Makros, Prozeduren, Variablen, Konstanten, Datenfelder etc.

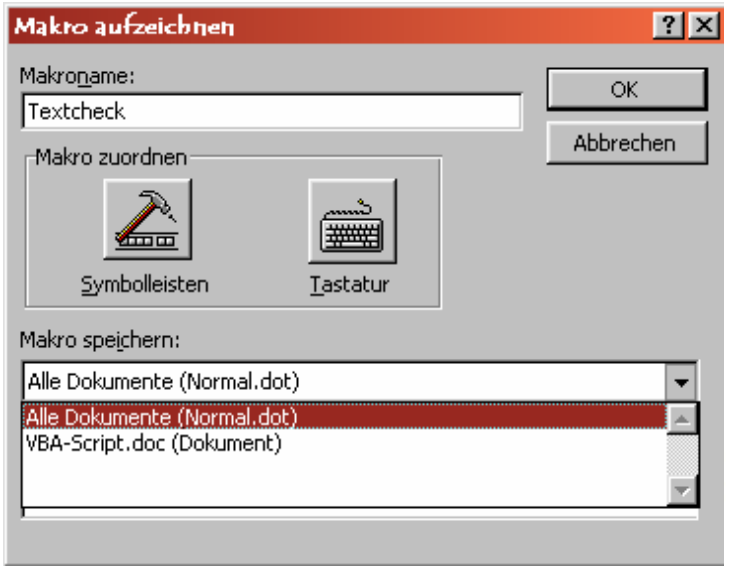
Beim bisherigen Nacharbeiten unserer Übungsaufgaben wurden Sie gelegentlich schon mit dem Begriff *Makro* konfrontiert.
Makros sind kleine Programme, die Sie erstellen können um sich z. B. immer wiederkehrende Arbeiten zu erleichtern oder auch Routinen und Abläufe zu automatisieren. Ein Makro wird - ähnlich einer Musikaufnahme – aufgezeichnet und kann anschließend so oft Sie wollen 'abgespielt' werden. Beachten Sie jedoch bei der Aufzeichnung, dass alle Ihre Aktionen, seien es Eingaben, Mausklicks, Scrollen etc. aufgenommen werden und vergessen Sie auch nicht die Aufnahme wieder zu beenden.

Die Makroaufzeichnung selbst können Sie im Menü *Extras – Makro – Aufzeichnen* oder *Aufzeichnung beenden* vornehmen. Alternativ können Sie Ihren Standard-Symbolleisten die Symbolleiste: Visual Basic hinzufügen und hier über die entsprechenden Icons den Ablauf steuern. Zum Beenden des Makros klicken Sie auf den Button *Makro beenden* 

der gleichnamigen Symbolleiste. Diese Symbolleiste wird in der Regel gleich beim Anklicken des Start-Buttons angezeigt. Sollte dies nicht der Fall sein, kann sie auch über das Menü: *Ansicht - Symbolleisten – anpassen – Symbolleisten – Aufzeichnung beenden* eingeblendet werden.

Aufzeichnen eines einfachen Makros

Starten Sie die Makroaufzeichnung wie vorstehend beschrieben und geben Sie als Makroname *Textcheck* ein. Beachten Sie, dass Doppelnamen mit Unterstrich verbunden werden müssen. Beispiel: Text_check
Legen Sie als nächstes fest, ob Ihr Makro in sämtlichen Worddokumenten verfügbar sein soll oder nur in Ihrem aktuellen Dokument. Klicken Sie auf OK und geben Sie an der Cursorposition im Worddokument das Wort 'checken!' ein. Beenden Sie danach die Aufzeichnung wie eingangs beschrieben.



Wenn Sie nun in den Visual Basic-Editor (Alt + F11) wechseln wird der Code Ihres Makros wie folgt dargestellt:

```
Sub Textcheck()
'
' Textcheck Makro
' Makro aufgezeichnet am 23.01.02 von nb

    Selection.TypeText Text:="checken!"

End Sub
```

Beim Ausführen des Makros (Menü: Extras – Makro – Makros – ausführen) wird nun an der jeweiligen Cursorposition das Wort "checken!" eingefügt.

Einfache Makros können Sie wie eben beschrieben erstellen; Sie können sie im Visual Basic-Editor weiter bearbeiten, ergänzen oder auch in komplexer Form direkt als Text eingeben. Im letzteren Fall spricht man dann allerdings nicht mehr von *Makros,* sondern von *Prozeduren*.

Detailbeschreibung VBA-Fenster

Bestandteile einer Prozedur

Wie das vorstehende Codebeispiel zeigt, wird eine Prozedur immer mit dem Begriff *Sub* eingeleitet und mit *End Sub* beendet. Hinter dem Schlüsselwort *Sub* zu Beginn der Prozedur, steht der von Ihnen vergebene Makroname (maximal 80 Zeichen, keine Leerzeichen, Satz- und Sonderzeichen und kein Bindestrich), sowie ein Klammernpaar, das von Visual Basic automatisch hinzugefügt wird. Da Sie Prozeduren oder Funktionen Werte übergeben können, werden diese sog. *Argumente* oder *Parameter* in diesem Klammernpaar definiert. Da in der vorstehenden Muster-Prozedur keine Argumente definiert wurden, bleibt das Klammernpaar leer.
Eine Prozedur die einen Wert zurückgibt, wird auch als *Funktion* bezeichnet.
Schlüsselwörter sind Begriffe die in Visual Basic festgelegt sind - sie werden automatisch in blauer Schriftfarbe dargestellt.

Beispiel: `Sub Textcheck()`
Um in einer Prozedur beispielsweise beschreibenden Text als Kommentar einbinden zu können, muss der Textpassage ein Apostroph (') oder das Kürzel *Rem* (nur am Zeilenanfang möglich) vorangestellt werden. Die anschließenden Textteile werden dann automatisch in grüner Schriftfarbe dargestellt.

Beispiel:
```
' Textcheck Makro
Rem Makro aufgezeichnet am 23.01.02 von nb
```

Befehlszeile:

Die Befehlszeile unseres Beispiel-Makros gliedert sich in folgende Segmente:

Beispiel: `Selection.TypeText Text:="checken!"`

`Selection` - die Auswahl oder Cursorposition ist das zugrunde liegende *Objekt*. Anschließend folgt durch einen Punkt getrennt eine *Methode* dieses Objekts und zwar `.TypeText`. Am Ende wird einem *Argument* `Text` mit dem *Zuweisungsoperator* `:=` der Wert "checken!" zugewiesen.

Wenn Sie den vorstehenden Code im Visual Basic-Editor eingeben wird Ihnen das Programm nach Schreiben des Punktes nach *Selection* eine *Dropdown*-Liste öffnen und alle Eigenschaften und Methoden anzeigen, die in Bezug auf das Objekt *Selection,* Anwendung finden können. Diese Hilfestellung erhalten Sie übrigens immer, wenn Sie ein Objekt benennen und danach einen Punkt eingeben.

Der 'normale' Code wird vom Programm in schwarzer Schriftfarbe dargestellt. Wird allerdings beim Kompilieren (Übersetzen des Codes in binären Maschinencode) ein Fehler entdeckt, wird die fehlerhafte Codepassage in roter Schriftfarbe angezeigt.

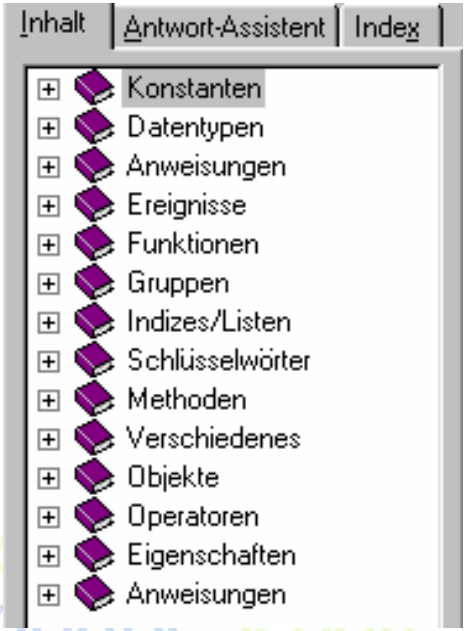
Anmerkung:

Eine Übersicht aller Objekte und der zugehörigen Elemente erhalten Sie im Visual Basic-Editor-Fenster im Menü *Ansicht* oder nach Anklicken des Icons *Objektkatalog.*

Weitere Hilfestellung zu diesem sehr umfangreichen Thema erhalten Sie auch, wenn Sie in den Namen eines Objekts, einer Eigenschaft oder Methode klicken und die F1-Taste drücken. Ferner erhalten Sie Quickinfos und Parameterinfos über das Kontextmenü angezeigt.

Nach Anklicken des  in der Menüsymbolleiste können Sie auf die Hilfebibliotheken von Word und Visual Basic zugreifen; beachten Sie jedoch, dass Ihnen im Word- und VB-Fenster unterschiedliche Hilfedateien zur Verfügung stehen, die Sie jeweils auch über öffnen können.

Beachten Sie hierzu auch die Inhalts-Übersicht der Visual Basic-Hilfe:



Objekte, Eigenschaften und Methoden

Objekte sind die Bausteine aus denen Word und die Word-Dokumente aufgebaut sind. Sie sind hierarchisch geordnet, das oberste Objekt ist das *Application*-Objekt, das gewissermaßen für die gesamte Word-Anwendung steht. Dieses oberste Objekt enthält nun eine Reihe von Objekten, die selbst wieder weitere Objekte enthalten und so weiter. Jedes dieser Word-Elemente (Tabellen, Dokumente, Absätze etc.) kann beim Programmieren als Objekt angesprochen werden, nahezu alles was Sie in Visual Basic machen ist eine Veränderung von Objekten.
Normalerweise sind einem Objekt noch weitere *Elemente* zugeordnet; sie verfügen über *Eigenschaften* und *Methoden* und sie enthalten weitere Objekte, meist sog. Auflistungen gleichartiger Objekte.

Eine *Eigenschaft* ist ein Merkmal, ein Attribut eines Objekts oder ein Aspekt seines Verhaltens. Eigenschaften kann man abfragen oder verändern. *Methoden* werden ausgeführt. Eine Methode ist eine Aktion, die ein Objekt durchführen kann. Methoden haben oft Argumente, die näher bestimmen, wie diese Aktion durchgeführt wird. Oftmals werden Methoden ein oder auch mehrere Werte übergeben. So wird im vorstehenden Beispiel der Methode *TypeText* der Text 'checken!' als Wert übergeben, der eingefügt werden soll. Ist kein konkreter Wert definiert, spricht man auch von *Parametern* die einer Methode zugeordnet sind.
Ein Objekt aus einer Auflistung kann bestimmt werden und man kann mit ihm entsprechend verfahren.

Wenn Sie sich mit dem Objektkatalog beschäftigen, werden Sie übrigens nach Anklicken des *Application*-Objekts feststellen, dass z.B. *Selection* auch eine Eigenschaft von *Application* ist.

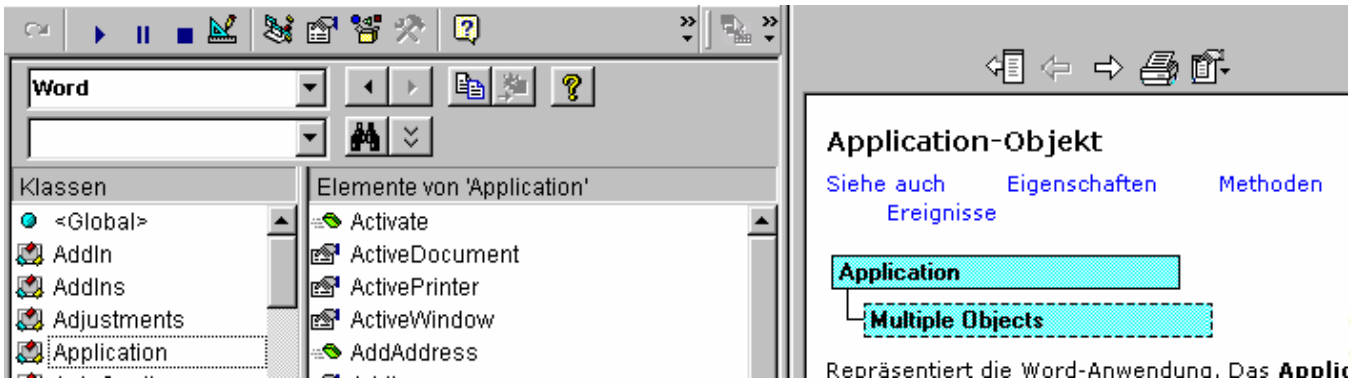
optische Darstellung der Eigenschaften: 

optische Darstellung der Methoden: 

Anmerkung:

Objekte	= alle Elemente einer Anwendung
Eigenschaften	= bestimmen Aussehen und Verhalten eines Objekts
Methoden	= verändern den Wert eines Objekts oder führen mit dessen Inhalt Aktionen durch

Eine Übersicht der Objekthierarchie finden Sie im Objektkatalog nach Öffnen der Word-Bibliothek und Anklicken des Eintrags *Application* im linken Listenfeld. Drücken Sie als nächstes die F1-Taste und klicken Sie in der danach öffnenden Hilfedatei den blau hinterlegten Eintrag *Application* an.



Variablen

Im nächsten Beispiel soll der Cursor mit Hilfe eines Makros von der aktuellen Position um 5 Zeilen nach unten bewegt werden. Fügen Sie in Ihr Dokument zunächst ein paar Zeilenschaltungen ein, damit ausreichend Platz für die folgenden Aktionen ist. Klicken Sie in die erste Zeile und starten Sie den Makrorekorder. Drücken Sie fünfmal die Cursortaste - nach unten - und beenden Sie das Makro wieder.
Im Visual Basic-Editor erscheint die folgende Codezeile, die sinngemäß folgendes bedeutet: Ab der ausgewählten Position (*Selection*) erfolgt eine Abwärtsbewegung (*MoveDown*) – zeilenweise (*wdLine*) – konstant um 5 Zeilen (*count*:=5).

```
Sub fünfmalabwärts()
    Selection.MoveDown unit:=wdLine, Count:=5
End Sub
```

Anmerkung:

Wie eingangs schon erwähnt, erhalten Sie nach Eingabe des Punktes nach dem Objektnamen *Selection* eine *Dropdown*-Liste mit den Methoden und Eigenschaften dieses Objektes angezeigt. Wenn Sie danach die Leertaste betätigen, wird mittels Quickinfo die restliche Syntax angezeigt und Sie können die noch fehlenden Argumente ergänzen.

Um sich im Text zu bewegen können Sie beispielsweise neben `MoveDown` noch die Methoden:

`MoveLeft`
`MoveRight`
`MoveUp`

anwenden. Neben dem Argument *wdLine*, welches gleichsam die Einheit für eine Zeile definiert, können auch:

`wdCharacter` - Zeichen
`wdWord` – Wort
`wdParagraph` – Absatz oder
`wdScreen` – Bildschirm
`wdStory` - Dokument

verwendet werden. Wenn Sie zusätzlich dem Argument *count* (Standardwert = 1) einen Wert – wie in der obigen Beispielsyntax (`Count:=5`) zu sehen – zuweisen, können Sie sich diese Anzahl Zeichen, Worte oder Absätze entsprechend 'bewegen'.

Die Methode *HomeKey* ist vergleichbar mit der Pos1-Taste und *EndKey* mit der Ende-Taste. So können Sie mit der Syntax:

```
Selection.Homekey unit:=wdLine
Selection.Endkey unit:=wdStory
```

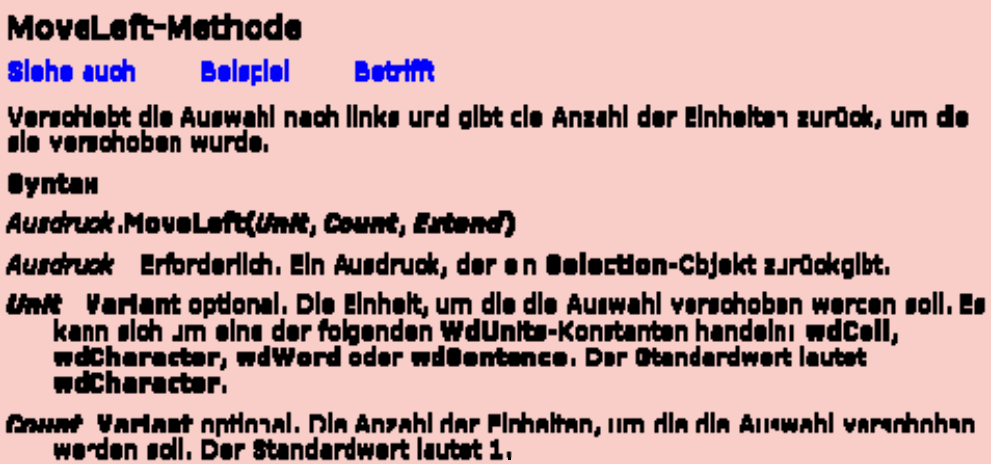
zum Beispiel den Cursor am Zeilenanfang und mit am Ende des Dokuments positionieren.

Um auf eine andere Seite zu wechseln, verwenden Sie die Methode *GoToNext* (nächste Seite) oder *GoToPrevious* (vorhergehende Seite) mit dem Argument *wdGoToPage*.

Übrigens... werden Argumenten nur Standardwerte übergeben, muss das Argument nicht explizit angeführt werden.

Erläuterung:

Geben Sie im Visual Basic-Editor die Syntax (`Selection.MoveLeft(unit:=wdCharacter, Count:=5)`) ein und klicken Sie anschließend in das Wort *MoveLeft.* Drücken Sie danach die F1-Taste und Sie erhalten die Hilfe zur Methode *MoveDown* angezeigt. Sie sehen unterhalb der Zeile *Syntax* den Code dieser Methode: *Ausdruck.MoveLeft(Unit, Count, Extend)* und in der darunter stehenden Erläuterung zu *Unit* den Hinweis, dass der Standardwert dieses Arguments: *wdCharacter* – also Zeichen ist. In der Erläuterung zu *Count* sehen Sie, dass der Standardwert: 1 ist. Wenn Sie also die Cursorposition nur im Rahmen der Standardwerte verschieben wollen, müssen Sie diesen Teil der Syntax nicht schreiben. Die Syntax lautet in diesem Fall nur: `Selection.MoveLeft`



MoveLeft-Methode

Siehe auch Beispiel Betrifft

Verschiebt die Auswahl nach links urd gibt cie Anzahl der Einheiten zurück, um die sie verschoben wurde.

Syntax

Ausdruck.MoveLeft(*Unit*, *Count*, *Extend*)

Ausdruck Erforderlich. Ein Ausdruck, der e n Selection-Cbjekt zurückgibt.

Unit Variant optional. Die Einheit, um die die Auswahl verschoben werden soll. Es kann sich um eine der folgenden WdUnits-Konstanten handeln: wdCell, wdCharacter, wdWord oder wdSentence. Der Standardwert lautet wdCharacter.

Count Variant optional. Die Anzahl der Einheiten, um die die Auswahl verschoben werden soll. Der Standardwert lautet 1.

Wenn die Anzahl der Zeilenbewegungen allerdings veränderbar sein soll, müssen die jeweils eingegebenen Werte einer sog. *Variablen* übergeben werden. Variablen können auch als Container interpretiert werden, die den jeweils eingegebenen Wert enthalten – im Gegensatz zu Konstanten, deren Wert unveränderbar ist.

Die Variable unseres Beispiels soll den Namen *anzahl* erhalten. Nunmehr wird im weiteren Verlauf nicht mehr auf einen bestimmten Wert Bezug genommen, sondern auf den jeweiligen Inhalt der Variablen *anzahl*. In seiner einfachsten Form löst der folgende Code die gestellte Aufgabe. Der Inhalt der Variablen *anzahl* wird zunächst der Funktion *InputBox*() zugewiesen und anschließend auch dem Argument: *count* der Methode: *MoveDown*.

```
Sub variabelabwärts()
anzahl = InputBox("Wie viele Zeilen abwärts?", "Office-Online-Schule")
Selection.MoveDown unit:=wdLine, Count:=anzahl
End Sub
```

Anmerkung:

Funktion: **InputBox**

Syntax: Variable = InputBox(Prompt, Title, Default, Xpos, Ypos, HelpFile, Context)

Erläuterung:

Variable	- Hier werden die Werte gespeichert, die von Ihnen eingegeben werden.
Prompt	- Der Text, der als Eingabeaufforderung angezeigt wird. Dieses Argument ist erforderlich, die übrigen Eingaben optional.
Title	- Überschrift der Eingabeaufforderung.
Default	- im Eingabefeld kann ein Standardwert angezeigt werden
Xpos, Ypos	- Position der InputBox auf dem Bildschirm – Standard ist horizontal zentriert
HelpFile, Context	- hier kann eine Hilfedatei für den Dialog definiert werden.

Damit wird der Cursor genauso viele Zeilen nach unten bewegt, wie in der *InputBox* eingegeben wurden.

Da Prozeduren im Allgemeinen etwas umfangreicher sind als das vorstehende Beispiel zeigt und in der Regel auch mehr als eine Variable enthalten, können Sie vom Programm prüfen lassen, ob beispielsweise alle Variablen deklariert wurden, oder ob es beim Benennen der Variablen zu Schreibfehlern kam. Ferner kann einer Variablen explizit ein Datentyp (z.B. Datum, Text etc.) zugewiesen werden, damit es in der Folge bei Falscheingaben zu entsprechenden Fehlermeldungen kommt. Eine Auflistung der verschiedenen Datentypen können Sie nach Anklicken des folgenden Buttons einsehen.

Datentypen-Übersicht

Anmerkung:

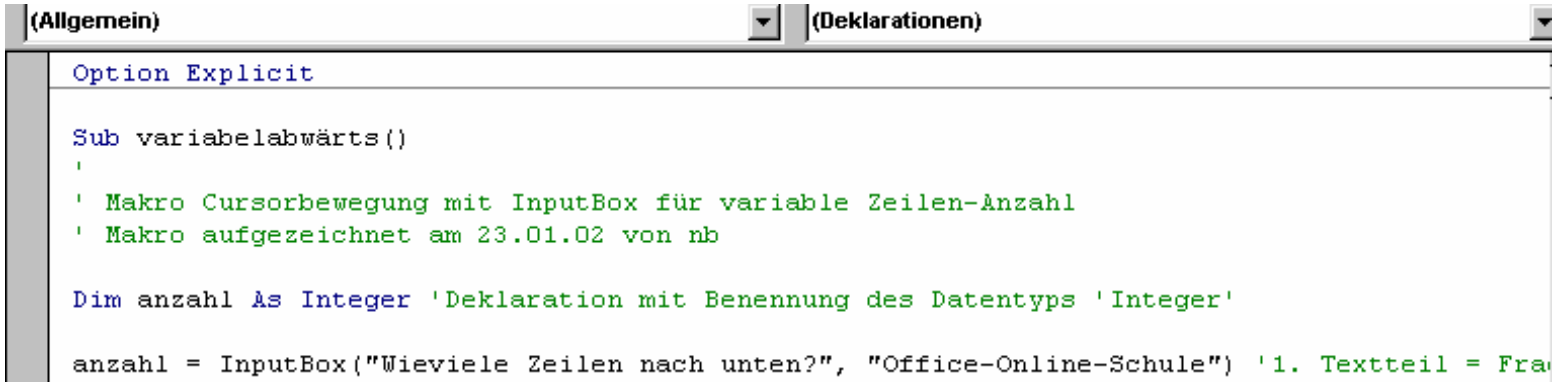
Versehen Sie Variablen mit einem eindeutigen Namen (AnzahlBesucher, Ergebnis_Verkauf_Sued, GesamtKfz etc.) Der Name kann 255 Zeichen lang sein, darf jedoch keine Leerzeichen, Punkte oder Typenbezeichnungen ($, %, & etc.), sowie Schlüsselwörter von VBA (*Dim, Sub, True* etc.) enthalten.

Deklaration von Variablen

Wechseln Sie zum Deklarieren der Variablen in den Visual Basic-Editor. Klicken Sie in das rechte obere Kombinationsfeld des Codefensters und öffnen Sie den Bereich *Deklarationen*. Geben Sie anschließend die Codezeile:

```
Option Explicit
```

ein. Wenn Sie im gleichen Feld nach einer Zeilenschaltung noch die Einträge (z. B. `Dim anzahl as Integer`) vornehmen, dann ist diese Variable nicht nur innerhalb eines Makros, sondern in allen Prozeduren Ihres Projekts gültig. In unserem nachstehenden Code-Beispiel (vgl. auch nachfolgenden Screenshot) wurde die Deklaration (`Dim anzahl as Integer`) nur in der Prozedur *variableabwärts()* vorgenommen, weshalb sie nun auch nur in diesem Makro gültig ist. In unserem Beispiel wurde nach dem Namen der Variablen (anzahl) noch der Datentyp (as `Integer`) zugewiesen.



Anmerkung:

Die Gültigkeit von Variablen kann auch noch in anderer Form bestimmt werden, nämlich mit den Anweisungen *Public* oder *Privat*. Mit *Public* werden die Variablen **allen** Prozeduren in **allen** Modulen zugänglich gemacht und mit *Privat* ausschließlich in dem Modul in dem sie deklariert wurde. Syntax:

```
Private Sub  Prozedurname()
Public Sub   Prozedurname()
End Sub
```

Datentypen definieren die Werte, die an bestimmten Stellen des Programms eingegeben werden können, seien es Zahlen, Buchstaben usw. Wird kein Datentyp angegeben - also nur: `Dim anzahl` geschrieben, weist VBA automatisch den Typ *Variant* zu. Der Nachteil dieser 'Universal-Deklaration', die praktisch für alle Datentypen steht, ist allerdings ein hoher Verwaltungsaufwand des Programms, verbunden mit einem höheren Speicherbedarf und einer reduzierten Verarbeitungsgeschwindigkeit.

Im übrigen ist die Entscheidung, ob Sie Datentypen definieren oder nicht auch davon abhängig, ob und wie Sie Ergebnisse anschließend weiterverarbeiten wollen. Aus Gründen der Übersichtlichkeit und zur Vermeidung von Fehlern empfiehlt es sich bei umfangreicheren Prozeduren, entsprechende Datentypen zuzuweisen.

Durch die vorstehende Deklaration der Variablen wird beim Kompilieren des Quellcodes (Übersetzung in binären Maschinencode – Menü: Debuggen – Kompilieren von Project) der Code auf mögliche Fehler überprüft und eine entsprechende Fehlermeldung angezeigt, wenn beispielsweise Variablen angesprochen werden, die nicht deklariert sind, oder Namen von Variablen falsch geschrieben wurden.

Der Code für das um die Variable `anzahl` erweiterte Beispiel hat nun folgendes Aussehen:

```
Sub variabelabwärts()
Dim anzahl as Integer
anzahl = InputBox("Wie viele Zeilen abwärts?", "Office-Online-Schule")
    Selection.MoveDown unit:=wdLine, Count:=anzahl
End Sub
```

Nach Deklaration der Variablen *anzahl* (im Beispiel mit der nicht zwingend notwendigen Angabe des Datentyps – *as Integer*), wird der Wert dieser Variablen der Funktion *InputBox()* zugewiesen. Wie schon erwähnt, sind Funktionen eigentlich das Gleiche wie Prozeduren mit dem Unterschied, dass sie einen Wert zurückgeben. Anschließend wird auch dem Argument: *count* der Methode: *MoveDown* der Inhalt der Variablen *anzahl* zugewiesen. Die Verwendung der Klammern im vorstehenden Beispiel macht deutlich, dass bei Methoden oder Prozeduren die **keinen** Wert zurückgeben – wie bei der Methode *MoveDown* - die Werte **ohne** Klammernpaar übergeben werden. Im Falle der Funktion *InputBox()* wird der Wert der Variablen *anzahl* übergeben, weshalb ein Klammernpaar erforderlich ist. Beachten Sie hierzu auch das nachfolgende Beispiel.

Die dritte Ergänzung unseres Beispiel-Codes soll dazu führen, dass mit Hilfe einer *MsgBox* angezeigt wird, um wie viele Zeilen der Cursor tatsächlich nach unten bewegt wurde.
Da es am Seitenende zwischen beabsichtigten und tatsächlichen Zeilensprüngen zu Abweichungen kommen kann, muss auch für die Anzahl der tatsächlichen Zeilensprünge eine Variable deklariert werden. Da diese Variable Werte des gleichen Wertebereichs aufnehmen soll wie die Variable *anzahl*, muss auch kein neuer Datentyp zugewiesen werden. Ergänzen Sie die Zeile mit dem Schlüsselwort *Dim* wie nachfolgend gezeigt mit der Variablen *ergebnis* und weisen Sie ihr den Wert der Methode *MoveDown* zu.

```
Sub variabelabwärts_meldung()
' Cursorbewegung abwärts mit InputBox für variable Eingabe der Zeilen-Anzahl
' und Rückgabe des Wertes der tatsächlichen Zeilensprünge.

Dim anzahl, ergebnis 'ohne Definition des Datentyps – VBA weist Typ Variant zu

anzahl = InputBox("Wie viele Zeilen abwärts?", "Office-Online-Schule")
ergebnis = Selection.MoveDown(wdLine,anzahl) 'alternative Kurzform-Schreibweise
    MsgBox "Cursorposition abwärts um: " & ergebnis & " Zeilen."
End Sub
```

Wie Sie sehen, werden nun die Werte der Methode *MoveDown* ebenfalls in ein Klammernpaar eingebunden, da sie jetzt als Funktion den Wert der Variablen *ergebnis* zurückgibt.

Anmerkung:

Beachten Sie die folgende Schreibweise bei der Zuweisung eines Datentyps:
Im vorstehenden Beispiel wurde mit:

```
Dim anzahl, ergebnis
```

den beiden Variablen automatisch der Datentyp *Variant* zugewiesen.

Wenn Sie beabsichtigen **beiden** Variablen beispielsweise den Datentyp *Integer* zuzuweisen, muss Ihre Deklaration folgendermaßen lauten:

```
Dim anzahl as Integer, ergebnis as Integer
```

Die Schreibweise:

```
Dim anzahl, ergebnis as Integer
```

führt dazu, dass die Variable anzahl den Datentyp Variant erhält und die Variable ergebnis den Datentyp Integer!

Anmerkung:

Funktion: **MsgBox**

Syntax – mit Rückgabe eines Wertes:
Variable = MsgBox(Prompt, Buttons, Title, HelpFile, Context)

Syntax – ohne Rückgabe eines Wertes:
MsgBox Prompt, Buttons, Title, HelpFile, Context

Erläuterung:

Variable	- Hier wird der Wert der Schaltfläche gespeichert, die von Ihnen angeklickt wird.
Prompt	- Der Text, der als Information angezeigt wird.
Buttons	- Hier werden die Schaltflächen und das Symbol der MsgBox festgelegt.
Title	- Überschrift der MsgBox.
HelpFile, Context	- hier kann eine Hilfedatei für den Dialog definiert werden.

Eine Auflistung aller bestehenden Button-Konstanten finden Sie in der VBA-Hilfe zur Funktion *MsgBox*.

Datenfelder (Arrays)

Die bisherigen Ausführungen zum Thema *Variablen* bezogen sich auf einzelne Variablen, in denen man verschiedene Werte mit unterschiedlichen Datentypen speichern kann. Nun geht es darum, mehrere Variablen vom gleichen Typ zusammenzufassen und die Inhalte der einzelnen *Datenfelder* mit Hilfe einer Indexzahl auszulesen. Da es sich bei *Datenfeldern* - *Arrays* – auch um Variablen handelt, werden sie genau wie diese mit einer *Dim*-Anweisung deklariert, allerdings mit dem Unterschied, dass grundsätzlich die untere und/oder die obere Grenze des Datenfeldes in Klammer geschrieben wird.

Beispiel: `Dim Monat (1 To 12) As Integer`

Diese Anweisung bedeutet sinngemäß, dass ein Datenfeld mit Namen *Monat* und 12 verschiedenen Elementen vom Datentyp *Integer* deklariert wird. Die Deklaration kann auch in der Form erfolgen, dass für das *Array* nur die Obergrenze deklariert wird.

Beispiel: `Dim Monat (12) As Integer`

Allerdings kann diese Schreibweise zu Irritationen führen, denn das erste Datenfeld erhält immer den Index Null. Das heißt, die Untergrenze des Arrays beginnt immer mit Null, was bedeutet, dass der zwölfte Monat mit der Indexzahl 11 identisch ist.

Mit Hilfe der *Option-Base*-Anweisung können Sie im allgemeinen Deklarationsbereich eines Moduls die Untergrenze von Arrays festlegen.

Beispiel: `Option Base 1` 'Index-Untergrenze kann Null oder Eins sein. Hier ist sie Eins.

Unabhängig von dieser generellen Einstellung können Sie aber trotzdem für bestimmte Arrays in einzelnen Prozeduren abweichende Deklarationen, beispielsweise mit Unter- und Obergrenze, vornehmen.

Konstanten:

Der Einsatz von Konstanten erfolgt in Word ähnlich der von Variablen. Das hauptsächliche Unterscheidungsmerkmal ist, dass bereits bei der Deklaration der Konstanten ein Wert zugewiesen wird, der anschließend nicht mehr verändert werden kann. Die Deklaration erfolgt am Anfang einer Prozedur – noch vor der Deklaration der Variablen. Allerdings beginnt die Deklaration nicht mehr mit dem Schlüsselwort `Dim` wie bei den Variablen, sondern mit `Const`. Beispiel: `Const MwStSatz = 0.16`. Außerdem ist es möglich, wie bei der Deklaration von Variablen, einen entsprechenden Datentyp (`Const MwStSatz as Single = 0.16`) zuzuweisen.
VBA arbeitet übrigens auch mit internen Konstanten. Diese Werte sind im Programm, ähnlich den bereits vorhandenen Funktionen definiert und geben eine Einheit wieder. Sie beginnen mit dem Präfix *wd* und beschreiben, wie im letzten Beispiel (*wdLine*) bereits zu sehen, in Verbindung mit der Methode *MoveDown*, die Einheit: Zeile.

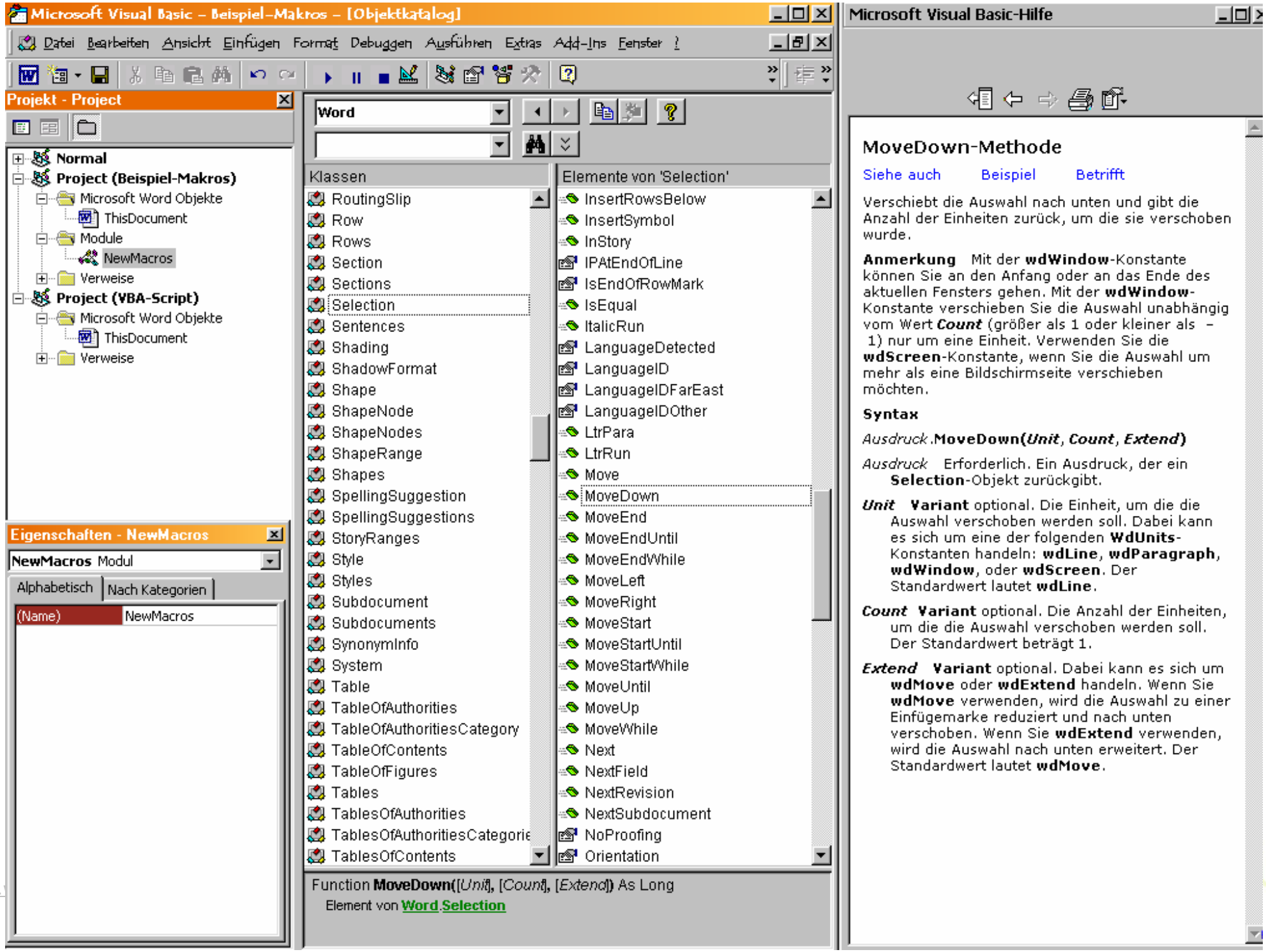
Noch ein Hinweis zur Verwendung des Objektkataloges...

...anhand der '*Selection.MoveDown*...' – Befehlszeile des letzten Beispielcodes.

Nachdem Sie den Objektkatalog über das Menü *Ansicht* oder durch Anklicken des entsprechenden Icons geöffnet haben, können Sie im oberen Bereich (Standard: <alle Bibliotheken>), beispielsweise die Bibliothek für die Word-Anwendung auswählen. Im linken Listenfeld des Objektkataloges sehen Sie unterhalb der Überschrift *Klassen* eine Auflistung der Word-Objekte. Im rechten Bereich sind die Elemente, also Eigenschaften und Methoden des jeweiligen Objekts aufgelistet. Scrollen Sie nun im linken Listenfeld abwärts bis Sie den Eintrag *Selection* finden, klicken Sie dieses Wort an und anschließend die F1-Taste. Es wird eine Hilfedatei geöffnet, die Ihnen Informationen zum Objekt *Selection* anzeigt. Wenn Sie diese Hilfedatei wieder schließen sehen Sie im rechten Listenfeld die Methoden und Eigenschaften des Objekts *Selection* aufgelistet. Scrollen Sie auch in diesem Feld nach unten bis Sie den Eintrag *MoveDown* finden. Auch hier können Sie sich nach Drücken der F1-Taste detaillierte Informationen zu dieser Methode anzeigen lassen und auch Beispiel-Codes ansehen. Im unteren Bereich des Objektkataloges wird Ihnen unterhalb der Klassen- und Elemente-Listen die Syntax der Function *MoveDown([Unit], [Count], [Extend])* As Long – mit Datentyp und den erforderlichen Argumenten angezeigt. Sie können sich auch hier durch Anklicken und Drücken der F1-Taste wieder gezielt Informationen auch zu diesen Positionen anzeigen lassen.

Auf die gleiche Weise können Sie sich auch die Bibliothek Ihres Projektes anzeigen lassen. Klicken Sie hierzu in der Bibliothekenauflistung auf den Namen Ihres Projekts und Sie sehen neben den verwendeten Objekten auch die möglichen Eigenschaften und Methoden, sowie im unteren Bereich die entsprechende Syntax des Objekts als VBA-Code angezeigt.

Zusätzliche Hinweise zu diesem Thema finden Sie auch in der Datei: Makro-Beispiele und in den ausführlichen Kommentierungen des jeweiligen Beispielcodes im Visual Basic-Editor.


Microsoft Visual Basic – Beispiel-Makros – [Objektkatalog]
Datei Bearbeiten Ansicht Einfügen Format Debuggen Ausführen Extras Add-Ins Fenster ?
Projekt - Project
Normal
Project (Beispiel-Makros)
Microsoft Word Objekte
ThisDocument
Module
NewMacros
Verweise
Project (VBA-Script)
Microsoft Word Objekte
ThisDocument
Verweise
Eigenschaften - NewMacros
NewMacros Modul
Alphabetisch
Nach Kategorien
(Name)
NewMacros
Word
Klassen
RoutingSlip
Row
Rows
Section
Sections
Selection
Sentences
Shading
ShadowFormat
Shape
ShapeNode
ShapeNodes
ShapeRange
Shapes
SpellingSuggestion
SpellingSuggestions
StoryRanges
Style
Styles
Subdocument
Subdocuments
SynonymInfo
System
Table
TableOfAuthorities
TableOfAuthoritiesCategory
TableOfContents
TableOfFigures
Tables
TablesOfAuthorities
TablesOfAuthoritiesCategorie
TablesOfContents
Elemente von 'Selection'
InsertRowsBelow
InsertSymbol
InStory
IPAtEndOfLine
IsEndOfRowMark
IsEqual
ItalicRun
LanguageDetected
LanguageID
LanguageIDFarEast
LanguageIDOther
LtrPara
LtrRun
Move
MoveDown
MoveEnd
MoveEndUntil
MoveEndWhile
MoveLeft
MoveRight
MoveStart
MoveStartUntil
MoveStartWhile
MoveUntil
MoveUp
MoveWhile
Next
NextField
NextRevision
NextSubdocument
NoProofing
Orientation
Function MoveDown([Unit], [Count], [Extend]) As Long
Element von Word.Selection
Microsoft Visual Basic-Hilfe
MoveDown-Methode
Siehe auch
Beispiel
Betrifft
Verschiebt die Auswahl nach unten und gibt die Anzahl der Einheiten zurück, um die sie verschoben wurde.
Anmerkung Mit der wdWindow-Konstante können Sie an den Anfang oder an das Ende des aktuellen Fensters gehen. Mit der wdWindow-Konstante verschieben Sie die Auswahl unabhängig vom Wert Count (größer als 1 oder kleiner als –1) nur um eine Einheit. Verwenden Sie die wdScreen-Konstante, wenn Sie die Auswahl um mehr als eine Bildschirmseite verschieben möchten.
Syntax
Ausdruck.MoveDown(Unit, Count, Extend)
Ausdruck Erforderlich. Ein Ausdruck, der ein Selection-Objekt zurückgibt.
Unit Variant optional. Die Einheit, um die die Auswahl verschoben werden soll. Dabei kann es sich um eine der folgenden WdUnits-Konstanten handeln: wdLine, wdParagraph, wdWindow, oder wdScreen. Der Standardwert lautet wdLine.
Count Variant optional. Die Anzahl der Einheiten, um die die Auswahl verschoben werden soll. Der Standardwert beträgt 1.
Extend Variant optional. Dabei kann es sich um wdMove oder wdExtend handeln. Wenn Sie wdMove verwenden, wird die Auswahl zu einer Einfügemarke reduziert und nach unten verschoben. Wenn Sie wdExtend verwenden, wird die Auswahl nach unten erweitert. Der Standardwert lautet wdMove.