ušetřit čas, a rovněž vám ukáže, jak ušetřit čas při samotném psaní kódu. Když hovoříme o kódu, rozumíme tím program, v tomto případě ve VBA. V praxi je možno se často setkat s výrazem „makro". Toto slovo v této knize nebudu pro program používat, protože se jedná o slovo matoucí. Slovo „Makro" v Excelu pochází z dob, kdy Excel používal takzvaný XLM jazyk (neplést s XML). Velmi neintuitivní syntaxe byla postupně vytlačena nástupem VBA. Makro příkazy XML mají však své výhody. Ukazuje se, že díky tomu, že jsou přímo součástí jádra Excelu, vykonávají příkazy mnohem rychleji než jejich ekvivalenty v VBA. Samozřejmě, většinou mluvíme v milisekundách, ale co když příkazem zpracováváme jednotlivě 20 000 řádek? A co když chceme dávkově zpracovat 100 takových souborů čekajících na nás ve složce „KeZpracovani"?

Asi vás příliš nepřekvapím, když řeknu, že efektivní kód znamená obvykle kratší kód, čímž ušetříte i čas na samotné programování. Proto, až se naučíte psát efektivní programy, zabijete dvě mouchy jednou ranou. Ušetříte čas jak při vývoji, tak později díky využívání vaší báječné aplikace.

Kniha není primárně určena pro začátečníky v programování, neboť poměrně do hloubky probírá základy VBA a nejpoužívanější části objektového modelu Excelu. V knize je velké množství konkrétních ukázek a řešení běžných problémů, se kterými jsem se setkal ve vlastní praxi a které rovněž vedou k častým dotazům na internetových fórech. Vysvětlení, jak kód pracuje, se spíše týká použitého algoritmu a pastí, než toho, jak ta která klauzule, funkce, metoda nebo vlastnost pracuje. Budu se snažit vysvětlit rovněž netypické a netradiční použití jinak standardních metod, které jsem ve velké míře odkoukal od špičkových mágů ze světa Excelu a v menší míře na ně přišel sám.

Přestože je kniha určena především pro pokročilé uživatele Excelu 2007, většina kódu prošla testem na Excel 2007 CZ, Excel 2007 ENG, Excel 2003 ENG a většina i na Excel 2000 ENG. Operační systémy byly Windows Vista i Windows XP SP2 v anglické verzi s českým místním nastavením. Při extrémnějších pokusech s mezinárodním prostředím jsem používal Windows XP RU a rovněž

Excel v ruské lokalizaci. Pokud bude některý kód nekompatibilní s verzemi nižšími než Excel 2007, výslovně na to upozorním.

Některým oblastem jsem úmyslně věnoval menší místo, než by možná bylo v knize o VBA obvyklé. Jedná se především o uživatelské formuláře. Přestože jim v této knize věnuji celou jednu lekci, ve vhodně navržené aplikaci jsou zřídkakdy potřeba, protože list v Excelu je vlastně výkonný a flexibilní formulář, za který můžeme schovat libovolný kód. Naopak, podle mé zkušenosti neustále vyskakující okna a formuláře vyžadující reakci uživatele spíše obtěžují. Objektový model Excelu je velmi rozvětvený a s každou novou verzí se dále rozšiřuje. V současné době se rozšiřování týká zejména propojení na vnější databáze, těsnější integrace XML a podobně. Tradiční funkce Excelu se od verze 2000 příliš nemění a výraznějšími změnami prošel zejména objektový model grafiky a jejího formátování. Podstatně rozšířen ve verzi 2007 byl i podmíněný formát.

Další „opomenutou" oblastí je ovládání vestavěného menu Excelu. Za prvé, Excel 2007 menu zásadně změnil a dosavadní objektový model pro něj neplatí, čímž se vaše aplikace stává nekompatibilní směrem dolů. A za druhé, zvládnutí úplného programování nového menu předpokládá znalost jazyka XML a to už je opravdu mimo rámec této knihy.

Obecně se tato kniha drží spíše hesla „Dej člověku rybu, a nakrmíš ho na jeden den. Nauč jej rybařit, a nakrmíš jej na celý život." Je prakticky nemožné a hlavně zhola zbytečné snažit se popisovat práci se všemi objekty Excelu do detailů. Autor samozřejmě taky nezná nazpaměť všechny objekty a už vůbec ne jejich metody a vlastnosti. Vtip je v tom, umět méně používané prvky nalézt a správně a hlavně efektivně je použít ve svém kódu. A i o tom je tato kniha, zejména její šestá lekce.

# Co je to VBA a jak se liší od VB

Visual Basic (VB) a Visual Basic for Applications (VBA) mají společný základ. Lze říci, že jádro jazyka Visual Basic 6.0 je součástí instalace Microsoft Office (od verze 9.0, neboli 2000) a komunikace s ním probíhá přes moduly kódu a uživatelské formuláře, které se od formulářů Visual Basic liší svou určitou omezeností, avšak pro většinu aplikací postačí.

Pokud mluvíme o VBA, nemluvíme o Excel VBA, Word VBA, nebo Access VBA. Syntaxe jazyka je pro všechny aplikace stejná, liší se pouze *objektový model* aplikace, se kterou pracujeme. Pokud se naučíte syntaxi jazyka VBA a programování aplikace Excel, je relativně jednoduché nabyté znalosti použít na jiné aplikace, které VBA rovněž používají. Jediné, co se musíte naučit, je objektový model aplikace, kterou chcete automatizovat.

Podívejte se na následující ukázky jednoduchého kódu, které vykonávají velmi podobnou činnost, totiž zapisují sekvenci čísel 1 až 10 do nové tabulky Excelu, do nového dokumentu Wordu a konečně do existující tabulky v Accessu.

Microsoft Excel:

```
'// Kód zapíše čísla 1 až 10 do oblasti A1:A10
'// listu "Sheet1" nového sešitu v Excelu

Const MAX As Long = 10
Dim i As Long
```

```
With Workbooks.Add
    For i = 1 To MAX
        Sheets(1).Cells(i, "A") = i
    Next i
End With
```

Microsoft Word:

```
'// Kód zapíše čísla 1 až 10, každé do nového odstavce,
'// do nového dokumentu ve Wordu

Const MAX As Long = 10
Dim i As Long

Documents.Add
With Application
    For i = 1 To MAX
        .Selection.TypeText Text:=CStr(i)
        .Selection.TypeParagraph
    Next i
End With
```

Microsoft Access:

```
'// Kód přidá nové záznamy do tabulky "tblMyTable" v databázi Access
'// a zapíše čísla 1 az 10 do pole "MyField"

Const MAX As Long = 10
Dim i As Long
Dim rs As DAO.Recordset
Set rs = CurrentDb.OpenRecordset("tblMyTable")

With rs
    For i = 1 To MAX
        .AddNew
        .Fields("MyField").Value = i
        .Update
    Next i
End With

rs.Close
Set rs = Nothing
```

Objektový model aplikace se skládá z objektů, jejich metod a vlastností. S objekty manipulujeme metodami a jejich charakteristiky stanovujeme a zjišťujeme přes jejich vlastnosti. V běžném jazyce řekneme:

„Natankujeme do automobilu naftu, poté ho nastartujeme, a jestli motor běží, rozjedeme se." V pseudokódu by tento proces vypadal takto:

```
Automobil.Natankovat Palivo:=Nafta
Automobil.Nastartovat
If Automobil.Motor.Bezi Then
```

```
....Automobil.RozjetSe
End If
```

Podobně, pokud chcete „nastartovat“ aplikaci Excel například z aplikace Word, spusťte Word, stiskněte Alt+F11, vložte nový modul, napište tento kód, umístěte kurzor kamkoli dovnitř kódu a stiskněte F5.

```
Sub NastartovatExcel()
Dim oAppXL As Object
Set oAppXL = CreateObject("Excel.Application")
If Not oAppXL Is Nothing Then
    oAppXL.Workbooks.Add
    oAppXL.ActiveWorkbook.Sheets(1).Range("A1") = "MS Word"
    oAppXL.Visible = True
End If
End Sub
```

Nevadí, jestli nevíte, jak tento kód pracuje. Vše se dozvíte v dalších částech knihy. Pro tento moment je důležité, abyste si uvědomili, že je jen jeden jazyk VBA, který vychází z jazyka Visual Basic. Co se liší, jsou objekty dané aplikace, jejich metody a vlastnosti.

# VBA pomáhá šetřit čas

Toto není vymyšlený příběh.

Představte si, že jste byli jmenováni manažerem v nové firmě, která provozuje desítky obchodů, obsluhuje desetitisíce zákazníků denně a prodává kolem deseti tisíc položek. Ve firmě existuje několik informačních systémů, včetně vlastní databáze nákupů a prodejů na bázi FoxPro a účetního systému na bázi SAP. Protože jsou výstupy z těchto systémů dosti uživatelsky nepřívětivé, každé oddělení si vytvořilo své standardní tabulky v Excelu, které jsou ovšem zčásti zaplňovány a upravovány ručně, a tudíž jsou náchylné k chybovosti. Na existující zprávy se nemůžete spolehnout, protože nevíte, jaká data zahrnují a jestli jsou informace úplné. Souhrnné přehledy integrující výsledky jednotlivých oddělení neexistují. Navíc neexistuje jednotná databáze obchodů, popisující jejich vybavení, příslušnost do regionů a kvalitativní kategorizaci. Databáze prodejů neumožňuje přímo generovat časový vývoj, srovnání s plánem a podobně. Výsledkem je, že nedokážete odpovědět na základní otázky týkající se silných a slabých stránek firmy a jejího možného ohrožení.

Jako profesionál víte přesně, jaké informace potřebujete, abyste dostal firmu rychle pod kontrolu. Vznesete proto dotaz na IT oddělení, jaké budou náklady na vytvoření standardních přehledů a sestav podle vašeho přání. Odpovědí budou pravděpodobně finanční náklady minimálně ve stovkách tisíc a časové náklady v řádu měsíců. Mimoto rozpočet neumožňuje zahrnout takové náklady do letošního roku, a tudíž je možné s pracemi začít nejdříve po schválení rozpočtu na rok příští a i schválení takového projektu je otázkou.

Vy ovšem nemáte čas čekat, protože akcionáři od vás očekávají rychlou analýzu situace a návrhy opatření. Navíc očekávají pravidelnou měsíční zprávu o situaci ohledně prodejů. A to je chvíle, kdy vám znalost programování aplikací Microsoft Office ve VBA může doslova zachránit kůži.

První den si ujasníte, která data budete potřebovat, abyste obdrželi potřebné souhrnné zprávy. Ještě tentýž den navrhnete několik základních tabulek v aplikaci Microsoft Access, které potřebujete pra-

videlně plnit daty, a definujete potřebné relace mezi nimi kvůli integritě dat (toto je velká přednost Accessu oproti Excelu). Uživatelské rozhraní zatím programovat nepotřebujete, protože databáze v Accessu bude sloužit pouze jako sklad dat.

Druhý den si necháte od IT oddělení předvést, jak a v jaké formě můžete data z jednotlivých systémů dostat. Je důležité, abyste dostali data na co nejnižší úrovni, protože nemůžete vědět, jestli agregační zprávy zahrnují všechny potřebné informace. Pokud bude mít předpokládaný objem dat do blízké budoucnosti několik set tisíc až několik málo milionů řádků, nevadí, protože SQL dotazy si s tímto objemem dat relativně hravě poradí. Omezení databáze Access (2GB) by se vás zdaleka týkat nemělo, navíc je možno data rozdělit a databáze propojit.

Když budete mít štěstí (autor ho většinou neměl), IT oddělení vám nainstaluje potřebné databázové ovladače a dá práva čtení k tabulkám ve firemní databázi. Pokud budete mít o něco méně štěstí, obdržíte ze systémů data v tabulární formě s jasně oddělenými sloupci ve formátu Microsoft Excel. Často ale nebudete mít štěstí vůbec, protože se bude jednat o textový soubor obsahující nepotřebné informace v záhlaví a v těžko čitelném formátu. Navíc není možno dostat všechna data v jednom souboru, protože systém umožňuje buď export prodejů po jednotlivých dnech pro jeden obchod, nebo pro všechny obchody pro zadané období. Pokud chcete sledovat denní vývoj, potřebujete obojí, a tudíž každý den exportujete data pro všechny obchody.

Nyní již víte, jaká data jsou k dispozici, a máte v Accessu připravené tabulky, do kterých budete data importovat. Je čas napsat první program ve VBA, jenž data upraví do podoby, kterou lze do tabulek správně naimportovat. Samozřejmě že elegantní je data upravovat přímo interakcí s textovým souborem, ale na psaní takového kódu nemáte čas a ostatně, je to zbytečné, neboť aplikaci Excel máte stejně otevřenou.

Založíte nový sešit, spustíte záznamník maker, otevřete soubor se syrovými daty a data upravíte do přijatelné podoby. Poté záznamník vypnete a napíšete efektivní kód, přičemž jako základ pro psaní složitějších metod a vlastností (`TextToColumns`, `FormulaR1C1`, apod.) použijete řádky nahrané záznamníkem. Během několika minut jste hotovi a kód sám otevře sešit, upraví data a opět jej uloží.

Dalším krokem je kód upravit tak, aby prohledal adresář na všechny podobné soubory, všechny upravil podobným způsobem a data zkopíroval pod sebe do hlavního sešitu. Odladění takového kódu chvíli trvá, ale odměnou vám bude konsolidace dat z několika desítek sešitů během několika sekund.

Dalším krokem bude napsání jednoduchého kódu, který data z konsolidovaného sešitu neimportuje do dočasné tabulky připravené v Accessu a spustí přidávací dotazy, které odpovídající data nahrají do příslušných tabulek. Během řádově několika desítek sekund máte ve vaší databázi nová aktuální data. Navíc jste si jisti, že pokud import proběhl zdárně, není narušena integrita dat (např. neexistují prodeje nepřiřazené prodejně).

Nyní je čas daty nakrmit vaše standardní sestavy v Excelu. Na kartě „Data“, skupina „Načíst externí data“ klepnete na „Z aplikace Access“. Postupujete podle instrukcí průvodce a případně nezapomenete přidat do buněk parametry, pokud nechcete do listu načíst všechny data, která dotaz připravený v Accessu poskytuje.

Jako konečný krok propojíte buňky ve vašich sestavách na listu Excelu s importovanými daty a jste hotovi. Pokud změníte parametr v buňce, během řádově několika sekund (podle objemu dat) získáte přehled pro nové období.

Poté se můžete zabývat doplňkovými tabulkami a údaji (adresy, vybavení, kategorie), abyste mohli udělat přesnou analýzu situace, ale s hlavní částí jste hotovi. Každodenní aktualizace dat zabere méně času, než pití ranní kávy a vy máte každý den k dispozici aktuální přehled.

Dále můžete automatizovat automatické rozesílání standardních zpráv na definované adresy přes aplikaci Microsoft Outlook a rovněž automaticky vygenerovat měsíční prezentaci pro akcionáře v aplikaci Microsoft Powerpoint.

Domníváme se, že při investici řádově desítek hodin to není špatný výsledek, a určitě je nyní jasné, proč má smysl se učit VBA. Tato kniha se věnuje téměř výhradně jazyku VBA a objektovému modelu Excel, a proto v ní nenaleznete podrobný návod, jak řešit všechny kroky z našeho příběhu. Přesto ale po nastudování všech příkladů byste měli být schopni podobné programy psát.

# Jak používat tuto knihu a vůbec, jak se učit VBA

Tato kniha si neklade za cíl stát se podrobným přehledem jazyka VBA a úplným popisem objektového modelu aplikace Microsoft Excel. O obou tématech byly napsány celé knihy a pouhý popis všech objektů aplikace Excel včetně jejich vlastností a metod by přesáhl obsah této knihy.

Jak jsme již řekli v úvodu, tato kniha je zaměřena na praktické využití jazyka VBA při programování aplikací v Excelu a řešení nejčastějších úloh. Autor je původcem více než 10000 příspěvků na mezinárodních fórech a zodpověděl tisíce dotazů. Samozřejmě že se dotazy často týkají podobných úloh, a přestože není možné kód napsat všeobecně, tato kniha se bude snažit představit techniky programování tak, aby je při rozumné míře přizpůsobení bylo možné použít pro vaše konkrétní potřeby.

I když předpokládáme, že čtenář zná základy programování ve VB nebo VBA, v prvních částech se zaměříme na popis nejpoužívanějších konstrukcí jazyka VBA a rovněž popis základních objektů aplikace Microsoft Excel. Zatímco jazyk VBA je obecný a program může být zapsán do modulu kterékoli aplikace, která hostování jazyka podporuje, objektový model Excelu je specifický pro tuto aplikaci a jazyk VBA ho přes standardní objekty, metody a vlastnosti využívá. Všechny důležité funkce jazyka VBA i základních objektů aplikace Excel se budeme vždy okamžitě snažit demonstrovat na příkladech konkrétního použitelného kódu. Někdy uvedeme vedle sebe kód běžný, který je snadno pochopitelný, a kód optimalizovaný pro výkon, který často obsahuje nezvyklé konstrukce. Takové konstrukce vždy hned objasníme.

V dalších částech knihy se zaměříme na praktické použití VBA pro řešení nejčastějších úloh, se kterými se pokročilý uživatel Excelu setkává, například:

1. Hromadná změna dat ve sloupci
2. Transformace dat do databázové podoby
3. Konsolidace dat z více sešitů

Všechny tyto techniky umožňují enormní ušetření času a nákladů na pracovní sílu. Vzhledem k tomu, s jakým množstvím různých sešitů Excel z různých zdrojů se uživatel setkává, je časová úspora prakticky neomezená.

Mocným nástrojem při konsolidaci dat může být jazyk SQL (Structured Query Language), což je obecný jazyk určený pro tvorbu dotazů nad databázovými tabulkami. Přestože jeho použití v Excelu má určitá omezení, ukážeme jeho použití na konkrétních příkladech. Samozřejmě že podrobné

vysvětlení jeho syntaxe není vzhledem k rozsahu této knihy možné. O tomto jazyku a jeho použití v aplikacích Microsoft Office přes technologii ADO (ActiveX Data Objects) byly rovněž napsány celé knihy.

Co se týče použití uživatelských formulářů, tato kniha vás podrobně provede tvorbou uživatelského formuláře pro editaci dat v databázové formě. Formulář bude záměrně navržen tak, aby představil použití všech praktických ovládacích prvků, které budete při návrhu vašeho konkrétního formuláře potřebovat. Pokud budete navrhovat váš vlastní formulář, měli byste v této lekci najít vždy odpověď na otázku, jak konkrétní ovládací prvek použít ve vašem kódu.

Samostatným tématem Excelu jsou kontingenční tabulky (Pivot Tables). Tento nástroj je velmi mocný při analýze databázových dat a jejich agregaci do přehledů. V praxi jsme se příliš často nesetkali s nutností kontingenční tabulku vytvořit od začátku v kódu, ale přesto si tento postup ukážeme, neboť se při něm seznámíte s jejím objektovým modelem.

Vzhledem k tomu, že tato kniha není referencí jazyka, ale jeho praktickou učebnicí, doporučujeme knihu alespoň jednou byť jen zběžně přečíst. Až se v budoucnu setkáte s konkrétním problémem, budete vědět, kde v knize najít konkrétní řešení, nebo alespoň návod k němu.

Jazyk VBA (a vlastně žádný programovací jazyk) se nenaučíte čtením popisu jednotlivých funkcí, objektů a možností jejich manipulace. Nejspolehlivější metodou je praxe. Programování je možno přirovnat k umění, jako je hudba. Můžete znát všechny noty a pravidla harmonie, ale to ještě neznamená, že jste schopni napsat skladbu, která se bude líbit. Podobně můžete do podrobností znát příkazy jazyka VBA a celý objektový model Excelu, ale to ještě neznamená, že napíšete program, který náročnou úlohu vyřeší za několik sekund. Přestože programování podobně jako hudba vyžaduje určitou dávku talentu, při určité dávce praxe a studia technik zkušených programátorů je možno za několik měsíců psát efektivní kód prakticky pro jakoukoli úlohu. A právě ke zkrácení této doby vám chce pomoci tato kniha. Najdete v ní mnoho ověřených řešení, která byste jinak dlouho hledali na internetu nebo v literatuře (podobně jako autor).

# Otázky k částem lekcí

Za většinou částí lekcí najdete kontrolní otázky, které prověří, jestli jste porozuměli vykládanému textu. I když na konci knihy naleznete řešení, snažte se otázky nejprve zodpovědět samostatně, protože se většinou jedná o základy dané lekce.

Pokud je v řešení uveden příkladový kód, nejedná se samozřejmě většinou o jedinou možnost. Snažili jsme se však napsat vždy kód co nejefektivnější, který splňuje nároky na rychlost a současně jednoduchou upravitelnost.

V řešení jsme vždy danou otázku zopakovali, a tak vlastně příloha slouží i jako rychlá příručka řešení mnoha malých úloh, se kterými se budete v praxi setkávat.

# Kód není vždy nejefektivnější řešení

Přestože tato kniha je téměř výhradně o programování aplikace Excel v jazyce VBA, musíme dát důrazné upozornění. Aplikace Microsoft Excel je složitý program a veškeré jeho části jsou zkompilovány (přeloženy) do jazyka, kterému procesor ve vašem počítači přímo rozumí a rychle reagu-

je. Oproti tomu váš kód VBA, zapsaný do normálních modulů nebo do modulů tříd, zůstává i po kompilaci (Debug => Compile Project) jazykem interpretovaným. To znamená, že příkazy nejsou předány procesoru přímo, nýbrž musí být nejprve za běhu programu přeloženy a vykonány. To znamená, že i ta nejefektivněji zapsaná uživatelská funkce ve VBA bude v naprosté většině pomalejší, než vestavěný vzorec Excelu.

Pokud programujete aplikaci pro Excel, snažte se vždy o spojení obou světů. Vytvořte list nebo sadu listů, která využívá metody dostupné přes uživatelské rozhraní (vzorce, ověření dat, podmíněné formátování, apod.), a programem psaným ve VBA tuto aplikaci pouze podpořte. V praxi je mnohem jednodušší a rychlejší vytvořit dokonalou šablonu, kterou přes VBA naplníte libovolnými daty, než vytvářet celý sešit přes VBA! A to nemluvíme o údržbě. Pokud si váš šéf vymyslí nový formát zprávy, musíte kód měnit na mnoha místech. Pokud budete mít vytvořenou šablonu, změníte formát jednoduše přes uživatelské rozhraní, případně v kódu upravíte oblasti, kam budete zapisovat hodnoty, a jste hotovi.

Další začátečnickou chybou je duplikace vestavěných dialogů a funkcí. Často se setkáváme s dotazem typu:

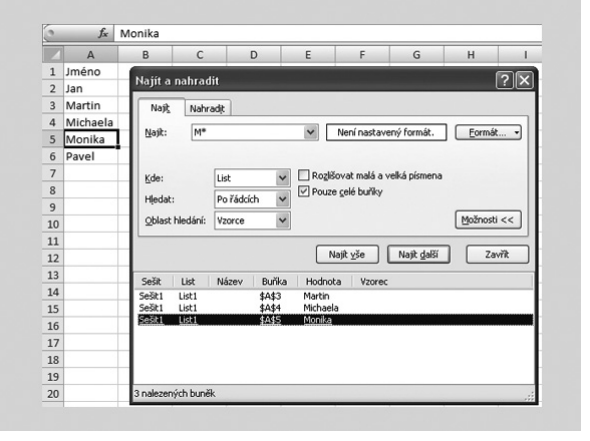
„Chci vytvořit formulář, do něhož uživatel zadá písmeno a program vybere první buňku, která začíná tímto písmenem."

Samozřejmě že Excel má pro tuto úlohu vestavěný poměrně mocný dialog „Najít a vybrat" ve skupině Úpravy. Tento dialog navíc umožňuje klepnout na tlačítko „Najít vše" a klikáním na jednotlivé odkazy se pohybovat po zvolených buňkách!

Je určitě jednodušší uživatelům vysvětlit použití tohoto dialogu (případně pokročilé uživatele odkázat na nápovědu), než vytvářet vlastní formulář, který bude funkčně omezený, a navíc budete muset kód měnit při každé změně zadání. Kromě toho lze například původní dialog „Najít" zobrazit přímo z VBA včetně parametrů:

```
Application.Dialogs(64).Show "M*", , , , , False, True
```

Jedinou výjimkou jsou takzvané diktátorské aplikace (*Dictatorship Applications*). V těchto vysoce profesionálních aplikacích, ušitých klientu na míru, se uživatelské rozhraní nahrazuje rozhraním vlastním a běžné funkce Excelu jsou pro uživatele nedostupné. Důvodem bývá nutnost omezit lidský faktor při zadávání dat, kdy například vložení sloupce nebo změna hlavičky může zcela zmást proceduru importu do jiné aplikace. Důvodem bývá i bezpečnost dat, ale jak se zmíníme v závěru knihy, tato bezpečnost je v Excelu dosti rozporuplná.




**Obrázek Ú.1:** Hromadné hledání buněk s obsahem odpovídajícím kritériu

Vyvarujte se i přílišného obtěžování uživatele vyskakujícími okny se zprávami (funkce `MsgBox`). Nejenže takové chování je nestandardní a běžného

uživatele může bez důkladné nápovědy zcela zmást, ale nevhodné naprogramování může celou aplikaci prakticky paralyzovat!

Příkladem může být tento dotaz:

„Ve sloupci A mám čísla faktur a ve sloupci B jejich lhůtu splatnosti. Chci napsat program ve VBA, který zobrazí upozornění pro každou fakturu po splatnosti."

Standardní odpověď může obsahovat tento kód, který úlohu vyřeší:

*Soubor 0 Uvod.xlsm*
*modul Module1*

```
Sub AlertOverdue()
Const MSG_TITLE As String = "Faktura po splatnosti"
Dim cl As Range, rg As Range
Dim msg As String
With ActiveSheet
    Set rg = .Range("B2", Cells(.Rows.Count, "B").End(3))
End With
For Each cl In rg
    If cl < Date Then
        MsgBox cl(, 0) & ": " & cl, vbOKOnly, MSG_TITLE
    End If
Next
End Sub
```

Jeho použití ovšem nedoporučujeme, pokud se – což je pravděpodobné – na seznamu vyskytuje po splatnosti faktur více. Jestliže má seznam několik desítek nebo dokonce stovek takových položek, uživatel by se z kódu nedostal, dokud by neodklepal všechna vyskakující okna se zprávou! Pokud chcete zobrazit všechny problematické položky v jedné zprávě, použijte tento kód:

*Soubor 0 Uvod.xlsm*
*modul Module1*

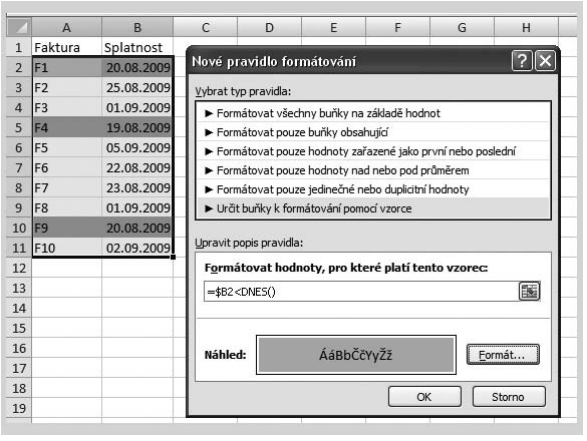
```
Sub AlertOverdue2()
Const MSG_TITLE As String = "Faktury po splatnosti"
Dim cl As Range, rg As Range
Dim msg As String
With ActiveSheet
    Set rg = .Range("B2", Cells(.Rows.Count, "B").End(3))
End With
For Each cl In rg
    If cl < Date Then
        msg = msg & cl(, 0) & ": " & cl & vbLf
    End If
Next
MsgBox msg, vbOKOnly, MSG_TITLE
End Sub
```

V praxi však zobrazení okna se zprávou mnoho nepomůže. Co s ním uživatel udělá? Opíše si všechna čísla, než klepne na OK? Práce s takovou aplikací by určitě nebylo nic příjemného.

Proto doporučujeme použít podmíněný formát:

1. Vyberte souvislou oblast s daty ve sloupcích A a B tak, aby buňka A2 byla aktivní
2. Na kartě „Domů“ klepněte na „Podmíněné formátování“ ve skupině „Styly“ a vyberte položku „Nové pravidlo“.
3. V následujícím dialogu klepněte na položku „Určit hodnoty k formátování pomocí vzorce“ a do textového pole zapište vzorec: =$B2<DNES(). Dialogové okno ještě nezavírejte.
4. Klepněte na tlačítko „Formát“, upravte formát podle potřeby (např. výplň buňky) a potvrďte klepnutím na „OK“.
5. Dialogové okno podmíněného formátu uzavřete rovněž klepnutím na tlačítko „OK“.

V naprosté většině případů je tazatel udiven robustností podmíněného formátu a volí toto řešení. Následujícím dotazem může být, jak implementovat podmíněný formát automaticky, pokud se seznam faktur dynamicky mění. Jeho hlavní problém je ale vyřešen rychle a elegantně, a to i ke spokojenosti ostatních uživatelů.




**Obrázek Ú.2:** Použití podmíněného formátu k upozornění na kritické řádky

Chtěli jsme uvést tyto příklady, abyste si uvědomili, že než začnete psát kód, je vždy vhodné si ujasnit, jaké je vlastně zadání. V posledním příkladu bylo přáním upozornit na faktury po splatnosti. Ačkoli je úlohu možné řešit přes VBA, podmíněný formát je o mnoho vhodnější a nabízí větší flexibilitu, například zobrazení různých dob prodlení různými barvami. Mimochodem, podmíněné formátování bylo v Excelu 2007 podstatně rozšířeno a podle nás je jedním z důvodů, proč přejít na vyšší verzi.

## Shrnutí

*Nesnažte se napodobovat standardní funkce Excelu.* Ve většině případů je to zbytečné a uživatele to jedině zmate. Navíc se vaše aplikace zpomaluje. Raději vysvětlete, jak se ta která funkce používá, a pokročilé uživatele odkažte na nápovědu.

*Pamatujte si, že v případě vyskakovacích oken a zpráv je méně více.* Neustálé zobrazování opakujících se zpráv a upozornění nesvědčí o vaší profesionalitě, ale o tom, že neumíte navrhnout list tak, aby uživatele logicky vedl ke správnému zadávání dat.

*Standardní vestavěné dialogy můžete zobrazit přímo z VBA*. V nápovědě hledejte výraz „XlBuiltInDialog Enumeration“ pro podrobný seznam dialogů. Pro seznam možných parametrů hledejte výraz „Built–In Dialog Box Argument Lists“.

# LEKCE 1

# Základy jazyka VBA

**V této lekci najdete:**

Jak jsme již řekli v úvodu, je jen jeden jazyk VBA, který může, ale nemusí používat objektové modely jiných aplikací a knihoven. Tato lekce vás seznámí se základy jazyka VBA a zaměří se na problematiku, kterou vývojář aplikací v Excelu hojně využije, totiž manipulaci s textem, čísly a daty. Většina příkladů, pokud nevyužívá pouze knihovnu VBA, využívá objekty aplikace Excel, protože koneckonců právě o programování Excelu tato kniha je. Kromě specifických příkladů je ale veškerý text platný i pro vývoj jiných aplikací, které hostují VBA na základě VB6 (od verze Office 9, neboli 2000). Většina kódu by pracovala i v Excelu 8 (97), který hostoval VBA na základě VB5. Nové funkce jsou například funkce `Join`, `Split`, `Filter` a `InStrRev`.

# Proměnné

Proměnnou si můžete představit jako pojmenované místo v paměti počítače, na nemž se nachází kousek dat, se kterými chcete pracovat. Data mohou mít samozřejmě různou podstatu a podobu, jako je celé číslo, záporný zlomek, iracionální číslo (např. druhá odmocnina ze dvou, číslo π či základ přirozených logaritmů, známý též jako Eulerovo číslo *e*), částka v domácí měně, seznam hodnot, datum, text, sešit Excelu, graf na listu Excelu, nebo dokonce celá běžící aplikace Microsoft Word.

Proměnných ve VBA je celá řada typů, jak je vidět na seznamu v tabulce 1.1.

**Tabulka 1.1:** Přehled typů proměnných

| Datový typ | Rozsah hodnot | Velikost v paměti |
|---|---|---|
| Boolean (Logický) | True nebo False | 2 byty |
| Byte | 0-255 | 1 byte |
| Currency (Měnový) | –922,337,203,685,477.5808 až 922,337,203,685,477.5807 | 8 bytů |
| Date (Datum) | 1. leden 100 až 31. prosinec 9999, čas 0:00:00 až 23:59:59 | 8 bytů |
| Decimal (Desetinný) | Maximální hodnota +/–79,228,162,514,264,337,593,543,950,335 s desetinnou čárkou v libovolném místě | 12 bytů |
| Double (Dvojitá přesnost) | –1.79769313486231 až –4.94065645841247E-324 pro záporná čísla, 4.94065645841247E-324 do 1.79769313486232E308 pro kladná čísla | 8 bytů |
| Integer (Krátké celé číslo) | –32,768 až 32,767 | 2 byty |
| Long (Dlouhé celé číslo) | –2,147,483,648 až 2,147,483,647 | 4 byty |
| Object (Objekt) | Uložený ukazatel (pointer) na libovolný objekt v paměti | 4 byty |
| Single (Jednoduchá přesnost) | –3.402823E38 až –1.401298E-45 pro záporné hodnoty a 1.401298E-45 až 3.402823E38 pro hodnoty kladné | 4 byty |
| String (Řetězec, „Text“) | Může obsahovat až 2^31 znaků, může mít i konstantní délku až do přibližně 64 000 znaků | Různá |
| User Defined (Uživatelský) | Může obsahovat jeden nebo více různých typů | Různá |
| Variant | Může obsahovat různé hodnoty a objekty, viz dále v této lekci | Různá |

## Nejdůležitější typy proměnných a kdy je použít

Protože tato kniha není vyčerpávajícím popisem jazyka VBA, nýbrž praktickou příručkou, zaměříme se pouze na nejpoužívanější typy při práci s aplikací Microsoft Excel.

### Proměnná typu Boolean

Proměnná obsahuje buď hodnotu `True` (Pravda) nebo `False` (Nepravda). Použijete ji jako podmínku v rozhodovacích příkazech a také jako návratovou hodnotu mnoha uživatelských funkcí, např.

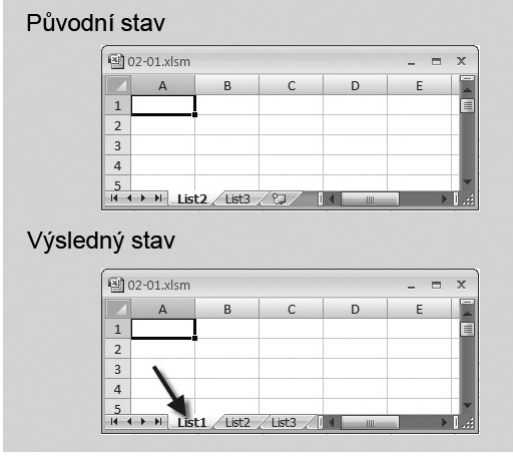
*01-01 Proměnné.xlsm*
*modul MBoolean*

```
'// Funkce vrátí True, pokud list zadaného jména v aktivním sešitu existuje

Public Function WorksheetExists(SheetName As String) As Boolean

On Error Resume Next
    WorksheetExists = Len(Worksheets(SheetName).Name) > 0
    Err.clear
On Error Goto 0

End Function
```

Přičemž tuto funkci použijete například takto:

```
'// Pokud list se zadaným jménem v aktivním sešitu neexistuje,
'// přidej ho, pojmenuj a přesuň na začátek

Const SheetName As String = "List1"
If Not WorksheetExists(SheetName) Then
    With Worksheets.Add
        .Name = SheetName
        .Move Before:=Sheets(1)
    End With
End If
```

Kód přidá list, pokud neexistuje, podle obrázku:




**Obrázek 1.1:** Přidání nového listu na začátek sešitu

Typické použití proměnné Boolean v kódu je

```
If PromennaBoolean Then
    ' Vykonej, pokud True
Else
    ' Vykonej, pokud False
End If
```

### Proměnná typu Long

Kromě proměnných, které ukládají ukazatel (pointer) na objekt, budete tento typ proměnné používat asi nejčastěji. Typické použití je čítač v cyklu a uložení čísla řádku nebo sloupce.


*01-01 Proměnné.xlsm*
*modul MLong*

```
'// Kód vynásobí každou druhou buňku ve sloupci A
'// aktivního listu dvěma, počínaje druhým řádkem

Dim i As Long 'Citac
Dim lLastRow As Long 'poslední řádek

lLastRow = Cells(Rows.Count, 1).End(xlUp).Row

For i = 2 To lLastRow Step 2
    Cells(i, "A") = Cells(i, "A") * 2
Next i
```

Proměnnou typu `Integer` jsme vynechali úmyslně. Ušetření paměti je minimální a s proměnnou typu Long zachází dvaatřicetibitová architektura počítače rychleji. Kromě toho je při velikosti listu v Excelu 2007 (a vlastně i v několika předešlých verzích) reálné nebezpečí, že bychom se do proměnné pokusili přiřadit hodnotu mimo její rozsah, což by vyvolalo *run-time* chybu (chybu za běhu) programu. Pokud budete pro celá čísla používat typ Long, ve většině případů bude váš kód bezpečný.

#### Proměnná typu String

Tuto proměnnou budete používat pro uložení řetězcových proměnných, čili zjednodušeně řečeno, textu. Mějte však na paměti, že operace s textem jsou jedny z nejpomalejších ve VBA, a používejte je s rozmyslem, zejména pak v cyklech.

Předpokládejme, že buňky A1:A5 obsahují po jednom slovu, které chcete spojit v jeden řetězec, kde budou jednotlivá slova oddělená čárkou.

Obvyklý kód vypadá přibližně takto:


*01-01 Proměnné.xlsm*
*modul MString*

```
'//spojení slov v jeden text v cyklu

Dim cl As Range
Dim strText ' výsledný text
For Each cl In Range("A1:A5")
    strText = strText & cl.Value & ","
Next cl
strText = Left$(strText, Len(strText) - 1)
Debug.Print strText
```

Efektivní kód však vykoná stejnou práci bez opakovaného spojování řetězců:

```
'//spojení slov v jeden text bez cyklu

Dim y
Dim strText ' výsledný text
y = Application.Transpose(Range("A1:A5"))
strText = Join(y, ",")
Debug.Print strText
```

Jak kód druhého typu pracuje, se dozvíte v následující části této lekce.

#### Proměnná typu Double a Currency

S těmito typy budete pracovat méně, neboť při programování aplikace Excel jsou hodnoty většinou vypočítávány vzorci na listu. Přesto se jejich použití nevyhnete například při výpočtu degresivní provize v závislosti na prodeji Proměnná typu Double má jednu velikou výhodu a zároveň nevýhodu. Výhodou je, že rozsah hodnot je skutečně obrovský a kromě mezních astrofyzikálních výpočtů týkajících se budoucnosti vesmíru pojme jakékoli smysluplné číslo. Na druhou stranu, čísel v daném rozsahu je nekonečné množství a proměnná má k dispozici omezený počet bitů. Proto rovněž nekonečné množství čísel nemůže být zobrazeno s absolutní přesností. Již relativně běžná čísla mohou vést k neočekávaným výsledkům:

*01-01 Proměnné.xlsm*
*modul MDouble*

```
Dim a1 As Double, a2 As Double, a3 As Double
a1 = 1
a2 = 0.3451
a3 = 0.6549
Debug.Print (a1 - a2 - a3) = 0
Debug.Print a1 = a2 + a3
Debug.Print a1 - a2 - a3
```

Protože a1 = a2 + a3, na výpisu v okně Immediate (Ctrl+G, pokud se nacházíme ve VB Editoru) bychom čekali následující údaje:

```
True
True
0
```

Výstup však zobrazí částečně jiné výsledky:

```
False
True
-5,55111512312578E-17
```

Přičemž první výsledek je vyloženě špatně a třetí je sice velmi blízko nule, avšak nula to není. Je nutné si uvědomit, že chyba se takzvaně *propaguje* do výpočtu, neboli její vliv se s nárůstem počtu operací s chybným číslem zvětšuje. Ještě větším nebezpečím je použití proměnné typu Double jako podmínky pro ukončení cyklu, neboť nikdy nemusí nabýt přesné očekávané hodnoty. Toto může váš kód poslat do nekonečného cyklu, ze kterého je vysvobozením pouze <Ctrl>+<Break> (v lepším případě), nebo „sestřelení“ Excelu přes Task Manager (v horším případě).

V naprosté většině nefyzikálních výpočtů je řešením nahradit typ `Double` typem `Currency`:

```
Dim a1 As Currency, a2 As Currency, a3 As Currency
a1 = 1
a2 = 0.3451
a3 = 0.6549
Debug.Print (a1 - a2 - a3) = 0
Debug.Print a1 = a2 + a3
Debug.Print a1 - a2 - a3
```

Což vrátí výpis

```
True
True
0
```

čili očekávaný výsledek.

Uvedeme ještě jeden příklad, demonstrující, jak nebezpečné je použít proměnnou typu `Double` v podmínce cyklu:

*01-01 Proměnné.xlsm*
*modul MDouble*

```
Sub TestDouble3()

'// Bez ochranného mechanizmu by tento kód skončil
'// v nekonečné smyčce

Const MAX_LOOP As Long = 999999
Dim i As Long
Dim dblHodnota As Double
i = 1
Do
    dblHodnota = dblHodnota + 0.0001
    If dblHodnota = 1 Then GoTo ExitSub

    '// Ochranný kód eliminující nekonečný cyklus
    i = i + 1
    If i > MAX_LOOP Then GoTo ExitSub

Loop

ExitSub:
    Debug.Print "Hodnota: " & dblHodnota & vbLf & "Pocet cyklu: " & i
End Sub
```

Kód vykonává velmi jednoduchou činnost. Přičítá hodnotu 0,0001 ke stávající hodnotě proměnné `dblHodnota` a po dosažení hodnoty rovné jedné se tato hodnota vypíše společně s počtem cyklů a program se ukončí. Na výstupu bychom tedy očekávali následující text:

```
Hodnota: 1
Pocet cyklu: 10000
```

Ovšem kód po svém ukončení vypíše něco jiného:

```
Hodnota: 99,9999000021961
Pocet cyklu: 1000000
```

Neboli kód se *neukončil* v momentě, kdy `dblHodnota` nabyla hodnoty jedna, protože ona přesně této hodnoty nikdy nenabyla! Přesně řečeno, v 10000. cyklu nabyla hodnoty 0,999899999999906, což je sice hodnota velmi blízká jedné, ale přesně jedna to není.

Opět, pro číslo do čtyř desetinných míst může pomoci použití typu `Currency`:

```
Sub TestCurrency()
Dim i As Long
Dim currHodnota As Currency
i = 1
Do
    currHodnota = currHodnota + 0.0001
    If currHodnota = 1 Then GoTo ExitSub
    i = i + 1
Loop

ExitSub:
    Debug.Print "Hodnota: " & currHodnota & vbLf & "Pocet cyklu: " & i
End Sub
```

Všimněte si, že tento kód nepotřebuje obranný kód proti nekonečné smyčce. Pro profesionální aplikace vám ale jeho použití vřele doporučujeme, neboť určitě nechcete, aby vám váš klient volal, že mu „zamrzl" Excel po půldenním usilovném zadávání dat. K této problematice se ještě vrátíme v části popisující cyklus `Do…Loop`.

Závěrem shrňme, že ačkoli Excel považuje čísla v buňkách implicitně za typ `Double`, v naprosté většině případů je vhodné je přiřadit do proměnné typu `Currency`. Uvědomte si, že do této proměnné nemusíte ukládat pouze údaje v penězích. Proměnnou můžete použít pro všechna čísla v daném rozsahu, která chcete zobrazit s absolutní přesností na čtyři desetinná místa.

V případě, že proměnnou typu `Currency` použít nemůžete, buďte opatrní při porovnávání hodnot a používejte zaokrouhlení a intervaly.

#### Proměnná typu Object

Proměnná tohoto typu bude pravděpodobně jednou z nejčastějších proměnných, se kterými budete ve VBA kódu pro Excel pracovat. Existují pro to dva důvody, *rychlost* a *čitelnost* kódu.

Existují dva základní druhy použití:

```
Dim oObject As Object

Dim wdApp as Word.Application
Dim ws As Worksheet
Dim rg As Range
```

V prvním případě předem nevíme, kterého typu objekt bude a tudíž kterou třídu mu přiřadit. Proto proměnnou deklarujeme s obecnou třídou `Object`. Typicky se u tohoto typu deklarace setkáte při automatizaci jiných s aplikací tzv. *pozdní vazbou (Late Binding)*.

V ostatních případech známe, jaké třídy objekt bude, *a zároveň* je knihovna obsahující tento objekt zahrnuta v referencích sešitu, ve kterém je tento kód zapsán (VBE=>Tools=>References). Toto je velmi důležité! Objektové knihovny Excelu, VBA a Office zde budou zaškrtnuty standardně, jiné, např. Word, ADO či Outlook, musíte vyhledat a zaškrtnout, případně dokonce nejprve nainstalovat, pokud chcete jejich objekty explicitně deklarovat a používat. Tento přístup má mnohé výhody, jak se dozvíte v části o automatizaci aplikací v lekci o psaní efektivního kódu.

### Proměnná typu Variant

Tuto specialitu jazyka VB a VBA jsme si nechali na konec přehledu nejpoužívanějších typů. `Variant` je flexibilní typ proměnné, který může pojmout jakoukoli hodnotu kromě řetězce o pevně dané délce (fixed–length string), včetně hodnot a objektů `Empty`, `Error`, `Nothing` a `Null`. Ačkoli od verze Excelu 2002 může být do proměnné typu `Variant` přiřazen i uživatelský typ proměnné, toto použití je velmi vzácné.

Ve všech těchto případech bude proměnná `var` typu `Variant`:

```
Dim var As Variant
Dim var, i As Long
Dim var
```

Pokud není v záhlaví modulu uvedeno `Option Explicit`, je možno i deklaraci vynechat a proměnnou použít rovnou v kódu. Bude automaticky typu `Variant`:

```
var = var + 1
```

Poslední přístup se zásadně nedoporučuje, jak si řekneme v části o deklaraci proměnných a jejich rozsahu.

Pro zjištění podtypu proměnné, která se skrývá v proměnné `Variant`, můžete použít funkci `VarType`:

```
If VarType(Var) = vbLong Then
'// kód, který pracuje pouze s celými čísly
End If
```

Jiná funkce, která zjistí typ proměnné, je funkce `TypeName()`.

Přestože použití proměnné typu `Variant` může vypadat lákavě, nepoužívejte ji kromě případů, kdy je to nutné. Jedním takovým případem je nutnost ošetřit chybu vrácenou funkcí za běhu programu. Tato funkce vrátí hodnotu typu `Error`, pokud hledaná hodnota není v seznamu:

*01-01 Proměnné.xlsm*
*modul MVariant*

```
Function NajdiMne(arg As Long)
    NajdiMne = Application.Match(arg, Array(1, 3, 5), 0)
End Function
```

Všimněte si, že celá funkce nemá deklarovaný typ, čili je implicitně typu `Variant`, aby mohla vrátit chybu jako typ proměnné. Pokud by byla deklarována jako typ `Long`, došlo by k run-time chybě 13: Type Mismatch, neboli kolizi typu (`Error` versus `Long`).

Vrácenou hodnotu můžete ověřit, než výsledek použijete v dalším kódu:

```
Sub TestujFunkci()
Dim Vysledek
Výsledek = NajdiMne(2)
If TypeName(Vysledek) = "Error" Then
    MsgBox "Hodnota nenalezena"
Else
'// kód pokračuje, hodnota nalezena
End If
End Sub
```

Druhou důležitou vlastností proměnné typu Variant je schopnost pojmout pole o předem neznámé velikosti a typu. Tato vlastnost je pro programátora Excelu natolik důležitá, že ji podrobněji rozvedeme v příští lekci.

Ve všech ostatních případech vám radíme používat konkrétní typ. Je to efektivnější z hlediska využití paměti a výkonu počítače, vyvarujete se některých základních chyb a váš kód bude mnohem čistší a snadněji udržovatelný.

### Poznámka

Proměnná typu `Variant` je jedním z důvodů, proč VBA není „opravdovými programátory" považován za plnohodnotný programovací jazyk. Visual Basic se jim zdá být příliš benevolentní, což vede ke zvýšenému riziku výskytu těžko odhalitelných chyb. „Opravdový programátor" není pojem, který jsem vymyslel, nýbrž jím sami sebe označují programátoři v jazycích C, C++, C# a podobně. Určitě je pravda, že VBA má daleko do „strong–typed" jazyka, tedy jazyka, který umožňuje pouze explicitní deklarace a konverze datových typů. Na druhou stranu ale moudré použití proměnných typu Variant umožňuje psát pro aplikace z rodiny Microsoft Office kód rychle a efektivně, a přesně o tom je tato kniha. Držte se především dvou pravidel: Mějte vždy zapnutou explicitní deklaraci (`Option Explicit`) a proměnné typu `Variant` používejte jen tehdy, když přesně vyhraněný typ selže. Při psaní kódu pravidelně používejte kompilaci (Debug=>Compile …) a budete mít jistotu, že jste – alespoň co se týče deklarace a následné použití jmen proměnných – nikde neudělali chybu.

Je výborné znát angličtinu nebo němčinu, ale pokud znáte oba jazyky, váš obzor se neskutečně rozšíří. A podobně je to i s jazyky programovacími. Ne náhodou nejlepší vývojáři ve VBA znají i C++ a jeho výhody při vytváření např. knihoven funkcí. Bohužel, toto téma již přesahuje rámec této knihy.

## Pole a jejich využití

Pole je indexovaná skupina dat, která se chová jako jedna proměnná. Na jednotlivé prvky pole se odkazujeme *celočíselným* indexem.

Jednorozměrné pole (pole o jedné dimenzi) můžeme deklarovat různým způsobem:

```
Dim Prodeje (2002 To 2009) As Double
Dim Prodeje (1 To 8) As Double
Dim Prodeje (7) As Double
Dim Prodeje() As Double
Dim Prodeje
```

Na prvním příkladu vidíte, že spodní hranice nemusí nutně začínat indexem 0 nebo 1. Pokud do pole ukládáte například roční hodnoty, může být výhodné pro čitelnost použít index korespondující s rokem.

Ve třetím příkladu bude mít pole osm prvků a obvykle je tato deklarace identická s deklarací

```
Dim Prodeje (0 To 7)
```

Klíčové je zde slovo *obvykle*. Pokud totiž uvedete do záhlaví modulu klauzuli

```
Option Base 1
```

bude pole bez deklarované spodní hranice začínat prvkem s indexem 1.

Stejně jako vám doporučujeme *vždy* v modulu uvádět `Option Explicit`, doporučujeme *nikdy* nepoužívat výraz `Option Base 1`. Důvod je ten, že pokud kód zkopírujete do modulu bez této klauzule, kód nemusí pracovat správně. Takovou chybu může být velmi složité odhalit, neboť se typicky jedná o chybu algoritmu, a ne o chybu syntaktickou nebo run-time chybu (chybu za běhu). Tím se zbavíte možnosti kód beze ztráty funkčnosti přenášet do jiných aplikací. Zvykněte si spodní hranici *vždy* explicitně deklarovat a kód bude vždy pracovat očekávaným způsobem.

Čtvrtý příklad deklarace je znám jako Dynamické pole a použijete jej v případě, že předem neznáte počet prvků pole. Příkladem je kód, který naplní pole SheetNames jmény listů v aktivním sešitu. Počet listů je předem neznámý:


*01-02 Pole.xlsm*
*modul MArrays*

```
Dim SheetNames() As String
Dim i As Long
ReDim SheetNames(1 To Sheets.Count)
For i = LBound(SheetNames) To UBound(SheetNames)
SheetNames(i) = Sheets(i).Name
Next i
```

Ačkoli je možné již zčásti naplněné pole předimenzovat při zachování již existujících prvků za pomoci příkazu `Redim Preserve`, používejte tento postup, jen pokud je to absolutně nezbytné. VBA musí nejprve vytvořit nové pole v paměti počítače a poté do něj prvky překopírovat, než uvolní paměť zabranou původním polem. Tento proces je (v řeči počítače) velmi pomalý. Pokud je tento příkaz použit v kódu jednou, nezaznamenáte žádné výrazné zpomalení, ale ve smyčce o velkém počtu průchodů může ztráta času exponenciálně narůstat s každým průchodem cyklem.

Jako příklad uvedeme naplnění pole jmény souborů typu Word, které se nacházejí v daném adresáři a jméno souboru začíná řetězcem „Test“.


*01-02 Pole.xlsm*
*modul MArrays*

```
Dim FileNames() As String
Dim FileName As String
Dim strFolder As String

'// Nadefinujeme adresář
strFolder = ThisWorkbook.Path & "\TestFiles\"

'// Pokusíme se najít první soubor
FileName = Dir$(strFolder & "Test*.doc")

If Len(FileName) > 0 Then
'// Víme, že alespoň jeden soubor existuje

    ReDim FileNames(1 To 1)
    FileNames(1) = FileName
    FileName = Dir$

    Do While Len(FileName) > 0
    '// Pokud nalezen další soubor, předimenzuj pole
```

```
    '// a ulož nové jméno na konec pole
        ReDim Preserve FileNames(1 To UBound(FileNames) + 1)
        FileNames(UBound(FileNames)) = FileName
        FileName = Dir$
    Loop
End If
```

Velmi zajímavou možností pro programátora Excelu je naplnit dynamické pole přímo daty z listu. Podmínkou je, že pole musí být typu `Variant`.

**Poznámka**

Od doby, kdy jsem objevil, že Excel umí oblast buněk načíst přímo do pole, se „quicked–stacked" pole stala neodmyslitelnou součástí mého stylu psaní kódu. Nevím, jak termín „quicked–stacked" přesně přeložit. Představte si to jako proces, kdy pole nadimenzujete i naplníte současně. Myslím, že kromě Visual Basicu žádný jiný jazyk tento přístup nepodporuje, takže pokud se učíte programovat například v C#, rozhodně na tento přístup zapomeňte.

Řekněme, že máme oblast buněk, ve kterých chceme odříznout první znak. Můžeme vložit pomocný sloupec a vložit vzorec =MID(A1,2,LEN(A1)), poté sloupec okopírovat a vložit jako hodnoty přes původní data, a nakonec jej opět vymazat. Můžeme rovněž napsat kód podobný tomuto:

```
'// Obvyklý Kód

Dim cl As Range
For Each cl In Range("A1:A10")
....cl.Value = Mid$(cl.Value,1,Len(cl.Value))
Next cl
```

Přestože na tomto kódu není vůbec nic špatného, jednu drobnou vadu přeci jen má. Při každém průchodu cyklem třikrát interaguje s aktivním listem. Dvakrát hodnotu čte a potřetí ji zapisuje. Samozřejmě, jedno čtení můžeme ušetřit uložením hodnoty do dočasné proměné, ale výrazného ušetření času nedosáhneme.

Pokud použijeme pole, veškerá kalkulace se odehraje pouze v paměti RAM počítače a konečný výsledek zapíšeme na list najednou:

```
'// Rychlý Kód

Dim y, i As Long
y = Range("A1:A10")
For i = LBound(y) To UBound(y)
y(i, 1) = Mid$(y(i, 1), 2, Len(y(i, 1)))
Next i
Range("A1").Resize(Ubound(y)).Value = y
```

První řádek deklaruje dvě proměnné, první typu Variant a druhou typu Long. VBA neumožňuje deklarovat více proměnných jednou klauzulí `As`. Každá proměnná, která bezprostředně nesousedí s klauzulí `As <TypProměnné>`, bude typu `Variant`.

Kvůli čitelnosti by řada programátorů zvolila explicitní deklaraci:

```
Dim y As Variant
Dim i As Long
```

Doporučuji používat deklaraci na jednom řádku, protože tyto dvě proměnné k sobě logicky patří a rovněž se tím zlepšuje čitelnost kódu.

**Poznámka**

Osobně příliš nepoužívám tzv. Maďarskou notaci, která stanovuje pravidla pojmenovávání proměnných. Vedl jsem velmi obsáhlé debaty s vývojáři ve VBA a nikdy jsme se neshodli, jestli standardní pravidla pojmenovávání proměnných ve VBA používat, či ne. Jediné, na čem jsme se vždy shodli, je důslednost. Každý programátor si časem vyvine určitý systém pojmenovávání proměnných, modulů apod. Není těžké po několika minutách jeho konvenci pochopit, ale je problém, když autor konvenci mění v každé rutině, kterou napíše. Mimochodem, malý trik: Když chcete zjistit, kterého typu je ta která proměnná, umístěte do ní v kódu kurzor a stiskněte Shift+F2. Editor vás přenese na její deklaraci. Bohužel, opačným směrem to funguje jen omezeně, ale Ctrl+Shift+F2 vás přenese alespoň na začátek řádku s proměnnou, jejíž typ jste zjišťovali.

Další řádek kódu

```
y = Range("A1:A10")
```

vykoná vlastně dva příkazy. Nejprve vytvoří dvojrozměrné pole typu Variant o rozměrech 10 × 1 (počet řádků × počet sloupců) a poté jej naplní hodnotami z buněk. Důležité je, že spodní hranice pole je vždy 1, na rozdíl od implicitní spodní hranice polí ve VBA, čímž skvěle koresponduje s adresami buněk v listu.

Alespoň věříme, že příkazy vykonává v tomto pořadí. Pokud by totiž načetl hodnotu, předimenzoval pole, hodnotu uložil a pokračoval až do konce oblasti, vykonával by vlastně příkaz `Redim Preserve` v cyklu. Tento příkaz společně s funkcí `Union` je jednou z nejpomalejších funkcí VBA a v cyklu se zásadně nedoporučuje, jak jsme se již zmínili v předešlé části.

Vzhledem k tomu, že nás zajímá rozměr pouze první dimenze (víme, že druhá je rovna jedné, podle rozměru oblasti), můžeme druhý parametr funkcí `Lbound(pole,<dimenze>)` a `Ubound(pole,<dimenze>)` vynechat a spustit cyklus s čítačem i, který všechny hodnoty v poli upraví:

```
For i = LBound(y) To UBound(y)
y(i, 1) = Mid$(y(i, 1), 2, Len(y(i, 1)))
Next i
```

Nakonec pomocí metody `Resize` objektu `Range` určíme oblast, která přesně pojme naše pole, a jako parametr předáme velikost první dimenze pole. Pole „vyprázdníme“ na list opět jediným příkazem:

```
Range("A1").Resize(Ubound(y)).Value = y
```

Samozřejmě nemusíme původní rozsah přepisovat, můžeme zaměnit „tvar“ pole na šířku, a dokonce bez jakékoli interakce s listy můžeme pole „vyprázdnit“ na jiný list:

```
Sheets("Sheet2")Range("A1").Resize(,Ubound(y)).Value= _

Application.Transpose(y)
```

Všimněte si, že tentokrát je rozměr pole použit jako druhý parametr metody `Resize` (protože měníme počet sloupců). K transformaci pole jsem si vypůjčil funci Excelu TRANSPOSE(). Je to další příklad toho, že vhodnou kombinací VBA a vestavěných fukcí získáme to nejlepší z obou světů.

Nyní uvedeme praktičtější příklad. Řekněme, že máte prvek ComboBox umístěný na formuláři a při otevření formuláře jej chcete naplnit seznamem ve sloupci A listu Sheet1. Buňka A1 obsahuje hlavičku a data o neznámém počtu začínají v buňce A2. Vše, co potřebujete, je umístit tento kód do modulu třídy formuláře:

*01-02 Pole.xlsm*
*modul Userform1*

```
Private Sub UserForm_Initialize()
    Dim y
    With Sheets("Sheet1")
        y = .Range("A2", .Cells(.Rows.Count, "A").End(3))
    End With
    Me.cboSeznam.List = y
End Sub
```

Obvyklý kód bez použití polí vypadá takto:

```
Private Sub UserForm_Initialize()
    Dim cl As Range
    With Sheets("Sheet1")
        For Each cl In .Range("A2", .Cells(.Rows.Count, "A").End(3))
            Me.cboSeznam.AddItem cl.Value
        Next cl
    End With
End Sub
```

Všimněte si, že kód volá metodu `AddItem` prvku `ComboBox` při každém průchodu cyklem. Elegance uložení seznamu do pole vynikne zejména tehdy, když potřebujeme seznam například setřídit před jeho předáním do ovládacího prvku a nemůžeme nebo nechceme třídit hodnoty na listu.

Tolik tedy ke „quick–stacked arrays" neboli rychle naplněným polím. V úvodu jsme ale slíbili, že vás budeme učit psát rychlý a efektivní kód. Pokud zůstaneme u našeho příkladu s formulářem, tady je:

```
Private Sub UserForm_Initialize()
With Sheets("Sheet1")
        Me.cboSeznam.List = _
        Application.Transpose(.Range("A2", .Cells(.Rows.Count, "A").End(3)))
End With
End Sub
```

Všimněte si, že celý příkaz je jeden jediný řádek. Osobně ale dáváme přednost kódu využívajícímu pole, protože je minimálně slušnost dbát rovněž na čitelnost kódu. Pokud tedy v knize uvidíte proměnné typu y a z, vězte, že se jedná o pole typu Variant, naplněná hodnotami z oblasti listu. Jestli vás toto téma zaujalo, experimentujte s přiřazováním oblasti do proměnné typu variant v kombinaci s funkcí `Transpose` a testujte horní hranici obou rozměrů. Ke „quick-stacked" polím se ještě jednou vrátíme v lekci o funkcích `Split`, `Join` a `Filter`.

## Vlastní kolekce a jejich využití

Kolekce je soubor objektů stejné třídy. List je objekt třídy Worksheet a patří do kolekce Worksheets, otevřený sešit je objektem třídy Workbooks a patří do kolekce Workbooks. Na rozdíl od polí se na prvky v kolekci můžeme odkazovat jak indexem, tak i klíčem, který bývá shodný se jménem objektu (vlastnost Name):

```
Debug.Print Worksheets("Sheet1").Visible = True
Debug.Print Worksheets(2).Name
```

Kolekce je zároveň objektem, který poskytuje různé metody a vlastnosti, jež umožňují objekty do kolekce přidávat, ubírat, zjišťovat jejich vlastnosti apod. Tyto metody však nejsou v každém objektovém modelu identické. Například, ve většině aplikací Micorsoft Office přidáte nový objekt zavoláním metody Add objektu kolekce:

```
Workbooks.Add
Documents.Add
Worksheets.Add
```

Avšak pokud chcete přidat například novou tabulku do databáze za pomoci knihovny ADOX, musíte nejprve vytvořit novou instanci objektu třídy Table, poté nastavit všechny potřebné vlastnosti a nakonec zavolat metodu `Append` kolekce `Tables` ADOX katalogu.

VBA umožňuje vytvořit kolekci vlastní. K manipulaci s jejími objekty potřebujete znát pouze tři metody a jednu vlastnost:

- **Metoda Add** přidá objekt do kolekce. Musíte předat odkaz na objekt a volitelně i jednoznačný identifikátor (např. jméno, kterým se později můžete na prvek odkazovat).
- **Metoda Remove** odebere objekt z kolekce. Metodě předáte index objektu nebo jednoznačný identifikátor.
- **Metoda Item** vrátí konkrétní objekt v kolekci, respektive odkaz na něj (pointer). Jako parametr opět předáte index objektu nebo jednoznačný identifikátor.
- **Vlastnost Count** vrátí počet objektů v kolekci.

Jedním z obvyklých použití kolekcí v Excelu je extrakce jedinečných hodnot z listu. Níže uvedený kód využívá faktu, že žádný objekt (v tomto případě hodnota) nemůže mít stejný jedinečný identifikátor. Proto je příkazem `On Error Resume Next` a `On Error Goto` vypnuto a opět zapnuto ošetření chyb (debugging). Výsledkem je, že kód neohlásí chybu, ale stejná hodnota se dvakrát nepřidá.

*01-03 Kolekce.xlsm*
*modul MCollections*

```
Dim Uniques As New Collection
Dim cl As Range, rg As Range
Dim i As Long
With ActiveSheet
    Set rg = .Range("A2", .Cells(.Rows.Count, "A").End(xlUp))
    On Error Resume Next
        For Each cl In rg
            Uniques.Add cl.Value, CStr(cl.Value)
        Next cl
    On Error GoTo 0
    .Range("B1") = "Uniques"
```

```
    For i = 1 To Uniques.Count
        .Cells(.Rows.Count, "B").End(xlUp)(2) = Uniques.Item(i)
    Next i
End With
```

Jiným příkladem může být kolekce, obsahující otevřené uživatelské formuláře (dialogy). Na rozdíl od Accessu, který má kolekci `Forms`, Excel žádnou kolekci typu Userforms, která by obsahovala otevřené formuláře, nemá. Můžeme ji však vytvořit:

```
Public MyForms As New Collection
'
Sub ShowForm()
    Dim lNr As Long, strUFName As String
    lNr = 1
    strUFName = "Userform" & CStr(lNr)
    VBA.UserForms.Add(strUFName).Show vbModeless
    MyForms.Add UserForms(0), strUFName
End Sub
```

Tento kód přidá do vlastní kolekce formulář Userform1 (který jste předtím vytvořili) a otevře ho. Poté se můžete na formulář odkazovat podobně jako v Accessu:

```
MyForms("UserForm1").Caption = "Muj formular"
```

Kolekce i pole mohou být velmi užitečné. Kolekce použijte, pokud potřebujete často přidávat a odebírat prvky, nebo pokud se chcete na prvky odkazovat jejich jménem. Pole jsou většinou rychlejší, ale mají své nevýhody při změně velikosti pole. V případě kolekce všechnu práci udělá VBA za vás.

## Deklarace, rozsah a platnost proměnných

Existují dvě základní vlastnosti každé proměnné, nezávisle na jejím typu:

- **Rozsah proměnné** určuje, které procedury mohou tuto proměnnou použít
- **Životnost proměnné** určuje, jak dlouho si proměnná podrží hodnotu, která byla do ní vložena

Pokud spustíte opakovaně tento kód:

*01-01 Proměnné.xlsm*
*modul MVarScope*

```
Sub Zivotnost()
    Dim Hodnota As Long
    Hodnota = Hodnota + 100
    MsgBox "Hodnota je " & Hodnota
End Sub
```

bude hodnota vždy 100, neboť proměnná Hodnota si drží svoji hodnotu, pouze dokud kód nedosáhne řádek `End Sub`. Poté je paměť uvolněna.

Životnost můžete prodloužit příkazem `Static`:

```
Sub Zivotnost2()
    Static Hodnota As Long
    Hodnota = Hodnota + 100
    MsgBox "Hodnota je " & Hodnota
End Sub
```

Pokud nyní budete střídavě spouštět oba kódy, v prvním případě bude hodnota 100, zatímco ve druhém případě se bude hodnota po stovkách zvyšovat. Je to proto, že druhá proměnná si podrží svoji hodnotu i po skončení běhu kódu a proměnná Hodnota v proceduře Zivotnost *není* stejná proměnná jako proměnná hodnota v proceduře Zivotnost2, protože jsou proměnné deklarovány *uvnitř* těla kódu. S oběma proměnnými je nakládáno *nezávisle*.

Pokud proměnnou nadeklarujete v záhlaví modulu (sekce deklarací, která se nachází pod příkazy typu `Option Explicit`), můžete ji použít ve více procedurách:

```
Dim mHodnota As Long
Sub Zivotnost3()
    mHodnota = mHodnota + 10
    MsgBox "Hodnota je " & mHodnota
End Sub
Sub Zivotnost4()
    mHodnota = mHodnota + 100
    MsgBox "Hodnota je " & mHodnota
End Sub
```

Pokud nyní spouštíte oba kódy, proměnná změní svoji hodnotu o 10 nebo o 100 podle toho, kterou proceduru spustíte. Rozsah proměnné jsou všechny procedury v modulu, kde je proměnná deklarována, a proměnná si podrží svoji hodnotu po celou dobu, kdy je sešit s tímto modulem otevřený.

Pokud chcete, aby proměnnou „viděly“ i procedury v jiných modulech, nadeklarujte proměnnou jako „veřejnou“ (klíčové slovo `Public`):

```
Public gHodnota As Long
```

**Tip**

I když podobně jako autor nedodržujete vždy tzv. Maďarskou konvenci pojmenovávání proměnných, zvykněte si proměnným dávat příponu „m“, pokud je platná pro modul, a „g“, pokud je platná pro celou aplikaci.

Jako odstrašující příklad se podívejte na tento kód, který se výše uvedeným tipem neřídí:

```
Dim Hodnota As Long

Sub Zivotnost()
    Dim Hodnota As Long
    Hodnota = Hodnota + 100
    MsgBox "Hodnota je " & Hodnota
End Sub
```

Proměnná deklarovaná na úrovni modulu bude pro proceduru neviditelná, protože proměnná s tímto jménem je deklarována uvnitř těla procedury a tato má prioritu! Pokud užijete předponu, na první pohled budete vědět, jestli se odkazujete na lokální, modulovou, nebo globální proměnnou.

## Otázky

**Otázka 1.1:**

Pokud potřebujete uložit do proměnné číslo posledního řádku jenž obsahuje data jenž typ proměnné použijete a proč?

**Otázka 1.2:**

Proč je následující kód nebezpečný a jak ho můžete upravit, aby plnil zamýšlený účel? Uveďte alespoň dvě možnosti.

```
Dim dblHodnota As Double

Do
    dblHodnota = dblHodnota + 0.01
    If dblHodnota = 1 Then Exit Do
Loop
```

**Otázka 1.3:**

Ve kterých případech je nutné použít proměnnou typu Variant? A ve kterém, pro Excel specifickém případě je její použití výhodné?

**Otázka 1.4:**

Proč je nebezpečné použití výrazu Option Base 1 v záhlaví modulu?

**Otázka 1.5:**

Napište co nejrychlejší kód, který v buňkách A1:A1000 odřízne poslední znak.

**Otázka 1.6:**

Napište kód, který do prvku typu listbox ve formuláři při jeho otevření nahraje data z prvního listu, od buňky A2 dolů, o neznámém počtu prvků.

**Otázka 1.7:**

Proč je výhodné využít vlastní kolekci k vygenerování jedinečných hodnot ze seznamu?

**Otázka 1.8:**

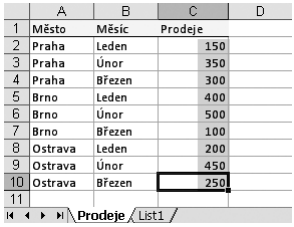
Jak nadeklarujete proměnnou, aby si svou hodnotu uchovala i poté, co kód doběhne?

# Důležité konstrukce VBA použité v knize

## With ... End With

*01-04 VBA Konstrukce.xlsm*
*modul MConstructions*

Pro příklad předpokládejme, že chcete ve VBA sečíst hodnoty na listu „Prodeje“ ve sloupci C od druhého řádku po poslední buňku podle obrázku 1.2.



| | A | B | C | D |
|---|---|---|---|---|
| 1 | Město | Měsíc | Prodeje | |
| 2 | Praha | Leden | 150 | |
| 3 | Praha | Únor | 350 | |
| 4 | Praha | Březen | 300 | |
| 5 | Brno | Leden | 400 | |
| 6 | Brno | Únor | 500 | |
| 7 | Brno | Březen | 100 | |
| 8 | Ostrava | Leden | 200 | |
| 9 | Ostrava | Únor | 450 | |
| 10 | Ostrava | Březen | 250 | |
| 11 | | | | |

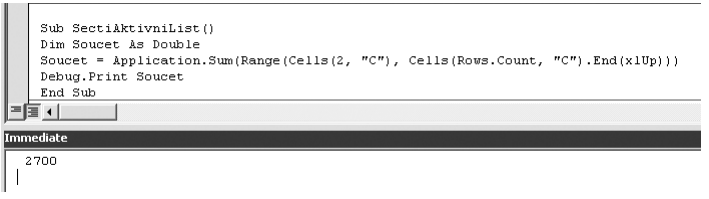
Prodeje | List1

**Obrázek 1.2:** Úkolem je sečíst čísla ve sloupci C

Pokud je list „Prodeje“ aktivní, následující kód bude pracovat správně,

```
Dim Soucet As Double
Soucet = Application.Sum(Range(Cells(2, "C"), Cells(Rows.Count, "C").End(xlUp)))
Debug.Print Soucet
```

a vrátí hodnotu 2700 do okna Immediate:



```
Sub SectiAktivniList()
Dim Soucet As Double
Soucet = Application.Sum(Range(Cells(2, "C"), Cells(Rows.Count, "C").End(xlUp)))
Debug.Print Soucet
End Sub
```

Immediate

```
 2700
```

**Obrázek 1.3:** Výpis výsledku procedury v okně Immediate

Problém ale nastane, pokud list „Prodeje“ není aktivní. Potom musíme vlastnost `Cells` (a raději i `Rows`) kvalifikovat správným objektem:

```
Dim Soucet As Double
Soucet = Application.Sum(Range(Sheets("Prodeje").Cells(2, "C"), _
        Sheets("Prodeje").Cells(Sheets("Prodeje").Rows.Count, "C") _
        .End(xlUp)))
Debug.Print Soucet
```

Tento kód se však velmi rychle čte a navíc, pokud byste jméno listu změnili, musíte kód upravit na třech (!) různých místech. Zdůrazňujeme, že se jedná o jeden řádek kódu, pro snazší čitelnost v editoru VBE zalomený mezerou a podtržítkem do více řádků. Právě v tomto případě je vhodné použít konstrukci `With...End With`:

```
Dim Soucet As Double
With Sheets("Prodeje")
    Soucet = Application.Sum(.Range(.Cells(2, "C"), _
                .Cells(.Rows.Count, "C").End(xlUp)))
End With
Debug.Print Soucet
```

Když nyní změníte jméno listu, musíte změnit kód na jednom jediném místě. V praxi by bylo ještě lepší jméno uložit do konstanty a umístit hned na začátek kódu. Případná úprava pak bude provedena velmi rychle a spolehlivě. Rovněž jsme si mohli dovolit umístit dodatečnou kvalifikaci před

objekt Range, což sice v tomto případě není nutné, ale kód je čistší, protože je zřejmé, že vrácená oblast bude patřit do listu jmenovaného za klíčovým slovem `With`.

Doporučujeme vám nikdy nepsat kód, který se implicitně spoléhá na aktivní list, protože to může vést k těžko odstranitelným chybám a navíc, kód vám může například **nenávratně přepsat a uložit data na aktivním listu!!!** Pokud má kód skutečně pracovat pro kterýkoli aktivní list, můžete vždy použít explicitní definici:

```
Dim Soucet As Double
With ActiveSheet
    Soucet = Application.Sum(.Range(.Cells(2, "C"), _
                .Cells(.Rows.Count, "C").End(xlUp)))
End With
Debug.Print Soucet
```

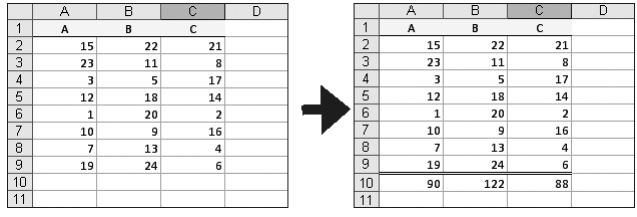
Pokud se později rozhodnete pro omezení, pro které listy má kód pracovat, změníte pouze jméno listu, případně zkontrolujete jméno aktivního listu v bloku `If - End If`. Zbytek kódu můžete nechat beze změny.

Na závěr uveďme, že bloky `With...End With` je možno i vnořovat:

```
With Sheets("List1")
    With .Cells(.Rows.Count, "A").End(xlUp).Resize(, 3)
        .Offset(1).FormulaR1C1 = "=SUM(R2C:R[-1]C)"
        .Borders(xlEdgeBottom).LineStyle = xlDouble
    End With 'poslední řádek
End With ' List1
```

Uvedený kód dvakrát podtrhne poslední řádek na listu „List1“ a vloží vzorce sčítající sloupce:

Všimněte si komentářů za klíčovými slovy `End With`. Pokud je počet řádků uvnitř bloku `With...End With` dlouhý, je vhodné označit, který z bloků se uzavírá. Na výsledek to samozřejmě nemá vliv, ale usnadňuje to čitelnost kódu.



| | A | B | C | D |
|---|---|---|---|---|
| 1 | A | B | C | |
| 2 | 15 | 22 | 21 | |
| 3 | 23 | 11 | 8 | |
| 4 | 3 | 5 | 17 | |
| 5 | 12 | 18 | 14 | |
| 6 | 1 | 20 | 2 | |
| 7 | 10 | 9 | 16 | |
| 8 | 7 | 13 | 4 | |
| 9 | 19 | 24 | 6 | |
| 10 | | | | |
| 11 | | | | |

| | A | B | C | D |
|---|---|---|---|---|
| 1 | A | B | C | |
| 2 | 15 | 22 | 21 | |
| 3 | 23 | 11 | 8 | |
| 4 | 3 | 5 | 17 | |
| 5 | 12 | 18 | 14 | |
| 6 | 1 | 20 | 2 | |
| 7 | 10 | 9 | 16 | |
| 8 | 7 | 13 | 4 | |
| 9 | 19 | 24 | 6 | |
| 10 | 90 | 122 | 88 | |
| 11 | | | | |

**Obrázek 1.4:** Vložení součtů a podtržení posledního řádku, jehož pozici předem neznáme

## Blok If … Else … End If

Tento blok je jedním z nejdůležitějších ve VBA a má několik variant.

Nejjednodušší varianta je zápis na jednom řádku:

```
If Intersect(Selection,Rows(1)) Is Nothing Then Exit Sub
```

Výše uvedený kód zajistí, že pokud je vybrána jakákoli oblast kromě oblasti náležející do prvního řádku aktivního listu, program se okamžitě ukončí.

Nenechte se zmást případným rozdělením řádku mezerou a podtržítkem na dva řádky. Jedná se stále o jeden řádek kódu:

```
If IsEmpty(cl.Offset(,1)) Then _
    cl.Value = "No Data"
```

Čili pokud je buňka nalevo od buňky přiřazené do objektové proměnné prázdná, do buňky je vložen text „No Data“.

Běžnější je použití `If` v bloku, kdy příkazy nacházející se uvnitř těla bloku budou vykonány, pouze je-li splněna podmínka mezi klíčovými slovy `If` a `Then`:

```
If Err.Number = 0 Then
    Call ImportData
    Call FormatSheet
End If
```

Pokud nedošlo k chybě, kód zavolá procedury, které naimportují data a zformátují list.

Blok `If` může vykonat jednu sadu příkazů, pokud je podmínka splněna, a jinou sadu příkazů, pokud je podmínka nesplněna:

```
If Plat > 100000 Then
    Dan = 30000 + (Plat-100000) * 0.40
Else
    Dan = Plat * 0.30
End If
```

Program spočítá daň, která je nad 100000 progresivní a stoupne z 30 % na 40 %.

Blok `If` může mít rovněž jednu nebo více sekcí `ElseIf`, pokud potřebujeme rozlišit mezi více podmínkami. Typickou úlohou je podbarvení buněk podle obsahované hodnoty (ačkoli v praxi tuto úlohu lépe řeší Podmíněný formát).

*01-04 VBA Konstrukce.xlsm*
*modul MConstructions*

```
Dim cl As Range
For Each cl In Range("A:A").SpecialCells(xlCellTypeConstants)
If cl.Value > 8 And cl.Value <= 10 Then
    cl.Interior.Color = vbGreen
ElseIf cl.Value > 5 And cl.Value <= 8 Then
    cl.Interior.Color = vbYellow
ElseIf cl.Value > 0 And cl.Value <= 5 Then
    cl.Interior.Color = vbRed
Else
    cl.Interior.Color = xlNone
End If
Next
```

Kód změní barvu pozadí buňky na červenou pro hodnoty mezi 8 a 10 včetně, na žlutou pro hodnoty od 5 do 8 včetně, a červenou pro hodnoty mezi 0 a 5 včetně. Pro ostatní buňky barvu pozadí změní na průhlednou (standardně bílou).

Pokud je klíčové slovo `If` následováno jedním nebo několika testy `ElseIf`, VBA provádí test, dokud nenajde splněnou podmínku. Vykoná příkazy v tomto bloku a pokračuje příkazem, který se nachází pod řádkem `End If`. Pokud není splněna ani jedna podmínka, jsou vykonány příkazi v sekci `Else`, pokud existuje.

Pokud není splněna ani jedna podmínka a sekce `Else` chybí, blok `If` nevykoná nic.

Bloky `If` můžete do sebe vnořovat:


*01-04 VBA Konstrukce.xlsm*
*modul MConstructions*

```
Dim Ans
Do
    Ans = InputBox("Zadejte číslo od 1 do 5")
    If Len(Ans) <> 0 Then
        If IsNumeric(Ans) Then
            If Ans < 1 Or Ans > 5 Then
                'číslo přesahuje hranice
                MsgBox "Číslo musí být od 1 do 5"
            Else
                'testy splněny, pokračuj
                Exit Do
            End If
        Else
            'Nesplněn test na číslo
            MsgBox "Musíte zadat číslo"
        End If
    Else
        'Žádné zadání, ukončit
        Exit Sub
    End If
Loop
```

Kód zavolá funkci VBA `Inputbox` a testuje zadání. Pokud není zadáno nic (klepnuto na tlačítko „Cancel"), kód se ukončí. Pokud není zadáno číslo, je zobrazeno varovné hlášení a dialogové okno se znovu otevře. Pokud číslo nesplňuje podmínku zadání, zobrazí se jiné varovné hlášení a dialogové okno se znovu otevře. Pokud jsou podmínky splněny, program pokračuje příkazem `Exit Do` a řádkem následujícím za klíčovým slovem `Loop`. V praxi se v podobném případě často ještě zavádí čítač, umožňující omezený počet pokusů.

## Blok Select Case

Blok `Select Case` má mnoho společného s blokem `If...ElseIF...Else...End If`, ale je podle našeho názoru elegantnější. Předpokládejme, že máme v listu sloupec s údajem o věku a chceme spočítat jízdné, které je 7 korun pro děti od 4 do 7 let včetně, 13 korun pro děti od 8 do 15 let, 26 korun pro osoby nad 15 let, a jízdné neplatí děti do tří let a senioři nad 66 let.

Funkce bude vypadat takto:

```
Function Jizdne(Vek As Long)
    Select Case Vek
    Case 0 To 3, Is > 66
        Jizdne = 0
    Case 4 To 7
        Jizdne = 7
    Case 8 To 15
        Jizdne = 13
```

```
    Case Is > 15
        Jizdne = 26
    Case Else
        Jizdne = CVErr(xlErrNA)
    End Select
End Function
```

V listu ji použijeme jako každou jinou vestavěnou funkci Excelu:

Funkce rozhodne o výši jízdného, a pokud žádná podmínka není splněna, vrátí chybovou hodnotu. Pokud máte pro každou podmínku jeden příkaz, můžete vložit dvojtečku mezi podmínku a příkaz a zapsat je na jeden řádek:

*01-04 VBA Konstrukce.xlsm*
*modul MFunctions*

```
Function Jizdne(Vek As Long)
    Select Case Vek
        Case 0 To 3, Is > 66:   Jizdne = 0
        Case 4 To 7:            Jizdne = 7
        Case 8 To 15:           Jizdne = 13
        Case Is > 15:           Jizdne = 26
        Case Else:              Jizdne = CVErr(xlErrNA)
    End Select
End Function
```

Pro větší přehlednost jsme příkazy oddělili tabulátorem.

VBA považuje příkazy oddělené dvojtečkou za samostatné řádky. Tento způsob zápisu může váš kód zkrátit a zpřehlednit (na rychlost provádění samozřejmě vliv nemá).

Tento způsob zápisu se často používá u rutinních příkazů, které následují za sebou, například tento kód zavře sadu záznamů a odstraní objektové proměnné z paměti

```
rs.Close: Set rs = Nothing
```

a tento kód ukončí automatizovanou aplikaci a odstraní objektové proměnné z paměti:

```
objWord.Quit: Set objWord = Nothing
```

Zajímavým a užitečným příkladem je automatické přihlášení na webovou stránku, např. na seznam.cz:

*01-04 VBA Konstrukce.xlsm*
*modul MConstructions*

```
Sub OpenSeznamAndLogin()
    Dim objIE As Object
    On Error GoTo Err_Handler
    Set objIE = CreateObject("InternetExplorer.Application")
    With objIE
      .Navigate "http://www.seznam.cz"
```

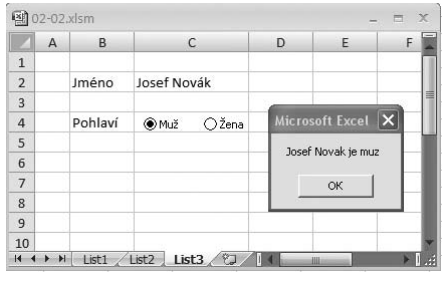

**Obrázek 1.5:** Použití vlastní funkce jako vzorce v listu

```
      Do While .Busy Or .ReadyState <> 4: DoEvents: Loop

      .Visible = True
      With .document
        .all.Item("login").Value = "VaseJmeno"
        .all.Item("password").Value = "VaseHeslo"
        .forms("login-fom").Submit
      End With
      .Visible = True
    End With
ExitHere:
    Set objIE = Nothing
    Exit Sub
Err_Handler:
    MsgBox "Unexpected Error, I'm quitting."
    objIE.Quit
    Resume ExitHere
End Sub
```

Všimněte si standardní smyčky `Do...Loop`, která *čeká na úplné načtení stránky*, než zapíše uživatelské jméno a heslo a formulář odešle na server k přihlašovacímu procesu. Je nutno podotknout, že automatizace přihlášení a vůbec interakce s HTML kódem webových stránek velmi závisí na designu stránky. Pokud seznam.cz stránku změní (testovali jsme v srpnu 2009), je možné, že tento příklad již fungovat nebude. Příklad obsahuje standardní postup ošetření chyb, o kterém pojednáme později.

Zajímavým způsobem využití bloku `Select Case` je jeho obrácené použití. Místo, abychom za klíčová slova `Select Case` zapsali výraz, který chceme testovat, zapíšeme za ně očekávanou hodnotu a testujeme jednotlivé výrazy na rovnost.

Typické použití je při testování ovládacích prvků typu Přepínač (OptionButton) na listu. Tyto prvky umožňují většinou vybrat jen jednu hodnotu z několika, které se vzájemně vylučují. Proto se tento ovládací prvek také někdy nazývá Radio Button jako vzpomínka na staré radiopřijímače, kdy zmáčknutí mechanického knoflíku pro jedno pásmo přinutilo knoflík ostatních pásem vyskočit.




**Obrázek 1.6:** Získání hodnoty podle stisknutého přepínače

V našem příkladu jsme na list umístili dva ovládací prvky typu Přepínač. Jména jsme změnili na optMale (označuje muže) a optFemale (označuje ženu). Úkolem je zobrazit zprávu o pohlaví osoby zapsané v buňce C2.

Úkol vyřeší tento kód:

*01-04 VBA Konstrukce.xlsm*
*modul MConstructions*

```
With ActiveSheet
    Select Case 1
    Case .Shapes("optMale").ControlFormat
        MsgBox .[C2] & " je muz"
```

```
    Case .Shapes("optFemale").ControlFormat
        MsgBox .[C2] & " je zena"
    End Select
End With
```

Pokud je přepínač v poloze „Zapnuto“, vrací vlastnost `Value` hodnotu 1. Po nalezení prvku, který tuto hodnotu splňuje, se vypíše zpráva a program se ukončí, nebo pokračuje dál příkazem na řádku následujícím za řádkem `End Select`.

## For ... Next

Smyčka `For...Next` je velmi běžným cyklem v procedurách VBA a jeho použití musíte bezpečně ovládat. Obsahuje vnitřní čítač, který je automaticky při každém průchodu zvyšován o zvolený krok, standardně o jedničku. Příkazy uvnitř těla cyklu jsou vykonávány tak dlouho, dokud čítač nepřekročí definovanou maximální hodnotu. Typické použití ve VBA pro Excel je procházení přes všechny prvky pole. Následující příklad načte do pole hodnoty v buňkách A2:A10, vynásobí je samy sebou (neboli vypočítá druhou mocninu) a výsledek zapíše do buněk B2:B10:

*01-04 VBA Konstrukce.xlsm*
*modul MConstructions*

```
Sub Mocnina()
    Dim y, i As Long
    y = Range("A2:A10")
    For i = LBound(y, 1) To UBound(y, 1)
        y(i, 1) = y(i, 1) ^ 2
    Next i
    Range("B2:B10") = y
End Sub
```

Pole ve VBA mohou mít obecně jakýkoli celočíselný spodní i horní index (horní hranice je v rámci omezení VBA, samozřejmě). Spodní hranice záleží na případné klauzuli Option Base v záhlaví modulu, ale i na způsobu načtení pole. Například pokud načteme do pole hodnoty z listu, bude pole vždy dvourozměrné a první prvek bude mít indexy 1,1.

Proto doporučujeme při iteraci přes všechny prvky pole *vždy* používat funkce `LBound()` a `UBound()` pro zjištění spodního i horního indexu, abyste první nebo poslední prvek nevynechali.

Další použití je procházení přes prvky, které *nejsou* součástí kolekce. Pro procházení prvků v kolekci je vhodnější cyklus `For...Each`, kterému se budeme věnovat v další části této lekce. Následující příklad není příliš užitečný, ale demonstruje procházení přes jednotlivé znaky v buňce. Liché znaky mění na velká písmena a sudá na malá. Navíc font lichých znaků změní na tučný. Pokud tedy buňka A1 obsahuje slovo Velbloud, výsledkem bude **V**e**L**b**L**o**U**d:

```
Sub Velbloud()
Dim i As Long
With Range("A1")
    For i = 1 To .Characters.Count
        With .Characters(i, 1)
            If i Mod 2 = 1 Then
                .Text = UCase(.Text)
                .Font.Bold = True
```

```
            Else
                .Text = LCase(.Text)
            End If
        End With
    Next i
End With
End Sub
```

Pokud v cyklu `For...Next` vynecháme klíčové slovo `Step`, bude se čítač zvyšovat o jedničku. Můžeme však definovat krok jiný, například pokud chceme ve sloupcích změnit barvu každého druhého řádku, počínaje druhým řádkem listu (prvním řádkem dat) a konče řádkem posledním (předem neznámým):

```
Sub AlternateRows()
Dim lLastRow As Long, i As Long
lLastRow = Cells(Rows.Count, "A").End(xlUp).Row
For i = 2 To lLastRow Step 2
    With Cells(i, "A").Resize(, 2)
        .Interior.Color = RGB(127, 127, 127)
        .Font.Color = vbWhite
    End With
Next i
End Sub
```

Čítač se nemusí nutně zvyšovat inkrementálně. Můžeme definovat i krok záporný. Následující příklad, který mění pořadí znaků v buňce A1, rovněž není velmi praktický:

```
Sub Shenanigan()
Dim i As Long
Dim tmp As String
With Range("A1")
    For i = Len(.Value) To 1 Step -1
        tmp = tmp & Mid$(.Value, i, 1)
    Next i
    .Value = tmp
End With
End Sub
```

Častějším případem, kdy je nutné použít zpětnou smyčku, je mazání vybraných řádků. Algoritmus nemůže postupovat dopředu, protože by při vymazání řádku byl další řádek přeskočen. Pokud by čítač dosáhl hodnoty 3 a řádek splňoval podmínku, řádek by byl vymazán a původní čtvrtý řádek by postoupil na místo třetí. Čítač však mezitím nabude hodnoty čtyři a testuje původně pátý řádek. Čtvrtý řádek bude při testu přeskočen.

Původní stav

| | A | B |
|---|---|---|
| 1 | Hodnota | Mocnina |
| 2 | 11 | 121 |
| 3 | 12 | 144 |
| 4 | 13 | 169 |
| 5 | 14 | 196 |
| 6 | 15 | 225 |
| 7 | 16 | 256 |
| 8 | 17 | 289 |
| 9 | 18 | 324 |
| 10 | 19 | 361 |

Výsledný stav

| | A | B |
|---|---|---|
| 1 | Hodnota | Mocnina |
| 2 | 11 | 121 |
| 3 | 12 | 144 |
| 4 | 13 | 169 |
| 5 | 14 | 196 |
| 6 | 15 | 225 |
| 7 | 16 | 256 |
| 8 | 17 | 289 |
| 9 | 18 | 324 |
| 10 | 19 | 361 |

**Obrázek 1.7:** Střídavé vybarvení řádků

Přestože pro mazání vybraných řádků existují účinnější techniky, které podrobně představíme ve čtvrté lekci, pro malé objemy dat je možno použít následující kód:

```
Sub DeleteEmptyRows()
Dim lLastRow As Long, i As Long
lLastRow = Cells.SpecialCells(xlLastCell).Row
For i = lLastRow To 2 Step -1
    If Application.CountA(Rows(i)) = 0 Then
        Rows(i).Delete
    End If
Next i
End Sub
```

Kód využívá vestavěný vzorec POČET2() (v angličtině COUNTA), který sečte neprázdné buňky v předané oblasti.

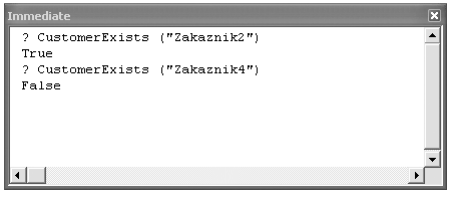
**Tip**

Přestože to není zcela v souladu se zásadami strukturovaného programování, někdy je vhodné cyklus předčasně ukončit, například proto, že kód narazil na buňku s chybovou hodnotou, nebo aby se neprováděly nadbytečné příkazy. Bývá dobrým zvykem, aby cyklus obsahoval pouze jeden příkaz předčasného opuštění cyklu `Exit <typ cyklu>`.

Následující funkce vrátí `True`, pokud se řetězec předaný do proměnné Zakaznik vyskytuje v poli „Zakaznici“:

*01-04 VBA Konstrukce.xlsm*
*modul MFunctions*

```
Function CustomerExists(Zakaznik As String) As Boolean
Dim Zakaznici
Dim i As Long
Zakaznici = Array("Zakaznik1", "Zakaznik2", "Zakaznik3")
For i = LBound(Zakaznici) To UBound(Zakaznici)
    If Zakaznici(i) = Zakaznik Then
        'zákazník nalezen, není nutné hledat dále
        CustomerExists = True
        Exit For
    End If
Next i
End Function
```




**Obrázek 1.8:** Použití okna Immediate pro testování výsledku vlastní funkce

Funkci můžete ověřit v okně Immediate:

Po napsání řádku začínajícího otazníkem (ekvivalent příkazu `Debug.Print`) stiskněte Enter.

**Tip**

Patří k dobrým programátorským praktikám nepoužívat proměnnou, obsahující čítač cyklu `For...Next` k žádnému jinému účelu, než právě k řízení cyklu. Přestože se chová jako kterákoli jiná proměnná a je možné jí přiřadit novou hodnotu dokonce i uvnitř cyklu, tento postup zásadně odporuje principům strukturovaného programování a váš kód se stává prakticky nečitelným i pro vás jako autora. Navíc je nutné si uvědomit, že čítač mění hodnotu při každém průchodu přes klíčové slovo `Next` a novou hodnotu porovnává s horní hranicí. Pokud je hodnota čítače vyšší (nebo nižší v případě záporného kroku), cyklus je ukončen a program pokračuje dalším řádkem pod klíčovým slovem `Next`. Čítač tedy po opuštění cyklu není roven horní hranici!

## For Each ... Next

Cyklus `For Each...Next` je v procedurách pro Excel zřejmě ještě běžnější než cyklus `For...Next`. Je tomu tak proto, že v objektovém modelu Excel se většina běžných objektů vyskytuje v *kolekcích* a cyklus `For Each...Next` je ušit na míru právě procházení prvků v kolekci.

Nejenže je použití cyklu `For Each...Next` pro kolekce bezpečnější, ale je i o něco rychlejší, jak se můžeme přesvědčit například na testu těchto nic zvláštního nedělajících procedur:

*01-04 VBA Konstrukce.xlsm*
*modul MConstructions*

```
Sub TestForEach()

Const NR_LOOPS = 1000
Dim lCounter As Long

Dim ws As Worksheet
For lCounter = 1 To NR_LOOPS
    For Each ws In ThisWorkbook.Sheets
        Debug.Print ws.Name
    Next ws
Next lCounter

End Sub


Sub TestForNext()

Const NR_LOOPS = 1000
Dim lCounter As Long


Dim i As Long
For lCounter = 1 To NR_LOOPS
    For i = 1 To ThisWorkbook.Sheets.Count
        Debug.Print Sheets(i).Name
    Next i
Next lCounter

End Sub
```

Procedury vypíší tisíckrát jména všech listů do okna Immediate. Přestože časový rozdíl není v absolutní hodnotě velký, druhá procedura trvá zhruba o 25 % déle.

Okno Immediate lze vymazat umístěním kurzoru do něj (aby okno získalo fokus), stisknutím Ctrl+A (vybere veškerý text) a následným stisknutím klávesy Delete. Objektový model VBA projektu bohužel neumožňuje přímou manipulaci s obsahem tohoto okna, ale uvedenou kombinaci kláves můžeme nasimulovat pomocí funkce `SendKeys`:

```
Sub ImmediateWindowClear()
    Application.VBE.Windows.Item("Immediate").SetFocus
    Application.SendKeys "^a"
    Application.SendKeys "{Del}"
End Sub
```

Následující příklad prochází kolekci souborů ve zvoleném adresáři a vypíše jejich jména a velikost v kilobytech:

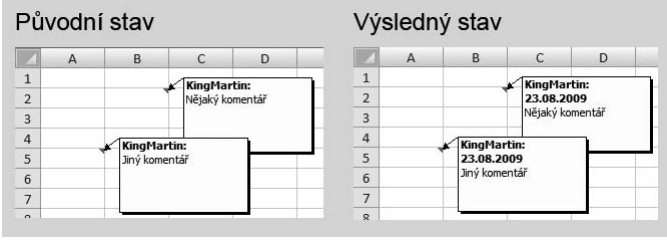
*01-04 VBA Konstrukce.xlsm*
*modul MConstructions*

```
Sub FileList()
Const FOLDER As String = "C:\"
Dim fso As Object, fld As Object, f As Object
Set fso = CreateObject("Scripting.FileSystemObject")
Set fld = fso.GetFolder(FOLDER)
For Each f In fld.Files
    Debug.Print f.Name & vbTab & Format$(f.Size \ 1024, "#,##0")
Next
Set fso = Nothing
End Sub
```

Kód využívá knihovnu VB skriptu, která leží mimo objektový model Excelu. Více se o ní dozvíte v sedmé lekci, kde se budeme zabývat hromadným zpracováním dat z více sešitů.

Jako poslední příklad uvedeme proceduru, která za jméno autora komentáře vloží dnešní datum:

```
Sub Comments()
    Dim cmt As Comment
    For Each cmt In ActiveSheet.Comments
    cmt.Text vbLf & Date, Len(cmt.Author) + 2, False
    Next cmt
End Sub
```




**Obrázek 1.9:** Vložení aktuálního data do všech komentářů

Obecným pravidlem je používat cyklus `For...Next` pro procházení prvky pole a cyklus `For Each... Next` pro procházení prvků kolekce. Pro prvky v kolekcích použijte cyklus `For...Next` tehdy, pokud potřebujete jiný krok než jedna, nebo krok záporný.

Podobně jako cyklus `For...Next`, i cyklus `For Each...Next` lze předčasně ukončit. Následující procedura prochází buňky v oblasti A1:A10 aktivního listu. Pokud buňka obsahuje hodnotu, interpretovatelnou jako číslo, vynásobí se dvěma. Pokud hodnota číselná není, zobrazí se varovné hlášení, smyčka se předčasně ukončí a kód pokračuje řádkem `End Sub`.

```
Sub ExitForEach()
Dim cl As Range
For Each cl In Range("A1:A10")
    If IsNumeric(cl) Then
        cl = cl * 2
    Else
        MsgBox "Bunka " & cl.Address & " neobsahuje číslo!"
        Exit For
    End If
Next
End Sub
```

## Do ... Loop

Cyklus `Do...Loop` je ve VBA pro Excel používána méně než cykly, které jsme popsali v předešlých částech lekce. Je tomu tak proto, že v Excelu a vůbec ve všech aplikacích Microsoft Office se většina objektů nachází v kolekcích o konkrétním počtu (vlastnost `Count`), případně lze horní hranici (například poslední řádek) snadno nalézt.

Konstrukce Do…Loop se používá v případě, kdy je počet průchodů předem neznámý. Proto je její použití obvyklé zejména při manipulaci s objekty, které se nacházejí mimo objektový model Microsoft Office.

V Excelu se cyklus Do…Loop vyskytuje většinou ve spojení s metodou `Find` objektu `Range`, kdy je počet nalezených buněk předem neznámý. Následující příklad předpokládá, že na aktivním listu jsou vzorce odkazující na list „List1“, typu „=List1!A1“. Úkolem je vzorce „přepojit“ na list „List2“, čili původní vzorec bude změněn na „=List2!A1“. Toto může být velmi důležitá technika, která vám může ušetřit ohromné množství času při konsolidaci dat.

*01-04 VBA Konstrukce.xlsm*
*modul MConstructions*

```
Sub PrepojList()
Const OldSheet As String = "List1"
Const NewSheet As String = "List2"

Dim cl As Range
With ActiveSheet
    Set cl = .Cells.Find(OldSheet, LookIn:=xlFormulas, LookAt:=xlPart)
    If Not cl Is Nothing Then
    'alespoň jedna buňka nalezena
        Do
            cl.Formula = Replace$(cl.Formula, OldSheet, NewSheet)
```

```
            Set cl = .Cells.FindNext
        Loop Until cl Is Nothing
    End If
End With

End Sub
```

Procedura nejprve zavolá poprvé metodu `Find` a pokusí se nalézt buňku, která obsahuje hledaný výraz. Všimněte si, že je metoda doplněna o parametry určující, že má hledat ve vzorcích, a ne ve vypočítaných hodnotách, a rovněž že se má vyhledávat část vzorce, a ne obsah celé buňky.

Pokud je alespoň jedna buňka nalezena, procedura vstoupí do smyčky `Do...Loop`, a protože je podmínka opuštění cyklu až za výrazem `Loop`, příkazy se vykonají alespoň jednou. Funkce Replace$() provede nahrazení starého řetězce řetězcem novým a poté se zavolá metoda `FindNext`, která se pokusí nalézt následující buňku. Podmínka za klíčovým slovem `Loop` zkontroluje, zda byla další buňka nalezena. Pokud ano, program pokračuje řádkem pod klíčovým slovem `Do`. Výraz

```
Loop Until cl Is Nothing
```

volně přeloženo znamená „pokračuj ve smyčce, *dokud* není proměnná cl prázdná, jinak řečeno, další buňka nebyla nalezena".

Jinou variantou na tutéž podmínku by bylo použití podmínky `While`:

```
Loop While Not cl Is Nothing
```

Což znamená, „pokračuj ve smyčce, *pokud* je proměnná cl neprázdná". Čeština má pro oba případy podobnou gramatiku, ale angličtina mezi oběma případy zřetelně odlišuje. V každém případě je jedno, jestli použijete variantu `Until` nebo variantu `While`. Používejte tu, jejíž logika je vám bližší.

Zajímavé je, že nápověda Microsoft Excel uvádí tento příklad použití metody `Find()`:

```
With Worksheets(1).Range("A1:A500")
    Set c = .Find(2, LookIn:=xlValues)
    If Not c Is Nothing Then
        firstaddress = c.Address
        Do
            c.Value = 5
            Set c = .FindNext(c)
        Loop While Not c Is Nothing And c.Address <> firstaddress
    End If
End With
```

Kód má za úkol vyhledat buňky, obsahující hodnotu 2, a nahradit ji hodnotou 5. Pokud definovaná oblast žádnou hodnotu 2 neobsahuje, vše proběhne normálně a kód dle očekávání nevykoná nic.

Pokud ovšem alespoň jedna taková buňka existuje, kód vstoupí do smyčky a pokračuje v hledání a nahrazování tak dlouho, až byly všechny buňky nahrazeny a výraz

```
Set c = .FindNext(c)
```

vrátí objekt `Nothing`, neboli „nic". Takový objekt ovšem nemůže mít vlastnost `Address`, a protože se u složených podmínek vždy vyhodnocují všechny jejich části, výraz

```
c.Address <> firstaddress
```