

1 Automatisierung , Programmierung mit Word	4
1.1 Aufbau und Allgemeininformationen	4
Aufbau und inhaltliche Konventionen des Skriptes	4
Typografische Konventionen	4
Symbole	4
Einleitung	4
Hinweise zu Funktionalitäten	5
1.2 Automatisierungsmöglichkeiten mit Word 2013	5
Welche Arbeiten können automatisiert werden?	5
Makros aufzeichnen	5
Makronamen festlegen	5
Arbeitsschritte aufzeichnen	6
Makroaufzeichnung beenden	6
Makro speichern	7
Makros ausführen, löschen und anzeigen	8
2 Grundlagen zum Visual Basic-Editor	9
2.1 Aufrufen und Einrichten	9
2.2 Makros bearbeiten und verwalten	12
Aufbau des Makro-Codes	12
Module bzw. Makros kopieren	13
Module exportieren	13
Module importieren	13
2.3 Aufruf eines Makros über Symbolleisten	13
3 VBA Grundlagen	15
3.1 Word-Objektmodell	15
3.2 Eigenschaften	16
3.3 Methoden	17
3.4 Ereignisse	17
3.5 Die VisualBasic-Syntax	17
Punkt-Syntax	18
Runde Klammern	18
Groß- und Kleinbuchstaben	19
Kommentare	20
Schlüsselwörter	20
Konstanten	21
Datentypen	22
Variablen	25
Bezeichnen einer Variablen	27
Gültigkeitsbereich einer Variablen	27
Typumwandlungen	28
Arrays	29
Eindimensionale Arrays	29
Datenfeldern	30
mehrdimensionale Datenfelder (mehrdimensionale Arrays)	31
Operatoren	32
Arithmetische Operatoren	32

Vergleichsoperatoren.....	32
Logische Operatoren	33
Verkettungsoperatoren	34
Operatorvorrang	34
Kontrollstrukturen	35
Fallunterscheidungen.....	35
Select Case-Anweisung.....	36
Schleifen	37
For Each . . .Next-Anweisungen.....	37
For . . .Next-Anweisungen.....	38
Do . . .Loop- Anweisung.....	39
With-Anweisung.....	39
Funktionen	40
Vorteile von Funktionen	40
Definition einer Funktion	41
Sub-Anweisung.....	41
3.6.1 Maßeinheiten.....	42
3.6.2 den Cursor in Word-Dokumenten bewegen.....	44
Die Methode GoTo	44
Die Methoden HomeKey und EndKey.....	45
Die Methoden Movexxx	45
Die Methoden StartOf und EndOf.....	46
3.6.3 Word-Dokument markieren	46
Die Methode MoveEnd.....	48
Die Methode WholeStory.....	48
Die Methode Select	48
3.6.4 Das Range-Objekt	48
Die Range-Methode.....	49
Die Range -Eigenschaft.....	49
Selection oder Range?.....	50
3.6.5 Word-Befehle verwenden.....	50
Texte eingeben und Bearbeiten	50
Absatzmarke einfügen	50
Seitenumbruch	50
Sonderzeichen	51
Text löschen.....	51
Text überschreiben.....	51
Texte formatieren.....	51
Zeichenformate	51
Absatzformate.....	52
Zeichen- als auch Absatzformate zurücksetzen	52
Formatvorlagen.....	52
Rahmen und Schattierungen	52
Seitenrahmen	53
Aufzählungen und Nummerierungen.....	53
3.6.6 Word Textmarke	53

Textmarken einfügen.....	53
Textmarke wieder finden.....	53
3.6.7 Mit Tabellen arbeiten	54
Tabelle füllen.....	54
3.6.8 Fußnoten und Endnoten einfügen.....	55
3.6.9 Kopf- und Fußzeilen einfügen	55
4 Dialogblatt (Userformblatt)	55
4.1 MsgBox.....	56
4.2 Die Methode InputBox	58
4.3 Steuerelemente aus <i>Selbstdefinierte Dialoge - Userformblätter</i>	59
4.3.1 Einfügen der Elemente ins Userformblatt.....	60
4.3.2 Eigenschaften der Elemente	61
4.3.3 Das Bezeichnungsfeld (Label)	63
4.3.4 Schaltflächen, Wechselschaltflächen (CommandButton, ToggleButton)	63
4.3.5 Textfelder (TextBox).....	64
4.3.6 Listen, Kombinationsfelder (ListBox, ComboBox).....	65
4.3.7 Drehfelder, Laufleisten (SpinButton,ScrollBar)	65
4.3.8 Kontrollkästchen, Optionsfelder (CheckBox, OptionButton)	66
Aufgabe A1: Automatisierung und Programmierung mit Word 2013 Professional	67
Allgemeines zu dem Hintergrund der Aufzinsung	67
Vorgehensweise.....	67
Erläuterung zu den Steuerelementfunktionen	68
Erläuterung zu dem Programmablauf.....	69
LITERATUR	70
VBA-Codes für die Aufzinsung.....	71
Aufgabe A2: Automatisierung und Programmierung mit Word 2013 Professional	80
Allgemeines zu dem Hintergrund	80
Vorgehensweise.....	80
Erläuterung zu den Steuerelementfunktionen	82
Erläuterung zu dem Programmablauf.....	83
LITERATUR	84
VBA-Codes für die Aufzinsung.....	85

1 Automatisierung , Programmierung mit Word

1.1 Aufbau und Allgemeininformationen

Aufbau und inhaltliche Konventionen des Skriptes

- ✓ Am Anfang jedes Kapitels finden Sie eine kurze Information über die allgemeine Einführung zu dem Programmbeispiel mit wichtigen Funktionen.
- ✓ Die meisten Kapitel enthalten das Programmbeispiel, mit dessen Hilfe praktische Übungen ausgeübt werden können.
- ✓ Das Ziel dieses Skriptes ist einen schnelleren Einstieg in Excel zu ermöglichen.

Typografische Konventionen

Im Text sind drei Schriftarten

- Calibri: für den Satz des Skriptes
- Adobe Arabic: für alle zugewiesenen Namen wie Abbildungen, Tabellennamen, Formularnamen, Steuerelementen usw.
- Courier New: für Programmcodes

angewandt.

Symbole



Besondere Informationen, die Ihnen weiterhelfen können.



Besondere Hinweise oder Tipps



Warnhinweise oder besondere Aufmerksamkeit

Einleitung

Dieses Skript wurde für einen Kurs des Studienganges „Bekleidung- Technik und Management Lehrstuhl Elektronische Datenverarbeitung“ erstellt. Deshalb sind die Beispiele darauf zugeschnitten. Die Beispieldateien sowie die in den Übungen verwendeten Übungs- und Ergebnisdateien können Sie im Abschnitt „Aufgaben A1-A5“ vollständig finden.

Außerdem war nicht Vollständigkeit gefragt, sondern es sollte um die Formulare und VBA (*Visual Basic for Application*) der Office-Anwendung „Word 2013“ gehen. Grundlegende Kenntnisse wie z.B. Arbeiten mit Absätzen, Formatvorlagen, Schriftarten, sowie Formatieren von Tabellen, Anpassen der Wordkonfigurationen und ähnliches werden hier vorausgesetzt. In diesem Skript lernen Sie die folgenden Aufgaben zu meistern:

- Hinweise zur Funktionalitäten der Software
- Erstellen und Implementieren von Aufgaben aus den Bereichen Serienbrief- und Artikelkalkulation, Zinseszinsrechnung, usw.
- Umgang mit den Tabellenblättern und Import und Export mit den Excel-Tabellen
- Umgang mit den Steuerelementen zur Steuerung der Daten

- Umgang mit einem Userformblatt, um die Daten z.B. von einer Tabelle in eine Worddokument zu übertragen
- Umgang mit VBA-Syntax (u.a. Variablen und deren Datentypen, Gültigkeitsbereich einer Variable, Fallunterscheidungen, Schleifen, Konstanten, Operatoren, eindimensionalen Arrays, benutzerdefinierten Prozeduren)
- Konzeption und Realisation der Algorithmen zu der Aufgabestellung

Hinweise zu Funktionalitäten

Word ist ein Textverarbeitungsprogramm und verfügt über VBA Funktionen, die beliebigen Daten aus den anderen Programmen wie z.B. Excel zu importieren und Ästhetik flexibel darzustellen. Es verfügt nicht über die Funktionalitäten des weitverbreiteten Konzepts der relationalen Datenbank. Es kann jedoch über Schnittstellen auf Daten aus Datenbanken zugegriffen werden und somit die Daten in einer gestalterischen Form für Printmedien zur Verfügung zu stellen. In diesem Skript behandelte Aufgaben geht es darum, das prinzipielle Konzept der Programmiersprachen kennenzulernen.

1.2 Automatisierungsmöglichkeiten mit Word 2013

Welche Arbeiten können automatisiert werden?

Visual Basic for Application, so lautet der ausgeschriebene Name für die Makrosprache der Microsoft Anwendungen. Damit kann man häufig wiederkehrende Befehlsabfolgen automatisieren, Bedingungen einfügen, Schleifen durchlaufen lassen und vieles mehr. Wenn Sie z.B. die Schriftart, den Schriftgrad und die Schriftfarbe einer Zeichenkette ändern wollen, können Sie ein Makro einsetzen, um Ihre Arbeit schneller und einfacher zu erledigen.

Makros aufzeichnen

Um ein Makro aufzuzeichnen, brauchen Sie keine VBA-Programmierkenntnisse. Sie benötigen das automatische Aufzeichnen, indem Sie im Register „Ansicht“ in der Gruppe „Makros“ auf den Pfeil des Symbols „MAKROS“ klicken und den Eintrag „MAKRO AUFZEICHNEN...“ in der geöffneten Liste wählen. Nach dem Sie einen Makronamen eingeben haben, wird jeder Arbeitsschritt, den Sie während der Aufzeichnung des Makros ausführen (z.B. Schriftart, Schriftfarbe und Schriftgrad eines Textes zu ändern) wird automatisch sogenannter VBA-Code generiert.

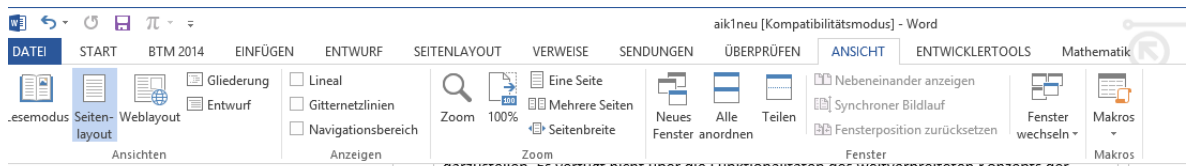


Abb. 1.2.1 Makro aufzeichnen

Makronamen festlegen

Ein Makro beinhaltet eine Folge von automatisch abzuarbeitenden Aufgaben, die in Form der VBA-Anweisungen vorkommen. Ein Makro muss einen Namen haben, mit dem die Aufgaben

ausgeführt bzw. aufgerufen wird. Wird ein Makro Abb. 1.2.1 Makro aufzeichnen. Erstellt, erscheint ein Dialogfenster Abb. 1.2.2 Makroeigenschaften festlegen.

- Geben Sie im Textfeld bei *Makroname* einen Namen für Ihr Makro. Dabei sind auf die folgenden Regeln zu beachten:

- ✓ Der Makroname muss mit einem Buchstaben beginnen.
- ✓ Sonderzeichen sind nicht erlaubt.

- Wenn Sie das Makro später über eine Tastenkombination ausführen wollen, drücken Sie eine Taste ins Textfeld bei *Tastenkombination*.

- Achten Sie darauf, wo Ihr Makro gespeichert wird. Meistens will man das Makro in demselben Dokument speichern. Wählen Sie Ihre Auswahl im Kombinationsfeld bei *Makro speichern in:*

- Zweck des Makros wird in das Textfeld bei *Beschreibung* eingetragen.
- Anschließend bestätigen Sie die Schaltfläche OK, um Ihre Makroaufzeichnung zu starten.

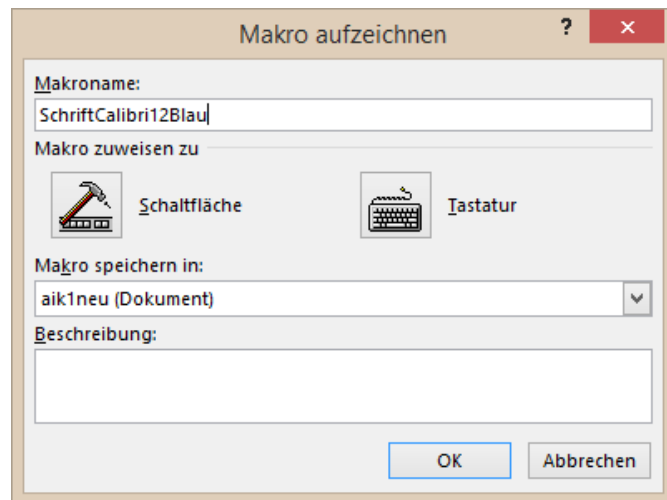



Abb. 1.2.2 Makroeigenschaften festlegen

Arbeitsschritte aufzeichnen

Nachdem Sie in Abb. 1.2.2 Makroeigenschaften festlegen auf OK geklickt haben, wird Word im Aufzeichnungsmodus befindet und erscheint das Symbol  in der Statusleiste.



Wenn Sie ein Makro aufzeichnen und beibehalten wollen, müssen Sie das entsprechende Dokument speichern. Während der Aufzeichnung dürfen Sie den Programmbereich nicht verlassen. Dabei wird jeder Arbeitsschritt aufgenommen.



Makros müssen in Word 2013 im Dateiformat WORD MIT MAKROS gespeichert werden. Als Dateinamenerweiterung wird .docm verwendet.



Wenn Sie die Arbeitsschritte nicht genau kennen, testen Sie vorher ohne Aufzeichnung.

Makroaufzeichnung beenden

Zum Beenden Ihres Makros können Sie entweder

- Auf das Symbol  klicken oder

- Im Register ANSICHT in der Gruppe MAKROS in der Liste auf den Eintrag AUFZEICHNUNG BEENDEN klicken.

Makro speichern

Wenn Sie ein Makro aufzeichnen wollen, achten Sie darauf, wo das Makro gespeichert wird.

Makros werden standardmäßig innerhalb eines Dokumentes gespeichert. Wie es im Abschnitt „Makronamen festlegen“ beschrieben wurde.

Sollte der Makroname in einem Dokument existieren, meldet Word ein Fenster (Abb. 1.2.3), in dem Sie aufgefordert werden, zu ersetzen.

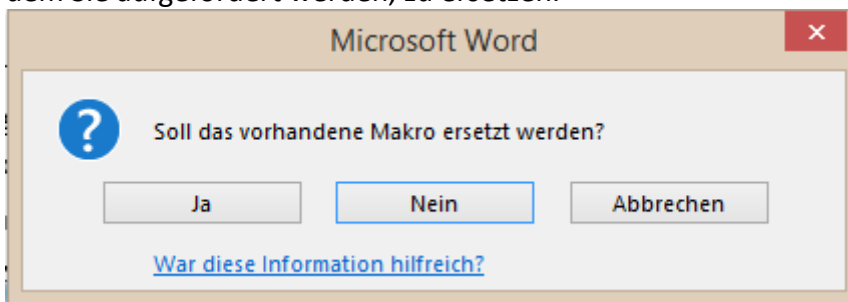


Abb. 1.2.3 Rückfrage zum Speichern von Makros in dem persönlichen Dokument

Damit Ihre Makros endgültig in dem Dokument gespeichert wird, gehen Sie wie folgt vor:

- wählen Sie in Menü DATEI den Menüpunkt SPEICHERN UNTEN. Darauf erscheint das Fenster (Abb. 1.2.4).
- Wählen Sie für den Dateityp „Excel-Arbeitsmappe mit Makros.“
- Geben Sie im Listenfeld DATEINAME der gewünschten Dateinamen ein.
- Bestätigen Sie mit der Schaltfläche SPEICHERN.

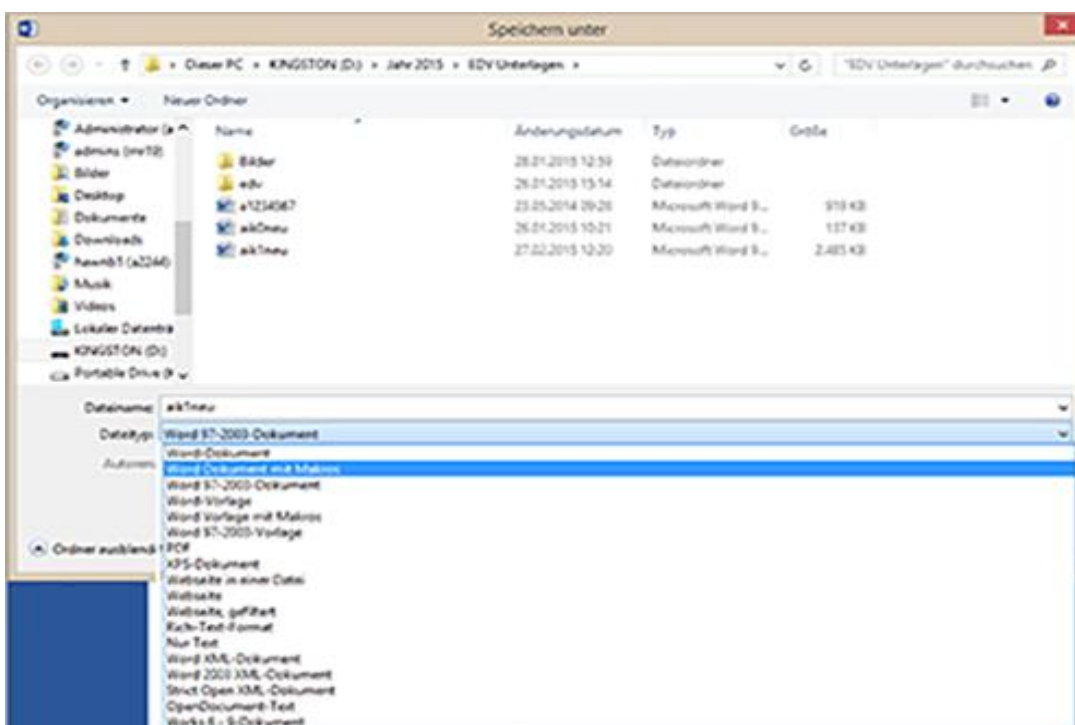


Abb. 1.2.4 Dialogfenster zum Datei speichern

Makros ausführen, löschen und anzeigen

Beim Ausführen von Makros müssen Sie darauf achten, dass Makros aus vertrauenswürdigen Quellen entstanden wurde. Denn Makros können Viren enthalten. Standardmäßig werden Makros in Word unterdrückt, wenn Sie ein Dokument mit gespeicherten Makros öffnen. In diesem Fall erhalten Sie eine Sicherheitswarnung (Abb. 1.2.5), wenn Sie noch nicht die Einstellungen für Makros auf ALE MAKROS AKTIVIEREN (Abb. 1.2.6) eingestellt haben.

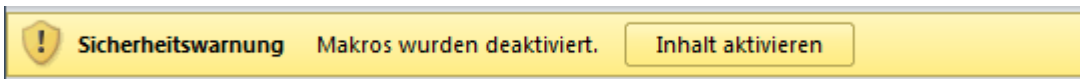


Abb. 1.2.5 Sicherheitswarnung für die Makroaktivierung



Bestätigen Sie die Schaltfläche INHALT AKTIVIEREN nur, wenn Sie der Herkunft der Makros vertrauen. Nach Bestätigung wird die Datei dauerhaft zu einem vertrauenswürdigen Dokument erklärt und die nachfolgend beschriebenen Wahlmöglichkeiten stehen Ihnen dann nicht mehr zur Verfügung.

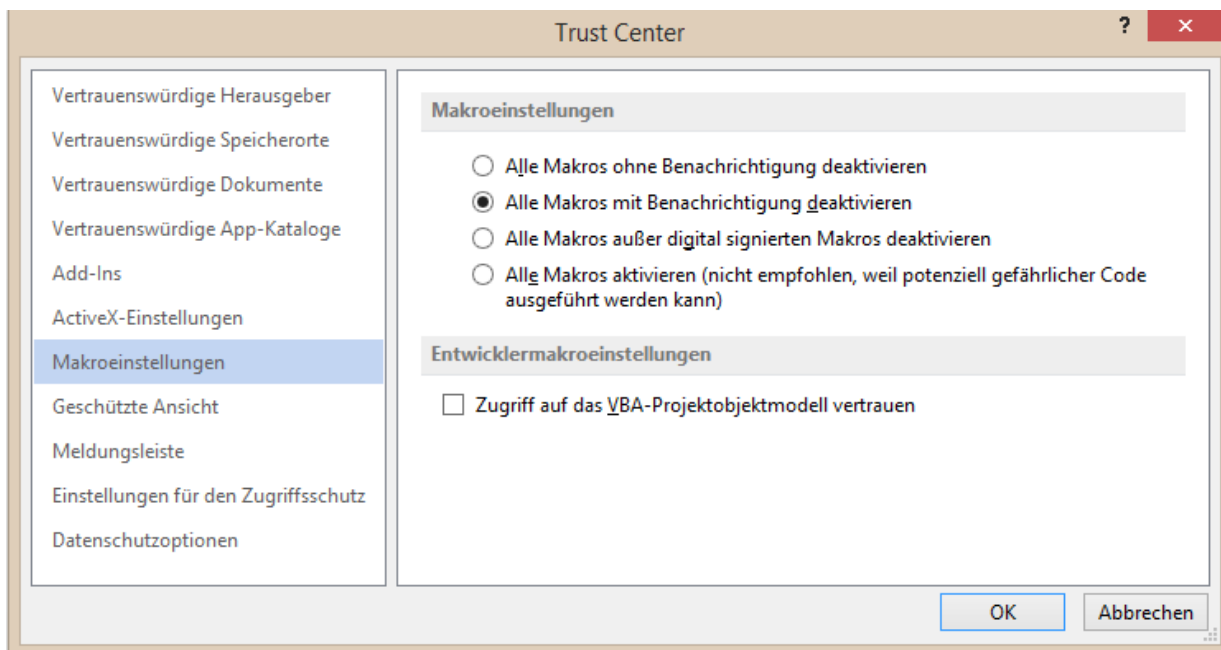


Abb. 1.2.6 Sicherheitswarnung für die Makroaktivierung

Wenn Sie der Herkunft der Makros vertrauen, können Sie auf das Optionsfeld ALLE MAKROS AKTIVIEREN (Abb. 1.2.6) klicken. Um zu diesem Optionsfeld gelangen, gehen Sie wie folgt vor:

- Wählen Sie WEITERE BEFEHLE in SYMBOLLEISTE FÜR DEN SCHNELLZUGRIFF ANPASSEN.
- Wählen Sie dann im Menüpunkt TRUST CENTER die Schaltfläche EINSTELLUNGEN FÜR DAS TRUST CENTER.
- Klicken Sie anschließend auf den Menüpunkt EINSTELLUNGEN FÜR MAKROS und wählen Sie das Optionsfeld ALLE MAKROS AKTIVIEREN.

Nach dem Sie Makros-Einstellungen vorgenommen haben, können Sie im Register ANSICHT in der Gruppe MAKROS auf den Pfeil des Symbols MAKROS klicken und den Eintrag MAKRO ANZEIGEN in der geöffneten Liste wählen. Dann erscheint das Dialogfenster (Abb. 1.2.7).

In diesem Dialogfenster haben Sie die Möglichkeiten, ein Makro ausführen, löschen oder auch bearbeiten bzw. anzeigen.

Wenn Sie ein Makro auswählen und auf die Schaltfläche AUSFÜHREN klicken, wird das Makros ausgeführt. Sollte ein Fehler in Makro vorkommen, wird der Fehler im Editor (Siehe im Abschnitt „der Visual Basic Editor“) angezeigt.

Wenn Sie ein Makro löschen wollen, klicken Sie auf die Schaltfläche LÖSCHEN.

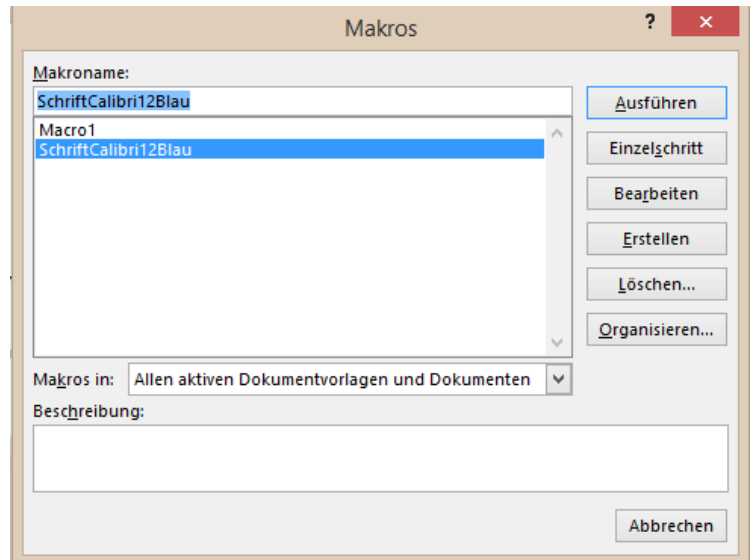


Abb. 1.2.7 Dialogfenster MAKRO ANZEIGEN

Das Makro wird aus Ihrem Word-Projekt für immer gelöscht, wenn Sie Ihr Dokument dann speichern. Wenn Sie die Codes eines Makros anzeigen und bearbeiten wollen, klicken Sie auf die Schaltfläche BEARBEITEN. In diesem Fall wird der Visual Basis Editor angezeigt.

2 Grundlagen zum Visual Basic-Editor

2.1 Aufrufen und Einrichten

In allen Microsoft Anwendungen, also auch in Word, kann man mit der Tastenkombination ALT + F11 den Editor aufrufen. Alternativ geht das auch mit der Befehlsfolge **ENTWICKLERTOOLS** und *Visual Basic Editor*. Es öffnet sich ein *eigenes Fenster*, das nicht Bestandteil des Word-Bildschirms ist. Das Bild, das sich dann bietet, ist noch nicht besonders arbeitsfreundlich, deshalb sollte man etwas über die drei wichtigsten Teile des Fensters wissen.

Der **Projekt-Explorer** sitzt meistens oben links in der Ecke und zeigt wie in einem Verzeichnisbaum die geöffneten Dateien, wobei jede für ein Projekt steht. Unterhalb dieser Ebene sieht man die jeweils zugehörigen Objekte. Das können Module sein, Formulare, Klassenmodule oder auch Tabellenblätter. Jedes Objekt aus Tabellen und Formularen kann aus den Objektteil und Programmcode bestehen. In diesem Skript werden hauptsächlich

Das **Eigenschaftsfenster** sitzt direkt darunter und zeigt die verschiedenen einzustellenden Eigenschaften der jeweiligen Objekte an. Die Graphik zeigt die Eigenschaften eines Formulars, wobei man hier mal das erste Fremdwort lernen sollte, nämlich USERFORM. So heißen diese Objekte in VBA. Falls eines dieser beiden Fenster nicht zu sehen ist, kann man sie über **Ansicht Projekt-Explorer** bzw. **Eigenschaftsfenster** anschalten

Programmcodes in Formularen behandelt.

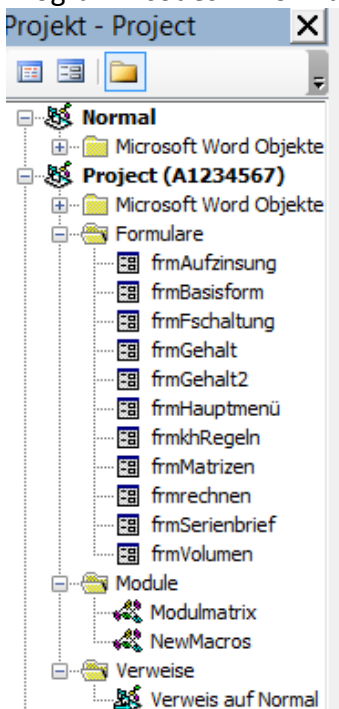


Abb. 2.1.1 Projekt-Explorer

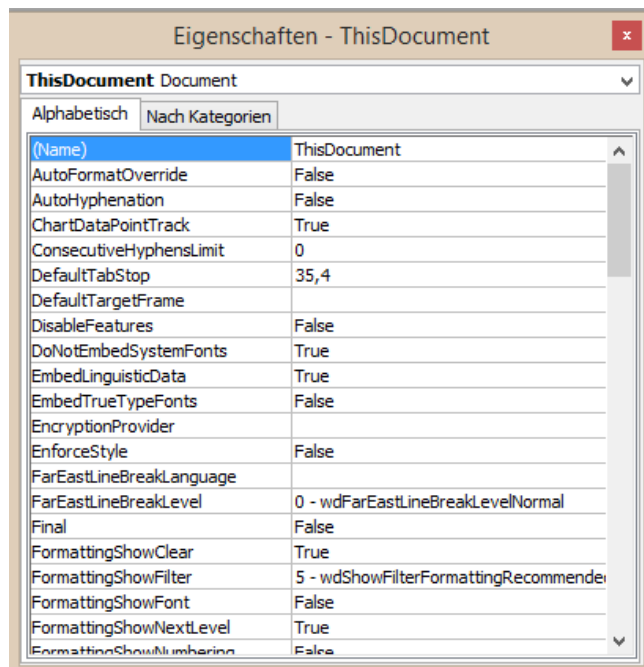


Abb. 2.1.2 Eigenschaftsfenster

Rechts davon, den größten Teil des Bildschirms (Abb. 2.1.3) einnehmend, findet das sogenannte **Modul-fenster** seinen Platz. Je nach dem, was man als Objekt im Projekt-Explorer markiert hat, sieht man dort Programmcode oder ein User form (Objektteil).

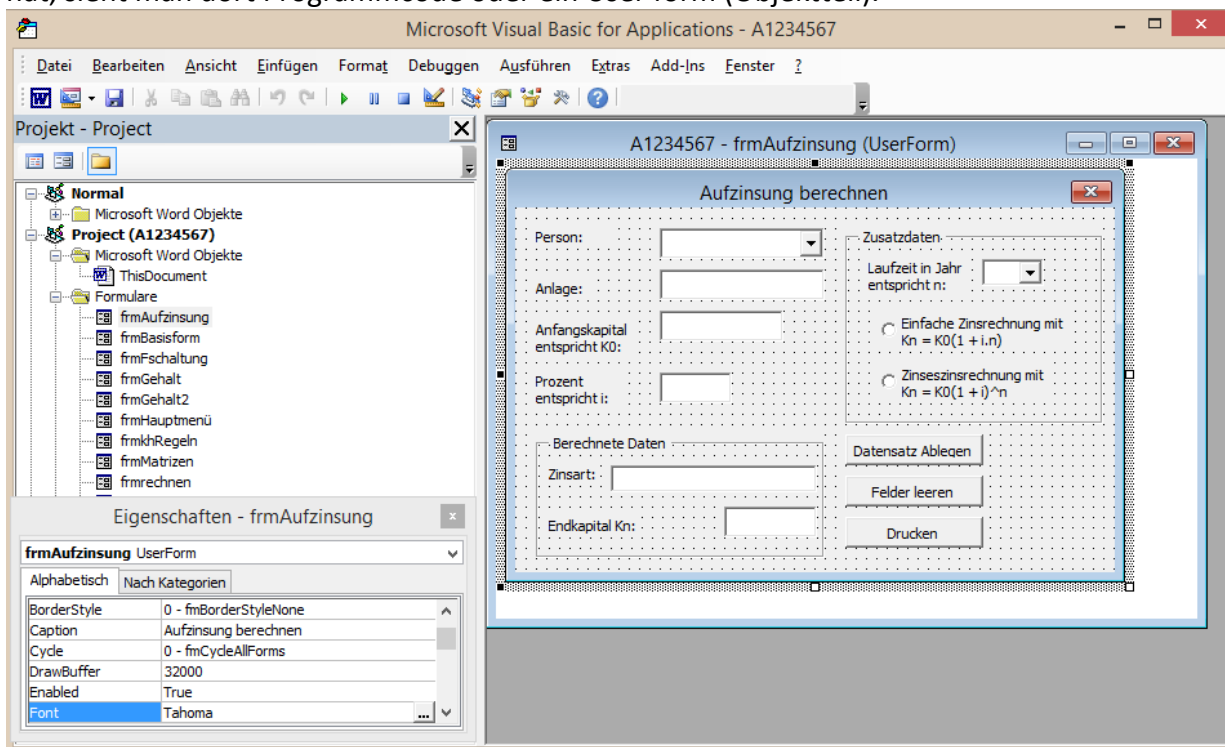


Abb. 2.1.3 Objektansicht von dem Userformblatt z.B. „frmAufzinsung“

Bei diesem Beispiel ist links das Userformblatt mit dem Namen „frmAufzinsung“ markiert und rechts ist es als Graphik zu sehen.

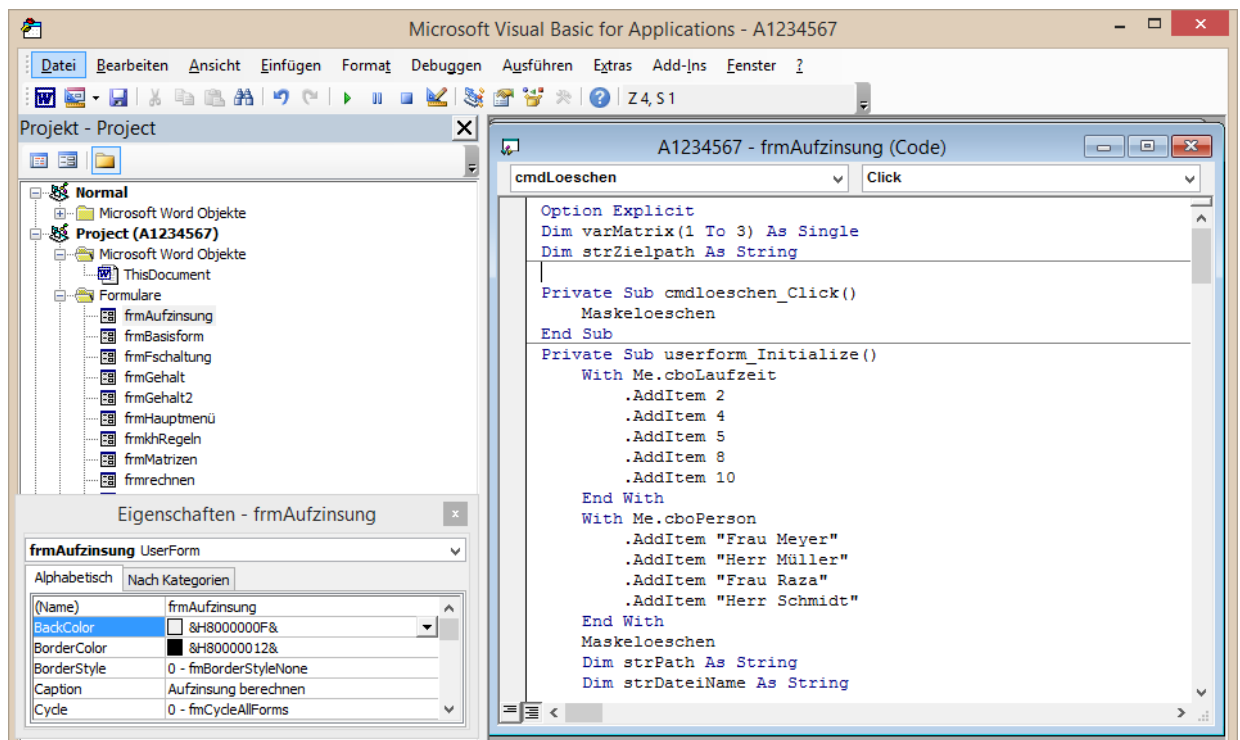
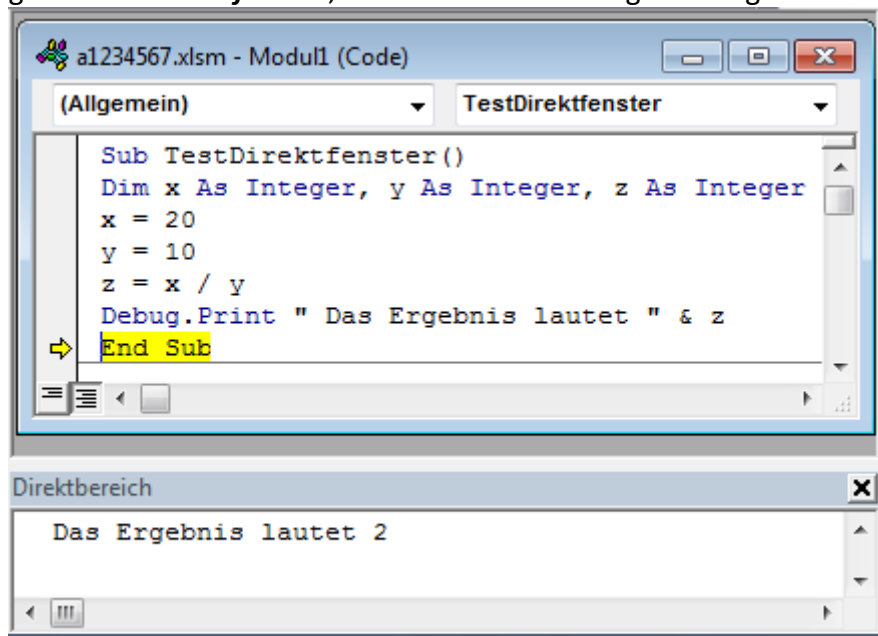


Abb. 2.1.4 Codeansicht von dem Userformblatt z.B. „frmAufzinsung“

In dieser Ansicht ist links ein Modul markiert und rechts wird der dort abgelegte Code angezeigt und später dann natürlich auch eingegeben. Zwei weitere Fenster sind noch interessant, allerdings hat man sie in den wenigsten Fällen immer eingeschaltet. Zum einen gibt es das **Direktfenster**, in dem man mit `Debug.Print` Ergebnisse von Variablen ausdrucken...

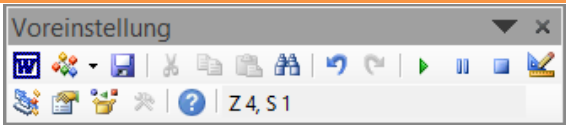





Das **Lokalfenster** dient zur direkten Überprüfung von Variablen. Es liefert zu allen globalen und lokalen Variablen des aktuellen Moduls den Wert und den Datentyp, wenn Sie mit **F8** durch den Code hoppeln.



Beim Halten der direkten Überprüfung klicken Sie auf das Symbol ZURÜCKSETZEN  in der Symbolleiste VOREINSTELLUNG.

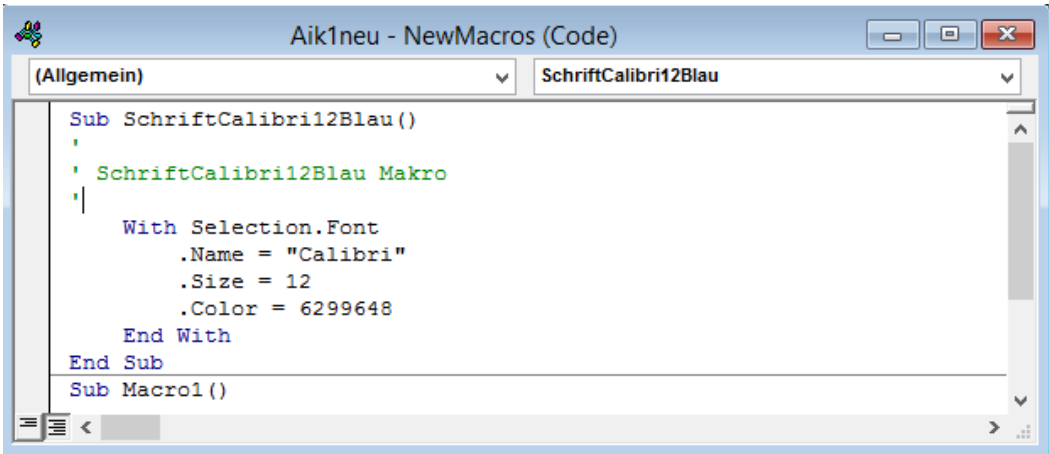
Symbolleisten des Visual Basic-Editors

Symbolleiste	Beschreibung
	Die häufigsten verwendeten Befehle befinden sich in dieser Symbolleiste. Die Befehle wie Makros starten, beenden, Zurücksetzen, Einblenden des Objektkatalogs die Rückgängigfunktionen, Einfügen von Objektelementen und die Suchfunktion.
	Enthält häufig verwendete Befehle zum Bearbeiten von Quellcode
	Enthält Befehle für das Testen von Visual Basic-Programmcode, z.B. Makro Starten, unterbrechen und beenden, Einzelschrittmodus, Anzeige von Lokal- und Direktfenster.
	Enthält Befehle, mit den Sie die Steuerelemente in den selbst erstellten Dialogfenstern positionieren können.

2.2 Makros bearbeiten und verwalten

Aufbau des Makro-Codes

Der Makro-Code besteht aus Visual Basic-Anweisungen, die beim Ausführen des Makros nacheinander abgearbeitet werden. Ein aufgezeichnetes Makros hat den folgenden Aufbau:



```

Sub SchriftCalibri12Blau()
    '
    ' SchriftCalibri12Blau Makro
    '
    With Selection.Font
        .Name = "Calibri"
        .Size = 12
        .Color = 6299648
    End With
End Sub
Sub Macro1()

```

1. Jedes Makro wird mit dem Schlüsselwort `Sub` (für engl. Subroutine = Unterprogramm) eingeleitet. Anschließend folgt der Name des Makros und ein Rundenklammernpaar. Man nennt hier die Deklaration eines Makros.

2. Hochkomma-Zeichen bedeutet für Visual Basic for Application Kommentare. Kommentare haben keinen Einfluss auf die Ausführung des Makrocodes und dienen nur der Beschreibung des Programmcodes.
3. Mit der `With ... End With` – Anweisung wird eine vereinfachte Schreibweise ermöglicht. Hier beziehen sich die in der `With`-Anweisung eingeschlossenen Anweisungen alle auf das Objekt `Selection.Font`. In der `With`-Anweisung kann deshalb z.B. `.Color` anstelle von `Selection.Font.Color` verwendet werden.
4. Das Ende des Makros ist durch die Schlüsselwörter `End Sub` gekennzeichnet.

Module bzw. Makros kopieren

Makros werden in Module gespeichert. Sie können nur Module aus Dokument kopieren, die geöffnet sind. Um einmal erstellte Makros auch in anderen Dokumenten benutzen zu können, wird das entsprechende Modul, welches die Makros enthält dorthin kopiert. Mit dem Modul kopieren Sie alle darin enthaltenen Makros bzw. VBA-Programmmodule.

Module exportieren

Sie haben die Möglichkeit, ein Modul als selbstständige Datei zu speichern. Diese Datei lässt sich später in andere Dokumente importieren. Abhängig von der Art des Moduls werden verschiedene Dateinamenserweiterungen verwendet: Standardmodule erhalten die Endung `*.bas`, Formularemodule `*.frm` und Klassenmodule `*.cls`.

Module importieren

Ein exportiertes Modul in Form einer Datei können Sie in die im Projekt-Explorer ausgewählte Arbeitsmappe einfügen bzw. importieren.

2.3 Aufruf eines Makros über Symbolleisten

Für den Aufruf eines Makros über eine Symbolleiste ist eine Schaltfläche in einer der vorhandenen oder einer neuen Symbolleiste nötig. Die Erstellung und Zuordnung ist in folgenden Schritten möglich:

- Symbolleiste für den schnellzugriff anpassen wählen.
- Weitere Befehle wählen.
- Das Dialogfenster (Abb. 2.3.1) WORD OPTIONEN MENÜBAND ANPASSEN wählen.
- Beim BEFEHLE AUSWÄHLEN MAKROS auswählen.
- AUF die Schaltfläche NEUE REGISTERKARTE klicken.
- Registerkarte nach einem gewünschten Namen z.B. EDV-SYMBOLLEISTE umbenennen.
- Neue Gruppe in dieser Registerkarte nach einem gewünschten Namen z.B. BTM 2015 umbenennen.
- Ihren Makronamen z.B. SchriftCalibri12Blau auswählen und auf die Schaltfläche HINZUFÜGEN>> klicken.
- Mit der rechten Maus auf das Symbol in der Registerkarte SchriftCalibri12Blau klicken, und Menüpunkt UMBENENNEN auswählen. Dann erscheint ein Dialogfenster (Abb. 2.3.2).
- Ein Symbol nach Ihrer Wahl dem Makro zuordnen.

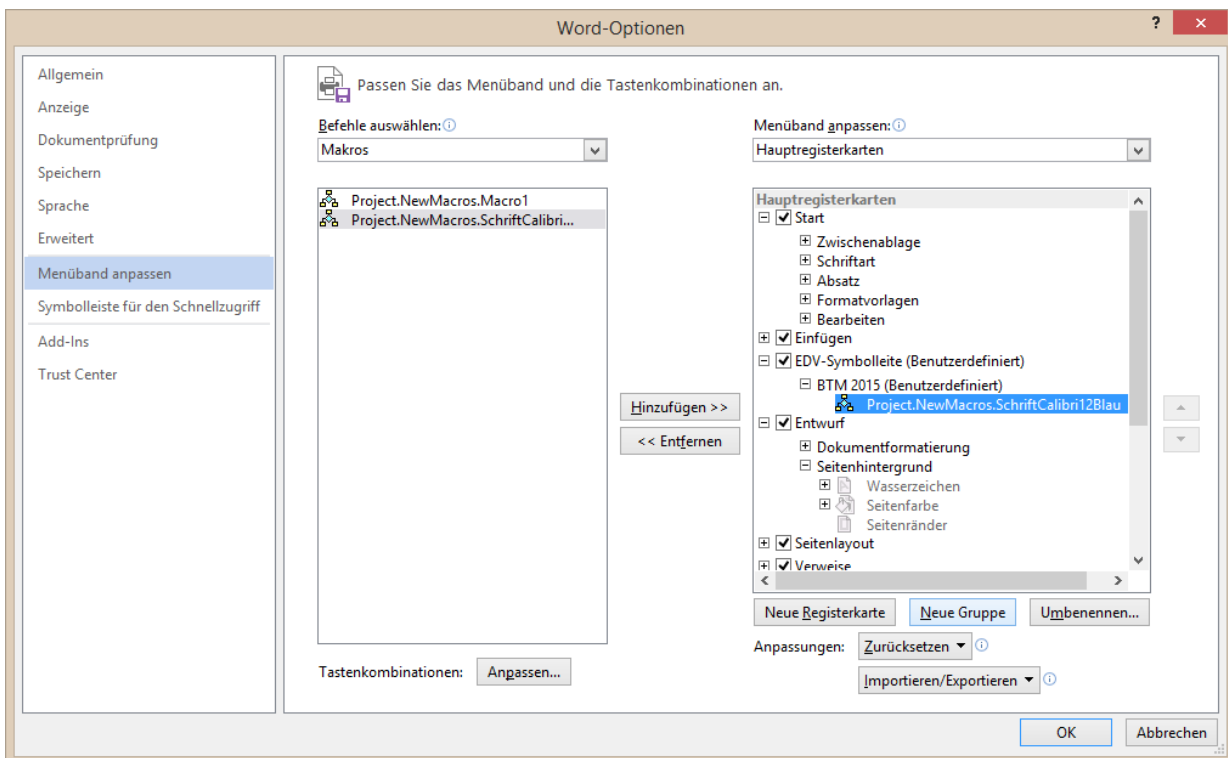


Abb. 2.3.1 Word-Optionen zur Entrichtung der Registerkarte

Wenn Sie mit allen Schritten fertig sind, bestätigen Sie Ihre Einstellung mit der Schaltfläche OK. Dann generiert Word eine Menüleiste mit den Gruppennamen und darunter Symbol bzw. Symbole für Ihre Makros (Abb. 2.3.3).

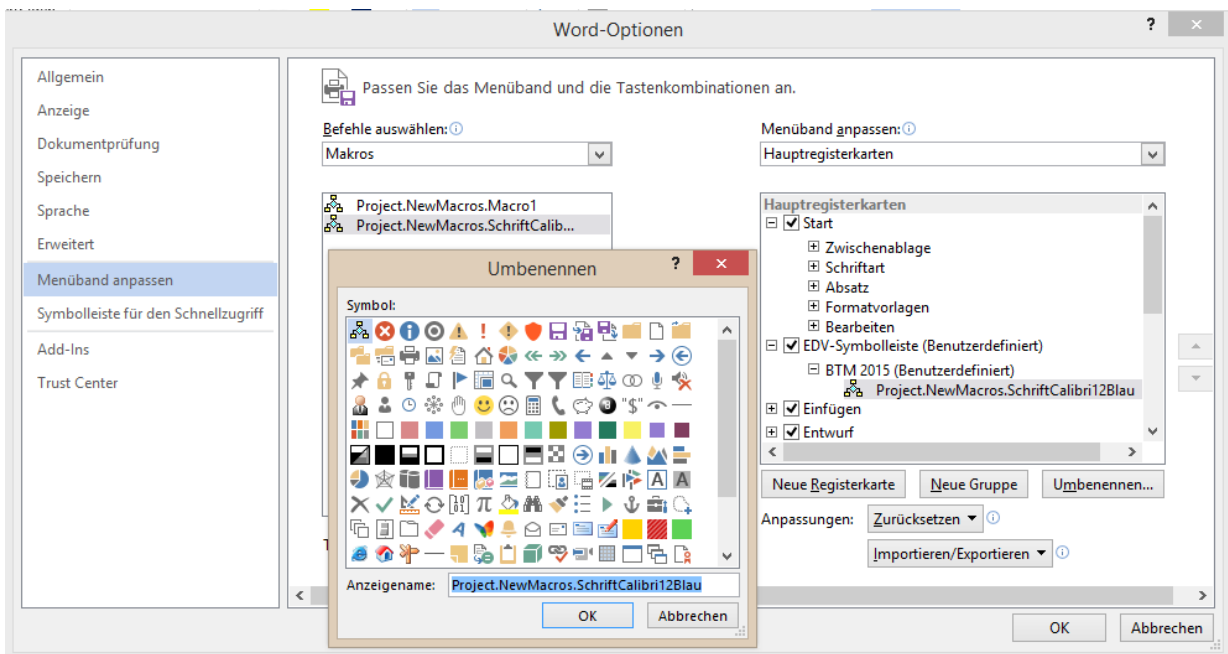


Abb. 2.3.2 Zuordnung eines Symbols für ein Makro in der Registerkarte



Abb. 2.3.3 Generierte Register, Gruppe und Makros



Registerkarte gehört zu der Microsoft Office Benutzeroberfläche. Daher wird diese nur auf einem Rechner sichtbar, auf dem Sie eingerichtet haben.

3 VBA Grundlagen

VBA – *Visual Basic for Applications* ist eine objektorientierte Makroprogrammiersprache mit einer sehr umfangreichen Auswahl von Funktionen und Anweisungen zur Erstellung eigenständiger Programme, die MS Office – Anwendungen automatisieren und oft aus der Sicht des Anwenders komplexe Abläufe in Ihrer Bedienung vereinfachen.

VBA – Module unterscheiden sich je nach Office – Produkt in Ihrer Struktur, hier sollen die Spezifika des Word – VBA in ihren Grundrissen vorgestellt werden. VBA – Module sind vom Sprachumfang her in zwei inhaltliche Bereiche teilbar:

- Den die Programmstruktur definierenden *Sprachkern*, der mit dem von VB (Visual Basic) gleichzusetzen ist, mit Schlüsselwörtern, Befehlen, Funktionen und Kontrollstrukturen, die für alle Office – Anwendungen gelten.
- Die anwendungsspezifischen *Objektmodelle*, mit eigenen, den Anwendungen angepassten OBJEKTEN, EIGENSCHAFTEN, EREIGNISSEN und METHODEN.

Da insbesondere diese vier Begriffe immer wieder als Grundbegriffe der objektorientierten Programmierung im Zusammenhang mit VBA auftreten und somit auch im Inhalt dieser Unterlage öfter vorkommen, sollen sie wie folgt vorgestellt werden.

OBJEKTE stellen einen zentralen Bestandteil fast aller VBA – Anwendungen dar. Sie sind bestimmte Teile einer Anwendung – Word selbst, das Dokument, die Textmarke, Tabellen. Objekte besitzen EIGENSCHAFTEN (benannte Attribute), beispielsweise Größe, Farbe, Position, Name usw.

Das Verhalten eines Objekts wird über die METHODEN (= Prozeduren), die auf ein Objekt angewandt werden, bestimmt.

3.1 Word-Objektmodell

Word besteht aus Objekten, deren Eigenschaften und Methoden frei zugänglich sind. Zu den wichtigsten Objekten zählen:

- Application-Objekt
- Document-Objekt
- Selection-Objekt
- Range-Objekt
- Bookmark-Objekt

Kurzdarstellung des Word-Objektmodells

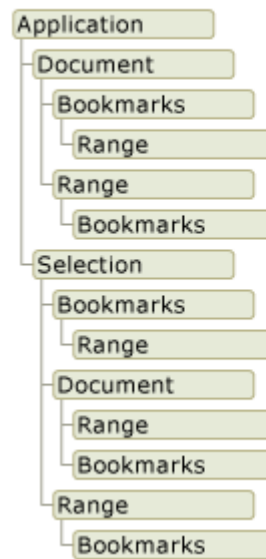


Abb. 3.1 Objektmodell in Word

Die Objekte stehen in hierarchischer Abhängigkeit zueinander. Objekte höherer Stufe beinhalten Objekte untergeordnet Stufe(n). Für o.g. Objekte gilt die Hierarchieordnung. Das gesamte Objektmodell (Abb. 3.1) ist natürlich wesentlich komplizierter und umfangreicher.

Für die Programmierung ist es wichtig zu wissen, dass die Objekthierarchie nicht nur ein willkürliches Ordnungsschema ist. Die Kenntnis der festgelegten Position eines Objekts in der Hierarchie ist nötig, um das Objekt zu referenzieren – darauf zuzugreifen.

Beispiel:

```
ActiveDocument.Bookmarks("Wk227").Range.Font.Name = "Calibri"
```

3.2 Eigenschaften

Während ein Objekt weitgehend abgeschlossen ist, d.h. sein innerer Aufbau lässt sich in der Regel nicht beeinflussen, lassen sich seine EIGENSCHAFTEN per Programmanweisung in vielen Fällen verändern. Die Veränderung der Eigenschaften nimmt oft einen wesentlichen Teil eines Programms ein.

Eigenschaften sind benannte Attribute eines Objekts. Sie bestimmen seine Charakteristika wie Größe, Farbe oder Bildschirmposition, aber auch den Zustand, wie beispielsweise *aktiviert* oder *deaktiviert*.

Es gibt Eigenschaften, die lesbar und veränderbar sind, z.B. *Value* (Wert) oder *Name* (Name), andere lassen sich nur abfragen, aber nicht verändern – es sind sog. *Nur – Lese Eigenschaften*. Zu den Eigenschaften, die besonders oft verändert werden, gehören:

CAPTION	Beschriftungen von Objekten
NAME	Die Bezeichnung eines Objekts, unter der es angesprochen (referenziert) wird.
SELECTION	Bezeichnet das markierte Application – Objekt.
VALUE	Wert / Inhalt eines Objekts (Zellinhalt, Textfeld – Eintrag)

Alle Eigenschaften haben immer einen aktuellen Wert. Die Anzahl möglicher Werte ist unterschiedlich groß. So besitzen beispielsweise die Eigenschaften *Color* oder *Value* sehr viele Werte, während andere, beispielsweise *Selected*, nur die Werte *False* oder *True* annehmen.

Beispiel:

```
Me.optEZins.Value = False
```

Die Anweisung weist den Wert Falsch dem Optionsfeld „optEZins“ zu.

3.3 Methoden

Zu den Objekten gehören neben Eigenschaften auch METHODEN. Über Methoden lässt sich das Verhalten von Objekten steuern / verändern. Eine Methode ist eine Aktion, die eine Operation auf einem Objekt ausführen kann. Zu den am häufigsten benutzten Methoden gehören:

OPEN	öffnet ein Dokument
CLOSE	schließt ein Dokument
CLEAR	löscht den Inhalt eines Kombinationsfeldes.
ADD	Elemente in ein Objekt hinzufügen
SELECT	wählt ein Objekt aus

Für den Einsteiger ist es oft problematisch zwischen Eigenschaften und Methoden zu unterscheiden. So sind beispielsweise die beiden Anweisungen:

Beispiel:

```
ActiveDocument.Bookmarks("Wk227").Range.Font.Name = "Calibri"  
ActiveDocument.Bookmarks.Add "Wk227", ActiveDocument.Bookmarks("Wk227").Range
```

zumindest optisch sehr ähnlich. Es stellt sich die Frage - sind *NAME* und *ADD* Eigenschaften oder Methoden und wenn nicht welches Element von beiden ist eine Methode und welches ein Eigenschaft.

Die Antwort ist hier relativ einfach: Bei Zuweisungen von Eigenschaften wird das Gleichheitszeichen benutzt, bei Methoden benutzt man (optionale) Parameter ohne Gleichheitszeichen.

Bei der Programmierung in der VBA – Umgebung wird die Entscheidung Eigenschaft /Methode zusätzlich durch spezifische Symbole in entsprechenden Auswahlfenstern erleichtert (s. weiter im Text).

3.4 Ereignisse

Für die meisten Objekte sind explizite Ereignisse definiert, denen fest zugeordnete *Ereignisprozeduren* zugehören. Ereignisse können Aufrufe von Menüfunktionen, Anklicken von Schaltern, Symbolen und Tasten aber auch Öffnen von Dokumenten, Berechnungen, Veränderungen von Inhalten usw. Allgemein gesehen - Mausclicks, Tastatureingaben und systeminterne Aktionen lösen Ereignisse aus, auf die über entsprechende Prozeduren reagiert werden muss.

3.5 Die VisualBasic-Syntax

In diesem Teil wollen wir uns schrittweise und systematisch mit der Syntax von VisualBasic vertraut machen. Die Syntax informiert Sie, welche Sprachelemente verfügbar sind und wie sie syntaktisch richtig eingesetzt werden. Wir beginnen mit den lexikalischen Elementen.

VisualBasic besitzt Grammatik- und Zeichensetzungsregeln, die festlegen, welche Zeichen und Wörter einen Sinn geben und in welcher Reihenfolge sie geschrieben werden können.

Beispiel 1) Geben Sie am Anfang eines Dokuments einen Bereich an, und fügen Sie den Text **New Text** ein. Das folgende Codebeispiel kann in einer Anpassung auf Dokumentebene verwendet werden.

```
Dim rng As Word.Range
rng = Me.Range (Start:=0, End:=0)
rng.Text = " New Text "
```

Punkt-Syntax

In VisualBasic verweist ein Punkt (.) auf die zu einem Objekt gehörigen Eigenschaften und Methoden. Ein Punkt-Syntax-Ausdruck beginnt mit dem Namen des Objektes, gefolgt von einem Punkt, und endet mit der Eigenschaft, Methode oder Variablen, die Sie angeben möchten.

Beispiel 2) Das folgende Codebeispiel kann in einem Add-In auf Anwendungsebene verwendet werden. In diesem Code wird das aktive Dokument verwendet.

```
Dim rng As Word.Range
Rng = Me.Application.ActiveDocument.Range (Start:=0, End:=0)
rng.Text = " New Text "
```

Wählen Sie das Range-Objekt aus, das von einem Zeichen auf die Länge des eingefügten Texts erweitert wurde.

```
rng.Select()
```

Runde Klammern

1. Bei der Definition einer Prozedur (sub, Function und Property) werden die Argumente in runde Klammern eingeschlossen

Beispiel 3) Definition von Sub-Prozedur Maskeloeschen

```
Sub Maskeloeschen()
Me.cboLaufzeit.Text = ""
Me.cboLaufzeit.BackColor = RGB(255, 255, 255)
Me.txtAnlage.Text = ""
Me.txtAnlage.BackColor = RGB(255, 255, 255)
Me.txtKapital.Text = ""
Me.txtKapital.BackColor = RGB(255, 255, 255)
Me.txtProzent.Text = ""
Me.txtProzent.BackColor = RGB(255, 255, 255)
Me.txtZArt.Text = ""
Me.txtKn.Text = ""
Me.optEZins.Value = False
Me.optZZins.Value = False
End Sub
```

2. Beim Aufruf einer Funktion werden die Argumente der Funktion in runden Klammern übergeben, beispielsweise:

Beispiel 4) In dem folgenden Beispiel wird die MsgBox-Funktion mittels benannter Argumente aufgerufen und der Rückgabewert der Variablen Ergebnis zugewiesen:

```
Mldg = "Wollen Sie die alte Daten löschen?"
```

```
Stil = vbYesNo + vbCritical + vbDefaultButton2
Title = "Daten löschen"
Ergebnis = MsgBox(Mldg, Stil, Title)
```

3. Sie können mit runden Klammern auch die Reihenfolge von Operationen in VisualBasic beeinflussen oder VisualBasic-Anweisungen leichter lesbar machen.

Beispiel 5)

Rem Das Ergebnis ist 14 ohne runde Klammer.

```
Sum = 2 + 4 * 3
```

Wenn die Addition jedoch geklammert wird, führt VisualBasic zuerst die Addition aus:

Beispiel 6)

Rem Das Ergebnis ist hier 18.

```
Sum = (2 + 4) * 3
```



Nähere Informationen finden Sie im Abschnitt „Vorrang des Operators“.

4. Runde Klammern dienen auch zum Auswerten eines Ausdrucks. In der folgenden Anweisung bewirken die runden Klammern beispielsweise, dass das Parameter an der Funktion übergeben wird. Die Funktion gibt dann eine Meldung aus. Wenn der Inhalt vom Textfeld `txtProzent` eine numerische Zahl ist, zeigt die Zahl in einem Fenster an, ansonsten wird der Inhalt als nicht Korrekt angezeigt und das Textfeld wird rot dargestellt.

Beispiel 7)

Rem Funktionsdeklaration

```
Function Textfeldpruefen(Textfeld As Object) As Double
    If IsNumeric(Textfeld.Value) Then
        Textfeldpruefen = Textfeld.Value
    Else
        MsgBox "Der Inhalt " & Textfeld & " ist nicht korrekt!"
        Textfeld.BackColor = RGB(255, 255, 0)
        Exit Function
    End If
End Function
```

Rem Aufrufen von Textfeldpruefen.

```
MsgBox Textfeldpruefen (Me.txtProzent.Text)
```

Rem Der Inhalt vom Textfeld `txtProzent` wird angezeigt.

Groß- und Kleinbuchstaben

VisualBasic unterscheidet zwischen Groß- und Kleinschreibung. Die folgenden Anweisungen sind nicht gleichwertig:

Beispiel 8)

```
Dim meineVar As String
meineVar = "Christine"           'richtig
meinevar = "Christine"         'falsch
```

Sie sollten sich konsistente Großschreibungskonventionen angewöhnen, beispielsweise die in diesem Manuskript verwendeten, um Funktions- und Variablennamen beim Lesen von VisualBasic-Code leichter zu erkennen. Wenn Sie bei Schlüsselwörtern nicht korrekt zwischen

Groß- und Kleinschreibung unterscheiden, ist Ihr Skript fehlerhaft, das wird dann rot angezeigt. Die richtig geschriebenen Schlüsselwörter werden blau angezeigt. Weitere Informationen erhalten Sie unter Schlüsselwörter und Hervorheben und Überprüfen der Syntax.

Kommentare

Im VisualBasic Modus können Sie den Codes mit Hilfe einer Kommentar-Anweisung Anmerkungen hinzufügen, um die gewünschten Ergebnisse der Aktion festzuhalten. Kommentare erleichtern auch die Weitergabe von Informationen, wenn in einem Entwicklerteam gearbeitet oder Beispiele weitergegeben werden.

Wenn Sie die Kommentare eingeben, müssen Sie davor das Zeichen ' einfügen. Selbst ein einfaches Skript ist leichter verständlich, wenn Sie bei der Erstellung Anmerkungen einfügen:

Beispiel 9)

```

` Beim Klicken auf das Kombinationsfeld
Private Sub cboLaufzeit_Click()
    Rem Die Werte der beiden Optionsfelder auf Falsch setzen.
    Me.optEZins.Value = False
    Me.optZZins.Value = False
End Sub
    
```

Kommentare werden im Codefenster in der Farbe Grün angezeigt. Sie können beliebig lang sein, ohne dass die Größe der exportierten Datei dadurch beeinflusst wird, sie unterliegen nicht den Regeln bezüglich VisualBasic-Syntax oder Schlüsselwörtern, und sie können neu in jeder Zeile eingefügt werden.

Schlüsselwörter

In VisualBasic sind einige Wörter für eine bestimmte Verwendung reserviert. Diese können daher nicht als Namen für Variablen, Funktionen oder Labels verwendet werden. In der folgenden Tabelle sind einige VisualBasic-Schlüsselwörter aufgeführt:

• Ablaufsteuerung-Schlüsselwörter (Zusammenfassung)	
Aktion	Schlüsselwörter
Ausführen einer Schleife	Do...Loop, For...Next, For Each...Next, While...Wend, With
Beenden oder Anhalten des Programms	DoEvents, End, Exit, Stop
Entscheidungen treffen	Choose, If...Then...Else, Select Case, Switch
Verwenden von Prozeduren	Call, Function, Property Get, Property Let, Property Set, Sub
Verzweigen	GoSub...Return, GoTo, On Error, On...GoSub, On...GoTo
• Compiler-Anweisung-Schlüsselwörter (Zusammenfassung)	
Definieren einer Compiler-Konstante	#Const
Kompilieren ausgewählter Code-Blöcke	#If...Then...#Else
• Datenfeld-Schlüsselwörter (Zusammenfassung)	

Ändern der Standarduntergrenze	Option Base
Deklarieren und Initialisieren eines Datenfeldes	Dim, Private, Public, ReDim, Static
Erneutes Initialisieren eines Datenfeldes	Erase, ReDim
Erstellen eines Datenfeldes	Array
Suchen der Grenzen eines Datenfeldes	LBound, UBound
Überprüfen eines Datenfeldes	IsArray



Weitere Informationen zu bestimmten Schlüsselwörtern finden Sie in Microsoft-VisualBasic- Hilfe <https://msdn.microsoft.com/de-de/library/office/ee861527.aspx>

Konstanten

Ein benanntes Element, das während der gesamten Programmausführung einen konstanten Wert behält. Eine Konstante kann ein Zeichenfolge- oder numerisches Literal, eine andere Konstante oder eine beliebige Kombination sein, die arithmetische oder logische Operatoren enthält. Jede Host-Anwendung kann ihren eigenen Satz von Konstanten bestimmen. Zusätzliche Konstanten können vom Benutzer mit der `Const`-Anweisung definiert werden. Sie können Konstanten überall in Ihrem Code anstelle der tatsächlichen Werte einsetzen.

Die folgenden Konstanten sind in Visual Basic für Applikationen definiert, um die Programmierung zu vereinfachen. Sie können daher überall im Code anstelle der tatsächlichen Werte verwenden:

- Compiler-Konstanten
- Datumskonstanten
- Dir-, GetAttr- und SetAttr-Konstanten
- Farbkonstanten
- IMEStatus-Konstanten
- Instr-, StrComp-Konstanten
- Kalender-Konstanten
- MsgBox-Konstanten
- QueryClose-Konstanten
- Shell-Konstanten
- StrConv-Konstanten
- Systemfarben-Konstanten
- Tasten-Code-Konstanten
- VarType-Konstanten
- Verschiedene Konstanten

MsgBox-Konstanten

Konstante	Wert	Beschreibung
vbOK	1	Schaltfläche OK gedrückt
vbCancel	2	Schaltfläche Abbrechen gedrückt
vbAbort	3	Schaltfläche Abbruch gedrückt

vbRetry	4	Schaltfläche Wiederholen gedrückt
vbIgnore	5	Schaltfläche Ignorieren gedrückt
vbYes	6	Schaltfläche Ja gedrückt
vbNo	7	Schaltfläche Nein gedrückt

Die Konstanten `vbOk`, `vbCancel`, `vbAbort`, `vbRetry`, `vbIgnore`, `vbYes` und `vbNo` sind beispielsweise Eigenschaften einer Schaltfläche und verweisen auf Rückgabe-Wert in `MsgBox`-Funktion. Mit der folgenden Anweisung können Sie prüfen, ob der Anwender die Yes- oder No-Schaltfläche gedrückt hat:

Beispiel 10) In diesem Beispiel werden alle Absätze im aktiven Dokument durchlaufen. Wenn die Einstellung "Abstand vor" für einen Absatz 6 Punkt lautet, wird der Abstand in diesem Beispiel auf 12 Punkt geändert.

```
Private Sub cmdaendern_Click()
    Meldung = "Wollen Sie wirklich den Absatz auf 12 Punkt ändern?"
    Titel = "Absatz ändern"
    ' Die Konstanten vbYesNo, vbCritical und vbDefaultButton2?
    Stil = vbYesNo + vbCritical + vbDefaultButton2
    Ergebnis = MsgBox(Meldung, Stil, Titel)
    ' Die Konstante vbYes?
    If Ergebnis = vbYes Then

        Dim parCount As Paragraph

        For Each parCount In ActiveDocument.Paragraphs

            If parCount.SpaceBefore = 12 Then parCount.SpaceBefore = 6

        Next parCount
    Else
        Exit Sub
    End If
End Sub
```



Weitere Informationen zu bestimmten Schlüsselwörtern finden Sie in Microsoft-VisualBasic- Hilfe <https://msdn.microsoft.com/de-de/library/office/ee861527.aspx>

Datentypen

Datentypen sind das, was Einsteigern bei anderen Programmiersprachen Kopfzerbrechen bereitet. Was es mit einem Datentyp auf sich hat? Für jede Variable oder Konstante wird ein bestimmter Platz im Speicher reserviert. Je nachdem, ob es sich um eine Zeichenkette oder eine Zahl handelt, wird ein mehr oder weniger großer Bereich des Speichers benötigt. Damit nun die Arbeit mit diesem Speicherbereich keine Fehler verursacht, muss der entsprechende Datentyp vorher deklariert werden.



Variable ist eine benannte Speicherposition, die Daten enthalten kann, die während der Programmausführung verändert werden können. Jede Variable hat einen Namen, der sie eindeutig in ihrem Gültigkeitsbereich identifiziert. Zusätzlich kann ein Datentyp angegeben werden. Variablennamen müssen mit einem Zeichen des Alphabets beginnen, innerhalb des Gültigkeitsbereichs müssen sie eindeutig sein, und sie dürfen nicht länger als 255 Zeichen sein. Variablennamen dürfen keinen Punkt und kein Typkennzeichen enthalten.

In VisualBasic gibt prinzipiell diese Datentypen: Byte, Boolean, Integer, Long, Currency, Decimal, Single, Double, Date, String, Object, Variant. Dabei ist Variant (Voreinstellung) und die benutzerdefinierte Typen sind auch spezielle Objekttypen.

- **Boolean-Datentyp:** Variablen vom Datentyp `Boolean` werden als 16-Bit-Zahlen (2 Bytes) gespeichert, die nur die Werte `True` oder `False` annehmen können. Variablen vom Datentyp `Boolean` werden als Wahr oder Falsch ausgegeben, Beim Umwandeln von Werten des Datentyps `Boolean` in andere Datentypen wird `False` zu 0 und `True` zu -1.
- **Byte-Datentyp:** Variablen vom Datentyp `Byte` werden als einzelne 8-Bit-Zahlen (1 Byte) ohne Vorzeichen gespeichert und haben einen Wert im Bereich von 0 bis 255. Der Datentyp `Byte` ist zur Speicherung binärer Daten nützlich.
- **Currency-Datentyp:** Variablen vom Datentyp `Currency` werden als 64-Bit-Zahlen (8 Bytes) in einem ganzzahligen Format gespeichert und durch 10.000 dividiert, was eine Festkommazahl mit 15 Vorkomma- und 4 Nachkommastellen ergibt. Diese Darstellung ergibt einen Wertebereich von -922.337.203.685.477,5808 bis 922.337.203.685.477,5807. Das Typkennzeichen für `Currency` ist das Zeichen (`@`). Der Datentyp `Currency` eignet sich besonders für Berechnungen mit Geldbeträgen und für Festkommaberechnungen, die eine hohe Genauigkeit erfordern.
- **Date-Datentyp:** Variablen vom Datentyp `Date` werden als 64-Bit-Gleitkommazahlen (8 Bytes) nach IEEE gespeichert und können ein Datum im Bereich vom 01. Januar 100 bis zum 31. Dezember 9999 und eine Uhrzeit im Bereich von 0:00:00 bis 23:59:59 speichern. Jeder gültige Wert eines Datums- oder Zeitliterals kann einer Variablen vom Datentyp `Date` zugewiesen werden. Ein Datumsliteral muss durch das Zeichen (`#`) eingeschlossen sein, zum Beispiel: `#January 1, 1993#` oder `#1 Jan 93#`.

Beispiel 11)

```
Sub test()
    Application.DisplayAlerts = False
    Heute = Now
    Verfalldatum = #1/15/1998# 'Die Datei ist veraltet
    'Verfalldatum = #15.02.98# 'Die Datei ist brauchbar
    If Verfalldatum > Heute Then
        MsgBox "Alles klar!"
    Else
        Prompt = "Diese Mappe wird geschlossen!"
        DialogArt = vbOK
        Title = "Arbeitsmappe zu alt"
        Antwort = MsgBox(Prompt, DialogArt, Title)
        ActiveDocument.Close SaveChanges:=wdDoNotSaveChanges
    End If
End Sub
```

- **Decimal-Datentyp:** Variablen des Datentyps `Decimal` werden als 96-Bit-Ganzzahlen (12 Bytes) ohne Vorzeichen mit einer variablen Potenz zur Basis 10 gespeichert. Die Potenz zur Basis 10 wird als Skalierungsfaktor verwendet und bestimmt die Anzahl der Nachkommastellen, die in einem Bereich von 0 bis 28 liegen kann. Der Datentyp `Decimal` kann nur mit einem Wert vom Typ `Variant` benutzt werden, d.h., Sie können keine Variable als `Decimal` deklarieren. Mit der `CDec`-Funktion können Sie jedoch einen Wert vom Typ `Variant` erstellen, dessen Untertyp `Decimal` ist.

- **Double-Datentyp:** Variablen vom Datentyp `Double` (Gleitkommazahl mit doppelter Genauigkeit) werden als 64-Bit-Gleitkommazahlen (8 Bytes) nach IEEE im Bereich von $-1,79769313486232E308$ bis $-4,94065645841247E-324$ für negative Werte und von $4,94065645841247E-324$ bis $1,79769313486232E308$ für positive Werte gespeichert. Das Typkennzeichen für `Double` ist das Zeichen (`#`).
- **Decimal-Datentyp:** Variablen vom Datentyp `Integer` werden als 16-Bit-Zahlen (2 Bytes) in einem Bereich von -32.768 bis 32.767 gespeichert. Das Typkennzeichen für `Integer` ist das Zeichen (`%`). Mit Variablen vom Datentyp `Integer` können Sie auch Aufzählungswerte darstellen. Ein Aufzählungswert besteht aus einer endlichen Menge eindeutiger ganzer Zahlen, von denen jede im verwendeten Kontext eine spezielle Bedeutung hat. Aufzählungswerte erlauben eine einfache Auswahl aus einer bestimmten Anzahl von Möglichkeiten, z.B. `0` = schwarz, `1` = weiß usw. Sie sollten die `Const`-Anweisung verwenden, um Konstanten für jeden Aufzählungswert zu definieren.
- **Long-Datentyp:** Variablen vom Datentyp `Long` (lange Ganzzahl) werden als 32-Bit-Zahlen (4 Bytes) mit Vorzeichen im Bereich von $-2.147.483.648$ bis $2.147.483.647$ gespeichert. Das Typkennzeichen für `Long` ist das Zeichen (`&`).
- **Object-Datentyp:** Variablen vom Datentyp `Object` werden als 32-Bit-Adressen (4 Bytes) gespeichert, die auf Objekte in einer Anwendung verweisen. Einer Variablen, die als `Object` deklariert wurde, kann anschließend mit der `Set`-Anweisung ein Verweis auf jedes von der Anwendung erzeugte Objekt zugewiesen werden.
- **Single-Datentyp:** Variablen vom Datentyp `Single` (Gleitkommazahl mit einfacher Genauigkeit) werden als 32-Bit-Gleitkommazahlen (4 Bytes) nach IEEE im Bereich von $-3,402823E38$ bis $-1,401298E-45$ für negative Werte und von $1,401298E-45$ bis $3,402823E38$ für positive Werte gespeichert. Das Typkennzeichen für `Single` ist das Ausrufezeichen (`!`).
- **String-Datentyp:** Es gibt zwei Arten von Zeichenfolgen: Zeichenfolgen variabler Länge und Zeichenfolgen fester Länge.
 - ✓ Zeichenfolgen variabler Länge können bis zu 2 Milliarden (oder 2^{31}) Zeichen enthalten.
 - ✓ Zeichenfolgen fester Länge können 1 bis etwa 64 KB (2^{16}) Zeichen enthalten.

Die Codes für Zeichen vom Datentyp `String` liegen im Bereich von 0 bis 255 (einschließlich). Die ersten 128 Zeichen (0 bis 127) entsprechen den Buchstaben und Symbolen auf einer US-amerikanischen Standardtastatur. Diese ersten 128 Zeichen stimmen mit den im ASCII-Zeichensatz definierten Zeichen überein. Die zweiten 128 Zeichen (128 bis 255) sind Sonderzeichen, z.B. Buchstaben aus internationalen Alphabeten, Akzentzeichen, Währungssymbole und Symbole für mathematische Brüche. Das Typkennzeichen für `String` ist das Dollarzeichen (`$`).

- **Type-Datentyp:** Jeder Datentyp, den Sie mit der `Type`-Anweisung definieren. Benutzerdefinierte Datentypen können ein oder mehrere Elemente eines beliebigen Datentyps, eines Datenfeldes oder eines bereits bestehenden benutzerdefinierten Datentyps enthalten.


```
Type Typ1
  Name1 As String      ' String-Variable für Namen.
  Geburtstag As Date   ' Date-Variable für Geburtstag.
  Geschlecht As Integer ' Integer-Variable für
End Type              '(0 für weiblich, 1 für männlich).PersonalNr = "5423";
```

- **Variant-Datentyp:** Der Datentyp `Variant` ist der Datentyp für alle Variablen, die nicht explizit (durch eine Anweisung wie `Dim`, `Private`, `Public` oder `Static`) als anderer Datentyp deklariert werden. Für den Datentyp `Variant` gibt es kein Typkennzeichen.
 - ✓ `Variant` ist ein besonderer Datentyp, der beliebige Daten mit Ausnahme von `String`-Daten fester Länge und benutzerdefinierten Typen enthalten kann. Ein `Variant` kann auch die speziellen Werte `Empty`, `Error`, `Nothing` und `Null` enthalten. Mit der Funktion `VarType` oder `TypeName` können Sie festlegen, wie die Daten in einer Variablen vom Datentyp `Variant` interpretiert werden.
 - ✓ Als numerische Daten sind beliebige Ganzzahlen oder reelle Zahlen im Bereich von `-1,797693134862315E308` bis `-4,94066E-324` für negative Werte und von `4,94066E-324` bis `1,797693134862315E308` für positive Werte zulässig. Im Allgemeinen behalten numerische Daten vom Datentyp `Variant` den ursprünglich festgelegten Datentyp als Untertyp innerhalb des `Variant`s bei. Wenn Sie zum Beispiel einem `Variant` einen Wert vom Datentyp `Integer` zuweisen, interpretieren alle nachfolgenden Operationen den `Variant` als Datentyp `Integer`. Wenn Sie jedoch mit einem `Variant` mit dem Typ `Byte`, `Integer`, `Long` oder `Single` eine arithmetische Operation ausführen und das Ergebnis den zulässigen Bereich für den ursprünglichen Datentyp überschreitet, wird das Ergebnis innerhalb des `Variant` automatisch zu dem nächstgrößeren Datentyp erweitert. `Byte` wird zu `Integer`, `Integer` wird zu `Long`, und `Long` bzw. `Single` werden zu `Double` umgewandelt. Werden die zulässigen Bereiche für den Datentyp `Currency`, `Decimal` oder `Double` in einem `Variant` überschritten, so tritt ein Fehler auf.
 - ✓ Der Datentyp `Variant` kann anstelle jedes anderen Datentyps verwendet werden, um im Umgang mit Daten flexibler zu sein. Enthält eine `Variant`-Variable Ziffern, so können diese (je nach Kontext) entweder als Zeichenfolgendarstellung der Ziffern oder als deren tatsächlicher Wert interpretiert werden.

Beispiel 13)

```
Dim TestVar As Variant      ' Eine Variable als eine Zahl.
TestVar = 44137
Dim Matrix(1 To 3) As Variant ' Ein Array mit 3 Elemente.
```



Weitere Informationen zu bestimmten Schlüsselwörtern finden Sie in Microsoft-VisualBasic- Hilfe. <https://msdn.microsoft.com/en-us/library/office/gg264383.aspx>

Variablen

Variablen spielen eine grundsätzliche und wichtige Rolle in jeder Programmiersprache. Sie sind es im Grunde genommen, was unsere dynamische Software von heute vereinfacht. Eine Variable ist eigentlich ein Behälter für Informationen mit einem Wert. Der Behälter an sich bleibt dabei immer gleich, nur der Inhalt kann sich ändern. Jede Variable hat einen Namen, der sie eindeutig in ihrem Gültigkeitsbereich identifiziert. Zusätzlich kann ein Datentyp

angegeben werden. Variablennamen müssen mit einem Zeichen des Alphabets beginnen, innerhalb des Gültigkeitsbereichs müssen sie eindeutig sein, und sie dürfen nicht länger als 255 Zeichen sein. Variablennamen dürfen keinen Punkt und kein Typkennzeichen enthalten.

Zur Deklaration von Variablen verwenden Sie normalerweise eine `Dim`-Anweisung. Eine Deklarationsanweisung kann innerhalb einer Prozedur zur Erstellung einer Variablen auf Prozedurebene oder zu Beginn eines Moduls im Deklarationsabschnitt zur Erstellung einer Variablen auf Modulebene plziert werden.

Beispiel 14) In dem folgenden Beispiel wird die Variable `strName` erstellt und der Datentyp `String` angegeben.

```
Dim strName As String
```

Ist diese Anweisung Teil einer Prozedur, so kann die Variable `strName` nur in dieser Prozedur verwendet werden. Befindet sich diese Anweisung im Deklarationsabschnitt des Moduls, so ist die Variable `strName` für alle Prozeduren innerhalb des Moduls, aber nicht für Prozeduren in anderen Modulen des Projekts verfügbar. Damit diese Variable für alle Prozeduren des Projekts verfügbar ist, stellen Sie ihr die `Public`-Anweisung voran.

Beispiel 15)

```
Public strName As String
```



Weitere Informationen zur Benennung Ihrer Variablen finden Sie in der Visual Basic-Hilfe unter dem Thema "Namenskonventionen in Visual Basic".

Variablen können als einer der folgenden Datentypen deklariert werden: `Boolean`, `Byte`, `Integer`, `Long`, `Currency`, `Single`, `Double`, `Date`, `String` (für Zeichenfolgen variabler Länge), `String * Länge` (für Zeichenfolgen fester Länge), `Object` oder `Variant`. Wenn Sie keinen Datentyp angeben, wird der Datentyp `Variant` standardmäßig zugewiesen. Sie können auch einen benutzerdefinierten Typ über die `Type`-Anweisung erstellen.

Sie können mehrere Variable in einer Anweisung deklarieren. Wenn Sie einen Datentyp angeben möchten, müssen Sie den Datentyp für jede Variable angeben. In der folgenden Anweisung werden die Variablen `intX`, `intY` und `intZ` als `Integer` deklariert.

```
Dim intX As Integer, intY As Integer, intZ As Integer
```

In der folgenden Anweisung werden `intX` und `intY` als `Variant`, und nur `intZ` als `Integer` deklariert.

```
Dim intX, intY, intZ As Integer
```

Sie müssen die Datentypen der Variablen in der Deklarationsanweisung nicht angeben. Wenn Sie den Datentyp nicht angeben, so wird die Variable auf den Typ `Variant` festgelegt.

Sie können die `Public`-Anweisung zur Deklaration öffentlicher Variablen auf Modulebene verwenden.

```
Public strName As String
```

Öffentliche Variablen können in allen Prozeduren des Projekts verwendet werden. Wenn eine öffentliche Variable in einem Standardmodul oder einem Klassenmodul deklariert wird, kann diese auch in allen Projekten verwendet werden, die auf das Projekt verweisen, in dem die öffentliche Variable deklariert wurde.

Sie können die `Private`-Anweisung zur Deklaration privater Variablen auf Modulebene verwenden.

```
Private MeinName As String
```

Private Variablen können nur von Prozeduren des gleichen Moduls verwendet werden.



Wird die `Dim`-Anweisung auf Modulebene verwendet, gleicht sie der `Private`-Anweisung. Möglicherweise möchten Sie die `Private`-Anweisung verwenden, um die Lesbarkeit des Codes zu erhöhen und die Interpretation zu vereinfachen.

Wenn Sie die `Static`-Anweisung anstelle einer `Dim`-Anweisung verwenden, so behält die deklarierte Variable ihren Wert zwischen Aufrufen.

Bezeichnen einer Variablen

Der Name einer Variablen muss folgende Bedingungen erfüllen:

- Er muss ein Bezeichner sein.
- Er darf kein Schlüsselwort und kein Literal vom Typ `Boolean` (`true` oder `false`) sein.
- Er muss innerhalb seines Gültigkeitsbereichs eindeutig sein.

Gültigkeitsbereich einer Variablen

Der Gültigkeitsbereich ("Scope") einer Variablen bezieht sich auf den Bereich, in dem die Variable bekannt ist und in dem auf sie verwiesen werden kann.

Es gibt drei Gültigkeitsbereichebenen:

- Prozedurebene,
- private Modulebene
- öffentliche Modulebene.

Sie bestimmen den Gültigkeitsbereich einer Variablen bei deren Deklaration. Durch die explizite Deklaration aller Variablen können Fehler aufgrund von Namenskonflikten zwischen Variablen mit unterschiedlichen Gültigkeitsbereichen vermieden werden.

Definieren von Gültigkeitsbereichen auf Prozedurebene

Eine innerhalb einer Prozedur definierte Variable oder Konstante ist außerhalb dieser Prozedur nicht verfügbar. Nur die Prozedur, die die Variablendeklaration enthält, kann diese Variable verwenden. Im folgenden Beispiel zeigt die erste Prozedur ein Meldungsfeld mit einer Zeichenfolge an. Die zweite Prozedur zeigt ein leeres Meldungsfeld an, da die Variable `strMldg` für die erste Prozedur lokal ist.

Beispiel 16)

```
Sub LokaleVariable()
    Dim strMldg As String
    strMldg = "Diese Variable kann nur innerhalb " & "dieser Prozedur verwendet werden."
    MsgBox strMldg
End Sub

Sub AußerhalbGültigkeitsbereich()
    MsgBox strMldg
```

End Sub

Definieren von Gültigkeitsbereichen auf privater Modulebene

Sie können Variablen und Konstanten auf Modulebene im Deklarationsabschnitt eines Moduls definieren. Variablen auf Modulebene können entweder öffentlich oder privat sein. Öffentliche Variablen sind für alle Prozeduren in allen Modulen eines Projekts verfügbar, private Variablen sind nur für die Prozeduren in diesem Modul verfügbar. Standardmäßig werden Variablen mit der `Dim`-Anweisung im Deklarationsabschnitt im privaten Gültigkeitsbereich deklariert. Durch Voranstellen des Schlüsselworts `Private` jedoch wird der Gültigkeitsbereich der Variablen in Ihrem Code erkennbar.

Im folgenden Beispiel ist die Zeichenfolgenvariable `strMldg` für alle in dem Modul definierten Prozeduren verfügbar. Bei Aufruf der zweiten Prozedur werden die Inhalte der Zeichenfolgenvariablen `strMldg` in einem Dialogfeld angezeigt.

Beispiel 17)

```
' Fügen Sie zum Deklarationsabschnitt des Moduls folgenden Code hinzu.
Private strMldg As String
Sub InitialisierePrivateVariable()
    strMldg = "Diese Variable kann nur innerhalb " & "dieses Moduls verwendet werden."
End Sub

Sub VerwendePrivateVariable()
    MsgBox strMldg
End Sub
```

Definieren von Gültigkeitsbereichen auf öffentlicher Modulebene

Wenn Sie eine Variable auf Modulebene als öffentlich deklarieren, ist sie für alle Prozeduren in diesem Projekt verfügbar. Im folgenden Beispiel kann die Zeichenfolgenvariable `strMldg` von jeder Prozedur in jedem Modul dieses Projekts verwendet werden.

```
' Fügen Sie diesen Code in den Deklarationsabschnitt des Moduls ein.
Public strMldg As String
```

Alle Prozeduren (ausgenommen Ereignisprozeduren) sind standardmäßig öffentlich. Wenn Visual Basic eine Ereignisprozedur erstellt, wird das Schlüsselwort `Private` automatisch vor der Prozedurdeklaration eingefügt. Für alle anderen Prozeduren müssen Sie die Prozedur explizit mit dem Schlüsselwort `Private` deklarieren, wenn die Prozedur nicht öffentlich sein soll.

Typumwandlungen

In jeder Sprache kommt vor, dass der Datentyp einer Variablen innerhalb ihres Gültigkeitsbereiches geändert werden muss. In VisualBasic legt jede Funktion für einen bestimmten Datentyp zwingend einen Ausdruck fest. Die folgenden Funktionen werden in VisualBasic für Typumwandlungen verwendet:

Funktion	Rückgabetyt	Bereich des Arguments Ausdruck
CBool	Boolean	Eine gültige Zeichenfolge oder ein gültiger numerischer Ausdruck.
CByte	Byte	0 bis 255.
CCur	Currency	-922.337.203.685.477,5808 bis 922.337.203.685.477,5807.

CDate	Date	Ein beliebiger gültiger Datumsausdruck.
CDbl	Double	-1,79769313486232E308 bis -4,94065645841247E-324 für negative Werte; 4,94065645841247E-324 bis 1,79769313486232E308 für positive Werte.
CDec	Decimal	+/-79.228.162.514.264.337.593.543.950.335 für skalierte Ganzzahlen, d.h. Zahlen ohne Dezimalstellen. Für Zahlen mit 28 Dezimalstellen gilt der Bereich +/-7,9228162514264337593543950335. Die kleinste mögliche Zahl ungleich Null ist 0,00000000000000000000000000000001.
CInt	Integer	-32.768 bis 32.767; Nachkommastellen werden gerundet.
CLng	Long	-2.147.483.648 bis 2.147.483.647; Nachkommastellen werden gerundet.
CSng	Single	-3,402823E38 bis -1,401298E-45 für negative Werte; 1,401298E-45 bis 3,402823E38 für positive Werte.
Cvar	Variant	Numerische Werte im Bereich des Typs Double. Nichtnumerische Werte im Bereich des Typs String.
CStr	String	Rückgabe für CStr hängt vom Argument Ausdruck ab.

Das erforderliche Argument Ausdruck ist ein Zeichenfolgenausdruck oder ein numerischer Ausdruck.

In diesem Beispiel wird die CStr-Funktion verwendet, um einen numerischen Wert in den Typ String umzuwandeln.

Beispiel 18)

```
Dim TestDouble, TestString
TestDouble = 437.324           ' TestDouble hat den Typ Double.
TestString = CStr(TestDouble) ' TestString enthält "437,324".
```

In diesem Beispiel wird die CDate-Funktion verwendet, um eine Zeichenfolge in einen Wert vom Typ Date umzuwandeln. In der Regel sollten Sie Datums- und Zeitangaben nicht als Zeichenfolgen direkt in den Code eingeben, wie in diesem Beispiel. Verwenden Sie stattdessen Datums- und Zeitlitterale, wie #2/12/1969#, #4:45:23 PM#.

Beispiel 19)

```
Dim TestDatum, TestKurzesDatum, TestZeit, TestKurzeZeit
TestDatum = "12. Februar 1969"           ' Datum definieren.
TestKurzesDatum = CDate(TestDatum)       ' In Datentyp Date umwandeln.
TestZeit = "16:35:47"                   ' Zeit definieren.
TestKurzeZeit = CDate(TestZeit)          ' In Datentyp Date umwandeln.
```

Arrays

Arrays stellen eine besondere Art von Variablen dar. Im Gegensatz zu den üblichen Variablen sind sie in der Lage, mehrere Werte aufzunehmen, wie es im obigen Abschnitt erklärt wurde. Dabei werden wieder zwei Arten von Arrays unterschieden. Zum einen können Arrays werte in Form einer einfachen Liste oder auch Aufzählung aufnehmen. Hier ist die Rede von *eindimensionalen Arrays*. Zum anderen können Arrays auch Werte in Form einer Tabelle aufnehmen, hier ist die Rede von *zweidimensionalen Arrays*.

Eindimensionale Arrays

Eindimensionale Arrays nehmen Werte in der Art einer Liste auf. Um auf die Werte innerhalb dieser Liste zuzugreifen, werden diese mit einem fortlaufenden Index versehen. In VisualBasic gibt die Array-Funktion, mit der ein Wert vom Typ Variant zurückgibt. Dieser Wert enthält ein Datenfeld.

Syntax

```
Array(ArgListe)
```

Das erforderliche Argument `ArgListe` enthält eine durch Kommas getrennte Liste von Werten, die den in `Variant` enthaltenen Elementen des Datenfeldes zugewiesen werden. Werden keine Argumente angegeben, so erstellt die Funktion ein Null-Datenfeld.



Der Zugriff auf ein beliebiges Element eines Datenfeldes erfolgt durch Angabe des Variablennamens mit einem nachfolgenden Klammernpaar, das die Indexnummer des gewünschten Elements enthält. Im folgenden Beispiel erstellt die erste Anweisung die Variable `DayOfWeek` als `Variant`. Die zweite Anweisung weist der Variablen `DayOfWeek` ein Datenfeld zu. Die letzte Anweisung weist den Wert, der im zweiten Datenfeldelement enthalten ist, einer anderen Variablen zu.

Beispiel 20)

```
Dim DayOfWeek As Variant
DayOfWeek = Array("Mo", "Di", "Mi", "Do", "Fr", "Sa", "So");
B = DayOfWeek(2) ' Also zweites Element ist Di
```

Die untere Grenze für ein mit der `Array`-Funktion erstelltes Datenfeld ist immer Null. Im Gegensatz zu anderen Datenfeldtypen ist es nicht von der unteren Grenze betroffen, die mit der Option `Base`-Anweisung festgelegt wurde.



Ein `Variant`, der nicht als Datenfeld deklariert ist, kann trotzdem ein Datenfeld enthalten. Eine Variable vom Typ `Variant` kann ein Datenfeld eines beliebigen Typs enthalten, außer Zeichenfolgen konstanter Länge und benutzerdefinierte Typen. Obwohl ein `Variant`, der ein Datenfeld enthält, sich konzeptionell von einem Datenfeld unterscheidet, dessen Elemente vom Typ `Variant` sind, erfolgt der Zugriff auf die Datenfeldelemente in gleicher Weise.

Datenfeldern

Sie können ein Datenfeld deklarieren, das mit einer Gruppe von Werten des gleichen Datentyps arbeitet. Ein Datenfeld ist eine einzelne Variable mit mehreren Feldern, in denen Werte gespeichert werden, während eine typische Variable nur über ein Speicherfeld verfügt, in dem nur ein Wert gespeichert werden kann. Sie können auf das Datenfeld als Ganzes verweisen, wenn Sie auf alle darin befindlichen Werte verweisen möchten, oder auf die einzelnen Elemente.

Damit Sie z.B. die täglichen Kosten für jeden Tag des Jahres speichern können, deklarieren Sie eine Datenfeldvariable mit 365 Elementen anstelle von 365 Variablen. Jedes Element in einem Datenfeld enthält einen Wert. Die folgende Anweisung deklariert die Datenfeldvariable `curKosten` mit 365 Elementen. Standardmäßig wird ein Datenfeld beginnend mit Null (0) indiziert, so daß die obere Grenze des Datenfeldes statt 365 nur 364 entspricht.

```
Dim curKosten (364) As Currency
```

Damit Sie den Wert eines einzelnen Elements festlegen können, geben Sie den Index des Elements an. Das folgende Beispiel weist jedem Element im Datenfeld den Anfangswert 20 zu.

```
Sub FülleDatenfeld ()
    Dim curKosten (364) As Currency
```

```
Dim intI As Integer
For intI = 0 to 364
    curKosten (intI) = 20
Next
End Sub
```

Sie können die Option `Base`-Anweisung am Anfang eines Moduls dazu verwenden, den Standardindex des ersten Elements von 0 auf 1 zu ändern. Im folgenden Beispiel ändert die Option `Base`-Anweisung den Index für das erste Element und die `Dim`-Anweisung deklariert die Datenfeldvariable `curKosten` mit 365 Elementen.

```
Option Base 1
Dim curKosten(365) As Currency
```

Sie können auch explizit die untere Grenze eines Datenfelds über den `To`-Abschnitt festlegen.

Beispiel 21)

```
Dim curKosten(1 To 365) As Currency
Dim strWochentag(7 To 13) As String
```

Es gibt zwei Möglichkeiten, Datenfelder für Werte des Typs `Variant` zu erstellen. Sie können ein Datenfeld als Datentyp `Variant` deklarieren. Beispiel:

```
Dim varDaten(3) As Variant
varDaten(0) = "Christine Müller"
varDaten(1) = "44137 Dortmund"
varDaten(2) = 38
varDaten(3) = Format("06-09-1952", "General Date")
```

Sie können aber auch das von der `Array`-Funktion zurückgegebene Datenfeld einer `Variant`-Variablen zuweisen.

Beispiel 22)

```
Dim varDaten As Variant
varDaten = Array("Christine Müller", "44137 Dortmund", 38, _
Format("06-09-1952", "Allgemeines Datum"))
```

Sie kennzeichnen die Elemente in einem Datenfeld von `Variant`-Werten über den Index, unabhängig von der bei der Erstellung des Datenfeldes verwendeten Methode. Die folgende Anweisung kann z.B. zu jedem der vorhergehenden Beispiele hinzugefügt werden.

```
MsgBox "Daten für " & varDaten(0) & " wurden aufgezeichnet."
```

mehrdimensionale Datenfelder (mehrdimensionale Arrays)

In Visual Basic können Sie Datenfelder mit bis zu 60 Dimensionen deklarieren. Die folgende Anweisung deklariert z.B. ein zweidimensionales, 5-zu-10-Datenfeld.

```
Dim sngMulti(1 To 5, 1 To 10) As Single
```

Wenn Sie sich das Datenfeld als Matrix vorstellen, so stellt das erste Argument die Zeilen und das zweite Argument die Spalten dar.

Verwenden Sie verschachtelte `For...Next`-Anweisungen für die Verarbeitung mehrdimensionaler Datenfelder. Die folgende Prozedur füllt ein zweidimensionales Datenfeld mit `Single`-Werten.

```
Sub FülleDatenfeldMulti ()
    Dim intI As Integer, intJ As Integer
```

```

Dim sngMulti(1 To 5, 1 To 10) As Single
' Ausfüllen des Datenfeldes mit Werten.
For intI = 1 To 5
    For intJ = 1 To 10
        sngMulti(intI, intJ) = intI * intJ
        Debug.Print sngMulti(intI, intJ)
    Next intJ
Next intI
End Sub

```



Weitere Informationen zu bestimmten Schlüsselwörtern finden Sie in Microsoft-VisualBasic- Hilfe. <https://msdn.microsoft.com/en-us/library/office/gg264844.aspx>

Operatoren

Operatoren werden öfter benötigt als man im ersten Moment glauben mag. Operatoren sind Zeichen, die festlegen, wie Werte in einem Ausdruck kombiniert, verglichen oder geändert werden. Die Elemente, auf die der Operator angewendet wird, werden als Operanden bezeichnet. Operatoren werden in verschiedene Gruppe eingeteilt, wobei jede dieser Gruppen eigene Aufgaben zu erfüllen hat.

Arithmetische Operatoren

Arithmetische Operatoren addieren, subtrahieren, multiplizieren, dividieren und führen weitere arithmetische Operationen durch. In der folgenden Tabelle sind die numerischen Operatoren in VBA aufgeführt:

Operator	Bezeichnung	Beispiel	Beschreibung
+	Addition	Res = a + b	Summe von a und b
-	Subtraktion	Res = a - b	Differenz von a und b
*	Multiplikation	Res = a * d	Produkt von a und d
/	Division	Res = a / b	Dividieren a durch b , Res ist Fließkommazahl
\	Division	Res = d / a;	Dividieren a durch b , Res ist Ganzzahl
Mod	Modulo	Res = a Mod b	Rest der Division von a und b
^	Potenzieren	Res = a ^ b	Potenzieren von a mit Exponent b

Vergleichsoperatoren

Diese Operatoren gehören zu den mit Sicherheit am meisten benötigten. Sie vergleichen Werte von Ausdrücken und geben einen Wert vom Typ `Boolean` (true oder false) zurück. Diese Operatoren werden meist in Schleifen und bedingten Anweisungen verwendet. In diesem Beispiel werden verschiedene Einsatzmöglichkeiten für Vergleichsoperatoren zum Vergleichen von Ausdrücken aufgezeigt.

Beispiel 23)

```

Dim Ergebnis, Var1, Var2
Ergebnis = (45 < 35)           ' Liefert False.
Ergebnis = (45 = 45)         ' Liefert True.
Ergebnis = (4 <> 3)          ' Liefert True.
Ergebnis = ("5" > "4")      ' Liefert True.
Var1 = "5": Var2 = 4          ' Variablen initialisieren.
Ergebnis = (Var1 > Var2)     ' Liefert True.
Var1 = 5: Var2 = Empty
Ergebnis = (Var1 > Var2)     ' Liefert True.
Var1 = 0: Var2 = Empty
Ergebnis = (Var1 = Var2)     ' Liefert True.

```

In der folgenden Tabelle sind die Vergleichsoperatoren in VBA aufgeführt:

Operator	Bezeichnung	Beispiel	Beschreibung
<	Kleiner als	If (a < b)	a ist kleiner als b.
>	Größer als	If (b > a)	b ist größer als b.
<=	Kleiner oder gleich	If (a <= b)	a ist kleiner oder gleich b.
>=	Größer oder gleich	If (a >= b)	a ist kleiner oder gleich b.
=	gleich	If (a = b)	a ist gleich b.
<>	ungleich	If (a <> b)	a ist kleiner oder größer als b.

Logische Operatoren

Logische Operatoren vergleichen Werte vom Typ Boolean (true und false) und geben einen dritten Wert vom Typ Boolean zurück. Wenn beide Operanden z.B True ergeben, gibt der logische Operator UND (&&) das Ergebnis true zurück. Wenn einer oder beide Operanden true ergeben, gibt der logische Operator ODER (||) das Ergebnis true zurück. Logische Operatoren werden häufig in Verbindung mit Vergleichsoperatoren verwendet, um die Bedingung für eine if-Aktion festzulegen. In diesem Beispiel wird der Operator And verwendet, um zwei Ausdrücke mit einer logischen Konjunktion zu verknüpfen.

Beispiel 24)

```

Dim A, B, C, D, Test1
A = 10: B = 8: C = 6: D = Null ' Variablen initialisieren.
Test1 = A > B And B > C       ' Liefert True.
Test1 = B > A And B > C       ' Liefert False.
Test1 = A > B And B > D       ' Liefert Null.
Test1 = A And B               ' Liefert 8 (Bit-weiser Vergleich).

```

In der folgenden Tabelle sind die logischen Operatoren in VisualBasic aufgeführt:

Operator	Bezeichnung	Beispiel	Beschreibung
And	Logisches UND	a1 And a2	Ergibt true, wenn beide a1 und a2 true sind.

Or	Logisches ODER	a1 Or a2	Ergibt true, wenn mindestens einer a1 oder a2 true sind.
Eqv	Logisches Äquivalenz	a1 Eqv a2	Ergibt true, wenn entweder a1 und a2 true oder false sind.
Not	logische Negation	Not a1	Ergibt true, wenn a1 false ist.
Xor	logischen Exklusion	a1 Xor a2	Ergibt true, wenn entweder a1 oder a2 true ist, aber nicht beides

Verkettungsoperatoren

Verkettungsoperatoren dienen zu Verkettung zweier Zeichenketten (in diesem Fall mit dem Operator &) und Addition von zweier Zahlen (in diesem Fall mit dem Operator +).



Wenn Sie den Operator + verwenden, können Sie nicht immer bestimmen, ob eine Addition oder eine Zeichenverkettung erfolgt. Für die Verkettung zweier Zeichenfolgen sollten Sie den Operator & verwenden, um Mehrdeutigkeiten auszuschließen und Code zu erstellen, der sich selbst dokumentiert.

Operator	Bezeichnung	Beispiel	Beschreibung
&	Bitweises UND	a & b	a und b werden verkettet.
+	Addition zweier zahlen	Res = a + b	Es werden alle Bits beeinflusst, die in a oder b gesetzt sind.

In diesem Beispiel wird der Operator + verwendet, um Zahlen zu addieren. Der Operator + kann auch Zeichenfolgen verketteten. Um Mehrdeutigkeiten zu vermeiden, sollten Sie jedoch dazu den Operator & verwenden. Wenn die Komponenten eines Ausdrucks, der mit dem Operator + erstellt wurde, sowohl Zeichenfolgen als auch Zahlen umfassen, wird das arithmetische Ergebnis zugewiesen. Wenn die Komponenten ausschließlich Zeichenfolgen sind, werden diese Zeichenfolgen verkettet.

Beispiel 25)

```
Dim hufPlusmwzhu, huf, mwzhu
huf = 38, mwzhu=2
HufPlusmwzhu = huf + mwzhu      ' Liefert 40
```

In diesem Beispiel wird der Operator & verwendet, um Zeichenfolgen zu verketteten.

Beispiel 26)

```
Dim Textkmg
Textkmg = "Körpermaße" & " Größe"      ' Liefert "Körpermaße Größe"
```

Operatorvorrang

Wenn ein Ausdruck mehrere Operationen enthält, werden die einzelnen Teilausdrücke in einer bestimmten Rangfolge ausgewertet und aufgelöst, die als Operatorvorrang bezeichnet wird. Wenn Ausdrücke Operatoren aus mehreren Kategorien enthalten, werden zunächst die **arithmetischen Operatoren**, dann die **Vergleichsoperatoren** und zuletzt die **logischen Operatoren** ausgewertet. Die Vergleichsoperatoren haben alle dieselbe Priorität und werden

daher von links nach rechts in der Reihenfolge ihres Auftretens ausgewertet. Für die arithmetischen und logischen Operatoren gilt die folgende Rangfolge:

Arithmetisch	Vergleich	Logisch
Potenzierung (^)	Gleich (=)	Not
Negation (-)	Ungleich (<>)	And
Multiplikation und Division (*, /)	Kleiner als (<)	Or
Ganzzahldivision (\)	Größer als (>)	Xor
Restwert (Mod)	Kleiner oder gleich (<=)	Eqv
Addition und Subtraktion (+, -)	Größer oder gleich (>=)	Imp
Zeichenverkettung (&)	Like, Is	

Multiplikation und Division innerhalb eines Ausdrucks sind gleichrangig und werden von links nach rechts in der Reihenfolge ihres Auftretens ausgewertet. Dasselbe gilt für Additionen und Subtraktionen, die zusammen in einem Ausdruck auftreten. Mit Klammern kann diese Rangfolge außer Kraft gesetzt werden, damit bestimmte Teilausdrücke vor anderen Teilausdrücken ausgewertet werden. In Klammern gesetzte Operationen haben grundsätzlich Vorrang. Innerhalb der Klammern gilt jedoch wieder die normale Rangfolge der Operatoren. Der Zeichenverkettungsoperator (&) ist zwar kein arithmetischer Operator, liegt aber in der Rangfolge zwischen den arithmetischen Operatoren und den Vergleichsoperatoren. Der Operator `Like` ist eigentlich ein Operator zum Mustervergleich, wird aber in der Rangfolge den anderen Vergleichsoperatoren gleichgestellt. Der Operator `Is` dient zum Vergleichen von Verweisen auf Objekte. Er vergleicht nicht die Objekte oder deren Werte, sondern überprüft lediglich, ob sich zwei Objektverweise auf dasselbe Objekt beziehen.



Weitere Informationen Operatoren finden Sie in der Visual Basic-Hilfe unter dem Thema "Operatoren in VBA". <https://msdn.microsoft.com/en-us/library/office/jj692799.aspx>

Kontrollstrukturen

Um Variablen und Objekten manipulieren und steuern zu können oder den Zustand des Programms zu überprüfen, kommen oft Anweisungen und Schleifen zum Einsatz. Bei solchen Anweisungen und Schleifen handelt es sich um die Auswertung und Überprüfung Zuständen sowie Werten. Zu Bewältigung solcher Aufgaben können so genannte Kontrollstrukturen eingesetzt werden. Das Prinzip solcher Kontrollstrukturen ist immer gleich: Es findet ein Vergleich oder eine Überprüfung statt, deren Ergebnis entweder den Wert als `Boolean true` (wahr) oder `false` (falsch) hat. Je nachdem, wie die Überprüfung oder Vergleich ausfällt, kann das Programm in eine bestimmte Richtung verzweigt und weitergeführt werden. Allerdings unterscheiden sich die Kontrollstrukturen wesentlich in ihrem Aufbau und ihrer Funktionsweise.

Fallunterscheidungen

if...Then...Else-Anweisung

Die `if...Then...Else`-Anweisung führt eine Gruppe von Anweisungen aus, wenn bestimmte Bedingungen erfüllt sind, die vom Wert eines Ausdrucks abhängen.

Syntax:

```
If Bedingung Then
    [Anweisungen]
[ElseIf Bedingung-n Then
    [elseifAnweisungen] ...
[Else
    [elseAnweisungen]]
End If
```

Beispiel 27) In diesem Beispiel wird die Blockform und die einzeilige Form der `If...Then...Else`-Anweisung demonstriert. Eine Zahl wird überprüft. Das Ergebnis wird überprüft.

```
'Variable initialisieren.
Dim Zahl As Integer
Zahl = 50 ' Initialize variable.
If Zahl > 10 Then
    Debug.Print "Zahl ist größer als 10."
ElseIf Zahl < 100 Then
    'Bedingung ist erfüllt, also wird die nächste Anweisung ausgeführt.
    Debug.Print "Zahl ist kleiner als 100."
Else
    Debug.Print "Diese Zahl ist nicht gültig."
End If
```

Select Case-Anweisung

Bei `IF`-Konstruktionen lassen sich immer nur zwei alternative Programmteile ausführen, da eine Bedingung als logischer Ausdruck nur die Werte `True` bzw. `False` ergeben kann. Bei einer mehrseitigen Auswahl testen Sie den Wert einer Variablen oder eines komplexen Ausdrucks. Diese Variable bzw. dieser Ausdruck wird Testausdruck bezeichnet. In Abhängigkeit vom Wert des Testausdrucks können je nach Fall (`case`) verschiedene Anweisungsblöcke ausgeführt werden. Die mehrseitige Auswahl wird daher auch Testausdruck(Selection) genannt. Also `Select case`-Anweisung führt eine von mehreren Gruppen von Anweisungen aus, abhängig vom Wert eines Ausdrucks.

Syntax:

```
Select Case Testausdruck
    [Case Ausdrucksliste-n
        [Anweisungen-n]] ...
    [Case Else
        [elseAnw]]

End Select
```

Beispiel 28) In diesem Beispiel wird die `Select Case`-Anweisung verwendet, um den Wert einer Variablen auszuwerten. Der zweite `Case`-Abschnitt enthält den Wert der ausgewerteten Variablen, und nur die zugehörige Anweisung wird ausgeführt.

```
Dim Zahl
Zahl = 8      ' Variable initialisieren.
Select Case Zahl      ' Zahl auswerten.
Case 1 To 5      ' Zahl von 1 bis 5.
    Debug.Print "Zahl von 1 bis 5"
    ' Das ist der einzige Case-Abschnitt, der True ergibt.
Case 6, 7, 8      ' Zahl von 6 bis 8.
    Debug.Print "Zahl von 6 bis 8"
Case Is 9 To 10      ' Zahl ist 9 oder 10.
    Debug.Print "Größer als 8"
Case Else      ' Andere Werte.
    Debug.Print "Außerhalb von 1 bis 10"
End Select
```



Weitere Informationen über die Fallunterscheidungen wie `Select Case`-Anweisung, `Switch`-Funktion, `#if...#Then...#Else`-Anweisung finden Sie in VBA <https://msdn.microsoft.com/en-us/library/office/jj692812.aspx>

Schleifen

Eine weitere Form der Kontrollstrukturen sind Schleifen. Während eine Fallunterscheidung nur ein einziges Mal durchlaufen wird, sind bei Schleifen mehrere Durchläufe, also Wiederholungen möglich. Jeder Schleifentypus weist einen Schleifenkopf auf, der durch eine Bedingung gekennzeichnet ist. Diese Bedingung legt die Anzahl der Aufrufe des Anweisungsblocks fest. Wenn die Bedingung einer Schleife nicht erfüllt ist, wird die Schleife ignoriert und die Programmausführung nach dem Funktionsblock der Schleife fortgesetzt. Falls ein vorzeitiges Abbrechen der Iteration durch den Funktionsblock einer Schleife erwünscht ist, so kann dies durch einen Aufruf der Funktion `Exit` erzielt werden.

For Each...Next-Anweisungen

Die `for Each...Next`-Schleife findet ihren Einsatz in Bereichen, in denen es auf die Ausführung einer Anzahl von Anweisungen in einer bestimmten Häufigkeit ankommt. Sie wiederholen einen Block von Anweisungen für jedes Objekt in einer Auflistung oder jedes Element in einem Datenfeld.

Syntax:

```
For Each Element In Gruppe
    [Anweisungen]
[Exit For]
[Anweisungen]
```

```
Next [Element]
```

Visual Basic stellt automatisch bei jedem Schleifendurchlauf eine Variable ein. Die folgende Prozedur z.B. schließt alle Formulare mit Ausnahme des Formulars, das die auszuführende Prozedur enthält.

Die erste Zeile von der Tabelle wird mit einem 3D-Rahmen versehen.

```
Dim MTab As Table
Set MTab = ActiveDocument.Tables(1)
For Each seite In MTab.Rows(1).Borders
    seite.LineStyle = wdLineStyleEmboss3D
Next
```

For...Next-Anweisungen

Sie können For...Next-Anweisungen verwenden, um einen Block von Anweisungen eine unbestimmte Anzahl von Wiederholungen ausführen zu lassen. For-Schleifen verwenden eine Zählervariable, deren Wert mit jedem Schleifendurchlauf erhöht oder verringert wird.

Syntax:

```
For Zähler = Anfang To Ende [Step Schritt]
    [Anweisungen]
    [Exit For]
    [Anweisungen]
Next [Zähler]
```

Beispiel 29)

```
Sub TabErstellen()
    Rem Eine Tabelle an der aktuellen Cursorposition mit 5 Zeilen und 2 Spalten können
    Rem Sie einfügen mit:
    ActiveDocument.Tables.Add Range:=Selection.Range,numRows:=5, numColumns:=2
    Rem Zum Füllen der Tabelle kann zum Beispiel folgende Schleife dienen:
    With ActiveDocument.Tables(1)
        For zeile = 1 To 5
            For spalte = 1 To 2
                .Cell(zeile, spalte).Range.InsertAfter "Zelle " &
                    zeile & "," & spalte
            Next
        Next
    End With
End Sub
```



Innerhalb einer Schleife kann eine beliebige Anzahl von Exit For-Anweisungen an beliebiger Stelle als alternative Möglichkeit zum Verlassen der Schleife verwendet werden.

Exit For wird oft in Zusammenhang mit der Auswertung einer Bedingung (zum Beispiel If...Then) eingesetzt und überträgt die Steuerung an die unmittelbar auf Next folgende Anweisung.

Do . . . Loop- Anweisung

Wiederholt einen Block mit Anweisungen, solange eine Bedingung den Wert True hat oder bis eine Bedingung den Wert False erhält.

Syntax:

```
Do [{While | Until} Bedingung]
  [Anweisungen]
  [Exit Do]
  [Anweisungen]
Loop
```



Innerhalb einer Do...Loop-Anweisung kann eine beliebige Anzahl von Exit Do-Anweisungen an beliebiger Stelle als Alternative zum Verlassen einer Do...Loop-Anweisung verwendet werden. Exit Do wird oft in Zusammenhang mit der Auswertung einer Bedingung (zum Beispiel If...Then) eingesetzt und hat zur Folge, dass die Ausführung mit der ersten Anweisung im Anschluss an Loop fortgesetzt wird.

Wir formen das Beispiel 26 in diesem Abschnitt in einer Do...Loop –Schleife um.

Beispiel 30)

```
Sub TabErstellen()
  Dim i As Integer, flag As Boolean
  i = 1
  flag = True
  Rem Eine Tabelle an der aktuellen Cursorposition mit 5 Zeilen und 2 Spalten können
  Rem Sie einfügen mit:
  ActiveDocument.Tables.Add Range:=Selection.Range,numRows:=5, numColumns:=2
  Rem Zum Füllen der Tabelle kann zum Beispiel folgende Schleife dienen:
  With ActiveDocument.Tables(1)
    Do While flag
      If zeile < 6 Then
        For spalte = 1 To 2
          .Cell(zeile, spalte).Range.InsertAfter "Zelle " &
            zeile & "," & spalte
        Next
        Zeile = Zeile + 1
      Else
        flag = False
      End If
    Loop
  End with
End Sub
```

With-Anweisung

In Visual Basic gehört die With-Anweisung zu den Schleifen, weil Sie damit Sie eine Reihe von Anweisungen für ein bestimmtes Objekt ausführen können, ohne den Namen des Objekts mehrmals angeben zu müssen. Wenn Sie zum Beispiel mehrere Eigenschaften eines bestimmten Objekts verändern möchten, sollten Sie die Zuweisungsanweisungen für die Eigenschaft in der With-Kontrollstruktur unterbringen. Sie brauchen dann den Namen des

Objekts nicht bei jeder einzelnen Zuweisung, sondern nur einmal zu Beginn der Kontrollstruktur anzugeben.

Syntax:

```
With Objekt  
    [Anweisungen]  
End With
```

Das folgende Beispiel veranschaulicht die Verwendung der `With`-Anweisung, um mehreren Eigenschaften desselben Objekts Werte zuzuweisen.

```
With Selection.Font  
    .Name = "Calibri"  
    .Size = 12  
    .Color = 6299648  
End With
```

Funktionen

Eine Funktion besteht aus einem geschlossenen Codeblock, der unter einem bestimmten Namen aufgerufen und ausgeführt werden kann. Dabei spricht man von einer *Funktion*. Eine Funktion kann eine beliebige Anzahl von ausführbaren Anweisungen enthalten. Außerdem ist es möglich, aus einer Funktion heraus weitere Funktionen aufzurufen. Die Funktion kann sich an einer beliebigen Stelle innerhalb des Codes befinden und auch von jeder beliebigen Stelle aus aufgerufen werden. Nach dem Beenden der Funktion wird der Programmablauf an der Stelle fortgesetzt, an der er durch den Aufruf der Funktion unterbrochen wurde.

Vorteile von Funktionen

Mit Hilfe von Funktionen können Sie eigene, in sich abgeschlossene Visual Basic-Prozeduren programmieren, die Sie dann über den Aufruf der Funktion ausführen können. Dabei können Sie bestimmen, bei welchem Ereignis (zum Beispiel, wenn der Anwender einen Button anklickt) die Funktion aufgerufen und ihr Programmcode ausgeführt wird. So vermeiden Sie mehrfache Nutzung verschiedener Programmteile. Mit Hilfe einer Funktion können Sie folgende Faktoren erreichen:

- **Übersichtlicher Programmaufbau:** Der in sich geschlossene Codeblock einer Funktion kann mehrfach und unabhängig vom Fortschritt des Programms aufgerufen werden. Damit bietet es sich geradezu an, häufig verwendete Routinen in einer Funktion zusammenzufassen. Damit sparen Sie viele Zeilen Programmcode, und das gesamte Programm wird transparenter und übersichtlicher.
- **Modularisierung und Wiederverwendbarkeit:** Ein weiterer Vorteil liegt in der einfachen Wiederverwendbarkeit und damit der Modularisierung von Code. Code, der in langer Liste eingebunden ist, kann meist nur schwer herausgefiltert und für andere Programme wiederverwendet werden. Eine Funktion hingegen lässt sich leicht an einer Stelle deklarieren oder definieren und diese aus verschiedenen Skripten in einem Film aufrufen. Mit der Verwendung von einheitlichen Namen für Variable schaffen Sie leicht überschaubare Schnittstellen zu diesen Funktionen. Damit können Sie zukünftige Programme in Modulbauweise zusammenstellen und verkürzen die Entwicklungszeit erheblich.

Definition einer Funktion

Eine `Function`-Prozedur ist eine Folge von `Visual Basic`-Anweisungen, die durch die Anweisungen `Function` und `End Function` eingeschlossen sind. Eine `Function`-Prozedur ähnelt einer `Sub`-Prozedur, kann aber auch einen Wert zurückgeben. Eine `Function`-Prozedur kann Argumente, wie z.B. Konstanten, Variablen oder Ausdrücke verwenden, die über die aufgerufene Prozedur übergeben werden. Wenn eine `Function`-Prozedur über keine Argumente verfügt, muß deren `Function`-Anweisung ein leeres Klammernpaar enthalten. Eine Funktion gibt einen Wert zurück, indem ihrem Namen ein Wert in einer oder mehreren Anweisungen der Prozedur zugewiesen wird.

Syntax:

```
[Public | Private] [Static] Function Name [(ArgListe)] [As Typ]
    [Anweisungen]
    [Name = Ausdruck]
    [Exit Function]
    [Anweisungen]
    [Name = Ausdruck]
End Function
```

Im folgenden Beispiel berechnet die `Celsius`-Funktion `Celsius`-Grade aus `Fahrenheit`-Graden. Wenn die Funktion aus der `Tempberechnen`-Prozedur aufgerufen wird, wird eine den Argumentwert enthaltende Variable der Funktion übergeben. Das Ergebnis der Berechnung wird an die aufrufende Prozedur zurückgegeben und in einem Meldungsfeld angezeigt.

```
Sub Tempberechnen()
    temp = InputBox(Prompt:= "Geben Sie die Temperatur in Fahrenheit ein.")
    MsgBox "Die Temperatur entspricht " & Celsius(temp) & " Grad Celsius."
End Sub

Function Celsius(fGrad)
    Celsius = (fGrad - 32) * 5 / 9
End Function
```

Sub-Anweisung

Prozedur ist eine benannte Folge von Anweisungen wie eine Funktion, die als Einheit ausgeführt werden. `Function`, `Property` und `Sub` sind zum Beispiel Prozedurtypen. Der Name einer Prozedur wird immer auf Modulebene definiert. Der gesamte ausführbare Code muss in einer Prozedur enthalten sein. Prozeduren können nicht in andere Prozeduren eingesetzt werden. `Sub`-Anweisung deklariert den Namen, die Argumente und den Code für den Rumpf einer `Sub`-Prozedur.

Syntax:

```
[Private | Public] [Static] Sub Name [(ArgListe)]
    [Anweisungen]
    [Exit Sub]
    [Anweisungen]
End Sub
```



Sub-Prozeduren sind standardmäßig öffentlich, wenn sie nicht explizit mit `Public` oder `Private` festgelegt werden. Wird `Static` nicht angegeben, so bleiben die Werte lokaler Variablen zwischen den Aufrufen nicht erhalten. Eine Sub-Prozedur können Sie nicht mit `GoSub`, `GoTo` oder `Return` aufrufen oder verlassen.

In diesem Beispiel wird die Sub-Anweisung verwendet, um Namen, Argumente und Prozedurrumpf einer Sub-Prozedur zu definieren.

Beispiel 31)

```
Sub SchreibInTextmark(ByVal sBmName As String, ByVal sBmText As String)
Rem Schreibt einen neuen Wert in ein vorhandenes Textmarke
  If ActiveDocument.Bookmarks.Exists(sBmName) Then
    Dim r As Range
    Set r = ActiveDocument.Bookmarks(sBmName).Range
    r.Text = sTmText
    ActiveDocument.Bookmarks.Add sBmName, r
  End If
End Sub
```

In diesem Beispiel wird die Sub-Anweisung verwendet, um Prozedurrumpf für die Zuordnung an einer Schaltfläche oder in einem Programmablauf zu definieren.

Beispiel 32)

```
'Löscht den Inhalt der Eingabefelder und Kombinationsfeld im Dialogblatt
' frmAufzinsung
Sub Maskeloeschen()
  Me.cboPerson.Clear
  Me.cboPerson.Text = ""
  Me.txtAnlage.Text = ""
  Me.txtKapital.Text = ""
  Me.txtProzent.Text = ""
  Me.cboLaufzeit.Text = ""
End Sub
```



Weitere Informationen über die Prozeduren finden Sie in der Visual Basic-Hilfe unter dem Thema "Statements Function and Sub ". <https://msdn.microsoft.com/en-us/library/office/jj692812.aspx>

3.6.1 Maßeinheiten

In Makros werden für alle Maßeinheiten generell Punkte verwendet. Bei der Einstellung der Schriftgröße ist uns das ja auch ganz geläufig. Wenn Sie aber zum Beispiel die Seitenränder Ihres Dokumentes einstellen möchten, so ist die Verwendung von Zentimetern sinnvoller. Zur Umrechnung von Punkten in andere Maßeinheiten stellt Word-VBA verschiedene Methoden zur Verfügung. Z.B. wird mit der Methode

```
CentimetersToPoints(Zentimeter-Angabe)
```

die angegebene Zentimeterzahl für die Word-internen Berechnungen in Punkte umgerechnet. Wollen Sie also in Ihrem Makro die Maßeinheit 10 cm benutzen, so müssen Sie schreiben:

```
CentimetersToPoints(10)
```

Weitere Methoden, die zum Teil vielleicht nicht so gebräuchlich sind, können Sie der nachstehenden Tabelle entnehmen:

Methode	Bedeutung	Berechnung
CentimetersToPoints(cm)	Zentimeter in Punkte	1 cm = 28,35 Punkte
MillimetersToPoints(mm)	Millimeter in Punkte	1mm = 2,85 Punkte
InchesToPoints(inch)	Zoll in Punkte	1 Zoll = 72 Punkte
LinesToPoints(lines)	Zeilen in Punkte	1 Zeile = 12 Punkte

Beispiel 33)

' Das folgende Makro öffnet ein neues Dokument und stellt die Seitenränder,
' sowie die Abstände vor und nach jedem Absatz ein.

```
Sub SeitenRänder()
    ' Seitenränder und Absatzabstände einstellen
    Documents.Add
    ' Seitenränder einstellen mit Zentimeter-Angaben
    With ActiveDocument.PageSetup
        .LeftMargin = CentimetersToPoints(3)
        .RightMargin = CentimetersToPoints(5)
        .TopMargin = CentimetersToPoints(2.5)
        .BottomMargin = CentimetersToPoints(2.5)
    End With
    ' Absatzabstände einstellen mit Zeilen-Angaben
    With ActiveDocument.Paragraphs
        .SpaceAfter = LinesToPoints(2)
        .SpaceBefore = LinesToPoints(1)
    End With
End Sub
```

' Das folgende Makro erzeugt eine Tabelle mit einer Zeile und zwei Spalten und stellt
' die 1. Spalte mit 7 cm und die zweite Spalte mit 5 cm ein.

```
Sub Tabellen12()
    ActiveDocument.Tables.Add Range:=Selection.Range, NumRows:=1, _
        NumColumns:=2, DefaultTableBehavior:=wdWord9TableBehavior, _
        AutoFitBehavior:= wdAutoFitFixed
    With Selection.Tables(1)
        If .Style <> "Tabellenraster" Then
            .Style = "Tabellenraster"
        End If
        .ApplyStyleHeadingRows = True
        .ApplyStyleLastRow = False
        .ApplyStyleFirstColumn = True
        .ApplyStyleLastColumn = False
        .ApplyStyleRowBands = True
        .ApplyStyleColumnBands = False
    End With
    ' Spaltenbreite einstellen

    Selection.Tables(1).Columns(1).PreferredWidthType=wdPreferredWidthPoints
    Selection.Tables(1).Columns(1).PreferredWidth = CentimetersToPoints(7)
    Selection.Move Unit:=wdColumn, Count:=1
    Selection.SelectColumn
    Selection.Columns.PreferredWidthType = wdPreferredWidthPoints
    Selection.Columns.PreferredWidth = CentimetersToPoints(5)
End Sub
```

3.6.2 den Cursor in Word-Dokumenten bewegen

Um sinnvolle Makros zu schreiben, ist es auf jeden Fall notwendig, den Cursor innerhalb des Dokumentes zu bewegen. Für Texteingaben und Markierungen, die wir später noch kennenlernen werden, kann man den Cursor zum Beispiel an den Anfang oder das Ende des Dokumentes platzieren oder auch um zwei Zeichen nach rechts oder unten bewegen. Die Stelle, an der sich der Cursor befindet, kann über das `Selection`-Objekt angesprochen werden. Für dieses Objekt existiert eine Reihe von Methoden, von denen wir uns im folgenden einige anschauen wollen.

Die Methode `GoTo`

Mit der `GoTo`-Methode können Sie den Cursor an einer ganz bestimmten Stelle im Dokument positionieren. Dazu benötigen Sie die Angabe verschiedener Parameter:

```
Selection.GoTo [what:=Element, which:=Richtung, count:=Zähler, name:=Name]
```

Die Angaben zu den Parametern `what` und `which` werden über Word-Konstante definiert. Einige Beispiele:

`what` definiert das Element, zu dem positioniert werden soll. Die zugehörige Konstante beginnt immer mit `wdGoTo` und endet mit dem Namen des Elementes:

<code>wdGoToBookmark</code> (Textmarken)	<code>wdGoToLine</code> (Zeilen)
<code>wdGoToField</code> (Felder)	<code>wdGoToObject</code> (Objekte)
<code>wdGoToFootnote</code> (Fußnoten)	<code>wdGoToPage</code> (Seiten)
<code>wdGoToGraphic</code> (Graphiken)	<code>wdGoToTable</code> (Tabellen)
<code>wdGoToHeading</code> (Überschriften)	<code>wdGoToSection</code> (Abschnitte)

Word-Konstante des `what`-Parameters

`which` gibt die Richtung an. Auch hier beginnen alle Konstanten mit `wdGoTo` und enden mit einer Angabe zur Richtung:

<code>wdGoToFirst</code> (erste)	<code>wdGoToLast</code> (letzte)
<code>wdGoToNext</code> (nächste)	<code>wdGoToPrevious</code> (vorherige)

Word-Konstante des `which`-Parameters

`Count` bestimmt die Anzahl der Elemente, um die der Cursor bewegt werden soll. Er ist voreingestellt auf den Wert 1 und muss immer positiv sein. Soll rückwärts positioniert werden, so muss das über das Argument `which` eingestellt werden.

Die Angabe eines Namens ist dann vorteilhaft, wenn Sie zum Beispiel eine bestimmte Textmarke, ein Feld oder einen Kommentar anspringen wollen.

Beispiel 34)

' Das folgende Makro positioniert den Cursor an verschiedene Stellen im Dokument
' unter Verwendung der unterschiedlichen Word-Konstanten.

```
Sub CursorBewegen()  
    ' die Auswahl in die vierte Zeile des Dokuments verschieben  
    Selection.GoTo What:=wdGoToLine, Which:=wdGoToAbsolute, Count:=4  
    ' die Auswahl um zwei Zeilen nach oben verschieben
```

```

Selection.GoTo What:=wdGoToLine, Which:=wdGoToPrevious, Count:=2
' zu nächstem Date-Feld springen
Selection.GoTo What:=wdGoToField, Name:="Date"
' zur ersten Zelle der nächsten Tabelle verschieben
Selection.GoTo What:=wdGoToTable, Which:=wdGoToNext
End Sub

```

Die Methoden `HomeKey` und `EndKey`

Die Tasten **Pos 1** und **Ende** können ebenfalls in der VBA-Programmierung benutzt werden. Dazu dienen die folgenden Methoden:

```

Selection.HomeKey [Unit, Extend]
Selection.EndKey [Unit, Extend]

```

`HomeKey` entspricht dabei der Taste **Pos 1** und `EndKey` der Taste **Ende**. Ob an das Ende der Zeile oder an das Ende des gesamten Dokumentes gesprungen werden soll, legen Sie über die `Unit` fest. Diese wird durch eine Word-Konstante repräsentiert.

<code>wdStory</code> (Dokument)	<code>wdColumn</code> (Tabellenspalte)
<code>wdLine</code> (Zeile)	<code>wdRow</code> (Tabellenzeile)

Word-Konstante der Methoden `HomeKey/EndKey`

Beispiel 35)

' Das folgende Makro positioniert den Cursor an verschiedene Stellen im Dokument unter
' Verwendung der Pos1/Ende-Tasten in Kombination mit den unterschiedlichen Units.

```

Sub CursorAnfangEnde()
' die Markierung an den Anfang der aktuellen Tabellenspalte verschieben,
' und anschließend die Markierung bis zum Ende der Spalte erweitern.
If Selection.Information(wdWithInTable) = True Then
    Selection.HomeKey Unit:=wdColumn, Extend:=wdMove
    Selection.EndKey Unit:=wdColumn, Extend:=wdExtend
End If
' die Markierung an das Ende der aktuellen Dokumentkomponente verschieben
Selection.EndKey Unit:=wdStory, Extend:=wdMove
End Sub

```

Die Methoden `Movexxx`

Eine Reihe weiterer Methoden verschiebt den Cursor ab der aktuellen Position in eine beliebige Richtung. Diese Methoden werden unter `Movexxx` zusammengefasst.

```

Selection.MoveUp           ' nach oben
Selection.MoveDown        ' nach unten
Selection.MoveLeft        ' nach links
Selection.MoveRight       ' nach rechts

```

Move-Methoden

Die genaue Syntax sieht für alle Methoden gleich aus:

```

Selection.Movexxx [Unit, Count, Extend]

```

Mit `Unit` wird dabei festgelegt, welche Einheit für die Cursorbewegung verwendet werden soll. Ähnlich, wie in den vorhergehenden Methoden, kommen auch hier wieder Word-Konstante zum Einsatz, die abhängig von der verwendeten Move- Methode sind:

Für `MoveUp` und `MoveDown`:

<code>wdLine</code> (Zeile)	<code>wdParagraph</code> (Absatz)
<code>wdWindow</code> (Rand des aktiven Fensters)	<code>wdScreen</code> (Bildschirmseite)

Für `MoveRight` und `MoveLeft`

<code>wdCharacter</code> (Zeichen)	<code>wdWord</code> (Wort)
<code>wdSentence</code> (Satz)	<code>wdCell</code> (Tabellen-Zelle)

Word-Konstante der `Move`-Methoden

Mit `Count` wird die Anzahl der Einheiten festgelegt, um die der Cursor bewegt werden soll. (Anmerkung: bei `Unit:=wdWindow` wird immer `Count:=1` gesetzt). Der Parameter `Extend` wird im folgenden Kapitel näher beschrieben.

Beispiel 36)

```
Sub Cursorverschieben()
    ' das Ausgewählte Zeichen um ein nach links verschieben
    Selection.MoveLeft Unit:=wdCharacter, Count:=1

    ' um 10 Linien nach oben verschieben
    Selection.MoveUp Unit:=wdLine, Count:=10

    ' die Markierung um 10 Zeilen nach unten erweitern.
    Selection.MoveDown Unit:=wdLine, Count:=10, Extend:=wdExtend
End Sub
```

Die Methoden `StartOf` und `EndOf`

Die Methoden `StartOf` und `EndOf` beziehen sich immer auf den Anfang bzw. das Ende einer Texteinheit, die über eine der üblichen Textkonstanten näher beschrieben wird. So eine Texteinheit kann ein Absatz, eine Tabelle, ein Abschnitt und ähnliches sein.

Die Syntax:

```
Selection.StartOf [Unit, Extend]
Selection.EndOf [Unit, Extend]
```

Als `Unit` können die nun schon bekannten Word-Konstanten verwendet werden.

Um zum Beispiel an den Anfang eines Absatzes zu springen, benutzen Sie den Befehl:

```
Selection.StartOf Unit:=wdParagraph
```

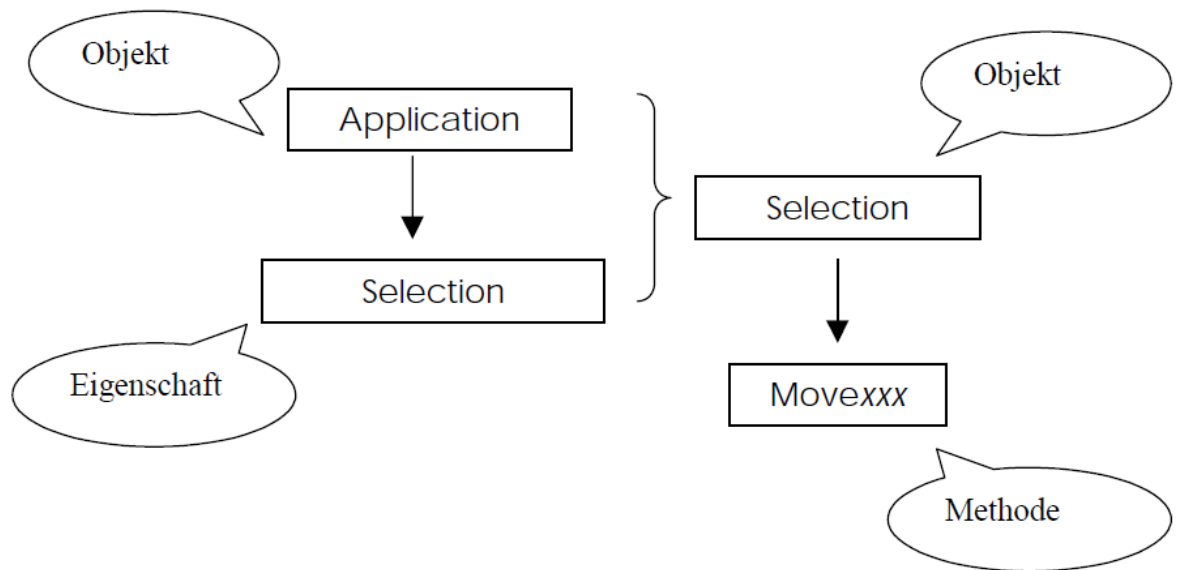
Die `Extend`-Angabe wird im nächsten Kapitel vorgestellt.

3.6.3 Word-Dokument markieren

Das A und O bei der Bearbeitung von Word-Dokumenten besteht in der Voraussetzung, Dokumentteile zu markieren. Bevor Sie Absätze verschieben oder löschen können, Text

formatieren oder kopieren, müssen Sie Word die zu bearbeitenden Stellen mit Hilfe der Markierung bekannt machen. Das geht natürlich auch in VBA-Makros.

Aus dem vorherigen Kapitel wissen wir bereits, dass die Cursor-Position als `Selection`-Objekt bezeichnet wird. Aber auch der markierte Teil eines Dokumentes wird so bezeichnet. Dieses Objekt ist immer ein zusammenhängender Bereich von Zeichen. Aus unserer täglichen Arbeit mit Word wissen wir, dass gleichzeitig immer nur ein Bereich markiert sein kann; folgerichtig existiert in einem Dokument immer genau ein `Selection`-Objekt. Genau genommen, ergibt sich übrigens das `Selection`-Objekt aus der `Selection`-Eigenschaft des `Application`-Objektes.



Statt

```
Application.Selection.MoveRight
```

Schreiben Sie dann lediglich

```
Selection.MoveRight
```

Um nun eine Markierung im Dokument zu erzeugen, können wir zum Teil die Methoden verwenden, die wir bereits im vorherigen Kapitel kennengelernt haben. Die Methoden werden lediglich um den Parameter `Extend` ergänzt.

'Wort markieren

```
Selection.MoveRight Unit:=wdWord, Extend:=wdExtend
```

'Alles bis zum Dokumentanfang markieren

```
Selection.HomeKey Unit:=wdStory, Extend:=wdExtend
```

'Den aktuellen Absatz markieren

```
Selection.StartOf Unit:=wdParagraph
```

```
Selection.MoveDown Unit:=wdParagraph, Extend:=wdExtend
```

'Absatz ab Cursorposition markieren

```
Selection.EndOf Unit:=wdParagraph, Extend:=wdExtend
```

Der Parameter `Extend` ist voreingestellt auf den Wert `Extend:=wdMove`. Dabei wird lediglich der Cursor versetzt, ohne dass eine Markierung erzeugt wird. Mit `Extend:=wdExtend` wird während der Cursorbewegung gleichzeitig markiert.

Die Methode `MoveEnd`

Die `MoveEnd`-Methode erweitert die Markierung ab Cursorposition/Markierungsende bis zum Ende des angegebenen Textbereiches. Hierbei wird immer markiert, es ist also nicht möglich, lediglich den Cursor zu verschieben. Aus diesem Grund ist auch kein `Extend`-Parameter erforderlich.

Die Syntax:

```
Selection.MoveEnd [Unit, Count]
```

Ist die `Count`-Angabe ein positiver Wert, so wird die Markierung in Richtung Dokumentende erweitert, ist der Wert negativ, so wird die Markierung Richtung Dokumentanfang reduziert.

'Markierung um drei Absätze erweitern

```
Selection.MoveEnd Unit:=wdParagraph, Count:=3
```

'Markierung um einen Absatz reduzieren

```
Selection.MoveEnd Unit:=wdParagraph, Count:=-1
```

Die Methode `WholeStory`

Die einfachste Möglichkeit, um das gesamte Dokument zu markieren, bietet die Methode `WholeStory`.

Die Syntax

```
Selection.WholeStory
```

Diese Methode zusammen mit der Methode `Cut` kommt in diesem Skript öfter vor, weil mehrere Aufgaben in einer Datei zusammengefasst worden sind. Für jede Aufgabe muss also Dokument markiert und anschließend ausgeschnitten werden.

Die Methode `Select`

Mit `Select` kann ein Element des Dokumentes markiert werden: das können zum Beispiel Wörter, Graphiken, Textmarken, Tabellen usw. sein.

Beispiele:

```
ActiveDocument.Words(1).Select  
ActiveDocument.Bookmarks(1).Select  
ActiveDocument.Tables(1).Select
```

3.6.4 Das `Range`-Objekt

`Range`-Objekte sind eines der wichtigsten Objekte in Word-VBA. Sie existieren auch in anderen Office-Anwendungen, haben aber unterschiedliche Bedeutungen.

In Word sind `Range`-Objekte definiert als eine zusammenhängende Folge von Zeichen. Ein Bereich (`Range`) hat einen genauen Anfangs- und Endpunkt; er kann das gesamte Dokument, Teile des Dokumentes oder auch lediglich eine einzelne Stelle enthalten. Sie sind zunächst einmal dem `Selection`-Objekt, das eine Markierung enthält sehr ähnlich, weisen aber einige Unterschiede auf:

- `Range`-Objekte sind unabhängig von der Markierung im Dokument. Sie verändern einen eventuell markierten Bereich nicht.

- Es können gleichzeitig mehrere Range-Objekte definiert und benutzt werden (zur Erinnerung: ein `Selection`-Objekt existiert genau einmal im Dokument).
- `Range`-Objekte sind im Prinzip Teile des Dokumentes, die über einen Namen angesprochen werden können. Dieser Name bleibt während der gesamten Ausführung eines Makros verfügbar, nicht aber über die Beendigung des Makros hinaus.

Neben dem `Range`-Objekt stellt Word-VBA auch eine `Range`-Methode zur Verfügung, mit der der Anfang und das Ende des `Range`-Objektes definiert werden können und eine `Range`-Eigenschaft, die andere Objekte zu `Range`-Objekten macht.

Die `Range`-Methode

Die `Range`-Methode macht aus einem Teilbereich des Dokumentes ein `Range`-Objekt und benennt diesen Teilbereich gleichzeitig. Diese Methode kann nur auf das `Document`-Objekt angewandt werden.

Die Syntax:

```
ActiveDocument.Range [(Start:=zahl, End:=zahl)]
```

Über die einzugebenden Zahlen wird der Anfang und das Ende des Ranges festgelegt. Dabei werden alle Zeichen mitgezählt, auch die nicht druckbaren und verborgen formatierten. Die Zählung beginnt mit 0 vor dem ersten Zeichen, wird mit 1 nach dem ersten Zeichen fortgeführt, usw. Werden die Parameter `Start` und `End` nicht angegeben, so wird das gesamte Dokument zum `Range`-Objekt.

Beispiele, angewandt auf den nachfolgenden Text

Das ist ein Beispiel

```
ActiveDocument.Range (Start:=0, End:=3)
' Das ist ein Beispiel
ActiveDocument.Range (Start:=12, End:=12)
' Das ist ein Beispiel
ActiveDocument.Range (Start:=8, End:=12)
' Das ist ein Beispiel
```

Die `Range`-Eigenschaft

Viele Objekte innerhalb von Word verfügen über die Eigenschaft `Range`. Wird sie auf ein Objekt angewandt, so ist das Ergebnis ein `Range`-Objekt.

Wollen Sie den ersten Absatz Ihres Dokumentes als `Range`-Objekt benutzen, so können Sie die folgenden Anweisungen verwenden:

```
Sub RangeObjekte()
    Dim Dok As Document
    Dim Absatz As Range
    Set Dok = ActiveDocument
    Set Absatz = Dok.Paragraphs(1).Range
End Sub
```

Die Variable `Absatz` enthält anschließend den Inhalt des ersten Absatzes. Weitere Objekte, für die Sie die `Range`-Eigenschaft verwenden können, sind z.B. `Character`, `Words`, `Sentences`, `Bookmarks` usw.

Die `Range`-Methode kann lediglich für das `Document`-Objekt benutzt werden. Dazu werden zusätzlich die Parameter `Start` und `End` zur Eingrenzung des `Range`-Objektes zur Verfügung

gestellt. Die Range-Eigenschaft wird von verschiedenen Objekten zur Verfügung gestellt, besitzt allerdings keine Parameter. Egal, ob Sie lieber die Methode oder die Eigenschaft verwenden: das Ergebnis ist immer ein Range-Objekt.

Selection **oder** Range?

Viele Funktionen lassen sich sowohl über Selection - als auch über Range - Objekte lösen. Da bei Verwendung von Range -Objekten aber die Positionierung und Markierung der Textbereiche entfällt, ist ihre Verwendung in der Regel den Selection -Objekten vorzuziehen. Um zum Beispiel den ersten Absatz eines Dokumentes zu löschen, würden Sie mit Selection -Objekten folgende Befehle benutzen:

```
Selection.HomeKey unit:=wdStory  
Selection.MoveDown unit:=wdParagraph, Extend:=wdExtend  
Selection.Delete
```

Mit einem Range -Objekt genügt die nachstehende Anweisung:

```
Dok.Paragraphs(1).Range.Delete
```

3.6.5 Word-Befehle verwenden

Die Verwendung von Word-Befehlen kann hier natürlich nicht komplett vorgestellt werden. Aber anhand von weiteren Beispielen sollen Sie zumindest einige Grundlagen erfahren, mit denen Sie selbst weiter experimentieren können.

Texte eingeben und Bearbeiten

Einige der vorangegangenen Beispiele haben bereits Texte in leere Dokumente eingefügt. Hier sollen nun einige weitere Befehle aufgezählt werden, mit deren Hilfe Texte eingegeben und auch wieder gelöscht werden können.

Die nachfolgenden Anweisungen beziehen sich alle auf die Cursor-Position im Dokument und verwenden dazu das Selection-Objekt.

```
Selection.TypeText Text := "Aufzinsung für die Kapitalanlagen"
```

Der angegebene Text wird in das Dokument eingetragen.

Alternativen zum Einfügen von Text mit der TypeText-Methode sind zum Beispiel die Methoden InsertAfter und InsertBefore.

Absatzmarke einfügen

Einen Absatz fügen Sie mit der Methode TypeParagraph ein:

```
Selection.TypeParagraph
```

Alternativ kann auch die Word-Konstante vbCr als Text ausgegeben oder die Methode InsertParagraph benutzt werden.

Seitenumbruch

Um eine neue Seite zu beginnen, verwenden Sie die Methode InsertBreak:

```
Selection.InsertBreak
```

Die InsertBreak-Methode fügt jedoch nicht nur Seitenwechsel, sondern auch Spalten- und Abschnittswchsel ein. Die Steuerung dieser Methode wird über diverse Word-Konstanten

vorgenommen. Voreingestellt ist `InsertBreak (wdPageBreak)`; deshalb erfolgt ohne Angabe einer Konstanten ein Seitenwechsel.

Sonderzeichen

Zur Eingabe von Sonderzeichen verwenden Sie die `InsertSymbol`-Methode. Dazu muss man allerdings genau wissen, in welchem Font sich das Sonderzeichen befindet und vor allen Dingen: an welcher Stelle im `Font` liegt das Symbol.

```
Selection.InsertSymbol CharacterNumber:=175, Font:="Symbol", Unicode:=False
```

Über den Parameter `Font` wählen Sie den gewünschten Zeichensatz aus, in unserem Beispiel den `Symbol`-Font. Die Angabe `CharacterNumber` legt den ANSI- Code des benötigten Zeichens fest, sofern `Unicode:=False` eingestellt ist. Der ANSI-Code eines Zeichens ergibt sich, indem Sie auf die Position des Zeichens in der Tabelle die Zahl 31 addieren. Über die ANSI-Codes 32 bis 255 können Sie somit alle Zeichen einer Zeichentabelle ansprechen. Der oben verwendete ANSI-Code 175 erzeugt einen Pfeil nach unten ↓.

Text löschen

Zeichen können links oder rechts vom Cursor gelöscht werden: links mit der Methode

`TypeBackspace`, rechts mit `Delete`.

```
Selection.TypeBackspace
```

Löscht, wie die Rückschritt-Taste, ein Zeichen links vom Cursor

```
Selection.Delete Unit:=wdWord, Count:=2
```

löscht zwei Wörter rechts vom Cursor. Der Befehl entspricht der `Entf`-Taste.

Text überschreiben

Mit der `Text`-Eigenschaft eines `Selection`- oder `Range`-Objektes können Sie Text überschreiben. Die nachfolgende Anweisung überschreibt das erste Wort des Dokumentes:

```
ActiveDocument.Words(1).Text = "Die "
```

Texte formatieren

Wir haben es in den vorangegangenen Kapiteln schon häufig benutzt: Die Formatierung von Texten. Sowohl Zeichen- als auch Absatzformate können verwendet werden.

Zeichenformate

Zeichenformate werden über das `Font`-Objekt festgelegt. Dazu wird die `Font`- Eigenschaft eines `Selection`- oder `Range`-Objektes verändert.

Beispiele:

```
With Selection.Font
    .Bold = True
    .Name = "Courier New"
    .Size = 10
    .Underline = wdUnderlineSingle
    .Outline = True

    .Color = wdColorBlue
End With
```

Das vorstehende Beispiel setzt den markierten Text in einer 10pt großen, fetten `Courier New`-Schrift Blaufarbe, verwendet eine einfache Unterstreichung. Gleichzeitig wird die

Schrift als Outline verwendet – es werden also nur die äußeren Linien gezeichnet.

Absatzformate

Um Absätze zu formatieren, können verschiedene Eigenschaften des Paragraph-Format-Objektes benutzt werden.

```
With ActiveDocument.Paragraphs(1).Range.ParagraphFormat
    .Alignment = wdAlignParagraphJustify
    .FirstLineIndent = CentimetersToPoints(1)
    .LeftIndent = CentimetersToPoints(1)
    .RightIndent = CentimetersToPoints(1)
    .SpaceBefore = 6
    .SpaceAfter = 6
    .Space15
End
```

Der erste Absatz des Dokumentes wird in Blocksatz erstellt. Gleichzeitig wird er rechts und links um einen Zentimeter eingerückt. Die erste Zeile wird zusätzlich um einen Zentimeter eingerückt. Vor und nach dem Absatz werden 6 Punkte Abstand gesetzt; der Zeilenabstand wird durch `.Space15` auf anderthalb Zeilen festgelegt.

Zeichen- als auch Absatzformate zurücksetzen

Um sowohl Zeichen- als auch Absatzformate wieder zurückzusetzen auf das Format, das in der mit dem Text verbundenen Formatvorlage vorgegeben ist, können Sie die `Reset`-Methode verwenden:

```
With ActiveDocument.Content
    .Font.Reset
    .ParagraphFormat.Reset
End With
```

Formatvorlagen

Formatvorlagen können über die `Style`-Eigenschaft von `Range/Selection`-Objekten zugewiesen werden.

```
ActiveDocument.Content.Style = "Standard Absatz"
```

Der obige Befehl verbindet den gesamten Text des Dokumentes mit der Formatvorlage "Standard Absatz".

Rahmen und Schattierungen

Zum Einrahmen und Schattieren von Absätzen verwenden Sie die Eigenschaft `Borders` bzw. `Shading`

```
With ActiveDocument.Paragraphs(1)
    .Borders(wdBorderBottom).LineStyle = wdLineStyleDouble
    .Borders(wdBorderTop).LineStyle = wdLineStyleSingleWavy
    .Borders(wdBorderTop).ColorIndex = wdDarkRed
    .Shading.Texture = wdTexture10Percent
    .Shading.BackgroundPatternColorIndex = wdYellow
End With
```

Der erste Absatz des Dokumentes erhält eine untere doppelte Rahmenlinie, eine obere gewellte Rahmenlinie in rot, und eine gelbe Hintergrundfarbe mit 10%-Schattierung. Ist das angegebene Objekt kein Absatz, sondern ein Wort, so wird der Rahmen um das Wort gesetzt. `Borders` ist eigentlich eine Objekt-Auflistung, die die vier einzelnen Seiten, die es einzurahmen gilt, enthält. In einer Schleife können somit alle vier Seiten ein Rahmenformat zugewiesen bekommen.

Seitenrahmen

Auch Seitenrahmen können mit der `Borders`-Eigenschaft erstellt werden:

```
For Each Seite In ActiveDocument.Sections(1).Borders
    Seite.ArtStyle = wdArtChampagneBottle
    Seite.ArtWidth = 12
Next
```

Mit `ArtStyle` wird über eine Konstante das Aussehen des Rahmens festgelegt und mit `ArtWidth` die Breite des Rahmens. Obiges Beispiel erstellt für alle vier Seiten einen Rahmen, der Champagnerflaschen in einer 12-Punkte-Größe enthält. Da Seitenrahmen für die verschiedenen Abschnitte des Dokumentes getrennt definiert werden können, wird obiger Rahmen für den ersten Abschnitt festgelegt.

Aufzählungen und Nummerierungen

Zum Erstellen von Aufzählungen und Nummerierungen verwenden Sie die `ListFormat`-Eigenschaft, die weitere Methoden zur Verfügung stellt:

```
ActiveDocument.Content.ListFormat.ApplyBulletDefault
ActiveDocument.Content.ListFormat.ApplyNumberDefault
```

Der erste Befehl versieht alle Absätze im Dokument mit dem standardmäßig voreingestellten Aufzählungszeichen, der zweite Befehl erzeugt stattdessen eine Standardnummerierung.

3.6.6 Word Textmarke

Textmarken haben in Word mehrere Vorteile:

- schnelles Springen zu einer bestimmten Stelle, die Textmarke quasi als Lesezeichen
- die Textmarke als Verweismöglichkeit auf Seiten ohne Überschriften, so kann bequem im Text auch auf andere Seiten verwiesen und um dies zu automatisieren wird hier die Textmarke eingesetzt und die Seitennummer wird automatisch korrekt gesetzt.
- Über Textmarken können Berechnungen durchgeführt werden.

Textmarken einfügen

Verwenden Sie die `Add`-Methode einem Dokumentbereich ein Lesezeichen hinzu. Das folgende Beispiel aktiviert die Auswahl durch Hinzufügen der Textmarke mit dem Namen `"tmPerson"`.

```
With ActiveDocument.Bookmarks
    .Add Range:=Selection.Range, Name:="tmPerson"
    .DefaultSorting = wdSortByName
    .ShowHidden = False
End With
```

Textmarke wieder finden

Mithilfe der `Exists`-Methode bestimmen, ob eine Textmarke bereits in der Auswahl, Bereich oder im Dokument vorhanden ist. Im folgende Beispiel wird sichergestellt, dass die Textmarke `"tmPerson"` im aktiven Dokument vorhanden ist, bevor die Auswahl der Textmarke geprüft.

```
If ActiveDocument.Bookmarks.Exists("tmPerson") = True Then
    ActiveDocument.Bookmarks("tmPerson").Select
End If
```

3.6.7 Mit Tabellen arbeiten

Um Tabellen in Ihr Dokument einzufügen, können Sie den Befehl

```
ActiveDocument.Tables.Add [Range, numRows, numColumns]
```

benutzen. Dazu muss zunächst die Stelle über den Parameter Range angegeben werden, an der die Tabelle eingefügt werden soll. Anschließend wird die Anzahl der Zeilen und Spalten festgelegt.

Eine Tabelle an der aktuellen Cursorposition mit 5 Zeilen und 2 Spalten können Sie einfügen mit:

```
ActiveDocument.Tables.Add Range:=Selection.Range,numRows:=5, numColumns:=2
```

Alternativ können Sie eine Tabelle auch folgendermaßen einfügen:

```
Dim MyTab As Table  
Set MyTab = ActiveDocument.Tables.Add(Selection.Range, 5, 2)
```

Tabelle füllen

Zum Füllen der Tabelle kann zum Beispiel folgende Schleife dienen:

```
With ActiveDocument.Tables(1)  
  For zeile = 1 To 5  
    For spalte = 1 To 2  
      .Cell(zeile, spalte).Range.InsertAfter "Zelle " & _  
        zeile & "," & spalte  
    Next  
  Next  
End With
```

Weitere Beispiele zeigen, wie man auf die einzelnen Bestandteile der Tabelle zugreifen kann. Vorausgesetzt, wir haben folgende Vereinbarung getroffen:

```
Dim MyTab As Table  
Set MyTab = ActiveDocument.Tables(1)
```

Dann können wir die letzte Zelle der Tabelle mit Inhalt füllen:

```
MyTab.Cell(MyTab.Rows.Count,  
MyTab.Columns.Count).Range.InsertAfter "Letzte Zelle"
```

Die erste Zeile löschen

```
MyTab.Rows(1).Delete
```

Alle Spalten auf die optimale Breite setzen

```
MyTab.Columns.AutoFit
```

Eine neue Spalte vor der zweiten Spalte einfügen

```
MyTab.Columns.Add BeforeColumn:=MyTab.Columns(2)
```

Die neue Spalte auf 5cm Breite setzen, wobei alle anderen Spaltenbreiten unverändert bleiben

```
MyTab.Columns(2).SetWidth _  
columnwidth:=CentimetersToPoints(5), _  
RulerStyle:=wdAdjustNone
```

Die erste Zeile mit einem 3D-Rahmen versehen

```
For Each seite In MyTab.Rows(1).Borders  
  seite.LineStyle = wdLineStyleEmboss3D  
Next
```

Die zweite Spalte grau schattieren

```
MyTab.Columns(2).Shading.BackgroundPatternColorIndex = wdGray25
```

AutoFormate verwenden

```
MyTab.AutoFormat Format:=wdTableFormatColorfull
```

3.6.8 Fußnoten und Endnoten einfügen

Um eine automatisch nummerierte Fußnote an der Stelle einzufügen, an der sich der Cursor befindet, verwenden Sie nachfolgenden Befehl:

```
ActiveDocument.Footnotes.Add Range:=Selection.Range,  
Text:="Dies ist eine Fußnote"
```

Endnoten können Sie analog über den Befehl `Endnotes` einfügen. Mit Hilfe weiterer Eigenschaften können Sie die Startnummer und das Format der Nummerierung vorgeben. Um zum Beispiel ab dem Wert 2 mit Symbolen zu nummerieren, benutzen Sie die folgenden Eigenschaften:

```
ActiveDocument.Footnotes.NumberStyle = wdNoteNumberStyleSymbol  
ActiveDocument.Footnotes.StartingNumber = 2
```

3.6.9 Kopf- und Fußzeilen einfügen

Das Einfügen von Kopf- und Fußzeilen können Sie folgendermaßen vornehmen:

```
With ActiveDocument.Sections(1)  
    'Text für alle Kopfzeilen definieren  
    .Headers(wdHeaderFooterPrimary).Range.Text = "Dies ist eine Kopfzeile"  
    'Doppelte Linie unter den Text setzen  
    .Headers(wdHeaderFooterPrimary).Range.Borders _  
        (wdBorderBottom).LineStyle = wdLineStyleDouble  
    'Text für alle Fußzeilen definieren und  
    'Cursor auf den zweiten TabStop setzen  
    .Footers(wdHeaderFooterPrimary).Range.Text = "Seite" & vbTab & vbTab  
End With
```



Für alle weiteren Befehle, die Sie unter Word benutzen können, stehen entsprechende VBA-Objekte, -Methoden und -Eigenschaften zur Verfügung. Details zur Verwendung und jede Menge Beispiele beschreibt die Online-Hilfe des VBAEditors. <https://msdn.microsoft.com/de-de/library/office/ff837519.aspx>

4 Dialogblatt (Userformblatt)

Für die Programmsteuerung ist es oft nötig, Meldungen an Benutzer auszugeben bzw. Eingaben des Benutzers ins Programm zu übernehmen. Diese Aufgaben können vom System

zur Verfügung gestellte Dialogfunktionen oder individuell durch den Benutzer definierte Oberflächen übernehmen. Die Handhabung der System - Elemente wird im folgende näher beschrieben.

4.1 MsgBox

Die in den bisherigen Beispielen 5 und 8 im Abschnitt 3. 5 schon öfter benutzte MSGBOX ist eine den Programmablauf unterbrechende Funktion, die eine Meldung in einem Dialogfeld anzeigt und auf die Auswahl einer Schaltfläche wartet.

Die Funktion hat die Syntax:

```
MsgBox (Prompt [, Buttons][,Title][,Helpfile][,Context]
```

PROMPT	ist ein Zeichenfolgeausdruck mit der maximalen Länge von 1024 Zeichen, der als Meldung im Dialogfeld der <code>MSGBOX</code> erscheint. Soll der Meldungstext aus mehreren Zeilen bestehen, müssen die Zeilen durch manuelles Einfügen von Zeilenumbrüchen mittels der Funktion <code>CHR</code> (Zeichen <code>CHR(13)</code>) umbrochen werden.
BUTTONS	ist ein numerischer Ausdruck mit einem kombinierten Wert, der die Anzahl und den Typ der Schaltflächen, ein evtl. verwendetes Symbol, die aktivierte Schaltfläche und die Bindung des Dialogfeldes.
TITEL	ist ein Zeichenfolgeausdruck, der als Titel der Dialogbox erscheinen soll.
HELPPFILE	Nur in Verbindung mit <code>CONTEXT</code> zu verwenden – definiert die kontextbezogene Hilfedatei für das Dialogfeld-
CONTEXT	Numerischer Ausdruck, der dem Hilfethema in der unter <code>HELPPFILE</code> zugeordneten Hilfedatei zugeordnet ist

Die einfachste Form der Anwendung ist die Angabe der Funktion nur mit `PROMPT`:

```
MsgBox "Geben Sie eine reelle Zahl ein!"  
MsgBox Me.txtKapital.Text * 1,19  
MsgBox Me.cboLaufzeit.Text
```

Wird das Argument `BUTTONS` verwendet, müssen entweder numerische Werte oder Konstanten zur Definition der Schaltflächen angegeben werden:

Schaltflächen:

Wert	Konstante	Funktion
0	<code>vbOKOnly</code>	(Voreinstellung) Schaltfläche OK erzeugen
1	<code>vbOKCancel</code>	OK und ABBRECHEN erzeugen
2	<code>vbAbortRetryIgnore</code>	ABBRECHEN, WIEDERHOLEN IGNORIEREN erzeugen
3	<code>vbYesNoCancel</code>	JA, NEIN, ABBRECHEN erzeugen
4	<code>VbYesNo</code>	JA, NEIN erzeugen
5	<code>VbRetryCancel</code>	WIEDERHOLEN, ABBRECHEN erzeugen

Dialogfeldsymbole:

Wert	Konstante	Funktion
16	<code>vbCritical</code>	Stop – Symbol
32	<code>vbQuestion</code>	Fragezeichen - Symbol
48	<code>vbExclamation</code>	Ausrufezeichen – Symbol

64	vbInformation	Info - Symbol
----	---------------	---------------

Aktiviere Schaltflächen:

Wert	Konstante	aktivierte Schaltfläche
0	vbDefaultButton1	erste Schaltfläche
256	vbDefaultButton2	zweite Schaltfläche
512	vbDefaultButton3	dritte Schaltfläche

Bindung (Modalverhalten) des Dialogfeldes:

Wert	Konstante	Funktion
0	vbApplicationModal	Gebunden an die Anwendung. Die aktuelle Anwendung kann nur fortgesetzt werden, wenn der MSGBOX – Dialog beendet wird. Alle anderen Anwendungen sind nicht betroffen.
256	vbSystemModal	Systemgebunden – alle Anwendungen werden angehalten, bis der MSGBOX – Dialog beendet ist.

Bei der Definition der Schaltflächen und Symbole können entweder die Wert oder die Konstanten benutzt werden.

Soll beispielsweise ein Dialogfeld mit einer JA-, einer NEIN – Schaltfläche, versehen mit dem Fragezeichen – Symbol und der aktivierten NEIN – Schaltfläche erzeugt werden, geschieht es in der folgenden Form:

```
vbYesNo + vbQuestion + vbDefaultButton2
```

Oder

```
4 + 32 + 256
```

oder einfach

```
292
```

wobei die letzte Version die Summe der Einzelwerte ist. Hier ist allerdings kaum erkennbar, was genau definiert wurde. Ein Dialogfeld mit einer JA-, einer NEIN – Schaltfläche, versehen mit dem Fragezeichen – Symbol und der aktivierten NEIN – Schaltfläche, welches die Meldung „Soll der Datensatz gelöscht werden?“ anzeigt und den Titel „Datensatz löschen“ besitzt, kann durch die folgende Anweisung erzeugt werden:

```
MsgBox " Soll die erste Zeile gelöscht werden?", vbYesNo + vbQuestion + _  
vbDefaultButton2, "Daten löschen"
```

Soll der durch das Dialogfeld gelieferte Rückgabewert ausgewertet werden, muss die Funktionsschreibweise (mit Argumenten-klammern!) benutzt werden:

```
Ergebnis = MsgBox("Soll die erste Zeile gelöscht werden?", vbYesNo + _  
vbQuestion + vbDefaultButton2, "Zeile löschen")
```

Der zugewiesene Rückgabewert der Variablen Ergebnis kann anschließend ausgewertet werden.

Die Auswertung des Rückgabewertes wird meistens in einer Abfrage realisiert:

```
If Ergebnis = vbYes then....
```

Oder

```
If Ergebnis = 6 then....
```

Der Umweg über die Variable kann gespart werden, wenn in die Abfrage die MSGBOX- Funktion aufgenommen wird:

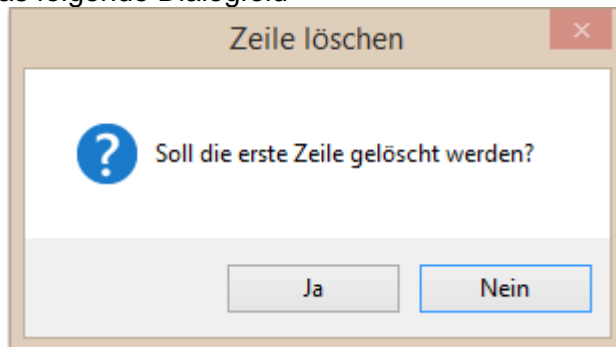
```
Private Sub cmdloeschen_Click()  
Dim Meldung As String, Titel As String, Stil As Integer  
Dim Ergebnis As Integer
```

```

Meldung = "Soll die erste Zeile gelöscht werden?"
Titel = "Zeile löschen"
Stil = vbYesNo + vbQuestion + vbDefaultButton2
Ergebnis = MsgBox(Meldung, Stil, Titel)
If Ergebnis = vbYes Then
    ActiveDocument.Tables(1).Rows(1).Delete
Else
    Exit Sub
End If
End Sub

```

Das Beispiel erzeugt das folgende Dialogfeld



Weitere Informationen finden Sie in der Visual Basic-Hilfe unter dem Thema "Functions".

<https://msdn.microsoft.com/en-us/library/office/gg251811.aspx>

4.2 Die Methode InputBox

Die Methode `INPUTBOX` unterscheidet sich von der Funktion `INPUTBOX` vor allem dadurch, dass sie eine Möglichkeit bittet, den Typ des eingegebenen Wertes zu definieren. Hierzu wird die Syntax um das Argument `TYPE` erweitert:

```

Object.InputBox(Prompt [,Titel] [,default] [,Left] [,Top]
[,helpfile] [,context] [,Type])

```

Wird dieses Argument nicht angegeben, so wird der Eingabewert als Zeichenfolge interpretiert und bei der Übergabe an eine Zelle je nach Form als Zahl, Text, boolescher Wert oder Formel interpretiert. Dem Argument `TYPE` können folgende Werte zugewiesen werden:

Wert	Typ
0	Formel
1	Numerisch
2	Text
4	Logisch
8	Zellbezug (Bereichs – Objekt)
16	Fehlerwert (z.B. #NV)
64	Wertematrix

Das folgende Beispiel demonstriert die Anwendung der `InputBox` – Methode zum Eingeben von `seiten` mit Werten, um die Anzahl der Seiten zu drucken. Über das Dialogfeld wird der Benutzer aufgefordert, jeweils eine Zahl für `von Seite` bis `Seite` einzugeben.

```

Private Sub cmdDrucken_Click()
    Dim Seite As String
    Dim Seite2 As String

```

```

ActivePrinter = "Adobe PDF"
Seite = InputBox("von Seite!", "Eingabe")
If Val(Seite) < 1 Then Exit Sub
Seite2 = InputBox("bis Seite!", "Eingabe")
If Val(Seite2) < Seite Then Exit Sub

ActiveDocument.ActiveWindow.PrintOut _
Range:=wdPrintFromTo, From:=Seite, To:=Seite2, Copies:=1, Collate:=True

Rem ActivePrinter = "Druckername"
End Sub

```

Die Anweisungen des Beispiels erzeugen das Dialogfeld (hier das erste Dialogfeld):

Nach Bestätigung mit Ok das zweite Dialogfeld:



Weitere Informationen finden Sie in der Visual Basic-Hilfe unter dem Thema "Functions".
<https://msdn.microsoft.com/en-us/library/office/gg251811.aspx>

4.3 Steuerelemente aus *Selbstdefinierte Dialoge - Userformblätter*

Für den Aufbau eines Userdialogs wird in der VBA – Entwicklungsumgebung über die Funktionskombination EINFÜGEN / USERFORM ein leeres Dialogformular (Abb. 4.3) erzeugt.

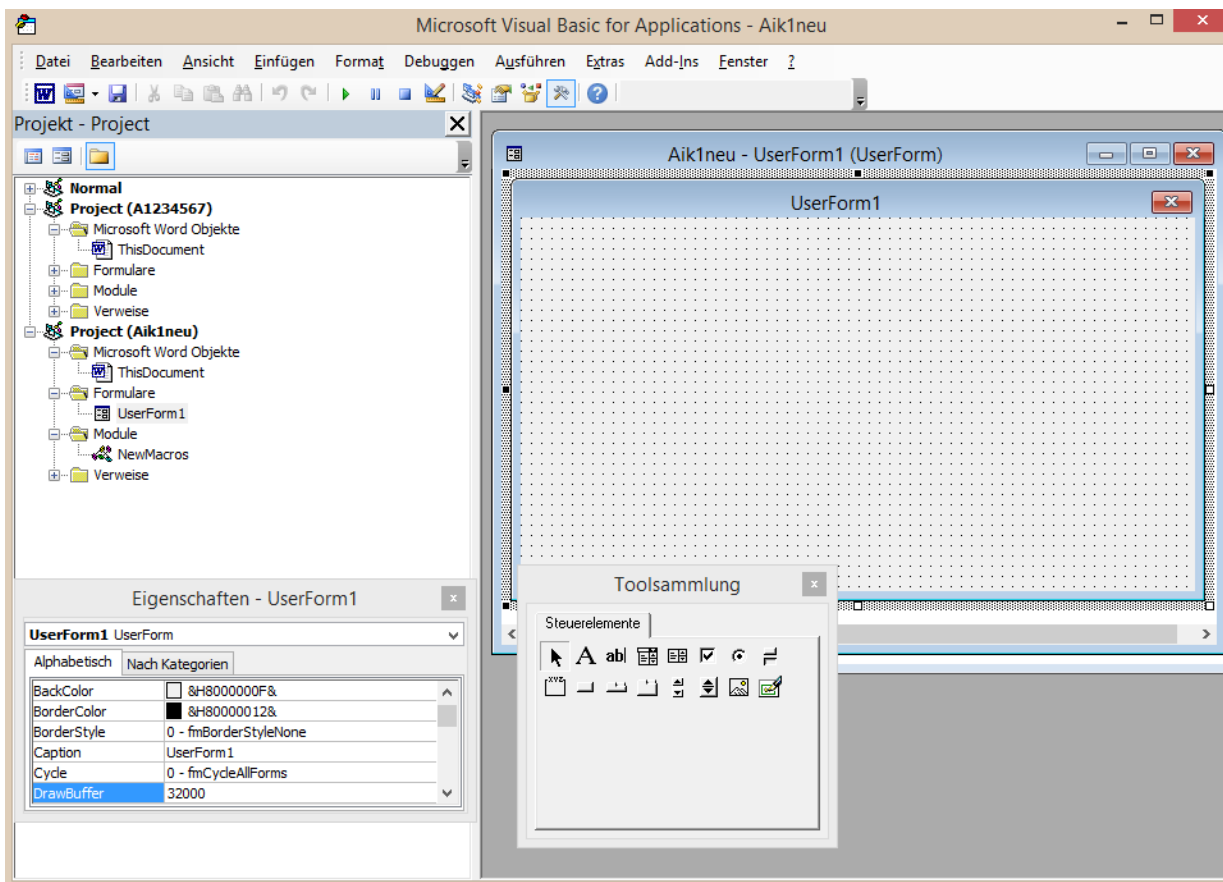


Abb. 4.3 leere Userform

Gleichzeitig wird die für den Formularaufbau benötigte Werkzeugsammlung (Toolsammlung) eingeblendet.

4.3.1 Einfügen der Elemente ins Userformblatt

Die Steuerelemente werden in der Symbolleiste angeklickt und bei gedrückter linker Maustaste ins Userformblatt eingetragen (z.B. Rahmen zeichnen, nach dem Loslassen der Maustaste wird das Steuerelement in der eingezeichneten Größe angezeigt).

Die Bedeutung der Symbole (von Links nach rechts und von Oben nach Unten):

Objekte auswählen	Zur Auswahl der Objekte bzw. Steuerelemente in einem Formular.
Beschriftungsfeld (LABEL) Oder Bezeichnungsfeld	Zu Hinweise, Beschriftungen oder kurze Anleitungen auf einem Formular.
Textfeld (TEXTBOX)	Zur Aufnahme von Eingaben während der Laufzeit eines Programms.
Kombinationsfeld (COMBOBOX)	Kombination von Listenfeld und Textfeld.
Listenfeld (LISTBOX)	Definition von Auswahllisten.
Kontrollfeld (CHECKBOX)	Kann die Zustände aktiv / inaktiv annehmen. Für die Aufnahme von Optionen zu verwenden. Mehrere Kontrollfelder in einem Dialogblatt sind voneinander unabhängig.
Optionsfeld (OPTIONBUTTON)	Ähnlich der CHECKBOX. Kann allerdings zu Gruppen aus mehreren Elementen zusammengefasst werden, in denen nur ein Element aktiv sein muss. Wird ein Element aktiviert, deaktiviert die Aktion ein anderes.
Umschaltfläche (TOGGLEBUTTON)	Wie Schaltfläche, jedoch zur Benutzung als Auswahl von mehreren alternativen Schaltflächen (aktiv = abgesenkt).
Rahmen (FRAME)	Zur Erstellung einer Optionsgruppe oder auch zur Gruppierung der

	Steuerelemente.
Schaltfläche (COMMANDBUTTON)	Schalterschaltfläche für dein Ereignis
Register (TABSTRIP)	Um inhaltlich zusammenhängende Steuerelemente in Gruppen mit jeweils unterschiedlichen Informationen zusammenzustellen und anzuzeigen.
Multiseiten (MULTIPAGE)	Zur Zuordnung der unterschiedlichen Kategorien, wenn mit größeren Mengen an Informationen gearbeitet wird.
Bildlaufleiste(SCROLLBAR)	Für die Eingabe numerischer Werte über Inkrement- und Dekrementschalter, Verschieben einer Schaltfläche oder Anklicken der Innenfläche.
Drehfeld (SPINBUTTON)	Für die Eingabe numerischer Werte über Inkrement- und Dekrementschalter.
Bildfeld (PICTURE) oder Anzeige	Feld zur Aufnahme einer Grafik.

4.3.2 Eigenschaften der Elemente

Die Steuerelemente können mit wesentlich mehr Eigenschaften ausgestattet werden als die der Formularleiste. Die Eigenschaften können per Dialogfenster (Abb. 4.3.2) oder direkt im Programm geändert werden.

Der Eigenschaften – Dialog wird aufgerufen, indem aus dem nach dem Klick mit der rechten Maustaste auf ein Steuerelement erscheinenden Auswahlmene die Position EIGENSCHAFTEN gewählt wird oder im Menü EINSICHT den Menüpunkt EIGENSCHAFTENFENSTER oder auch Klicken auf die Taste F4 gewählt wird.

Die Listen der daraufhin erscheinenden Eigenschaften von Steuerelementen sind unterschiedlich lang. Einige der Eigenschaften (Größe, Position) sind einfacher per Mausklick einzustellen, andere sollten über das Dialogfeld definiert werden.

Die Steuerelemente weisen eine Reihe gemeinsamer Eigenschaften auf, die hier vorab vorgestellt werden sollen. Sie können (neben vielen anderen) auch im Objektkatalog beim jeweiligen Control – Objekt eingesehen werden. Diese Eigenschaften können sowohl über den Eigenschaftsdialog, als auch insbesondere zur Laufzeit eines Programms zugewiesen bzw. verändert werden.

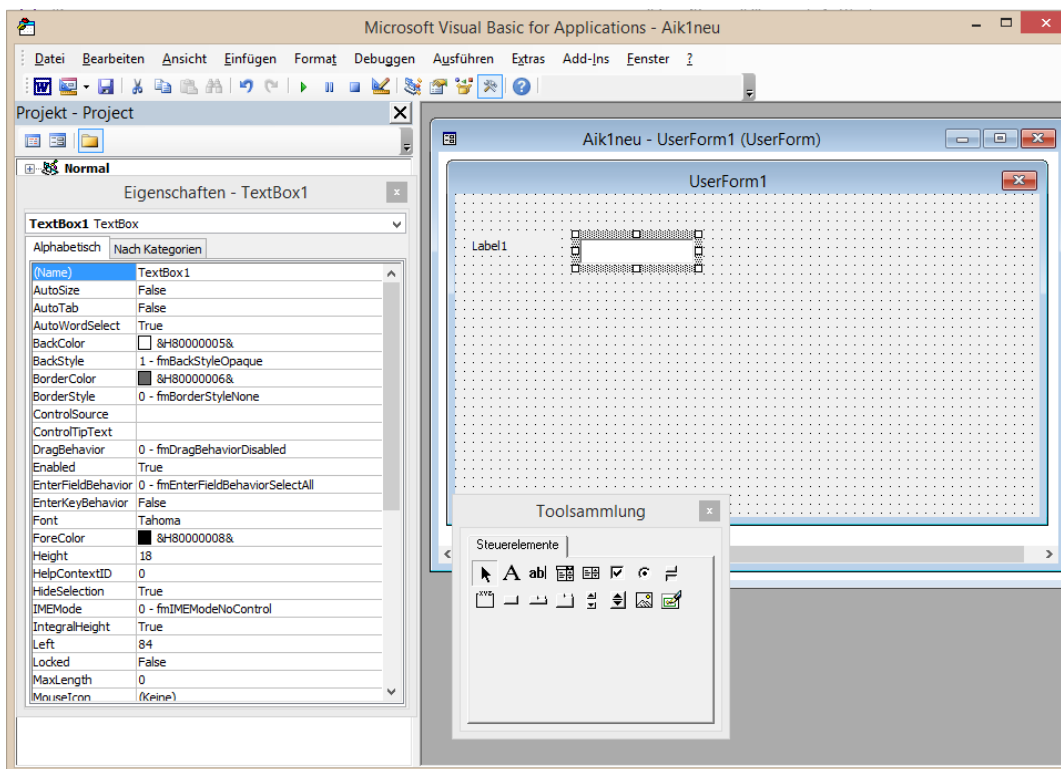


Abb. 4.3.2 Eigenschaften am Beispiel eines Textfeldes

Gemeinsame Eigenschaften

NAME	Name des Steuerelements. Muss eindeutig sein Ohne Namen lässt sich das Steuerelement nicht im Programm verwenden. Word bzw. MS-Office vergeben automatisch einen Namen, sobald ein Element im Dialogformular eingesetzt wird. Der Name kann verändert werden, um aussagefähige individuelle Namen benutzen zu können.
CAPTION	Eigenschaft für Schalter, Options- und Kontrollfelder. Steht für die Beschriftung der Steuerelemente.
TEXT	Eigenschaft, die den Inhalt von Steuerelementen steuert, z.B. Inhalt von Textfeldern.
VALUE	Gibt den Zustand oder Inhalt eines angegebenen Steuerelements an, z.B. den Inhalt eines Textfeldes oder den Zustand einer Schaltfläche oder Optionsfeldes usw.
CANCEL	= TRUE, wenn das Steuerelement durch ESC-Taste verlassen werden kann. Wird normalerweise verwendet, damit die nicht abgeschlossene Änderung abgebrochen werden kann und den vorhergehenden Zustand eines Formulars wieder hergestellt werden kann.
CONTROLTIPTEXT	Gibt den Text an, der angezeigt wird, wenn der Benutzer den Mauszeiger eine Weile über ein Steuerelement hält, ohne zu klicken.
TABINDEX	Gibt die Position eines einzelnen Objekts in der Aktivierreihenfolge des Formulars an.
TABSTOP	= TRUE, wenn das Element mit der Tab – Taste ausgewählt werden kann. Zeigt an, ob ein Objekt den Fokus erhalten kann, wenn die TAB - Taste drückt wird.
VISIBLE	= TRUE, wenn das Element sichtbar werden soll.

Gemeinsame Eigenschaften für die Stilbestimmung

BACKCOLOR	Bestimmt die Hintergrundfarbe eines Steuerelements (für Text-, Listen-, Options- und Kontrollfelder).
BORDERCOLOR	Bestimmt die Rahmenfarbe (für Text-, Listen-, Options- und Kontrollfelder).
BORDERSTYLE	Betsimmt den Rahmentyp (für Text-, Listen-, Options- und Kontrollfelder).
FONT	Schriftart und Schriftattribute.
FORECOLOR	Vordergrundfarbe des Elements.
SHADOW	Darstellung eines Schattens.
HEIGHT	Höhe des Elements.
WIDTH	Breite des Elements

SPECIALEFFEKT	Zuweisung von 2D / 3D – Effekten.
Gemeinsame Methoden	
SETFOCUS	Setzt den Eingabefokus auf das Element.

4.3.3 Das Bezeichnungsfeld (Label)

Das Bezeichnungsfeld wird zur Beschriftung eines Dialogs benutzt. Das wird neben Steuerelementen wie z.B. Textfeld gestellt, um sie zu beschriften bzw. Hinweise zu deren Benutzung geben zu können.



In diesem Skript wurden Namen der Bezeichnungsfelder nicht geändert sondern vom System übernommen.

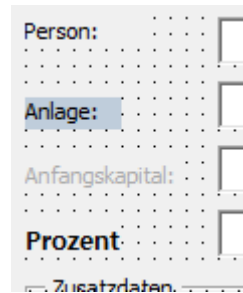


Abb. 4.3.3 Unterschiedliche Formen des LABEL-Feldes

Wichtigste Eigenschaften:

- ✓ Der darzustellende Text wird über die Eigenschaft CAPTION definiert.
- ✓ Bei mehrzeiligen Texten muss die Eigenschaft WORDWRAP auf TRUE gesetzt werden.
- ✓ Die Textausrichtung wird über die Eigenschaft TEXTALIGN geregelt.
- ✓ Soll sich die Größe des Feldes an die Länge des Textes anpassen, muss AUTOSIZE auf TRUE gesetzt werden.
- ✓ Schriftart und Schriftfarbe werden über FONT, BACKCOLOR bzw. FORECOLOR eingestellt.
- ✓ Die Art der Umrandung kann über BORDERSTYLE und BORDERCOLOR definiert werden.
- ✓ Mit PICTURE und PICTUREPOSITION kann eine Bitmap eingesetzt werden.



Die Einstellung der Eigenschaften eines Bezeichnungsfeldes ist zwar auch zur Laufzeit des Programms möglich, es empfiehlt sich aber es nur dann im Programm zu tun, wenn Inhalte des Feldes dynamisch zur Laufzeit verändert werden sollen. Ansonsten empfiehlt es sich, die Eigenschaften im Dialogfenster festzulegen.

4.3.4 Schaltflächen, Wechselschaltflächen (CommandButton, ToggleButton)

Die Schaltflächen – Definition ist recht einfach. Sowohl die normalen Schaltflächen als auch die Umschaltbuttons (Wechselschaltflächen) besitzen gleiche Definitionseigenschaften. Der Unterschied besteht darin, dass die ToggleButtons nach dem Aktivieren (anklicken oder Enter – Taste betätigen) solange im gedrückten Zustand verbleiben, bis sie wieder angeklickt werden.



Abb. 4.3.4 Unterschiedliche Formen des Schaltflächen

Wichtigste Eigenschaften:

- ✓ Der darzustellende Text wird über die Eigenschaft CAPTION definiert.

- ✓ Mit PICTURE kann eine Bitmap – Grafik statt Beschriftung eingefügt werden.
- ✓ Im Falle einer eingefügten Grafik kann mit CONTROLTIPTXT ein gelber Infotext definiert werden.
- ✓ Mit PICTUREPOSITION wird die Position der Grafik festgelegt.
- ✓ Soll die Schaltflächengröße an den Inhalt angepasst werden, geschieht es über AUTOSIZE.
- ✓ Der aktuelle Zustand kann über VALUE abgefragt werden.

4.3.5 Textfelder (TextBox)

Textfelder ermöglichen die Eingabe von Texten. Die Eigenschaften dieser Felder sind größtenteils mit den der Bezeichnungsfelder (Label) identisch, weswegen an dieser Stelle auf die nochmalige Beschreibung identischer Eigenschaften verzichtet wird.



Abb. 4.3.5 Unterschiedliche Formen des Textfeldes

Wichtigste zusätzliche Eigenschaften:

- ✓ Mehrzeilige Textfelder werden mit MULTILINE definiert.
- ✓ Bei mehrzeiligen Textfeldern können mit SCROLLBARS Laufleisten eingeblendet werden, wenn der Text über die definierte Breite hinausgeht.
- ✓ ENTERKEYBEHAVIOR = TRUE bewirkt einen Zeilenumbruch bei Betätigung der ENTER – Taste (MULTILINE muss auf TRUE gesetzt sein).
- ✓ Mit ENTERFIELDBEHAVIOR = 0 wird beim Aktivieren des Textfeldes der gesamte Inhalt markiert, was die Neueingaben, insbesondere bei einzeiligen Feldern praktischer macht.
- ✓ WORDWRAP bewirkt einen Zeilenumbruch, wenn der Text den rechten Rand erreicht (erfordert MULTILINE = TRUE).
- ✓ Die Textausrichtung wird mit TEXTALIGN definiert.
- ✓ Der Zugriff auf den Inhalt erfolgt über TEXT.
- ✓ Die Anzahl Zeilen kann mit LINECOUNT, die aktuelle Zeile mit CURLINE die Anzahl Zeichen durch LEN(FELDNAME.TEXT) ermittelt werden.
- ✓ Soll das Textfeld als Password-Feld (Password eingaben) dienen, können mit PASSWORDCHAR Zeichen definiert werden, die statt des eingegebenen Textes erscheinen.
- ✓ Die Auswahl SELECTMARGIN = TRUE erzeugt einen Leerraum am linken Rand, was eine bequeme Markierung von Zeilen in mehrzeiligen Textfeldern möglich macht.



Auf den markierten Text kann über SELTEXT zugegriffen werden. Die Eigenschaften SELSTART und SELLENGTH geben die Position des ersten markierten Zeichens und die Länge der Markierung zurück. Damit kann per Programm Markiert oder eine Textmarkierung manipuliert werden:

```
With Textfeld
    .SelLength = 0      'Löschen der Markierung
    .SelStart = 0      '
    .SelLength = 8     'die ersten 8 Zeichen markieren
    .SelText = ""      'markierter Text wird gelöscht
```



```
.SelStart = 15      'Cursor auf neue Position
.SelText = "ein"   'Zeichenkette an neuer Position einfügen
End With
```



Die Methoden CUT bzw. COPY übertragen den markierten Text in die Zwischenablage (Ausschneiden, Kopieren), die Methode PASTE überträgt den Inhalt der Zwischenablage in den markierten Bereich des Textes.

4.3.6 Listen, Kombinationsfelder (ListBox, ComboBox)

Listenfelder erlauben Auswahlen von Alternativen ohne direkte Eingabe, Kombinationsfelder (Abb. 4.3.6) bieten neben der gleichen Technik zusätzlich noch als Kombination von Listen- und Textfeldeigenschaften Eingabemöglichkeiten an.

Abb. 4.3.6 Einfaches Kombinationsfeld

Wichtigste Eigenschaften:

- Die Einträge in der Liste werden mit der Methode `AddItem` hinzugefügt. (in dem obigen Beispiel 5).
- Der Textinhalt des Kombinationsfeldes wird durch die Anweisung `If Me.cboPerson.Text = "" Then` abgefragt.
- Die Breite der geöffneten Liste wird über `LISTWIDTH` definiert (Angaben in Point; 1 Point= 1/72 Zoll). Die Breite des geschlossenen Feldes bleibt davon unberührt.
- Die Anzahl in der geöffneten Liste angezeigten Zeilen aus dem `LISTFILLRANGE` – Bereich wird über `LISTROWS` festgelegt. Enthält dieser Bereich mehr Zeilen, werden Laufleisten eingeblendet.
- `ROWSOURCE` Stellt die Verbindung zu einem Bereich her. Gibt die Quelle an, die eine Liste für ein Kombinationsfeld-Steuer-element (COMBOBOX) oder Listenfeld-Steuer-element (LISTBOX) zur Verfügung stellt.
- `SHOWDROPBUTTONWHEN` bestimmt, wann der Dropdown - Schalter eingeblendet werden soll (NEVER, ALWAYS, FOCUS).
- `STYLE` definiert, ob im Feld Eingaben gestattet werden sollen oder ob das Feld nur als unveränderbare Liste zur Verfügung stehen soll.

4.3.7 Drehfelder, Laufleisten (SpinButton, ScrollBar)

Bildlaufleisten und Drehfelder (Abb. 4.3.7) dienen dazu eine ganze Zahl aus einem vordefinierten Wertebereich zu selektieren. Der zulässige Zahlenbereich liegt im `LONG` – Zahlenraum (etwa + / - 2.109).

Abb. 4.3.7 Drehfeld

Bei Laufleisten kann der Wert durch das Anklicken einer der Inkrement- oder Dekrement Schaltflächen, das Verschieben eines Schiebers oder einen Klick in die Lauffläche verändert werden. Das Drehfeld ist eine „abgemagerte“ Variante der Laufleiste – es enthält nur eine Inkrement und eine Dekrement – Schaltfläche.

Wichtigste Eigenschaften:

- DELAY bestimmt die Verzögerung zwischen Klick und Ergebnis in Milisekunden.
- Die Grenzen des zulässigen Wertebereichs werden mit MIN / MAX festgelegt werden.
- ORIENTATION bestimmt die Ausrichtung des Steuerelements (vertikal / horizontal).
- Die Abmessungen des Schiebers einer Laufleiste werden mit PROPORTIONALTHUMP festgelegt. Steht der Wert auf TRUE, so ist die Schiebergröße umgekehrt proportional zur Größe des Wertebereichs. Bei großen Wertebereichen auf FALSE setzen, sonst ist der Schieber kaum sichtbar und bedienbar!
- SMALLCHANGE legt die Schrittweite der Wertänderung beim Klick auf die Inkrement bzw. Dekrement – Schaltflächen fest.
- Erzeugte Werte können über VALUE abgefragt werden.

4.3.8 Kontrollkästchen, Optionsfelder (CheckBox, OptionButton)

Kontrollkästchen (Abb. 4.3.8) eignen sich in für Ja / Nein – Entscheidungen. Der aktuelle Zustand wird durch ein Häkchen ✓ im quadratischen Fenster des Kontrollkästchens angezeigt.

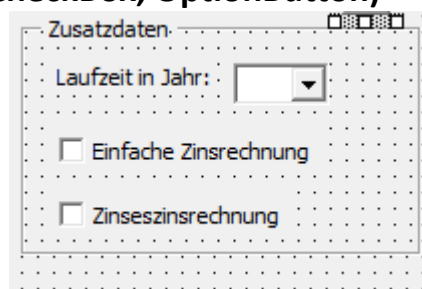


Abb. 4.3.8 Kontrollfelder

Wichtigste Eigenschaften:

- Der aktuelle Zustand kann über die Eigenschaft VALUE erfragt werden. Zulässige Werte sind FALSE, TRUE und NULL.
- Die Beschriftung kann über CAPTION festgelegt werden.
- Über die Eigenschaft GROUPNAME können Optionsfelder und Kontrollfelder gruppiert werden. Sollen mehrere Optionsfelder zu einer Gruppe gehören, wird bei allen unter GROUPNAME der gleiche Gruppenname angegeben.

Aufgabe A1: Automatisierung und Programmierung mit Word 2013 Professional

Allgemeines zu dem Hintergrund der Aufzinsung

Der Zins, als Kompensation für den zwischenzeitlichen Konsumverzicht des Kapitalgebers, stellt eine von vier Kategorien der Zinsrechnung dar. Er ergibt sich in der ersten Zinsperiode aus der Multiplikation des vereinbarten Zinssatzes i mit dem Anfangskapital K_0 . Weiterhin steht das Endkapital K_n im Interesse des Gläubigers bzw. Schuldners, welches wiederum von der Laufzeit n des Finanzkontraktes abhängig ist. Der Zinssatz i wird hierbei auch als Nominalzins bezeichnet. Daraus lassen sich die Formeln für das Endkapital bei einfacher Verzinsung und bei Zinseszinsrechnung ermitteln.

Vorgehensweise

A1.1 Erstellen Sie ein Worddokument mit dem Namen `Axxxxxxx.Docx` (xxxxxxx steht für Ihre Matrikelnummer). Zeichnen Sie in dieses Worddokument zunächst ein Macro mit dem Namen `mAufzinsung` und erstellen Sie ein Layout (Tab A1.1) mit folgenden Merkmalen:

- Eine Tabelle für **die Aufzinsung der Kapitalanlage** mit 18 Zeilen und 2 Spalten. Dabei sollen die Zeilenhöhen 0,6 cm, die beiden Spaltenbreiten jeweils 5 cm sein.
- Bei der ersten Zeile müssen die Zellen verbunden und mit der Hintergrundfarbe Dunkelblau, Text 2, dunkler, 25% gefärbt sein.
- Bei den Zeilen 2-6 sind die inneren Rahmen ausgeblendet.
- Die Beschriftung „Aufzinsung für die Kapitalanlage“ hat eine Schriftart Calibri, einen Schriftschnitt fett, einen Schriftgrad 18 und eine Schriftfarbe Weiß, Hintergrund 1.
- Alle anderen Beschriftungen haben eine Schriftart Calibri, einen Schriftschnitt fett, einen Schriftgrad 14 und eine Schriftfarbe Schwarz, Text 1, heller, 5%.
- Die Textausrichtung in jeder Zelle soll wie vorgegeben (Tab A1.1) ausgerichtet sein.
- Alle rot dargestellten Werte sind die Positionen der Textmarken, die über die VBA-Codes eingetragen werden. Bezeichnen Sie diese, wie Sie für sinnvoll halten.
- Beenden Sie das Macro und testen Sie, ob das einwandfrei funktioniert.



Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Aufzinsung für die Kapitalanlagen	
Investmentperson:	
Kapitalanlage:	
Anfangskapital:	
Prozentsatz:	
Laufzeit im Jahr:	
Jahr	Kapitalwert
0	
1	
2	
3	
4	
5	
6	
7	
8	
9	
10	

Bei der Aufgabe wurde die maximale Laufzeit nur für 10 Jahre berücksichtigt. Zinsart ist Zinseszinsrechnung

Tab A1.1 Das Layout für das Macro mAufzinsung

A1.2 Erstellen Sie ein Userformblatt (Abb. A1.1) mit dem Namen `frmAufzinsungMitWord` und benennen Sie Text-, Kombinations-, Optionsfelder und die Schaltflächen wie Sie es für sinnvoll halten.

Erläuterung zu den Steuerelementfunktionen

Das Userformblatt besteht aus den Zwei Kombinationsfeldern, 5 Textfeldern, 2 Optionsfeldern, 7 Beschriftungsfeldern, 2 Rahmen und 3 Befehlsschaltflächen (Felder leeren mit dem `cmdloeschen`, Datensatz Ablegen mit dem Namen `cmdAblegen` und Drucken mit dem Namen `cmdDrucken`).

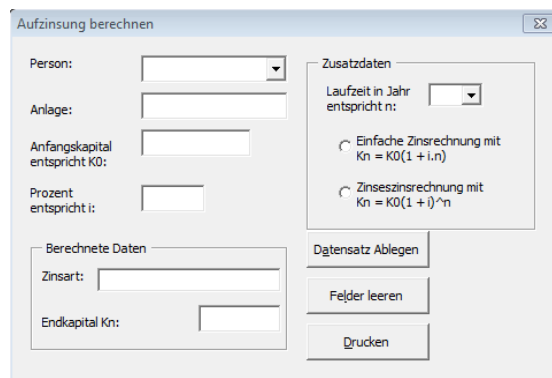


Abb. A1.1 Die Userformblatt `frmAufzinsungMitWord`

Eingesetzte Steuerelemente

Pos.	Bezeichnung (Caption)	Steuerelement	Name
1	Person:	Beschriftungsfeld	Nicht ändern
2	keine	Kombinationsfeld	cboPerson
3	Kapitalanlage:	Beschriftungsfeld	Nicht ändern
4	keine	Textfeld	txtAnlage
5	Anfangskapital:	Beschriftungsfeld	Nicht ändern
6	keine	Textfeld	txtKapital
7	Prozent:	Beschriftungsfeld	Nicht ändern
8	keine	Textfeld	txtProzent
9	Zusatzdaten	Rahmen	Nicht ändern
10	Laufzeit im Jahr:	Beschriftungsfeld	Nicht ändern
11	keine	Kombinationsfeld	cboLaufzeit
12	Einfache Zinsrechnung:	Optionsfeld	optEZins
13	Zinsenzinsrechnung:	Optionsfeld	optZZins
14	Berechnete Daten	Rahmen	Nicht ändern
15	Zinsart:	Beschriftungsfeld	Nicht ändern
16	keine	Textfeld	txtZArt
17	Endkapital Kn:	Beschriftungsfeld	Nicht ändern
18	keine	Textfeld	txtKn
19	Datensatz Ablegen	Befehlsschaltfläche	cmdAblegen
20	Drucken	Befehlsschaltfläche	cmddrucken
21	Felder leeren	Befehlsschaltfläche	cmdMaskeLoeschen

Erläuterung zu dem Programmablauf

A1.2.1 Beim Starten des Userformblattes sollen alle Inhalte der Textfelder leer und der Zustand der Optionsfelder nicht gewählt sein.

A1.2.2 Wenn der Benutzer keine Eingaben in den Text-, Kombinationsfeldern und keine Auswahl bei einem Optionsfeld vornimmt und auf die Schaltflächen `Datensatz Ablegen` oder `Drucken` klickt, soll eine Meldung ausgegeben werden, dass alle Felder ausgefüllt bzw. ausgewählt werden sollten.


A1.2.3 Wenn der Benutzer die Felder `Person`, `Anlage`, `Anfangskapital`, `Prozent` und `Laufzeit im Jahr` auswählt bzw. eingibt und darauf eine Zinsart (einfache Zinsrechnung oder Zinsenzinsrechnung) auswählt, müssen die Werte der Zinsart und des Endkapitales in den entsprechenden Textfeldern in dem Rahmenbereich „Berechnete Daten“ erscheinen.

A1.2.4 Wenn der Benutzer auf die Schaltfläche `Datensatz Ablegen` klickt, müssen die Eingabedaten mit berechneten Werten für den Kapitalwert (maximal 11 Werten: Laufzeit im Jahr muss maximal für 10 Jahre berücksichtigt werden) in das Layout ausgegeben bzw. übertragen werden. Dabei soll das Programm aus Sicht der Korrektheit vorherige Inhalte aus dem Layout (ThisDocument) löschen.

A1.2.5 Die Schaltfläche `Felder leeren` soll die Textfelder leeren und Optionsfelder deaktivieren.

A1.2.6 Die Schaltfläche **Drucken** soll bewirken, dass das Layout (ThisDocument) an den Standard eingestellten Drucker geschickt wird.

A1.3 Wie schon bei den Punkten A1.2.1 bis A1.2.4 bei der Erläuterung zu dem Programmablauf beschrieben wurde, muss das Programm als Ergebnis **Abb. A1.4** für den Fall Datensatz **Ablegen** liefern.


 Hochschule für Angewandte Wissenschaften Hamburg
 Hamburg University of Applied Sciences

Aufzinsung für die Kapitalanlagen	
Investmentperson:	Frau Raza
Kapitalanlage:	Immobilien
Anfangskapital:	190000
Prozentsatz:	0,04
Laufzeit im Jahr:	8
Jahr	Kapitalwert
0	190000
1	197600
2	205504
3	213724,16
4	222273,13
5	231164,05
6	240410,61
7	250027,04
8	260028,12
9	
10	

Bei der Aufgabe wurde die maximale Laufzeit nur für 10 Jahre berücksichtigt. Zinsart ist

Aufzinsung berechnen ✕

Person: <input type="text" value="Frau Raza"/>	Zusatzdaten
Anlage: <input type="text" value="Immobilien"/>	Laufzeit in Jahr entspricht n: <input type="text" value="8"/>
Anfangskapital entspricht K0: <input type="text" value="190000"/>	<input type="radio"/> Einfache Zinsrechnung mit $K_n = K_0(1 + i \cdot n)$
Prozent entspricht i: <input type="text" value="0,04"/>	<input checked="" type="radio"/> Zinseszinsrechnung mit $K_n = K_0(1 + i)^n$
Berechnete Daten	<input type="button" value="Datensatz Ablegen"/>
Zinsart: <input type="text" value="Zinseszinsrechnung"/>	<input type="button" value="Felder leeren"/>
Endkapital K _n : <input type="text" value="260028,12"/>	<input type="button" value="Drucken"/>

Abb. A1.4 Das Layout beim Datensatz Ablegen

LITERATUR

- Word-Formulare (www.rrzn.uni-hannover.de/buecher.html)
- Einführung in Word von Manuela Jürgens (VBA mit Word von Uni-Hagen.pdf)
- Investition und Finanzierung von Ulrich Ermschel et al., ISBN-978-3-7908-2745-3, Springer-Verlag Berlin Heidelberg 2011(<https://kataloge.uni-hamburg.de/DB=2/SET=5/TTL=1/SHW?FRST=3>)

VBA-Codes für die Aufzinsung

```

Sub mAufzinsung(DasBild As String)
    Selection.WholeStory
    Selection.Cut
    If DasBild > "" Then
        Selection.InlineShapes.AddPicture FileName:=DasBild, LinkToFile:=False, _
            SaveWithDocument:=True

    Else
        MsgBox "Das Bild existiert nicht. legen Sie das Haw-logo in den Ordner Bilder!"
        Exit Sub
    End If
    Selection.ParagraphFormat.Alignment = wdAlignParagraphRight
    Selection.TypeParagraph
    Selection.ParagraphFormat.Alignment = wdAlignParagraphLeft
    Call mAufzinsung
End Sub
Rem Aufgezeichnetes Macro
Sub mAufzinsung()

    ActiveDocument.Tables.Add Range:=Selection.Range, NumRows:=18, NumColumns _
        :=2, DefaultTableBehavior:=wdWord9TableBehavior, AutoFitBehavior:= _
        wdAutoFitFixed
    With Selection.Tables(1)
        If .Style <> "Tabellenraster" Then
            .Style = "Tabellenraster"
        End If
        .ApplyStyleHeadingRows = True
        .ApplyStyleLastRow = False
        .ApplyStyleFirstColumn = True
        .ApplyStyleLastColumn = False
        .ApplyStyleRowBands = True
        .ApplyStyleColumnBands = False
    End With
    Selection.Tables(1).Columns(1).SetWidth ColumnWidth:=155, RulerStyle:= _
        wdAdjustNone
    Selection.Tables(1).Columns(2).SetWidth ColumnWidth:=140, RulerStyle:= _
        wdAdjustNone
    Selection.MoveRight Unit:=wdCharacter, Count:=2, Extend:=wdExtend
    Selection.Cells.Merge
    With Selection.Cells
        With .Shading
            .Texture = wdTextureNone
            .ForegroundColor = wdColorAutomatic
            .BackgroundPatternColor = -553582695
        End With
        With .Borders(wdBorderLeft)
            .LineStyle = wdLineStyleSingle
            .LineWidth = wdLineWidth150pt
            .Color = wdColorAutomatic
        End With
        With .Borders(wdBorderRight)
            .LineStyle = wdLineStyleSingle
            .LineWidth = wdLineWidth150pt
            .Color = wdColorAutomatic
        End With
        With .Borders(wdBorderTop)
            .LineStyle = wdLineStyleSingle
            .LineWidth = wdLineWidth150pt
            .Color = wdColorAutomatic
        End With
        With .Borders(wdBorderBottom)
            .LineStyle = wdLineStyleSingle
            .LineWidth = wdLineWidth150pt
            .Color = wdColorAutomatic
        End With
        .Borders(wdBorderDiagonalDown).LineStyle = wdLineStyleNone
        .Borders(wdBorderDiagonalUp).LineStyle = wdLineStyleNone
        .Borders.Shadow = False
    End With
    With Options
        .DefaultBorderLineStyle = wdLineStyleSingle
        .DefaultBorderLineWidth = wdLineWidth150pt
        .DefaultBorderColor = wdColorAutomatic
    End With

```

```
Selection.Font.Color = -603914241
Selection.Font.Size = 18
Selection.Font.Name = "Calibri"
Selection.Font.Bold = wdToggle
Selection.MoveDown Unit:=wdLine, Count:=1
Selection.MoveRight Unit:=wdCharacter, Count:=2, Extend:=wdExtend
Selection.MoveDown Unit:=wdLine, Count:=4, Extend:=wdExtend
With Selection.Cells
  With .Shading
    .Texture = wdTextureNone
    .ForegroundColor = wdColorAutomatic
    .BackgroundColor = -738132173
  End With
  With .Borders(wdBorderLeft)
    .LineStyle = wdLineStyleSingle
    .LineWidth = wdLineWidth150pt
    .Color = wdColorAutomatic
  End With
  With .Borders(wdBorderRight)
    .LineStyle = wdLineStyleSingle
    .LineWidth = wdLineWidth150pt
    .Color = wdColorAutomatic
  End With
  With .Borders(wdBorderTop)
    .LineStyle = wdLineStyleSingle
    .LineWidth = wdLineWidth150pt
    .Color = wdColorAutomatic
  End With
  With .Borders(wdBorderBottom)
    .LineStyle = wdLineStyleSingle
    .LineWidth = wdLineWidth150pt
    .Color = wdColorAutomatic
  End With
  .Borders(wdBorderHorizontal).LineStyle = wdLineStyleNone
  .Borders(wdBorderVertical).LineStyle = wdLineStyleNone
  .Borders(wdBorderDiagonalDown).LineStyle = wdLineStyleNone
  .Borders(wdBorderDiagonalUp).LineStyle = wdLineStyleNone
  .Borders.Shadow = False
End With
With Options
  .DefaultBorderLineStyle = wdLineStyleSingle
  .DefaultBorderLineWidth = wdLineWidth150pt
  .DefaultBorderColor = wdColorAutomatic
End With
Selection.Font.Name = "Calibri"
Selection.Font.Bold = wdToggle
Selection.Font.Size = 14
Selection.Font.Color = -587137025
Selection.MoveRight Unit:=wdCharacter, Count:=2
Selection.MoveUp Unit:=wdLine, Count:=1
Selection.MoveDown Unit:=wdLine, Count:=4, Extend:=wdExtend
Selection.Font.Italic = wdToggle
Selection.Font.Color = -654278401
Selection.MoveDown Unit:=wdLine, Count:=1
Selection.MoveLeft Unit:=wdCharacter, Count:=1
Selection.MoveRight Unit:=wdCharacter, Count:=2, Extend:=wdExtend
With Selection.Cells
  With .Shading
    .Texture = wdTextureNone
    .ForegroundColor = wdColorAutomatic
    .BackgroundColor = -738132122
  End With
  With .Borders(wdBorderLeft)
    .LineStyle = wdLineStyleSingle
    .LineWidth = wdLineWidth150pt
    .Color = wdColorAutomatic
  End With
  With .Borders(wdBorderRight)
    .LineStyle = wdLineStyleSingle
    .LineWidth = wdLineWidth150pt
    .Color = wdColorAutomatic
  End With
  With .Borders(wdBorderTop)
    .LineStyle = wdLineStyleSingle
    .LineWidth = wdLineWidth150pt
```



```

        .Color = wdColorAutomatic
    End With
    With .Borders(wdBorderBottom)
        .LineStyle = wdLineStyleSingle
        .LineWidth = wdLineWidth150pt
        .Color = wdColorAutomatic
    End With
    With .Borders(wdBorderVertical)
        .LineStyle = wdLineStyleSingle
        .LineWidth = wdLineWidth150pt
        .Color = wdColorAutomatic
    End With
    .Borders(wdBorderDiagonalDown).LineStyle = wdLineStyleNone
    .Borders(wdBorderDiagonalUp).LineStyle = wdLineStyleNone
    .Borders.Shadow = False
End With
With Options
    .DefaultBorderLineStyle = wdLineStyleSingle
    .DefaultBorderLineWidth = wdLineWidth150pt
    .DefaultBorderColor = wdColorAutomatic
End With
Selection.Font.Name = "Calibri"
Selection.Font.Size = 16
Selection.Font.Bold = wdToggle
Selection.Font.Color = -587137025
Selection.MoveDown Unit:=wdLine, Count:=1
Selection.MoveDown Unit:=wdLine, Count:=10, Extend:=wdExtend
Selection.Font.Name = "Calibri"
Selection.Font.Size = 14
Selection.Font.Bold = wdToggle
Selection.Font.Color = -587137025
With Selection.Cells
    With .Shading
        .Texture = wdTextureNone
        .ForegroundColor = wdColorAutomatic
        .BackgroundColor = -738132173
    End With
    With .Borders(wdBorderLeft)
        .LineStyle = wdLineStyleSingle
        .LineWidth = wdLineWidth150pt
        .Color = wdColorAutomatic
    End With
    With .Borders(wdBorderRight)
        .LineStyle = wdLineStyleSingle
        .LineWidth = wdLineWidth050pt
        .Color = wdColorAutomatic
    End With
    With .Borders(wdBorderTop)
        .LineStyle = wdLineStyleSingle
        .LineWidth = wdLineWidth150pt
        .Color = wdColorAutomatic
    End With
    With .Borders(wdBorderBottom)
        .LineStyle = wdLineStyleSingle
        .LineWidth = wdLineWidth150pt
        .Color = wdColorAutomatic
    End With
    With .Borders(wdBorderHorizontal)
        .LineStyle = wdLineStyleSingle
        .LineWidth = wdLineWidth050pt
        .Color = wdColorAutomatic
    End With
    .Borders(wdBorderDiagonalDown).LineStyle = wdLineStyleNone
    .Borders(wdBorderDiagonalUp).LineStyle = wdLineStyleNone
    .Borders.Shadow = False
End With
With Options
    .DefaultBorderLineStyle = wdLineStyleSingle
    .DefaultBorderLineWidth = wdLineWidth050pt
    .DefaultBorderColor = wdColorAutomatic
End With
Selection.MoveRight Unit:=wdCharacter, Count:=1
Selection.MoveDown Unit:=wdLine, Count:=10, Extend:=wdExtend
With Selection.Cells
    With .Shading

```

```

        .Texture = wdTextureNone
        .ForegroundColor = wdColorAutomatic
        .BackgroundPatternColor = -738132173
    End With
    With .Borders(wdBorderLeft)
        .LineStyle = wdLineStyleSingle
        .LineWidth = wdLineWidth050pt
        .Color = wdColorAutomatic
    End With
    With .Borders(wdBorderRight)
        .LineStyle = wdLineStyleSingle
        .LineWidth = wdLineWidth150pt
        .Color = wdColorAutomatic
    End With
    With .Borders(wdBorderTop)
        .LineStyle = wdLineStyleSingle
        .LineWidth = wdLineWidth150pt
        .Color = wdColorAutomatic
    End With
    With .Borders(wdBorderBottom)
        .LineStyle = wdLineStyleSingle
        .LineWidth = wdLineWidth150pt
        .Color = wdColorAutomatic
    End With
    With .Borders(wdBorderHorizontal)
        .LineStyle = wdLineStyleSingle
        .LineWidth = wdLineWidth050pt
        .Color = wdColorAutomatic
    End With
    .Borders(wdBorderDiagonalDown).LineStyle = wdLineStyleNone
    .Borders(wdBorderDiagonalUp).LineStyle = wdLineStyleNone
    .Borders.Shadow = False
End With
With Options
    .DefaultBorderLineStyle = wdLineStyleSingle
    .DefaultBorderLineWidth = wdLineWidth050pt
    .DefaultBorderColor = wdColorAutomatic
End With
Selection.Font.Name = "Calibri"
Selection.Font.Size = 14
Selection.Font.Bold = wdToggle
Selection.Font.Italic = wdToggle
Selection.Font.Color = -654278401
Selection.MoveUp Unit:=wdLine, Count:=7
Selection.TypeText Text:="Aufzinsung für die Kapitalanlagen"
Selection.MoveDown Unit:=wdLine, Count:=1
Selection.MoveLeft Unit:=wdCharacter, Count:=1
Selection.TypeText Text:="Investmentperson:"
Selection.MoveDown Unit:=wdLine, Count:=1
Selection.TypeText Text:="Kapitalanlage:"
Selection.MoveDown Unit:=wdLine, Count:=1
Selection.TypeText Text:="Anfangskapital:"
Selection.MoveDown Unit:=wdLine, Count:=1
Selection.TypeText Text:="Prozentsatz:"
Selection.MoveDown Unit:=wdLine, Count:=1
Selection.TypeText Text:="Laufzeit im Jahr:"
Selection.MoveUp Unit:=wdLine, Count:=4
Selection.MoveRight Unit:=wdCharacter, Count:=4
Selection.ParagraphFormat.Alignment = wdAlignParagraphRight
With ActiveDocument.Bookmarks
    .Add Range:=Selection.Range, Name:="tmPerson"
    .DefaultSorting = wdSortByName
    .ShowHidden = False
End With
Selection.MoveDown Unit:=wdLine, Count:=1
Selection.ParagraphFormat.Alignment = wdAlignParagraphRight
With ActiveDocument.Bookmarks
    .Add Range:=Selection.Range, Name:="tmAnlage"
    .DefaultSorting = wdSortByName
    .ShowHidden = False
End With
Selection.MoveDown Unit:=wdLine, Count:=1
Selection.ParagraphFormat.Alignment = wdAlignParagraphRight
With ActiveDocument.Bookmarks
    .Add Range:=Selection.Range, Name:="tmKapital"

```

```

        .DefaultSorting = wdSortByName
        .ShowHidden = False
    End With
    Selection.MoveDown Unit:=wdLine, Count:=1
    Selection.ParagraphFormat.Alignment = wdAlignParagraphRight
    With ActiveDocument.Bookmarks
        .Add Range:=Selection.Range, Name:="tmProzent"
        .DefaultSorting = wdSortByName
        .ShowHidden = False
    End With
    Selection.MoveDown Unit:=wdLine, Count:=1
    Selection.ParagraphFormat.Alignment = wdAlignParagraphRight
    With ActiveDocument.Bookmarks
        .Add Range:=Selection.Range, Name:="tmLaufzeit"
        .DefaultSorting = wdSortByName
        .ShowHidden = False
    End With
    Selection.MoveDown Unit:=wdLine, Count:=1
    Selection.MoveLeft Unit:=wdCharacter, Count:=1
    Selection.TypeText Text:="Jahr"
    Selection.ParagraphFormat.Alignment = wdAlignParagraphRight
    Selection.MoveRight Unit:=wdCharacter, Count:=1
    Selection.TypeText Text:="Kapitalwert"
    Selection.ParagraphFormat.Alignment = wdAlignParagraphRight
    Selection.MoveDown Unit:=wdLine, Count:=1
    Selection.ParagraphFormat.Alignment = wdAlignParagraphRight
    With ActiveDocument.Bookmarks
        .Add Range:=Selection.Range, Name:="tmK0"
        .DefaultSorting = wdSortByName
        .ShowHidden = False
    End With
    Selection.MoveDown Unit:=wdLine, Count:=1
    Selection.ParagraphFormat.Alignment = wdAlignParagraphRight
    With ActiveDocument.Bookmarks
        .Add Range:=Selection.Range, Name:="tmK1"
        .DefaultSorting = wdSortByName
        .ShowHidden = False
    End With
    Selection.MoveDown Unit:=wdLine, Count:=1
    Selection.ParagraphFormat.Alignment = wdAlignParagraphRight
    With ActiveDocument.Bookmarks
        .Add Range:=Selection.Range, Name:="tmK2"
        .DefaultSorting = wdSortByName
        .ShowHidden = False
    End With
    Selection.MoveDown Unit:=wdLine, Count:=1
    Selection.ParagraphFormat.Alignment = wdAlignParagraphRight
    With ActiveDocument.Bookmarks
        .Add Range:=Selection.Range, Name:="tmK3"
        .DefaultSorting = wdSortByName
        .ShowHidden = False
    End With
    Selection.MoveDown Unit:=wdLine, Count:=1
    Selection.ParagraphFormat.Alignment = wdAlignParagraphRight
    With ActiveDocument.Bookmarks
        .Add Range:=Selection.Range, Name:="tmK4"
        .DefaultSorting = wdSortByName
        .ShowHidden = False
    End With
    Selection.MoveDown Unit:=wdLine, Count:=1
    Selection.ParagraphFormat.Alignment = wdAlignParagraphRight
    With ActiveDocument.Bookmarks
        .Add Range:=Selection.Range, Name:="tmK5"
        .DefaultSorting = wdSortByName
        .ShowHidden = False
    End With
    Selection.MoveDown Unit:=wdLine, Count:=1
    Selection.ParagraphFormat.Alignment = wdAlignParagraphRight
    With ActiveDocument.Bookmarks
        .Add Range:=Selection.Range, Name:="tmK6"
        .DefaultSorting = wdSortByName
        .ShowHidden = False
    End With
    Selection.MoveDown Unit:=wdLine, Count:=1
    Selection.ParagraphFormat.Alignment = wdAlignParagraphRight

```

```

With ActiveDocument.Bookmarks
    .Add Range:=Selection.Range, Name:="tmK7"
    .DefaultSorting = wdSortByName
    .ShowHidden = False
End With
Selection.MoveDown Unit:=wdLine, Count:=1
Selection.ParagraphFormat.Alignment = wdAlignParagraphRight
With ActiveDocument.Bookmarks
    .Add Range:=Selection.Range, Name:="tmK8"
    .DefaultSorting = wdSortByName
    .ShowHidden = False
End With
Selection.MoveDown Unit:=wdLine, Count:=1
Selection.ParagraphFormat.Alignment = wdAlignParagraphRight
With ActiveDocument.Bookmarks
    .Add Range:=Selection.Range, Name:="tmK9"
    .DefaultSorting = wdSortByName
    .ShowHidden = False
End With
Selection.MoveDown Unit:=wdLine, Count:=1
Selection.ParagraphFormat.Alignment = wdAlignParagraphRight
With ActiveDocument.Bookmarks
    .Add Range:=Selection.Range, Name:="tmK10"
    .DefaultSorting = wdSortByName
    .ShowHidden = False
End With
Selection.MoveLeft Unit:=wdCharacter, Count:=1
Selection.MoveUp Unit:=wdLine, Count:=10
Selection.MoveDown Unit:=wdLine, Count:=10, Extend:=wdExtend
Selection.ParagraphFormat.Alignment = wdAlignParagraphRight
Selection.MoveUp Unit:=wdLine, Count:=1
Selection.MoveDown Unit:=wdLine, Count:=1
Selection.TypeText Text:="0"
Selection.MoveDown Unit:=wdLine, Count:=1
Selection.TypeText Text:="1"
Selection.MoveDown Unit:=wdLine, Count:=1
Selection.TypeText Text:="2"
Selection.MoveDown Unit:=wdLine, Count:=1
Selection.TypeText Text:="3"
Selection.MoveDown Unit:=wdLine, Count:=1
Selection.TypeText Text:="4"
Selection.MoveDown Unit:=wdLine, Count:=1
Selection.TypeText Text:="5"
Selection.MoveDown Unit:=wdLine, Count:=1
Selection.TypeText Text:="6"
Selection.MoveDown Unit:=wdLine, Count:=1
Selection.TypeText Text:="7"
Selection.MoveDown Unit:=wdLine, Count:=1
Selection.TypeText Text:="8"
Selection.MoveDown Unit:=wdLine, Count:=1
Selection.TypeText Text:="9"
Selection.MoveDown Unit:=wdLine, Count:=1
Selection.TypeText Text:="10"
Selection.MoveDown Unit:=wdLine, Count:=1
Selection.TypeParagraph
Selection.MoveDown Unit:=wdLine, Count:=1

Selection.TypeText Text:= _
    "Bei der Aufgabe wurde die maximale Laufzeit nur für 10 Jahre berücksichtigt.
Zinsart ist "
Selection.Font.Name = "Calibri"
Selection.Font.Size = 14
Selection.Font.Bold = wdToggle
Selection.Font.Italic = wdToggle
Selection.Font.Color = -654278401
With ActiveDocument.Bookmarks
    .Add Range:=Selection.Range, Name:="tmZArt"
    .DefaultSorting = wdSortByName
    .ShowHidden = False
End With
Selection.TypeParagraph
End Sub

```

Rem Globale Variablen in dem Code-Bereich von der frmAufzinsung

```

Option Explicit
Dim varMatrix(1 To 3) As Single

```

```
Dim strZielpath As String
```

Rem Wenn Userform gestartet wird, wird die Funktion Userform_Initialize() automatisch gestartet

```
Private Sub userform_Initialize()
    With Me.cboLaufzeit
        .AddItem 2
        .AddItem 4
        .AddItem 5
        .AddItem 8
        .AddItem 10
    End With
    With Me.cboPerson
        .AddItem "Frau Meyer"
        .AddItem "Herr Müller"
        .AddItem "Frau Raza"
        .AddItem "Herr Schmidt"
    End With
    Maskeloeschen
    Dim strPath As String
    Dim strDateiname As String
    strPath = ActiveDocument.FullName
    strDateiname = Dir(strPath)
    strZielpath = Left(strPath, Len(strPath) - Len(strDateiname)) + "Bilder\"
End Sub
```

Rem Wenn der Benutzer alle Daten von Userform in das Layout überträgt, wird es auf die Schaltfläche

Rem Datensatz Ablegen geklickt

```
Private Sub cmdAblegen_Click()
    If Inhalterkennen Then
        Dim i As Integer
        mAufzinsung strZielpath & "haw-logo.jpg"
        With ActiveDocument.Bookmarks("tmZart").Range
            .Text = Me.txtZart.Value
            .Font.Name = "Calibri"
            .Font.Bold = False
            .Font.Italic = True
            .Font.Size = 14
            .Font.Color = wdColorBlack
        End With
        With ActiveDocument.Bookmarks("tmPerson").Range
            .Text = cboPerson.Value
            .Font.Name = "Calibri"
            .Font.Bold = False
            .Font.Italic = True
            .Font.Size = 14
            .Font.Color = wdColorRed
        End With
        With ActiveDocument.Bookmarks("tmAnlage").Range
            .Text = Me.txtAnlage.Value
            .Font.Name = "Calibri"
            .Font.Bold = False
            .Font.Italic = True
            .Font.Size = 14
            .Font.Color = wdColorRed
        End With
        With ActiveDocument.Bookmarks("tmKapital").Range
            .Text = CCur(Me.txtKapital.Value * 1)

            .Font.Name = "Calibri"
            .Font.Bold = False
            .Font.Italic = True
            .Font.Size = 14
            .Font.Color = wdColorRed
        End With
        With ActiveDocument.Bookmarks("tmProzent").Range
            .Text = Me.txtProzent.Value
            .Font.Name = "Calibri"
            .Font.Bold = False
            .Font.Italic = True
            .Font.Size = 14
            .Font.Color = wdColorRed
        End With
        With ActiveDocument.Bookmarks("tmLaufzeit").Range
            .Text = Me.cboLaufzeit.Text
            .Font.Name = "Calibri"
            .Font.Bold = False
        End With
    End If
End Sub
```

```

        .Font.Italic = True
        .Font.Size = 14
        .Font.Color = wdColorRed
    End With
    If Me.optEZins.Value = True Then
        i = 0
        For i = 0 To Me.cboLaufzeit.Value
            With ActiveDocument.Bookmarks("tmk" & i).Range
                .Text = Round(Me.txtKapital.Text * (1 + Me.txtProzent.Text * i), 2)
                .Font.Name = "Calibri"
                .Font.Bold = False
                .Font.Italic = True
                .Font.Size = 14
                .Font.Color = wdColorRed
            End With
        Next i
    End If
    If Me.optZZins.Value = True Then
        i = 0
        For i = 0 To Me.cboLaufzeit.Value
            With ActiveDocument.Bookmarks("tmk" & i).Range
                .Text = Round(Me.txtKapital.Text * (1 + Me.txtProzent.Text) ^ i, 2)
                .Font.Name = "Calibri"
                .Font.Bold = False
                .Font.Italic = True
                .Font.Size = 14
                .Font.Color = wdColorRed
            End With
        Next i
    End If

Else
    MsgBox "Sie müssen die Werte eingeben!"
    Exit Sub
End If
End Sub

```

Rem Funktion für Erkennung einer numerischen Zahl. Als Ergebnis wird geliefert wahr oder falsch.

```

Function Inhalterkennen() As Boolean
    varMatrix(1) = Textfeldpruefen(Me.txtKapital)
    varMatrix(2) = Textfeldpruefen(Me.txtProzent)
    varMatrix(3) = Textfeldpruefen(Me.cboLaufzeit)
    Dim i As Integer
    i = 0
    For i = 1 To 3
        If varMatrix(i) Then
            Inhalterkennen = True
        Else
            Inhalterkennen = False
            Exit Function
        End If
    Next i
End Function

```

Rem die Prüfung der Textfelder

```

Function Textfeldpruefen(Textfeld As Object) As Double
    If IsNumeric(Textfeld.Value) Then
        Textfeldpruefen = Textfeld.Value
    Else
        MsgBox "Der Inhalt " & Textfeld & " ist nicht korrekt!"
        Textfeld.BackColor = RGB(255, 255, 0)
        Exit Function
    End If
End Function

```

Rem Wenn der Benutzer auf die Schaltfläche Felder leeren klickt, wird Textfeldinhalte gelöscht.

```

Private Sub cmdMaskeLoeschen_Click()
    Me.cboLaufzeit.Text = ""
    Me.cboLaufzeit.BackColor = RGB(255, 255, 255)
    Me.txtAnlage.Text = ""
    Me.txtAnlage.BackColor = RGB(255, 255, 255)
    Me.txtKapital.Text = ""
    Me.txtKapital.BackColor = RGB(255, 255, 255)
    Me.txtProzent.Text = ""
    Me.txtProzent.BackColor = RGB(255, 255, 255)

```

```

Me.txtZArt.Text = ""
Me.txtKn.Text = ""
Me.optEZins.Value = False
Me.optZZins.Value = False
End Sub

```

Rem Ereignis „Click“ auf das Kombinationsfeld für Zinsrechnung

```

Private Sub cboLaufzeit Click()
    Me.optEZins.Value = False
    Me.optZZins.Value = False
End Sub

```

Rem Ereignis „Click“ auf das Optionsfeld

```

Private Sub optEZins_Click()
    Me.optZZins.Value = False
    If Inhalterkennen Then
        Dim varKn As Currency
        varKn = Me.txtKapital.Text * (1 + Me.txtProzent.Text * Me.cboLaufzeit.Text)
        Me.txtZArt.Text = "Einfache Verzinsung"
        Me.txtKn.Text = Round(varKn, 2)
    Else
        Me.optEZins.Value = False
    End If
End Sub

```

Rem Ereignis „Click“ auf das Optionsfeld für Zinsenzinsrechnung

```

Private Sub optZZins Click()
    Me.optEZins.Value = False
    If Inhalterkennen Then
        Dim varKn As Currency
        varKn = Me.txtKapital.Text * (1 + Me.txtProzent.Text) ^ Me.cboLaufzeit.Text
        Me.txtZArt.Text = "Zinsenzinsrechnung"
        Me.txtKn.Text = Round(varKn, 2)
    Else
        Me.optZZins.Value = False
    End If
End Sub

```

Rem Wenn der Benutzer auf die Schaltfläche Drucken klickt, soll das Layout an dem Drucker geschickt Rem werden.

```

Private Sub cmdDrucken Click()
    Application.PrintOut FileName:="", Range:=wdPrintAllDocument, Item:=
    wdPrintDocumentWithMarkup, Copies:=1, Pages:="", PageType:= _
    wdPrintAllPages, Collate:=True, Background:=True, PrintToFile:=False, _
    PrintZoomColumn:=0, PrintZoomRow:=0, PrintZoomPaperWidth:=0,
    PrintZoomPaperHeight:=0
End Sub

```

Aufgabe A2: Automatisierung und Programmierung mit Word 2013 Professional

Allgemeines zu dem Hintergrund

Wir erweitern die A1-Aufgabe, um die Möglichkeit einer Datenbank zu demonstrieren. Dabei könne Sie als Word-Entwickler Daten aus Excel-Dateien lesen oder auch in Excel-Tabellen schreiben. Excel lässt sich genau wie die übrigen Office-Anwendungen per VBA steuern, sodass Sie Excel für den Zugriff auf die Daten in einer Excel-Datei nicht manuell öffnen müssen. Wie dies funktioniert und was Sie alles anstellen können, erfahren Sie in diesem Grundlagenbeitrag. Für die Automatisierung von Excel von Word aus kann es verschiedene Gründe geben. Im Wesentlichen teilen sich diese auf die folgenden beiden Gruppen auf:

- Sie möchten Daten aus einer Excel-Datei auslesen.
- Sie möchten eine Excel-Datei mit Daten füllen oder die enthaltenen Daten bearbeiten und dafür gegebenenfalls zunächst eine neue Excel-Datei erstellen.

In beiden Fällen greifen Sie auf eine Excel-Instanz und die damit geöffnete Excel-Datei zu. In diesem Beispiel befassen wir uns nur mit der Auslesung einer Excel-Datei und deren Datenverarbeitung in Word.

Vorgehensweise

A2.1.1 Erstellen Sie eine Excel-Datei mit dem Namen `axxxxxxxneu.xlsm` (`xxxxxxx` steht für Ihre Matrikelnummer) und eine Tabelle (Tab A2.2) mit dem Namen „tblZinsEin“. Speichern Sie die Datei in Ihrem Pfad `/Database/axxxxxxxneu.xlsm`.

A2.1.2 Erstellen Sie ein Worddokument mit dem Namen `Axxxxxxx.Docx` (`xxxxxxx` steht für Ihre Matrikelnummer). Speichern Sie die Datei in Ihrem Pfad `/Axxxxxxx.Docx`. Zeichnen Sie in dieses Worddokument zunächst ein Macro mit dem Namen `mAufzinsung` und erstellen Sie ein Layout (Tab A2.1) mit folgenden Merkmalen:

- Eine Tabelle für **die Aufzinsung der Kapitalanlage** mit 18 Zeilen und 2 Spalten. Dabei sollen die Zeilenhöhen 0,6 cm, die beiden Spaltenbreiten jeweils 5 cm sein.
- Bei der ersten Zeile müssen die Zellen verbunden und mit der Hintergrundfarbe Dunkelblau, Text 2, dunkler, 25% gefärbt sein.
- Bei den Zeilen 2-6 sind die inneren Rahmen ausgeblendet.
- Die Beschriftung „Aufzinsung für die Kapitalanlage“ hat eine Schriftart Calibri, einen Schriftschnitt fett, einen Schriftgrad 18 und eine Schriftfarbe Weiß, Hintergrund 1.
- Alle anderen Beschriftungen haben eine Schriftart Calibri, einen Schriftschnitt fett, einen Schriftgrad 14 und eine Schriftfarbe Schwarz, Text 1, heller, 5%.
- Die Textausrichtung in jeder Zelle soll wie vorgegeben (Tab A1.1) ausgerichtet sein.
- Alle rot dargestellten Werte sind die Positionen der Textmarken, die über die VBA-Codes eingetragen werden. Bezeichnen Sie diese, wie Sie für sinnvoll halten.
- Beenden Sie das Macro und testen Sie, ob das einwandfrei funktioniert.

Z9S7

	1	2	3	4	5	6	7
1	Nr.	Anrede	Vorname	Nachname	Anlage	Kapital	Prozent
2	1	Herr Dr.	Marc	Wershoven	Immobilien	50.000,00 €	3,00%
3	2	Herr	Max	Mustermann	Anleihen	200.000,00 €	2,00%
4	3	Frau	Betina	Bertelt	Aktienfonds	240.000,00 €	5,00%
5	4	Frau Dr.	Petra	Plural	Aktien	212.000,00 €	8,00%
6	5	Frau	Elke	Raza	Zertifikat	450.000,00 €	4,00%
7							
8							

tblZinsEin Tabelle1

BEREIT

Tab A2.2 Tabelle (tblZinsEin) in Excel



Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Aufzinsung für die Kapitalanlagen	
Investmentperson:	
Kapitalanlage:	
Anfangskapital:	
Prozentsatz:	
Laufzeit im Jahr:	
Jahr	Kapitalwert
0	
1	
2	
3	
4	
5	
6	
7	
8	
9	
10	

Bei der Aufgabe wurde die maximale Laufzeit nur für 10 Jahre berücksichtigt. Zinsart ist Zinsenzinsrechnung

Tab A2.1 Das Layout für das Macro mAufzinsung

A2.2.1 Erstellen Sie ein Userformblatt (Abb. A2.1.1 und Abb. A2.1.2) mit dem Namen frmAufzinsungMitExcel und benennen Sie Listenfeld, Text-, Kombinations-, Optionsfelder und die Schaltflächen wie Sie es für sinnvoll halten.

Erläuterung zu den Steuerelementfunktionen

Das Userformblatt besteht aus zwei Seiten mit den Beschriftungen „Daten aus Exceltabelle“ und „Zinsberechnung“. In der ersten Seite sind ein Listenfeld und 7 Beschriftungsfelder und in der zweiten Seite sind aus den zwei Kombinationsfeldern, 5 Textfeldern, 2 Optionsfeldern, 7 Beschriftungsfeldern, 2 Rahmen und 3 Befehlsschaltflächen (Datensatz Ablegen mit dem Namen `cmdAblegen` und Drucken mit dem Namen `cmdDrucken`).

Eingesetzte Steuerelemente

Pos.	Bezeichnung (Caption)	Steuerelement	Name
1	Daten aus Exceltabelle	Multipage (Page0)	Page1
2	Zinsberechnung	Multipage (Page1)	Page2
3	Wählen Sie eine Person.	Beschriftungsfeld	Nicht ändern
4	Anrede:	Beschriftungsfeld	Nicht ändern
5	Vorname:	Beschriftungsfeld	Nicht ändern
6	Nachname:	Beschriftungsfeld	Nicht ändern
7	Anlage:	Beschriftungsfeld	Nicht ändern
8	Kapital:	Beschriftungsfeld	Nicht ändern
9	Prozent:	Beschriftungsfeld	Nicht ändern
10	keine	Listenfeld	IstPerson
11	Person:	Beschriftungsfeld	Nicht ändern
12	keine	Kombinationsfeld	cboPerson
13	Kapitalanlage:	Beschriftungsfeld	Nicht ändern
14	keine	Textfeld	txtAnlage
15	Anfangskapital:	Beschriftungsfeld	Nicht ändern
16	keine	Textfeld	txtKapital
17	Prozent:	Beschriftungsfeld	Nicht ändern
18	keine	Textfeld	txtProzent
19	Zusatzdaten	Rahmen	Nicht ändern
20	Laufzeit im Jahr:	Beschriftungsfeld	Nicht ändern
21	keine	Kombinationsfeld	cboLaufzeit
22	Einfache Zinsrechnung:	Optionsfeld	optEZins
23	Zinsenzinsrechnung:	Optionsfeld	optZZins
24	Berechnete Daten	Rahmen	Nicht ändern
25	Zinsart:	Beschriftungsfeld	Nicht ändern
26	keine	Textfeld	txtZArt
27	Endkapital Kn:	Beschriftungsfeld	Nicht ändern
28	keine	Textfeld	txtKn
29	Datensatz Ablegen	Befehlsschaltfläche	cmdAblegen
30	Drucken	Befehlsschaltfläche	cmdDrucken

Anrede:	Vorname:	Nachname:	Anlage:	Kapital:	Prozent:
Herr Dr.	Marc	Wershoven	Immobilien	50000	0,03
Herr	Max	Mustermann	Anleihen	200000	0,02
Frau	Betina	Bertelt	Aktienfonds	240000	0,05
Frau Dr.	Petra	Plural	Akben	212000	0,08
Frau	Elke	Raza	Zertifikat	450000	0,04

Abb. A2.1.2 Das Userformblatt (Page1)
frmAufzinsungMitExcel

Abb. A2.1.2 Die Userformblatt (Page2)
frmAufzinsungMitExcel

Erläuterung zu dem Programmablauf

A2.2.1 Beim Starten des Userformblattes sollen alle Inhalte der Textfelder und Kombinationsfelder und Listenfeld geleert werden. Dann sollen alle Daten aus der Excel-Tabelle in das Listenfeld und nur Anrede, Vorname und Nachname in die Kombinationsfeld-Person einzugefügt werden. Anschließend sollen Konstante Werte ins Kombinationsfeld - Laufzeit im Jahr eingefügt werden.

A2.2.2 Wenn der Benutzer keine Eingaben in den Text-, Kombinationsfeldern und keine Auswahl bei einem Optionsfeld vornimmt und auf die Schaltflächen `Datensatz Ablegen` oder `Drucken` klickt, soll eine Meldung ausgegeben werden, dass alle Felder ausgefüllt bzw. ausgewählt werden sollten.

A2.2.3 Wenn der Benutzer einen Eintrag aus dem Listenfeld wählt, dann muss der Eintrag bzw. der Datensatz in der Seite Zinsberechnung angezeigt werden.

A2.2.4 Wenn der Benutzer einen Eintrag aus dem Kombinationsfeld Person wählt, muss der entsprechende Dantesatz angezeigt werden.

A2.2.5 Wenn die Felder `Person`, `Anlage`, `Anfangskapital`, `Prozent` und `Laufzeit im Jahr` und eine Zinsart (einfache Zinsrechnung oder Zinsenzinsrechnung) ausgewählt werden, müssen die Werte der Zinsart und des Endkapitales in den entsprechenden Textfeldern in dem Rahmenbereich „Berechnete Daten“ erscheinen.

A2.2.6 Wenn der Benutzer auf die Schaltfläche `Datensatz Ablegen` klickt, müssen die ausgewählte Daten mit berechneten Werten für den Kapitalwert (maximal 11 Werten: Laufzeit im Jahr muss maximal für 10 Jahre berücksichtigt werden) in das Layout ausgegeben bzw. übertragen werden. Dabei soll das Programm aus Sicht der Korrektheit vorherige Inhalte aus dem Layout (`ThisDocument`) löschen.

A2.2.7 Die Schaltfläche `Drucken` soll bewirken, dass das Layout (`ThisDocument`) an den Standard eingestellten Drucker geschickt wird.

A2.3 Wie schon bei den Punkten A2.2.1 bis A1.2.6 bei der Erläuterung zu dem Programmablauf beschrieben wurde, muss das Programm als Ergebnis `Abb. A2.4` für den Fall `Datensatz Ablegen` liefern.



Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Aufzinsung für die Kapitalanlagen	
Investmentperson:	<i>Frau Betina Bertelt</i>
Kapitalanlage:	<i>Aktienfonds</i>
Anfangskapital:	<i>240000</i>
Prozentsatz:	<i>0,05</i>
Laufzeit im Jahr:	<i>8</i>
Jahr	Kapitalwert
0	<i>240000</i>
1	<i>252000</i>
2	<i>264600</i>
3	<i>277830</i>
4	<i>291721,5</i>
5	<i>306307,58</i>
6	<i>321622,95</i>
7	<i>337704,1</i>
8	<i>354589,31</i>
9	
10	

Bei der Aufgabe wurde die maximale Laufzeit nur für 10 Jahre berücksichtigt. Zinsart ist

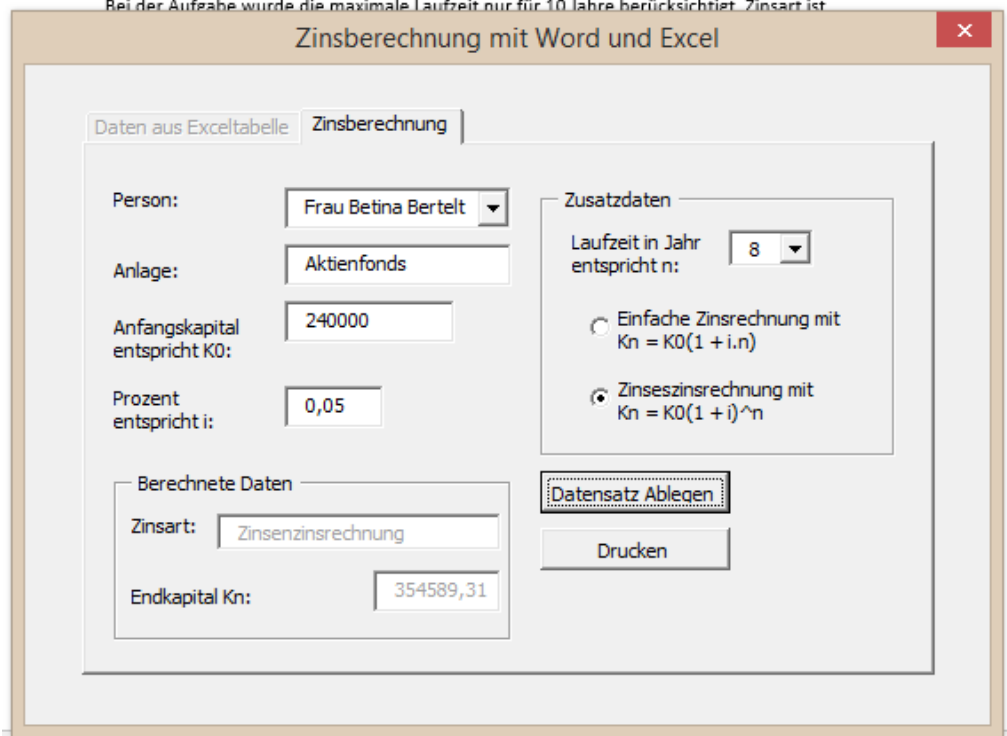


Abb. A2.4 Das Layout beim Datensatz Ablegen

LITERATUR

- Word-Formulare (www.rrzn.uni-hannover.de/buecher.html)
- Einführung in Word von Manuela Jürgens (VBA mit Word von Uni-Hagen.pdf)
- Investition und Finanzierung von Ulrich Ermschel et al., ISBN-978-3-7908-2745-3, Springer-Verlag Berlin Heidelberg 2011(<https://kataloge.uni-hamburg.de/DB=2/SET=5/TTL=1/SHW?FRST=3>)

VBA-Codes für die Aufzinsung

```

Rem Wie im A1
Sub mAufzinsung(DasBild As String)
    Selection.WholeStory
    Selection.Cut
    If DasBild > "" Then
        Selection.InlineShapes.AddPicture FileName:=DasBild, LinkToFile:=False, _
            SaveWithDocument:=True

    Else
        MsgBox "Das Bild existiert nicht. legen Sie das Haw-logo in den Ordner Bilder!"
        Exit Sub
    End If
    Selection.ParagraphFormat.Alignment = wdAlignParagraphRight
    Selection.TypeParagraph
    Selection.ParagraphFormat.Alignment = wdAlignParagraphLeft
    Call mAufzinsung
End Sub

```

Rem Globale Variablen in dem Code-Bereich von der frmAufzinsung

```

Option Explicit
Dim varMatrix(1 To 3) As Single
Dim strBildPath As String

```

Rem Wenn Userform gestartet wird, wird die Funktion Userform_Initialize() automatisch gestartet

```

Private Sub UserForm_Initialize()
    Dim oExcelApp As Object
    Dim oExcelWorkbook As Object
    Dim lZeile As Integer
    Dim lSpalte As Integer
    Dim strPath As String
    Dim strTabPath As String
    Dim strTabDatei As String
    Dim strTab As String
    strTab = "tblZinsEin"
    strPath = ActiveDocument.FullName
    strTabPath = Left(strPath, Len(strPath) - Len(Dir(strPath)))
    strTabDatei = strTabPath & "\database\al234567neu.xlsm"
    strBildPath = Left(strPath, Len(strPath) - Len(Dir(strPath))) + "\Bilder\"
    Maskeloeschen
    Set oExcelApp = CreateObject("Excel.Application")
    Set oExcelWorkbook = oExcelApp.Workbooks.Open(strTabDatei)
    With Me.cboLaufzeit
        .AddItem 2
        .AddItem 4
        .AddItem 5
        .AddItem 8
        .AddItem 10
    End With
    lZeile = 2 'Wir starten in Zeile 2, da in der ersten Zeile überschrieben stehen
    lSpalte = 2 'Wir starten in Spalte 2, da in der ersten Spalte die Nummer stehen
    Me.lstPerson.ColumnWidths = "1,7 cm; 1,8 cm; 2,1 cm; 2,6 cm; 2 cm; 1,6 cm"
    Me.lstPerson.ColumnCount = 6
    Do While oExcelWorkbook.sheets(strTab).Cells(lZeile, lSpalte) <> ""
        If lSpalte < 8 Then
            Me.lstPerson.AddItem oExcelWorkbook.sheets(strTab).Cells(lZeile, 1).Value
            Me.lstPerson.List(lZeile - 2, 0) = oExcelWorkbook.sheets(strTab).Cells(lZeile, 2).Value
            Me.lstPerson.List(lZeile - 2, 1) = oExcelWorkbook.sheets(strTab).Cells(lZeile, 3).Value
            Me.lstPerson.List(lZeile - 2, 2) = oExcelWorkbook.sheets(strTab).Cells(lZeile, 4).Value
            Me.lstPerson.List(lZeile - 2, 3) = oExcelWorkbook.sheets(strTab).Cells(lZeile, 5).Value
            Me.lstPerson.List(lZeile - 2, 4) = oExcelWorkbook.sheets(strTab).Cells(lZeile, 6).Value
            Me.lstPerson.List(lZeile - 2, 5) = oExcelWorkbook.sheets(strTab).Cells(lZeile, 7).Value

            Rem im Kombination auch eintragen
            Me.cboPerson.AddItem oExcelWorkbook.sheets(strTab).Cells(lZeile, 2).Value & " " _
                & oExcelWorkbook.sheets(strTab).Cells(lZeile, 4).Value
            lZeile = lZeile + 1
            lSpalte = lSpalte + 1
        Else
            Exit Sub
        End If
    Loop

    oExcelWorkbook.Close False
    oExcelApp.Quit

```

```

Set oExcelWorkbook = Nothing
Set oExcelApp = Nothing
Rem mAufzinsung strBildPath & "haw-logo.jpg"
Rem Me.ImgBild.Picture = LoadPicture(strBildPath & "blank.jpg")
End Sub

```

Rem Wenn der Benutzer aus dem Listenfeld einen Eintrag wählt

```

Private Sub lstPerson Click()
    Dim WelcherEintrag As Integer
    WelcherEintrag = Me.lstPerson.ListIndex
    Me.cboPerson.Text = Me.lstPerson.List(WelcherEintrag, 0) & " " &
        Me.lstPerson.List(WelcherEintrag, 1) & " " &
        Me.lstPerson.List(WelcherEintrag, 2)
    Me.txtAnlage.Text = Me.lstPerson.List(WelcherEintrag, 3)
    Me.txtKapital.Text = Me.lstPerson.List(WelcherEintrag, 4) * 1
    Me.txtProzent.Text = Me.lstPerson.List(WelcherEintrag, 5) * 1
    Me.MultiPage1.Pages(0).Enabled = False
    Me.MultiPage1.Pages(1).Enabled = True
End Sub

```

Rem Wenn der Benutzer aus dem Kombinationsfeld einen Eintrag wählt

```

Private Sub cboPerson Click()
    Dim WelcherEintrag As Integer
    WelcherEintrag = Me.cboPerson.ListIndex
    Me.cboPerson.Text = Me.lstPerson.List(WelcherEintrag, 0) & " " &
        Me.lstPerson.List(WelcherEintrag, 1) & " " & Me.lstPerson.List(WelcherEintrag, 2)
    Me.txtAnlage.Text = Me.lstPerson.List(WelcherEintrag, 3)
    Me.txtKapital.Text = Me.lstPerson.List(WelcherEintrag, 4) * 1
    Me.txtProzent.Text = Me.lstPerson.List(WelcherEintrag, 5) * 1
    Me.MultiPage1.Pages(0).Enabled = False
    Me.MultiPage1.Pages(1).Enabled = True
End Sub

```

Rem Wenn der Benutzer alle Daten von Userform in das Layout überträgt, wird es auf die Schaltfläche

Rem Datensatz Ablegen geklickt

```

Private Sub cmdAblegen Click()
    If Inhalterkennen Then
        Dim i As Integer
        mAufzinsung strBildPath & "haw-logo.jpg"
        With ActiveDocument.Bookmarks("tmZArt").Range
            .Text = Me.txtZArt.Value
            .Font.Name = "Calibri"
            .Font.Bold = False
            .Font.Italic = True
            .Font.Size = 14
            .Font.Color = wdColorBlack
        End With
        With ActiveDocument.Bookmarks("tmPerson").Range
            .Text = cboPerson.Value
            .Font.Name = "Calibri"
            .Font.Bold = False
            .Font.Italic = True
            .Font.Size = 14
            .Font.Color = wdColorRed
        End With
        With ActiveDocument.Bookmarks("tmAnlage").Range
            .Text = Me.txtAnlage.Value
            .Font.Name = "Calibri"
            .Font.Bold = False
            .Font.Italic = True
            .Font.Size = 14
            .Font.Color = wdColorRed
        End With
        With ActiveDocument.Bookmarks("tmKapital").Range
            .Text = CCur(Me.txtKapital.Value * 1)

            .Font.Name = "Calibri"
            .Font.Bold = False
            .Font.Italic = True
            .Font.Size = 14
            .Font.Color = wdColorRed
        End With
        With ActiveDocument.Bookmarks("tmProzent").Range
            .Text = Me.txtProzent.Value
            .Font.Name = "Calibri"
            .Font.Bold = False
            .Font.Italic = True

```

```

        .Font.Size = 14
        .Font.Color = wdColorRed
    End With
    With ActiveDocument.Bookmarks("tmLaufzeit").Range
        .Text = Me.cboLaufzeit.Text
        .Font.Name = "Calibri"
        .Font.Bold = False
        .Font.Italic = True
        .Font.Size = 14
        .Font.Color = wdColorRed
    End With
    If Me.optEZins.Value = True Then
        i = 0
        For i = 0 To Me.cboLaufzeit.Value
            With ActiveDocument.Bookmarks("tmk" & i).Range
                .Text = Round(Me.txtKapital.Text * (1 + Me.txtProzent.Text * i), 2)
                .Font.Name = "Calibri"
                .Font.Bold = False
                .Font.Italic = True
                .Font.Size = 14
                .Font.Color = wdColorRed
            End With
        Next i
    End If
    If Me.optZZins.Value = True Then
        i = 0
        For i = 0 To Me.cboLaufzeit.Value
            With ActiveDocument.Bookmarks("tmk" & i).Range
                .Text = Round(Me.txtKapital.Text * (1 + Me.txtProzent.Text) ^ i, 2)
                .Font.Name = "Calibri"
                .Font.Bold = False
                .Font.Italic = True
                .Font.Size = 14
                .Font.Color = wdColorRed
            End With
        Next i
    End If

    Else
        MsgBox "Sie müssen die Werte eingeben!"
        Exit Sub
    End If
End Sub

```

Rem Funktion für Erkennung einer numerischen Zahl. Als Ergebnis wird geliefert wahr oder falsch.

```

Function Inhalterkennen() As Boolean
    varMatrix(1) = Textfeldpruefen(Me.txtKapital)
    varMatrix(2) = Textfeldpruefen(Me.txtProzent)
    varMatrix(3) = Textfeldpruefen(Me.cboLaufzeit)
    Dim i As Integer
    i = 0
    For i = 1 To 3
        If varMatrix(i) Then
            Inhalterkennen = True
        Else
            Inhalterkennen = False
            Exit Function
        End If
    Next i
End Function

```

Rem die Prüfung der Textfelder

```

Function Textfeldpruefen(Textfeld As Object) As Double
    If IsNumeric(Textfeld.Value) Then
        Textfeldpruefen = Textfeld.Value
    Else
        MsgBox "Der Inhalt " & Textfeld & " ist nicht korrekt!"
        Textfeld.BackColor = RGB(255, 255, 0)
        Exit Function
    End If
End Function

```

Rem Wenn Userform gestartet wird, wird mir derm Prozedur alle Inhalte geleert.

```

Sub Maskeloeschen()
    Me.lstPerson.Clear

```

```
Me.cboLaufzeit.Text = ""
Me.cboLaufzeit.BackColor = RGB(255, 255, 255)
Me.txtAnlage.Text = ""
Me.txtAnlage.BackColor = RGB(255, 255, 255)
Me.txtKapital.Text = ""
Me.txtKapital.BackColor = RGB(255, 255, 255)
Me.txtProzent.Text = ""
Me.txtProzent.BackColor = RGB(255, 255, 255)
Me.txtZArt.Text = ""
Me.txtKn.Text = ""
Me.optEZins.Value = False
Me.optZZins.Value = False
End Sub
```

Rem Ereignis „Click“ auf das Kombinationsfeld für Zinsrechnung

```
Private Sub cboLaufzeit_Click()
    Me.optEZins.Value = False
    Me.optZZins.Value = False
End Sub
```

Rem Ereignis „Click“ auf das Optionsfeld

```
Private Sub optEZins_Click()
    Me.optZZins.Value = False
    If Inhalterkennen Then
        Dim varKn As Currency
        varKn = Me.txtKapital.Text * (1 + Me.txtProzent.Text * Me.cboLaufzeit.Text)
        Me.txtZArt.Text = "Einfache Verzinsung"
        Me.txtKn.Text = Round(varKn, 2)
    Else
        Me.optEZins.Value = False
    End If
End Sub
```

Rem Ereignis „Click“ auf das Optionsfeld für Zinsenzinsrechnung

```
Private Sub optZZins_Click()
    Me.optEZins.Value = False
    If Inhalterkennen Then
        Dim varKn As Currency
        varKn = Me.txtKapital.Text * (1 + Me.txtProzent.Text) ^ Me.cboLaufzeit.Text
        Me.txtZArt.Text = "Zinsenzinsrechnung"
        Me.txtKn.Text = Round(varKn, 2)
    Else
        Me.optZZins.Value = False
    End If
End Sub
```

Rem Wenn der Benutzer auf die Schaltfläche Drucken klickt, soll das Layout an dem Drucker geschickt Rem werden.

```
Private Sub cmdDrucken_Click()
    Application.PrintOut FileName="", Range:=wdPrintAllDocument, Item:=
    wdPrintDocumentWithMarkup, Copies:=1, Pages="", PageType:=
    wdPrintAllPages, Collate:=True, Background:=True, PrintToFile:=False,
    PrintZoomColumn:=0, PrintZoomRow:=0, PrintZoomPaperWidth:=0,
    PrintZoomPaperHeight:=0
End Sub
```