

## OBSAH

<b>1.1</b>	<b>LEKCE 0 – ÚVOD DO VBA.....</b>	<b>4</b>
1.1.1	KARTA VÝVOJÁŘE .....	4
1.1.2	VISUAL BASIC EDITOR .....	6
1.1.3	ROZLOŽENÍ PROJEKTU .....	7
1.1.4	ÚKOLY K PROCVIČENÍ .....	7
<b>1.2</b>	<b>LEKCE 1 – PROCEDURY .....</b>	<b>8</b>
1.2.1	MAKRO .....	8
1.2.2	ZÁZNAM MAKRA .....	9
1.2.3	UKLÁDÁNÍ SEŠITU .....	10
1.2.4	FUNKCE.....	11
1.2.5	ÚKOLY K PROCVIČENÍ .....	12
<b>1.3</b>	<b>LEKCE 2 – PROMĚNNÉ, DATOVÉ TYPY A VÝPOČTY.....</b>	<b>13</b>
1.3.1	PROMĚNNÉ .....	13
1.3.2	DATOVÉ TYPY .....	13
1.3.3	OPERÁTORY PŘI PRÁCI S PROMĚNNÝMI .....	15
1.3.4	PŘEVODY MEZI DATOVÝMI TYPY .....	16
1.3.5	ÚKOLY K PROCVIČENÍ .....	17
<b>1.4</b>	<b>LEKCE 3 – VĚTVENÍ A PODMÍNĚNÉ PŘÍKAZY.....</b>	<b>18</b>
1.4.1	PŘÍKAZY IF, ELSE, ELSE IF .....	18
1.4.2	POROVNÁVACÍ OPERÁTORY.....	19
1.4.3	LOGICKÉ OPERÁTORY .....	20
1.4.4	PŘÍKAZ SELECT CASE.....	20
1.4.5	PODMÍNĚNÉ FUNKCE .....	21
1.4.6	ÚKOLY K PROCVIČENÍ .....	23
<b>1.5</b>	<b>LEKCE 4 – KALENDÁŘNÍ A ČASOVÉ HODNOTY .....</b>	<b>24</b>
1.5.1	KALENDÁŘNÍ HODNOTY .....	24
1.5.2	ČASOVÉ HODNOTY .....	24
1.5.3	PRÁCE S KALENDÁŘEM A ČASEM.....	25
1.5.4	ÚKOLY K PROCVIČENÍ .....	28
<b>1.6</b>	<b>LEKCE 5 – OPAKOVÁNÍ, CYKLY A POLE.....</b>	<b>29</b>
1.6.1	CYKLUS WHILE .....	29
1.6.2	CYKLUS FOR .....	29
1.6.3	CYKLUS DO.....	30
1.6.4	VNOŘENÉ CYKLY .....	31
1.6.5	REKURZIVNÍ FUNKCE.....	32
1.6.6	POLE A MATICE .....	32
1.6.7	CYKLUS FOR EACH .....	34
1.6.8	ÚKOLY K PROCVIČENÍ .....	35
<b>1.7</b>	<b>LEKCE 6 – PRÁCE S OBLASTMI VÝBĚRU.....</b>	<b>36</b>

1.7.1	AKTIVNÍ OZNAČENÍ .....	36
1.7.2	PASIVNÍ PŘÍSTUP .....	37
1.7.3	ŘÁDKY A SLOUPCE SEŠITU .....	39
1.7.4	ÚKOLY K PROCVIČENÍ .....	40
<b>1.8</b>	<b>LEKCE 7 – STYLOVÁNÍ TEXTU A BUNĚK .....</b>	<b>41</b>
1.8.1	PŘÍSTUP POMOCÍ PŘÍKAZU WITH .....	41
1.8.2	VÝŠKA A ŠÍŘKA BUŇKY .....	41
1.8.3	OHRANIČENÍ BUŇKY .....	42
1.8.4	SLUČOVÁNÍ BUNĚK .....	43
1.8.5	BARVA A TEXT BUŇKY .....	43
1.8.6	FORMÁTOVÁNÍ BUNĚK .....	45
1.8.7	UZAMKNUTÍ BUŇKY .....	46
1.8.8	ÚKOLY K PROCVIČENÍ .....	46
<b>1.9</b>	<b>LEKCE 8 – PRÁCE SE SEŠITEM.....</b>	<b>47</b>
1.9.1	VYBRÁNÍ KONKRÉTNÍHO LISTU SEŠITU .....	47
1.9.2	ZAMYKÁNÍ A ODEMYKÁNÍ LISTŮ .....	47
1.9.3	ZOBRAZENÍ LISTŮ .....	48
1.9.4	PŘIDÁNÍ A ODEBRÁNÍ LISTŮ .....	49
1.9.5	KOPÍROVÁNÍ A PŘESUNOVÁNÍ LISTŮ .....	49
1.9.6	UDÁLOSTI V LISTU .....	50
1.9.7	ÚKOLY K PROCVIČENÍ .....	51
<b>1.10</b>	<b>LEKCE 9 – PRÁCE S GRAFY .....</b>	<b>52</b>
1.10.1	VYTVOŘENÍ GRAFU .....	52
1.10.2	ZOBRAZENÍ GRAFU .....	53
1.10.3	PRVKY GRAFU .....	53
1.10.4	LIST GRAFU .....	56
1.10.5	ÚKOLY K PROCVIČENÍ .....	57
<b>1.11</b>	<b>LEKCE 10 – TŘÍDY A OBJEKTY .....</b>	<b>58</b>
1.11.1	PRÁCE S OBJEKTY.....	<b>CHYBA! ZÁLOŽKA NENÍ DEFINOVÁNA.</b>
1.11.2	VYTVOŘENÍ OBJEKTU .....	59
1.11.3	UDÁLOSTI OBJEKTU.....	<b>CHYBA! ZÁLOŽKA NENÍ DEFINOVÁNA.</b>
1.11.4	KOLEKCE.....	60
1.11.5	VLASTNOSTI A METODY .....	60
1.11.6	PŘÍKAZ LET .....	<b>CHYBA! ZÁLOŽKA NENÍ DEFINOVÁNA.</b>
1.11.7	PŘÍKAZ SET .....	59
1.11.8	ÚKOLY K PROCVIČENÍ .....	<b>CHYBA! ZÁLOŽKA NENÍ DEFINOVÁNA.</b>
<b>1.12</b>	<b>LEKCE 11 – PRÁCE S KONTINGENČNÍMI TABULKAMI .....</b>	<b>CHYBA!</b>
	<b>ZÁLOŽKA NENÍ DEFINOVÁNA.</b>	
1.12.1	ÚKOLY K PROCVIČENÍ .....	<b>CHYBA! ZÁLOŽKA NENÍ DEFINOVÁNA.</b>
<b>1.13</b>	<b>LEKCE 12 – PRÁCE S UŽIVATELSKÝMI DIALOGY A FORMULÁŘI .....</b>	<b>62</b>

1.13.1 PRVEK MSGBOX .....	62
1.13.2 PRVEK INPUTBOX .....	62
1.13.3 OVLÁDACÍ PRVKY .....	<b>CHYBA! ZÁLOŽKA NENÍ DEFINOVÁNA.</b>
1.13.4 DESIGN FORMULÁŘ .....	<b>CHYBA! ZÁLOŽKA NENÍ DEFINOVÁNA.</b>
1.13.5 ÚKOLY K PROCVIČENÍ .....	<b>CHYBA! ZÁLOŽKA NENÍ DEFINOVÁNA.</b>
<b>1.14 LEKCE 13 – PRÁCE S TEXTOVÝMI SOUBORY .....</b>	<b>64</b>
1.14.1 ZJIŠTĚNÍ SOUBORU NEBO ADRESÁŘE .....	<b>CHYBA! ZÁLOŽKA NENÍ DEFINOVÁNA.</b>
1.14.2 OTEVŘENÍ SOUBORU.....	<b>CHYBA! ZÁLOŽKA NENÍ DEFINOVÁNA.</b>
1.14.3 SMAZÁNÍ SOUBORU .....	<b>CHYBA! ZÁLOŽKA NENÍ DEFINOVÁNA.</b>
1.14.4 EXPORT SOUBORU .....	64
1.14.5 ULOŽIT SOUBOR .....	65
1.14.6 SOUBOROVÝ SYSTÉM .....	<b>CHYBA! ZÁLOŽKA NENÍ DEFINOVÁNA.</b>
1.14.7 ÚKOLY K PROCVIČENÍ .....	<b>CHYBA! ZÁLOŽKA NENÍ DEFINOVÁNA.</b>

## 1.1 Lekce 0 – Úvod do VBA

Visual Basic pro Aplikace (VBA) je programovací jazyk, který umožňuje rozšířit funkce aplikací v balíku kancelářského softwaru Microsoft 365 (dříve označován jako balík Microsoft Office nebo Office 365). Mezi tyto aplikace patří například Excel, Access, PowerPoint, Word a další. Jedná se o objektově orientovaný programovací jazyk, který je zároveň v kódu řízen různými událostmi.

Microsoft s pomocí VBA umožňuje uživatelům zautomatizovat jakoukoliv operaci, kterou lze provést pomocí myši nebo klávesnice. Cílem je usnadnit práci, kterou uživatelé v těchto programech opakují neustále dokola. VBA dovoluje i propojení mezi jednotlivými kancelářskými aplikacemi, například lze převést kontakty z aplikace Outlook přímo do tabulky v aplikaci Excel. Mezi další cíle jazyka patří pro vývojáře jednoduše pochopitelná syntaxe (vycházející z anglického jazyka) a udržení dlouhodobé podpory pro aplikace vytvořené v tomto jazyce.

Pod pojmem Visual Basic se v rámci historie objevila řada několika programovacích jazyků. Při programování s VBA je nutné rozlišovat tyto jazyky, protože některé z nich mohou mít velmi podobnou syntaxi, každopádně nemusí zaručovat obdobnou funkci. Jedná se například o Classic Visual Basic (VB 6.0), .NET Visual Basic (VB.NET) či Visual Basic Script (VBScript).

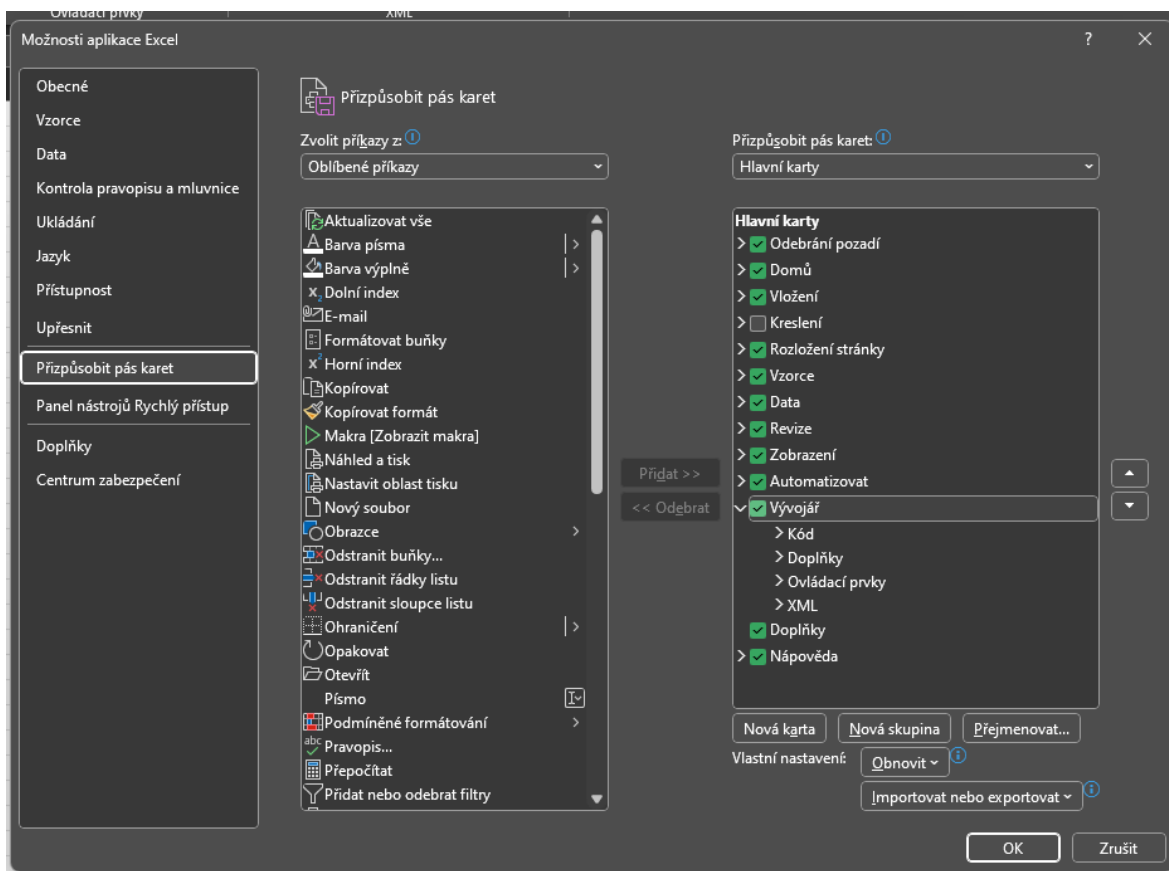
I když VBA není pouze pro platformu operačního systému Windows, součástí jako uživatelské formuláře a další jsou na platformě Mac výrazně omezeny. Microsoft představil tzv. Office Scripts, které jsou založené na programovacím jazyce JavaScript. Přestože tento jazyk neumožňuje stejné množství funkcí jako VBA, nahrazuje tyto funkce možností multiplatformního vývoje např. i pro prohlížečovou verzi programu Excel. Rozvoj v této oblasti by se mohl jevit jako jedna z možností budoucí automatizace Office.

Programátorské prostředí pro vývoj ve VBA je implementováno přímo ve většině aplikací balíčku 365 a stačí jej v aplikaci pouze spustit.

### 1.1.1 Karta vývojáře

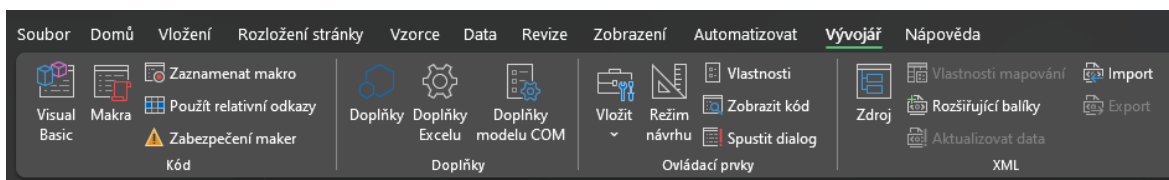
Pro spuštění vývojářského prostředí je nejdříve potřeba zobrazit kartu Vývojář v aplikaci Excel. Ta při prvním spuštění programu není v nabídce karet zobrazena. Pro zobrazení karty stačí kliknout na *Soubor* a ve spodní části levého menu vybrat tlačítko *Možnosti*. Otevře se

nové okno *Možnosti aplikace*. Z nabídky postranního menu je potřebné zvolit *Prizpůsobit pás karet* a následně vybrat volbu *Vývojář*. Nové nastavení se potvrzuje tlačítkem *Ok*.



Obrázek 1 – Okno Možnosti aplikace Excel s výběrem zobrazení karty vývojáře

Karta *Vývojář* se skládá z více skupin, kde každá představuje jinou část možného vývoje softwaru pro aplikaci Excel. Vývoj s jazykem VBA se převážně vyskytuje ve skupině *Kód* a ve skupině *Ovládací prvky*, která obsahuje možnost vkládat ovládací prvky přímo do sešitu Excelu.

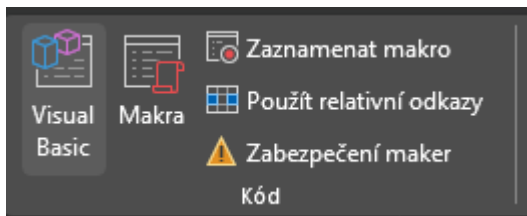


Obrázek 2 – Karta Vývojář

Skupina *Doplňky* se zabývá převážně vývojem externích doplňků pro aplikaci Excel a skupina *XML* se zabývá prací s daty ve stejnojmenném formátu.

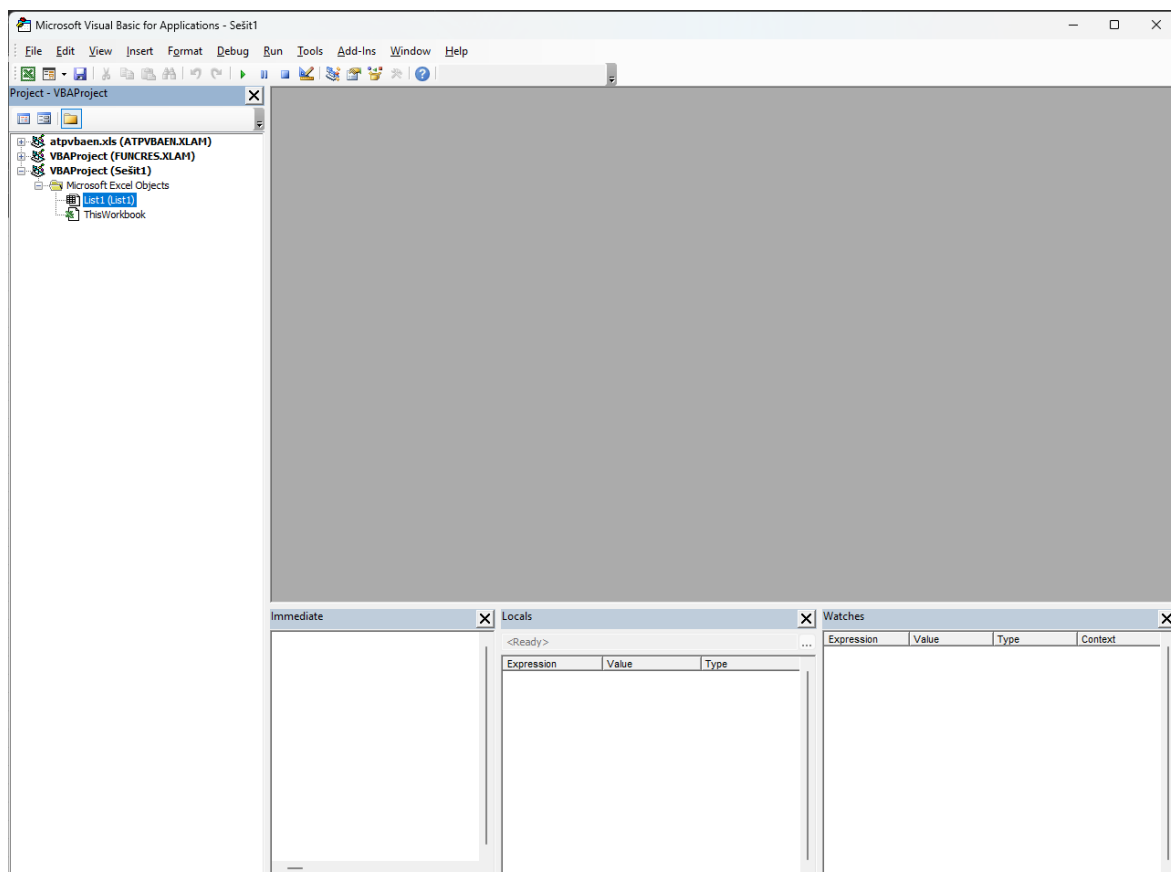
### 1.1.2 Visual Basic Editor

Pro spuštění vývojářského prostředí editoru stačí na kartě *Vývojář* zvolit ve skupině *Kód* tlačítko *Visual Basic*.



Obrázek 3 – Tlačítko *Visual Basic* pro spuštění VBE

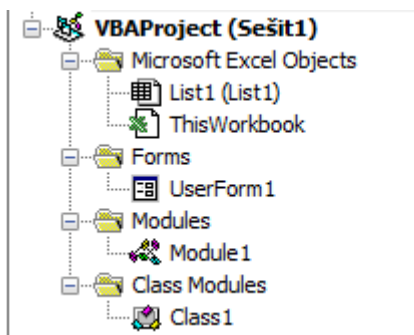
Visual Basic Editor (VBE) je integrované vývojové prostředí pro tvorbu maker. To znamená, že uživatelům umožňuje vytvářet, upravovat a ladit kód VBA. Editor zahrnuje barevné zvýrazňování jazyka, editor pro uživatelské rozhraní a možnost práce se strukturou projektu. Dovednosti VBE budou postupně rozvedeny v rámci tohoto průvodce.



Obrázek 4 – Visual Basic Editor

### 1.1.3 Rozložení projektu

Projekt VBA je skoro vždy vázán na konkrétní sešit Excelu a makra se ukládají přímo do tohoto sešitu v binární podobě. Základní strukturu projektu lze vidět v levé části po otevření VBE, kde se nachází všechny soubory celého projektu, popř. dalších projektů.

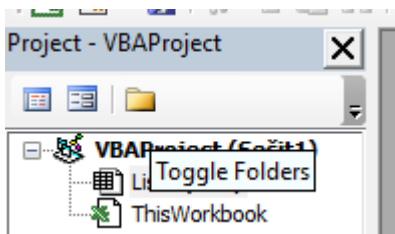


Obrázek 5 – Struktura projektu VBA

V postranním menu, které zobrazuje strukturu projektu, jsou zobrazeny složky *Microsoft Excel Objects*, *Forms*, *Modules* a *Class Modules*. Tyto složky představují rozložení jednotlivých uživatelských kódů dle použití a pomáhají s přehlednou organizací.

Složka *Microsoft Excel Objects* převážně obsahuje objekt právě otevřeného sešitu a pak jednotlivé objekty listů sešitu. Dvoj kliknutí na jeden z objektů zobrazí nové podokno ve VBE, kde lze psát kód k příslušnému objektu. Složka je výchozí pro Excel VBA projekt a jako jediná v novém projektu zobrazená.

Ostatní složky se zobrazí až po přidání nových objektů do projektu dle jejich požadovaného typu. Pokud projekt není ve stromové struktuře, ale zobrazuje všechny soubory kódu najednou, stačí stisknout tlačítko *Toggle Folders* s ikonou složky na struktuře projektu.



Obrázek 6 – Tlačítko přepínání typu zobrazení struktury projektu

### 1.1.4 Úkoly k procvičení

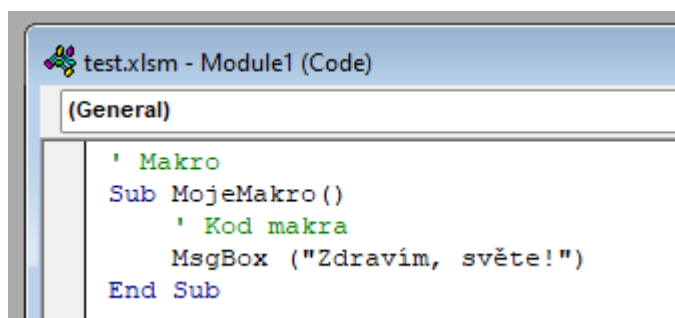
- 1) Spusťte VBE a vytvořte nový VBA projekt v sešitě.
- 2) Přejmenujte projekt na *MujPrvniVBAProjekt* a přidejte vlastní popis k projektu.
- 3) Uložte sešit a projekt VBA ve formátu *.xslm*, aby uložený sešit obsahoval makra.

## 1.2 Lekce 1 – Procedury

Při používání programu Excel může jeho uživatel rozšířit funkčnost svého sešitu nebo listu pomocí dvou hlavních procedur, do kterých může zapsat nebo nahrát kód. Jedná se o procedury makro a funkce. Rozdíly mezi nimi jsou vysvětleny v následujících podkapitolách.

### 1.2.1 Makro

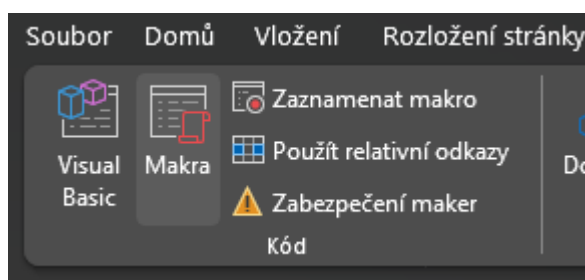
Makrem je označována každá procedura, která je definována v kódu pomocí klíčového slova *Sub* (vychází z anglického názvu *subroutine* – v češtině subrutina, podprogram nebo makro). Makro představuje sadu instrukcí, které se počítač při spuštění daného makra pokusí splnit. K úspěšnému dosažení konce makra, označeného pomocí klíčového slova *End Sub*, musí být všechny instrukce správně syntakticky zapsány, jinak dojde k chybě při běhu programu.



```
test.xlsm - Module1 (Code)
(General)
' Makro
Sub MojeMakro ()
    ' Kod makra
    MsgBox ("Zdravím, světe!")
End Sub
```

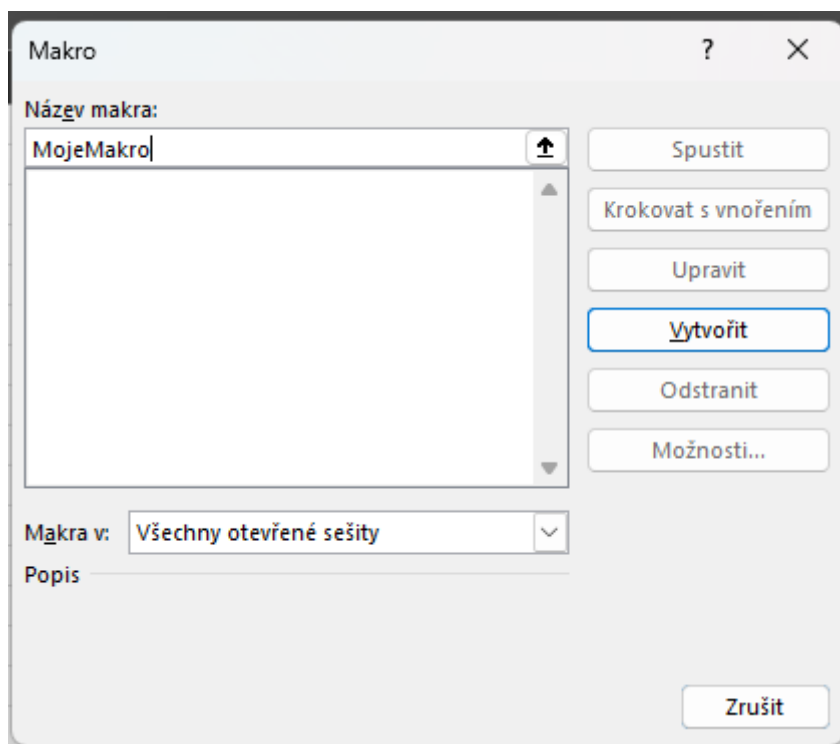
Obrázek 7 – Syntaktická ukázka kódu makra

Vytvořit makro lze několika způsoby. Jedním ze způsobů je pomocí karty *Vývojář* a tlačítka *Makra* ve skupině *Kód*.



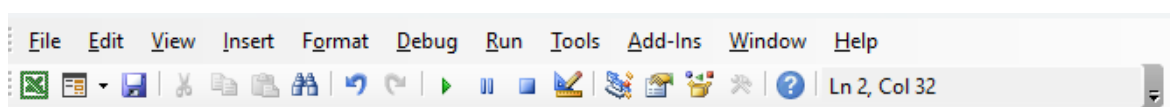
Obrázek 8 – Tlačítko Makra pro otevření podokna s organizací maker

Po kliknutí na tlačítko se zobrazí nové menší podokno, které zobrazuje všechna doposud vytvořená makra. Uživatel musí zapsat název svého makra bez mezer, čísel a diakritiky. Pokud název již neexistuje, mělo by se zpřístupnit tlačítko *Vytvořit*. Stiskem tohoto tlačítka se otevře Visual Basic Editor s vytvořenou kostrou makra.



Obrázek 9 – Podokno s organizací maker v sešitech

Po doplnění kódu, který by měla procedura splnit, lze makro spustit ve stejném podokně, akorát s označením daného makra a s pomocí tlačítka *Spustit*. Makro lze přímo spustit i z editoru pomocí zelené šipky v liště pod nabídkou menu, v menu pomocí *Run Sub/User Form* nebo pomocí klávesy *F5*. Pomocí lišty lze makro i pozastavit či kompletně zastavit.

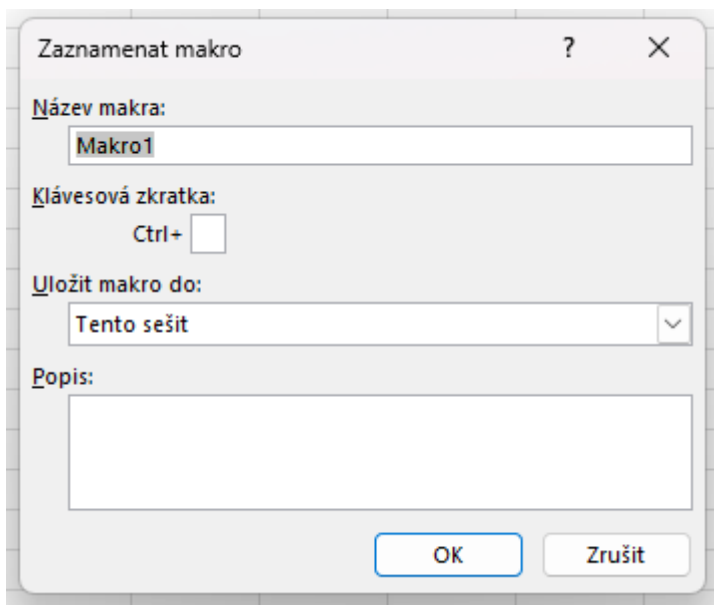


Obrázek 10 – Lišta s často používanými funkcemi, i pro ladění kódu

### 1.2.2 Záznam makra

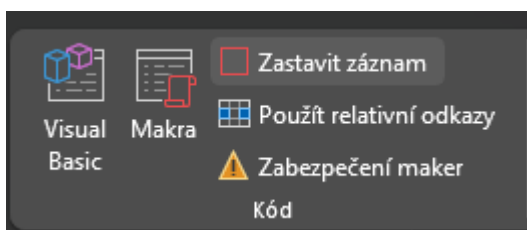
Pro urychlení a ulehčení vývojářům psaní kódu VBA existuje možnost zaznamenat kód přímo v aplikaci Excel. To znamená, že pokud vývojář stiskne nahrávání, veškeré jeho akce, které v programu Excel provede se zapíší do kódu VBA, dokud nahrávání neukončí.

Nahrávání makra se spouští na kartě *Vývojář* pomocí tlačítka *Zaznamenat makro* ve skupině *Kód*. Po kliknutí na tlačítko se zobrazí nové podokno, které vyžaduje zadat název nově zaznamenávaného makra, klávesovou zkratku pro rychlé spuštění, informaci o místě uložení makra a popis makra.



Obrázek 11 – Podokno se záznamem makra

Po potvrzení údajů pomocí tlačítka *OK* se spustí nahrávání makra. Každá změna v sešitě se nyní zaznamená do VBA jako instrukce v subrutině. Záznam se zastaví pomocí tlačítka *Zastavit záznam* na kartě *Vývojář*.



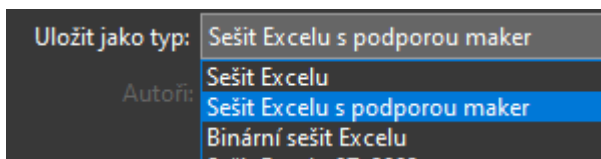
Obrázek 12 – Tlačítko zastavení záznamu makra

Po dokončení nahrávání lze ve VBE nahrané makro upravit dle potřeb vývojáře. Makro lze najít v podokně maker a upravit tlačítkem *Upravit*, které otevře VBE na místě, kde se zaznamenané makro nachází.

V rámci nahrávání makra může uživatel vybrat možnost *Použít relativní odkazy*. Při vybrání této možnosti se makro bude nahrávat s odsazením od aktuálně vybrané buňky, místo toho, aby si zapisovalo vybrané souřadnice buňky.

### 1.2.3 Ukládání sešitu

Ukládání souboru do normálního excel dokumentu *Sešit Excelu* (koncovka .xlsx) způsobí, že vytvořená makra se v souboru neuloží. K uložení dokumentu včetně maker je potřeba soubor uložit jako typ *Sešit Excelu s podporou maker* (.xlsm).



Obrázek 13 – Ukázka možnosti uložení sešitu s podporou maker

#### 1.2.4 Funkce

Procedury typu funkce na rozdíl od maker vracejí tzv. návratovou hodnotu, jinými slovy nějaký finální výsledek, ke kterému během svého chodu dospěli. Tento výsledek může být výsledek výpočtu, hláška, datum atd.

Funkci lze zavolat v kódu a její výsledek uložit do proměnné nebo ji lze použít přímo v sešitě Excel. Návratová hodnota se v takovém případě objeví v buňce, ze které funkce byla volána.

Před použitím funkce je potřebné prvně danou funkci definovat. Definice funkce probíhá klíčovým slovem *Function*, které je následované názvem. Název funkce musí být v rámci dokumentu jedinečný, aby nedošlo ke kolizi funkcí. Konec funkce je obdobně jako u makra označen klíčovými slovy *End Function*. Před ukončením funkce se v určitém momentu měla uložit návratová hodnota. Tento proces proběhne přiřazením dané hodnoty názvu funkce pomocí operátoru rovná se. V případě nepřirazení funkce nebude vracet žádnou hodnotu.

```
test.xlsm - Module1 (Code)
(General)
' Funkce
Function MojeFunkce()
'Kod funkce
MsgBox ("Zdravim, z funkce!")
'Přiřazení návratové hodnoty funkce
MojeFunkce = "Pozdrav"
End Function
```

Obrázek 14 - Syntaktická ukázka kódu funkce

V rámci kódu je možné funkci opustit příkazem *Exit Function*, který ukončí kód v místě použití. Klíčové slovo *End* tedy označuje pouze konec definice subrutiny, zatímco *Exit* ji opouští s tím, že nebude pokračovat ve vykonávání dalšího kódu.

Předtím zmíněné definice funkce by neustále vracely stejnou hodnotu a jejich užitek by byl poměrně malý. Z tohoto důvodu lze funkcím přiřadit vstupní parametry, které ovlivňují dosažený výsledek funkce. Vstupní parametry funkce, lze nadefinovat ve funkci do závorek po uvedení názvu funkce. Množství vstupních parametrů není nijak omezeno.

```
' Funkce výpočet obsahu čtverce
Function ObsahCtverce(stranaA)
    'Kód funkce
    ObsahCtverce = stranaA * stranaA
End Function
```

Obrázek 15 – Ukázka funkce *ObsahCtverce* s parametrem *stranaA*

Pro příklad řekněme, že existuje funkce pojmenovaná *ObsahCtverce*. Jelikož nechceme po každé upravovat funkci pro výpočet obsahu čtverce s tím, že v ní změním číslo s velikostí strany, musí tato funkce obsahovat vstupní parametr. Ten by se měl měnit dle potřeb případu použití funkce. Pro výpočet obsahu čtverce je potřeba znát délku jeho strany, kterou lze označit například *stranaA*. Z toho vyplývá, že *stranaA* bude vstupním parametrem pro tuto funkci. Návrátová hodnota funkce bude obsahovat výpočet za použití tohoto vstupního parametru, v tomto případě  $stranaA * stranaA$ .

Některé parametry funkcí mohou být volitelné (uživatel funkce je nemusí zadávat k tomu, aby funkce fungovala v pořádku). Ve VBA se takový parametr označuje klíčovým slovem *Optional*. Pokud je potřeba přiřadit tomuto parametru výchozí hodnotu, přiřadí se pomocí znaku rovná se.

```
'Nadefinována funkce s názvem "MojeFunkce"
Function MojeFunkce(Jmeno, Optional Prijmeni = "ve VBA")
    'Návratová hodnota funkce - Vrátí vložené jméno a příjmení, pokud bylo vloženo
    'Pokud Prijmeni nebylo vloženo, vrátí místo něj "ve VBA"
    MojeFunkce = Jmeno & " " & Prijmeni
End Function
```

Obrázek 16 – Ukázka funkce s volitelným parametrem s výchozí hodnotou

### 1.2.5 Úkoly k procvičení

- 1) Vytvořte makro *Pozdrav*, které zobrazí zprávu „Pozdrav z UTB“.
- 2) Nahrajte makro *NovyList*, které vytvoří nový list v sešitě.
- 3) Vytvořte funkci *MojeJmeno*, která po použití zobrazí v dané buňce Vaše jméno.
- 4) Vytvořte v dokumentu *osoby.xlsm* nový modul a novou funkci *CeleJmeno*, která bude mít vstupní parametry (povinný) *Jmeno*, (povinný) *Prijmeni*, (volitelný) *TitulPred* a (volitelný) *TitulZa*. Výstupem této funkce bude kompletní jméno včetně titulů, pokud jsou zadány. Doplňte pomocí funkce sloupec *Celé Jméno* v daném sešitě. Bonus: Odstraňte všechny mezery navíc před a za celými jmény.

## 1.3 Lekce 2 – Proměnné, datové typy a výpočty

### 1.3.1 Proměnné

Při vytváření procedur je občas zapotřebí ukládat určité hodnoty, výsledky výpočtů a jiná data, které algoritmus zapsaný v kódu potřebuje. K tomuto účelu slouží proměnné, které alokují místo v paměti počítače a uloží do něj tyto potřebné hodnoty.

Deklarace proměnných ve VBA proběhne, pokud se dříve nepoužitému názvu (názvu proměnné) přiřadí požadovaná hodnota. Přiřazení hodnoty probíhá napsáním názvu proměnné následovaného znakem rovná se a hodnotou, kterou chce programátor do proměnné uložit.

Pro přečtení hodnoty z proměnné se na místo v kódu napíše název proměnné. Při běhu kódu pak počítač na toto místo vloží hodnotu proměnné, kterou má uloženou v paměti. Jedná se o podobný princip jako při používání parametrů ve funkcích.

Proměnná a její hodnota je v paměti počítače, dokud běh programu nedosáhne klíčového slova *End*. Pokud je proměnná deklarována mimo procedury, její platnost vyprší až s uzavřením celého programu.

### 1.3.2 Datové typy

Kontext hodnot v proměnných je udržován pomocí datových typů. V paměti počítače jsou tyto hodnoty zapsány pomocí jedniček a nul a bez kontextu se nedá určit, jestli je tato kombinace číslo, text, datum nebo něco kompletně odlišného. Pro vývojáře i počítač zároveň značí, jak s těmito hodnotami zacházet (alokovat místo v paměti, používání určitých funkcí, ...).

Binární číslo v paměti	0100 0001 0100 0000
Číslo (desítková soustava)	16 704
Text (UTF-16; znak)	A ( <i>U+0041</i> )
Datum	24.09.1945

Tabulka 1 – Ukázka příkladu reprezentace dat a kontextu

Pokud je proměnná definována implicitně bez datového typu, přiřadí se jí automaticky datový typ *Variant*, který se přizpůsobuje na základě vložené hodnoty. To vyžaduje vyšší náklady na paměť a řízení práce s proměnou. V dnešní době jsou tyto vyšší náklady při vyšších výkonech počítačů a větších kapacit pamětí zanedbatelné, ale v minulosti mohly značit

problémy. Datový typ *Variant* zároveň umožňuje vývojářům představit v chodu programu chyby v podobě načtení dat jiného datového typu do proměnné, než je potřeba.

Z těchto důvodů se většinou vývojáři v jazyce VBA snaží implicitní deklaraci vyvarovat a místo ní použít deklaraci explicitní, která zaručí, že v deklarované proměnné může být pouze jeden datový typ. Při přidělení jiného datového typu do takto deklarované proměnné program ohlásí chybu.

Vývojář si může nastavit tvrdé vyžadování explicitní deklarace pomocí klíčového spojení *Option Explicit*, které by se mělo uvádět na začátku každého modulu. Toto nastavení omezí implicitní deklaraci pro celý modul a nutí vývojáře uvádět datové typy. Při spuštění procedury s implicitní deklarací se zobrazí chyba kompilátoru s hláškou „*Proměnná není definována.*“

Datový typ	Význam	Hodnoty
<b>Boolean</b>	Logická hodnota	True / False (Pravda / Nepravda)
<b>Integer</b>	Celé číslo	-32 768 až 32 767
<b>String</b>	Text (řetězec znaků)	0 až 2 <sup>32</sup>
<b>Double</b>	Desetinné číslo	-1,79769313486231E308 až -4,94065645841247E-324 pro záporná čísla 4,94065645841247E-324 až 1.79769313486232E308 pro kladná čísla
<b>Date</b>	Datum	1. ledna 100 až 31. prosince 9999
<b>Currency</b>	Měna	-922 337 203 685 477,5808 až 922 337 203 685 477,5808
Long	Celé číslo	-2 147 483 648 až 2 147 483 647
LongLong	Celé číslo	-9 223 372 036 854 775 808 až 9 223 372 036 854 775 807
Byte	Bajt	0 až 255
Single	Desetinné číslo	-3.402823E38 až -1.401298E-45
<b>Variant</b>	Automaticky přizpůsobitelný datový typ	Podle přiřazené hodnoty
<b>Object</b>	Objekt	Podle vlastností objektu
Type	Uživatelsky definovaný datový typ	Libovolné (dle použitých typů)

Tabulka 2 – Tabulka důležitých datových typů a jejich význam a hodnoty

Datový typ *Boolean* je vhodné použít v případě, pokud je potřeba vyjádřit v logice programu stav o nějakém předmětu – například zapnuto a vypnuto, nebo platí a neplatí. Ostatní použití

datových typů (*Integer, Double, Date, String, Currency, ...*) v logice algoritmu je poměrně odvoditelné od jejich významu.

### 1.3.3 Operátory při práci s proměnnými

Do proměnných lze ukládat i výsledky výpočtu, které lze provést pomocí aritmetických operátorů, jiných proměnných a konstant. Číselné konstanty mohou být celá nebo desetinná čísla, která se ovšem musí vždy psát s desetinnou tečkou a nikoliv čárkou.

Operátor	Popis
+	Sečte dvě čísla (na levé a pravé straně)
-	Odečte od levého čísla pravé číslo
*	Vynásobí dvě čísla (na levé a pravé straně)
/	Podělí číslo na levé straně pravým číslem
Mod	Modulo, zjistí celočíselný zbytek po dělení levého čísla pravým číslem
^	Umocní levé číslo pravým číslem

Tabulka 3 – Tabulka aritmetických operátorů ve VBA

Výpočet může obsahovat závorky pro nejvyšší přednost ve výpočtu. Počítač nadále vypočítá veškerá umocnění ve výpočtu. Pokud jich je v příkladu více, postupuje z levé strany na stranu pravou. Po umocnění následuje násobení, dělení a modulo vše v pořadí zleva doprava. Na závěr počítač provede veškerá sčítání a odečítání opět ve stejném pořadí z levé strany.

```
Dim x As Integer
x = 10 + 5 * 2 ^ 3 / 4
'Výsledek x bude 20
```

Obrázek 17 – Ukázka kódu s příkladem priorit výpočtu

Pro práci s řetězcí znaků v proměnných existují pouze dva operátory, které řetězce spojují. Ostatní funkce pro práci s textem nelze provádět pomocí operátorů a musí být použity specializované funkce o kterých bude zmínka později.

Operátor	Popis
+	Spojí řetězec znaků na levé straně s řetězcem na straně pravé
&	Spojí řetězec znaků na levé straně s řetězcem na straně pravé

Tabulka 4 – Tabulka spojovacích operátorů ve VBA

```
levaStrana = "ABE"
pravaStrana = "CEDA"
dohromady = levaStrana + pravaStrana
```

Obrázek 18 – Ukázka kódu s použitím spojovacího operátoru

### 1.3.4 Převody mezi datovými typy

Pro případy, kdy je například potřeba převést řetězec obsahující číslo na číselnou hodnotu, se kterou lze počítat existují tzv. Funkce převodu typů (anglicky *Type conversion functions*). Tyto funkce se ze vstupního argumentu (přijímané hodnoty) pokusí vytvořit hodnotu s požadovaným datovým typem. Název funkce tvoří velké písmeno C (z anglického *convert*) následované zkratkou potřebného návratového datového typu.

Název funkce	Návratový datový typ	Přijímané hodnoty
<b>CBool()</b>	Boolean	Validní textová nebo číselná hodnota.
<b>CByte()</b>	Byte	0 až 255.
<b>CCur()</b>	Currency	-922 337 203 685 477,5808 až 922 337 203 685 477,5808.
<b>CDate()</b>	Date	Validní datum.
<b>CDbl()</b>	Double	-1,79769313486231E308 až -4,94065645841247E-324 pro záporná čísla 4,94065645841247E-324 až 1.79769313486232E308 pro kladná čísla.
<b>CInt()</b>	Integer	-32 768 až 32 767 (zaokrouhluje desetinná).
<b>CLng()</b>	Long	-2 147 483 648 až 2 147 483 647.
<b>CLngLng()</b>	LongLong	-9 223 372 036 854 775 808 až 9 223 372 036 854 775 807.
<b>CSng()</b>	Single	-3.402823E38 až -1.401298E-45.
<b>CStr()</b>	String	Různé, většinou z nich vyjadřuje vstup převedený na text.
<b>CVar()</b>	Variant	Podle typu.

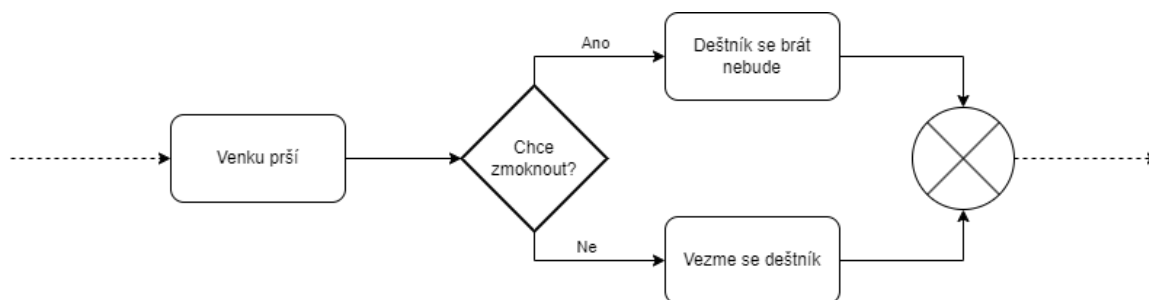
Tabulka 5 – Tabulka funkcí převodu datových typů pro proměnné

### 1.3.5 Úkoly k procvičení

- 1) Vytvořte funkce *palecNaCm* a *cmNaPalec*, které budou převádět mezi jednotkami palec a centimetry. (1 palec = 2,54 cm).
- 2) Vytvořte funkci *obvodObdelniku*, která vypočítá obvod obdélníku na základě vložených parametrů *stranaA* a *stranaB*.
- 3) Vytvořte funkci *prevodNaCela*, která bude převádět desetinná čísla na celá.
- 4) Vytvořte funkci *jeSude*, která vrátí *PRAVDA* nebo *NEPRAVDA* podle toho, jestli je vstupní číslo sudé nebo ne. Použijte operátory pro výpočet a funkce pro převod datových typů. Otestujte funkci v sešitě Excel na několika číslech.
- 5) Vytvořte funkci *obsahKruhu*, která vypočítá obsah kruhu na základě argumentu poloměru  $r$ . Pro přesnější  $\pi$  použijte funkci *WorksheetFunction.Pi()*.

## 1.4 Lekce 3 – Větvení a podmíněné příkazy

Při tvorbě algoritmů vývojář narazí na místo v toku programu, kde je potřeba kód rozdělit na dvě a více možných cest. Rozdělení probíhá na základě určitých podmínek, které v rámci programu v dané situaci nastali. Pokud je podmínka pro danou větev splněna, program se rozhodne tuto větev následovat a ostatní větve jsou ignorovány, dokud se větve opět nespojí.



Obrázek 19 – Diagram příkladu více toků v algoritmu

Pro příklad, pokud bude venku pršet, uživatel si položí otázku, zda chce zmoknout? Pokud ne, musí si vzít deštník, což vyžaduje jinou činnost, než pokud by se deštník nevzal. V tomto bodě se kód větví dle informace na existující otázku.

Ve VBA existují dva hlavní způsoby pro vypořádání se s větvením algoritmu – Příkazy *If*, *Else* a *Else If* nebo příkaz *Select Case*.

### 1.4.1 Příkazy If, Else, Else If

Příkaz *If* lze použít samostatně, a to v případě, kdy je potřeba provést kód pouze pokud platí podmínka a není potřeba řešit nic jiného. Podmínka musí být datového typu *Boolean*, a pokud nabývá hodnoty *PRAVDA*, provede se příslušný kód. Tento blok kódu začíná na novém řádku po klíčovém slově *Then* a je ukončen klíčovými slovy *End If*.

```
' Kód
If podmínka Then
    ' Kód, který se spustí, pokud je podmínka PRAVDA (True)
End If
' Pokračování kódu
```

Obrázek 20 – Struktura jednoduchého podmíněného příkazu

Příkaz *ElseIf* přidává další možnost větvení kódu, pokud první podmínka (u příkazu *If*) obsahuje hodnotu *NEPRAVDA*. Blok pod *ElseIf* se spustí v případě, že jeho podmínka je opět *PRAVDA*. Tento příkaz s bloky kódu se může opakovat v libovolném množství a může se

vyzkoušet stejné libovolné množství podmínek. Konečný příkaz *Else* již nepotřebuje žádnou podmínku, protože kód se spustí vždy, pokud žádný z předchozích příkazů *If* a *ElseIf* nebyl *PRAVDA*.

```

If podmínka Then
    ' Kód pro první podmínku
ElseIf jinaPodminka Then
    ' Kód pro jinou podmínku
Else
    ' Kód pro všechny ostatní případy
End If

```

Obrázek 21 – Struktura příkazu *If*, *ElseIf* a *Else*, ukázka větvení

### 1.4.2 Porovnávací operátory

Kromě proměnných lze v příkazech *If*, *ElseIf* a *Else* použít porovnávací operátory, k dosažení logické (*Boolean*) hodnoty pro správné spuštění kódu. Porovnávací operátory lze použít, pokud se snažíme získat logickou hodnotu i na jiných místech, například při ukládání do proměnné nebo při získání návratové hodnoty z funkce.

Operátor	Popis
=	Pokud je levá strana rovna pravé straně, vrátí <i>PRAVDA (True)</i> , jinak <i>NEPRAVDA (False)</i> .
<	Pokud je levá strana menší než pravá strana, vrátí <i>PRAVDA (True)</i> , jinak <i>NEPRAVDA (False)</i> .
>	Pokud je levá strana větší než pravá strana, vrátí <i>PRAVDA (True)</i> , jinak <i>NEPRAVDA (False)</i> .
<=	Pokud je levá strana menší než pravá strana nebo rovna pravé straně, vrátí <i>PRAVDA (True)</i> , jinak <i>NEPRAVDA (False)</i> .
=>	Pokud je levá strana větší než pravá strana nebo rovna pravé straně, vrátí <i>PRAVDA (True)</i> , jinak <i>NEPRAVDA (False)</i> .
<>	Pokud není levá strana rovna pravé straně, vrátí <i>PRAVDA (True)</i> , jinak (pokud se rovnají) <i>NEPRAVDA (False)</i> .

Tabulka 6 – Tabulka porovnávacích operátorů a jejich popis činnosti

Pro příklad, uživatel má vstupní číslo, u kterého je potřeba zjistit, zda je záporné, kladné, nebo nula. Problém je možné vyřešit pomocí příkazu *If*, který rozdělí cestu algoritmu dle potřeby. Potřebné jsou k tomu ovšem podmínky, které splňují určená pravidla. K získání těchto podmínek je možné použít správně dosazené porovnávací operátory do porovnání čísla společně s číslem nula.

```

' Proměnná s číslem
cislo = 11

If (cislo > 0) Then
    ' Číslo je větší jak nula => číslo je kladné
    MsgBox ("Číslo je kladné!")
ElseIf (cislo < 0) Then
    ' Číslo je menší jak nula => číslo je záporné
    MsgBox ("Číslo je záporné!")
Else
    ' Číslo je nula, protože jiná možnost neexistuje
    MsgBox ("Číslo je nula!")
End If

```

Obrázek 22 – Ukázka kódu použití porovnávacích operátorů

### 1.4.3 Logické operátory

Využití logických operátorů nastává v případě, kdy je potřeba použít více podmínek dohromady a přidat k nim další možnou rozšiřující logiku.

Operátor	Popis
AND	Pokud je levá strana <i>PRAVDA (True)</i> a pravá strana je <i>PRAVDA (True)</i> , vrátí <i>PRAVDA (True)</i> , jinak <i>NEPRAVDA (False)</i> .
OR	Pokud je levá strana <i>PRAVDA (True)</i> nebo pravá strana <i>PRAVDA (True)</i> , vrátí <i>PRAVDA (True)</i> , jinak <i>NEPRAVDA (False)</i> .
XOR	Pokud je levá strana <i>PRAVDA (True)</i> a pravá strana <i>NEPRAVDA (False)</i> nebo pokud je levá strana <i>NEPRAVDA (False)</i> a pravá strana <i>PRAVDA (True)</i> vrátí <i>PRAVDA (True)</i> , jinak pokud jsou obě strany <i>PRAVDA (True)</i> nebo <i>NEPRAVDA (False)</i> , vrátí <i>NEPRAVDA (False)</i>
NOT	Pokud je hodnota na pravé straně <i>PRAVDA (True)</i> , vrátí <i>NEPRAVDA (False)</i> . Pokud je hodnota na pravé straně <i>NEPRAVDA (False)</i> , vrátí <i>PRAVDA (True)</i> .

Tabulka 7 – Tabulka logických operátorů ve VBA

### 1.4.4 Příkaz Select Case

Alternativou k příkazům *If*, *ElseIf* a *Else* je příkaz *Select Case*, který jako vstup přijme hodnotu (většinou z proměnné), kterou se následně snaží přirovnat ke všem možnostem, které má nadefinované. Možnosti jsou nadefinované pomocí klíčového slova *Case* následované možnou hodnotou. Pokud žádná hodnota nevyhovuje, lze nadefinovat možnost *Case Else*, která bude výchozí možností, pokud nebudou odpovídat možnostem jiným.

```

' Proměnná obsahující nějakou hodnotu
moznost = 1
' Ukázka použití Select Case
Select Case moznost
    Case 1
        MsgBox ("Vybrána možnost číslo 1.")
    Case 2
        MsgBox ("Vybrána možnost číslo 2.")
    Case 3, 4
        MsgBox ("Vybrána možnost číslo 3 a/nebo 4.")
    Case Else
        MsgBox ("Vybrána jiná možnost.")
End Select

```

Obrázek 23 – Ukázka kódu použití příkazu *Select*

V rámci jedné možnosti lze nadefinovat i více hodnot, které stačí oddělit pomocí čárky. Blok kódu pod takovou možností se spustí, pokud bude platit jedna z nadefinovaných variant. V rámci možnosti *Case* lze použít i porovnávací operátor vůči porovnávané hodnotě. Takový případ vyžaduje použít klíčové slovo *Is* následované operátorem a hodnotou.

```

Function VysledekZBodu(Optional pocetBodu = 100)
' Určí výsledek podle dosaženého počtu bodů
Select Case pocetBodu
    Case Is > 100
        VysledekZBodu = "Exceloval"
    Case 50 To 100
        VysledekZBodu = "Uspěl"
    Case Else
        VysledekZBodu = "Neuspěl"
End Select

End Function

```

Obrázek 24 – Ukázka kódu použití příkazů *Select*, *To* a *Is*

Pokud je potřeba vytvořit možnost pro řadu čísel, lze použít příkaz *To*. Například, pokud chceme, aby možnost byla spustitelná pro čísla od 0 po 10, použije se *Case 0 To 10*, které zajistí tuto hranici.

#### 1.4.5 Podmíněné funkce

Psaní příkazů *If* a *Switch* je poměrně textově rozsáhlé, a ne vždy vývojář potřebuje takové množství řádků. K dosažení jednoduchých podmiňovacích příkazů lze i za pomoci podmiňovacích funkcí *Switch*, *IIf* a *Choose*.

Funkce *IIf* vrací jednu ze dvou definovaných hodnot na základě podmínky, která je v prvním parametru funkce. Dochází ke zjednodušení zápisu příkazů *If* a *Else*, pokud vývojář

potřebuje určit pouze jednu hodnotu, na základě nějaké podmínky. Výstup funkce je ovšem omezen pouze na jednu ze dvou možností a více možností s touto funkcí nelze použít.

```
Function VysledekZBoduIIf(Optional pocetBodu = 100)
    VysledekZBoduIIf = IIf(pocetBodu > 50, "Uspěl", "Neuspěl")
End Function
```

---

Obrázek 25 – Ukázka kódu použití funkce *IIf*

Funkce *Switch* zjednodušuje formát příkazu *Select Case* a definici jeho možností, pokud je potřeba vrátit pouze jednu hodnotu a nepoužít celý blok kódu. Funkce má neomezené množství parametrů a vždy se střídá podmínka, která pro hodnotu musí platit, a hodnota, která se navrátí, pokud podmínka platí.

```
Function VysledekZBoduSwitch(Optional pocetBodu = 100)
    VysledekZBoduSwitch = Switch(pocetBodu > 100, "Exceloval", _
        pocetBodu > 50, "Uspěl", True, "Neuspěl")
End Function
```

---

Obrázek 26 – Ukázka kódu použití funkce *Switch*

Průchod podmínek probíhá od první k poslední a jakmile je určitá podmínka splněna, vrátí její návratovou hodnotu a dále nepokračuje. To znamená, že i když platí více podmínek, návratová hodnota funkce bude mít hodnotu první, která je *PRAVDA*.

Princip funkce *Choose* spočívá ve vybírání z možností. Prvním parametrem funkce je hodnota typu kladného čísla. Tato hodnota dle pořadí určuje, která z hodnot se použije. První hodnota (druhý parametr funkce) je pod indexem 1 a ostatní (další parametry funkce) se po jedné navyšují.

```
Function VysledekZBoduChoose(Optional pocetBodu = 100)
    ' Znamka musí nabývat hodnot 1, 2, 3 - ty určí, která možnost z Choose se vybere
    ' Stupnice je po 50, proto dělení, funkce RoundUp zaokrouhluje na celá čísla nahoru
    ' Zaokrouhlením vznikají čísla 1, 2, 3, 4, ...
    znamka = WorksheetFunction.RoundUp(pocetBodu / 50, 0)
    VysledekZBoduChoose = Choose(znamka, "Neuspěl", "Uspěl", "Exceloval")
End Function
```

Obrázek 27 – Ukázka kódu použití funkce *Choose*

#### 1.4.6 Úkoly k procvičení

- 1) Vytvořte funkci *JePřestupnyRok* s číselným parametrem *rok*, která vrátí hodnotu *PRAVDA* pokud je rok přestupný a hodnotu *NEPRAVDA*, pokud rok přestupný není.
- 2) Vytvořte funkce *DenVTydmu* s parametrem *den*, který bude představovat pořadí dne v týdnu. Návrátová hodnota funkce bude vracet (text) řetězec s názvem daného dne anebo řetězec „*CHYBA*“, pokud vstup nebude odpovídat danému dni.
- 3) Vytvořte funkci *Prospel*, která bude vracet text „Prospěl“ nebo „Neprospěl“ v sešitě *zaci.xlsx*. Pro úspěšné hodnocení musí mít žák alespoň 250 bodů a maximálně 3 absence.
- 4) Vytvořte funkci *KvadratickaRovnice* pro výpočet kořenů kvadratické rovnice. Parametry funkce budou *a*, *b*, *c*. Funkce bude vracet hodnoty „*Nemá reálné kořeny.*“ nebo dvě hodnoty spojené v řetězci pomocí „*a*“.

## 1.5 Lekce 4 – Kalendářní a časové hodnoty

I když by bylo možné v předtím jmenovaných datových typech udržovat hodnoty pro čas a kalendář, z vývojářského a uživatelského pohledu by tato akce byla příliš nepraktická. Z tohoto důvodu VBA nabízí možnost práce s časem jiným (jednodušším) způsobem.

### 1.5.1 Kalendářní hodnoty

Do proměnné lze uložit pevnou kalendářní hodnotu pomocí zápisu ve tvaru *#měsíc/den/rok#*, který vychází z anglického formátování kalendářního data. Mřížky v zápise označují, že se jedná o celek jako datum, jinak by bylo provedeno dělení se zapsanými čísly.

```
Function PocetDniOdDnes (datum)
    dnes = Date
    PocetDniOdDnes = Abs (dnes - datum)
End Function
```

---

```
Function PocetDniOdNovehoRoku2023 (datum)
    ' 1. leden 2023 --> leden / 1 / 2023
    novyRok = #1/1/2023#
    PocetDniOdNovehoRoku = datum - novyRok
End Function
```

Obrázek 28 – Ukázka kódu práce s kalendářními hodnotami

Od kalendářních hodnot lze přičítat a odečítat počet dní obdobně jako při počítání příkladů. VBA umí pracovat i s přičtením a odečtením pevně zapsaných kalendářních hodnot. Aktuální datum lze zjistit pomocí funkce *Date*, která navrátí aktuální den v kalendáři.

### 1.5.2 Časové hodnoty

Podobně jako v případě kalendářních hodnot se pro proměnné zapisují hodnoty času ve tvaru *#hodiny:minuty:sekundy#*. Mřížky opět označují, že se jedná o celek ve formátu času. Editor VBA uzpůsobí zapsaný formát vloženého času do anglického formátu.

```
Sub ScheduleMeeting ()
    ' Zadáno ve formátu #22:35:44#
    ' --> 22 hodin 35 minut a 44 sekund
    meeting = #10:35:44 PM#

    ' Upozornění dvě hodiny předem (formát #02:00:00#)
    upozorneni = meeting - #2:00:00 AM#
End Sub
```

Obrázek 29 – Ukázka kódu práce s časovými hodnotami

U časových hodnot nelze přičítat ani odečítat samostatná čísla, ale pouze vždy časové hodnoty. Aktuální čas lze získat funkcí *Time*.

### 1.5.3 Práce s kalendářem a časem

Mezi další funkce pro práci s časem patří funkce *Now*, která navrátí aktuální datum a čas spojené v jedné hodnotě. Hodnota data a času lze zapsat společně ve tvaru *#měsíc/den/rok hodiny/minuty/sekundy AM/PM#*.

Funkce *Weekday* vrátí na základě vloženého data (první parametr), číselnou hodnotu dne v týdnu. Druhý parametr určuje, který den v týdnu je jako první. Výchozí hodnota je nastavena na neděli (hodnota *vbSunday*, vychází z anglicky mluvících zemí). Pro českou lokalizaci je nutné uvést hodnotu *vbMonday* (pondělí), aby pro nás funkce fungovala správně.

Mezi ostatní funkce, které navrátí číselnou hodnotu z parametru datumu patří:

- *Day* – Navrátí číselnou hodnotu dne datumu
- *Month* – Navrátí číselnou hodnotu měsíce datumu
- *Year* – Navrátí číselnou hodnotu roku datumu
- *WorksheetFunction.WeekNum* – Navrátí číselnou hodnotu týdne v roce datumu (začínající v pondělí, pokud je druhý parametr *vbMonday*)
- *Hour* – Navrátí číselnou hodnotu hodin datumu
- *Minute* – Navrátí číselnou hodnotu minut datumu
- *Second* – Navrátí číselnou hodnotu sekund datumu

```
' datum 13.8.2023
datum = #8/13/2023 10:55:01 AM#

' Navrátí hodnotu (den v týdnu) 7 (Neděle)
MsgBox (Weekday(datum, vbMonday))
' Navrátí hodnotu (den) 13
MsgBox (Day(datum))
' Navrátí hodnotu (měsíc) 8
MsgBox (Month(datum))
' Navrátí hodnotu (rok) 2023
MsgBox (Year(datum))
' Navrátí hodnotu (týden) 33
MsgBox (WorksheetFunction.WeekNum(datum, vbMonday))
' Navrátí hodnotu (hodina) 10
MsgBox (Hour(datum))
' Navrátí hodnotu (minuta) 55
MsgBox (Minute(datum))
' Navrátí hodnotu (sekunda) 01
MsgBox (Second(datum))
```

Obrázek 30 – Ukázka použití funkcí pro práci s datumem

Opakem funkcí pro převod určité informace z datumů je funkce *DateSerial*, která z drobných informací je schopna vytvořit kompletní datum. Funkci je vhodné použít při

kombinování rozložených dat. Funkce přijímá celočíselné parametry *rok*, *měsíc* a *den* v daném pořadí.

```
den = 17
mesic = 2 ' Únor
rok = 2001

' Vytvoří datum datového typu Date
datum = DateSerial(rok, mesic, den)
' Zobrazí "17.02.2001"
MsgBox (datum)
```

Obrázek 31 – Ukázka kódu použité funkce *DateSerial*

Pro pomoc s počítáním nových dat od předem zadaného data existuje funkce *DateAdd*. Funkce na základě prvního parametru *interval* zjistí, jakou jednotku času má přičítat/odečítat. Jednotky jsou zadávány do řetězce dle tabulky nastavení. Druhý parametr *number* určuje, kolik jednotek se má k datu přičíst nebo odečíst. Třetí parametr funkce *date* je pro datum, které určuje časový bod, od kterého se nové datum bude počítat.

```
Dim datum As Date
datum = #2/17/2001#

' o 8 týdnů později
noveDatum = DateAdd("ww", 8, datum)

' Zobrazí "14.4.2001"
MsgBox (noveDatum)
```

Obrázek 32 – Ukázka kódu použité funkce *DateAdd*

Nastavení	Popis
yyyy	Rok
q	Čtvrtletí
m	Měsíc
y	Pořadové číslo v roce
d	Den měsíce
w	Den v týdnu
ww	Týden
h	Hodina
n	Minuta
s	Sekunda

Tabulka 8 – Nastavení pro parametr *interval* v *DateAdd*, *DateDiff*, *DatePart*

Pro měření časových intervalů (čtvrtletí, týdnů, měsíců, čtvrtků atd.) mezi dvěma daty je možné použít funkci *DateDiff*. Funkce obsahuje 3 povinné parametry a 2 nepovinné. První parametrem *interval* se vyjadřuje, v jaké jednotce času bude funkce měřit. Druhý a třetí parametr (*date1*, *date2*) jsou daty, které ohraničují měřený úsek. Čtvrtý parametr *firstday-ofweek* určuje, který den v týdnu je jako první – pro Českou republiku by zde měla být nastavena hodnota *vbMonday*.

Poslední parametr *firstweekofyear* určuje, jakým způsobem se má započítat první týden v novém roce. Výchozím nastavením je týden, ve kterém se objeví datum 1. ledna (hodnota *vbFirstJan1*). Kromě výchozího nastavení existují nastavení *vbFirstFourDays* (první týden je ten, který má v sobě alespoň 4 dny v novém roce) a *vbFirstFullWeek* (první týden je ten, který má celý týden v novém roce).

```
Dim datum As Date
datum = #2/17/2001#
' o 8 týdnů později
noveDatum = DateAdd("ww", 8, datum)

' Spočítá počet týdnů mezi daty
pocetTydnu = DateDiff("ww", datum, noveDatum, vbMonday, vbFirstJan1)

' Zobrazí 8
MsgBox (pocetTydnu)
```

Obrázek 33 – Ukázka kódu použité funkce *DateDiff*

Funkce *DatePart* vrací ze vstupní kalendářní hodnoty část, která je specifikovaná v parametru *interval*. Jedná se o podobnou funkčnost jako mají funkce *Year*, *Month*, *Day*, *Hour*, *Minute* a *Second*. Parametry funkce jsou obdobné funkci *DateDiff*.

#### 1.5.4 Úkoly k procvičení

- 1) Vytvořte makro *OdRoku2000*, které po spuštění oznámí uživateli informaci o počtu dní, které uběhly od 1. ledna 2000.
- 2) Vytvořte makro *ZaStoLet*, které po spuštění oznámí uživateli, jaký den v týdnu bude za 100 let od dnešního dne.
- 3) Vytvořte funkci *DoJeziska*, která zobrazí počet hodin do nejbližšího vánočního dne. Ověřte, zda bude funkce fungovat i v příštích letech.
- 4) Upravte funkci *DenVTydu*, aby jako argument přijala datum místo pořadí dne v týdnu a navrátila textovou hodnotu dne v týdnu, podle tohoto data.
- 5) Vytvořte makro *Nejblizi29*, které uživateli vypíše datum nejbližšího 29. února společně s dobou, která do tohoto data uplyne.

## 1.6 Lekce 5 – Opakování, cykly a pole

V programování (nejen) VBA kódu je občas počítači nutné říct, aby vykonal určitou činnost několikrát. Jedná se například o případy, kdy je potřeba projít několik podobných dat nebo je potřeba zopakovat určitou činnost, dokud není dosaženo požadovaného výsledku.

Jedním ze způsobů implementace opakování jsou cykly. Cykly jsou příkazy, které opakují svůj vnitřní blok kódu, dokud není dosaženo nějaké podmínky. Ve VBA jsou celkově 3 základní cykly: *While*, *For* a *Do* (ve variantách *Until* a *While*).

### 1.6.1 Cyklus While

Definice cyklu *While* začíná klíčovým slovem *While* a končí klíčovým slovem *Wend*. Za příkaz *While* následně patří podmínka, která bude celý cyklus řídit. Pokud má daná podmínka hodnotu *PRAVDA*, cyklus se bude opakovat. Ke kontrole podmínky dochází vždy při začátku cyklu a pokud její hodnota bude *NEPRAVDA*, cyklus bude v daném místě ukončen.

```
Dim i As Integer
i = 0

' Počítání do 20
While (i <= 20)
    MsgBox (i)
    i = i + 1
Wend
```

Obrázek 34 – Ukázka kódu *While* cyklu

### 1.6.2 Cyklus For

Na rozdíl od cyklu *While* se cyklus *For* snaží mít proměnou, která se posunuje k určitému cíli. Může se jednat o číselný index nebo jinou hodnotu. Při cyklu *For* se ve výchozím stavu nekontroluje žádná ukončovací podmínka. Cyklus se vždy posune o další běh, dokud nedosáhne svého posledního běhu.

```
' Počítání do 20
For i = 0 To 20
    MsgBox (i)
Next i
```

Obrázek 35 – Ukázka kódu *For* cyklu

Při deklaraci cyklu *For* se definuje proměnná cyklu a přidělí se jí počáteční celočíselná hodnota. Hodnota je následována klíčovým slovem *To*, které označuje cílovou hodnotu, které

cyklus musí dosáhnout k jeho ukončení. Po konečné hodnotě může příkaz pokračovat klíčovým slovem *Step*, které určuje velikost kroku, o který se proměnná po doběhnutí cyklu zvýší.

```
zprava = ""
' Počítání do 20 po dvou
For i = 0 To 20 Step 2
    zprava = zprava & CStr(i) & " "
Next i

' Zobrazí "0 2 4 6 8 10 12 14 16 18 20 "
MsgBox (zprava)
```

Obrázek 36 – Ukázka kódu *For* cyklu se zvýšeným krokem

Krok cyklu je zakončen klíčovým slovem *Next*, které označuje konec bloku kódu, který se opakuje. Příkaz *Next* zároveň provede potřebný posun proměnné cyklu dle určené velikosti kroku. Tento příkaz by měl vždy být následován názvem proměnné cyklu, aby bylo v kódu určeno, kterou proměnnou má posunovat – více v podkapitole „Vnořené cykly.“

### 1.6.3 Cyklus Do

U příkazu cyklu *Do* existují dvě varianty jeho implementace. První z nich je možnost použít klíčové slovo *While*, které zajistí stejnou funkcionalitu cyklu s podmínkou, jako předem jmenovaný cyklus *While*. Obě varianty cyklu jsou vždy ukončeny klíčovým slovem *Loop*, které označuje konec opakujícího se bloku.

```
Dim i As Integer
i = 0

' Počítání do 20
Do While (i <= 20)
    MsgBox (i)
    i = i + 1
Loop
```

Obrázek 37 – Ukázka kódu *Do While* cyklu

Druhou variantou je cyklus *Do Until*, který na první pohled vypadá obdobně jako cyklus *Do While*. Rozdíl je v tom, že cyklus pokračuje, když je podmínka hodnoty *NEPRAVDA* a ukončí se v momentě, kdy je kontrolní podmínka hodnoty *PRAVDA*.

```
Dim i As Integer
i = 0

' Počítání do 20
Do Until (i > 20)
    MsgBox (i)
    i = i + 1
Loop
```

Obrázek 38 – Ukázka kódu *Do Until* cyklu

### 1.6.4 Vnořené cykly

V rámci programování ve VBA je možné použít i více cyklů vnořených do sebe, klidně i pomocí jiných příkazů. Jednotlivé cykly lze předčasně ukončit pomocí klíčového slova *Exit* následované názvem příkazu cyklu k ukončení (*Exit For*, *Exit Do*, *Exit While*). Opuštění cyklu bývá doprovázeno určitou podmínkou, která určuje situaci, ve které se má cyklus ukončit.

```
' Počítání do 20
Do Until (i > 20)
    MsgBox (i)

    ' Pokud má i hodnotu 15 - ukončí se cyklus
    If (i = 15) Then
        Exit Do
    End If

    i = i + 1
Loop
```

Obrázek 39 – Ukázka kódu s předčasným ukončením cyklu

Při použití více cyklů příkazu *For* je nutné přiřadit správné proměnné cyklu pomocí příkazu *Next*. Jinak by mohlo dojít v chybě logiky nebo kompilátoru při spuštění špatně napsaného příkazu.

```
zprava = ""

' Vytvoří 4 řádky s hvězdičkami
For radky = 1 To 4 Step 1
    For hvezdciky = 1 To 10
        zprava = zprava & "*"
    Next hvezdciky

    ' Přidá nový řádek na konec
    zprava = zprava & vbNewLine
Next radky

' Zobrazí:
' *****
' *****
' *****
' *****
MsgBox (zprava)
```

Obrázek 40 – Ukázka kódu s vnořeným cyklem *For*

Ve VB.NET je v rámci cyklů používán příkaz *Continue* (*Do* | *While* | *For*), který pokračuje s další iterací cyklu v momentě, kde byl použit. Jinými slovy příkaz ukončuje běh aktuálního

bloku kódu a spouští cyklus znovu, jako by byl celý blok kódu dokončen. VBA tuto funkci postrádá a vývojář musí psát logiku svých cyklů bez potřeby této funkce.

### 1.6.5 Rekurzivní funkce

Dalším způsobem opakování jsou rekurzivní volání funkce. V principu jde o volání dané funkce uvnitř bloku kódu této funkce. Volání by vždy mělo mít určitou ukončovací podmínku, která zamezí nekonečnému volání funkce. Kdyby volání nebylo ukončeno, program by vyčerpal veškerou paměť, kterou by měl k dispozici, a nakonec by spadl.

```
' Výpočet faktoriálu, např. 6! = 720
Function FaktorialRekurzivne (cislo)

    If cislo = 1 Then
        FaktorialRekurzivne = 1
    ElseIf cislo <= 0 Then
        FaktorialRekurzivne = 0
    Else
        FaktorialRekurzivne = cislo * FaktorialRekurzivne (cislo - 1)
    End If

End Function
```

Obrázek 41 – Ukázka kódu rekurzivní funkce (faktoriál) rekurzivním způsobem

### 1.6.6 Pole a matice

Občas je potřeba zapsat hodnoty podobného charakteru do několika proměnných. Například řekněme, že existují data, která určují vzdálenost od jednotlivých míst trasy. Museli by se tvořit číselné proměnné *vzdalenost12*, *vzdalenost23*, *vzdalenost34* atd. Vytvářet tolik proměnných by ovšem bylo nepraktické, z tohoto důvodu existují pole (*array*).

Pole představuje jednu proměnnou určitého datového typu, která ukládá více hodnot. Hodnoty jsou indexovány od 0 do hodnoty o jedna menší, než je velikost pole. Pomocí indexu lze k určité hodnotě v poli přistoupit a přečíst ji nebo přepsat.

Ukázka indexování pole o velikosti 10 prvků										
Index prvku	0	1	2	3	4	5	6	7	8	9
Pořadí prvku	1.	2.	3.	4.	5.	6.	7.	8.	9.	10.

Tabulka 9 – Ukázka indexování pole

V našem případě by se pole mohlo jmenovat *vzdalenosti* a pod indexem 0, by mohla být vzdálenost ze startu k prvnímu bodu, v indexu 1 by mohla být vzdálenost od prvního bodu k druhému bodu atd.

```
' Dynamické pole
Dim vzdalenosti() As Variant
vzdalenosti = Array(3.14, 3.2, 4, 5.2, 6.8, 2.7, 0.5, 13, 0.2, 0.8)

' Statické pole s omezenou velikostí
Dim sVzdalenosti(10) As Double

For i = 0 To 10
    sVzdalenosti(i) = vzdalenosti(0)
Next i

' Vypiše první vzdálenost
MsgBox (vzdalenosti(0))
```

Obrázek 42 – Ukázka deklarace statického a dynamického pole

Pole se převážně zapisují a čtou pomocí *For* a jiných cyklů, protože čtení z pole je opakující se činnost, která potřebuje iterační proměnnou. Statické pole má omezené množství prvků podle jeho deklarace. Množství je omezeno celý číslem v závorce deklarovaného pole. Dynamické pole se deklaruje bez čísel a nemá omezené množství.

```
' Dynamické pole
vzdalenosti = Array(3.14, 3.2, 4, 5.2, 6.8, 2.7, 0.5, 13, 0.2, 0.8)

soucetVzdalenosti = 0
' Index má maximum 9, protože pole má 10 prvků
For i = 0 To (10 - 1)
    soucetVzdalenosti = soucetVzdalenosti + vzdalenosti(i)
Next i

' Zobrazí 39,54
MsgBox (soucetVzdalenosti)
```

Obrázek 43 – Ukázka součtu všech prvků v poli pomocí cyklu *For*

Kromě jedno rozměrových polí existují vícedimenzionální pole. Pro jednoduchou představu lze říct, že dvourozměrné pole (matice) má pro každý svůj prvek další vlastní jednodimenzionální pole. Dvoudimenzionální pole mají dva indexy pro určení hodnoty, představující souřadnice – podobně jako je řádek a sloupec v tabulce. Pro procházení matic je potřeba použít vnořené cykly.

```

' První číslo představuje souřadnici rok, druhé číslo pak měsíc
Dim ziskMesic(2001 To 2023, 1 To 12) As Currency

ziskMesic(2001, 12) = 1235485.35

' Zobrazí zisk 1235485 Kč v lednu 2001
MsgBox (ziskMesic(2001, 12))

```

Obrázek 44 – Ukázka kódu vytvoření matice

### 1.6.7 Cyklus For Each

V poli a jiných kolekcích, které lze procházet, lze použít *For* cyklus s příkazem *Each*, které místo počítání s proměnou cyklu postupně přebere prvky v určité řadě a přiřadí je do proměnné pro dané kolo cyklu. Tímto způsobem projde všechny prvky v kolekci a aplikuje pro ně naprogramovaný blok kódu. Vývojář se nemusí starat o indexy jednotlivých prvků a všechny je může projít a použít dle vhodného uvážení.

```

' Dynamické pole
vzdalenosti = Array(3.14, 3.2, 4, 5.2, 6.8, 2.7, 0.5, 13, 0.2, 0.8)

soucetVzdalenosti = 0
' Index se neřeší, protože jej cyklus postupně prochází
For Each vzdalenost In vzdalenosti
    soucetVzdalenosti = soucetVzdalenosti + vzdalenost
Next vzdalenost

' Zobrazí 39,54
MsgBox (soucetVzdalenosti)

```

Obrázek 45 – Ukázka kódu použití cyklu *For Each*

### 1.6.8 Úkoly k procvičení

- 1) Vytvořte funkci *Pyramida* s vnořeným cyklem *For*, které pomocí znaku #, vytvoří ze symbolů 5 patrovou pyramidu v buňce sešitu.

```
#  
###  
#####  
#####  
#####
```

- 2) Vytvořte makro *Nasobilka* s dvoudimenzionálním polem, o rozměrech 10x10, které bude obsahovat hodnoty násobilky malých čísel od 1x1 až 10x10. Obsah matice vypište pomocí jedné zprávy.
- 3) Zkopírujte makro *Nasobilka* a přejmenujte ho na *NasobilkaObracene*. Upravte cyklus makra, tak aby vznikla nová matice otočená přes stoupající diagonálu s využitím cyklů.

## 1.7 Lekce 6 – Práce s oblastmi výběru

V rámci předchozích lekcí se rozebírala převážně témata ohledně programovacího jazyku VBA a jak jej použít. Důležitou součástí je i propojení tohoto jazyka s programem Microsoft Excel. Propojení zatím bylo znázorněno ohledně funkcí a maker, ale nikoliv ohledně dat, se kterými tabulkový software pracuje.

### 1.7.1 Aktivní označení

První možná manipulace s listem je skrz objekt *ActiveCell*. Jedná se o odkaz na buňku, která je v listu sešitu aktuálně označena. Přes tento odkaz lze číst hodnotu buňky pomocí *ActiveCell.Value* a zapisovat nové hodnoty pomocí přiřazení k objektu pomocí znaku rovná se. Při zápisu se přepíše původní hodnota buňky.

```
' Vložení hodnoty do označené buňky
ActiveCell = 100

' Čtení hodnoty z označené buňky
precteno = ActiveCell.Value

' Zobrazí 100
MsgBox (precteno)
```

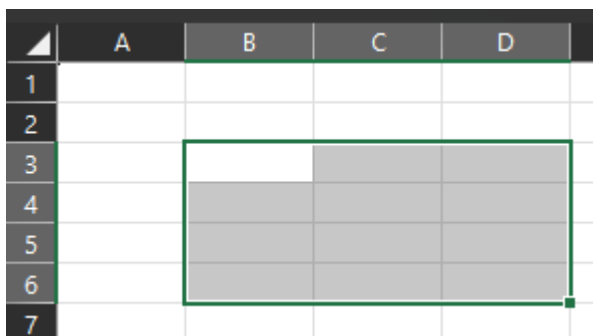
Obrázek 46 – Ukázka kódu se zápisem a čtením z označené buňky

Do buňky lze vložit i funkce, které se nacházejí samostatně v Excelu. Název funkce musí být v angličtině, nikoliv v češtině – např. SUMA musí být SUM. Do označené buňky se musí funkce vložit přes odkaz *ActiveCell.Formula*.

```
' Vložení funkce Excelu do označené buňky
ActiveCell.Formula = "=SUM(E12:E20)"
```

Obrázek 47 – Ukázka kódu pro vložení funkce do buňky

V rámci tabulky Excelu lze vybrat více buněk najednou, které by se dali označit za „aktivní“. Objekt *ActiveCell* odkazuje pouze na označenou buňku bez šedého podkresu, a nikoliv na celý výběr. K odkazu na celý výběr se používá objekt *Selection*.



Obrázek 48 – Vybraná oblast (*Selection*) a označená buňka (bílá, *ActiveCell*)

Do vybrané oblasti *Selection* lze zapisovat stejným způsobem jako do *ActiveCell*. Nová hodnota přepíše všechny hodnoty v označené oblasti. Čtení oblasti pomocí *Selection.Value* navrátí pole datového typu *Variant*, které v označených dimenzích postupně obsahuje hodnoty jednotlivých buněk.

### 1.7.2 Pasivní přístup

Pasivní přístup k datům listu sešitu dovoluje přístup k celému sešitu bez nutnosti označení uživatelem. Nejčastěji se k datům přistupuje za pomoci objektu *Range*, která díky souřadnicovému systému Excelu je schopna vybrat konkrétní buňku nebo celou oblast buněk.

Souřadnice sešitu tvoří písmeno(a) pro sloupce a číslo pro řádky. Pokud je potřeba vybrat oblast pomocí souřadnic, použije se znak dvojtečky mezi souřadnicemi buněk protilehlých rohů vybírané oblasti. Pokud se vybíraná buňka nebo oblast vyskytuje na specifickém listu sešitu, přidá se před souřadnice název daného sešitu a znak vykřičníku. Čtení a zapisování do vybraných buněk přes objekt *Range* probíhá obdobně jako u objektů *ActiveCell* a *Selection*.

```
' Vybrání jedné buňky
Range("A1") = "Jedna buňka"

' Vybrání oblasti
Range("3G:13I") = "Oblast"

' Vybrání jiného listu
Range("Ukázka2!A1:Z1") = "Jiný list"
```

Obrázek 49 – Ukázka kódu užití souřadnic s objektem *Range*

Pomocí objektu *Range* lze přečíst data přímo do pole, se kterým se dá nadále v kódu VBA pracovat. Čtení hodnot do pole lze provést i z více pozic najednou pomocí *Array*. Zapisování hodnot jde taktéž ulehčit pomocí zápisu s *Array*.

```

' Čtení dat jako pole
Dim Data As Variant
Data = Range("List1!A1:A10").Value

' Průchod polem
For Each Item In Data
    MsgBox (Item)
Next Item

' Ukázka zápisu s pomocí Array
Range("A1:D1") = Array("Titul Před Jménem", "Jméno", "Příjmení", "Titul za jménem")

' Ukázka čtení různých buněk s pomocí Array
hlavicka = Array(Range("A1"), Range("B1"), Range("C1"), Range("D1"))

```

Obrázek 50 – Ukázka kódu získání dat jako pole

Alternativním přístupem k datům buněk v listě sešitu Excel je použit vlastnost *Cells*. Tato vlastnost používá systém číselných souřadnic, přesněji pořadí řádků a sloupců. To znamená, že souřadnice ke konkrétní buňce v *Range* „A1“ odpovídá *Cells(1, 1)*.

V rámci výběru oblasti *Range* lze zřetězit i vlastnost *Cells*, která následně nevybírá buňku z celého sešitu, ale z oblasti vybrané pomocí *Range*. To umožňuje používat klasické *For* cykly s iterační proměnnou k procházení jednotlivých buněk a měnění jejich vlastností.

```

' Výběr buňky z vybrané oblasti, buňka 2. řádek a 3.sloupec
Range("D6:A2").Cells(2, 3) = "Podoblast"

```

Obrázek 51 – Ukázka kódu použití *Cells* v *Range*

Objekt *Range* a vlastnost *Cells* lze použít i k posunutí objektů *ActiveCell* a *Selection*. Po označení buňky nebo oblasti stačí zřetězit příkaz pomocí *Activate* nebo pomocí *Select*. Příkazem bude označena pouze jedna oblast, pro vybrání více oblastí je nutné upravit parametr *Range* a vepsat do něj obě oblasti oddělené středníkem. Po použití příkazu se vybraná oblast posune na specifikovanou oblast pomocí *Range* nebo *Cells*.

```

' Aktivní označení oblasti
Range("D16:H25").Cells(2, 3).Activate

' Aktivní označení více oblastí
Range("A2:D6, B17:A33").Select

```

Obrázek 52 – Ukázka kódu posunutí označené oblasti

Pokud je potřeba vybrat buňku s odsazením, která se nachází od určitého výběru, lze použít *Offset*. Odsazení má číselnou hodnotu řádku a sloupce, která určuje o kolik se má výběr posunout.

```
' Posune celou oblast o 2 řádky níže a 3 sloupce doprava
Range("D6:A2").Offset(2, 3).Select
```

Obrázek 53 – Ukázka kódu odsazení výběru pomocí *Offset*

Kromě *Offset* lze použít *Resize*, který naopak rozšiřuje vybranou oblast o počet buněk určitým směrem. Vývojář tak nemusí udržovat přehled o souřadnicích, na kterých se nachází, ale může se v kódu dynamicky posouvat od uvedeného výběru.

### 1.7.3 Řádky a sloupce sešitu

Obdobně jako *Cells* fungují *Columns* a *Rows*, které na základě uvedeného čísla řádku nebo sloupce pracují s daty celého sloupce.

```
' Vybere 5. sloupec (E)
Columns(5).Select
Columns("E").Select

' Vybere 3. řádek (3)
Rows(3).Select
```

Obrázek 54 – Ukázka kódu použití objektů *Columns* a *Rows*

Vlastnost *EntireColumn* přistupuje ve vybrané oblasti k celému sloupci a umožňuje sloupce odebrat, vymazat, přidat nebo skrýt. Sloupce je možné odebrat pomocí *Delete*, vymazat pomocí *Clear*, přidat pomocí *Clear* a skrýt pomocí nastavení vlastnosti *Hidden* na hodnotu *PRAVDA* (*True*). Sloupec se znovu zobrazí po změně této hodnoty zpět na *NEPRAVDA* (*False*).

```
' Smaže hodnoty v celém sloupci
Columns("B").EntireColumn.Clear

' Odstraní sloupec
Columns("B").EntireColumn.Delete

' Přidá nový sloupec
Columns("B").EntireColumn.Insert

' Skryje sloupec
Columns("B").EntireColumn.Hidden = True

' Odkryje sloupec
Columns("B").EntireColumn.Hidden = False
```

Obrázek 55 – Ukázka kódu využití *EntireColumn*

Obdobně jako *EntireColumn* funguje vlastnost *EntireRow*, která místo sloupců umožňuje obdobné řešení pro řádky v sešitu.

#### 1.7.4 Úkoly k procvičení

- 1) Vytvořte makro *PridatCtyriRadky*, které přidá do sešitu k aktuální pozici 4 nové řádky
- 2) V sešitě *bludiste.xlsx* vytvořte ovládací makra, které budou posouvat aktivní buňku o jednu pozici dle směru kliknutí tlačítka. V buňce se bude mazat a zapisovat znak tečky (•), který bude společně s aktivní buňkou označovat polohu v bludišti.
- 3) V sešitě *mena.xlsx* vytvořte novým makrem *NaEuro* nový sloupec vedle CZK, který vytvoří sloupec s cenami v EUR. Použijte aktuálně dostupný kurz.

## 1.8 Lekce 7 – Stylování textu a buněk

Buňky v sešitě Excel jde stylovat různými barvami textu, pozadí, tloušťkou ohraničení a dalšími vlastnostmi, které buňky mají. Tyto vlastnosti jsou přístupné i z programovacího jazyka VBA.

### 1.8.1 Přístup pomocí příkazu *With*

Pro ulehčení práce se změnami několika vlastností vybrané oblasti buněk je možné použít příkaz *With*, který označuje následně zapsaný objekt (např. výběr *Range*, *Cells*, *Columns*, *Selection*, ...) pro editaci jeho vlastností.

```
' Ukázka správně používaného With
With Cells(1, 1)
    .RowHeight = 10
    .ColumnWidth = 20
    With .Font
        .ColorIndex = 2
    End With
End With
```

Obrázek 56 – Ukázka kódu použití *With*

Tímto způsobem se nemusí znovu opisovat jméno nebo přístup k objektu neustále dokola a může se přímo přistupovat k jeho vlastnostem. Blok kódu, kde jsou určeny vlastnosti musí být ukončen příkazem *End With*. Používání klauzule *With* usnadňuje práci a přehlednost kódu, proto je vhodné ji využívat.

### 1.8.2 Výška a šířka buňky

Nastavení výšky buňky se dělá pomocí číselné vlastnosti *RowHeight*. Pokud je potřebné změnit výšku na všech řádcích, je vhodné přistoupit k řádkům pomocí *Cells.Rows.RowHeight*. Pokud je potřebné změnit výšku pouze jednoho řádku nebo jedné buňky, je vhodné přistupovat pomocí *Range*. Automatická výška řádku podle velikosti textu v buňkách se nastaví pomocí *Cells.Rows.AutoFit*.

```
' Konkrétní nastavení u buněk ve výběru
Range("A1:B2").RowHeight = 12

' Nastaví velikost všech řádků na automatickou
Cells.Rows.AutoFit
```

Obrázek 57 – Ukázka kódu použití *RowHeight*

Obdobně jako u řádků lze všechny činnosti provést se sloupci a jejich šířkou. Vlastnost nebude *RowHeight*, ale nově *ColumnWidth*. Automatická šířka podle textu se opět nastaví pomocí *Cells.Columns.AutoFit*.

```
' Konkrétní nastavení u buněk ve výběru
Range("A1:B2").ColumnWidth = 12

' Nastaví velikost všech řádků na automatickou
Cells.Columns.AutoFit
```

Obrázek 58 – Ukázka kódu použití *ColumnWidth*

Šířka řádku může být ovlivněna zalomením textu v buňce. Tato vlastnost se nastavuje pomocí *WrapText* a může nabývat hodnoty *PRAVDA* nebo *NEPRAVDA*, představující zapnuté a vypnuté zalamování textu.

```
' Zapne v buňce zalomení textu
Range("C12").WrapText = True

' Vypne v buňce zalomení textu
Range("C12").WrapText = False
```

Obrázek 59 – Ukázka kódu se zalomením textu

### 1.8.3 Ohraničení buňky

K ohraničení buněk v listu je prvně nutné buňky vybrat (např. pomocí *Range*) a následně se použije metoda *Borders*. Parametr v metodě určuje, kterou stranou ohraničení se budou vlastnosti ohraničení určovat. Může nabývat hodnot *xlEdgeTop* (horní ohraničení), *xlEdgeLeft* (levé ohraničení), *xlEdgeRight* (pravé ohraničení), *xlEdgeBottom* (dolní ohraničení), *xlDiagonalDown* (diagonála z levého horního rohu k pravému dolnímu), *xlDiagonalUp* (diagonála z levého dolního rohu k pravému hornímu), *xlInsideVertical* (vnitřní vertikální ohraničení), *xlInsideHorizontal* (vnitřní horizontální ohraničení).

```
' Horní ohraničení
With Range("C12").Borders(xlEdgeTop)
    .Weight = xlThick
    .LineStyle = xlDash
    .ColorIndex = 3
End With

' Dolní ohraničení
With Range("C12").Borders(xlEdgeBottom)
    .Weight = xlThick
    .LineStyle = xlDash
    .ColorIndex = 4
End With
```

Obrázek 60 – Ukázka kódu použití ohraničení buňky

Každé ohraničení *Borders* má pak vlastnosti *LineStyle* (styl čáry), *Weight* (tloušťka čáry) a *ColorIndex* (barevný index). Tloušťka čáry může nabývat hodnot *xlHairline* (tenká čára), *xlMedium* (střední tloušťka), *xlThick* (tlustá čára) a *xlThin* (tenčí čára). Vlastnost stylu čáry může nabývat hodnot *xlContinuous*, *xlDash*, *xlDashDot*, *xlDashDotDot*, *xlDot*, *xlDouble*, *xlLineStyleNone*, *xlSlantDashNot*.

Pokud je potřeba z buňky všechny ohraničení odstranit, je nutné nastavit pro všechny strany ohraničení vlastnost *LineStyle* na hodnotu *xlNone*. Hromadně lze toto nastavení aplikovat pomocí *Borders* bez zadaného parametru.

```
' Odstraní ohraničení buňky C12
With Range("C12").Borders
    .LineStyle = xlNone
End With
```

Obrázek 61 – Ukázka kódu kompletního odstranění ohraničení buňky

#### 1.8.4 Slučování buněk

Vybraná oblast buněk (např. pomocí *Range*) lze sloučit příkazem *Merge* a rozdělit pomocí příkazu *UnMerge*. Slučování buněk může zapříčinit ztrátu hodnot ve slučovaných buňkách, obvykle zůstává pouze hodnota, v buňce, která se nachází nejvíce vlevo.

```
' Sloučení buněk
Range("A2:C3").Merge

' Rozdělení buněk
Range("A2:C3").UnMerge
```

Obrázek 62 – Ukázka kódu sloučení a rozdělení buněk

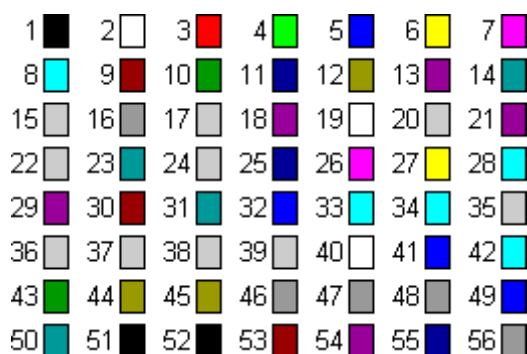
#### 1.8.5 Barva a text buňky

Barvy buněk lze měnit pomocí objektu *Interior* a jeho vlastnosti *ColorIndexu*, který určuje barvu z předdefinovaných pomocí čísla, nebo pomocí vlastnosti *Color*, kde je potřeba zadat přes *RGB* funkci barevné hodnoty pro červenou, zelenou a modrou barvu. Pro odstranění barvy musí mít *ColorIndex* přiřazenou hodnotu *xlNone*.

```
' Předdefinované barvy pomocí ColorIndex
Range("A2").Interior.ColorIndex = 9

' Nastavení vlastní barvy pomocí hodnot RGB
Range("A2").Interior.Color = RGB(250, 160, 150)
```

Obrázek 63 – Ukázka kódu nastavení barvy pozadí v buňce



Obrázek 64 – Hodnoty barev *ColorIndex*. Zdroj: <https://learn.microsoft.com/en-us/office/vba/api/excel.colorindex>

Obdobně jako u pozadí buňky se mění barva fontu buňky. Místo objektu *Interior* se přistupuje k vlastnosti *Font* v buňce. Vlastnosti *Color* a *ColorIndex* zůstávají stejné.

```
' Výběr barvy písma
Range("A2").Font.ColorIndex = 9

' Nastavení vlastní barvy písma pomocí hodnot RGB
Range("A2").Font.Color = RGB(250, 160, 150)
```

Obrázek 65 – Ukázka kódu změny barvy písma

Přes vlastnost *Font* lze měnit i kurzívu a tučnost textu. Vlastnosti *Bold* (tučnost) a *Italic* (kurzíva) nabývají hodnot PRAVDA a NEPRAVDA, které odpovídají zapnutí nebo vypnutí vlastnosti. Další číselná vlastnost *Size* slouží pro zvětšení textu. Rodina fontu lze změnit pomocí *Font.Name* následovaná řetězcem s názvem nového fontu.

```
' Nastavení ostatních vlastností fontu v buňce
With Range("A3").Font
    .Bold = True
    .Italic = False
    .Size = 22
    .Name = "Comic Sans MS"
    .Strikethrough = True
End With
```

Obrázek 66 – Ukázka kódu nastavení vlastností Fontu

V buňkách listu je možné zarovnávat text vertikálně a horizontálně. Obě vlastnosti jsou přístupné z výběru buňky, *HorizontalAlignment* pro horizontální a *VerticalAlignment* pro vertikální. Vlastnost může nabývat hodnot *xlBottom*, *xlCenter*, *xlDistributed*, *xlJustify* a *xlTop*. Text jde otočit pomocí číselné vlastnosti *Orientation*, např. o 90° nastavením hodnoty 90. Limity otáčení jsou omezeny na minimální hodnotu -90 a maximální hodnotu +90 a pokud jsou překročeny, akce skončí chybou programu.

```
' Nastavení zarovnání textu v buňce
With Range("A3")
    .HorizontalAlignment = xlCenter
    .VerticalAlignment = xlCenter
    .Orientation = 36
End With
```

Obrázek 67 – Ukázka kódu využití vlastností zarovnání textu v buňce

### 1.8.6 Formátování buněk

Pro vhodnější reprezentaci dat v buňkách tabulek Excelu je potřeba data formátovat. Formátování upravuje zobrazení dat pro lepší čitelnost a prezentaci.

Vlastnost buněk *NumberFormat* umožňuje zadat ve VBA vlastní formát buňky. Obdobně je to v Excelu možné v záložce *Domů*, ve skupině *Číslo* a z nabídky *Formát buněk* vybrat možnost *Vlastní*. Například řetězec „#,##0.00“ by v buňce odděloval tisíce a zároveň dvě desetinná čísla. Pokud je potřeba nastavit v buňce obecný tvar, použije se hodnota *General*.

```
' Oddělovač tisíců a dvou desetinných míst
Range("A23").NumberFormat = "#,##0.00"
' Z hodnoty 1000000,365
' bude v buňce zobrazena 1 000 000,37
```

Obrázek 68 – Ukázka použití vlastnosti buněk *NumberFormat*

Obdobnou funkci zajišťuje funkce *Format*, která místo nastavení formátu na buňku, převádí ve VBA vložené hodnoty do řetězce podle zadaného formátu v parametru.

```
' Získání původní hodnoty
hodnota = Range("A22").Value

' Zformátování hodnoty
retezec = Format(hodnota, "#,##0.00")

' Uložení řetězce jako hodnota buňky
Range("A24") = retezec
```

Obrázek 69 – Ukázka vytvoření naformátovaného řetězce

22	1000000,365	<-- Původní hodnota v buňce
23	1 000 000,37	<-- Naformátovaná buňka, hodnota lze měnit
24	1 000 000,37	<-- Řetězec (TEXT), s hodnotou lze pracovat jen jako text, nikoliv číslo

Obrázek 70 – Ukázka rozdílu mezi formátem buňky a funkcí *Format*

### 1.8.7 Uzamknutí buňky

K uzamčení určitých buněk v listu sešitu se po jejich vybrání (např. pomocí *Range*) může použít vlastnost *Locked*, která může nabývat hodnot *PRAVDA* a *NEPRAVDA* odpovídající uzamčení a odemčení buněk. Obdobně lze i skrýt vzorce v buňkách pomocí vlastnosti *FormulaHidden*, například ke skrytí výpočetního firemního tajemství.

```
' Zamknutí buněk
Range("A2:B14").Locked = True

' Skrytí vzorců v buňkách
Range("A2:B14").FormulaHidden = True
```

Obrázek 71 – Ukázka kódu zamknutí buněk

K tomu, aby bylo možné vlastnosti zamykání a skrývání buněk využít, je potřeba, aby byl list ve stavu *Protected*. Bez této vlastnosti funkce nebudou fungovat.

### 1.8.8 Úkoly k procvičení

- 1) V sešitě *omalovanka.xlsm* doplňte makra pro tlačítka k vytvoření možnosti pomocí marker vybarvit obrázek.
- 2) V sešitě *omalovanka.xlsm* vytvořte makro, které upraví nadpis na font Comic Sans a otočí jej o 5 stupňů.
- 3) Vytvořte makro *MujFormat*, které aktivně vybrané buňce nastaví formát s oddělenými mezerami po tisíci.
- 4) Vytvořte makro, které nastaví Vámi zvolené barevné ohraničení u Vámi zvolené oblasti

## 1.9 Lekce 8 – Práce se sešitem

Kromě manipulace buněk v rámci listů je v Excelu možné přistupovat k celému sešitu. V programovacím jazyce VBA je manipulování celý sešitem primárně řešeno objekty *Worksheets* a *Sheets*, o kterých bude primární řeč v této lekci.

### 1.9.1 Vybrání konkrétního listu sešitu

Přistoupit ke konkrétnímu objektu listu sešitu lze pomocí objektu *Sheets*. Vývojář může jako argument vložit index listu nebo celý název listu jako řetězec. Vybírání sešitu probíhá pomocí funkce *Select*, která po zavolání zobrazí vybraný list uživateli v Excelu.

```
' použití názvu listu
Sheets("NovyList").Select

' použití indexu listu
Sheets(2).Select
```

Obrázek 72 – Ukázka kódu objektu *Sheets* a funkce *Select*

Objekt *Sheets* lze procházet jako pole pomocí *For Each* cyklu, například jde zobrazit nebo vypsat všechny názvy listů v sešitě pomocí vlastnosti *Name*. Objekt *Sheets* zahrnuje i listy s grafy, na rozdíl od objektu *Worksheets*.

```
' Vypíše názvy všech listů v sešitě
For Each Sh In Sheets
    MsgBox (Sh.Name)
Next Sh
```

Obrázek 73 – Ukázka procházení objektu *Sheets*

<poznámka ohledně aktivování kódu pomocí *Activate*>

### 1.9.2 Zamykání a odemykání listů

V objektu *Sheets* je dostupná metoda *Protect*, která uzamkne vybraný list sešitu. Metoda má několik nepovinných parametrů. Nejdůležitějším parametrem je *password*, který uzamkne sešit společně se zadaným heslem. Ostatní parametry nastavují možnosti práce s listem, jak jej po zamknutí lze upravovat.

```

' Zápís zamknutí pouze s heslem
Sheets(1).Protect ("heslo")
' Alternativně
Sheets(1).Protect Password:="heslo"

' Ostatní parametry funkce Protect
' DrawingObjects, Contents, Scenarios, UserInterfaceOnly,
' AllowFormattingCells, AllowFormattingColumns, AllowFormattingRows,
' AllowInsertingColumns, AllowInsertingRows, AllowInsertingHyperlinks,
' AllowDeletingColumns, AllowDeletingRows, AllowSorting, AllowFiltering,
' AllowUsingPivotTables

```

Obrázek 74 – Ukázka kódu uzamknutí listu

Odemykání listu probíhá podobnou procedurou *Unprotect*, která má volitelný parametr *Password* se stejnou funkcí jako u procedury *Protect*.

```

' Odemknutí listu sešitu
Sheets(1).Unprotect ("heslo")

```

Obrázek 75 – Ukázka kódu odemknutí sešitu

### 1.9.3 Zobrazení listů

Listy lze zobrazit a skrýt pomocí vlastnosti objektu *Sheets* pojmenované *Visible*. Pokud má vlastnost hodnotu *PRAVDA*, list je zobrazený a viditelný v sešitě. Naopak s hodnotou *NEPRAVDA* je list skryt.

```

Sub SkrytNovyList()

' Skryje list, lze obnovit z Excelu
Sheets("NovyList").Visible = xlSheetHidden

' Skryje list, nelze obnovit z Excelu
Sheets("NovyList").Visible = xlSheetVeryHidden

' Zviditelní list
Sheets("NovyList").Visible = xlSheetVisible

End Sub

```

Obrázek 76 – Ukázka kódu skrytí a zobrazení sešitu v listě

Alternativně lze použít předdefinované hodnoty *xlSheetHidden*, *xlSheetVeryHidden* a *xlSheetVisible*. Hodnota *xlSheetVeryHidden* nastaví list skrytý bez možnosti obnovit jeho viditelnost uživatelem v Excelu. Vždy je nutné takový list zviditelnit pomocí makra.

#### 1.9.4 Přidání a odebrání listů

Přidání listů do sešitu probíhá pomocí metody *Add*, která může obsahovat dva parametry o nové pozici listu. Parametr *Before* určuje pozici před zmíněným sešitem, kde se nový sešit vloží. Parametr *After* zase pozici po určeném sešitě, kde se nový sešit objeví.

```
' Přidá nový list na začátek sešitu
Worksheets.Add

' Přidá nový list před událostí
Worksheets.Add Before:=Worksheets("Události")

' Přidá nový list za událostí
Worksheets.Add After:=Worksheets("Události")
```

Obrázek 77 – Ukázka kódu přidání listu do sešitu

Odstranění listu probíhá jeho výběrem a zavoláním metody *Delete*. Smazaný list a jeho data nelze zpětně obnovit. Při odstraňování se Excel zeptá, zda chcete sešit opravdu odstranit.

```
' Odebere list
Worksheets("Přidaný list").Delete
```

Obrázek 78 – Ukázka kódu odebrání listu ze sešitu

#### 1.9.5 Kopírování a přesunování listů

V rámci objektu *Worksheets* je dostupná metoda *Copy*, která bez uvedeného parametru zkopíruje vybraný list aktivního sešitu a vloží jej do sešitu nového. Zkopírovaný list v novém sešitě bude mít stejný název jako měl předtím.

```
' Zkopíruje sešit do nového sešitu
Worksheets("NovyList").Copy

' Zkopíruje list NovyList za NovyList
Worksheets("NovyList").Copy After:=Worksheets("NovyList")

' Zkopíruje list NovyList před NovyList
Worksheets("NovyList").Copy Before:=Worksheets("NovyList")
```

Obrázek 79 – Ukázka kódu kopírování listu v sešitě

Vývojář ovšem ve většině případů nepotřebuje nakopírovat listy do nového sešitu. Metoda *Copy* má z tohoto důvodu dva parametry *After* a *Before*. Parametry označují nové místo pro kopírovaný list sešitu.

Pokud je potřeba novou kopii sešitu přejmenovat, stačí upravit vlastnost *Name* objektu *ActiveSheet* novým řetězcem s požadovaným názvem. Po kopírování by vždy měla nastat situace, že se zkopírovaný list zobrazí a stane se aktivním (objektem *ActiveSheet*).

```
' Označení více listů
Worksheets("Události").Select
Worksheets("NovyList").Select False

' Kopírování vybraných listů
ActiveWindow.SelectedSheets.Copy
```

Obrázek 80 – Ukázka kódu s kopírováním více listů zároveň

Pro kopii více sešitů zároveň, existuje výběr *SelectedSheets* v objektu *ActiveWindow*, který odkazuje na všechny právě vybrané listy. Označení více listů probíhá pomocí *Select* s přidáním parametrem *False*, díky kterému se neodznačí předchozí vybrané listy.

### 1.9.6 Události v listu

V sešitě mohou existovat předdefinovaná makra, která se automaticky spustí při nějaké akci spuštěné uživatelem (události). Akcí může být relativně cokoliv, od jednoduché změny hodnoty v listu až po dvoj kliknutí na buňku v listě.

```
Private Sub Worksheet_SelectionChange(ByVal Target As Range)
    ' Nastala změna v listu Události ve výběru
    MsgBox ("Nastala událost výběru v listu! Nový výběr bude označen oranžově. ")

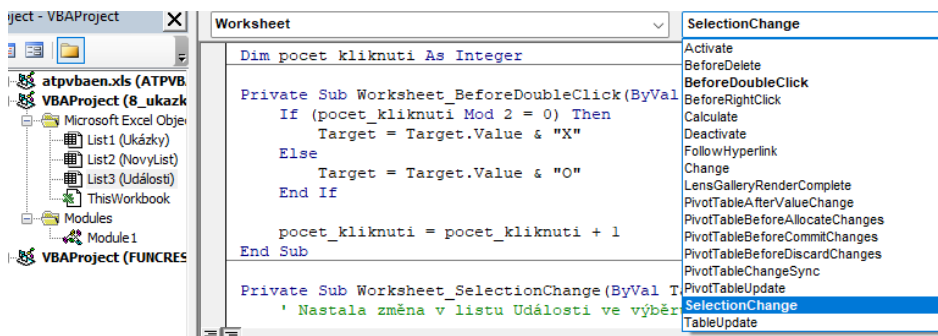
    ' Zruší barvu předchozího výběru
    Sheets("Události").Cells.Interior.ColorIndex = xlNone

    ' Označí novou oblast barvou
    Target.Interior.Color = RGB(252, 186, 3)

End Sub
```

Obrázek 81 – Ukázka kódu události *SelectionChange*

Všechn kód metod událostí musí být v objektu listu, a nikoliv Modulu jako u předchozích případů. Ve vybraném objektu se vybere objekt *Worksheet* a z druhé nabídky procedur událost, pro kterou je potřeba vytvořit kód, který se vždy při vzniku takové události spustí.



Obrázek 82 – Ukázka přidání makra události ke konkrétnímu listu

Jednotlivé typy událostí je vhodné řádně prostudovat v rámci dokumentace VBA, např. na adrese <https://learn.microsoft.com/en-us/office/vba/excel/concepts/events-worksheetfunctions-shapes/worksheet-object-events>.

---

```
Dim pocet_kliknuti As Integer
```

---

```
Private Sub Worksheet_BeforeDoubleClick(ByVal Target As Range, Cancel As Boolean)
    If (pocet_kliknuti Mod 2 = 0) Then
        Target = Target.Value & "X"
    Else
        Target = Target.Value & "O"
    End If

    pocet_kliknuti = pocet_kliknuti + 1
End Sub
```

---

Obrázek 83 – Ukázka kódu jiné události *BeforeDoubleClick*

### 1.9.7 Úkoly k procvičení

- 1) Vytvořte makro *Zamek*, které zamkne a odemkne list sešitu dle vašeho hesla
- 2) Vytvořte nové makro *NovaTabulka*, které založí nový list s hlavičkou tabulky: Typ paliva, množství (v l), cena celkem.
- 3) Pomocí dalšího makra *SkrytNoveMakro* skryjte nově vytvořený list.
- 4) Vytvořte kód, který při změně vybraného listu ohlásí název vybraného listu.

## 1.10 Lekce 9 – Práce s grafy

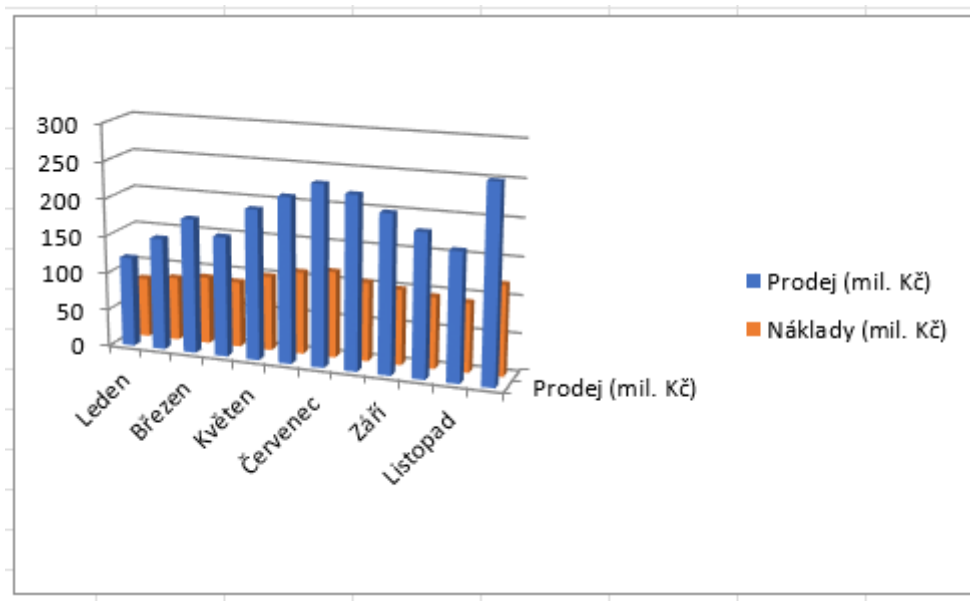
Neodmyslitelnou součástí programu Excel jsou grafy pro data, které se vyskytují na listech v tabulkách. Pomocí programovacího jazyka VBA lze tuto tvorbu grafů zautomatizovat a měnit jejich vlastnosti.

### 1.10.1 Vytvoření grafu

Grafy jsou přidávány do sešitu pomocí přidání nového grafu do tvarů listu, pomocí metody *AddChart*. Graf je následně přejmenován, aby se na něj bez potřeb objektového programování dalo odkazovat. Pomocí *Select* dochází k výběru požadovaného grafu do objektu *ActiveChart*, kde lze přistupovat k jeho vlastnostem a měnit je. Je vybrán typ grafu pomocí vlastnosti *ChartType* a nakonec jsou do něj přidány data pomocí metody *SetSourceData* s odkazem na umístění v listě *DataGrafu*. Vznikne velmi jednoduchý graf, jak lze vidět na obrázku níže.

```
Sub VytvorGraf ()  
  
    ' Graf Prodeje a Nákladů  
  
    ' Vybrání listu pro graf  
    Sheets ("ListGrafu").Select  
  
    ' Vytvoří prázdný graf na určené pozici  
    ' Přidání jména pro pozdější identifikaci  
    Sheets ("ListGrafu").Shapes.AddChart  
  
    ' Přejmenování grafu pro další reference  
    Sheets ("ListGrafu").Shapes ( _  
        Sheets ("ListGrafu").Shapes.Count _  
    ).Name = "GrafProdejeNakladu"  
  
    ' Vybrání grafu pro ActiveChart  
    Sheets ("ListGrafu").Shapes ("GrafProdejeNakladu").Select  
  
    ' Určení typu grafu  
    ActiveChart.ChartType = xl3DColumn  
  
    ' Přidání dat ke grafu  
    ActiveChart.SetSourceData Source:=Range ("DataGrafu!B4:D16")  
  
End Sub
```

Obrázek 84 – Ukázka kódu vytvoření jednoduchého grafu s daty



Obrázek 85 – Výsledný graf z ukázky kódu

### 1.10.2 Zobrazení grafu

Graf může být viditelný a neviditelný, vše ovlivňuje jeho vlastnost *Visible* stejně jak je to u ostatní podobných prvků.

```

Sub ZobrazitGraf()
    ' Vybrání sešitu s grafem
    Sheets("ListGrafu").Select

    ' Zobrazí kompletní graf
    Sheets("ListGrafu").ChartObjects("GrafProdejeNakladu").Visible = True
End Sub

Sub SkrytGraf()
    ' Vybrání sešitu s grafem
    Sheets("ListGrafu").Select

    ' Skryje kompletní graf
    Sheets("ListGrafu").ChartObjects("GrafProdejeNakladu").Visible = False
End Sub

```

Obrázek 86 – Zobrazení a skrytí grafu

### 1.10.3 Prvky grafu

Mezi prvky grafu patří například nadpis, popisy os, legendy, datové tabulky a další. Každý prvek grafu má většinou vlastnost *HasTitle* nebo *HasDatatable*, která odkazuje na informaci, zda má graf tento prvek zapnutý a viditelný. Ostatní prvky se nastavují podle jejich vlastností: nadpis například *ChartTitle*, atd.

```

Sub PridaniNadpisu()
    ' Vybrání grafu pro ActiveChart
    Sheets("ListGrafu").Shapes("GrafProdejeNakladu").Select

    If (Not ActiveChart.HasTitle) Then
        ' Zapnutí nadpisu grafu
        ActiveChart.HasTitle = True

        With ActiveChart.ChartTitle
            ' Přidání textu do nadpisu
            .Text = "Graf Prodeje a Nákladů"
            ' Změna barvy nadpisu na červenou
            .Format.TextFrame2.TextRange.Font
                .Fill.ForeColor.RGB = RGB(255, 0, 0)
        End With
    End If

End Sub

```

Obrázek 87 – Ukázka kódu přidání nadpisu grafu s červenou barvou

```

Sub OdebraniNadpisu()
    ' Vybrání grafu pro ActiveChart
    Sheets("ListGrafu").Shapes("GrafProdejeNakladu").Select

    If (ActiveChart.HasTitle) Then
        ' Odebere nadpis grafu
        ActiveChart.HasTitle = False
    End If

End Sub

```

Obrázek 88 – Ukázka kódu odebrání nadpisu grafu

```

Sub PridatPopisOs()
    ' Vybrání sešitu s grafem
    Sheets("ListGrafu").Select

    ' Vybrání grafu pro ActiveChart
    Sheets("ListGrafu").Shapes("GrafProdejeNakladu").Select

    ' Zobrazení hlavní vodorovné osy (xlCategory)
    ActiveChart.Axes(xlCategory, xlPrimary).HasTitle = True
    ActiveChart.Axes(xlCategory, xlPrimary).AxisTitle.Characters.Text = "Měsíc"

    ' Zobrazení hlavní svislé osy (xlValue)
    ActiveChart.Axes(xlValue, xlPrimary).HasTitle = True
    ActiveChart.Axes(xlValue, xlPrimary).AxisTitle.Characters.Text = "Kč"

End Sub

```

Obrázek 89 – Ukázka kódu přidání popisu os

```

Sub OdstraneniPopisOs()
    ' Vybrání sešitu s grafem
    Sheets("ListGrafu").Select

    ' Vybrání grafu pro ActiveChart
    Sheets("ListGrafu").Shapes("GrafProdejeNakladu").Select

    ' Skrytí hlavní vodorovné osy (xlCategory)
    ActiveChart.Axes(xlCategory, xlPrimary).HasTitle = False

    ' Skrytí hlavní svislé osy (xlValue)
    ActiveChart.Axes(xlValue, xlPrimary).HasTitle = False
End Sub

```

Obrázek 90 – Ukázka kódu odstranění popisu os

```

Sub ZobrazeniLegendyGrafu()
    ' Vybrání sešitu s grafem
    Sheets("ListGrafu").Select

    ' Vybrání grafu pro ActiveChart
    Sheets("ListGrafu").Shapes("GrafProdejeNakladu").Select

    ' Zapnutí zobrazení legendy
    ActiveChart.HasLegend = True

    ' Náhodná barva textu legendy
    ActiveChart.Legend.Font.Color = RGB(Int(255 * Rnd), Int(255 * Rnd), Int(255 * Rnd))
End Sub

```

Obrázek 91 – Zobrazení legendy grafu

```

Sub SkrytiLegendyGrafu()
    ' Vybrání sešitu s grafem
    Sheets("ListGrafu").Select

    ' Vybrání grafu pro ActiveChart
    Sheets("ListGrafu").Shapes("GrafProdejeNakladu").Select

    ' Vypnutí zobrazení legendy
    ActiveChart.HasLegend = False
End Sub

```

Obrázek 92 – Skrytí legendy

Všechny grafové typy jsou dostupné ze stránek <https://learn.microsoft.com/en-us/office/vba/api/excel.xlcharttype>. Vývojář si může najít svůj přesně specifický a ten používat.

```

Sub ZobrazeniTabulkyDatGrafu()
' Vybrání sešitu s grafem
Sheets("ListGrafu").Select

' Vybrání grafu pro ActiveChart
Sheets("ListGrafu").Shapes("GrafProdejeNakladu").Select

' Zobrazí tabulku dat grafu
ActiveChart.HasDataTable = True
End Sub

```

Obrázek 93 – Zobrazení tabulky dat v grafu

```

Sub SkrytiTabulkyDatGrafu()
' Vybrání sešitu s grafem
Sheets("ListGrafu").Select

' Vybrání grafu pro ActiveChart
Sheets("ListGrafu").Shapes("GrafProdejeNakladu").Select

' Skryje tabulku dat grafu
ActiveChart.HasDataTable = False
End Sub

```

Obrázek 94 – Skrytí tabulky dat v grafu

#### 1.10.4 List grafu

List grafu představuje list sešitu, který je plně zaměřen na jeden graf. Informace tak vynikají na rozdíl od použitých grafů v normálním listě. Grafový list sešitu se přidává metodou *Sheets.Add* s typem *xlChart*.

```

' Vybrání listu s původním grafem
Sheets("ListGrafu").Select

' Vybrání původního grafu ke kopii
Sheets("ListGrafu").ChartObjects("GrafProdejeNakladu").Select

' Vytvoření prázdného listu s grafem
Sheets.Add Type:=xlChart
ActiveSheet.Name = "Graf Prodeje a Nákladů"

```

Obrázek 95 – Přidání listu grafu

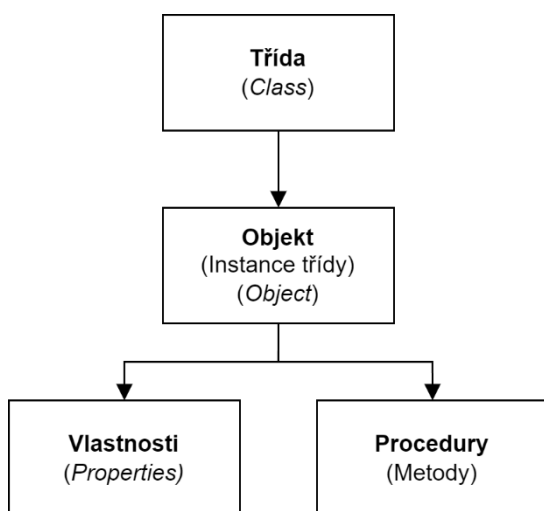
### 1.10.5 Úkoly k procvičení

- 1) Vytvořte nové makro *SloupcovyGraf*, které vytvoří sloupcový graf z dostupných dat v sešitě *brouci.xlsx*.
- 2) Přidejte k vytvořenému grafu nadpis s textem „Výskyt brouků“ pomocí kódu ve VBA.
- 3) Vytvořte nové makro *GrafSinusCosinus* a na základě hodnot v sešitě *matematika.xlsx* vytvořte bodový graf zobrazující funkce *Sinus* a *Cosinus*.
- 4) Přidejte k vytvořenému grafu pomocí VBA legendu a tabulku zobrazených dat.
- 5) Vytvořený graf překopírujte na novou stránku sešitu jako samostatný objekt na listě.

## 1.11 Lekce 10 – Třídy a objekty

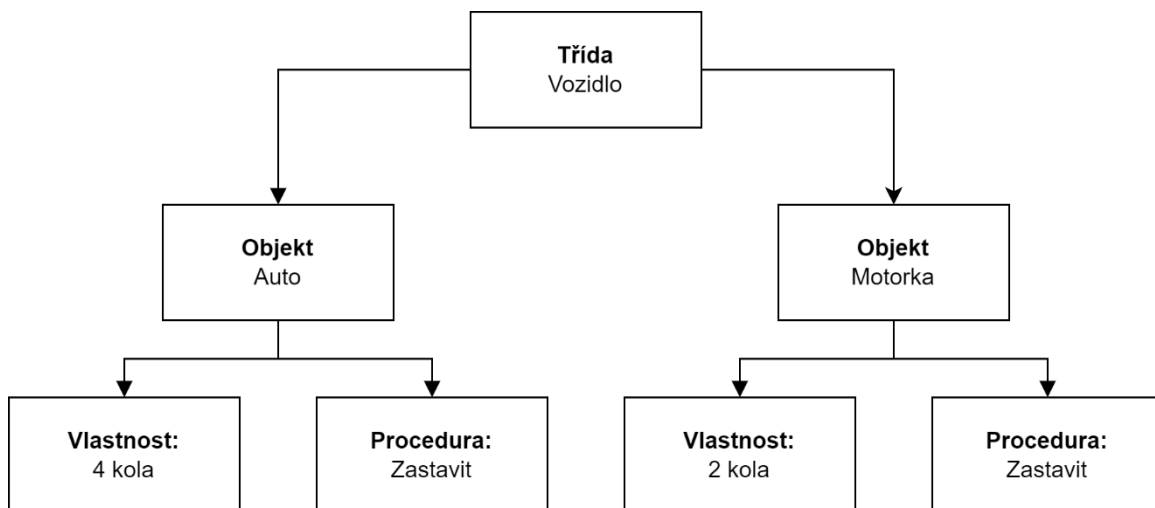
Pojem objekt můžeme chápat jako ucelenou kombinaci kódu a dat, která může být určitou částí aplikace, jako například ovládací prvek nebo okno formuláře. V principu může být i celá aplikace praktickým objektem. Při programování ve VBA se pracuje s objekty neustále, i když to na první pohled nemusí být zřejmé.

Každý objekt je ve VBA definován třídou, která popisuje proměnné, vlastnosti, procedury a události v objektu. Před použitím těchto prvků je potřeba vytvořit danou instanci třídy, tedy objekt. Přes vytvořený objekt pak lze přistupovat k jednotlivým metodám a vlastnostem.



Obrázek 96 – Ukázka schématu objektu

Pro příklad, řekněme, že máme naprogramovanou třídu *Vozidlo*, která popisuje metody a vlastnosti všech možných vozidel. Konkrétními objekty je pak například *motorka* nebo *auto*. Řekněme, že tato třída má vlastnost počet kol u vozidla a metodu zastavit.



Obrázek 97 – Schéma aplikované na příklad

Z jiného úhlu pohledu by se dalo říct, že třída určuje, jaké vlastnosti a metody mají mít její objekty, jistým smyslem je utváří a formuje, zatímco objekty jsou již hodnoty, které se řídí určenou formou.

### 1.11.1 Práce s objekty a příkaz Set

Příkaz Set je ve VBA používán k přiřazení reference na objekt do proměnné. Objekty jsou ve VBA pracovány pomocí referencí, což znamená, že proměnná neuchovává samotný objekt, ale pouze odkaz na něj. Tímto způsobem lze manipulovat s objekty a provádět s nimi různé akce.

```
Sub ZkouskaSetRange ()  
    ' vytvoří referenci na objekt  
    Dim oblast As Range  
  
    'nastaví referenci  
    Set oblast = Range("A1:A999")  
  
    ' Označí oblast a může se odkazovat na další jeho metody a vlastnosti  
    oblast.Activate  
End Sub
```

Obrázek 98 – Uchovávání referencí na třídy

### 1.11.2 Vytvoření objektu

Metoda konstruktoru tříd je speciální funkce v objektově orientovaném programování, která je volána při vytváření nové instance třídy. Tato metoda slouží k inicializaci členů objektu a provádění počátečních operací, které jsou potřebné při vytváření instance.

```
' Konstruktor třídy  
Public Sub Initialize (Jmeno_ As String, Prijmeni_ As String, HodinovaMzda_ As Double)  
    Jmeno = Jmeno_  
    Prijmeni = Prijmeni_  
    HodinovaMzda = HodinovaMzda_  
End Sub
```

---

Obrázek 99 – Konstruktor třídy

V jazyce VBA, konkrétně při definování třídy, můžete vytvořit konstruktor pomocí procedury se stejným názvem jako třída. Tím umožníte, aby se při vytváření instance této třídy automaticky zavolala tato konstrukční procedura, což umožňuje inicializovat výchozí hodnoty a nastavení objektu.

```
' Vytvoření nových zaměstnanců
Dim zaml As New Zamestnanec
zaml.Initialize "Jan", "Novák", 150
```

Obrázek 100 – Vytváření objektů s pomocí konstruktoru

### 1.11.3 Kolekce

V jednoduchých polích je poměrně složité měnit rozměry pole, přidávat a odebírat prvky do pole a pracovat s těmito prvky, např. řadit prvky nebo hledat určité prvky.

Kolekce může obsahovat objekty různých tříd a lze k nim přistupovat pomocí klíče nebo indexu. Klíč je unikátní identifikátor, kterým lze objekty v kolekci identifikovat a vyhledávat. Index je číslo určující pozici objektu v kolekci.

```
> Main()
Dim firma As New Collection
```

Obrázek 101 – Založení nové kolekce s objekty

```
firma.Add zaml
```

Obrázek 102 – Přidání objektu do kolekce

```
' Práce s kolekcí zaměstnanců
For Each Zamestnanec In firma
```

Obrázek 103 – Procházení objektů v kolekci

### 1.11.4 Vlastnosti a metody

Vlastnosti jsou charakteristiky objektu, které popisují jeho stav, vlastnosti nebo atributy. Jsou často reprezentovány pomocí proměnných, které jsou spjaty s daným objektem. Například u objektu reprezentujícího buňku v tabulce Excelu by mohla být vlastnost *Value* reprezentující hodnotu buňky.

```
' Třída pro reprezentaci zaměstnance|
Public Jmeno As String
Public Prijmeni As String
Public HodinovaMzda As Double
```

---

Obrázek 104 – Vlastnosti třídy

Metody jsou funkce, které jsou spojovány s objektem a umožňují provádět akce nebo operace s tímto objektem. Metody definují chování objektu a mohou být volány až po konkrétním objektu.

```
' Metoda pro výpočet hrubé mzdy
Public Function VypocetHrubeMzdy(odpracovaneHodiny As Double) As Double
    VypocetHrubeMzdy = odpracovaneHodiny * HodinovaMzda
End Function
```

Obrázek 105 – Metoda třídy

```
Dim odpracovaneHodiny As Double
odpracovaneHodiny = InputBox("Zadejte odpracované hodiny pro za

Dim vysledek As Double
vysledek = Zamestnanec.VypocetHrubeMzdy(odpracovaneHodiny)
```

Obrázek 106 – Ukázka užití metody ve třídě

### 1.11.5 Úkoly k procvičení

- 1) Vytvořte nové makro, které vytvoří nový objekt Worksheet.
- 2) Vytvořte vlastní třídu Ovoce
- 3) Přidejte do této třídy vlastnosti jako Název, Hmotnost a datum expirace
- 4) Přidejte do této třídy metodu JePoExpiraci, která bude vracet ano, pokud je dnešní den po datumu expirace
- 5) Vytvořte ovocný košík v podobě kolekce objektů ovoce a použijte jej

## 1.12 Lekce 11 – Práce s uživatelskými dialogy a formuláři

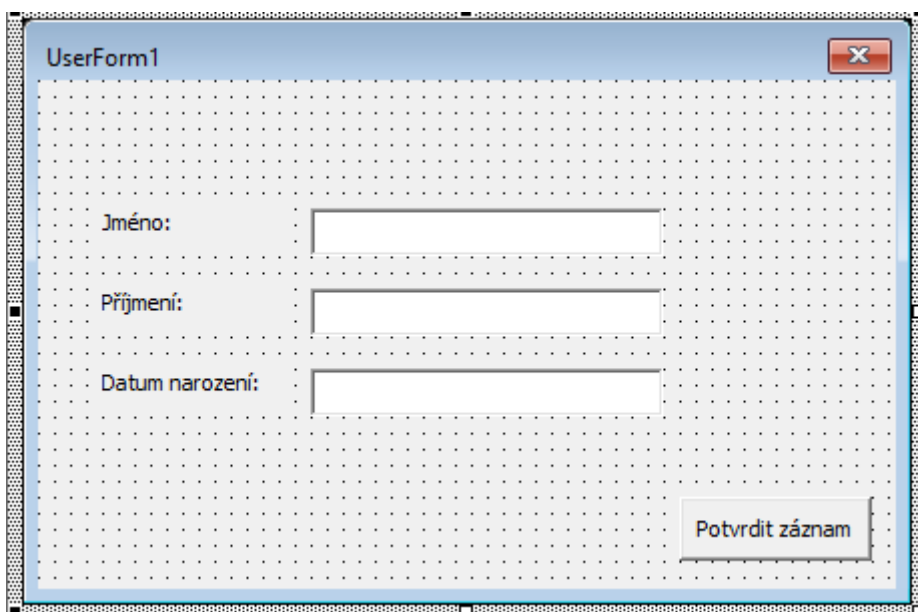
### 1.12.1 Prvek MsgBox

MsgBox (Message Box) je vestavěná funkce ve Visual Basic for Applications (VBA) a také v dalších jazycích jako je Visual Basic, která umožňuje zobrazit dialogové okno s zprávou pro uživatele. Toto dialogové okno může obsahovat textovou zprávu, tlačítka a ikonu, a slouží k zobrazení důležitých informací uživateli nebo k získání jeho reakce

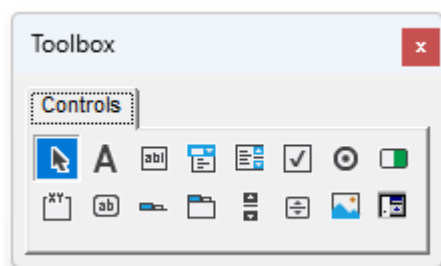
### 1.12.2 Prvek InputBox

InputBox je vestavěná funkce ve Visual Basic for Applications (VBA) a také v jazyce Visual Basic, která umožňuje zobrazit dialogové okno, kde uživatel může zadat textový vstup. Tato funkce je užitečná pro získání vstupních údajů od uživatele, jako jsou jména, čísla, hesla apod.

### 1.12.3 Ovládací prvky formuláře



Obrázek 107 – Jednoduchý formulář s prvky textových boxů a tlačítka



Obrázek 108 – Toolbox pro přidávání prvků do Formu

#### **1.12.4 Propojení formuláře s kódem**

Všechny přidané prvky jde primárně propojit do kódu pomocí dvoj kliknutí na prvek. Tento klik vás zavede přímo na funkci, kterou formulář použije, pokud například nastane událost s kliknutím na tlačítko. Výběr události je velmi podobný tomu, jako u listů.

#### **1.12.5 Úkoly k procvičení**

- 1) Vytvořte nové makro s využitím MsgBox a použitím rozhodnutí uživatele ano/ne.
- 2) Vytvořte nové makro s využitím InputBox, které bude představovat jednoduchou kalkulačku, kde uživatel zadá matematický výraz a input box jej vyhodnotí.
- 3) Vytvořte nový formulář, který přidá nový záznam zákazníků (Jméno, Příjmení, Datum narození)

## 1.13 Lekce 12 – Práce s textovými soubory

### 1.13.1 Zjištění souboru nebo adresáře

```
' Zjištění existence cesty k souboru/adresáři
Function ExistujeSoubor(filePath As String) As Boolean
    ExistujeSoubor = (Dir(filePath) <> "")
End Function
```

---

```
' Test existence cesty k souboru
Sub TestExistenceCesty()
    Dim cesta As String
    cesta = "C:\Cesta\K_MujSoubor.txt"

    If ExistujeSoubor(cesta) Then
        MsgBox "Soubor existuje."
    Else
        MsgBox "Soubor neexistuje."
    End If
End Sub
```

Obrázek 109 – Funkce pro zjištění existence adresáře

### 1.13.2 Otevření souborů

```
' Otevření souboru csv
Sub OtevritSoubor()
    Dim cesta As String
    cesta = "C:\Cesta\K_MujSoubor.csv"

    Dim textovySoubor As Workbook
    Set textovySoubor = Workbooks.OpenText(FileName:=cesta, DataType:=xlDelimited, Comma:=True)
End Sub
```

---

Obrázek 110 – Otevírání souborů .csv

### 1.13.3 Uložení souborů

```
' Uložení souboru
Sub UlozitSoubor()
    Dim cesta As String
    cesta = "C:\Cesta\K_MujSoubor.xlsx"

    ActiveWorkbook.SaveAs FileName:=cesta
End Sub
```

Obrázek 111 – Uložení sešitu

#### 1.13.4 Smazání souboru

```
' Smazání PDF
Sub SmazatPDF()
    Dim FilePath As String
    Dim PDFFileName As String

    ' Cesta k souboru a název PDF souboru
    FilePath = "C:\Users\Cesta\Pro\Smazání\"
    PDFFileName = "exportovana_tabulka.pdf"

    Kill (FilePath & PDFFileName)

    MsgBox ("PDF bylo smazáno.")
End Sub
```

Obrázek 112 – Smazání souboru

#### 1.13.5 Export souboru

```
' Export PDF
Sub ExportPDF()
    Dim FilePath As String
    Dim PDFFileName As String

    ' Cesta k souboru a název PDF souboru
    FilePath = "C:\Users\Cesta\Pro\Uložení\"
    PDFFileName = "exportovana_tabulka.pdf"

    ' Exportování sešitu do PDF
    ActiveSheet.ExportAsFixedFormat Type:=xlTypePDF, _
        Filename:=FilePath & PDFFileName, Quality:=xlQualityStandard, _
        IncludeDocProperties:=True, IgnorePrintAreas:=False, OpenAfterPublish:=False

    MsgBox ("Sešit byl úspěšně exportován do PDF.")
End Sub
```

---

Obrázek 113 – Export souboru